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	东南大学
参赛队号	**No.21102860212**
队员姓名	**1.** 张昀蔚
	2. 刘梦阳
	3. 王喻星

中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

题　目　F 航空公司机机组优化排班问题的研究

摘　　要:

机组排班问题是运筹学的经典优化问题之一，航空业一直以来都是国家战略性支柱产业，2020 年我国机场旅客达到吞吐量 85715.9 万人次。航空公司需要对机组人员进行合理排班满足日益增长的需求。针对机组排班问题，本文提出了多阶段的线性规划模型，使用求解器对具体算例求解，对于大规模算例，本文设计了启发式算法求解，给出在不同优化目标下的和不同规模的数据集下的机组排班方案。

针对问题一，我们设计了基于航班和机组人员的组合两阶段的 0-1 整数规划模型，最大化符合最低配置要求的航段数量，并在此基础上最小化换乘次数和替补次数。调用 Gurobi 求解器对数据集 A 求解得到符合要求的航段数量为 **206**，机组整体利用率为 **100%**，乘机人次数为 **8**，替补人次数为 **0**。对于规模较大的数据集 B，我们设计了时间复杂度为 $O(n^2)$ 的基于贪婪算法的多阶段启发式算法，得到数据集 B 的符合要求的航段数量为 **13411**，机组整体利用率为 **83.87%**，乘机人次数为 **0**，替补资格人次数为 **12**。

针对问题二，我们在问题一的模型的基础上，引入执勤的概念，扩展约束条件，设计了两阶段的 0-1 整数规划模型，最大化符合最低配置要求的航段数量，根据权重在问题一的次优化目标的基础上最小化总执勤成本和平衡总体执勤时长。为了减少模型变量数量和约束条件数量，设计了改进 0-1 整数规划模型，调用 Gurobi 求解器对改进模型在数据集 A 上求解得到符合要求的航段数量为 **206**，机组整体利用率为 **100%**，最小化执勤总成本为 **53.3473 万元**。对数据集 B，在问题一的基础上设计改进贪婪算法，求解得到符合要求的航段数量为 **11682**，机组整体利用率为 **85.59%**，最小化执勤总成本为 **3341.53 万元**。

针对问题三，考虑任务环，在问题二的数学模型的基础上扩展设计了两阶段的 0-1 整数规划模型，最大化符合最低配置要求的航段数量，在问题二的次优化的基础上最小化总任务环成本和平衡总体任务环时长。设计相应的改进 0-1 整数规划模型，调用 Gurobi 求解器对改进模型在数据集 A 上求解得到符合要求的航段数量为 **206**，机组整体利用率为 **100%**，最小化任务环总成本为 **6.27533 万元**。对于数据集 B，类似地拓展问题二的贪婪算法，得到符合要求的航段数量为 **4769**。

针对航空公司机组优化排班问题，本问提出了航段和机组人员组合的 0-1 整数规划模型，并根据不同问题的要求，扩展相应约束条件，三个问题均在规模较

小的数据集上求得性能良好的解。针对大规模数据提出基于贪婪算法的启发式算法进行求解。

关键字：机组排班问题　组合优化　数学规划　贪婪算法

目录

1. 问题重述

1.1 问题背景

众所周知，一趟民航航班必须在满足特定的条件下才能起飞，这些条件包括国家法律法规，国际公约，政府的行政条例，和公司自身的政策利益，一些国家的工会组织还会对机组人员的福利偏好有规章约束。所有这些条件都是对保证飞航安全和旅客服务质量最重要的因素之一。广义的机组人员包括飞行员，乘务员和空警。所谓机组排班问题，就是构造特定时间段的机组日程安排，包括每个机组人员在何时何地及哪个航班执行何种任务。机组排班问题是运筹学应用的最早典范之一，本题的宗旨是建立线性优化模型，明确表达飞行时间、执勤时间、休息时间等约束，把航班直接分配给机组人员。

在机组优化排班问题，本题中机组人员仅针对飞行员要求，每个机组人员都有一个固定的基地，并且具有一个且仅有一个主要资格，但也可以具备替补资格，机组排班需要优先为机组人员安排主要资格，也可以安排替补资格。除了执行飞行任务外，机组人员还可以执行乘机任务，乘机任务是指机组人员乘坐正常航班从一机场摆渡到另一机场执行飞行任务。

机组人员执行航段任务，若干航段任务连接组成执勤，若干执勤组成任务环，若干任务环组成一个排班周期，其关系如图1。

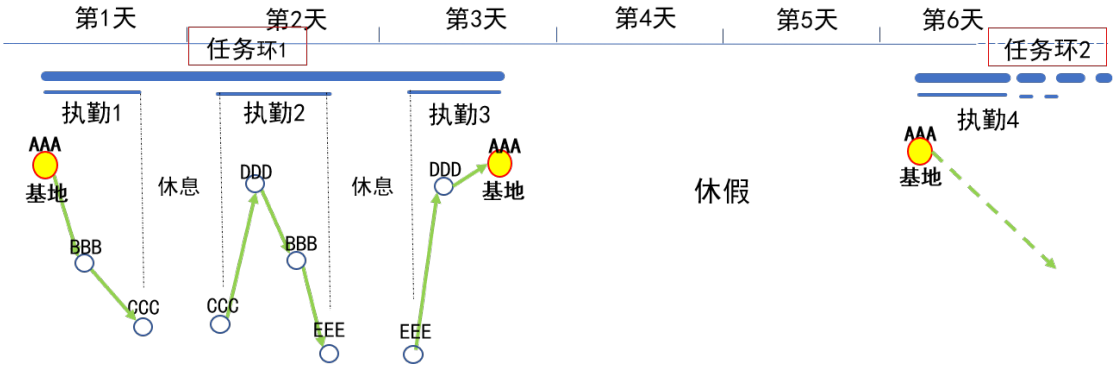

图 1　航段，执勤，任务环与任务周期的关系图

航班 (Flight)：指飞机的一次起飞和降落，每个航班都有给定的起飞日期、时间、机场和到达日期、时间、机场。每个航班都有给定的最低机组资格配置，一个航班只有满足了最低机组资格配置才能起飞。

执勤 (Duty)：一次执勤由一连串飞行任务和乘机任务和间隔连接时间组成，同一次执勤的航段需要满足上一躺航段的到达和下一趟航段的出发机场一致；上一趟航段的到达和下一趟航段的出发之间的间隔时间必须满足最短连接时间约束；同一次执勤里的每个航段必须在同一天起飞，但到达时间不受此限制。

任务环 (Pairing)：由一连串的执勤和休息时间组成，从自己的基地出发并最终回到自己的基地，属于同一任务环的执勤之间满足关系：上一次执勤的结束机场必须和下一次执勤的起始机场相一致；两个相邻执勤之间的休息时间必须满足最少休息时间限制；每个任务换里的第一次执勤必须从基地出发，最后一次执勤必须回到基地。

排班计划 (Roster)：每个机组人员在每个排班周期都有一个排班计划。这个计划由一系列的任务环和休假组组成，任务环之间满足一定的休假天数。机组人员的排班计划按排班周期编排，每个周期开始前几天完成编排并向机组人员发布。排班周期之间有严格的时间分割，机组人员在给定排班周期内的最后一个任

务环可能跨越这个时间分割点而进入下一个排班周期。本赛题假定所有机组人员的初始位置和排班周期结束时的终了位置都是在其基地，并把排班周期进行了适当的延展。

1.2 问题的提出

本赛题由三个子问题组成，每个子问题都基于前一个子问题并与之相容，每个问题具体如下所描述：

问题 1 要求建立线性规划模型给航班分配机组人员，依编号次序满足目标

① 尽可能多的航班满足机组配置；

④ 尽可能少的总体乘机次数；

⑦ 尽可能少使用替补资格。

约束条件：

1. 每个机组人员初始从基地出发并最终回到基地；
2. 每个机组人员的下一航段的起飞机场必须和上一航段的到达机场一致；
3. 每个机组人员相邻两个航段之间的连接时间不小于 MinCT 分钟.

问题 2 本问题引进执勤概念，假定每个机组人员的每单位小时执勤成本给定。本问题除了需要满足子问题 1 的所有目标外，还需满足如下目标：

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。

约束条件:

1. 每个机组人员每天至多只能执行一个执勤；
2. 每次执勤的飞行时间最多不超过 MaxBlk 分钟；
3. 每次执勤的时长最多不超过 MaxDP 分钟；
4. 每个机组人员下一执勤的起始机场必须和上一执勤的结束机场一致；
5. 每个机组人员的相邻两个执勤之间的休息时间不小于 MinRest 分钟。

问题 3 编制排班计划，假定每个机组人员的每单位小时任务环成本给定。本问题除了需要满足子问题 1 和子问题 2 的所有目标外，还需满足如下目标：

③ 机组人员的总体任务环成本最低；

⑥ 机组人员之间的任务环时长尽可能平衡。

约束条件：

1. 每个机组人员每个排班周期的任务环总时长不超过 MaxTAFB 分钟；
2. 每个机组人员相邻两个任务环之间至少有 MinVacDay 天休息；
3. 每个机组人员连续执勤天数不超过 MaxSuccOn 天。

2. 模型假设

1. 机组人员之间可以任意组合；
2. 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；
3. 不满足最低机组资格配置的航班不能配置任何机组人员；
4. 机组人员可以乘机摆渡，即实际机组配置可以超过最低配置要求，成绩机组人员的航段时间计入执勤时间，但不计入飞行时间。

3. 符号说明

表 1　符号说明

符号	变量说明
常量	
i, j	航班编号
t	日期编号
k	机组人员编号
a	机场编号
C	机组人员集合, $k \in C$
D	日期集合,$t \in D$
AP	机场集合,$p \in AP$
F	航段集合,$i, j \in D$
$Base$	基地集合
C_1	不可兼任副机长的机长集合
C_2	可以兼任副机长的机长集合
C_3	副机长集合
F_p^a	降落机场为 a 的航段集合,$p \in AP$
F_p^l	起飞机场为 a 的航段集合,$p \in AP$
F_t^{Date}	出发日期为 t 的航段集合,$t \in T$
(i, j)	航段 i,j 的航段组合
FF	航段组合的集合,$(i, j) \in FF$
$base_k$	机组人员 k 所属的基地
$date_i$	航段 i 的起飞日期
$DCost_i$	机组人员 i 的每小时执勤成本
$RCost_i$	机组人员 i 的每小时任务环成本
T_i^l	航段 i 的出发时间
T_i^a	航段 j 的到达时间
ST	时间块
$CNeed_i, FNeed_i$	航段 i 的机长，副机长的数量要求
MinCT	航段之间的最小连接时间
MaxBlk	一次执勤最大飞行时长

表 1 （续）

符号	变量说明
MaxDP	一次执勤最大时长
MinRest	相邻执勤最小休息时间
MaxTFRB	最大任务环总时长
MaxSuccOn	最大连续执勤天数
MinVacDay	相邻任务环最小休息时间
μ	优化目标优先级权重
0-1 变量	
x_{ik}^{cap}	若机组人员 k 作为正机长执行航段 i 的飞行任务则取 1，否则取 0
x_{ik}^{fo}	若机组人员 k 作为副机长执行航段 i 的飞行任务则取 1，否则取 0
x_{ik}^{dh}	若机组人员 k 执行航班 i 的乘机任务则取 1，否则取 0
z_i	若航段 i 满足机组配置则取 1，否则取 0
y_{ijk}	航段之间的执行任务连接，若机组人员 k 执行航班 i 后执行航班 j 则取 1，否则取 0
v_{ijk}	执勤之间的执行任务连接，若机组人员 k 执行航段 i 作为上一个执勤的结束，执行航段 j 作为下一个执勤的起始则取 1，否则取 0
w_{ijk}	任务环之间的执行任务连接，若机组人员 k 执行航班 i 作为上一个任务环的结束，执行航班 j 作为下一个任务环的起始则取 1，否则取 0
u_{ijk}	任务环起始连接，若机组人员 k 执行航段 i 作为当前任务环的开始，执行航段 j 作为当前任务环的结束则取 1，否则取 0
r_{ik}^{s}, r_{ik}^{f}	若机组人员 k 将航班 i 作为任务周期的起始（结束）则取 1，否则取 0
d_{ik}^{s}, d_{ik}^{f}	若机组成员 k 将航班 i 作为执勤的起始（结束）任务则取 1，否则取 0
整数变量	
$FNum$	符合配置要求的航段数量
$MaxDT, MinDT$	机组人员最大执勤时长和最小执勤时长
$MaxRT, MinRT$	机组人员最大任务环时长和最小任务环时长

4. 问题一模型的求解及算法设计

4.1 基于航段的机组排班模型的建立与求解

航班排班计划为经典运筹学问题 [1–4]。针对问题一，每个排班周期直接由航班组成，航段和机组人员构成组合，该问题是一个非常典型的多目标优化问题，优化目标的优先级依次为：最大化机组配置的航班数量；最小化总体乘机次数，最小化的使用替补次数。针对该问题，考虑到保证最大化满足要求的航段数量的最高优先级，同时考虑后续需要引进较多目标函数，因此我们建立一个两阶段的 0-1 规划数学模型，第一阶段将优先度最高的最大化机组配置的航班数量作为目标函数，根据赛题所给要求，设计相应的约束条件。

对第一阶段数学模型进行求解, 在得到该模型的解之后，将该目标函数及其解转化为第二阶段的约束条件，在保证可以最大化航段数量的基础上，目标函数则是优先度其次的最小化总体乘机次数的最小化替补资格次数的加权函数。

4.1.1 数学模型的建立

1. 第一阶段的模型建立

(1) 目标函数

优先度最高的目标为最大化满足机组配置的航段数量，z_i 表示航班 i 是否满足航班配置，若是则取 1，记该目标函数为 f_1，如下所示：

$$\max f_1 = \sum_{i \in F} z_j \tag{4.1}$$

(2) 约束条件

约束条件 1 保证任一个机组人员 k，若其在航段 i 执行任务，则只有可能作为机长或者副机长执行飞行任务，或者执行乘机任务：

$$x_{ik}^{cap} + x_{ik}^{fo} + x_{ik}^{dh} \geq 1, \quad \forall i \in F, k \in C, \tag{4.2}$$

其中 x_{ik}^{cap}, 表示机组人员 k 在航段 i 是否执行机长任务，x_{ik}^{fo} 表示机组人员 k 在航段 i 是否执行副机长任务,x_{ik}^{dh} 表示机组人员 k 在航段 i 是否执行乘机任务，若是则取 1；

约束条件 2 保证任一个机组人员 k 的任务执行内容符合其资格：

$$x_{ik}^{cap} = 0 \quad , \forall i \in F, k \in C_3, \tag{4.3}$$

$$x_{ik}^{fo} = 0 \quad , \forall i \in F, k \in C_1, \tag{4.4}$$

式 (4.3) 中 C_3 指仅具有副机长资格的机组人员 k 集合，保证他们无法执行机长任务；式 (4.4) 中 C_1 指仅具有机长资格的机组人员 k 集合，保证他们无法执行副机长任务。

约束条件 3 变量 z_i 为 0-1 变量，表示航段 i 是否满足航班机组配置，若是则取 1，保证正常起飞航段 i 的机长与副机长数量分别符合配置要求：

$$\sum_{k \in C} x_{ik}^{cap} = z_i \dot{C} Need_i, \quad \forall i \in F, \tag{4.5}$$

$$\sum_{k \in C} x_{ik}^{fo} = z_i \dot{F} Need_i, \quad \forall i \in F, \tag{4.6}$$

其中 $CNeed_i, FNeed_i$ 分别指航段 i 对机长和副机长的数量要求，约束 (4.5)-(4.6) 保证当 $z_i = 1$ 时，必有航段 i 上执行机长和副机长飞行任务的机组人员数量符合要求。

约束条件 4 保证不满足最低配置的航班不能配置任何机组成员，保证任意一个航班，如果存在机组人员执行任务，则说明该航班符合配置要求：

$$\sum_{k \in C} x_i k^{cap} + x_i k^{fo} + x_i k^{dh} \leq M z_i, \quad \forall i \in F, \tag{4.7}$$

其中 M 是一个充分大的正整数，式 (4.7) 保证当 $z_i = 0$ 时，没有机组人员在该航班 i 上执行任务, 同时如果任意的 x_{ik}^{cap}，x_{ik}^{fo} 或 x_{ik}^{dh} 取值为 1 则有相应的 z_i 取值为 1。

约束条件 5 变量 y_{ijk} 为 0-1 变量，说明对机组人员 k 航班 i 和航班 j 之间是否存在连接，若存在则取 1，保证每个机组人员的下一航段的起飞机场必须和上一航段的到达机场的一致：

$$y_{ijk} = 0, \quad \forall (i,j) \in FF_1, \quad \forall k \in C, \tag{4.8}$$

其中 FF_1 为航班组合的集合，对 $\forall (i,j) \in FF_1$，有航段 j 的到达机场和航段 i 的起飞机场不一致，此时航段 i 和航段 j 之间不存在连接。

约束条件 6 保证每个机组人员相邻两个航段之间的连接时间不小于 MinCT 分钟：

$$y_{ijk} = 0, \quad \forall (i,j) \in FF_2, \quad \forall k \in C, \tag{4.9}$$

对 $\forall (i,j) \in FF_2$，有 $T_i^a + MinCT > T_i^l$，即航段 i 的到达时间和航段 j 的起飞时间不满足间隔 MinCT 的时间关系时，航段 i 和航段 j 之间不存在连接。

约束条件 7 变量 r_{ik}^s 为 0-1 变量，说明机组人员 k 是否将航班 i 作为本次任务周期的起始航段，若是则取 1，保证每个机组人员 k 初始从基地出发并最终回到基地：

$$r_{ik}^s = 0, \quad \forall i \in F / F_{base_k}^l, \forall k \in C, \tag{4.10}$$

$$r_{ik}^f = 0, \quad \forall i \in F / F_{base_k}^a, \forall k \in C, \tag{4.11}$$

其中 $F_{base_k}^l, F_{base_k}^a$ 分别表示所有出发机场，到达机场为机组人员 k 所属基地的航段集合，式 (4.10) 保证若航段 i 不从机组人员 i 所属基地出发，则该机组人员不会将该航段作为任务周期的起始；式 (4.11) 保证若航段 i 不到达机组人员 i 所属基地，则该机组人员不会将该航段作为任务周期的结束。

约束条件 8 保证任一机组人员 k 每个任务周期内至多存在一个起始航段 i 并存在与之对应的一个结束航段 j：

$$\sum_{i \in F} r_{ik}^s \leq 1, \quad \forall k \in C, \tag{4.12}$$

$$\sum_{i \in F} r_{ik}^s - \sum_{j \in F} r_{jk}^f = 0, \tag{4.13}$$

约束条件 9 保证任一航段前序航段连接和后序航段连接流量相等：

$$(\sum_{j \in F} y_{ijk} + r_{ik}^f) - (\sum_{j \in F} y_{jik} + r_{ik}^s) = 0, \quad \forall i \in F, \forall k \in C, \tag{4.14}$$

约束条件 10 保证任一航段若满足配置要求，则其存在连接流量；任一航段不满足配置要求，则其不存在连接流量：

$$(1-(\sum_{j\in F} y_{ijk}+r_{ik}^{s}+r_i k^{f}))\le M*(1-(x_i k^{cap}+x_i k^{fo}+x_i k^{dh})),\quad \forall i\in F,\forall k\in C, \tag{4.15}$$

$$\sum_{j\in F} y_{ijk}+r_{ik}^{f}\le x_{ik}^{cap}+x_{ik}^{fo}+x_{ik}^{dh},\quad \forall i\in F,\forall k\in C, \tag{4.16}$$

式 (4.15) 中若有 $x_{ik}^{cap}+x_{ik}^{fo}+x_{ik}^{dh}$ 取值为 1，则其对应执行任务的机组人员存在前后连接航段或者将该航段作为起始结束航段；式 (4.16) 中若有 $x_{ik}^{cap}+x_{ik}^{fo}+x_{ik}^{dh}$ 取值为 0，则其不存在前后航段且不作为起始结束航段。

2. 第二阶段的模型建立

在求解第一阶段模型之后，我们可以得到最大化满足配置要求的航班数量，定义该值为 $MaxFNum$，第二阶段模型具体过程如下：

(1) 目标函数

该阶段的优化目标为最小化乘机次数和最小化替补资格次数。

针对最小化乘机次数，x_{ik}^{dh} 表示机组人员 k 在航段 i 执行乘机任务，若是则取 1，记该目标函数为 f_4：

$$\min f_4=\sum_{k\in C} x_{ik}^{dh}, \tag{4.17}$$

针对最小化替补资格，x_{ikf^o} 表示机组人员在航段 i 执行副机长 k 任务，若是则取 1，其中 C_2 是指可以兼任副机长的机长集合，记该目标函数为 f_7：

$$\min f_7=\sum_{k\in C_2} x_{ik}^{fo}, \tag{4.18}$$

设定权重 μ_4,μ_7 分别对目标函数进行加权，构造单目标数学规划模型 f_12

$$\min f_{12}=\mu_4 f_4+\mu_7 f_7, \tag{4.19}$$

(2) 约束条件

保持第一阶段的约束条件 1-10 不变，新增以下约束：

约束条件 11 保证符合配置要求的航班数量为第一阶段所求的目标函数值，即保证优先级最高的目标函数最大化不变：

$$\sum_{i\in F} z_i=maxFNum, \tag{4.20}$$

4.1.2 数学模型的求解

本文使用 Gurobi 优化器辅助进行模型求解，它是一种商用数学规划优化求解器，与同类产品进行比较在速度上具备一定优势，同时对多种常用编程语言如 Python、C、C++ 等提供多种方便轻巧的接口。本题具有两套不同规模的数据集，其中 A 数据集中有机组人员 21 人，航班 206 架次，跨越 15 天规模总体可控。而 B 数据集中有机组人员 465 人，航班 13954 架次，跨越 31 天，规模庞大对传统的线性规划模型而言在计算上十分具有挑战性。

对于 A 数据集本模型具有 912992 个 0-1 决策变量以及 790007 个约束；而对于 B 数据集本模型具有 90568234605 个 0-1 决策变量，这不仅是求解速度存在

限制，也对计算的存储提出了挑战，求解使用的 32G 内存计算机已无法完整加载模型。

进行求解的计算机硬件信息如表2所示。

表 2　用于求解模型的计算机配置信息

中央处理器	11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz 8 核 16 线程
内存	Samsung DDR4 D-Die 32G
硬盘	512G （已开启虚拟内存）

对于 A 数据集，我们通过 Python 调用 Gurobi 求得了最优的排班策略，成功**为 206 架次即所有的航班进行了人员排班**，并仅有 **8 位机组人员执行了乘机任务，替补资格使用次数为 0**。其结果如表3所示。

表 3　问题一 Gurobi 求解 a 数据集成功结果指标

不满足机组配置航班数	0
满足机组配置航班数	206
机组人员总体乘机次数	8
替补资格使用次数	0
程序运行分钟数	2.06

通过甘特图可视化排班效果如图2所示，通过不同颜色体现了某一航段机组人员的身份，可以看到乘机任务多发生在刚一起从基地出发时，此时的航段调度需求较大。

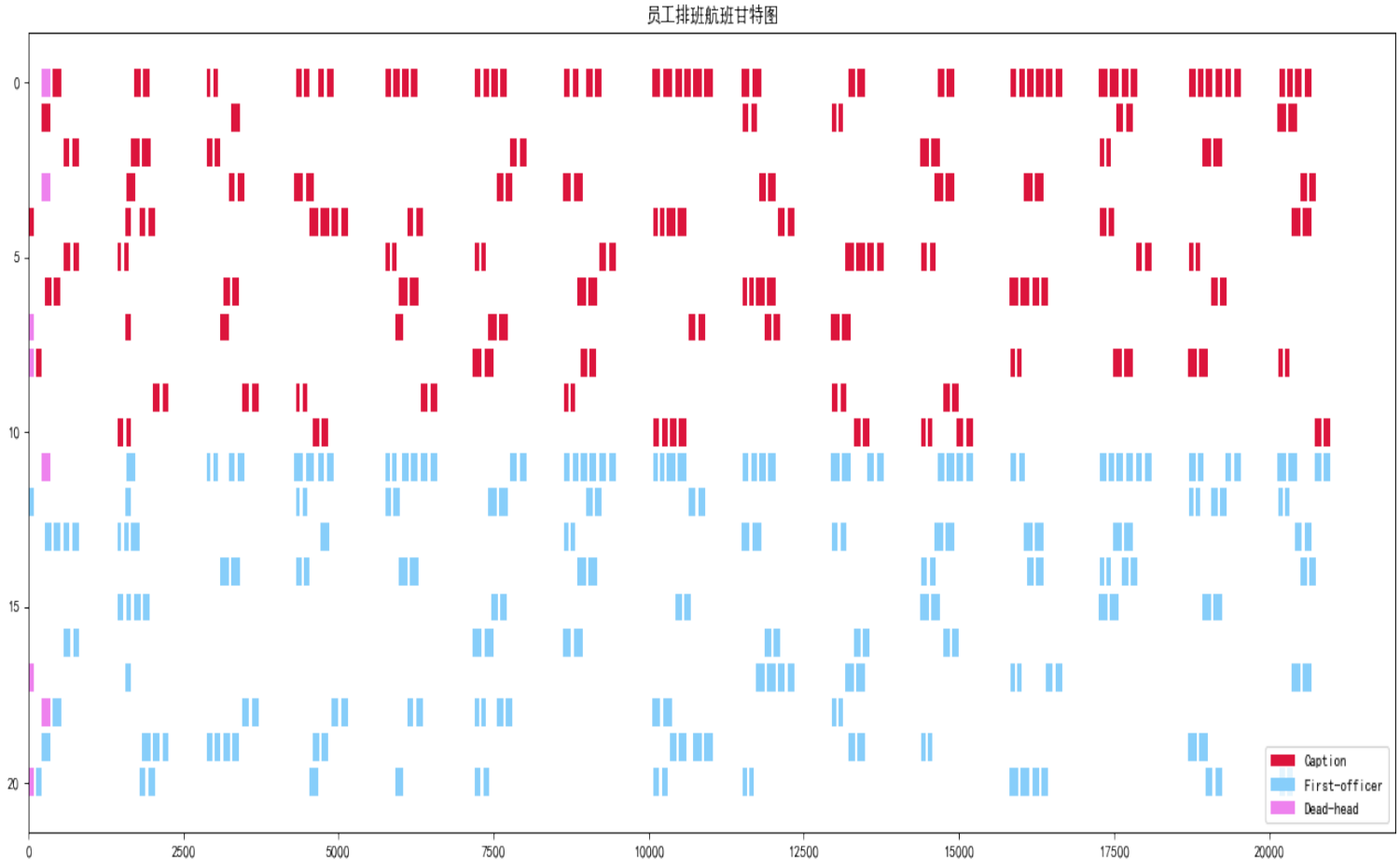

图 2　问题一 Gurobi 求解 a 数据集航班排班甘特图

4.2 优化算法的设计与求解

4.2.1 优化算法的设计

问题一的目标依次是最大化满足资格配置的航班数量，最小化乘机人次数，最小化替补人次数，将每一个航班作为一个节点（航班号相同但不是同一出发时间的航班视为不同的航班）。然后根据约束条件，下一航段的起飞机场要和上一航段的到达机场一致，相邻两个航段的连接时间要大于等于 MinCT 分钟。

根据这两个约束条件，遍历所有的航班对，对于符合要求的航班对建立有向边，得到一个有向无环图（DAG）。题目要求满足每个基地人员从基地出发并回到基地，因此定义从基地出发并到达基地的一条路径称为有效路径。

该算法主要分为三阶段，第一阶段使用贪婪算法生成含航班数量较多的可行路径，第二阶段为匹配问题，将第一阶段所得的可行路径与机组人员进行匹配，第三阶段中如果仍存在没有执行任务的机组人员，和没有满足配置要求的航段，则通过换乘将进行分配。

具体算法流程如下所示：

生成有效路径的贪婪算法

如果对该有向无环图进行遍历找出所有有效路径，该算法的算法复杂度为 $O(n!)$，是经典 NP-Hard 问题，时间复杂度较高。为了降低时间复杂度，在对有向边的搜索中，不考虑任意两点间的公共边情况，即假设该 DAG 为一个简单图，同一航班只能出现在一条可行路径中。

将所有航班根据起飞时间从小到大排序，有贪心准则如下所示：

a. 将所有起飞机场为基地，且不存在时间上满足从基地出发到达另一飞机场，并从另一飞机场返回基地的前序航段的航段作为有效路径的起点。

b. 将起点航段作为前序航段向后遍历，如果当前航段的起飞机场和前序航段的到达机场一致，且满足时间间隔 MinCT，则将该航段加入该有效路径。

c. 将该航段作为前序航段，重复遍历，得到一条有效路径。

d. 对该有效路径中的航段从后向前遍历每个航段的结束地点，若不为基地则从该有效路径中删除，直到出现到达机场为基地的航段，则遍历终止，输出该有效路径，并将该路径中航段标记。寻找可行路径的算法设计大致如图3所示。

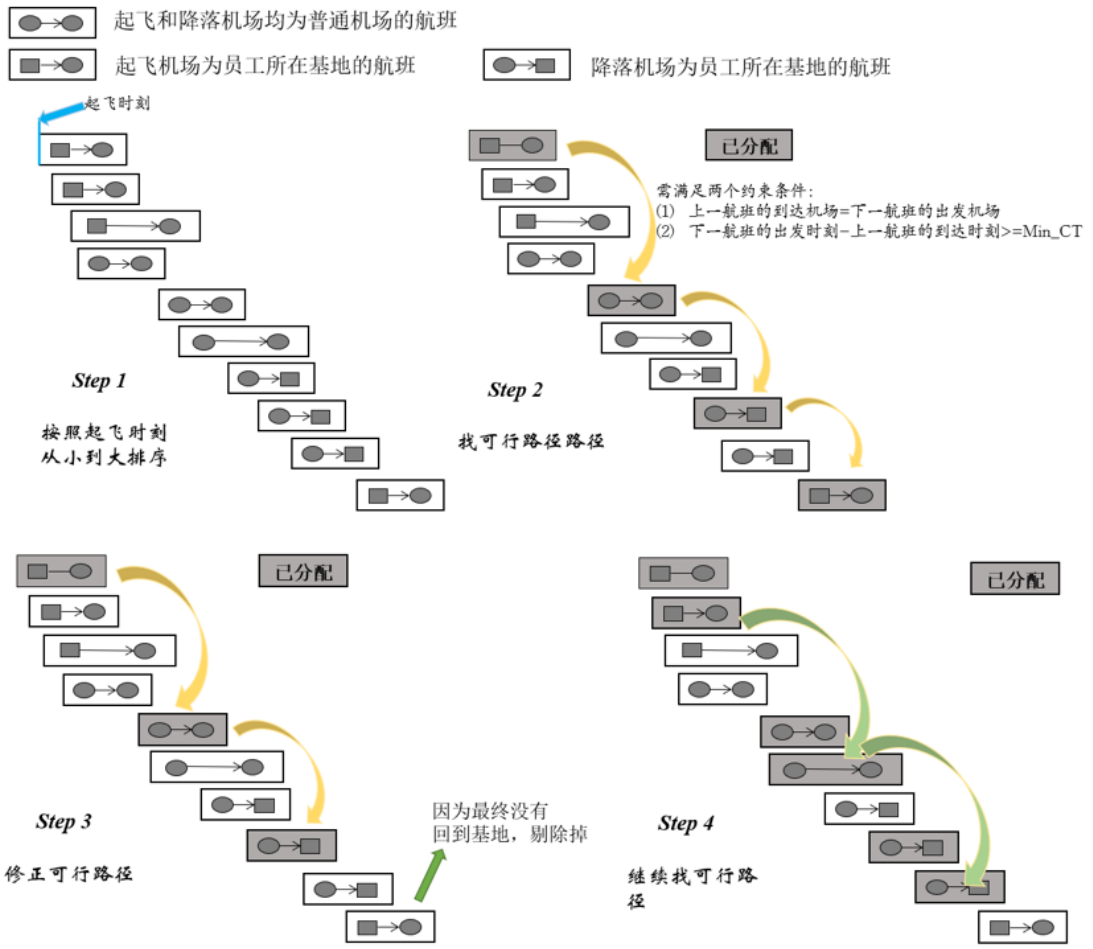

图 3　贪婪算法寻找有效路径思路示意图

e. 对所有未标记的航段重复上述流程。

f. 对当前已有的起点航段全部完成遍历后，若仍存在未标记航段，则从中选取起飞机场为基地的航段，并在未标记的航段中不存在从基地出发到达另一飞机场再返回基地的前序航段，得到新的起点航段，重复上述流程；否则跳出该流程。贪婪算法流程如图4所示。

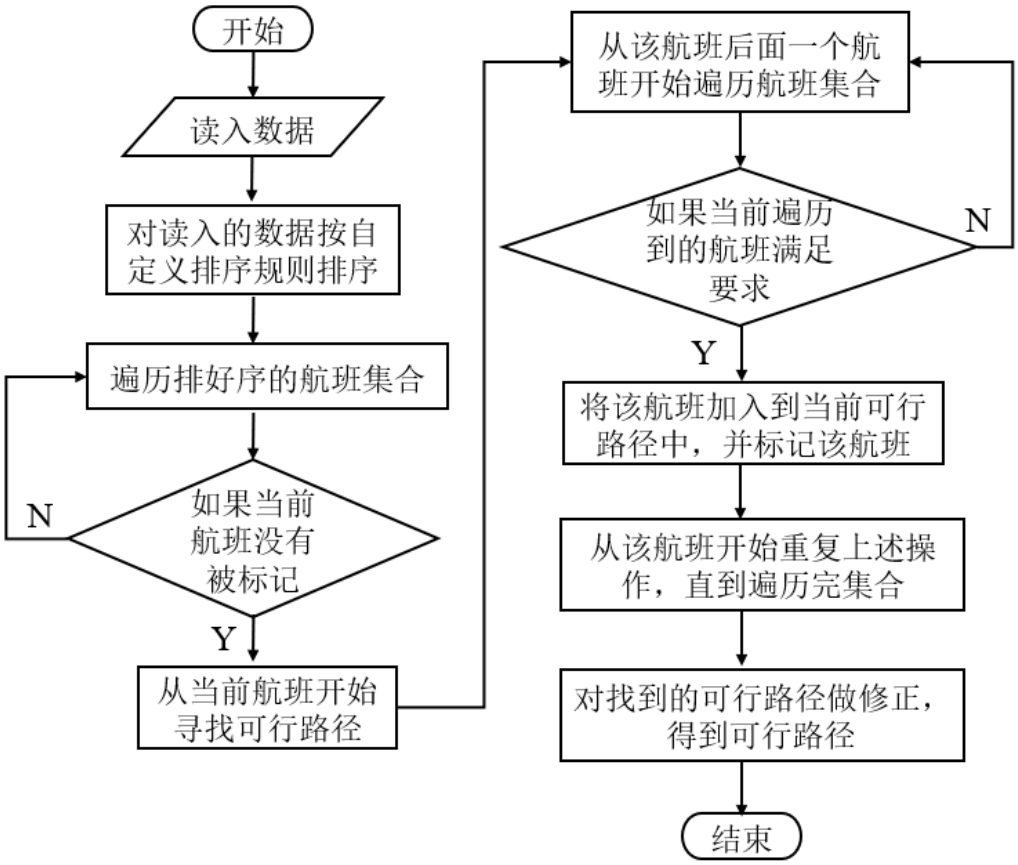

图 4　贪婪算法流程图

路径与员工的匹配问题算法

为了减少替补次数，对只可以担任正机长的机组人员和只可以担任副机长的机组人员进行配对和分配有效路径，该匹配贪婪算法具体流程如下所示：

a. 根据最低航段配置要求，将只能担任正机长的机组人员与只能担任副机长的机组人员机型配对，将所有有效路径根据其中航段数量进行排序，依次进行分配。

b. 对于剩余机组员工，如果配对完成后，将剩余为配对的机组人员中仅具备单一资格的，将其与可替补的人员组合，对剩余未分配可行路径进行分配。

考虑换乘的完善算法设计

如果仍存在未分配任务的机组人员和没有被执行的航段，则考虑换乘满足最大化执行航段的要求，其具体算法如下所示：

a. 将未标记的航段作为起点路径遍历所有未被标记的航段，生成一路径片段，将该路径片段作为当前路径，若发生换乘，在有效路径中需要存在满足从基地出发和时间间隔要求的前序航段，和到达基地和时间间隔要求的后序航段。

b. 遍历所有被标记的航段，若到达机场为当前航段的出发机场，时间间隔满足要求，则将其所在有效路径进行分割，取路径起点到该航段的路径片段加入前序路径集合。

c. 若起飞机场为当前航段的到达机场，且满足时间间隔要求，对其所在有效路径进行分割，取该航段到路径重点的路径片段加入后序路径集合。

d. 从前序路径集合和后序路径集合中分别选择航段数量最少的路径片段，和当前路径片段组合得到新的有效路径，对剩余机组人员进行分配。

该贪婪算法的时间复杂度为 $O(n^2)$，航段排序的时间 $O(n\log n)$，对航段排序进行搜索的时间复杂度为 $0(n^2)$，对可行路径的排序时间复杂度为 $O(n^2)$。对所有被标记的点进行搜索的时间复杂度为 $O(n^2)$。

考虑换乘的完善算法设计如图5所示.

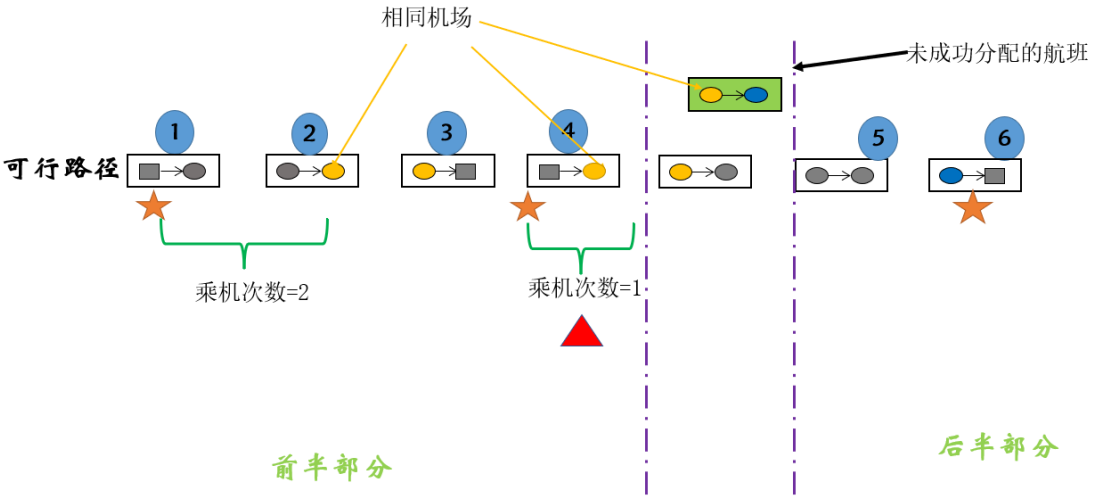

图 5　寻找最小换乘次数示意图

4.2.2 优化算法的求解

对于数据集 A 的求解

根据上述算法找到所有的有效路径，一共形成 3 条有效路径，选择 3 个机长和 3 个副机长进行匹配。航班总数为 202，总体乘机次数为 0，使用的替补次数为 0。不满足机组配置的航班数为 4，通过完善算法对这些航段进行分配，得到结果如4所示：

具体计算结果如4所示。

表 4　问题一贪婪算法求解 b 数据集成功结果指标

不满足机组配置航班数	543
满足机组配置航班数	13411
机组人员总体乘机次数	0
替补资格使用次数	12
程序运行分钟数	0.0009

可以看出该贪婪算法的计算结果和建立数学模型使用 Gurobi 求解的结果相同，在程序运行时间上，对同一数据集，该贪婪算法用时小于 1s，远远小于 Gurobi 用时，说明该算法具有可行性，可操作性，具有较好的性能。

对于数据集 B 的求解

对于数据集 B 的处理为 B 数据集有两个基地，所以所有的可行路径在分配时，就不能像数据集 A 那样直接分配，而是应该先将可行路径根据其起点和节点所在的基地分成两个集合，即基地 1 对应的可行路径以及基地 2 对应的可行路径。然后对在基地 1 的机组人员分配基地 1 对应的可行路径，在基地 2 的机组

人员分配基地 2 对应的可行路径。通过计算得出可行路径数量为：213 条，然后根据每条路径对应的起点，终点所在基地，将其分成两个集合，其中基地 1 对应的可行路径有 54 条，基地 2 对应的可行路径有 159 条。然后将其分配给各个基地的机组人员组合，最终得到满足机组配置的航班数为 13411，不满足机组配置的航班数为 543。如图6所示：

按照之前的策略通过加入换乘次数满足尚未分配的航班，但是结果发现，大部分情况下通常要通过牺牲很多的换乘次数才能满足很少的航班数，所以决定不采取换乘。

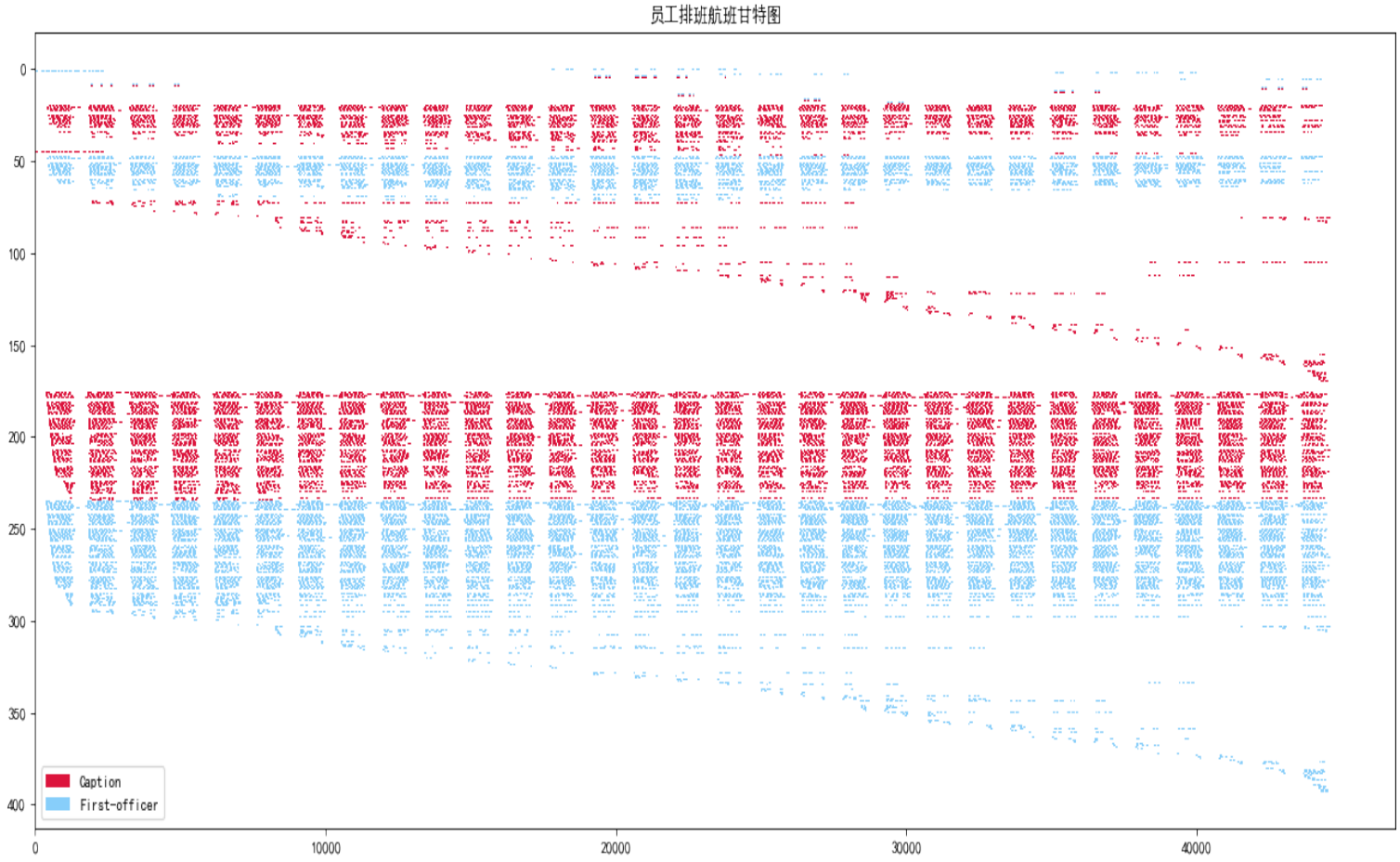

图 6　问题一贪婪算法求解 b 数据集航班排班甘特图

5. 问题二模型的求解及算法设计

5.1 基于执勤的机组排班模型的建立与求解

5.1.1 数学模型的建立

问题二在问题一的基础上，引入了执勤的概念。执勤由航班与连接组成，任务周期由航班组成。

结合问题一已有的优化目标和问题二新提出的优化目标，其最高优先级目标仍为最大化满足配置的航班数量，次优先级目标依次为：最小化机组人员总体执勤成本；最小化乘机次数；机组人员之间的执勤时长尽可能平衡；最小化替补资格次数。

针对问题二，我们在问题一所提出的模型基础上得到改进多阶段 0-1 数学规划模型，在对于多目标的求解上，同样分为两阶段进行求解，第一阶段的目标函数为最大化满足配置要求的航班数量。在第二阶段中，将第一阶段所求得模型解与其函数作为约束条件，对次优目标根据其优先级进行加权，得到第二阶段目标函数。

结合新的优化目标重新设计目标函数，并针对问题二的进一步要求在原有模型的约束条件的基础上，增加新的约束条件。数学规划模型具体建立过程如下：

1. 第一阶段数学模型

(1) 目标函数

在第一阶段中目标函数保持问题一中所给出的 f_1 不变, 该优化目标为最大化满足配置要求的机组数量。

(2) 约束条件

同时考虑执勤起始航段和执勤结束航段，考虑执勤的概念与相关约束，我们引入 0-1 变量 v_{ijk} 航段 i 和航段 j 之间是否存在执勤连接，即若机组人员 k 以航段 i 作为上一执勤的结束航段，航段 j 作为下一执勤的起始航段，则 v_{ijk} 取值为 1，否则为 0。

执勤与执勤之间的连接实际上是一种特殊的航段与航段之间的连接，需要满足额外的执勤连接的要求，任何作为执勤起始的航段若其同时为排班周期起始的航段，则其必须从机组人员所在基地出发，若其不是排班周期起始的航段，则其存在一个前序执勤，前序执勤的结束航段的到达机场和该起始航段的出发机场必须保持一致，并且同时满足对执勤间隔的时间要求。

同理可得，任何作为执勤结束的航段若其为排班周期的结束航段，则其必须返回机组人员所在基地，否则存在一个后序执勤，该后序执勤的起始航段需要满足机场和间隔时间的约束条件。

该问题同时要求任意机组人员一天之内至多存在一个执勤，若该机组人员当天内存在执行任务航段，则必存在执勤，反之亦然。而若存在一个起始航段，则必定在当天内存在一个对应的结束航段。

引入 0-1 变量 d_{ik}^{s} 描述机组人员 k 是否以航段 i 作为执勤起始航段，若是则取 1，否则为 0；0-1 变量 d_{ik}^{f} 描述机组人员 k 是否以航段 i 作为执勤结束航段，若是则取 1，否则为 0。

同天内的执勤起始航段和结束航段可以看作一个执勤内部的连接，但是基于航段对连接变量其数量为 $F \times F \times C$，因此基于同一天至多存在一个执勤这个约束，我们对代表同一执勤连接的变量进行简化，只取起始航段和结束航段设置 0-1 变量。变量设置与说明如图7所示。

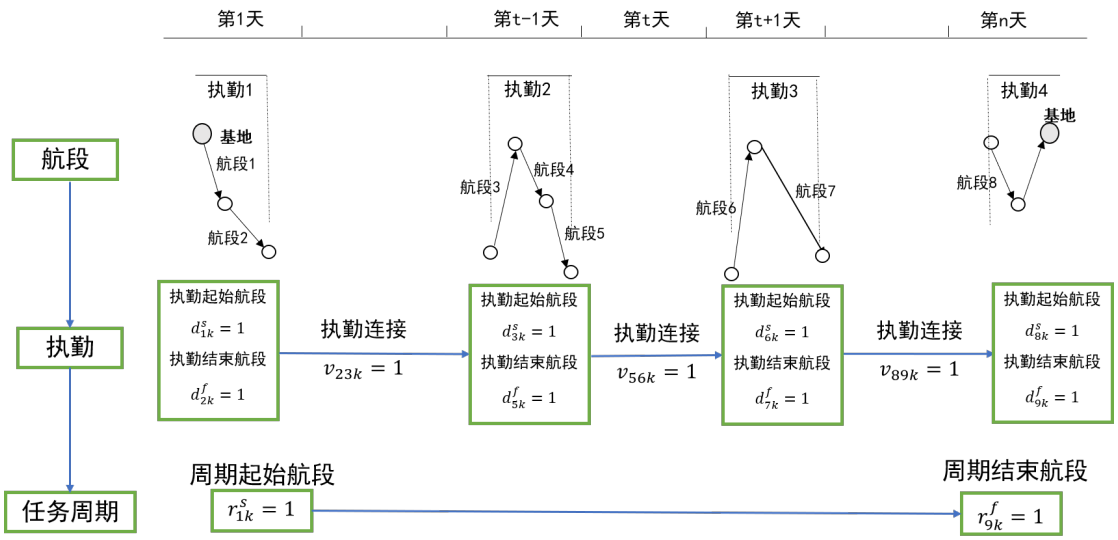

图 7　问题二的变量说明图

对同一天内的起始航段和结束航段之间存在执行任务的航段，需要满足问题一所给的基本条件，同时需要满足问题二飞行时长和执勤时长的要求。

通过上述分析，保持问题一所建立的约束 1-10 不变，在此基础上新增下述约束：

约束条件 12　保证每个机组人员每天至多只能执行一次执勤，对任一天 t 有

$$\sum_{\forall i \in F_t^{date}} u_{ik}^{sta} \geq, \quad \forall t \in Date, \tag{5.1}$$

$$\sum_{\forall i \in F_t^{date}} u_{ik}^{sta} - \sum_{\forall i \in F_t^{date}} u_{ik}^{sta} = 0, \quad \forall t \in Date, \tag{5.2}$$

其中 F_t^{date} 为所有出发日期为第 t 天的航段集合，约束 (5.1) 保证第 t 天内至多存在一个执勤开始航段，即最多存在一次执勤；约束 (5.2) 保证一天内的执勤起始航段存在对应的执勤结束航段。

约束条件 13　保证每个机组人员当天若存在执行航段任务，则必定有执勤，若不存在执行航段，则不存在执勤：

$$M \sum_{\forall i \in F_t^{Date}} u_{ik}^{s} \geq \sum_{\forall i \in F_t^{Date}} (x_i k^{cap} + x_i k^{fo} + x_i k^{dh}), \quad \forall t \in Date, \tag{5.3}$$

$$\sum_{\forall i \in F_t^{Date}} u_{ik}^{s} \leq \sum_{\forall i \in F_t^{Date}} (x_{ik}^{cap} + x_{ik}^{fo} + x_{ik}^{dh}), \quad \forall t \in Date, \tag{5.4}$$

其中 $\sum_{\forall i \in F_t^{Date}} (x_{ik}^{cap} + x_{ik}^{fo} + x_{ik}^{dh})$ 说明第 t 天机组人员 k 是否存在执行航段，若不存在取值为 0，有对应 u_{ik}^{s} 取值为 0，即该机组人员当天不不存在执勤；式 (5.4) 保证当若第 t 天机组人员 k 存在执行航段，则存在对应的 u_{ik}^{s} 取值为 1，即该机组人员当天存在执勤。

约束条件 14　保证执勤的连接符合航段连接的基本要求：

$$v_{ijk} \leq y_{ijk}, \quad \forall i, j \in F, \forall k \in C, \tag{5.5}$$

约束条件 15　保证每个机组人员下一执勤的起始机场必须和上一执勤的结束机场保持一致：

$$v_{ijk} = 0, \quad \forall (i,j) \in FF_3, \forall k \in C, \tag{5.6}$$

其中 FF_3 为航班组合的集合，对 $\forall (i,j) \in FF_3$，有航段 i 的到达机场和航段 j 的到达机场不一致。

约束条件 16　保证每个机组人员的相邻两个执勤之间的休息时间不小于 MinRest 分钟：

$$v_{ijk} = 0, \quad \forall (i,j) \in FF_4, \forall k \in C, \tag{5.7}$$

其中 FF_4 为航班组合的集合，对 $\forall (i,j) \in FF_3$，有 $T_i^a + \text{MinRest} > T_j^l$，即航段 i 的和航段 j 之间的间隔时间不满足执勤间隔时间要求。

约束条件 17　对于一个排班周期内，存在若干执勤和执勤连接，一个执勤可能是该排班周期的起始执勤或者存在前序执勤，也有可能是该排班周期的结束执勤或者存在后序执勤，保证每个执勤之间存在连接：

$$d_{ik}^s = r_{ik}^s + \sum_{\forall j \in F} v_{ijk}, \quad \forall i \in F, \forall k \in C, \tag{5.8}$$

$$d_{ik}^f = r_{ik}^f + \sum_{\forall j \in F} v_{jik}, \quad \forall i \in F, \forall k \in C, \tag{5.9}$$

约束条件 18　保证每次执勤的时长最多不超过 MaxDP 分钟：

$$\sum_{i \in F_{Date_t}} d_{ik}^f T_i^a - \sum_{i \in F_{Date_t}} d_{ik}^s T_i^l \le MaxDP, \quad \forall t \in Date, \forall k \in C, \tag{5.10}$$

约束条件 19　保证每次执勤的飞行时间最多不超过 MaxBlk 分钟：

$$\sum_{i \in F_{Date_t}} (x_{ik}^{fo} + x_{ik}^{cap})(T_i^a - T_i^l) \le MaxBlk, \quad \forall t \in Date, \forall k \in C, \tag{5.11}$$

2. 第二阶段的数学模型

(1) 目标函数

多目标的优先级依次为最小化执勤成本，最小化乘机次数，平衡机组人员的执勤时长，最小化乘机次数。

针对最小化总体执勤成本，T_i^l 表示航段 i 的起飞时间，T_i^a 表示航段 i 的到达时间，$DCost_k$ 表示机组人员 k 的执勤成本，目标函数如下所示，记为 f_2：

$$\min f_2 = \sum_{k \in C} (\sum_{i \in F} d_{ik}^f T_i^a - \sum_{i \in F} d_{ik}^s T_i^l) Dcost_k, \tag{5.12}$$

其中 $\sum_{i \in F} d_{ik}^f T_i^a - \sum_{i \in F} d_{ik}^s T_i^l$ 为机组人员 k 的执勤时长和。

针对平衡机组人员之间的执勤时长要求，考虑到模型线性化，引入机组人员最大执勤时长 $MaxDTime$ 和最小执勤时长 $MinDTime$，引入辅助约束 1，目标函数记为 f_4：

$$\min f_4 = MaxDT - MinDT, \tag{5.13}$$

辅助约束 1:

$$MaxDT \ge \sum_{i \in F} d_{ik}^f T_i^a - \sum_{i \in F} d_{ik}^s T_i^l, \quad , \forall k \in C, \tag{5.14}$$

$$MinDT \leq \sum_{i \in F} d_{ik}^{f} T_i^{a} - \sum_{i \in F} d_{ik}^{s} T_i^{l}, \quad , \forall k \in C, \tag{5.15}$$

在目标函数 (4.19) 的基础上，设置权重 μ_2, μ_5，得到问题二第二阶段的单目标函数如下所示，记为 f_{22}：

$$\min f_{22} = \mu_2 f_2 + \mu_4 f_4 + \mu_5 f_5 + \mu_7 f_7 \tag{5.16}$$

(2) 约束条件

保持问题一的两阶段模型约束 1-约束 11 不变，保持问题二第一阶段的约束 12-约束 19 不变，增加上述辅助约束 1，得到该模型为：

$$\begin{aligned} &\min f_{22} \\ s.t. &\text{约束式}(4.2) - (4.20), \\ &\text{约束式}(5.1) - (5.11), \\ &\text{辅助约束式}(5.14) - (5.15). \end{aligned} \tag{5.17}$$

5.1.2 改进数学模型的建立

针对问题二，我们已经得到考虑执勤的两阶段机组排班数学规划模型，在该模型中由每天的航段组成当天的执勤，再由一个排班周期内的执勤组成整个排班周期。

在实际数据测试中，在问题一的模型上增加变量 v_{ijk}，该变量规模与 y_{ijk} 一致，取决于航班规模和机组人员数量，约束条件数量与变量数量呈指数级增长，因此我们根据该模型提出一个改进优化模型，通过减少变量数量与约束条件数量对模型进行改进。

该优化模型的基本思想有：我们将一个排班周期分割成一定数量的时间块，时间块由执勤组成，时间块与时间块之间的连接与执勤之间的连接要求一致。每个时间块内执勤与执勤之间的连接，航段与航段之间的连接仍保持原问题要求，目标函数不变。该模型具体建立方法如下所示：

时间块与时间块之间的连接其实是人为添加的特殊执勤与执勤之间的连接，因此对每个时间块的起始执行航班和结束执行航班的约束进行适量的松弛，通过上一时间块所得到的解构建下一时间块的可行解，从而将基于排班周期的多阶段优化模型转化为若干个有一定次序的相关基于时间块的多阶段优化模型。

将该排班周期分成 n 个连续的时间块，记时间块分别 $TS_1, TS_2, \ldots, TS_n$，记 F_{TSm} 为起飞时间在时间块 m 内的航班，多阶段模型构造过程如下所示，流程图如8所示。

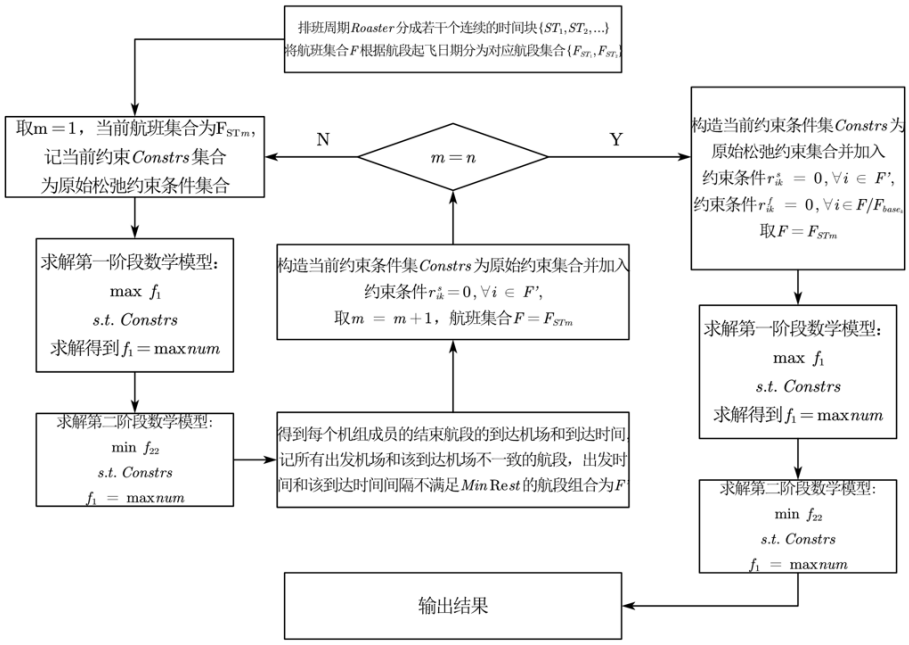

图 8　改进规划模型的求解流程图

时间块 1 的数学模型

保持两阶段数学模型求解不变，在第一阶段中，保持原目标函数 f_1 不变，保持约束条件 1-6, 约束条件 8-20 不变，将其对航班的取值范围更改为在该时间块内的航班集合 F_{ST_1}，其余集合均保持不变。

在约束 7 中，式 (4.11) 规定每个机组人员必须回到基地，但是在执勤与执勤之间的连接，并没有要求机组人员回到基地，因此在时间块模型中对该约束进行松弛，允许机组人员可以不回到所在基地，得到时间块 1 第一阶段模型如下所示：

$$\begin{aligned} &\min f_1 \\ s.t.&\text{约束式}(4.2)-(4.10), \\ &\text{约束式}(4.12)-(4.16), \\ &\text{约束式子}(5.1)-(5.11), \end{aligned} \tag{5.18}$$

在得到第一阶段目标函数的解，将其作为阶段二中的约束，更改目标函数为 f_{22} 得到第二阶段的数学模型。

时间块 2 的数学模型

通过对基于时间块 1 的数学规划模型求解，我们可以得到不同变量的具体取值，得到机组人员 k 的时间块 1 的结束航班的到达机场和到达时间，时间块 2 的数学规划模型中，对于每一个机组人员 k，其时间块 2 和时间块 1 之间构成一个时间块连接，要求其起始航段所在机场和上一时间段的结束航班的到达机场航段，其起始航段的起飞时间和上一时间块的结束航段的到达时间满足执勤之间连接最小时间要求 MinRest，保持优化模型 (5.18) 的目标函数, 删去对起始航段的起飞机场约束 (4.10)，保持约束不变，增加下述约束：

约束条件 21　保证时间块 2 的出发航段和时间块 1 的到达航段之间符合连接要求，其中 F_{ST2} 指所有不满足上述要求的航段集合：

$$r_{ik}^s = 0, \quad \forall i \in F_{ST2}, \forall k \in C, \tag{5.19}$$

对时间块 2 的第一阶段数学模型进行求解，根据上述第二阶段的数学模型构建方式构建相应的第二阶段数学模型。

以此类推，对时间块 3，时间块 4 等依次构造数学模型并进行求解。

.....

时间块 n 的数学模型

以此类推，我们可以对每个时间块模型进行依次求解，并将上一时间块所得到的取值结果构造相应约束，加入到下一时间块的数学规划模型之中作为约束条件，再对下一时间块的数学规划模型进行求解，在最后时间块的数学模型中，重新加入之前删去的约束 (4.11)，要求所有机组人员回到基地，完成排班周期的约束要求。得到时间块 n 的第一阶段模型如下所示：

$$\begin{aligned} &\min f_1 \\ s.t.&\text{约束}(4.2)-(4.9), \\ &\text{约束}(4.11)-(4.16), \\ &\text{约束}(5.1)-(5.11), \\ &r_{ik}^{s}=0, \quad \forall i \in F_{ST_m}, \forall k \in C. \end{aligned} \tag{5.20}$$

求解后构造相应的第二阶段数学模型。

该改进模型通过将一个排班周期分割成若干个时间块，考虑了执勤与执勤之间连接的约束要求，将原模型进行松弛，分解为若干个相关的基于时间块的机组排班问题，并通过上一时间块的解更改下一阶段时间块的约束条件，在丢失较少优化解的同时大大减少了变量数量和约束条件数量，提高求解效率。

5.1.3 改进数学模型的求解

对于 A 数据集，我们仍然通过 Python 调用 Gurobi 求得了多阶段改进数学模型的最优排班策略，成功**为 206 架次即所有的航班进行了人员排班**，并仅有 **21 位机组人员执行了乘机任务，替补资格使用次数为 0，机组总体利用率为 100%，总执勤成本为 53.3473 万元**。其结果如表6所示。

表 5　问题一 Gurobi 求解 a 数据集成功结果指标

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	21
替补资格使用次数	0
机组总体利用率	100%
最小/平均/最大一次执勤飞行时长	65/147.02/520
最小/平均/最大一次执勤执勤时长	65/162.29/680
最小/平均/最大机组人员执勤天数	15/15/15
总执勤成本	53.3473
程序运行分钟数	39.38

通过甘特图可视化排班效果如图9所示，由于执勤因素的出现，机组排班变得更加稀疏了，同时为了给更多的航班排班，增加了一定的乘机次数。

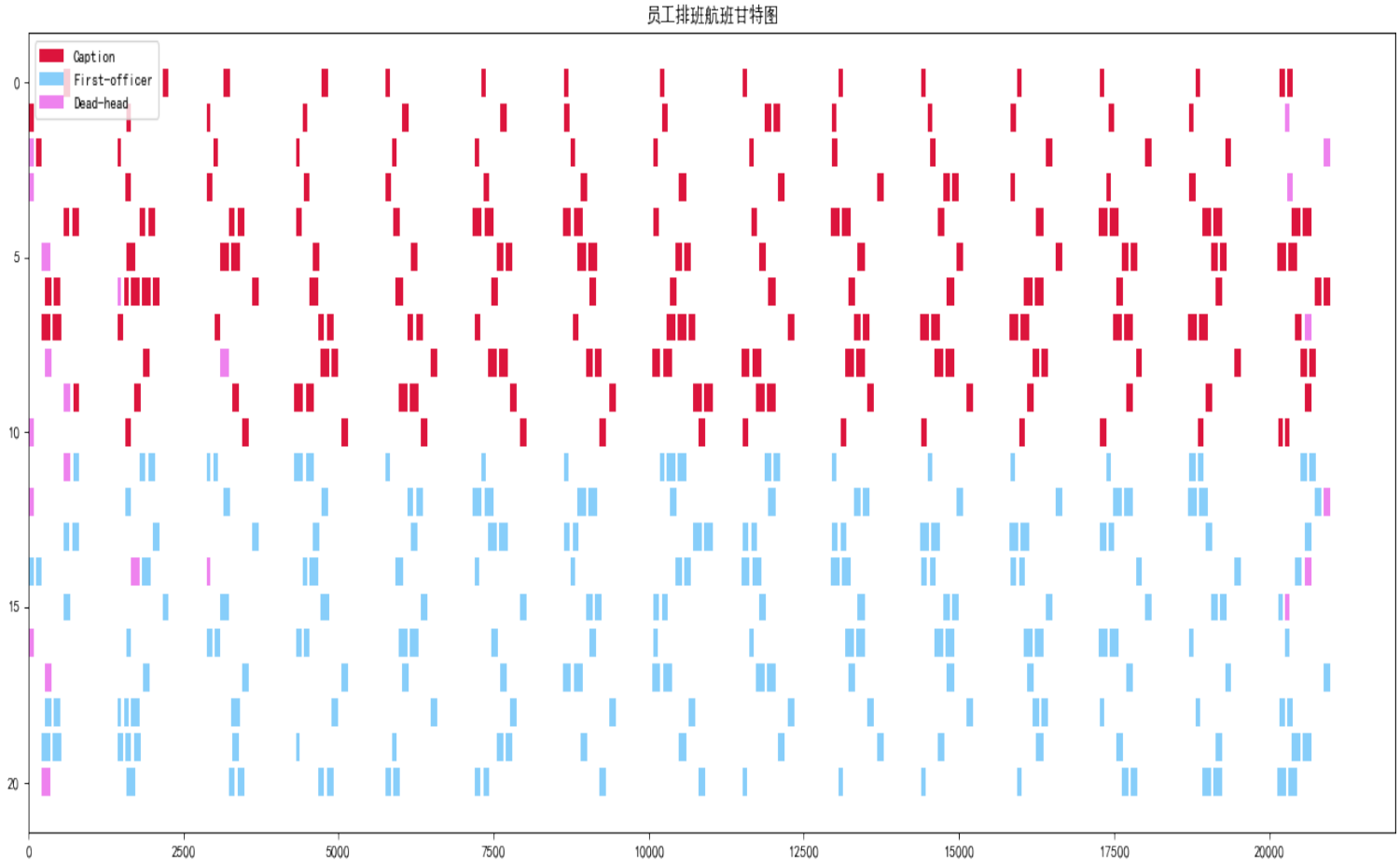

图 9　问题二 Gurobi 求解 a 数据集航班排班甘特图

对数据集 a 中每个机组人员，其总执勤时长与平均执勤时长如图10所示

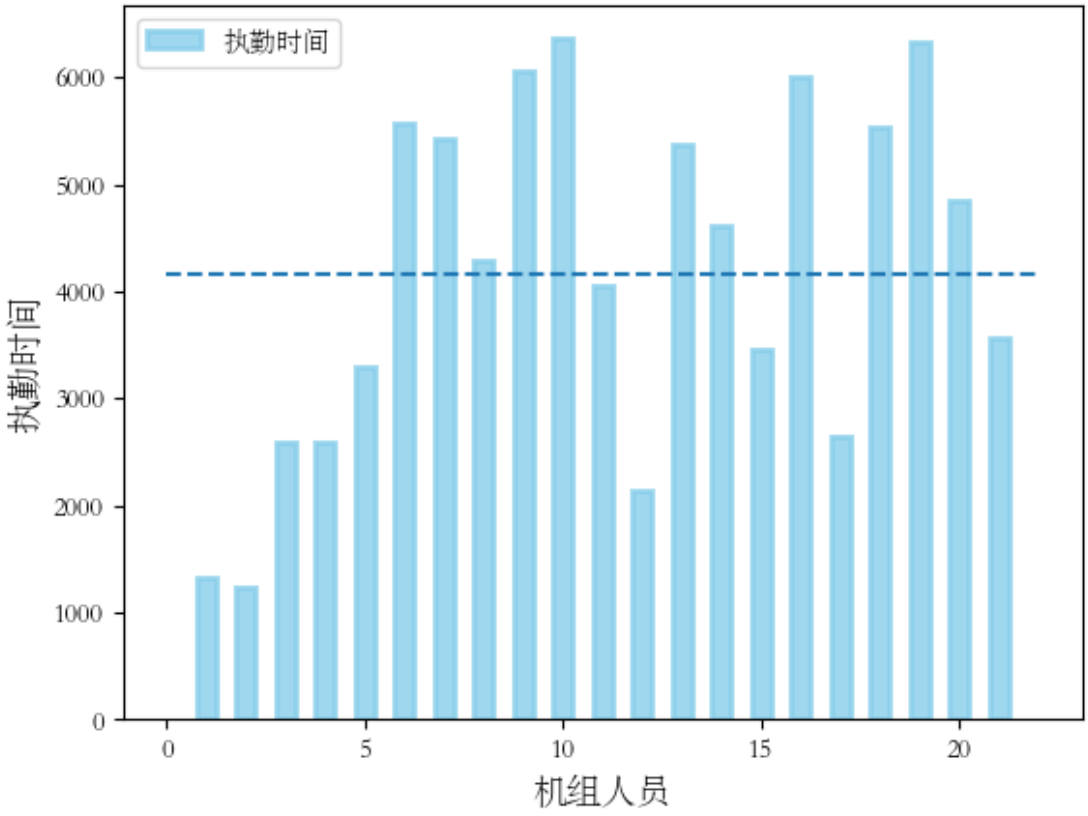

图 10　问题二数据集 a 机组人员执勤时长图

5.2 优化算法的设计与求解

5.2.1 优化算法的设计

问题二需要考虑最小化执勤成本，尽可能平衡执勤时长，在问题一所给出的优化算法的基础上进行改进完善，该改进优化算法的主要分为三个阶段，第一阶段仍未查找所有的可行任务集合，第二阶段为配对机组人员，其具体流程如下所示：

生成可行任务的贪婪算法在问题一中所给出的后序航段搜索中，仅考虑了满足时间间隔的和起飞机场要求的航段，考虑执勤约束，允许加入有效路径的点需要满足：

a. 加入后当日执勤总时长不超过 MaxDP;

b. 加入后当日飞行总时长不超过 MaxBlk;

c. 若该航段为次日的第一个航段，则与前续航段之间时间不小于 MinRest。

同时考虑到为了平衡机组人员的平衡时长，若一条有效路径增加一个航段后回到基地，则停止该有效路径对后序航段的遍历，生成一个可行任务，有利于后序进行匹配。

具体的算法思路的流程如图11所示:

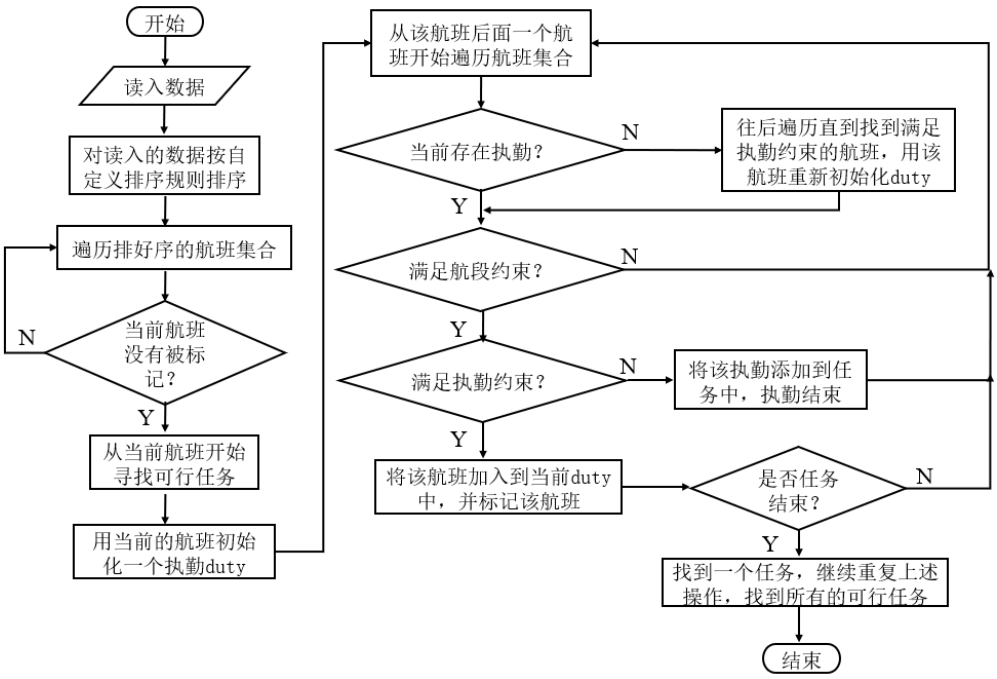

图 11　查找可行任务流程图

机组人员与可行任务的匹配贪婪算法

在问题一所提出的贪婪算法的基础上，考虑最小化执勤成本的优化目标，进行改进，将配对机组人员根据其执勤成本之和升序排序，对可行任务集合根据各个任务的执勤时长降序排序，依次进行配对，使得执勤成本。

5.2.2 优化算法的求解

根据问题一的对数据集 A 的测试结果说明该算法在规模较小的数据集上具有良好的性能和测试结果。直接将该算法应用在大规模数据集 B 上。

对数据集 B 的求解

根据所给算法，找出其中的可行任务集合，共计找到 3512 个可行任务，其中基地”HOM”的可行任务数为：236，基地“TGD”的可行任务数为：3276。

然后进行人员的配对，其中基地”HOM”的配对数为 36，基地”TGD”的配对数为 163。然后让配对机组人员挑选可行任务，最终得到满足机组配置的航班数为 11682，不满足机组配置的航班数为 2272。具体结果如表6所示。

表 6　问题一优化算法求解 b 数据集成功结果指标

不满足机组配置航班数	2272
满足机组配置航班数	11682
机组人员总体乘机次数	0
替补资格使用次数	12
机组总体利用率	85.59%
最小/平均/最大一次执勤飞行时长	45/220.81/545
最小/平均/最大一次执勤执勤时长	45/316.00/720
最小/平均/最大机组人员执勤天数	0/21.72/31
总执勤成本	3341.53
程序运行分钟数	0.0114

甘特图如图12所示。

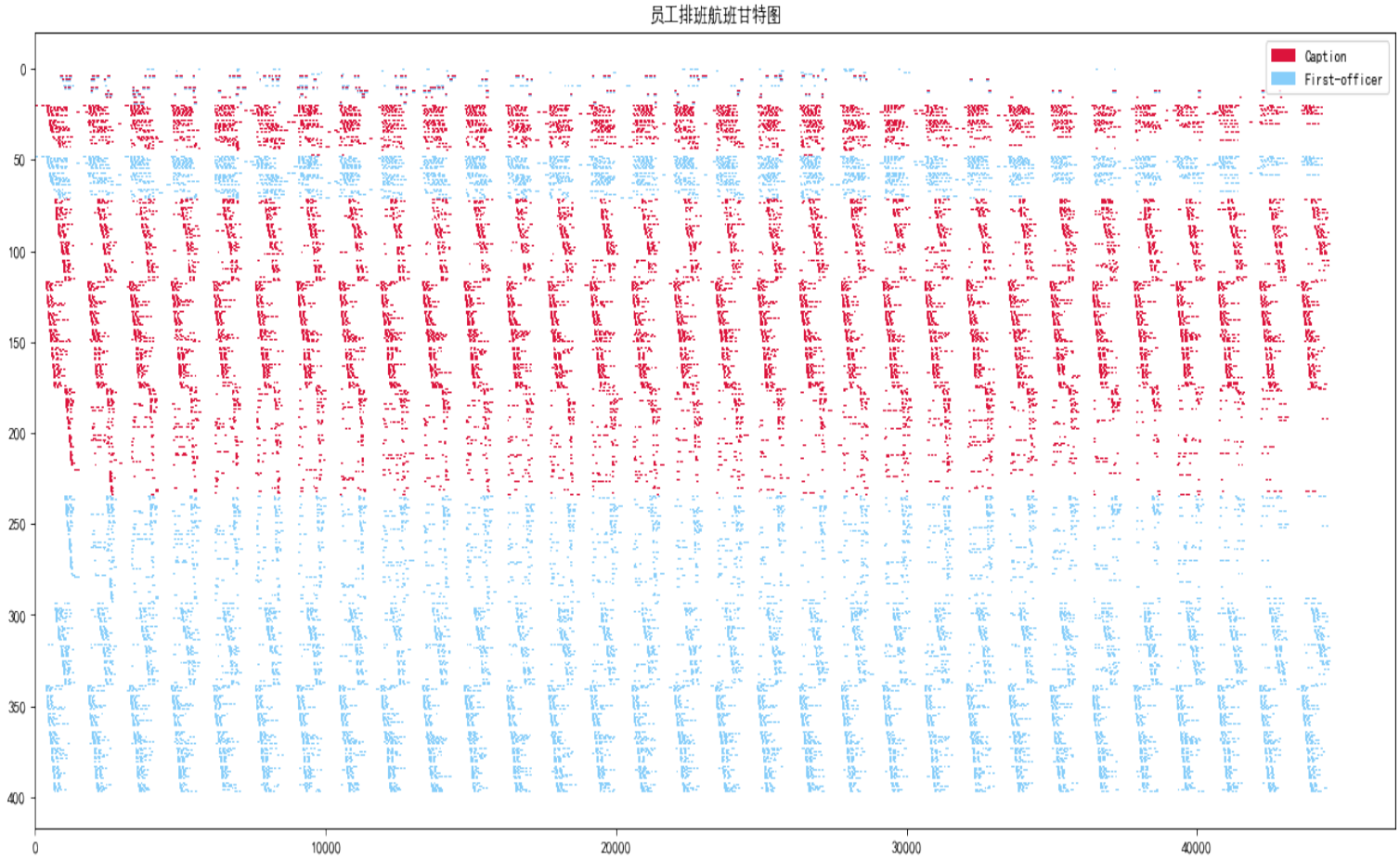

图 12　问题二优化算法求解 b 数据集航班排班甘特图

6. 问题三模型的求解及算法设计

6.1 基于任务环的数学模型的建立与求解

6.1.1 数学模型的建立

问题三在问题二的基础上，引入了任务环的概念。任务环由若干天的执勤和执勤和执勤之间的连接组成，任务周期由若干个任务环和任务环和任务环组成

其最高优先级目标仍为最大化满足配置的航班数量，次优先级目标依次为：最小化机组人员总体执勤成本；最小化机组人员任务环总成本；最小化乘机次数；机组人员之间的执勤时长尽可能平衡；机组人员的任务环时长尽可能平衡；最小化替补资格次数。

针对问题三，我们在问题二所提出的模型基础上建立新的两阶段 0-1 数学规划模型，结合新的优化目标重新设计目标函数，并针对问题三的要求设计新的约束条件。

数学规划模型具体建立过程如下：

1. 第一阶段数学模型

(1) 目标函数

保持问题一中的第一阶段所给出的目标函数 f_1 不变，该目标函数为最大化符合机组配置要求的航班数量。

(2) 约束条件

在问题二的模型中，我们建立同天执勤起始航段到执勤结束航段，上一执勤结束航段到下一执勤起始航段的开始，将一个排班周期划分成若干执勤和执勤与执勤之间的连接。

问题三引入任务环的概念与相关约束，一个任务环由若干个执勤和执勤之间的休息组成，因此上一任务环与下一任务环之间的连接实质是一种特殊的执勤与执勤之间的连接，在满足执勤连接的基础上，需要额外满足对任务环之间的约束要求。我们引入 0-1 变量 w_{ijk} 说明航段 i 和航段 j 之间是否存在任务环连接，即若机组人员 k 以航段 i 作为上一执勤的结束航段，航段 j 作为下一执勤的起始航段则 v_{ijk} 取值为 1，否则为 0。

一个航段若作为任务环起始航段，则其必为一个执勤起始航段，若该航段不是排班周期起始航段，则其必存在前序任务环；一个航段若作为任务环结束航段，则其必为一个执勤结束航段，若该航段不是排班周期结束航段，则其存在一个后序任务环。任何作为任务环起始或者任务环结束的航段，必须满足从基地出发并且回到基地的要求。

同时考虑变量环内部的连接，引入 0-1 变量 u_{ijk} 描述机组人员 k 是否以航段 i 和航段 j 分别作为一个任务环的起始航段和结束航段，若是则取 1，否则为 0；对任一任务环起始航段必定存在唯一与之对应的任务环结束航段。变量设置与说明如图13所示。

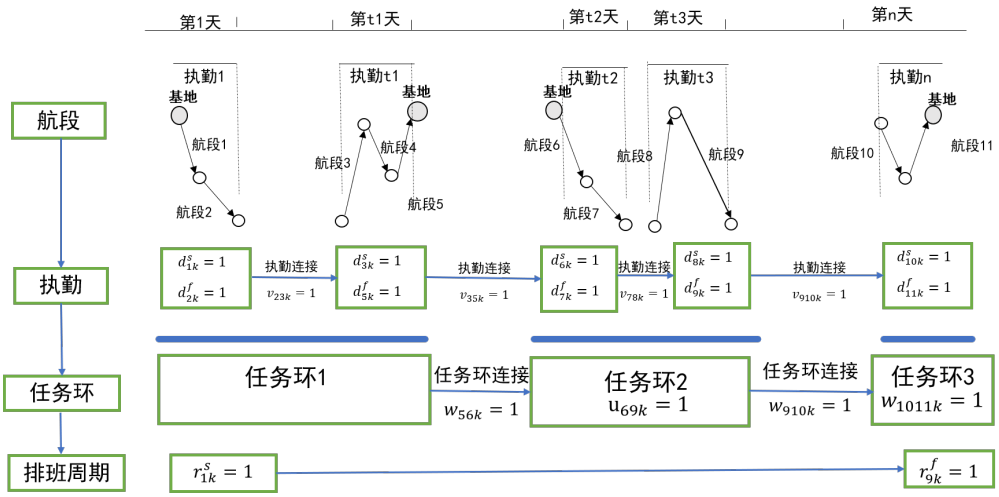

图 13　问题三的变量说明示意图

一个排班周期可以视为若干个任务环起始航段和任务环结束航段之间的连接，从一个任务环起始航段，同时也是排班周期起始航段开始，连接至其对应的任务环结束航段，再连接至下一任务环的起始航段，依次进行连接，最终到达一个任务环结束航段，同时也是排班周期结束航段。具体变量描述如下图所示：

通过上述分析，我们保持问题一与问题二所得到的约束 1-约束 10，约束 12-约束 21 保持不变，新增下述约束：

约束条件 23　保证上一任务环与下一任务环之间的连接满足基本对执勤与执执勤之间的连接要求：

$$w_{ijk} \leq v_{ijk}, \quad \forall i,j \in F, \forall k \in C, \tag{6.1}$$

约束条件 24　保证一个排班周期起始航段必定为一个任务环起始航段，一个排班结束航段必定为一个任务环结束航段：

$$r^s_{ik} \leq \sum_{j \in F} v_{ijk}, \quad \forall i \in F, \forall k \in C, \tag{6.2}$$

约束条件 25　保证一个任务环起始航段必定为一个执勤起始航段，一个任务环结束航段必定为一个执勤结束航段：

$$\sum_{j \in F} v_{ijk} \leq d^s_{ik}, \quad \forall i \in F, \forall k \in C, \tag{6.3}$$

$$\sum_{j \in F} v_{jik} \leq d^s_{ik}, \quad \forall i \in F, \forall k \in C, \tag{6.4}$$

约束条件 26　保证任务环起始航段，结束航段的流量相等：

$$\sum_{j \in F} v_{ijk} - (r^s_{ik} + \sum_{j \in F} w_{jik}) = 0, \quad \forall i \in F, \forall k \in C, \tag{6.5}$$

$$\sum_{j \in F} v_{jik} - (r^f_{ik} + \sum_{j \in F} w_{ijk}) = 0, \quad \forall i \in F, \forall k \in C, \tag{6.6}$$

式 (6.5) 说明若当前航班为任务环起始航段，则其为排班周期起始航段或存在一个上一任务环结束航段到该起始航段的连接；式 (6.6) 说明若当前航班为任

务环结束航段，则其为排班周期结束航段或存在一个到下一任务环起始航段的连接；

约束条件 27　保证任务环从基地出发并且回到基地：

$$u_{ijk} = 0, \quad \forall (i,j) \in FF_{5k}, \forall k \in C, \tag{6.7}$$

其中 FF_5 为航班与航班组合的集合，对 $\forall (i,j) \in FF_{5k}$, 有航段 i 不从机组人员 k 所在基地出发或者航段 j 不到达机组人员 k 所在基地。

约束条件 28　保证任务环连续执勤天数不超过 $MaxSuccOn$ 天：

$$u_{ijk} = 0, \quad \forall (i,j) \in FF_6, \forall k \in C, \tag{6.8}$$

其中 FF_6 为航班与航班组合的集合，对 $\forall (i,j) \in FF_6$, 有航段 i 的起飞日期与航段 j 的起飞日期相差超过 $MaxSuccOn$ 天。

约束条件 29　保证每个机组人员每个排班周期的任务环总时长不超过 $MaxTAFB$ 分钟：

$$\sum_{i \in F} \sum_{j \in F} w_{ijk}(T_j^a - T_i^l) \leq MaxTAFB, \quad \forall k \in C, \tag{6.9}$$

其中 T_j^l 表示航段 i 的到达时间，T_i^a 表示航段 i 的到达时间。

2. 第二阶段的数学模型

(1) 目标函数

多目标的优先级依次为最小化执勤成本，最小化任务环成本，最小化乘机次数，平衡机组人员的执勤时长，平衡机组人员的任务环时长，最小化乘机次数。

针对最小化任务环成本，$RCost_k$ 表示机组人员 k 的执勤成本，目标函数如下所示，记为 f_3：

$$\min f_3 = \sum_{k \in C} RCost_k \sum_{i \in F} \sum_{j \in F} w_{ijk}(T_j^a - T_i^l), \tag{6.10}$$

针对平衡机组人员之间的任务环时长，考虑到模型线性化，引入机组人员最大任务环时长 $MaxRT$ 和最小任务环时长 $MinRT$，引入辅助约束 2，目标函数记为 f_6：

$$\min f_6 = MaxRT - MinRT, \tag{6.11}$$

辅助约束 2:

$$MaxRT \geq \sum_{i \in F} \sum_{j \in F} w_{ijk}(T_j^a - T_i^l), \quad \forall k \in C, \tag{6.12}$$

$$MinRT \leq \sum_{i \in F} \sum_{j \in F} w_{ijk}(T_j^a - T_i^l), \quad \forall k \in C, \tag{6.13}$$

在目标函数 (5.16) 的基础上，设置权重 μ_3, μ_6，得到问题三第二阶段的单目标函数如下所示，记为 f_32：

$$\min f_32 = \mu_2 f_2 + \mu_3 f_3 + \mu_4 f_4 + \mu_5 f_5 + \mu_6 f_6 + \mu_7 f_7 \tag{6.14}$$

(2) 约束条件

保持问题一两阶段模型约束 1-约束 11 不变，问题二的约束 12-约束 21，辅助约束 1，问题三第一阶段的约束 23- 约束 29 不变，增加上述辅助约束 2，得到问题三第二阶段模型为：

$$\begin{aligned} &\min f_3 2 \\ &s.t. \text{约束式}(4.2)-(4.20), \\ &\quad \text{约束式}(5.1)-(5.11), \\ &\quad \text{约束式}(6.1)-(6.9), \\ &\quad \text{辅助约束 1}(5.14)-(5.15), \\ &\quad \text{辅助约束 2}(6.12)-(6.13). \end{aligned} \tag{6.15}$$

6.1.2 改进优化模型的建立

针对问题三，我们可以在问题二给出的改进优化模型的基础上考虑任务环的概念，在该模型中由每天的航段组成当天的执勤，若干个执勤组成一个任务环，再由若干个任务环组成整个排班周期。将基于排班周期的多阶段优化模型转化为若干个有一定次序的相关基于时间块的多阶段优化模型。该模型具体建立方法如下所示：

将该排班周期分成 n 个连续的时间块，记时间块分别 $TS_1, TS_2, \ldots, TS_n$，F_{TS_m} 为起飞时间在时间块 m 内的航班，多阶段模型构造过程如下所示：

时间块 1 的数学模型

保持两阶段数学模型求解不变，在第一阶段中，保持原目标函数 f_1 不变，保持约束 1-10，约束 12-19，约束 22-29，将其对航班的取值范围更改为在该时间块内的航班集合 F_{ST_1}，其余集合均保持不变，得到时间块 1 第一阶段模型。

在得到第一阶段目标函数的解，将其作为阶段二中的约束，更改目标函数为 $f_2 2$ 得到第二阶段的数学模型。

时间块 2，3 . . . n 的数学模型

通过对基于时间块 n-1 的两阶段数学规划模型求解，我们可以得到不同变量的具体取值，得到机组人员 k 的时间块 n-1 的结束航班的到达时间，在时间块 n 的数学规划模型中，对于每一个机组人员 k，其时间块 n 和时间块 n-1 之间构成一个时间块连接，要求其起始航段的起飞日期和上一时间块的结束航段的到达日期满足最小任务环休假要求 MinVacDay，保持时间块 n-1 中的约束不变，增加下述约束：

约束条件 30 保证时间块 n-1 的出发航段和时间块 1 的到达航段之间符合连接要求，其中 F_{STn} 指所有不满足上述要求的航段集合：

$$r_{ik}^{s} = 0, \quad \forall i \in F_{STn}, \forall k \in C, \tag{6.16}$$

6.1.3 改进数学模型的求解

对于 A 数据集，通过使用 Python 调用 Gurobi 求得了多阶段改进数学模型的最优排班策略，**成功为 206 架次即所有的航班进行了人员排班**，并**有 56 位机组人员执行了乘机任务，替补资格使用次数为 74**，以此灵活应对引进任务环所带来的限制，**机组总体利用率为 100%，总执勤成本为 53.1256 万元**。其结果如表7所示。通过甘特图可视化排班效果如图14所示。

表 7　问题三 Gurobi 求解 a 数据集成功结果指标

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	56
替补资格使用次数	74
机组总体利用率	100%
最小/平均/最大一次执勤飞行时长	65/298.51/775
最小/平均/最大一次执勤执勤时长	65/374.35/855
最小/平均/最大机组人员执勤天数	4/8/12
一/二/三/四天任务环数量分布	190/105/70/135
总体执勤成本（万元）	53.1256
总体任务环成本（万元）	6.27533
程序运行分钟数	1.9648

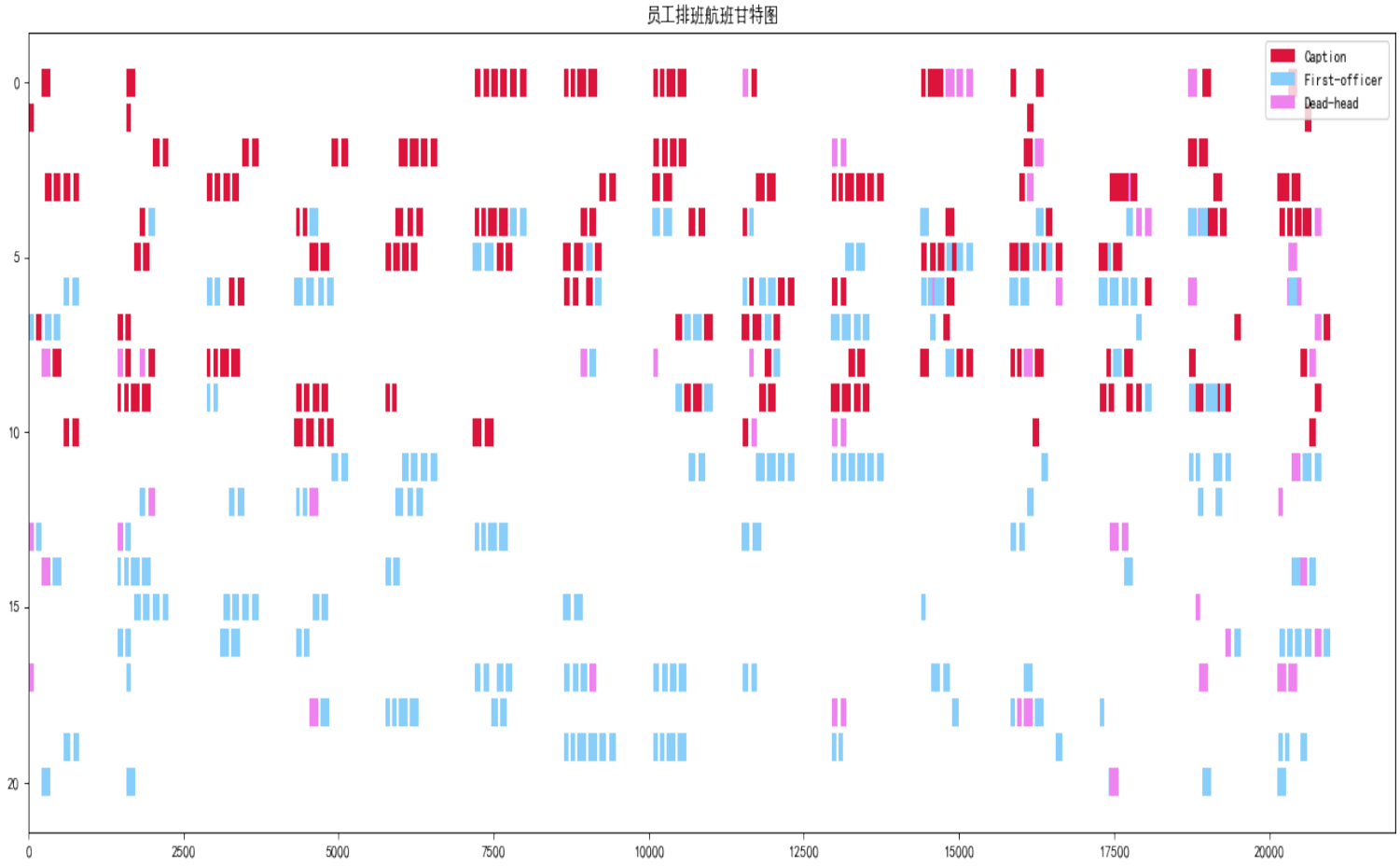

图 14　问题三数据集 a 中机组人员执行任务甘特图

对数据集 a 中每个机组人员，其总任务环时长与平均任务环时长如图15所示

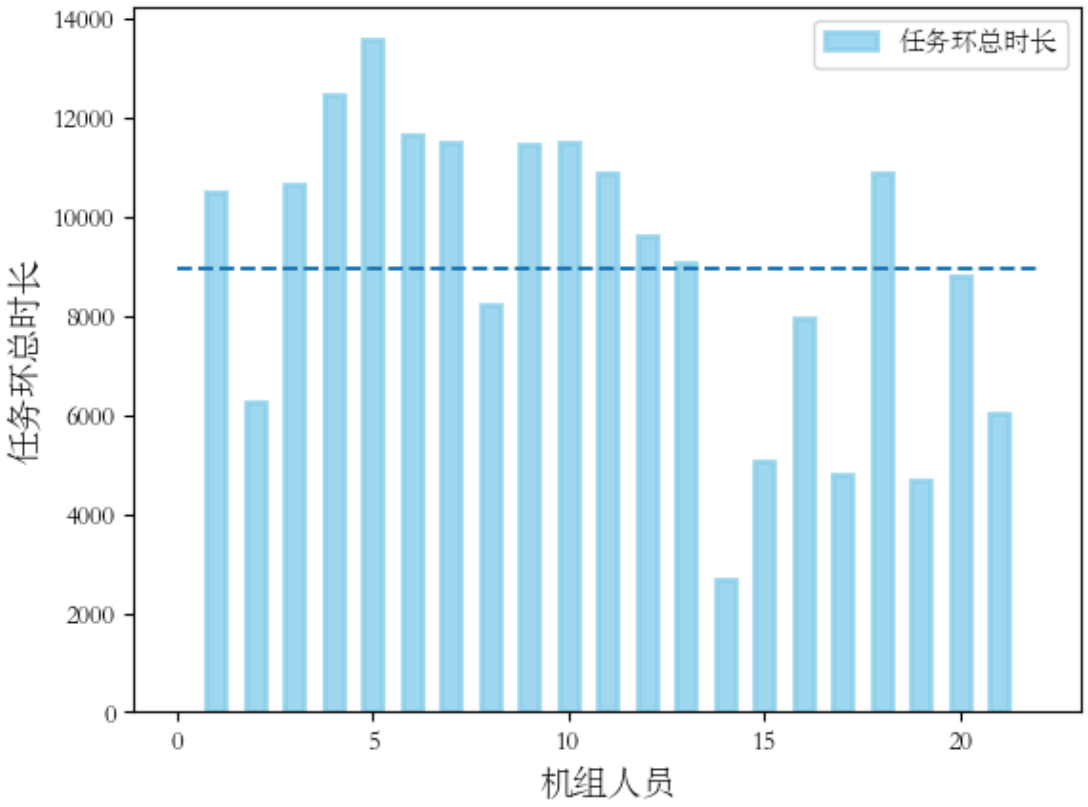

图 15　问题三数据集 a 中机组人员任务环总时长图

6.2 优化算法的设计与求解

6.2.1 优化算法的设计

考虑对任务环和执勤天数的约束，以问题二所设计的寻找可行任务的贪婪算法为基础，得到从基地出发并且返回基地的若干可行任务，如果该可行任务满足任务环市场约束以及执勤天数约束，则该可行任务即为一个可行任务环。

对任一可行任务因为其只在起始和结束时访问基地，所以一个可行任务中不存在有效路径片段可以生成一个可行任务环，在可行任务集合中删去所有不符合连续执勤要求和任务环约束的可行任务。得到可行任务环集合。

对可行任务环和机组人员进行分配，需要满足上一任务环和下一任务环中休假时间符合 MinVacDay 约束

该算法流程如下所示：

Step 1. 根据问题二得到若干可行路径集合；

Step 2. 在该可行路径集合中删去不满足任务环时长约束和执勤天数约束的可行任务；

Step3. 在对机组人员分配任务环时，考虑任务环间隔约束。

6.2.2 优化算法的求解

针对数据集 B 的求解，根据上述算法流程，共计找到 3512 个可行任务，删去其中不符合要求的可行任务集合，得到共计 3329 个满足要求的任务环集合，其中基地”HOM”的可行任务环数为 176，基地“TGD”的可行任务数为 3153。

对人员进行配最终得到满足机组配置的航班数为 4769，不满足机组配置的航班书为 9158，具体结果如表8所示。

表 8　问题三优化算法求解 b 数据集成功结果指标

不满足机组配置航班数	9185
满足机组配置航班数	4769
机组人员总体乘机次数	0
替补资格使用次数	2
机组总体利用率	81.29%
最小/平均/最大一次执勤飞行时长	50/235.06/545
最小/平均/最大一次执勤执勤时长	50/327.10/720
最小/平均/最大机组人员执勤天数	0/8.37/16
一/二/三/四天任务环数量分布	1566/120/24/17
总体执勤成本（万元）	1335.75
总体任务环成本（万元）	58.28
程序运行分钟数	0.012

甘特图如图16所示。

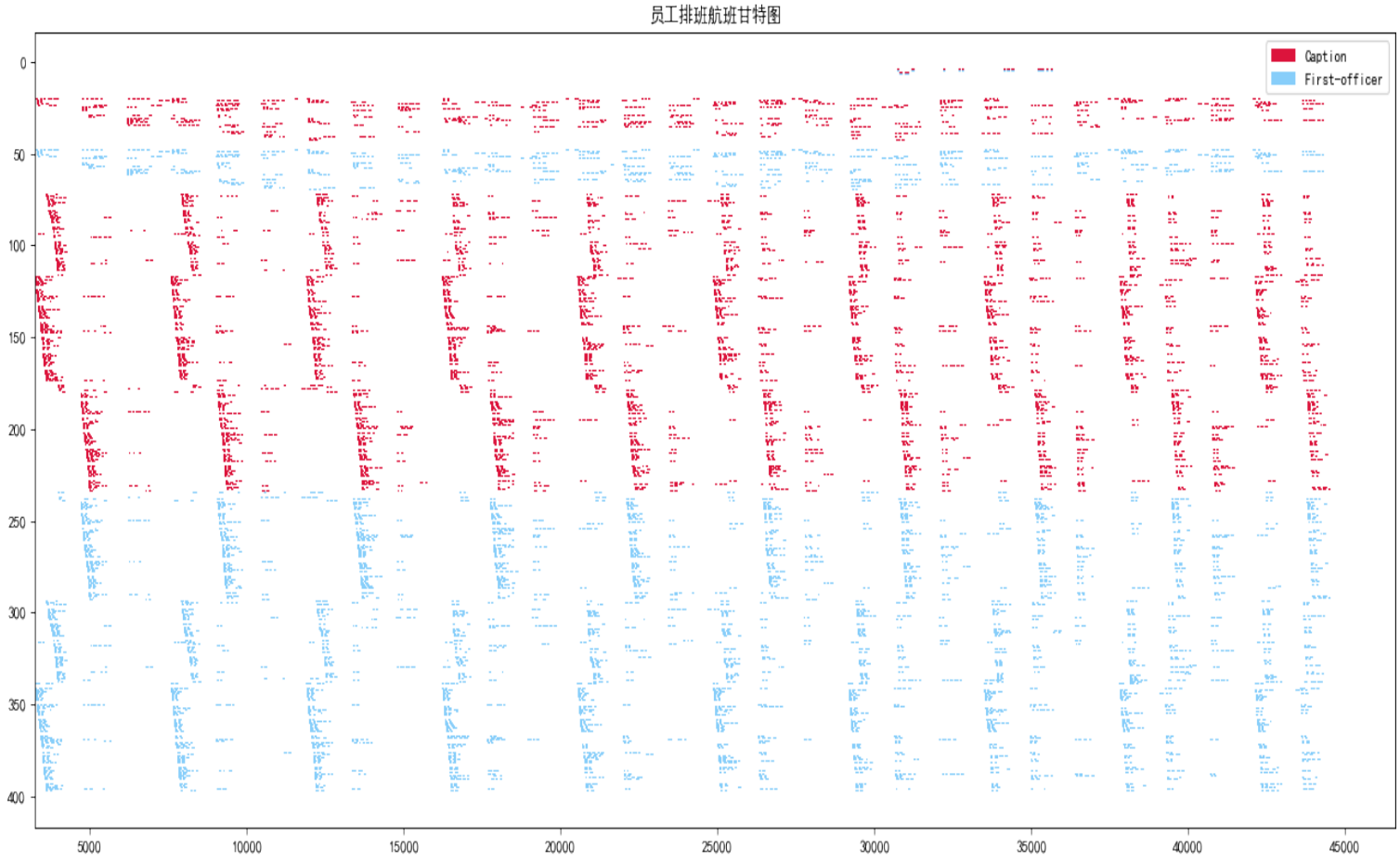

图 16　问题三优化算法求解 b 数据集航班排班甘特图

7. 结果与分析

本赛题的三个子问题均是多目标优化问题，而且后一个子问题是前一个子问题的拓展，约束条件与优化目标层层拓展。

针对问题一，我们设计了基于航班和机组人员的组合两阶段的 0-1 整数规划模型，最大化符合最低配置要求的航段数量，并在此基础上最小化换乘次数和替补次数。调用 Gurobi 求解器对数据集 A 求解。对于规模较大的数据集 B，我们设计了时间复杂度为 $O(n^2)$ 的基于贪婪算法的多阶段启发式算法，首先通过对航段的搜索构造可行路径，然后与机组人员进行匹配，最后考虑换乘因素增加满足要求的航段数量，其在小规模数据集上的计算结果与计算时间都呈现较好的性能，在大规模数据集 B 上应用该算法。

针对问题二，我们在问题一的模型的基础上，引入执勤的概念，扩展约束条件，设计了两阶段的 0-1 整数规划模型，最大化符合最低配置要求的航段数量，根据权重在问题一的次优化目标的基础上最小化总执勤成本和平衡总体执勤时长。为了减少模型变量数量和约束条件数量，设计了基于时间块的多阶段改进 0-1 整数规划模型，将上一阶段的求解结果构造新的约束加入下一阶段，分阶段进行求解，调用 Gurobi 求解器对改进模型在数据集 A 上求解。在考虑执勤的基础上，对问题一的启发式算法进行改进，考虑最小成本的要求将可行路径松弛为可行任务，构造机组人员和可行任务之间的分配问题。

针对问题三，考虑任务环，在问题二的数学模型的基础上扩展设计了两阶段的 0-1 整数规划模型，最大化符合最低配置要求的航段数量，在问题二的次优化的基础上最小化总任务环成本和平衡总体任务环时长。设计相应的改进 0-1 整数规划模型，类似地拓展问题二的贪婪算法，将可行任务严格限制得到可行任务环，再与机组人员进行匹配。

对数据集 A 有计算结果如表9所示。

表 9　数据集 A 的成功结果指标

指标	问题 1	问题 2	问题 3
不满足机组配置航班数	0	0	0
满足机组配置航班数	206	206	206
机组人员总体乘机次数	0	21	56
替补资格使用次数	0	0	74
机组总体利用率	100%	100%	100%
最小/平均/最大一次执勤飞行时长		65/147.02/520	65/298.51/775
最小/平均/最大一次执勤执勤时长		65/162.29/680	65/374.35/855
最小/平均/最大机组人员执勤天数		15/15/15	4/8/12
一/二/三/四天任务环数量分布			190/105/70/135
总体执勤成本（万元）		53.3473	53.1256
程序运行分钟数	2.06	39.38	1.9648

对数据集 B 有计算结果如表10所示。

表 10　数据集 B 的成功结果指标

指标	问题 1	问题 2	问题 3
不满足机组配置航班数	543	2272	9185
满足机组配置航班数	13411	11682	4769
机组人员总体乘机次数	0	0	0
替补资格使用次数	12	12	2
机组总体利用率		85.59%	81.29%
最小/平均/最大一次执勤飞行时长		45/220.81/545	50/235.06/545
最小/平均/最大一次执勤执勤时长		45/316.00/720	50/327.10/720
最小/平均/最大机组人员执勤天数		0/21.72/31	0/8.37/16
一/二/三/四天任务环数量分布			1566/120/24/17
总体执勤成本（万元）		3341.53	1335.75
程序运行分钟数	0.0009	0.0114	0.012

8. 参考文献

[1] 张米 . 航空公司机组排班模型研究 [D]. [S.l.]: 清华大学, 2014.

[2] 赵正佳 . 航空公司机组排班计划研究 [J]. 运筹与管理, 2011, 000(006): 106–113.

[3] 崔如玉 . 飞机排班问题模型及算法研究 [D]. [S.l.]: 北京交通大学, 2018.

[4] 李青 , 张军 , 张学军 . 解决排班问题的多目标优化模型及算法研究 [J]. 北京航空航天大学学报, 2003(09): 821–824.

附录 A 程序代码

1. 数据读取文件 preprocessing.py

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

from Flight import Flight
from Employee import Employee


class DataLoader():
  def __init__(self, path_employees, path_flights):
      self.df_employees = pd.read_csv(path_employees).fillna(0)
      self.df_flights = pd.read_csv(path_flights).fillna(0)
      self.employees = []
      self.flights = []

      self.C = []
      self.C1 = []
      self.C2 = []
      self.C3 = []
      self.C1_base = []
      self.C2_base = []
      self.C3_base = []
      self.C_base = []
      self.C_dic = {}

      self.F = []
      self.F_dic = {}
      self.F_ap_dpt_dic = {}
      self.F_ap_arr_dic = {}
      self.F_leave_base = []
      self.F_arrive_base = []
      self.nonF_leave_base = []
      self.nonF_arrive_base = []

      self.FF = []
      self.FF1 = []
      self.FF2 = []
      self.FD = {}

      self.AP = []
      self.AP_dic = {}
      self.AP_f_dpt_dic = {}
      self.AP_f_arr_dic = {}

      self.Base = []
      self.Base_dic = {}

      self.arrivedate = {}
      self.leavedate = {}
      self.arrivetime = {}
```

```
        self.leavetime = {}
        self.leavetimestr = {}
        self.arrivetimestr = {}

        self.Dates = []

        self.DCost = {}
        self.PCost = {}

        self.min_day = self.get_min_day()
        self.min_time = 10000

    def dump_data(self, cropped_date=None, use_optimize=True):
        # dump employees
        self.employees = []
        for i, row in enumerate(self.df_employees.iterrows()):
            EmpNo = row[1][0]
            if row[1][1] == 'Y':
                Caption = True
            else:
                Caption = False
            if row[1][2] == 'Y':
                FirstOfficer = True
            else:
                FirstOfficer = False
            if row[1][3] == 'Y':
                Deadhead = True
            else:
                Deadhead = False
            Base = row[1][4]
            DutyCostPerHour = row[1][5]
            ParingCostPerHour = row[1][6]
            empl = Employee(i, EmpNo, Caption, FirstOfficer,
                Deadhead, Base,
                                DutyCostPerHour, ParingCostPerHour)
            self.employees.append(empl)
            self.C_dic[i] = EmpNo

            self.DCost[i] = DutyCostPerHour
            self.PCost[i] = ParingCostPerHour
        # dump flights
        self.flights = []
        i = 0
        for row in self.df_flights.iterrows():
            FltNum = row[1][0]
            DptrDate= int(row[1][1].split('/')[1])
            DptrTime =
                60*int(row[1][2].split(':')[0])+int(row[1][2].split(':')[1])
            DptrStn = row[1][3]
            ArrvDate = int(row[1][4].split('/')[1])
            ArrvTime =
                60*int(row[1][5].split(':')[0])+int(row[1][5].split(':')[1])
            ArrvStn = row[1][6]
            if cropped_date is None:
```

```
            flight = Flight(i, FltNum, DptrDate, DptrTime, DptrStn,
                            ArrvDate, ArrvTime, ArrvStn)
            self.flights.append(flight)
            self.F_dic[i] = flight.get_unique_number()
            if DptrDate not in self.FD:
               self.FD[DptrDate] = [i]
            else:
               self.FD[DptrDate].append(i)
            self.leavedate[i] = DptrDate
            self.arrivedate[i] = ArrvDate
            self.leavetimestr[i] = row[1][2]
            self.arrivetimestr[i] = row[1][5]
            i+=1
         else:
            if DptrDate >= cropped_date[0] and DptrDate
               <=cropped_date[1]:
               flight = Flight(i, FltNum, DptrDate, DptrTime,
                  DptrStn,
                            ArrvDate, ArrvTime, ArrvStn)
               self.flights.append(flight)
               self.F_dic[i] = flight.get_unique_number()
               if DptrDate not in self.FD:
                  self.FD[DptrDate] = [i]
               else:
                  self.FD[DptrDate].append(i)
               self.leavedate[i] = DptrDate
               self.arrivedate[i] = ArrvDate
               self.leavetimestr[i] = row[1][2]
               self.arrivetimestr[i] = row[1][5]
               i+=1

   self.min_time = self.get_min_time()
   self.get_times()
   self.get_C()
   self.get_AP()
   self.get_Base()
   self.get_F()
   if use_optimize:
      self.get_FF()
   self.get_non_F()
   self.get_times()

def get_times(self):
   fl_len = len(self.flights)
   i = 0
   for flight in self.flights:
      if i % 100 == 0:
         print("Counting times...({}/{})".format(i,fl_len))
      i+=1
      self.leavetime[flight.index] =
         24*60*(flight.dpt_date-1)+flight.dpt_time-self.min_time
      self.arrivetime[flight.index] =
         24*60*(flight.arr_date-1)+flight.arr_time-self.min_time
```

```
# get different crew
def get_C(self):
    print("## Loading C")
    self.C = list(self.C_dic.keys()) # all crew
    self.C1 = [] # crew only with caption
    for empl in self.employees:
        if empl.cap_enable and not empl.fo_enable:
            self.C1.append(empl.index)
    self.C2 = [] # crew with caption and first officer
    for empl in self.employees:
        if empl.cap_enable and empl.fo_enable:
            self.C2.append(empl.index)
    self.C3 = [] # crew only with first officer
    for empl in self.employees:
        if empl.fo_enable and not empl.cap_enable:
            self.C3.append(empl.index)
    print("## C loaded")

# get airport
def get_AP(self):
    print("## Loading AP")
    self.AP = [] # all airports
    i = 0
    for flight in self.flights:
        if flight.dpt_stn not in self.AP_dic.values():
            self.AP_dic[i] = flight.dpt_stn
            self.AP.append(i)
            i+=1
        if flight.arr_stn not in self.AP_dic.values():
            self.AP_dic[i] = flight.arr_stn
            self.AP.append(i)
            i+=1

    print("## AP loaded")

# get base
def get_Base(self):
    print("## Loading Base")
    self.Base = [] # all bases
    self.C_base = [] # crew with different bases
    i = 0
    for emplo in self.employees:
        if emplo.base not in self.Base_dic.values():
            self.Base.append(i)
            self.Base_dic[i] = emplo.base
            i+=1
            temp_a = []
            temp_b = []
            temp_c = []
            temp_d = []
            self.C_base.append(temp_a)
            self.C1_base.append(temp_b)
            self.C2_base.append(temp_c)
            self.C3_base.append(temp_d)
```

```
        for key, value in self.Base_dic.items():
            if value == emplo.base:
                base_index = key
        self.C_base[base_index].append(emplo.index)
        if emplo.cap_enable and not emplo.fo_enable:
            self.C1_base[base_index].append(emplo.index)
        elif emplo.cap_enable and emplo.fo_enable:
            self.C2_base[base_index].append(emplo.index)
        elif emplo.fo_enable and not emplo.cap_enable:
            self.C3_base[base_index].append(emplo.index)
    print("## Base loaded")

# get flight
def get_F(self):
    print("## Loading F")
    self.F = [] # all flights
    self.F_leave_base = [] # all leave base flights
    self.F_arrive_base = [] # all arrive base flights
    self.Dates = []
    for i in range(len(self.Base)):
        temp_a = []
        temp_b = []
        self.F_leave_base.append(temp_a)
        self.F_arrive_base.append(temp_b)

    for flight in self.flights:
        self.F.append(flight.index)
        for i in range(len(self.Base)):
            if (flight.dpt_stn == self.Base_dic[i]):
                self.F_leave_base[i].append(flight.index)
            if (flight.arr_stn == self.Base_dic[i]):
                self.F_arrive_base[i].append(flight.index)
            if (flight.dpt_date not in self.Dates):
                self.Dates.append(flight.dpt_date)

        self.F_ap_dpt_dic[flight.index] =
            self.get_AP_index(flight.dpt_stn)
        self.F_ap_arr_dic[flight.index] =
            self.get_AP_index(flight.arr_stn)
        if self.get_AP_index(flight.dpt_stn) not in
            self.AP_f_dpt_dic.keys():
            self.AP_f_dpt_dic[self.get_AP_index(flight.dpt_stn)] =
                [flight.index]
        else:
            self.AP_f_dpt_dic[self.get_AP_index(flight.dpt_stn)].append(flight.ind
        if self.get_AP_index(flight.arr_stn) not in
            self.AP_f_arr_dic.keys():
            self.AP_f_arr_dic[self.get_AP_index(flight.arr_stn)] =
                [flight.index]
        else:
            self.AP_f_arr_dic[self.get_AP_index(flight.arr_stn)].append(flight.ind
    print("## F loaded")

# get the minimize day in dataset.
```

```
def get_min_day(self):
    day_min = 1000
    for row in self.df_flights.iterrows():
        this_day = int(row[1][1].split('/')[1])
        if this_day < day_min:
            day_min = this_day
    return day_min

def get_max_day(self):
    day_max = 0
    for row in self.df_flights.iterrows():
        this_day = int(row[1][1].split('/')[1])
        if this_day > day_max:
            day_max = this_day
    return day_max

def divide_days(self):
    return (self.get_max_day - self.get_min_day + 1) / 2

def get_min_time(self):
    min_day = self.min_day
    min_time = 1000000
    for flight in self.flights:
        if flight.dpt_time < min_time and flight.dpt_date <=
            min_day:
            min_time = flight.dpt_time
    return 24*60*(min_day-1) + min_time

# get False Flights in matrix
def get_FF(self, minCT=40, MinRest=660):
    print("## Loading FF")
    self.FF = []
    # FF_matrix = np.zeros([len(self.flights),len(self.flights)])
    len_FF = len(self.flights) * len(self.flights)
    count = 0
    for i, flight_a in enumerate(self.flights):
        for j, flight_b in enumerate(self.flights):
            count += 1
            if count % 10000 == 0:
                print("Loading data... ({}/{})".format(count,
                    len_FF))
            arrv_time_i = flight_a.get_arrv_gap()
            dptr_time_j = flight_b.get_dptr_gap()
            if flight_a.arr_stn == flight_b.dpt_stn and
                dptr_time_j - arrv_time_i >= minCT:
                continue

            else:
                # FF_matrix[i][j]=1
                self.FF.append((i,j))
    print("## FF loaded")
    print("## Loading FF1")
    self.FF1 = []
    count = 0
```

```
    for i, flight_a in enumerate(self.flights):
        for j, flight_b in enumerate(self.flights):
            count += 1
            if count % 10000 == 0:
                print("Loading data... ({}/{})".format(count,
                    len_FF))
            arrv_time_i = flight_a.get_arrv_gap()
            dptr_time_j = flight_b.get_dptr_gap()
            if flight_b.dpt_date > flight_a.dpt_date and
                flight_a.arr_stn == flight_b.dpt_stn and
                dptr_time_j - arrv_time_i >= MinRest:
                continue
            else:
                # FF_matrix[i][j]=1
                self.FF1.append((i,j))
    print("## FF1 loaded")
    print("## Loading FF2")
    self.FF2 = []
    count = 0
    for i, flight_a in enumerate(self.flights):
        for j, flight_b in enumerate(self.flights):
            count += 1
            if count % 10000 == 0:
                print("Loading data... ({}/{})".format(count,
                    len_FF))
            arrv_time_i = flight_a.get_arrv_gap()
            dptr_time_j = flight_b.get_dptr_gap()
            if flight_b.dpt_date - flight_a.dpt_date > 2 and
                flight_a.arr_stn == flight_b.dpt_stn and
                flight_a.arr_stn in self.Base_dic.values():
                continue
            else:
                # FF_matrix[i][j]=1
                self.FF2.append((i,j))
    print("## FF2 loaded")


def get_non_F(self):
    for i in range (len(self.Base)):
        temp_a = []
        temp_b = []
        self.nonF_leave_base.append(temp_a)
        self.nonF_arrive_base.append(temp_b)
        self.nonF_leave_base[i] = list(set(self.F) -
            set(self.F_leave_base[i]))
        self.nonF_arrive_base[i] = list(set(self.F) -
            set(self.F_arrive_base[i]))

def get_AP_index(self, ap_name):
    for key, value in self.AP_dic.items():
        if value == ap_name:
            return key
```

2. 结果展示文件 ResultViewer.py

```
import matplotlib.pyplot as plt
import pandas as pd
from collections import OrderedDict


class RViewer(object):
   def __init__(self, dataloader, data_cls="a"):
      super(object).__init__()
      self.dl = dataloader
      self.mathched_e_f = []
      self.mathched_e_f_dic = {'em_no':[], 'fl_no':[], 'cls':[]}
      self.mathched_cap_dic = {'em_no':[], 'fl_no':[]}
      self.matched_fo_dic = {'em_no':[], 'fl_no':[]}
      self.matched_dh_dic = {'em_no':[], 'fl_no':[]}
      self.dics = []
      self.data_cls = data_cls

      self.duty_flight_time_dic = {}
      self.duty_time_dic = {}
      self.duty_dates_dic = {}


   def load_results(self, matched_e_f):
      self.mathched_e_f = matched_e_f
      self.generate_dics()

   def generate_dics(self):
      for (em, fl, cls) in self.mathched_e_f:
         self.mathched_e_f_dic['em_no'].append(em)
         self.mathched_e_f_dic['fl_no'].append(fl)
         self.mathched_e_f_dic['cls'].append(cls)
         if cls == 1:
            self.mathched_cap_dic['em_no'].append(em)
            self.mathched_cap_dic['fl_no'].append(fl)
         elif cls == 2:
            self.matched_fo_dic['em_no'].append(em)kb
            self.matched_fo_dic['fl_no'].append(fl)
         elif cls == 3:
            self.matched_dh_dic['em_no'].append(em)
            self.matched_dh_dic['fl_no'].append(fl)
      self.dics = [self.mathched_cap_dic, self.matched_fo_dic,
         self.matched_dh_dic]

   def draw_ef_gantt(self, duration=None, people=None,
      save=None):
      plt.rcParams['font.sans-serif'] = ['SimHei']
      plt.rcParams['axes.unicode_minus'] = False
      plt.figure(figsize=(20,8),dpi=80)
      ax = plt.gca()
      ax.invert_yaxis()
      colors = ['#DC143C', '#87CEFA', '#EE82EE']
      labels = ['Caption', 'First-officer', 'Dead-head']
      for (em_no, fl_no, cls) in self.mathched_e_f:
```

```
            if duration is not None:
                if self.dl.leavedate[fl_no] > duration:
                    continue
            if people is not None:
                if em_no > people:
                    continue
            plt.barh(em_no,
                     self.count_flight_time(fl_no),
                     left=self.dl.leavetime[fl_no],
                     color=colors[cls-1],
                     label=labels[cls-1])
        handles, labels = plt.gca().get_legend_handles_labels()

        by_label = OrderedDict(zip(labels, handles))

        plt.legend(by_label.values(), by_label.keys())
        plt.title("员工排班航班甘特图")

        if save is not None:
            plt.savefig(save)

    def load_results_from_str_df(self, df):
        self.mathched_e_f = []
        for row in df.iterrows():
            em_no = self.find_emp_no(row[1][0])
            fl_no =
                self.find_fl_no(str(row[1][1])+"/"+str(row[1][2]))
            if row[1][3] == 'C':
                cls = 1
            elif row[1][3] == 'F':
                cls = 2
            elif row[1][3] == 'D':
                cls = 3
            self.mathched_e_f.append((em_no, fl_no, cls))
        self.generate_dics()

    def load_results_from_df(self, df):
        self.mathched_e_f = []
        for row in df.iterrows():
            em_no = row[1][0]
            fl_no = row[1][1]
            cls = row[1][2]
            self.mathched_e_f.append((em_no, fl_no, cls))
        self.generate_dics()

    def count_flight_time(self, fl_no):
        return self.dl.arrivetime[fl_no]-self.dl.leavetime[fl_no]

    def find_emp_no(self, emp_name):
        for key, value in self.dl.C_dic.items():
            if value == emp_name:
                return key

    def find_fl_no(self, fl_name):
```

```
        for key, value in self.dl.F_dic.items():
            if value == fl_name:
                return key
        print("not find"+str(fl_name))

    def get_results_df_b(self):
        results_dic = {'em_no':[],
                       'leg_no':[],
                       'fl_no':[],
                       'dpt_date':[],
                       'dpt_time':[],
                       'dpt_ap':[],
                       'arr_date':[],
                       'arr_time':[],
                       'arr_ap':[],
                       'cls':[]}
        for i in range(len(self.dl.C)):
            fl_list = []
            for (em_no, fl_no, cls) in self.mathched_e_f:
                if em_no == i:
                    fl_list.append((fl_no,cls))
            for j in range(len(fl_list)):
                for k in range(j+1,len(fl_list)):
                    if self.dl.leavetime[fl_list[k][0]] <
                        self.dl.leavetime[fl_list[j][0]]:
                        temp = fl_list[k]
                        fl_list[k] = fl_list[j]
                        fl_list[j] = temp
            m = 1
            for (fl_no, cls) in fl_list:
                results_dic['em_no'].append(self.dl.C_dic[i])
                results_dic['leg_no'].append(m)
                results_dic['fl_no'].append(self.dl.F_dic[fl_no].split('/')[0])
                if self.data_cls == "b":
                    results_dic['dpt_date'].append("8/"+str(self.dl.leavedate[fl_no])
                elif self.data_cls == "a":
                    results_dic['dpt_date'].append("8/"+str(self.dl.leavedate[fl_no])
                results_dic['dpt_time'].append(self.dl.leavetimestr[fl_no])
                results_dic['dpt_ap'].append(self.dl.AP_dic[self.dl.F_ap_dpt_dic[fl_
                if self.data_cls == "b":
                    results_dic['arr_date'].append("8/"+str(self.dl.arrivedate[fl_no]
                elif self.data_cls == "a":
                    results_dic['arr_date'].append("8/"+str(self.dl.arrivedate[fl_no]
                results_dic['arr_time'].append(self.dl.arrivetimestr[fl_no])
                results_dic['arr_ap'].append(self.dl.AP_dic[self.dl.F_ap_arr_dic[fl_
                if cls == 1:
                    results_dic['cls'].append('C')
                elif cls == 2:
                    results_dic['cls'].append('F')
                elif cls == 3:
                    results_dic['cls'].append('D')
                m+=1
        return pd.DataFrame(results_dic)
```

```
  def get_results_df_a(self):
     results_dic = {'fl_no':[],
                    'dpt_date':[],
                    'dpt_time':[],
                    'dpt_ap':[],
                    'arr_date':[],
                    'arr_time':[],
                    'arr_ap':[],
                    'comp':[]}
     fl_list = []
     fl_leave_list = []
     for (em_no, fl_no, cls) in self.mathched_e_f:
        if fl_no not in fl_list:
           fl_list.append(fl_no)
     for fl in self.dl.F:
        if fl not in fl_list:
           fl_leave_list.append(fl)
     for i in range(len(fl_leave_list)):
        for j in range(i+1,len(fl_leave_list)):
           if self.dl.leavedate[fl_leave_list[j]] <
              self.dl.leavedate[fl_leave_list[i]]:
              temp = fl_leave_list[j]
              fl_leave_list[j] = fl_leave_list[i]
              fl_leave_list[i] = temp
           elif self.dl.leavedate[fl_leave_list[j]] ==
              self.dl.leavedate[fl_leave_list[i]]:
              if self.dl.leavetime[fl_leave_list[j]] <
                 self.dl.leavetime[fl_leave_list[i]]:
                 temp = fl_leave_list[j]
                 fl_leave_list[j] = fl_leave_list[i]
                 fl_leave_list[i] = temp
              elif self.dl.leavetime[fl_leave_list[j]] ==
                 self.dl.leavetime[fl_leave_list[i]]:
                 if self.dl.F_ap_dpt_dic[fl_leave_list[j]] <
                    self.dl.F_ap_dpt_dic[fl_leave_list[i]]:
                    temp = fl_leave_list[j]
                    fl_leave_list[j] = fl_leave_list[i]
                    fl_leave_list[i] = temp
                 elif self.dl.F_ap_dpt_dic[fl_leave_list[j]] ==
                    self.dl.F_ap_dpt_dic[fl_leave_list[i]]:
                    if self.dl.F_ap_arr_dic[fl_leave_list[j]] <
                       self.dl.F_ap_arr_dic[fl_leave_list[i]]:
                       temp = fl_leave_list[j]
                       fl_leave_list[j] = fl_leave_list[i]
                       fl_leave_list[i] = temp
     for fl_no in fl_leave_list:
        results_dic['fl_no'].append(self.dl.F_dic[fl_no].split('/')[0])
        if self.data_cls == "b":
           results_dic['dpt_date'].append("8/"+str(self.dl.leavedate[fl_no])+"/
        elif self.data_cls == "a":
           results_dic['dpt_date'].append("8/"+str(self.dl.leavedate[fl_no])+"/
        results_dic['dpt_time'].append(self.dl.leavetimestr[fl_no])
        results_dic['dpt_ap'].append(self.dl.AP_dic[self.dl.F_ap_dpt_dic[fl_no]
        if self.data_cls == "b":
```

```
            results_dic['arr_date'].append("8/"+str(self.dl.arrivedate[fl_no])+"
        elif self.data_cls == "a":
            results_dic['dpt_date'].append("8/"+str(self.dl.leavedate[fl_no])+"/
        results_dic['arr_time'].append(self.dl.arrivetimestr[fl_no])
        results_dic['arr_ap'].append(self.dl.AP_dic[self.dl.F_ap_arr_dic[fl_no]
        results_dic['comp'].append('C1F1')
    return pd.DataFrame(results_dic)

def count_duty_time(self):
    self.duty_flight_time_dic = {}
    self.duty_time_dic = {}
    self.duty_dates_dic = {}
    for i in range(len(self.dl.C)):
        fl_list = []
        for (em_no, fl_no, cls) in self.mathched_e_f:
            if em_no == i:
                fl_list.append((fl_no,cls))
        for j in range(len(fl_list)):
            for k in range(j+1,len(fl_list)):
                if self.dl.leavetime[fl_list[k][0]] <
                    self.dl.leavetime[fl_list[j][0]]:
                    temp = fl_list[k]
                    fl_list[k] = fl_list[j]
                    fl_list[j] = temp
        duty_flight_time = {}
        duty_time = {}
        duty_dates = []
        duty_dpt_time = {}
        duty_arr_time = {}
        for (fl_no, cls) in fl_list:
            if self.dl.leavedate[fl_no] not in duty_dates:
                duty_dates.append(self.dl.leavedate[fl_no])
                duty_flight_time[self.dl.leavedate[fl_no]] =
                    self.dl.arrivetime[fl_no] -
                    self.dl.leavetime[fl_no]
                duty_dpt_time[self.dl.leavedate[fl_no]] =
                    [self.dl.leavetime[fl_no]]
                duty_arr_time[self.dl.leavedate[fl_no]] =
                    [self.dl.arrivetime[fl_no]]
            else:
                duty_flight_time[self.dl.leavedate[fl_no]] +=
                    self.dl.arrivetime[fl_no] -
                    self.dl.leavetime[fl_no]
                duty_dpt_time[self.dl.leavedate[fl_no]].append(self.dl.leavetime[
                duty_arr_time[self.dl.leavedate[fl_no]].append(self.dl.arrivetime
        for day in duty_dates:
            duty_time[day] = max(duty_arr_time[day]) -
                min(duty_dpt_time[day])
        self.duty_flight_time_dic[i] = duty_flight_time
        self.duty_time_dic[i] = duty_time
        self.duty_dates_dic[i] = duty_dates
```

3. 职员类 Employee.py

```
class Employee(object):
def __init__(self, Index, EmpNo, Caption, FirstOfficer,
                Deadhead, Base, DutyCostPerHour,
                ParingCostPerHour):
   super(Employee).__init__()
   self.index = Index
   self.number = EmpNo
   self.cap_enable = Caption
   self.fo_enable = FirstOfficer
   self.dh_enable = Deadhead
   self.base = Base
   self.d_cost = DutyCostPerHour
   self.p_cost = ParingCostPerHour

def show(self):
   print("EmpNo: ", self.number, ' (', type(self.number,), ')')
   print("Caption: ", self.cap_enable, ' (',
      type(self.cap_enable,), ')')
   print("FirstOfficer: ", self.fo_enable, ' (',
      type(self.fo_enable,), ')')
   print("DeadHead: ", self.dh_enable, ' (',
      type(self.dh_enable,), ')')
   print("Base: ", self.base, ' (', type(self.base,), ')')
   print("DutyCostPerHour: ", self.d_cost, ' (',
      type(self.d_cost,), ')')
   print("ParingCostPerHour: ", self.p_cost, ' (',
      type(self.p_cost,), ')')
   print("-----------------------------------")
```

4. 航班类 Flight.py

```
class Flight(object):
def __init__(self, Index, FltNum,
                DptrDate, DptrTime, DptrStn,
                ArrvDate, ArrvTime, ArrvStn):
   super(Flight).__init__()
   self.index = Index
   self.number = FltNum
   self.dpt_date = DptrDate
   self.dpt_time = DptrTime
   self.dpt_stn = DptrStn
   self.arr_date = ArrvDate
   self.arr_time = ArrvTime
   self.arr_stn = ArrvStn

def show(self):
   print("FltNum: " + self.number, ' (', type(self.number,), ')')
   print("DptrDate: " + str(self.dpt_date), ' (',
      type(self.dpt_date,), ')')
   print("DptrTime: " + str(self.dpt_time), ' (',
      type(self.dpt_time,), ')')
   print("DptrStn: " + self.dpt_stn, ' (', type(self.dpt_stn,),
      ')')
   print("ArrvDate: " + str(self.arr_date), ' (',
```

```
        type(self.arr_date,), ')')
    print("ArrvTime: " + str(self.arr_time), ' (',
        type(self.arr_time,), ')')
    print("ArrvStn: " + self.arr_stn, ' (', type(self.arr_stn,),
        ')')
    print("-----------------------------------")

# not minus min_day in dataset
def get_dptr_gap(self):
    return 60*24*self.dpt_date + self.dpt_time

def get_arrv_gap(self):
    return 60*24*self.arr_date + self.arr_time

def get_unique_number(self):
    return self.number+'/'+str(self.dpt_date)
```

5. 问题一 A 数据集求解模型 q1.py

```
from utils.preprocessing import DataLoader
from matplotlib import pyplot as plt
import gurobipy as gp
from gurobipy import *

data_loader_a = DataLoader('data/Data A-Crew.csv', 'data/Data
    A-Flight.csv')
data_loader_a.dump_data()

m=gp.Model('m1')

z=m.addVars(data_loader_a.F,vtype=gp.GRB.BINARY,name='z')
x_ikdh=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='x_dh'
x_ikfo=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='x_fo'
x_ikcap=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='x_ca

r_iksta=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='r_ik
r_jkfin=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='r_jk

y_ijk=m.addVars(data_loader_a.F,data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BIN


m.ModelSense=GRB.MINIMIZE

m.setObjective(-z.sum())
# m.setObjective(x_ikdh.sum())
# m.setObjective(gp.quicksum(x_ikfo[i,k] for i in
    data_loader_a.F for k in data_loader_a.C2))

#m.setObjectiveN(-z.sum(),index = 0,weight =0.8)
#m.setObjectiveN(x_ikdh.sum(),index=1,weight=0.2)
#m.setObjectiveN(gp.quicksum(x_ikfo[i,k] for i in
    data_loader_a.F for k in
    data_loader_a.C2),index=2,weight=0.05)
```

```
M=10000
#对X的约束
m.addConstrs(x_ikfo[i,k]==0 for i in data_loader_a.F for k in
    data_loader_a.C1 )
m.addConstrs(x_ikcap[i,k]==0 for i in data_loader_a.F for k in
    data_loader_a.C3 )
m.addConstrs(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]<=1 for i in
    data_loader_a.F for k in data_loader_a.C)

#对Z的约束
m.addConstrs(gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]
    for k in data_loader_a.C)<=M*(z[i]) for i in data_loader_a.F)
m.addConstrs(x_ikcap.sum(i,'*')== z[i] for i in data_loader_a.F )
m.addConstrs(x_ikfo.sum(i,'*')== z[i] for i in data_loader_a.F )
m.addConstrs(M*z[j]>=gp.quicksum(x_ikcap[j,k]+x_ikfo[j,k]+x_ikdh[j,k]
    for k in data_loader_a.C) for j in data_loader_a.F)

#对Y的约束
m.addConstrs(y_ijk[i,j,k]==0 for i,j in data_loader_a.FF for k
    in data_loader_a.C)
#m.addConstrs(gp.quicksum(y_ijk[i,j,k] for j in
    F)<=x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in F for k in
    C)
#m.addConstrs(gp.quicksum(y_ijk[j,i,k] for j in
    F)<=x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in F for k in
    C)

#对roaster周期的约束
m.addConstrs(r_jkfin[j,k]== 0 for j in
    data_loader_a.nonF_arrive_base[0] for k in data_loader_a.C)
m.addConstrs(r_iksta[i,k]== 0 for i in
    data_loader_a.nonF_leave_base[0] for k in data_loader_a.C)

m.addConstrs(gp.quicksum(r_iksta[i,k] for i in
    data_loader_a.F_leave_base[0] )<=1 for k in data_loader_a.C)
m.addConstrs(gp.quicksum(r_jkfin[j,k] for j in
    data_loader_a.F_arrive_base[0] )-gp.quicksum(r_iksta[i,k] for
    i in data_loader_a.F_leave_base[0] ) == 0 for k in
    data_loader_a.C)

#第一问中航班对应周期的约束
m.addConstrs(1-(y_ijk.sum(i,'*',k)+r_iksta[i,k]+r_jkfin[i,k])<=M*(1-(x_ikcap[i,k
    for i in data_loader_a.F for k in data_loader_a.C)
m.addConstrs(y_ijk.sum(i,'*',k)+r_jkfin[i,k]-(y_ijk.sum('*',i,k)+r_iksta[i,k])
    == 0 for i in data_loader_a.F for k in data_loader_a.C)

m.addConstrs(y_ijk.sum(i,'*',k)+r_jkfin[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in data_loader_a.F
    for k in data_loader_a.C)
m.addConstrs(y_ijk.sum('*',i,k)+r_iksta[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in data_loader_a.F
    for k in data_loader_a.C)
```

```
#多目标优化
# m.addConstr(z.sum()==206)
# m.addConstr(x_ikdh.sum()==8)

m.update()
# m.optimize()
```

6. 问题二 A 数据集求解模型 q2.py

```
from utils.preprocessing import DataLoader
import gurobipy as gp
from gurobipy import *

data_loader = DataLoader('data/Data A-Crew.csv', 'data/Data
    A-Flight.csv')
data_loader.dump_data()
data_loader_a1 = DataLoader('data/Data A-Crew.csv', 'data/Data
    A-Flight.csv')
data_loader_a1.dump_data(cropped_date=(11,18)) # 分阶段

m=gp.Model('m1')

z=m.addVars(data_loader_a1.F,vtype=gp.GRB.BINARY,name='z')
x_ikdh=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='x_dh
x_ikfo=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='x_fo
x_ikcap=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='x_c

r_iksta=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='r_s
r_jkfin=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='r_f
d_iksta=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='d_s
d_jkfin=m.addVars(data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.BINARY,name='d_f

y_ijk=m.addVars(data_loader_a1.F,data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.B
v_ijk=m.addVars(data_loader_a1.F,data_loader_a1.F,data_loader_a1.C,vtype=gp.GRB.B

# a1=m.addVar(1,vtype=gp.GRB.INTEGER,name='a1')
# a2=m.addVar(2,vtype=gp.GRB.INTEGER,name='a2')

m.ModelSense=GRB.MINIMIZE


# m.setObjectiveN(expr, index=0,pririty=1)
m.setObjective(-z.sum())
#
    m.setObjectiveN(gp.quicksum(gp.quicksum(data_loader_a1.DCost[k]*(gp.quicksum(
    for i in data_loader_a1.FD[t])
#                                           -
    gp.quicksum(d_iksta[i,k]*data_loader_a1.leavetime[i] for i in
    data_loader_a1.FD[t]))
#                                          for k in
    data_loader_a1.C) for t in data_loader_a1.Dates
    ),index=1,priority=9)

# m.setObjectiveN(x_ikdh.sum(),index=2,priority=8)
```

```
# m.setObjectiveN(gp.quicksum(x_ikfo[i,k] for i in
    data_loader_a1.F for k in data_loader_a1.C2), index=3,
    priority=7)

# m.setObjectiveN(a1-a2,index=4,priority=6)

# m.setObjectiveN(-z.sum(),index = 0,weight =0.8)
# m.setObjectiveN(x_ikdh.sum(),index=1,weight=0.2)
# m.setObjectiveN(gp.quicksum(x_ikfo[i,k] for i in
    data_loader_a.F for k in
    data_loader_a.C2),index=2,weight=0.05)


M=10000
#对X的约束
m.addConstrs(x_ikfo[i,k]==0 for i in data_loader_a1.F for k in
    data_loader_a1.C1 )
m.addConstrs(x_ikcap[i,k]==0 for i in data_loader_a1.F for k in
    data_loader_a1.C3 )
m.addConstrs(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]<=1 for i in
    data_loader_a1.F for k in data_loader_a1.C)

#对Z的约束
m.addConstrs(gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]
    for k in data_loader_a1.C)<=M*(z[i]) for i in
    data_loader_a1.F)
m.addConstrs(x_ikcap.sum(i,'*')== z[i] for i in data_loader_a1.F
    )
m.addConstrs(x_ikfo.sum(i,'*')== z[i] for i in data_loader_a1.F )
m.addConstrs(M*z[j]>=gp.quicksum(x_ikcap[j,k]+x_ikfo[j,k]+x_ikdh[j,k]
    for k in data_loader_a1.C) for j in data_loader_a1.F)

#对Y的约束
m.addConstrs(y_ijk[i,j,k]==0 for i,j in data_loader_a1.FF for k
    in data_loader_a1.C)
#m.addConstrs(gp.quicksum(y_ijk[i,j,k] for j in
    F)<=x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in F for k in
    C)
#m.addConstrs(gp.quicksum(y_ijk[j,i,k] for j in
    F)<=x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in F for k in
    C)

#对roaster周期的约束
# m.addConstrs(r_jkfin[j,k]== 0 for j in
    data_loader_a.nonF_arrive_base[0] for k in data_loader_a.C)
m.addConstrs(r_iksta[i,k]== 0 for i in
    data_loader_a1.nonF_leave_base[0] for k in data_loader_a1.C)

m.addConstrs(gp.quicksum(r_iksta[i,k] for i in
    data_loader_a1.F_leave_base[0] )<=1 for k in data_loader_a1.C)
m.addConstrs(gp.quicksum(r_jkfin[j,k] for j in data_loader_a1.F
    )-gp.quicksum(r_iksta[i,k] for i in
    data_loader_a1.F_leave_base[0] ) == 0 for k in
    data_loader_a1.C)
```

```
#第一问中航班对应周期的约束
m.addConstrs(1-(y_ijk.sum(i,'*',k)+r_iksta[i,k]+r_jkfin[i,k])<=M*(1-(x_ikcap[i,k
    for i in data_loader_a1.F for k in data_loader_a1.C)
m.addConstrs(y_ijk.sum(i,'*',k)+r_jkfin[i,k]-(y_ijk.sum('*',i,k)+r_iksta[i,k])
    == 0 for i in data_loader_a1.F for k in data_loader_a1.C)

m.addConstrs(y_ijk.sum(i,'*',k)+r_jkfin[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a1.F for k in data_loader_a1.C)
m.addConstrs(y_ijk.sum('*',i,k)+r_iksta[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a1.F for k in data_loader_a1.C)

#第二问
#对V的约束
m.addConstrs(v_ijk[i,j,k]==0 for i,j in data_loader_a1.FF1 for k
    in data_loader_a1.C)
m.addConstrs(v_ijk[i,j,k] <= y_ijk[i,j,k] for i in
    data_loader_a1.F for j in data_loader_a1.F for k in
    data_loader_a1.C)


#对duty执勤的约束
m.addConstrs(gp.quicksum(d_iksta[i,k] for i in
    data_loader_a1.FD[t]) - gp.quicksum(d_jkfin[i,k] for i in
    data_loader_a1.FD[t]) == 0 for i in data_loader_a1.F for k in
    data_loader_a1.C for t in data_loader_a1.Dates)
m.addConstrs(gp.quicksum(d_iksta[i,k] for i in
    data_loader_a1.FD[t]) <=
    gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a1.FD[t]) for i in data_loader_a1.F for k in
    data_loader_a1.C for t in data_loader_a1.Dates)
m.addConstrs(gp.quicksum(d_iksta[i,k] for i in
    data_loader_a1.FD[t])*M >=
    gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a1.FD[t]) for k in data_loader_a1.C for t in
    data_loader_a1.Dates)

#m.addConstrs(d_iksta[i,k] for i in data_loader_a.FD[t] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a.FD[t] for k in data_loader_a.C for t in
    data_loader_a.Dates)

#第二问中航班对应执勤，执勤对应周期的约束
m.addConstrs((v_ijk.sum(i,'*',k)+ r_jkfin[i,k] == d_jkfin[i,k])
    for i in data_loader_a1.F for k in data_loader_a1.C)
m.addConstrs((v_ijk.sum('*',i,k)+ r_iksta[i,k] == d_iksta[i,k])
    for i in data_loader_a1.F for k in data_loader_a1.C)

#m.addConstrs((d_jkfin[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]) for i in
    data_loader_a.F for k in data_loader_a.C)
#m.addConstrs((d_iksta[i,k] <=
```

```
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]) for i in
    data_loader_a.F for k in data_loader_a.C)

#第二问中其余约束
m.addConstrs(gp.quicksum(d_jkfin[i,k]*data_loader_a1.arrivetime[i]
    for i in data_loader_a1.FD[t]) -
    gp.quicksum(d_iksta[i,k]*data_loader_a1.leavetime[i] for i in
    data_loader_a1.FD[t]) <= 720 for k in data_loader_a1.C for t
    in data_loader_a1.Dates )
m.addConstrs(gp.quicksum((x_ikcap[i,k]+x_ikfo[i,k])*(data_loader_a1.leavetime[i]-
    for i in data_loader_a1.FD[t]) <= 600 for k in
    data_loader_a1.C for t in data_loader_a1.Dates )
#m.addConstrs(gp.quicksum(d_jkfin[i,k]*data_loader_a.arrivetime[i]
    for i in data_loader_a.FD[t]) -
    gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t]) >= 0 for k in data_loader_a.C for t in
    data_loader_a.Dates )


# m.addConstr(z.sum('*')==108)
#
    m.addConstr(gp.quicksum(gp.quicksum(data_loader_a1.DCost[k]*(gp.quicksum(d_jk
    for i in data_loader_a1.FD[t])
#                                                   -
    gp.quicksum(d_iksta[i,k]*data_loader_a1.leavetime[i] for i in
    data_loader_a1.FD[t]))
#                                                  for k in
    data_loader_a1.C) for t in data_loader_a1.Dates
    )==[16900000,17000000])

#
    m.addConstr(gp.quicksum(gp.quicksum(data_loader_a1.DCost[k]*(gp.quicksum(d_jk
    for i in data_loader_a1.FD[t]))
#                 -
    gp.quicksum(d_iksta[i,k]*data_loader_a1.leavetime[i] for i in
    data_loader_a1.FD[t])
#             for k in data_loader_a1.C) for t in
    data_loader_a1.Dates) <= 170000000)
#对执勤时长平衡的约束（辅助目标）
#
    m.addConstrs(a1>=(gp.quicksum(gp.quicksum(d_jkfin[i,k]*data_loader_a1.arrivet
    for i in data_loader_a1.FD[t]) for t in data_loader_a1.Dates)
    -
    gp.quicksum(gp.quicksum(d_iksta[i,k]*data_loader_a1.leavetime[i]
    for i in data_loader_a1.FD[t]) for t in
    data_loader_a1.Dates)) for k in data_loader_a1.C)
#
    m.addConstrs(a2<=(gp.quicksum(gp.quicksum(d_jkfin[i,k]*data_loader_a1.arrivet
    for i in data_loader_a1.FD[t]) for t in data_loader_a1.Dates)
    -
    gp.quicksum(gp.quicksum(d_iksta[i,k]*data_loader_a1.leavetime[i]
    for i in data_loader_a1.FD[t]) for t in
    data_loader_a1.Dates)) for k in data_loader_a1.C)
#m.addConstr(a2 >= 0)
```

```
# m.addContr(z.sum()==108)

m.update()
# m.setParam("MIPGap", )
m.optimize()
```

7. 问题三 A 数据集求解模型 q3.py

```
 from utils.preprocessing import DataLoader
 import gurobipy as gp
 from gurobipy import *
 import math

 data_loader = DataLoader('data/Data A-Crew.csv', 'data/Data
    A-Flight.csv')

 data_loader.dump_data()
 data_loader_a = DataLoader('data/Data A-Crew.csv', 'data/Data
    A-Flight.csv')
 data_loader_a.dump_data(cropped_date=(11, 15))

 MaxTAFB = 14400 / 15 * 5

m=gp.Model('m1')

z=m.addVars(data_loader_a.F,vtype=gp.GRB.BINARY,name='z')
x_ikdh=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='x_dh'
x_ikfo=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='x_fo'
x_ikcap=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='x_ca

r_iksta=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='r_ik
r_jkfin=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='r_jk
d_iksta=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='d_ik
d_jkfin=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='d_jk
p_iksta=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='p_ik
p_jkfin=m.addVars(data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BINARY,name='p_jk

y_ijk=m.addVars(data_loader_a.F,data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BIN
v_ijk=m.addVars(data_loader_a.F,data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BIN
w_ijk=m.addVars(data_loader_a.F,data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BIN
u_ijk=m.addVars(data_loader_a.F,data_loader_a.F,data_loader_a.C,vtype=gp.GRB.BIN

# a1=m.addVar(1,vtype=gp.GRB.INTEGER,name='a1')
# a2=m.addVar(2,vtype=gp.GRB.INTEGER,name='a2')

# a3=m.addVar(3,vtype=gp.GRB.INTEGER,name='a3')
# a4=m.addVar(4,vtype=gp.GRB.INTEGER,name='a4')


m.ModelSense=GRB.MINIMIZE

m.setObjective(-z.sum())
```

```
#
    m.setObjective(gp.quicksum(gp.quicksum(data_loader_a.DCost[k]*(gp.quicksum(d_
    for i in data_loader_a.FD[t])
#                                                   -
    gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t]))/60
#                                                 for k in
    data_loader_a.C) for t in data_loader_a.Dates ))
#
    m.setObjective(gp.quicksum(gp.quicksum(data_loader_a.PCost[k]*(u_ijk[i,j,k]*d
    - u_ijk[i,j,k]*data_loader_a.leavetime[i]) for i in
    data_loader_a.FD[t] for j in data_loader_a.FD[t])/60
#                                                 for k in
    data_loader_a.C for t in data_loader_a.Dates ))
# m.setObjectiveN(x_ikdh.sum(), index=2, priority=8)


# m.setObjectiveN(a1-a2, index=3, priority=7)
# m.setObjectiveN(a3-a4, index=4, priority=6)
# m.setObjectiveN(gp.quicksum(x_ikfo[i,k] for i in
    data_loader_a.F for k in data_loader_a.C2), index=5,
    priority=5)

# m.setObjectiveN(-z.sum(),index = 0,weight =0.8)
# m.setObjectiveN(x_ikdh.sum(),index=1,weight=0.2)
# m.setObjectiveN(gp.quicksum(x_ikfo[i,k] for i in
    data_loader_a.F for k in
    data_loader_a.C2),index=2,weight=0.05)


M=10000
#对X的约束
m.addConstrs(x_ikfo[i,k]==0 for i in data_loader_a.F for k in
    data_loader_a.C1 )
m.addConstrs(x_ikcap[i,k]==0 for i in data_loader_a.F for k in
    data_loader_a.C3 )
m.addConstrs(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]<=1 for i in
    data_loader_a.F for k in data_loader_a.C)

#对Z的约束
m.addConstrs(gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]
    for k in data_loader_a.C)<=M*(z[i]) for i in data_loader_a.F)
m.addConstrs(x_ikcap.sum(i,'*')== z[i] for i in data_loader_a.F )
m.addConstrs(x_ikfo.sum(i,'*')== z[i] for i in data_loader_a.F )
m.addConstrs(M*z[j]>=gp.quicksum(x_ikcap[j,k]+x_ikfo[j,k]+x_ikdh[j,k]
    for k in data_loader_a.C) for j in data_loader_a.F)

#对Y的约束
m.addConstrs(y_ijk[i,j,k]==0 for i,j in data_loader_a.FF for k
    in data_loader_a.C)
#m.addConstrs(gp.quicksum(y_ijk[i,j,k] for j in
    F)<=x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in F for k in
    C)
#m.addConstrs(gp.quicksum(y_ijk[j,i,k] for j in
```

```
    F)<=x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in F for k in
    C)

#对roaster周期的约束
m.addConstrs(r_jkfin[j,k]== 0 for j in
    data_loader_a.nonF_arrive_base[0] for k in data_loader_a.C)
m.addConstrs(r_iksta[i,k]== 0 for i in
    data_loader_a.nonF_leave_base[0] for k in data_loader_a.C)

m.addConstrs(gp.quicksum(r_iksta[i,k] for i in
    data_loader_a.F_leave_base[0] )<=1 for k in data_loader_a.C)
m.addConstrs(gp.quicksum(r_jkfin[j,k] for j in
    data_loader_a.F_arrive_base[0] )-gp.quicksum(r_iksta[i,k] for
    i in data_loader_a.F_leave_base[0] ) == 0 for k in
    data_loader_a.C)

#第一问中航班对应周期的约束
m.addConstrs(1-(y_ijk.sum(i,'*',k)+r_iksta[i,k]+r_jkfin[i,k])<=M*(1-(x_ikcap[i,k
    for i in data_loader_a.F for k in data_loader_a.C)
m.addConstrs(y_ijk.sum(i,'*',k)+r_jkfin[i,k]-(y_ijk.sum('*',i,k)+r_iksta[i,k])
    == 0 for i in data_loader_a.F for k in data_loader_a.C)

m.addConstrs(y_ijk.sum(i,'*',k)+r_jkfin[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in data_loader_a.F
    for k in data_loader_a.C)
m.addConstrs(y_ijk.sum('*',i,k)+r_iksta[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in data_loader_a.F
    for k in data_loader_a.C)

#第二问
#对V的约束
m.addConstrs(v_ijk[i,j,k]==0 for i,j in data_loader_a.FF1 for k
    in data_loader_a.C)
m.addConstrs(v_ijk[i,j,k] <= y_ijk[i,j,k] for i in
    data_loader_a.F for j in data_loader_a.F for k in
    data_loader_a.C)


#对duty执勤的约束
m.addConstrs(gp.quicksum(d_iksta[i,k] for i in
    data_loader_a.FD[t]) - gp.quicksum(d_jkfin[i,k] for i in
    data_loader_a.FD[t]) == 0 for i in data_loader_a.F for k in
    data_loader_a.C for t in data_loader_a.Dates)
m.addConstrs(gp.quicksum(d_iksta[i,k] for i in
    data_loader_a.FD[t]) <=
    gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a.FD[t]) for i in data_loader_a.F for k in
    data_loader_a.C for t in data_loader_a.Dates)
m.addConstrs(gp.quicksum(d_iksta[i,k] for i in
    data_loader_a.FD[t])*M >=
    gp.quicksum(x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a.FD[t]) for k in data_loader_a.C for t in
    data_loader_a.Dates)
```

```
#m.addConstrs(d_iksta[i,k] for i in data_loader_a.FD[t] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k] for i in
    data_loader_a.FD[t] for k in data_loader_a.C for t in
    data_loader_a.Dates)

#第二问中航班对应执勤，执勤对应周期的约束
m.addConstrs((v_ijk.sum(i,'*',k)+ r_jkfin[i,k] == d_jkfin[i,k])
    for i in data_loader_a.F for k in data_loader_a.C)
m.addConstrs((v_ijk.sum('*',i,k)+ r_iksta[i,k] == d_iksta[i,k])
    for i in data_loader_a.F for k in data_loader_a.C)

#m.addConstrs((d_jkfin[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]) for i in
    data_loader_a.F for k in data_loader_a.C)
#m.addConstrs((d_iksta[i,k] <=
    x_ikcap[i,k]+x_ikfo[i,k]+x_ikdh[i,k]) for i in
    data_loader_a.F for k in data_loader_a.C)

#第二问中其余约束
m.addConstrs(gp.quicksum(d_jkfin[i,k]*data_loader_a.arrivetime[i]
    for i in data_loader_a.FD[t]) -
    gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t]) <= 720 for k in data_loader_a.C for t in
    data_loader_a.Dates )
m.addConstrs(gp.quicksum((x_ikcap[i,k]+x_ikfo[i,k])*(data_loader_a.leavetime[i]-d
    for i in data_loader_a.FD[t]) <= 600 for k in data_loader_a.C
    for t in data_loader_a.Dates )
#m.addConstrs(gp.quicksum(d_jkfin[i,k]*data_loader_a.arrivetime[i]
    for i in data_loader_a.FD[t]) -
    gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t]) >= 0 for k in data_loader_a.C for t in
    data_loader_a.Dates )


#m.addConstr(z.sum('*')==206)
#
    m.addConstr(gp.quicksum(gp.quicksum(data_loader_a.DCost[k]*(gp.quicksum(d_jkf
    for i in data_loader_a.FD[t])
#                 -
    gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t]))
#             for k in data_loader_a.C) for t in
    data_loader_a.Dates )==[1324573-100000,1324573+100000])
#对执勤时长平衡的约束（辅助目标）
#
    m.addConstrs(a1>=(gp.quicksum(gp.quicksum(d_jkfin[i,k]*data_loader_a.arriveti
    for i in data_loader_a.FD[t]) for t in data_loader_a.Dates) -
    gp.quicksum(gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i]
    for i in data_loader_a.FD[t]) for t in data_loader_a.Dates))
    for k in data_loader_a.C)
#
    m.addConstrs(a2<=(gp.quicksum(gp.quicksum(d_jkfin[i,k]*data_loader_a.arriveti
    for i in data_loader_a.FD[t]) for t in data_loader_a.Dates) -
    gp.quicksum(gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i]
```

```
    for i in data_loader_a.FD[t]) for t in data_loader_a.Dates))
    for k in data_loader_a.C)

#第三问
#对W的约束
m.addConstrs(w_ijk[i,j,k]==0 for i,j in data_loader_a.FF2 for k
    in data_loader_a.C)
m.addConstrs(w_ijk[i,j,k] <= v_ijk[i,j,k] for i in
    data_loader_a.F for j in data_loader_a.F for k in
    data_loader_a.C)



#对U的约束
m.addConstrs(r_jkfin[i,k] <=gp.quicksum(u_ijk[j,i,k] for j in
    data_loader_a.F) for i in data_loader_a.F for k in
    data_loader_a.C)
m.addConstrs(r_iksta[i,k] <=gp.quicksum(u_ijk[i,j,k] for j in
    data_loader_a.F) for i in data_loader_a.F for k in
    data_loader_a.C)

m.addConstrs(gp.quicksum(u_ijk[j,i,k] for j in data_loader_a.F)
    <= d_jkfin[i,k] for i in data_loader_a.F for k in
    data_loader_a.C)
m.addConstrs(gp.quicksum(u_ijk[i,j,k] for j in data_loader_a.F)
    <= d_iksta[i,k] for i in data_loader_a.F for k in
    data_loader_a.C)

m.addConstrs(gp.quicksum(u_ijk[i,j,k] for j in
    data_loader_a.F)-(r_iksta[i,k] + gp.quicksum(w_ijk[j,i,k] for
    j in data_loader_a.F)) == 0 for i in data_loader_a.F for k in
    data_loader_a.C)
m.addConstrs(gp.quicksum(u_ijk[j,i,k] for j in
    data_loader_a.F)-(r_jkfin[i,k] + gp.quicksum(w_ijk[i,j,k] for
    j in data_loader_a.F))== 0 for i in data_loader_a.F for k in
    data_loader_a.C)


m.addConstrs(u_ijk[i,j,k]*(data_loader_a.leavedate[j] -
    data_loader_a.leavedate[i] )<= 4 for i in data_loader_a.F for
    j in data_loader_a.F for k in data_loader_a.C)
#m.addConstrs(w_ijk[i,j,k]*(data_loader_a.leavedate[j] -
    data_loader_a.leavedate[i] )>= 2 for i in data_loader_a.F for
    j in data_loader_a.F for k in data_loader_a.C)
m.addConstrs(gp.quicksum(u_ijk[i,j,k]*(data_loader_a.arrivetime[j]
    - data_loader_a.leavetime[i])for i in data_loader_a.F for j
    in data_loader_a.F )<= MaxTAFB for k in data_loader_a.C)
m.addConstrs(u_ijk[i,j,k]==0 for i,j in data_loader_a.FF for k
    in data_loader_a.C)

#对执勤时长平衡的约束（辅助目标）
#
    m.addConstrs(a3>=(gp.quicksum(gp.quicksum(v_ijk[i,j,k]*data_loader_a.arriveti
    - v_ijk[i,j,k]*data_loader_a.leavetime[i] for i in
```

```
    data_loader_a.FD[t] for j in data_loader_a.FD[t]) for t in
    data_loader_a.Dates )) for k in data_loader_a.C)
#
    m.addConstrs(a4<=(gp.quicksum(gp.quicksum(v_ijk[i,j,k]*data_loader_a.arriveti
    - v_ijk[i,j,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t] for j in data_loader_a.FD[t]) for t in
    data_loader_a.Dates )) for k in data_loader_a.C)

# m.addConstr(z.sum()==64)
#
    m.addConstr(gp.quicksum(gp.quicksum(data_loader_a.DCost[k]*(gp.quicksum(d_jkf
    for i in data_loader_a.FD[t])
#                                                 -
    gp.quicksum(d_iksta[i,k]*data_loader_a.leavetime[i] for i in
    data_loader_a.FD[t]))/60
#                                                 for k in
    data_loader_a.C) for t in data_loader_a.Dates) == 175316)

m.update()
m.optimize()
```

8. 问题二 B 数据集优化算法求解代码

```
#include<iostream>
#include<fstream>
#include<vector>
#include<string>
#include<math.h>
#include<stdio.h>
#include<algorithm>
#include<map>
#include<unordered_map>
#include<utility>
#include <time.h>
#include<queue>

using namespace std;

struct people
{
   string id;
   int type; //1是副机长，2是主机长，3是均可
   string base;
   int dcost;
   int pcost;
   vector<int>hx;//存储航线编号，通过sj[编号]找到对应航班
   char cur_pos; //C是正，F是副
   int tot_time;//执勤时长
   int duty_num;//执勤次数
   int fly_time; //飞行时长
};

struct flight
{
```

```
    int index;
    string ID;
    string des;
    string beg;
    int arr_date;
    int arr_time;
    int beg_date;
    int beg_time;
    int beg_tot;
    int arr_tot;
};
struct duty
{
    vector<flight>flis;//一个执勤所包含的所有航段
    vector<int>flies_index;
    int beg_time;
    int end_time;  //均是总时间
    string beg;
    string des;
    int duty_time_tot = 0;
    int duty_time_fly = 0; //==end_time-beg.time

};
struct path
{
    vector<int>lx;
    int hb_num;
};
struct node
{
    int num_fli;
    int num_dh;
    vector<int>vf;
    vector<int>vd;
};
bool cmp(flight a, flight b)
{
    return a.beg_tot < b.beg_tot;
    //return a.arr_tot < b.arr_tot;
}
struct renwu
{
    vector<duty>dutys;
    int flis;
    int beg_time;
    int arr_time;
    int tot_duty_time;
};
bool cmp2(renwu a, renwu b)
{
    return a.tot_duty_time > b.tot_duty_time;
}

struct partner
```

```
{
  people a, b;
  int work_time = 0;
  int tot_money;
  vector<renwu>rws;
  int left = 0;
  int end = 0;

  //工作时长不相等，时长短的在前面，便于均衡，工作时间相等，钱少的在前面，成本低
  bool operator < (partner x) const
  {
    if (work_time == x.work_time) return tot_money >
       x.tot_money; //想让小的在前面，用>
    return work_time > x.work_time;
       //想让大的在前面，用<
  }

};
vector<people>Emp2;
vector<flight>Fli2;
vector<path>paths;
vector<path>paths_base1;
vector<path>paths_base2;
vector<flight>sj; //sort过的Fli2
vector<bool>vis;
int min_date = 35;
int min_CT = 40;
int max_fly = 600;
int max_duty = 720;
int min_rest = 660;
string base1 = "HOM";
string base2 = "TGD";
vector<node>cjnum;
vector<flight>failed;
vector <people> T1_base1, T1_base2, T2_base1, T2_base2,
   T3_base1, T3_base2;
vector<path>valid_path_base1;
vector<renwu>renwus;
vector<partner>jz1; //放base1的员工
vector<partner>jz2; //放base2的员工

vector<renwu>rw1; //放基地1的任务
vector<renwu>rw2; //放基地2的任务

vector<partner>p_base1;
vector<partner>p_base2;

void B_sldata_C() //读入data_C的数据
{
  ifstream infile("data//Data_BC.csv", ios::in);
  if (!infile)
  {
    cout << "open error" << endl;
    exit(1);
```

```
  }
  string s, ss;
  while (!infile.eof())
  {
    people p;
    infile >> p.id;
    infile >> s >> ss;
    int num = 0;
    if (s == "Y") num += 2;
    if (ss == "Y") num += 1;
    p.type = num;
    infile >> s;//跳过
    infile >> p.base;
    infile >> s;
    p.dcost = stoi(s);
    infile >> s;
    p.pcost = stoi(s);
    if (p.id == "") continue;
    Emp2.push_back(p);
  }
  /*cout<<Emp2.size()<<endl;
  for(people x:Emp2)
  {
    cout<<x.id<<" "<<x.type<<" "<<x.base<<" "<<x.dcost<<"
       "<<x.pcost<<endl;
  }*/

}
int cal_date(int x)
{
  return x / 1440;
}
int str_to_date(string s)
{
  for (int i = 0; i < s.size(); i++)
  {
    if (s[i] == '/')
    {
      int j = i + 1;
      int data = 0;
      while (s[j] != '/')
      {
        data = data * 10 + (s[j] - '0');
        j++;
      }
      return data;
    }
  }
}
int str_to_time(string s)
{
  int res = 0;
  int num = 0;
  for (int i = 0; i < s.size(); i++)
```

```
    {
        if (s[i] == ':')
        {
            res += num * 60;
            num = 0;
        }
        else num = num * 10 + (s[i] - '0');
    }
    res += num;
    return res;
}

void B_sldata_F() //读入data_F的数据
{
    ifstream infile("data//Data_BF.csv", ios::in);
    if (!infile)
    {
        cout << "open error" << endl;
        exit(1);
    }
    string s;
    int num = 1;
    while (!infile.eof())
    {
        flight f;
        f.index = num++;
        infile >> f.ID;
        infile >> s;
        f.beg_date = str_to_date(s);
        infile >> s;
        f.beg_time = str_to_time(s);
        infile >> f.beg;
        infile >> s;
        f.arr_date = str_to_date(s);
        infile >> s;
        f.arr_time = str_to_time(s);
        infile >> f.des;
        infile >> s;
        if (f.ID == "") continue;
        Fli2.push_back(f);
    }
    cout << Fli2.size() << endl;
    /*for (flight x : Fli2)
    {
        cout << x.index << " " << x.ID << " " << x.beg_date << " "
            << x.beg_time << " " << x.beg << " " << x.arr_date << " "
            << x.arr_time << " " << x.des << endl;
    }*/
}

void cal_totB()
{
    int len = Fli2.size();
    for (int i = 0; i < len; i++)
```

```
  {
    flight x = Fli2[i];
    Fli2[i].arr_tot = (x.arr_date) * 24 * 60 + x.arr_time;
    Fli2[i].beg_tot = (x.beg_date) * 24 * 60 + x.beg_time;
  }
  /*unordered_map<int, bool>hh;
  int num = 0;
  for (flight x : Fli2)
  {
    int k1 = cal_date(x.beg_tot);
    int k2 = cal_date(x.arr_tot);

    if (!hh[k1])
    {
      cout << k1 << endl;
      hh[k1] = 1;
      num++;
    }
    if (!hh[k2])
    {
      cout << k2 << endl;
      hh[k2] = 1;
      num++;
    }
  }
  cout << num << endl;*/

}
bool isok(int i, int j)
{
  if (sj[i].des == sj[j].beg && sj[j].beg_tot - sj[i].arr_tot >=
     min_CT && !vis[j])
    return 1;
  else return 0;
}
bool iserror(int i, int j)
{
  if (sj[i].des == sj[j].beg && sj[j].beg_tot - sj[i].arr_tot >=
     min_CT)
    return 0;
  else return 1;
}
bool isdutyok(duty e, int j) //判断sj[j]放入duty中是否符合要求
{
  if (sj[j].arr_tot - e.beg_time > max_duty) return 0;
     //超过总的执勤时长
  if ((sj[j].arr_tot - sj[j].beg_tot) + e.duty_time_fly >
     max_fly) return 0;//超过总的飞行时长
  if (cal_date(e.beg_time) != cal_date(sj[j].arr_tot)) return 0;
     //加入的sj[j]和duty不在同一天
  return 1;
}

void crew_distrbution()
```

```
{
  vector <people> T1, T2, T3;
  for (auto x : Emp2)
    if (x.type == 1) T1.push_back(x);
    else if (x.type == 2) T2.push_back(x);
    else if (x.type == 3) T3.push_back(x);
  cout << "每个type的人员分布情况" << endl;
  cout << T1.size() << " " << T2.size() << " " << T3.size() <<
     endl;

  for (auto x : Emp2)
  {
    if (x.type == 1) if (x.dcost != 600) cout << x.id << endl;
    else if (x.type == 2) if (x.dcost != 680) cout << x.id <<
       endl;
    else if (x.type == 3) if (x.dcost != 640) cout << x.id <<
       endl;
  }

  for (auto x : Emp2)
  {
    if (x.type == 1 && x.base == base1) T1_base1.push_back(x);
    else if (x.type == 1 && x.base == base2)
       T1_base2.push_back(x);
    else if (x.type == 2 && x.base == base1)
       T2_base1.push_back(x);
    else if (x.type == 2 && x.base == base2)
       T2_base2.push_back(x);
    else if (x.type == 3 && x.base == base1)
       T3_base1.push_back(x);
    else if (x.type == 3 && x.base == base2)
       T3_base2.push_back(x);
  }
  cout << "每个type的人员分布和基地情况" << endl;
  cout << "基地1:" << endl;
  cout << T1_base1.size() << " " << T2_base1.size() << " " <<
     T3_base1.size() << endl;
  cout << "基地2:" << endl;
  cout << T1_base2.size() << " " << T2_base2.size() << " " <<
     T3_base2.size() << endl;


  //分配人员base1
  int i1 = 0, i2 = 0, i3 = 0;
  int num = 24;
  while (num--)
  {
    partner pp;
    people a = T2_base1[i2++]; //正
    a.cur_pos = 'C';
    people b = T1_base1[i1++]; //副
    b.cur_pos = 'F';
    pp.a = a, pp.b = b;
    pp.tot_money = (a.dcost + b.dcost);
```

```
    pp.work_time = 0;
    jz1.push_back(pp);
}
num = 4;
while (num--)
{
    partner pp;
    people a = T2_base1[i2++];
    a.cur_pos = 'C';
    people b = T3_base1[i3++];
    b.cur_pos = 'F';
    pp.a = a, pp.b = b;
    pp.tot_money = (a.dcost + b.dcost);
    pp.work_time = 0;
    jz1.push_back(pp);
}
num = 8;
while (num--)
{
    partner pp;
    people a = T3_base1[i3++];
    a.cur_pos = 'C';
    people b = T3_base1[i3++];
    b.cur_pos = 'F';
    pp.a = a, pp.b = b;
    pp.tot_money = (a.dcost + b.dcost);
    pp.work_time = 0;
    jz1.push_back(pp);
}
cout << "base1的机长对数:";
cout << jz1.size() << endl;

//分配人员base2
i1 = 0, i2 = 0, i3 = 0;
num = 59;
while (num--)
{
    partner pp;
    people a = T2_base2[i2++];
    a.cur_pos = 'C';
    people b = T1_base2[i1++];
    b.cur_pos = 'F';
    pp.a = a, pp.b = b;
    pp.tot_money = (a.dcost + b.dcost);
    pp.work_time = 0;
    jz2.push_back(pp);
}
num = 104;
while (num--)
{
    partner pp;
    people a = T3_base2[i3++];
    a.cur_pos = 'C';
    people b = T1_base2[i1++];
```

```
    b.cur_pos = 'F';
    pp.a = a, pp.b = b;
    pp.tot_money = (a.dcost + b.dcost);
    pp.work_time = 0;
    jz2.push_back(pp);
  }
  cout << "base2的机长对数:";
  cout << jz2.size() << endl;
  //检验是否有重复
  unordered_map<string, bool>hash;
  for (auto x : jz2)
  {
    string a = x.a.id;
    string b = x.b.id;
    if (hash[a] || hash[b]) cout << "error" << endl;
    hash[a] = 1;
    hash[b] = 1;
  }

}
void Find_min_cjnum()
{
  //统计所有的未乘机人员：放入left中
  int sum = 0;
  unordered_map<int, bool>hash;
  for (int i = 0; i < paths.size(); i++)
  {
    vector<int>lx = paths[i].lx;
    for (int j = 0; j < lx.size(); j++)
    {
      hash[lx[j]] = 1;
    }
  }
  vector <flight>left;
  for (int i = 0; i < sj.size(); i++)
  {
    if (hash[i] == 0) left.push_back(sj[i]);
  }
  cout << left.size() << endl;

  //处理left中的点使得连通的边变成一个对象
  sort(left.begin(), left.end(), cmp);
  vector<flight>new_left;
  vector<bool>vis(left.size(), 0);
  vector<vector<int>>com_path; //通过left中的顺序访问
  for (int i = 0; i < left.size(); i++)
  {
    vector<int>v;
    flight x = left[i];
    if (vis[i]) continue;
    vis[i] = 1;
    v.push_back(i);
    for (int j = i + 1; j < left.size(); j++)
    {
```

```
      flight y = left[j];
      if (x.des == y.beg && y.beg_tot - x.arr_tot >= min_CT &&
         !vis[j]) //合并y给x
      {
        x.des = y.des;
        x.arr_tot = y.arr_tot;
        vis[j] = 1;
        v.push_back(j);
      }
    }
    new_left.push_back(x);
    com_path.push_back(v);
}
cout << " new_left.size()" << new_left.size() << endl;

//测试存储的内容
/*cout << com_path.size() << endl;
for (int i = 0; i < com_path.size(); i++)
{
  vector<int>v = com_path[i];
  cout << endl << endl;
  cout << new_left[i].beg << " " << new_left[i].des << " " <<
     new_left[i].beg_tot << " " << new_left[i].arr_tot << endl;

  if (v.size() > 1)
  {
    cout << "-----------------" << endl;
    for (auto x : v)
    {
      flight ff = left[x];
      cout << ff.beg << " " << ff.des << " " << ff.beg_tot <<
         " " << ff.arr_tot << endl;
    }
  }
}*/

//begin
vector<vector<int>>cj_path =
   vector<vector<int>>(new_left.size());
for (int i = 0; i < new_left.size(); i++) //对每个left单独考虑
{
  flight fx = new_left[i];
  int res1 = 1000000;
  vector<int>v1; //base1前
  int res2 = 1000000;
  vector <int> v2; //base1后
  int res3 = 100000;
  vector<int>v3; //base2前
  int res4 = 100000;
  vector<int>v4; //base2后
  for (int j = 0; j < paths.size(); j++)
  {
    vector<int>lx = paths[j].lx;
```

```
        //查找前半段路的最小路径，长度存储在res1中，具体路径存储在v1中
        int pp = -1; //可以到达left出发点的节点
        for (int k = 0; k < lx.size(); k++)
        {
          flight x = sj[lx[k]];
          if (x.des == fx.beg && fx.beg_tot - x.arr_tot >= min_CT)
            pp = max(pp, k);
        }
        if (pp != -1)
        {
          for (int k = pp; k >= 0; k--)
          {
            flight cur_f = sj[lx[k]];
            if (cur_f.beg == base1)
            {
              int num = k - pp + 1;
              if (num < res1)
              {
                res1 = num;
                v1.clear();
                for (int u = k; u <= pp; u++)
                {
                  v1.push_back(lx[u]);
                }
              }
              break;
            }
          }
          for (int k = pp; k >= 0; k--)
          {
            flight cur_f = sj[lx[k]];
            if (cur_f.beg == base2)
            {
              int num = k - pp + 1;
              if (num < res3)
              {
                res3 = num;
                v3.clear();
                for (int u = k; u <= pp; u++)
                {
                  v3.push_back(lx[u]);
                }
              }
              break;
            }
          }
        }

        //查找后半段路的最小路径，长度存储在res2中，具体路径存储在v2中
        int qq = -1; //可以到达left出发点的节点
        for (int k = 0; k < lx.size(); k++)
        {
          flight x = sj[lx[k]];
          if (x.beg == fx.des && x.beg_tot - fx.arr_tot >= min_CT)
```

```
            qq = min(qq, k);
        }
        if (qq != -1)
        {
            for (int k = qq; k < lx.size(); k++)
            {
                flight cur_f = sj[lx[k]];
                if (cur_f.des == base1)
                {
                    int num = k - qq + 1;
                    if (num < res2)
                    {
                        res2 = num;
                        v2.clear();
                        for (int u = qq; u <= k; u++)
                        {
                            v2.push_back(lx[u]);
                        }
                    }
                    break;
                }
            }

            for (int k = qq; k < lx.size(); k++)
            {
                flight cur_f = sj[lx[k]];
                if (cur_f.des == base2)
                {
                    int num = k - qq + 1;
                    if (num < res2)
                    {
                        res4 = num;
                        v4.clear();
                        for (int u = qq; u <= k; u++)
                        {
                            v4.push_back(lx[u]);
                        }
                    }
                    break;
                }
            }
        }

    }
    cout << "第" << i << "组:" << endl;
    bool isvalid_1 = 1;
    bool isvalid_2 = 1;
    if (v1.size() == 0 || (v2.size() == 0 && fx.des != base1))
        isvalid_1 = 0;
    if (v3.size() == 0 || (v4.size() == 0 && fx.des != base2))
        isvalid_2 = 0;
    int print12 = 0;
    if (isvalid_1 && isvalid_2)
    {
```

```
      if (v1.size() + v2.size() < v3.size() + v4.size()) print12
         = 1;
      else print12 = 2;
    }
    else if (isvalid_1 && !isvalid_2) print12 = 1;
    else if (!isvalid_1 && isvalid_2) print12 = 2;

    if (print12 == 1)
    {
      node ee;
      ee.num_fli = com_path[i].size();
      ee.num_dh = (v1.size() + v2.size()) * 2;
      ee.vf = com_path[i]; //注意这个访问要用left
      vector<int>v;
      for (auto x : v1) v.push_back(x);
      for (int i = 0; i < sj.size(); i++) if (sj[i].index ==
         fx.index) v.push_back(i);
      for (auto x : v2) v.push_back(x);
      ee.vd = v; //要用sj访问
      cjnum.push_back(ee);

    }
    else if (print12 == 2)
    {
      node ee;
      ee.num_fli = com_path[i].size();
      ee.num_dh = (v3.size() + v4.size()) * 2;
      ee.vf = com_path[i]; //注意这个访问要用left
      vector<int>v;
      for (auto x : v3) v.push_back(x);
      for (int i = 0; i < sj.size(); i++) if (sj[i].index ==
         fx.index) v.push_back(i);
      for (auto x : v4) v.push_back(x);
      ee.vd = v; //要用sj访问
      cjnum.push_back(ee);
    }
    else
    {
      continue;
    }


  }
  cout << cjnum.size() << endl;
  for (int i = 0; i < cjnum.size(); i++)
  {
    node x = cjnum[i];
    cout << x.num_fli << " " << x.num_dh << endl;
  }

}
void path_base()
{
  for (int i = 0; i < paths.size(); i++)
```

```
  {
    path x = paths[i];
    vector<int>v = x.lx;
    if (sj[v[0]].beg == base1)
      paths_base1.push_back(x);
    else paths_base2.push_back(x);
  }
  cout << paths_base1.size() << endl;
  cout << paths_base2.size() << endl;
}

void print_table()
{
  ofstream outfile("E://math_data//m2B_b___2.txt", ios::out);
  outfile << "员工" << " " << "航班" << " " << "日期" << " " <<
     "类型" << endl;

  int tot_cost = 0;

  unordered_map<int, bool>ok_fli;
  vector<people>P_with_worktime;

  //先处理基地1的
  cout << "处理base1" << endl;
  cout << p_base1.size() << endl;
  for (auto x : p_base1)
  {
    partner pp = x;
    tot_cost += (pp.work_time*pp.tot_money);

    people a = pp.a;
    people b = pp.b;
    int fly_time = 0;
    int duty_num = 0;

    for (int i = 0; i < x.rws.size(); i++)
    {
      vector<duty> dutys = x.rws[i].dutys;
      duty_num += dutys.size();
      for (int k = 0; k < dutys.size(); k++)
      {
        duty dd = dutys[k];
        fly_time += dd.duty_time_fly;
        for (int u = 0; u < dd.flis.size(); u++)
        {
          flight f = dd.flis[u];
          //"员工" << " " << "航班" << " " << "日期" << " " <<
             "类型" << endl;
          outfile << a.id << " " << f.ID << " " <<
             cal_date(f.beg_tot) << " " << a.cur_pos << endl;
          outfile << b.id << " " << f.ID << " " <<
             cal_date(f.beg_tot) << " " << b.cur_pos << endl;

          ok_fli[f.index] = 1;
```

```
        }
      }
    }
    a.tot_time = pp.work_time;
    b.tot_time = pp.work_time;

    a.fly_time = fly_time;
    b.fly_time = fly_time;

    a.duty_num = duty_num;
    b.duty_num = duty_num;

    P_with_worktime.push_back(a);
    P_with_worktime.push_back(b);
}


//先处理基地2的
cout << "处理base2" << endl;
cout << p_base2.size() << endl;
for (auto x : p_base2)
{
    partner pp = x;
    tot_cost += (pp.work_time*pp.tot_money);

    people a = pp.a;
    people b = pp.b;

    int fly_time = 0;
    int duty_num = 0;

    for (int i = 0; i < x.rws.size(); i++)
    {
      vector<duty> dutys = x.rws[i].dutys;
      duty_num += dutys.size();
      for (int k = 0; k < dutys.size(); k++)
      {
        duty dd = dutys[k];
        fly_time += dd.duty_time_fly;

        for (int u = 0; u < dd.flis.size(); u++)
        {
          flight f = dd.flis[u];
          //"员工" << " " << "航班" << " " << "日期" << " " <<
              "类型" << endl;
          outfile << a.id << " " << f.ID << " " <<
              cal_date(f.beg_tot) << " " << a.cur_pos << endl;
          outfile << b.id << " " << f.ID << " " <<
              cal_date(f.beg_tot) << " " << b.cur_pos << endl;

          ok_fli[f.index] = 1;

        }
```

```
      }
    }
    a.tot_time = pp.work_time;
    b.tot_time = pp.work_time;

    a.fly_time = fly_time;
    b.fly_time = fly_time;

    a.duty_num = duty_num;
    b.duty_num = duty_num;

    P_with_worktime.push_back(a);
    P_with_worktime.push_back(b);
  }

  cout << "总成本:  " << tot_cost << endl;

  int fin_flis = 0; //成功分配的次数
  vector<flight>undistr; //未分配的集合
  for (int i = 0; i < Fli2.size(); i++)
  {
    int index = Fli2[i].index;
    if (ok_fli[index] == 1) fin_flis++;
    else undistr.push_back(Fli2[i]);
  }

  cout << "成功分配的航班数:  " << fin_flis << endl;
  cout << "未分配的航班数:  " << undistr.size() << endl;


  ofstream outfile2("E://math_data//m2B_a___2.txt", ios::out);
  outfile2 << "航班号" << " " << "日期" << endl;
  for (auto x : undistr)
  {
    outfile2 << x.ID << " " << cal_date(x.beg_tot) << endl;
  }

  ofstream outfile3("E://math_data//m2B_c___2.txt", ios::out);
  outfile3 << "员工号" << " " << "执勤时长" << " " << " 飞行时长" <<
     " " << "执勤天数" << endl;
  cout << "实际工作人数: " << P_with_worktime.size() << endl;
  cout << "机组总体利用率: " << double(P_with_worktime.size()) /
     Emp2.size() << endl;
  for (auto p : P_with_worktime)
  {
    outfile3 << p.id << " " << p.tot_time << " " << p.fly_time
       << " " << p.duty_num << endl;
  }

  int maxn = 10000;
  int max_flytime = 0, min_flytime = maxn, ave_flytime = 0;
  int max_dutytime = 0, min_dutytime = maxn, ave_dutytime = 0;
  int max_dutynum = 0, min_dutynum = maxn, ave_dutynum = 0;
```

```
for (auto p : P_with_worktime)
{
  //处理平均的
  ave_flytime += p.fly_time;
  ave_dutytime += p.tot_time;
  ave_dutynum += p.duty_num;
  //处理max
  max_flytime = max(max_flytime, p.fly_time);
  max_dutytime = max(max_dutytime, p.tot_time);
  max_dutynum = max(max_dutynum, p.duty_num);
  //处理min
  min_flytime = min(min_flytime, p.fly_time);
  min_dutytime = min(min_dutytime, p.tot_time);
  min_dutynum = min(min_dutynum, p.duty_num);
}
ave_flytime /= (P_with_worktime.size());
ave_dutytime /= (P_with_worktime.size());
ave_dutynum /= (P_with_worktime.size());

cout << "max_flytime: " << max_flytime << endl;
cout << "max_dutytime: " << max_dutytime << endl;
cout << "max_dutynum: " << max_dutynum << endl;

cout << "min_flytime: " << min_flytime << endl;
cout << "min_dutytime: " << min_dutytime << endl;
cout << "min_dutynum: " << min_dutynum << endl;

cout << "ave_flytime: " << ave_flytime << endl;
cout << "ave_dutytime: " << ave_dutytime << endl;
cout << "ave_dutynum: " << ave_dutynum << endl;


//输出一次时长:
vector<partner>p_tot;
for (auto x : p_base1) p_tot.push_back(x);
for (auto y : p_base2) p_tot.push_back(y);

int min_fly_time_1=maxn, max_fly_time_1=0, ave_fly_time_1=0;
int min_duty_time_1=maxn, max_duty_time_1=0, ave_duty_time_1=0;
int min_duty_num_1=maxn, max_duty_num_1=0, ave_duty_num_1=0;
int num = 0;
int date = 0;

for (auto x : p_tot)
{
  int duty_date = 0;
  for (auto y : x.rws)
  {
    vector<duty>dutys = y.dutys;
    duty_date += dutys.size();
    for (auto z : dutys) //一次执勤
    {
      num++;
      min_fly_time_1 = min(min_fly_time_1, z.duty_time_fly);
```

```
            min_duty_time_1 = min(min_duty_time_1, z.duty_time_tot);

            max_fly_time_1 = max(max_fly_time_1, z.duty_time_fly);
            max_duty_time_1 = max(max_duty_time_1, z.duty_time_tot);

            ave_duty_time_1 += z.duty_time_tot;
            ave_fly_time_1 += z.duty_time_fly;
          }
        }
        min_duty_num_1 = min(min_duty_num_1, duty_date);
        max_duty_num_1 = max(max_duty_num_1, duty_date);
        date += duty_date;
      }
      ave_fly_time_1 /= num;
      ave_duty_time_1 /= num;
      ave_duty_num_1 = date / p_tot.size();

      cout << "max_flytime_1: " << max_fly_time_1 << endl;
      cout << "max_dutytime_1: " << max_duty_time_1 << endl;
      cout << "max_dutynum_1: " << max_duty_num_1 << endl;

      cout << "min_flytime_1: " << min_fly_time_1 << endl;
      cout << "min_dutytime_1: " << min_duty_time_1 << endl;
      cout << "min_dutynum_1: " << min_duty_num_1 << endl;

      cout << "ave_flytime_1: " << ave_fly_time_1 << endl;
      cout << "ave_dutytime_1: " << ave_duty_time_1 << endl;
      cout << "ave_dutynum_1: " << ave_duty_num_1 << endl;

    }
    void Find_effe_duty()
    {
      int num = 0;
      sj = Fli2;
      sort(sj.begin(), sj.end(), cmp);
      //cout << sj.size() << endl;
      vis = vector<bool>(sj.size(), 0);
      vector<duty>dutys;
      for (int i = 0; i < sj.size(); i++)
      {
        flight fx = sj[i];
        if (!vis[i] && (fx.beg == base1 || fx.beg == base2)) //begin
            find
        {
          cout << i << " " << num << endl;
          //cout << fx.beg << endl;
          string cur_base = fx.beg;
          duty dd;
          dd.beg = fx.beg;
          dd.des = fx.des;
          dd.beg_time = fx.beg_tot;
          dd.end_time = fx.arr_tot;
          dd.duty_time_fly += (fx.arr_tot - fx.beg_tot);
          dd.flis.push_back(fx);
```

```
        dd.flies_index.push_back(i);

        bool dd_empty = 0;
        vis[i] = 1;
        int k = i;
        while (k < sj.size())
        {
          //cout << k << endl;
          bool isfind = 0;
          for (int j = k + 1; j < sj.size(); j++)
          {
            if (!vis[j] && isok(k, j) && isdutyok(dd, j))
            {
              dd.des = sj[j].des;
              dd.end_time = sj[j].arr_tot;
              dd.duty_time_fly += (sj[j].arr_tot - sj[j].beg_tot);
              dd.flis.push_back(sj[j]);
              dd.flies_index.push_back(j);


              k = j;
              vis[j] = 1;
              isfind = 1;
              break;
            }
            else if (isok(k, j) && !isdutyok(dd, j))
                //说明一个duty结束了
            {

              dd.duty_time_tot = dd.end_time - dd.beg_time;
              dutys.push_back(dd);

              //如果当前的执勤结束是基地，说明是一个有效的任务
              if (dd.des == cur_base)
              {
                renwu rw;
                rw.dutys = dutys;
                renwus.push_back(rw);
                dutys.clear();
              };
              num += dd.flis.size();
              //重新初始化dd 用sj[j]
              //因为isok==1，说明sj[j]的起点一定和dd的终点相等，也就是等于base，保证了每个
              dd.flis.clear();
              dd.flies_index.clear();
              dd.beg = sj[j].beg;
              dd.beg_time = sj[j].beg_tot;
              dd.des = sj[j].des;
              dd.end_time = sj[j].arr_tot;
              dd.duty_time_fly = 0;
              dd.duty_time_fly += (sj[j].arr_tot - sj[j].beg_tot);
              dd.flis.push_back(sj[j]);
              dd.flies_index.push_back(j);
```

```
            vis[j] = 1;
            k = j;
            isfind = 1;
            break;
          }
        }
        if (isfind == 0) //说明找不到后继节点
        {
          //判断当前的dd是否符合要求 基地到基地
          int len = dd.flis.size();
          bool isnone = 1;
          for (int t = len - 1; t >= 0; t--)
          {
            if (dd.flis[t].des == cur_base)
            {
              isnone = 0;
              break;
            }
            else
            {
              vis[dd.flies_index[t]] = 0;
              dd.flies_index.pop_back();
              dd.flis.pop_back();
            }
          }
          if (isnone == 1)
          {
            for (int ll = 0; ll < dd.flies_index.size(); ll++)
               vis[dd.flies_index[ll]] = 0;
            dd.flies_index.clear();
            dd.flis.clear();
          }
          if (dd.flis.size() > 0) //说明当前的dd有base
          {
            dd.duty_time_tot = dd.end_time - dd.beg_time;
            dutys.push_back(dd);
            num += dd.flis.size();

            renwu rw;
            rw.dutys = dutys;
            renwus.push_back(rw);
            dutys.clear();
          }
          break;
        }
      }

    }
}
cout << dutys.size() << endl;
cout << renwus.size() << endl;
```

```
  for (int i = 0; i < renwus.size(); i++)
  {
    if (i % 100 != 0) continue;
    cout << "任务" << i << endl;
    cout << renwus[i].dutys.size() << endl;
    vector<duty>dutys = renwus[i].dutys;
    for (int j = 0; j < dutys.size(); j++)
    {
      duty dt = dutys[j];
      cout << dt.beg << " " << dt.des << " " << dt.beg_time << "
         " << dt.end_time << " " << dt.flis.size() << endl;
    }

}


void update_renwu()
{
  for (int i = 0; i < renwus.size(); i++)
  {
    renwu x = renwus[i];
    int len = x.dutys.size();
    renwus[i].beg_time = x.dutys[0].beg_time;
    renwus[i].arr_time = x.dutys[len - 1].end_time;
    int sum = 0;
    vector<duty>dutys = x.dutys;
    for (int j = 0; j < dutys.size(); j++)
    {
      sum += dutys[j].duty_time_tot;
    }
    renwus[i].tot_duty_time = sum;
  }
  for (auto x : renwus)
  {
    if (x.arr_time <= x.beg_time) cout << "error_update" << endl;
    //cout << x.tot_duty_time << endl;
  }
}
duty ini_dd(flight f)
{
  duty dd;
  dd.beg = f.beg;
  dd.des = f.des;
  dd.beg_time = f.beg_tot;
  dd.end_time = f.arr_tot;
  dd.duty_time_fly = (f.arr_tot - f.beg_tot);
  dd.flis.push_back(f);
  return dd;
}
duty duty_add(duty dd, flight f)
{
  dd.des = f.des;
  dd.end_time = f.arr_tot;
  dd.duty_time_fly += (f.arr_tot - f.beg_tot);
```

```
  dd.flis.push_back(f);
  return dd;
}

bool can_add_duty(duty dd, flight f) //f可以作为一个新的dd的起点
   这里传进来的dd是之前的，因为要从之前的地点出发
{
  if (dd.des != f.beg) return 0;
  if (f.beg_tot - dd.end_time < min_CT) return 0;
  if (f.beg_tot - dd.end_time < min_rest) return 0;

  if (cal_date(dd.end_time) == cal_date(f.beg_tot)) return 0;

  return 1;
}
bool can_to_f(duty dd, flight f)
{
  if (dd.des == f.beg && f.beg_tot - dd.end_time >= min_CT)
     return 1;
  else return 0;
}
bool can_addf(duty dd, flight f)
{
  if (f.arr_tot - dd.beg_time > max_duty) return 0;
     //超过总的执勤时长
  if ((f.arr_tot - f.beg_tot) + dd.duty_time_fly > max_fly)
     return 0;//超过总的飞行时长
  if (cal_date(dd.beg_time) != cal_date(f.beg_tot)) return 0;
     //加入的sj[j]和duty不在同一天
  return 1;
}
void Find_renwu() //
{
  cout << "begin find renwu" << endl;
  vector<duty>dutys;
  sj = Fli2;
  sort(sj.begin(), sj.end(), cmp);
  //cout << sj.size() << endl;
  vis = vector<bool>(sj.size(), 0);
  for (int i = 0; i < sj.size(); i++)
  {
    if (!vis[i] && (sj[i].beg == base1 || sj[i].beg == base2))
       //说明从i开始
    {
      dutys.clear();
      flight fx = sj[i];
      string cur_base = fx.beg;
      duty dd;
      dd = ini_dd(fx);
      bool dd_empty = 0;
      vis[i] = 1;

      for (int j = i + 1; j < sj.size(); j++)
      {
```

```
            if (vis[j]) continue;
            flight f = sj[j];
            if (dd_empty)
            {
              if (can_add_duty(dd, f))
              {
                dd = ini_dd(f);
                vis[j] = 1;
                dd_empty = 0;
              }
              continue;
            }
            //往下必须要求dd有效
            if (can_to_f(dd, f)) //dd可以到达F, 但是不一定能够加入f
            {
              if (can_addf(dd, f)) //dd能够加入f
              {
                dd = duty_add(dd, f); //更新dd
                vis[j] = 1;
                if (dd.des == cur_base) //说明当前已经找到了有效任务
                    , 放入renwus中
                {
                  dd.duty_time_tot = dd.end_time - dd.beg_time;
                  dutys.push_back(dd);

                  renwu rw;
                  rw.dutys = dutys;
                  renwus.push_back(rw);
                  dutys.clear();
                  dd_empty = 1;
                }
              }
              else //不能加入f, 则结束当前dd, 放入dutys中, 重新找一个dd
              {
                dd.duty_time_tot = dd.end_time - dd.beg_time;
                dutys.push_back(dd);

                dd_empty = 1;
              }
            }
            //不能到达的话就直接看下一个节点

          }
        }
    }
    cout << "任务数量" << endl;
    cout << renwus.size() << endl;
}
void cal_ans()
{
    //求解base1:
    //结果放入p_base1
    sort(jz1.begin(), jz1.end(), [=](partner a, partner b) {
      return a.tot_money < b.tot_money;
```

```
});
unordered_map<int, bool>is_choosed;
sort(rw1.begin(), rw1.end(), [=](renwu a,renwu b) {
  return a.beg_time < b.beg_time;
});
for (int i = 0; i < jz1.size(); i++)
{
  partner x = jz1[i]; //拿出一对组合，给他们分配任务
  for (int j = 0; j < rw1.size(); j++)
  {
    if (is_choosed[j]) continue;
    renwu rw = rw1[j];
    if (rw.beg_time - x.end >= min_rest &&
       cal_date(rw.beg_time) != cal_date(x.end))
    {
      is_choosed[j] = 1; //标记已选
      if (x.rws.size() == 0) //更新
      {
        x.left = rw.beg_time;
      }
      x.end = rw.arr_time;
      x.work_time += rw.tot_duty_time;
      x.rws.push_back(rw);
    }
  }
  if (x.rws.size() > 0) p_base1.push_back(x);
}
cout <<"p_base1.size() : "<< p_base1.size() << endl;

//求解base2:
//结果放入p_base2
sort(jz2.begin(), jz2.end(), [=](partner a, partner b) {
  return a.tot_money < b.tot_money;
});
unordered_map<int, bool>is_choosed2;
sort(rw2.begin(), rw2.end(), [=](renwu a, renwu b) {
  return a.beg_time < b.beg_time;
});
for (int i = 0; i < jz2.size(); i++)
{
  partner x = jz2[i]; //拿出一对组合，给他们分配任务
  for (int j = 0; j < rw2.size(); j++)
  {
    if (is_choosed2[j]) continue;
    renwu rw = rw2[j];
    if (rw.beg_time - x.end >= min_rest &&
       cal_date(rw.beg_time) != cal_date(x.end))
    {
      is_choosed2[j] = 1; //标记已选
      if (x.rws.size() == 0) //更新
      {
        x.left = rw.beg_time;
      }
      x.end = rw.arr_time;
```

```
            x.work_time += rw.tot_duty_time;
            x.rws.push_back(rw);
          }
        }
        if (x.rws.size() > 0) p_base2.push_back(x);
    }
    cout<< "p_base2.size() :"<< p_base2.size() << endl;

}
void renwu_distribution()
{
    for (auto x : renwus)
    {
        if (x.dutys[0].beg == base1)
        {
            rw1.push_back(x);
        }
        else if (x.dutys[0].beg == base2)
        {
            rw2.push_back(x);
        }
        else cout << "error" << endl;
    }
    cout << "任务1和任务2的数量" << endl;
    cout << rw1.size() << " " << rw2.size() << endl;
    //check
    /*cout << "check" << endl;
    for (int i = 0; i <5; i++)
    {
        cout << rw1[i].dutys[0].beg << endl;
    }
    for (int i = 0; i < 5; i++)
    {
        cout << rw2[i].dutys[0].beg << endl;
    }

    sort(rw1.begin(), rw1.end(), [=](renwu a, renwu b) {
        return a.flis > b.flis;
    });
    sort(rw2.begin(), rw2.end(), [=](renwu a, renwu b) {
        return a.flis > b.flis;
    });
    cout << "sorted" << endl;
    for (int i = 0; i < 5; i++)
    {
        cout << rw1[i].flis<<" ";
    }
    cout << endl;
    for (int i = 0; i < 5; i++)
    {
        cout << rw2[i].flis << " ";
    }
    cout << endl;*/
}
```

```
void cal_renwus_flis()
{
  for (int i = 0; i < renwus.size(); i++)
  {
    int sum = 0;
    vector<duty>dutys = renwus[i].dutys;
    for (auto y : dutys)
    {
      sum += y.flis.size();
    }
    renwus[i].flis = sum;
  }
  //check
  /*int num = 0;
  for (auto x : renwus)
  {
    num += x.flis;
  }
  cout << num << endl;*/
}
void check()
{
  long long zong_cost = 0;
  for (auto x : renwus)
  {
    for (auto y : x.dutys)
    {
      for (auto z : y.flis)
      {
        zong_cost += (z.arr_tot - z.beg_tot) * 680 * 2;
      }
    }
  }
  cout << "总成本为: " << zong_cost << endl;

  /*for (auto rw : p_base1)
  {
    vector<renwu>rws = rw.rws;
    for (int j = 0; j < rws.size(); j++)
    {

    }
  }*/
}
int main()
{
  //----------2B------
  clock_t start, end;

  B_sldata_C();
  B_sldata_F();

  start = clock();
```

```
  cal_totB();
  Find_renwu();
  update_renwu();
  crew_distrbution();
  cal_renwus_flis();
  renwu_distribution();
  cal_ans();

  end = clock();

  print_table();
  check();
  cout << "程序运行时长" << endl;
  cout << double(end - start) / CLOCKS_PER_SEC << endl;

  return 0;
}
```

9. 问题三 B 数据集:(核心代码)

```
//删除不符合要求的有效任务
void delete_renwu()
{
  vector<renwu>new_renwus;
  cout << "----------------------" << endl;
  for (int i = 0; i < renwus.size(); i++)
  {
    renwu rw = renwus[i];
    if (rw.dutys.size() <= 4 && (rw.arr_time - rw.beg_time) <=
        max_cir)
    {
      new_renwus.push_back(rw);
    }
  }
  cout << "------------------" << endl;
  cout << renwus.size() << " " << new_renwus.size() << endl;
  renwus.clear();
  renwus = new_renwus;
}


for (int i = 0; i < jz1.size(); i++)
{
  partner x = jz1[i]; //拿出一对组合，给他们分配任务
  for (int j = 0; j < rw1.size(); j++)
  {
    if (is_choosed[j]) continue;
    renwu rw = rw1[j];
    //if(rw.beg_time-x.end>=2880)
    if (cal_date(rw.beg_time) - cal_date(x.end)>=3) //满足休假条件
    {
      is_choosed[j] = 1; //标记已选
      if (x.rws.size() == 0) //更新
      {
```

```
          x.left = rw.beg_time;
        }
        x.end = rw.arr_time;
        x.work_time += rw.tot_duty_time;
        x.rws.push_back(rw);
      }
    }
    if (x.rws.size() > 0) p_base1.push_back(x);
}
```