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	华中科技大学
参赛队号	**21104870099**
队员姓名	**1. 李淑清**
	2. 金　力
	3. 孙　哲

“华为杯”第十八届中国研究生

数学建模竞赛

题 目 **航空公司机组优化排班问题**

摘 要：

近几年航空业快速发展，随着运营管理环境日益复杂，机组排班计划在航空公司实现高效调度和节约运营成本目标中发挥关键作用。然而受到系统复杂性和多种约束条件的限制，机组排班问题被普遍认为是航线规划问题中最复杂的问题之一。在此背景下，本文考虑三个逐步细化优化目标的问题情景下，先后运行 A、B 两套数据求解航空公司机组优化排班问题。

问题一解决航班的机组人员分配，以尽可能多的航班满足机组配为首要目标，以总体乘机次数最少为次要目标，以替补资格使用次数最少为第三目标，求解得出最优决策。

对问题一求解，（1）运行 A 套数据得到结果：206 趟航班满足最低机组配置，航班机组配置成功率达 100%。，一共乘机 8 次，替补 0 次；（2） 运行 B 套数据得到结果：13850 趟航班满足最低机组配置，航班机组配置成功率达 99.75%。一共乘机 374 次，替补 0 次。

问题二引入执勤概念，在满足问题一的基础上增加两个优化目标，以机组人员的总体执勤成本最低为首要目标，以机组人员之间的执勤时长最大化平衡为次要目标，求解得出最优决策。

对问题二求解，（1）运行 A 套数据得到结果：206 趟航班满足最低机组配置，航班机组配置成功率达 100%。，一共乘机 14 次，替补 0 次，共产生总体执勤成本为 53.344 万元；（2） 运行 B 套数据得到结果：13807 趟航班满足最低机组配置，航班机组配置成功率达 99.44%，一共乘机 524 次，替补 0 次，总体执勤成本为 4110.192 万元。

问题三编制排班计划，在前两个问题的基础上，引入每个机组人员的每单位小时任务环成本，分别以机组人员的总体任务环成本最低和机组人员之间的任务环时长最大化平衡为首要和次要目标，求解得出最优决策。

对问题三求解，（1）运行 A 套数据得到结果：146 趟航班满足最低机组配置，航班机组配置成功率达 70.87%，一共乘机 12 次，替补 0 次，一共产生 6.934 万元的执勤成本和 39.48 万元的任务环成本；（2）运行 B 套数据得到结果：9067 趟航班满足最低机组配置，航班机组配置成功率达 65.30%，一共乘机 1820 次，替补 0 次，一共产生 1351.248 万元的执勤成本和 200.4580 万元的任务环成本。

本文的创新点在于尝试将任务环生成和任务环分配两个子任务合成为一个问题进行考虑，并将问题分两个步骤进行优化。第一部分提出了一个构造算法，用于构造初始解，构造过程以保障最大航班起飞数量为目的，在满足其他各项硬约束（如航班乘机人数、开始结束都是在基地、满足机组人员的执勤时长要求等）的基础上，获得一个相对较优的可行解。第二部分即用局部搜索对该可行解进行优化，通过交换机组人员已分配的航班，在保证解可行的范围内，以降低乘机人数、执勤成本、任务环成本为目标进行搜索。算法设计了 Exchange 和 Cross 两个算子，并通过 BestImprove 的方式更新每一次迭代的解，直至算法到达结束条件。

目　录

1 问题重述

1.1 背景介绍

一趟民航航班起飞必须满足国家法律法规、国际公约、公司政策利益等特定条件，以此保证飞航安全和旅客服务质量。广义的机组人员包括飞行员（Pilot，或 Flight Deck），乘务员（Cabin Crew）和空警（Air Marshal）。所谓机组排班问题指的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，该问题的示意图见图 1-1。

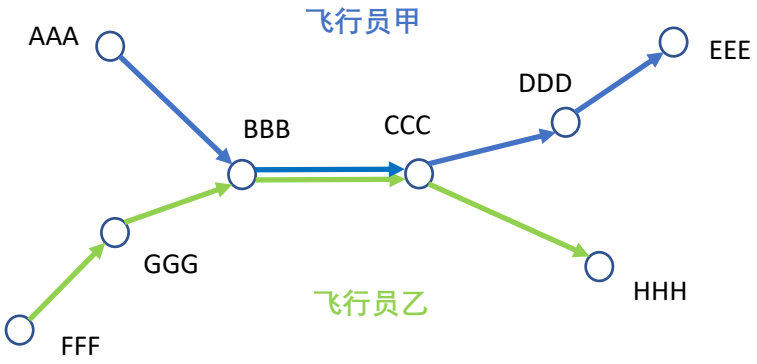

图 1-1　机组排班问题示意图

飞行员甲和飞行员乙有各自的飞行路径，他在执行航班 BBB→CCC 任务时相遇

机组排班问题已形成较成熟甚至通用的解决方案，即将问题拆解成两个子问题，对应形成 Pairing Optimization（PO）和 Roster Optimization（RO）两个优化问题，通过建立网络流模型（Network Flow）然后列生成法（Column Generation）求解。虽然列生成法在处理复杂约束条件、模型简化和计算性能上具有很大优势，但两阶段子问题求解法也存在缺陷，比如不能较好处理人工预排班造成的不完整任务环问题；缩小优化空间；各种规章约束参数主要是在列生成过程中考虑，优化模型本身缺乏这些约束参数的显性表达，通常的参数敏感性分析手段不能采用，因而难以对业务规则的制订起到指导作用。

在此背景下，本赛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。

1.2 问题描述

航空公司的运营管理非常复杂，很多过程需要经过长期-中期-短期等多层次的往复循环。机组排班问题假设航班规划阶段已经完成，机型分配已经结束，对机组人员数量及资格的需求已经明确，而且可用机组人员也已经确定。机组优化排班问题的描述通过以下概念的介绍展开。

时间：时间表达由年月日时分组成，秒以下的精度不计。所有时间均按单一指定时区定义（比如北京时区），相邻两天的时间分割点是给定时区的半夜零点。

机场：航班的起飞和到达，及机组人员的出发和到达，都是以机场为节点。机场的标识一般按照国际民航组织 IATA 标准。比如北京首都机场 PEK，上海浦东国际机场 PVG。

机组人员(Crew)：本题的描述只针对飞行员，但对乘务员和乘警完全适用。现代民航飞机的驾驶通常需要两种资格的飞行员：正机长（也叫机长或正驾驶，Captain）和副机长（也叫副驾驶，First Officer）。

1. 每个机组人员都有一个固定的基地，用所在地机场表达。
2. 每个机组人员都具有一个且仅有一个主要资格，但也可以具备其它的替补资格。机组排班优先安排主要资格，但当主要资格不足时，可以安排替补资格。本题假定具备正机长资格的飞行员其主要资格是正机长，否则其主要资格是副机长。部分正机长可以替补副机长执行飞行任务。机组人员按主要资格或替补资格执行的任务都叫飞行任务。

3. 机组人员除了执行飞行任务外，还可以执行乘机（Deadhead）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去执行飞行任务，乘机人数一般会有限制。

航班(Flight)：指飞机的一次起飞和降落。本题假定每个航班都是唯一的，虽然实际情况下的航班号可能按天或星期重复。当航班应用于机组人员排班时也叫航段（英文 Leg 或 Sector）。

1. 每个航班都有给定的起飞日期、时间，和到达日期、时间。
2. 每个航班都有给定的起飞和到达机场。
3. 每个航班都有给定的最低机组资格配置（Composition），用格式 C<数> F<数> 描述，其中 C<数>为正机长数，F<数>为副机长数。一个航班只有满足了最低机组资格配置才能起飞。本题假定只有一种机型一种配置，实际情况可不同。

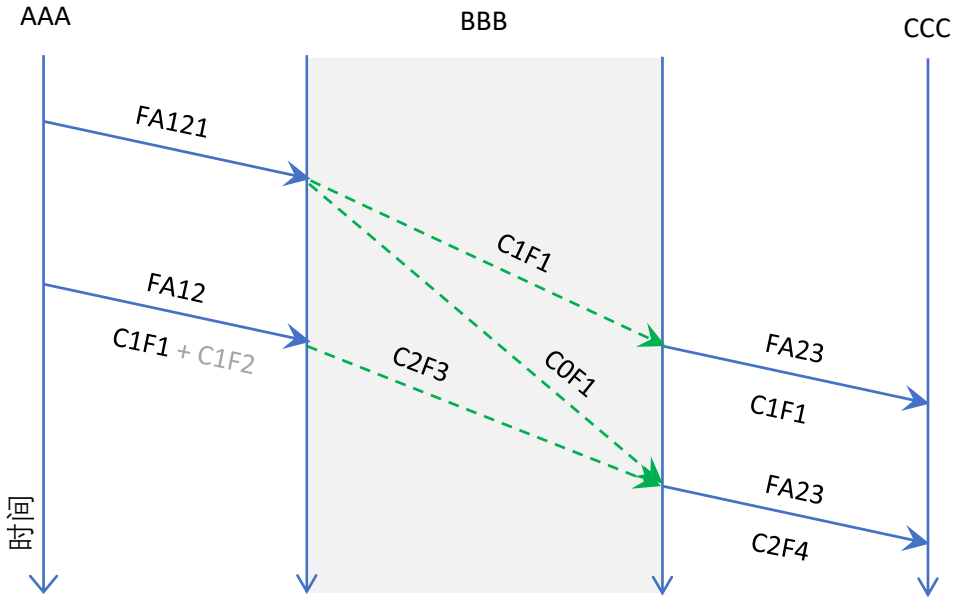

图 1-2　航段及机组分配流动示例图

比如，航班 FA121 机组配置 C1F2，到达机场 BBB 后，其中 C1F1 转服务 FA231，剩余 C0F1 转服务 FA232。航班 FA122 机组配置 C1F1，另有 C1F2 乘机去机场 BBB。

执勤（Duty）：一次执勤由一连串航段（飞行或乘机）和间隔连接时间组成，相当于一次上班。同一次执勤的航段之间满足如下关系：

1. 上一趟航段的到达和下一趟航段的出发机场一致。
2. 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。
3. 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。按起飞时间，我们说这次执勤属于那一天。

航段之间的连接时间计入执勤时间，但不计入飞行时间。执勤起始时间从当天所执行的第一趟航班的起飞时间算起，结束时间按最后一趟航班的到达时间计算。因此，执勤结束时间可以不和该执勤开始时间在同一天。

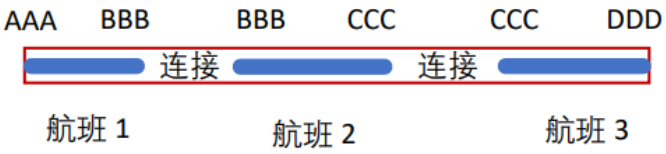

图 1-3　执勤示意图

任务环（Pairing）：任务环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，相当于一次出差。属于同一任务环的执勤之间满足如下关系：

1. 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；
2. 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，但没有上限；
3. 每个任务环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。

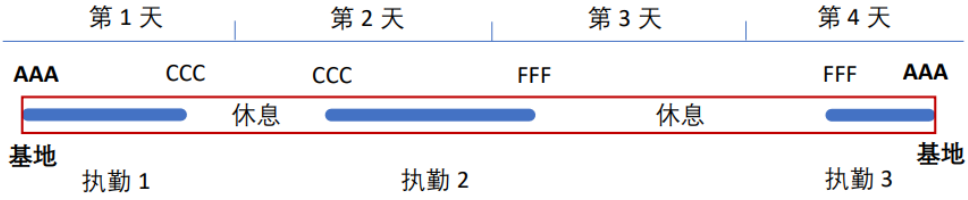

图 1-4　任务环示例：从基地 AAA 出发回到基地

排班计划（Roster）及排班周期：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成，任务环之间满足一定的休假天数。一个排班周期通常是两个星期（双周计划）或者一个月（月计划）。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。

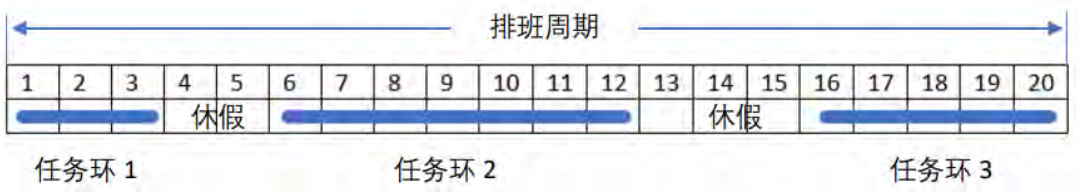

图 1-5　机组人员排班计划示例

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。为减少模型复杂度，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，并对排班周期进行了适当的延展。

1.3 问题提出

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。本题假定：

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，乘机机组人员的航段时间计入执勤时间，但不计入飞行时间。

问题一：建立线性规划模型给航班分配机组人员，依次满足三个目标：①尽可能多的航班满足机组配置;②尽可能少的总体乘机次数;③尽可能少使用替补。需要满足的约束条件为：

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（40）分钟。

问题二：给定每个机组人员的每单位小时执勤成本（小时工资），在满足问题一的基础上，进一步满足两个目标：①机组人员的总体执勤成本最低；②机组人员之间的执勤时长尽可能平衡。同时进一步满足约束：

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（600）分钟；
3. 每次执勤的时长最多不超过 MaxDP（720）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（660）分钟。

问题三：编制排班计划。给定每个机组人员的每单位小时任务环成本（不包括执勤成本，可设想为出差补贴），在满足前两个问题的基础上，还需满足目标：①机组人员的总体任务环成本最低；②机组人员之间的任务环时长尽可能平衡。同时约束条件进一步要求：

1. 每个机组人员每个排班周期的任务环总时长不超过 MaxTAFB（14400）分钟；
2. 每个机组人员相邻两个任务环之间至少有 MinVacDay（2）天休息；
3. 每个机组人员连续执勤天数不超过 MaxSuccOn（4）天。

1.4 数据说明

本赛题有两套检测数据，A 套数据包含 1 个基地，206 趟航班、21 位机组人员和 7 个机场，计划周期从 2021 年 8 月 11 号至 2021 年 8 月 25 号；B 套数据包含 2 个基地、13954 趟航班、465 位机组人员和 39 个机场，计划周期从 2021 年 8 月 1 号至 2021 年 8 月 31 号（部分航班到达时间可能会在计划周期结束后的第一天）。要求首先运行 A 套数据，然后运行 B 套数据。

表 1-1　A 和 B 套数据示例

数据	基地数	计划周期开始	计划周期结束	航班数	机组人数	机场数
A 套	1	2021-08-11	2021-08-25	206	21	7
B 套	2	2021-08-01	2021-08-31	13954	465	39

输入数据由输入参数、航班计划数据和机组人员数据三部分构成，输入参数数据已在问题提出部分说明，航班计划和机组人员数据示例分别见表 1-2 和表 1-3。

表 1-2　航班计划数据示例

航班号 FltNum	出发日期 DptrDate	出发时间 DptrTime	出发机场 DptrStn	到达日期 ArrvDate	到达时间 ArrvTime	达到机场 ArrvStn	最低资格配置 Comp
FA865	8/14/2021	20:00	PXB	8/14/2021	21:45	NKX	C1F1
FB1100	8/18/2019	23:10	TUK	8/19/2019	1:00	TGD	C1F1

表 1-3　机组人员信息数据示例

员工号 EmpNo	正机长 Captain	副机长 FirstOfficer	允许乘机 Deadhead	基地 Base	每单位小时执勤成本(¥) DutyCostPerHr	每单位小时任务环成本(¥) ParingCostPerHr
A0005	Y	Y	Y	NKX	640	20
B0200	Y		Y	TGD	680	20
B0050		Y	Y	HOM	600	20

2 模型假设

根据题意，本文提出如下假设：

1）时间表达由年月日时分组成，秒以下的精度不计。所有时间均按单一指定时区定义（比如北京时区），相邻两天的时间分割点是给定时区的半夜零点。

2）机组人员之间可以任意组合；

3）允许存在因为无法满足最低机组资格配置而不能起飞的航班；

4）不满足最低机组资格配置的航班不能配置任何机组人员；

5）机组人员可以乘机摆渡，即实际机组配置可以超过最低配置要求，乘机机组人员的航段时间计入执勤时间，但不计入飞行时间。

6）航班不存在晚点或提前值机的情况，即航班的实际到达或出发时间与原计划到达或出发时间一致；

7）机组人员不能自行更改航班任务以及任务类型（正机长、副机长、替补、乘机），即对于被分配的航班任务机组人员都严格执行。

8）每个机组人员都能顺利完成任务，不考虑意外导致任务失败的情况。

9）每个航班唯一，不考虑实际情况中航班号可能按天或星期重复的情况。

10）每个机组人员都仅有一个主要资格，但同时可具有替补资格，机组排班优先安排主要资格，部分具有正机长资格的飞行员可替补副机长执行飞行任务。

3 符号说明

符号	释义
K	员工k的集合
I	航班i的集合，特别地，$i=0$表示虚拟航班节点，连接每个员工i初始航班和终止航班（虚拟节点不包含在航班I集合中）
N	日期n的集合
L	任务环编号l的集合
d_i	航班i出发的机场号
d_i'	航班i到达的机场号
d_l^k	员工k第l个任务环出发的机场号
$d_l^{k\prime}$	员工k第l个任务环到达的机场号
$\bar{d}^k$	员工k的基地
e_i	表示航班i出发的日期
e_i'	表示航班i到达的日期
t_i	某趟航班飞行中航班i的出发时间
t_i'	某趟航班飞行中航班i的到达时间
b_{in}	$b_{in}=1$表示航班i的起飞日期在第n天；$b_{in}=0$表示航班i的起飞日期不第n天
m_n^k	员工k的在第n天的执勤时间
g_l^k	员工k的在第l个任务环的时长
p^k	员工k的单位执勤成本
$\bar{p}^k$	员工k的单位时间任务环成本
C^k	$C^k=1$表示员工k具有正机长资格；$C^k=0$表示员工k不具有正机长资格
F^k	$F^k=1$表示员工k具有负机长资格；$F^k=0$表示员工k不具有负机长资格

4 问题一建模与求解

4.1 问题分析

问题一解决机组人员的航班分配问题，包括三个优化目标：首要目标是尽可能多的航班满足机组配置，即通过合理地给航班分配机组人员使满足机组配置的航班最多；第二个目标是使总体乘机次数最少；第三个目标是替补资格的使用次数最少，对应本题即每个具有正机长资格的机组人员尽可能多地以正机长身份执行飞行任务。

本文建模过程与 TSP 的建模过程类似，首先本文以员工为出发点，把航班作为员工待经过的点，员工从基地出发，经过一系列航班中转或执行飞行或乘机任务，最终回到基地，形成一个闭环；在航班的中转过程中，需要考虑航班的起飞条件，包括最低机组配置，航班最多的乘机人数，员工执行的任务类型是否满足航班中转条件，以及航班之间的最小连接时间限制等，最后，在满足航班的飞行条件后，最后员工都有回到基地。

针对目标一，需要尽可能多的保证航班能够起飞。本文设置了一个航班是否能够起飞的变量，通过满足所有约束来保证所有航班起飞数量最大化。

针对目标二，需要最小化执行乘机任务的员工数量。本文设置了一个执行乘机任务的员工变量，在首先满足最大的航班机组设置的基础上，最小化执行乘机任务的员工。

针对目标三，即最小化执行替补资格的正机长数量。本文在优先满足目标一和二的基础上，最小化具备正机长资格的副机长数量。

4.2 模型建立

通过问题分析，我们建立了以下的数学模型。

1）目标函数：

目标一：最大化航班满足机组配置

$$Max \sum_{i \in I} a_i$$

目标二：最小化总体乘机次数

$$Min \sum_{j \in I} \sum_{i \in I} \sum_{k \in K} x_{ij}^k r_j^k$$

目标三：最小化使用替补资格

$$Min \sum_{j \in I} \sum_{i \in I} \sum_{k \in K} x_{ij}^k f_j^k C^k$$

2）决策变量：

$$x_{ij}^{k}=\begin{cases}1，\text{员工}k\text{从航班}i\text{转到航班}j\\0，\text{员工}k\text{不从航班}i\text{转到航班}j\end{cases}$$

$$c_{i}^{k}=\begin{cases}1，\text{员工}k\text{作为正机长执行航班}i\text{的飞行任务}\\0，\text{员工}k\text{不作为正机长执行航班}i\text{的飞行任务}\end{cases}$$

$$f_{i}^{k}=\begin{cases}1，\text{员工}k\text{作为副机长执行航班}i\text{的飞行任务}\\0，\text{员工}k\text{不作为副机长执行航班}i\text{的飞行任务}\end{cases}$$

$$r_{i}^{k}=\begin{cases}1，\text{员工}k\text{搭乘航班}i\text{执行乘机任务}\\0，\text{员工}k\text{不搭乘航班}i\text{执行乘机任务}\end{cases}$$

$$a_{i}=\begin{cases}1，\text{航班}i\text{可以正常起飞}\\0，\text{航班}i\text{不能正常起飞}\end{cases}$$

3）约束条件：

a）员工k从基地出发执行飞行或乘机任务所经历的航班构成一条长链：

$$\begin{cases}\sum_{i\epsilon I}x_{ij}^{k}=1\ \ \forall j\epsilon I,k\in K\\\sum_{j\epsilon I}x_{ij}^{k}=1\ \ \forall i\epsilon I,k\in K\\x_{ij}^{k}\left(a_{j}-1\right)=0\quad\forall i,j\epsilon I,k\in K\end{cases}\tag{4-1}$$

b）每个机组人员初始从基地出发并最终回到基地：

$$\begin{cases}x_{i0}^{k}\left(d_{i}^{\prime}-\bar{d}^{k}\right)=0\ \ \forall j\epsilon I\\x_{0j}^{k}\left(d_{j}-\bar{d}^{k}\right)=0\ \ \forall j\epsilon I\end{cases}\tag{4-2}$$

c）员工k在航班i的终点站与航班j的起点站相同时可以中转：

$$\left(d_{i}^{\prime}-d_{j}\right)x_{ij}^{k}=0\ \ \forall i,j\epsilon I/\{0\},k\in K\tag{4-3}$$

d）员工k在从航班i转到航班j时的中转时间满足:

$$x_{ij}^{k}\left(t_{i}^{\prime}+MinCT-t_{j}\right)\leq 0\ \ \forall i,j\epsilon I,k\in K\tag{4-4}$$

e）包括正机长和副机长在内一趟航班的乘机人数最多为$2+MaxDH$人：

$$\sum\nolimits_{i\epsilon I,k\epsilon K}x_{ij}^{k}\leq 2+MaxDH\ \ \forall j\epsilon I\tag{4-5}$$

f）员工k在航班i上只有一种身份，即要么为正机长或副机长或乘机人员：

$$c_{i}^{k}+f_{i}^{k}+r_{i}^{k}=1\ \ \forall i\epsilon I/\{0\},k\in K\tag{4-6}$$

g）员工k作为正机长时需要具有正机长资格：

$$c_i^k(C^k-1)=0\ \ \forall i\epsilon I,k\in K \tag{4-7}$$

h）员工k作为副机长时需要具有副机长资格：

$$f_i^k(F^k-1)=0\ \ \forall i\epsilon I,k\in K \tag{4-8}$$

i）航班i只能有一个正机长：

$$\sum\nolimits_{i\in I,k\in K} x_{ij}^k c_j^k=1\ \ \forall j\epsilon I \tag{4-9}$$

j）航班i只能有一个副机长：

$$\sum\nolimits_{i\in I,k\in K} x_{ij}^k f_j^k=1\ \ \forall j\epsilon I \tag{4-10}$$

综上，**针对问题一建立的数学模型如下：**

目标① $Max\sum_{i\in I}a_i$

目标④ $Min\sum_{j\in I}\sum_{i\in I}\sum_{k\in K}x_{ij}^k r_j^k$

目标⑦ $Min\sum_{j\in I}\sum_{i\in I}\sum_{k\in K}x_{ij}^k f_j^k C^k$

$$\text{s.t.}\begin{cases}\sum_{i\in I}x_{ij}^k=1\ \ \forall j\epsilon I,k\in K\\ \sum_{j\in I}x_{ij}^k=1\ \ \forall i\epsilon I,k\in K\\ x_{ij}^k\left(a_j-1\right)=0\quad \forall i,j\epsilon I,k\in K\\ x_{i0}^k\left(d_i'-\bar{d}^k\right)=0\ \ \forall i\epsilon I,k\in K\\ x_{0j}^k\left(d_j-\bar{d}^k\right)=0\ \ \forall j\epsilon I,k\in K\\ \left(d_i'-d_j\right)x_{ij}^k=0\ \ \forall i,j\epsilon I/\{0\},k\in K\\ x_{ij}^k\left(t_i'+MinCT-t_j\right)\le 0\ \ \forall i,j\epsilon I,k\in K\\ \sum_{i\in I,k\in K}x_{ij}^k\le 2+MaxDH\ \ \forall j\epsilon I\\ c_i^k+f_i^k+r_i^k=1\ \ \forall i\epsilon I/\{0\},k\in K\\ c_i^k(C^k-1)=0\ \ \forall i\epsilon I,k\in K\\ f_i^k(F^k-1)=0\ \ \forall i\epsilon I,k\in K\\ \sum_{i\in I,k\in K}x_{ij}^k c_j^k=1\ \ \forall j\epsilon I\\ \sum_{i\in I,k\in K}x_{ij}^k f_j^k=1\ \ \forall j\epsilon I\end{cases}$$

4.3 算法求解

模型建立完成后，采用构造法完成员工与航班的匹配，设计启发式算法对该问题进行求解，在满足约束集的条件下尽量满足目标函数。

算法步骤及流程图如下：

1）算法步骤

Step1：初始化航班和员工并将航班排序。按照航班的出发时间和到达时间从小到大排序；同时对航班的合理性进行判断，剔除不合理的航班（不合理的航班包括：①正副机长无法在航班起飞前通过任何途径到达；②正副机长完成航班后在一个排班周期内无法返回基地）。

Step2：给当前航班分配副机长。先遍历只具有副机长资格的员工，再遍历同时具有正副机长资格的员工。若没有满足时间和地点约束的副机长，则进入 Step4；否则进入 Step3。

Step3：给当前航班分配正机长。先遍历同时具有正副机长资格的员工，再遍历只有正机长资格的员工。若没有满足时间和地点约束的正机长，则进入 Step4；若航班还有未分配员工，则进入 Step2；否则进入 Step5。

Step4：在更早的航班中加入搭乘人员，执行相应飞行任务。若找到对应职务员工，进入 Step3；若找不到且还有更早的航班，则进入 Step4；否则寻找失败，将当前航班剔除，进入 Step2，继续分配下一班航班。

Step5：员工返回基地。遍历所有员工所在位置，判断是否在基地，搭乘最晚航班返回基地，若不能没有返回航班，则取消相应员工上次回到基地之前的航班，并重新搭乘其他航班返回基地。

Step6：所有航班匹配完成，算法结束。

2）算法流程图

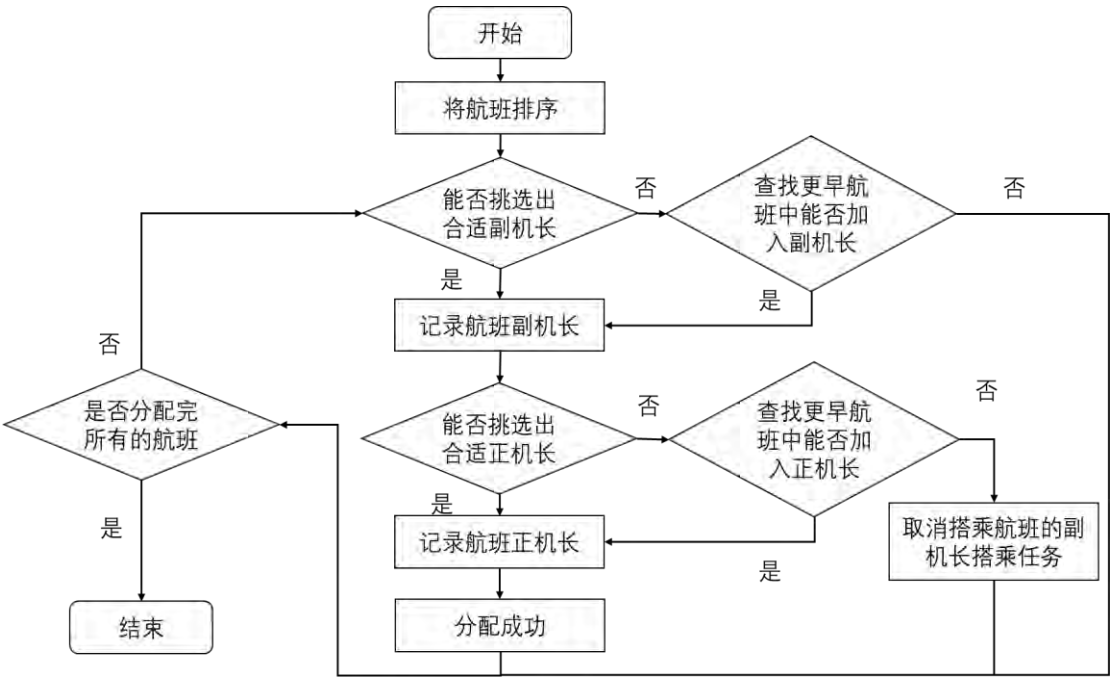

图 4-1　算法流程图

4.4 结果分析

利用以上的算法分别运行 A 套数据和 B 套数据，对应得出最优的航班机组人员分配方案，以下称为方案 1-A 和方案 1-B。表 4-1 给出了关键指标统计，并在表 4-2 中给出了 A 套数据部分结果的示意。

表 4-1 关键指标结果统计表

数据 \ 指标	不满足机组配置航班数	满足机组配置航班数	机组人员总体乘机次数	替补资格使用次数	程序运行时间（分）
A 套数据	0	206	8	0	0
B 套数据	35	13850	373	0	2

从计算结果可以看出，在 A 套数据中，206 趟航班全部满足机组配置，其中机组人员总体乘机次数为 8 次，替补次数为 0。在 B 套数据中，不满足机组配置的航班数为 35 个（剔除肯定不能起飞的航班 69 趟，不能起飞的原因在于最后一天出发的航班无法回来），占比 0.25%，满足机组配置的航班数为 13850 个航班机组配置成功率达 99.75%，其中机组人员总体乘机次数为 373 次，正机长使用替补资格执行飞行任务为 7 次。

表 4-2 A 套数据部分结果示意表

工号 EmpNo	航班号 FltNum	出发日期 DptrDate	出发时间 DptrTime	出发机场 DptrStn	到达日期 ArrvDate	达到时间 ArrvTime	达到机场 ArrvStn	正机长 Captain	副机长 FirstOffice	替补 Substitute	乘机 Deadhea
A0001	FA2	8/12/2021	10:10	PGX	8/12/2021	11:40	NKX	Y			
A0002	FA680	8/11/2021	8:00	NKX	8/11/2021	9:30	PGX				Y
A0002	FA3	8/12/2021	10:25	PGX	8/12/2021	11:40	NKX	Y			
A0003	FA884	8/11/2021	11:30	NKX	8/11/2021	13:50	XGS				Y
A0003	FA891	8/12/2021	10:30	XGS	8/12/2021	12:50	NKX	Y			
A0004	FA884	8/11/2021	11:30	NKX	8/11/2021	13:50	XGS				Y
A0004	FA891	8/15/2021	10:30	XGS	8/15/2021	12:50	NKX	Y			
A0005	FA680	8/11/2021	8:00	NKX	8/11/2021	9:30	PGX	Y			
A0005	FA681	8/11/2021	10:10	PGX	8/11/2021	11:40	NKX	Y			
A0005	FA812	8/11/2021	12:20	NKX	8/11/2021	14:05	PDK	Y			
A0005	FA813	8/11/2021	14:50	PDK	8/11/2021	16:40	NKX	Y			
A0005	FA854	8/11/2021	17:20	NKX	8/11/2021	19:00	CTH	Y			
A0005	FA855	8/11/2021	19:45	CTH	8/11/2021	21:30	NKX	Y			
A0005	FA872	8/12/2021	7:55	NKX	8/12/2021	9:00	PLM	Y			
A0005	FA873	8/12/2021	9:40	PLM	8/12/2021	10:50	NKX	Y			
A0005	FA884	8/12/2021	11:30	NKX	8/12/2021	13:50	XGS	Y			
A0005	FA885	8/12/2021	14:30	XGS	8/12/2021	16:50	NKX	Y			
A0005	FA864	8/12/2021	17:30	NKX	8/12/2021	19:15	PXB	Y			
A0005	FA865	8/12/2021	20:00	PXB	8/12/2021	21:45	NKX	Y			
A0005	FA872	8/13/2021	7:55	NKX	8/13/2021	9:00	PLM	Y			
A0005	FA873	8/13/2021	9:40	PLM	8/13/2021	10:50	NKX	Y			
A0005	FA884	8/13/2021	11:30	NKX	8/13/2021	13:50	XGS	Y			
A0005	FA885	8/13/2021	14:30	XGS	8/13/2021	16:50	NKX	Y			
A0005	FA864	8/13/2021	17:30	NKX	8/13/2021	19:15	PXB	Y			
A0005	FA865	8/13/2021	20:00	PXB	8/13/2021	21:45	NKX	Y			

1）机组人员任务分析

方案 1-A 中，机组人员的任务执行总体情况如表 4-3，从结果来看：仅需 13 位（占机组人员总数的 61.90%）机组人员执行任务就能实现所有航班起飞，意味着在航班满足机组配置最优化的结果下，能够为公司节省 8 位员工成本。另外，所有员工一共执行了 412 次任务，其中执行乘机任务 8 次，替补 0 次。

表 4-3　机组人员任务执行总体结果统计（A 套数据）

机组人员总数	执行任务的员工数	未执行任务的员工数	执行任务的员工人数占比	未执行任务的员工人数占比	飞行任务次数	替补次数	乘机任务次数	任务总次数
21	13	8	61.90%	38.10%	412	0	8	420

机组人员的任务执行的具体情况如表 4-4，从结果来看：①13 位执行任务的机组人员中包含 3 位同时具有正副机长资格的员工，10 位仅具有副机长资格的员工；②共完成 412 次飞行任务，包括了 206 次以正机长和 206 次以副机长资格执行的飞行任务；③一共执行了 8 次乘机任务和 0 次替补；④执行任务总次数为 420 次；⑤人均执行任务次数约为 32 次/人，但 A0005、A0006、A0007 这 3 位员工以正机长身份执行了 202 次飞行任务，A0012、A0013、A0014 这 3 位员工以副机长身份执行了 203 次飞行任务，人均任务量分配差距大，还有待进一步优化。

表 4-4　机组人员任务执行结果统计（A 套数据）

工号 EmpNo	是否具有替补资格	正机长 Captain	副机长 FirstOfficer	替补 Substitute	乘机 Deadhead	飞行任务次数	任务总次数（飞行+乘机）
A0001	N	1	0	-	1	1	2
A0002	N	1	0	-	1	1	2
A0003	N	1	0	-	1	1	2
A0004	N	1	0	-	1	1	2
A0005	Y	76	0	0	0	76	76
A0006	Y	76	0	0	0	76	76
A0007	Y	50	0	0	0	50	50
A0012	N	0	76	-	0	76	76
A0013	N	0	76	-	0	76	76
A0014	N	0	51	-	1	51	52
A0015	N	0	1	-	1	1	2
A0016	N	0	1	-	1	1	2
A0017	N	0	1	-	1	1	2
总计	-	206	206	0	8	412	420

方案 1-B 中，机组人员的任务执行的总体情况如表 4-5，从结果来看：仅需 360 位机组人员执行任务就能实现所有航班起飞，只占总员工的 77.42%，即在航班满足机组配置最优化的结果下，为公司节省了 105 位员工成本。另外，所有员工一共执行了 27770 次任务，人均执行约 78 次任务，其中最少执行 2 次任务，最多执行 214 次任务；与此同时，一共执行乘机任务 373 次，替补 0 次。

表 4-5　机组人员任务执行总体结果统计（B 套数据）

机组人员总数	执行任务的员工数	未执行任务的员工数	执行任务的员工人数占比	未执行任务的员工人数占比	飞行任务次数	乘机任务次数	替补次数	任务总次数
465	360	105	77.42%	22.58%	27770	373	0	27770

因执行任务的机组人员有 360 位，人数过多，取执行的任务次数区间分布图进行结果说明比较合适，如图 X 所示，46.94%的员工执行的任务次数不超过 20，15%的员工执行的任务次数在 181-200 次之间，执行任务次数在 41-60 区间的员工数最少，仅占 2.22%。

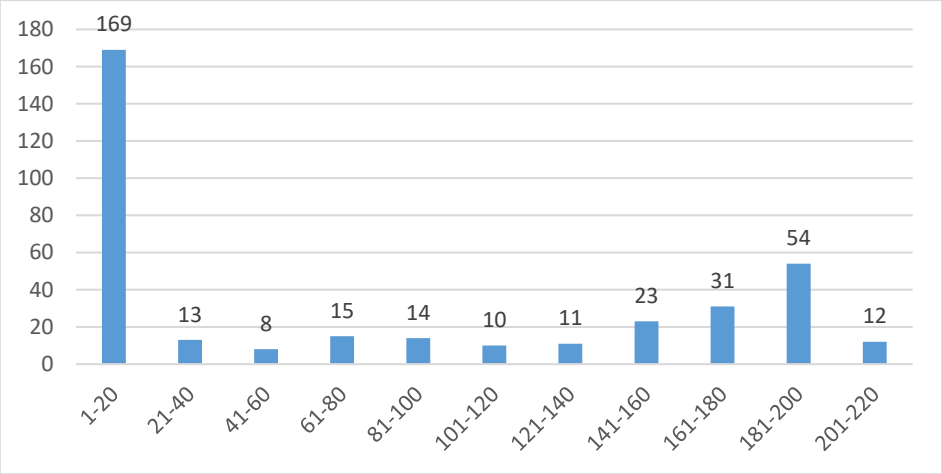

图 4-2　总体机组人员任务执行次数区间分布图（B 套数据）

另外，我们还考虑了 124 位同时具有正副机长资格的员工任务执行情况，他们一共完成 13758 次任务，包含 109 次乘机任务，因为替补次数为 0，故全部以正机长身份完成飞行任务 13649 次，占总任务次数高达 99.21%。执行任务次数区间分布图见表 4-3，人均执行任务次数约 111 次，且执行任务次数在 1-20 和 181-200 区间的人数最多。

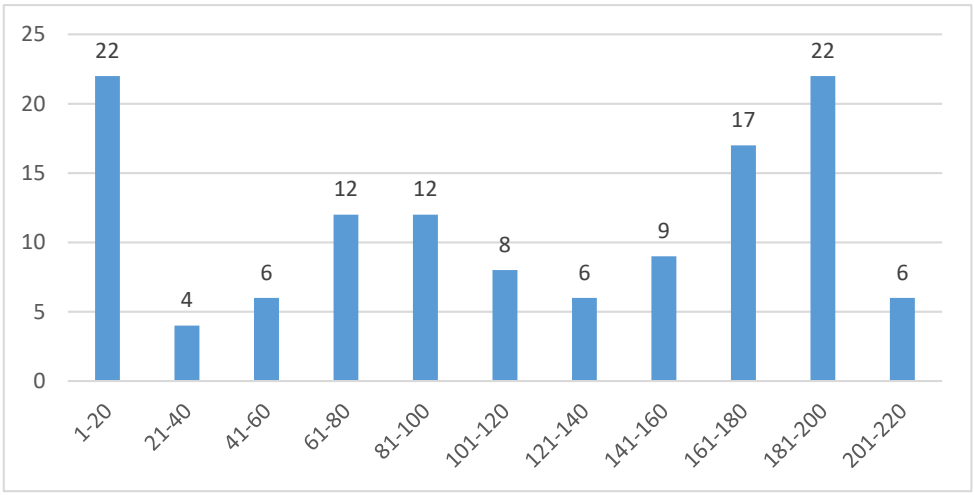

图 4-3　具有正副机长资格机组人员任务执行次数区间分布图（B 套数据）

2）机场航班起飞结果分析

由于 A 套数据机场航班全部满足，所以只需要分析 B 套数据的机场航班情况。在 B 套数据中，有 12 个机场的航班全部能够按照计划起飞，其他机场的航班起飞比例基本在 99%以上，只有 XXJ 机场的起飞比例比较低，为 95.65%。此

外，TGD 机场的计划起飞量远远超过其他机场，但其航班起飞比例也在 99.5%以上。

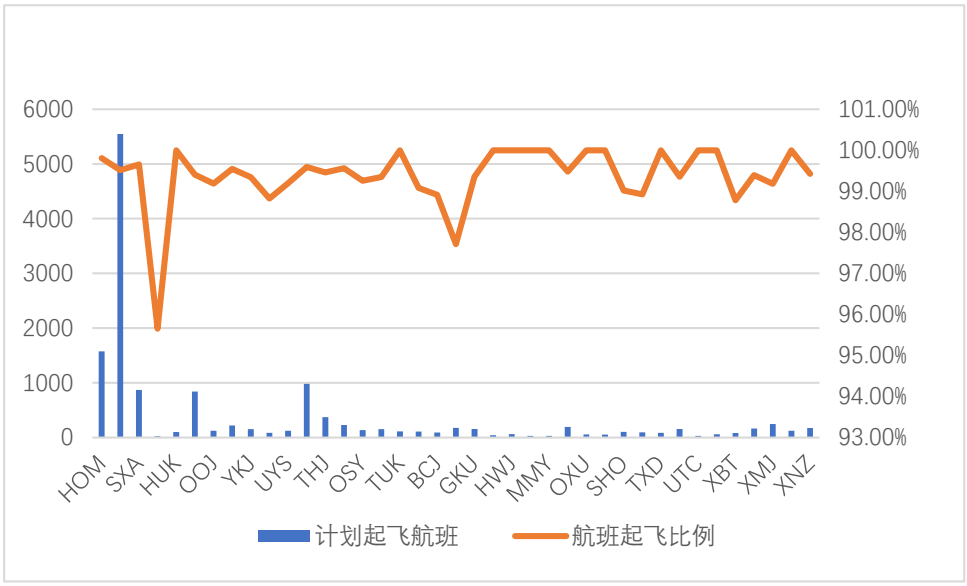

图 4-4　机场航班起飞

从数量上来看（图 4-5），航班不能起飞的数量最多是 TGD 机场的 27 次，其余机场的起飞数量控制在 5 次以下，少部分机场不能起飞的航班为 2 到 3 次，大部分机场不能起飞的航班为 1 次或者 0 次。

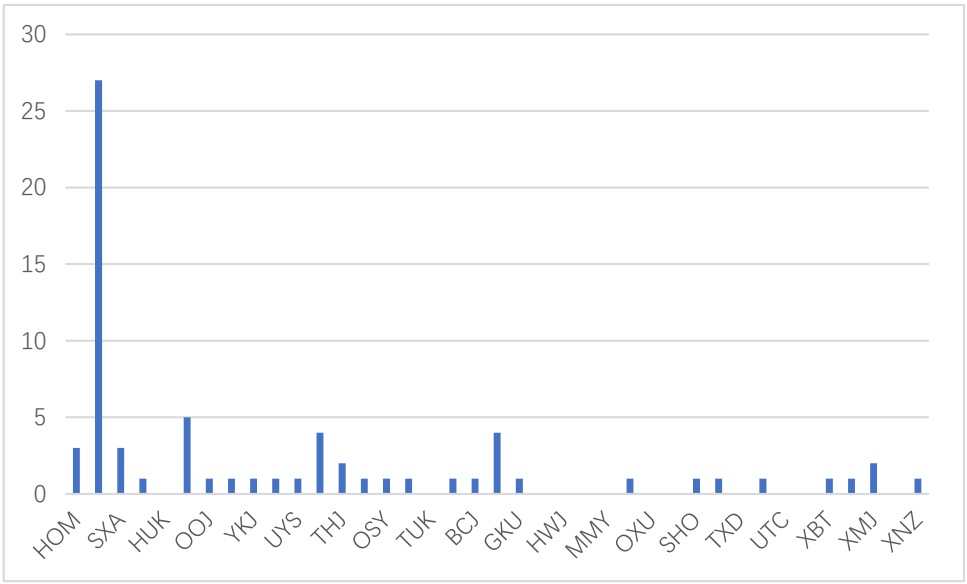

图 4-5　机场航班不能起飞数量

3）机场航班到达结果分析

在 B 套数据中，有 13 个机场的航班能按照计划到达，航班到达的比例基本控制在 99%以上，其中 GKS 机场的到达比例最低，为 98.54%。

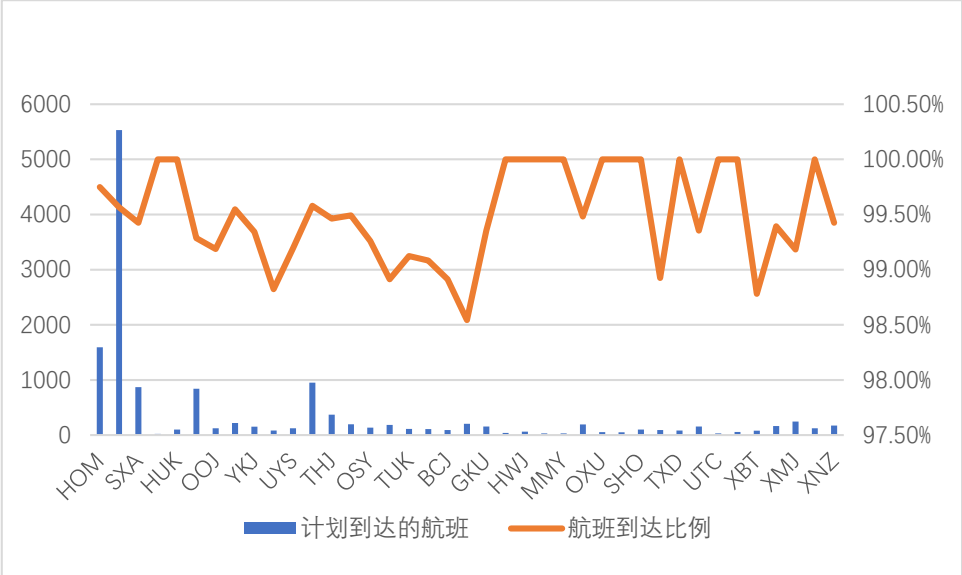

图 4-6　机场航班到达

从数量上来看（图 4-7），大部分不能按计划到达的航班集中在一个机场---TGD，机场不能到达的数量基本控制在 5 次以下。

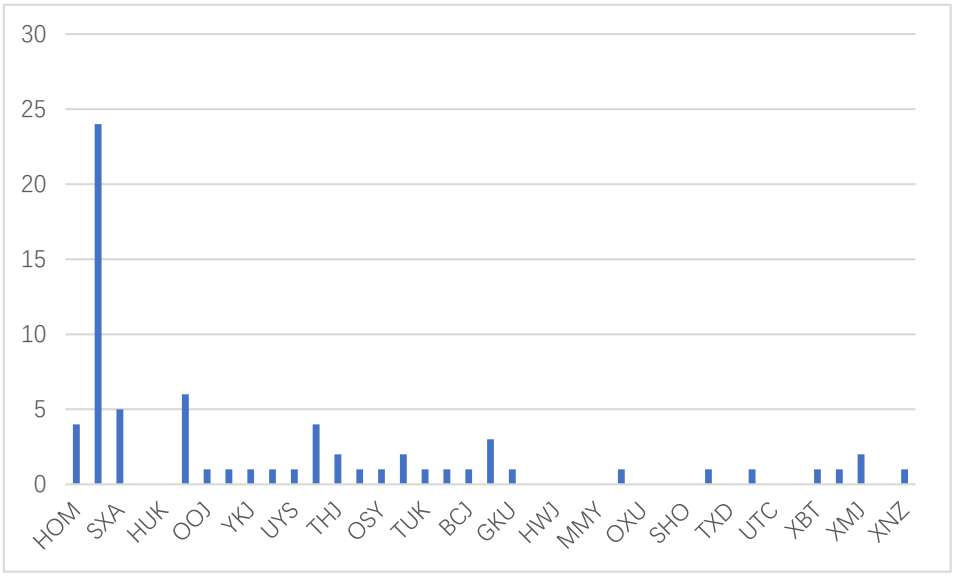

图 4-7　机场航班不能到达数量

5 问题二建模与求解

5.1 问题分析

问题二引进了执勤的概念，在满足问题一的目标和约束的基础上，需要满足以下两个目标：1）机组人员的总体执勤成本最低；2）机组人员之间的执勤时长尽可能平衡。

在执勤的过程中，每个机组人员每天只能执行一个执勤，因此，员工的执勤过程可以按照天数来进行切割。每次的执勤过程中包含了航段和连接时间，需要考虑执勤的飞行时间和执勤总时间满足约束，以及两次执勤之间的休息时间得到保障。

5.2 模型建立

问题二首先保证尽可能多的航班满足机组配置，然后最小化机组人员的总执勤成本，第三个目标是尽可能少的总体乘机次数，第四个目标是尽可能平衡机组人员的执勤时长，最后最小化替补资格。为了避免重复阐述，我们首先分析本问题新增的目标函数和约束，然后再展示完整的模型。

问题二新增的目标函数和约束如下：

1）目标函数：

目标一：机组人员的总体执勤成本最低

$$Max\sum_{k\epsilon K}\left(p^k\sum_{n\in N}m_n^k\right)$$

目标二：机组人员之间的执勤时长尽可能平衡

$$Min\left(Max\left(\sum_{n\in N}m_n^k\right)-Min\left(\sum_{n\in N}m_n^k\right)\right)$$

2）新增决策变量：

$$u_n^k=\begin{cases}1，\text{员工}k\text{在第}n\text{天执勤}\\0，\text{员工}k\text{不在第}n\text{天执勤}\end{cases}$$

3）新增约束条件：

a）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：：

$$x_{ij}^k\left(e_j'-e_i'\right)\left(t_i'+MinRest-t_j\right)\le 0\ \ \forall i,j\epsilon I,k\in K \qquad (5-1)$$

b）通过判断员工k是否有飞行任务，得到在第n天是否值班：

$$\begin{cases}\left(u_n^k-1\right)\left(\sum_{j\in I}\sum_{i\in I}x_{ij}^k\,b_{jn}\right)=0\\ u_n^k-\sum_{j\in I}\sum_{i\in I}x_{ij}^k\,b_{jn}\le 0\end{cases}，\ \forall k\epsilon K,n\epsilon N \qquad (5-2)$$

c）每次执勤的飞行时间最多不超过 MaxBlk 分钟：

$$u_n^k \sum_{j\in I}\sum_{i\in I}\left(x_{ij}^k b_{jn}\left(t_j' - t_j\right)\left(c_j^k + f_j^k\right)\right) \le MaxBlk，\ \forall k\epsilon K, n\epsilon N \qquad (5-3)$$

d）每次执勤的时长包括执行乘机或飞行任务时长和航班连接时间：

$$m_n^k = u_n^k \sum_{j\in I}\sum_{i\in I}\left(x_{ij}^k b_{jn}\left(t_j' - t_j + b_{in}\left(t_j - t_i'\right)\right)\right)，\ \forall k\epsilon K, n\epsilon N \qquad (5-4)$$

e）每次执勤的时长最多不超过 MaxDP 分钟：

$$m_n^k \le MaxDP，\ \forall k\epsilon K, n\epsilon N \qquad (5-5)$$

综上，**针对问题二建立的数学模型如下：**

目标① $Max \sum_{i\in I} a_i$

目标② $Max \sum_{k\epsilon K}(p^k \sum_{n\in N} m_n^k)$

目标④ $Min \sum_{j\in I}\sum_{i\in I}\sum_{k\in K} x_{ij}^k r_j^k$

目标⑤ $Min\left(Max(\sum_{n\in N} m_n^k) - Min(\sum_{n\in N} m_n^k)\right)$

目标⑦ $Min \sum_{j\in I}\sum_{i\in I}\sum_{k\in K} x_{ij}^k f_j^k C^k$

$$
\text{s.t.}\begin{cases}
\sum_{i\epsilon I} x_{ij}^{k} = 1 \ \ \forall j\epsilon I, k \in K \\
\sum_{j\epsilon I} x_{ij}^{k} = 1 \ \ \forall i\epsilon I, k \in K \\
x_{ij}^{k}\left(a_j - 1\right) = 0 \quad \forall i, j\epsilon I, k \in K \\
x_{i0}^{k}\left(d_i' - \bar{d}^{k}\right) = 0 \ \ \forall i\epsilon I, k \in K \\
x_{0j}^{k}\left(d_j - \bar{d}^{k}\right) = 0 \ \ \forall j\epsilon I, k \in K \\
\left(d_i' - d_j\right)x_{ij}^{k} = 0 \ \ \forall i, j\epsilon I/\{0\}, k \in K \\
x_{ij}^{k}\left(t_i' + MinCT - t_j\right) \le 0 \ \ \forall i, j\epsilon I, k \in K \\
\sum_{i\epsilon I, k\epsilon K} x_{ij}^{k} \le +MaxDH \quad \forall j\epsilon I \\
c_i^{k} + f_i^{k} + r_i^{k} = 1 \ \ \forall i\epsilon I/\{0\}, k \in K \\
c_i^{k}(C^{k} - 1) = 0 \ \ \forall i\epsilon I, k \in K \\
f_i^{k}(F^{k} - 1) = 0 \ \ \forall i\epsilon I, k \in K \\
\sum_{i\epsilon I, k\epsilon K} x_{ij}^{k} c_j^{k} = 1 \quad \forall j\epsilon I \\
\sum_{i\epsilon I, k\epsilon K} x_{ij}^{k} f_j^{k} = 1 \quad \forall j\epsilon I \\
x_{ij}^{k}\left(e_j' - e_i'\right)\left(t_i' + MinRest - t_j\right) \le 0 \ \ \forall i, j\epsilon I, k \in K \\
\begin{cases} (u_n^{k} - 1)\left(\sum_{j\epsilon I}\sum_{i\epsilon I} x_{ij}^{k} b_{jn}\right) \\ u_n^{k} - \sum_{j\epsilon I}\sum_{i\epsilon I} x_{ij}^{k} b_{jn} \le 0 \end{cases}, \ \forall k\epsilon K, n\epsilon N \\
u_n^{k}\sum_{j\epsilon I}\sum_{i\epsilon I}\left(x_{ij}^{k} b_{jn}\left(t_j' - t_j\right)\left(c_j^{k} + f_j^{k}\right)\right) \le MaxBlk, \ \forall k\epsilon K, n\epsilon N \\
m_n^{k} = u_n^{k}\sum_{j\epsilon I}\sum_{i\epsilon I}\left(x_{ij}^{k} b_{jn}\left(t_j' - t_j + b_{in}\left(t_j - t_i'\right)\right)\right), \ \forall k\epsilon K, n\epsilon N \\
m_n^{k} \le MaxDP, \ \forall k\epsilon K, n\epsilon N
\end{cases}
$$

5. 3 算法求解

模型建立完成后，采用构造法完成员工与航班的匹配，设计启发式算法对该问题进行求解，在满足约束集的条件下尽量满足目标函数。

1）算法一：构造初始解

Step1：初始化航班和员工并将航班排序。按照航班的出发时间和到达时间从小到大排序；同时对航班的合理性进行判断，剔除不合理的航班（不合理的航班包括：①正副机长无法在航班起飞前通过任何途径到达；②正副机长完成航班后在一个排班周期内无法返回基地）。

Step2：判断当前的时间节点，若开始安排新一天的航班，则说明进入下一天，需更新员工的执勤状态。

Step3：给航班分配副机长。先遍历只具有副机长资格的员工，再遍历同时具有正副机长资格的员工。若没有满足执勤和飞行时间约束以及地点约束的副机长，则进入 Step5；分配成功则进入 Step4。

Step4：给航班分配正机长。先遍历同时具有正副机长资格的员工，再遍历只有正机长资格的员工。若没有满足执勤和飞行时间约束以及地点约束的正机长，则进入 Step5；若分配成功，则进入 Step2，继续分配下一班航班，否则进入 Step6。

Step5：在更早的航班中加入搭乘人员，执行乘机任务到达指定机场。若找到对应职务员工，进入 Step3；否则寻找失败，将当前航班剔除，进入 Step2，继续分配下一班航班。

Step6：员工返回基地。遍历所有员工所在位置，判断是否在基地，搭乘最晚航班返回基地，若不能没有返回航班，则取消相应员工上次回到基地之前的航班，并重新搭乘其他航班返回基地。

Step7：所有航班匹配完成，算法结束。

2）算法二：邻域搜索算法

Step1：计算初始解的目标函数值。

Step2：对航班和人员进行编码。构建员工的航班链，并标识职位（正机长，副机长，搭乘人员）。

Step3：遍历所有员工的航班，进行扰动并判断扰动有效性。对相近时间和相同机场的航班执行 Exchange 和 Cross 操作，判断是否可行，若可行则计算目标函数值，当扰动得到结果更优时，记录航班并更新员工航班链。

Step4：遍历员工数组，执行 Step3 中扰动操作直到满足终止条件，记录最优的员工航班链。算法结束。

Exchange 扰动算子将相近两个航班的飞行员工进行交换，如图（5-1）所示，员工i和员工j的两个航班a和航班b起始和到达位置相同，若满足交换条件则交换a和b，两个员工的执勤时间可能会发生改变，若最终目标值由于现有值，则记录并更新航班链。

Cross 扰动算子将相近两个航班之后的航班飞行员交换，如图（5-2）所示，员工i和员工j的两个航班a和航班b到达位置，若满足交换条件则交换a和b后续航班，两个员工的执勤时间等可能会发生改变，若最终目标值优于现有值，则记录并更新航班链。

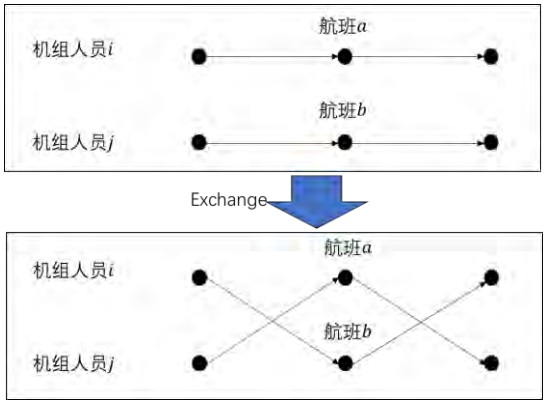

图 5-1　Exchange 算子

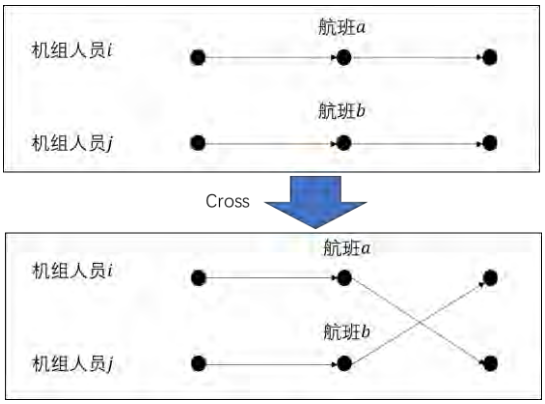

图 5-2　Cross 算子

5.4 结果分析

利用以上的算法分别运行 A 套数据和 B 套数据，对应得出最优的航班机组人员分配方案，以下称为方案 2-A 和方案 2-B。表 5-1 给出了关键指标统计结果。

表 5-1 关键指标结果统计表 1-1

数据 \ 指标	不满足机组配置航班数	满足机组配置航班数	机组人员总体乘机次数	替补资格使用次数	机组总体利用率	最小/最大/平均执勤飞行时长（时）		
A 套数据	0	206	14	0	79.64%	75	490	219
B 套数据	78	13807	524	0	63.02%	45	560	236

表 5-1 关键指标结果统计表 1-2

数据 \ 指标	最小/最大/平均执勤执勤时长（时）			最小/最大/平均机组人员执勤天数			总体执勤成本(万元）	程序运行时间（分）
A 套数据	75	705	275	2	15	9	53.344	0.021
B 套数据	45	720	375	8	31	24	4110.192	2.709

从计算结果可以看出，在 A 套数据中，206 趟航班全部满足机组配置，其中机组人员总体乘机次数为 8 次，替补次数为 0，机组总体利用率达 79.64%，执勤的最小、最大和平均飞行时长分别为 75h、490h、219h，执勤的最小、最大和平均执勤时长分别为 75h、705h、275h，执勤的最小、最大和平均执勤天数分别为 2 天、15 天、9 天，一共产生 53.344 万元的执勤成本。

在 B 套数据中，13885 趟航班中有 13807 趟航班满足最低机组配置，航班机组配置成功率达 99.44%，其中乘机任务 524 次，替补 0 次，机组总体利用率达 63.02%，执勤的最小、最大和平均飞行时长分别为 45h、560h、236h，执勤的最小、最大和平均执勤时长分别为 45h、709h、375h，执勤的最小、最大和平均执勤天数分别为 8 天、31 天、24 天，一共产生 4110.192 万元的执勤成本。

1）机组人员分配结果分析

方案 2-A 中，机组人员的任务执行的总体情况如表 5-2，从结果来看：21 位机组人员都执行任务，一共完成 426 次任务，包括 412 次飞行任务，1 次替补任务，与此同时产生 14 次乘机任务。

表 5-2　机组人员任务执行总体结果统计（A 套数据）

机组人员总数	执行任务的员工数	未执行任务的员工数	执行任务的员工人数占比	未执行任务的员工人数占比	飞行任务次数	替补次数	乘机任务次数	任务总次数
21	21	0	100.00%	0.00%	412	1	14	426

机组人员的任务执行的具体情况如表 5-3 所示，一位员工最少执行 2 次任务，最多执行 40 次任务，人均执行约 20 次任务。

表 5-3　机组人员任务执行结果统计（A 套数据）

工号 EmpNo	是否具有替补资格	正机长 Captain	副机长 FirstOfficer	替补 Substitute	乘机 Deadhead	飞行次数	任务总次数（飞行+乘机）
A0001	N	2	0	-	2	2	4
A0002	N	2	0	-	2	2	4
A0003	N	1	0	-	1	1	2
A0004	N	1	0	-	1	1	2
A0005	Y	29	0	0	1	29	30
A0006	Y	29	1	1	0	30	30
A0007	Y	32	0	0	0	32	32
A0008	Y	31	0	0	1	31	32
A0009	Y	40	0	0	0	40	40
A0010	Y	38	0	0	0	38	38
A0011	N	1	0	-	1	1	2
A0012	N	0	20	-	0	20	20
A0013	N	0	22	-	0	22	22
A0014	N	0	22	-	0	22	22
A0015	N	0	22	-	0	22	22
A0016	N	0	17	-	1	17	18
A0017	N	0	21	-	1	21	22
A0018	N	0	22	-	0	22	22
A0019	N	0	17	-	1	17	18
A0020	N	0	23	-	1	23	24
A0021	N	0	19	-	1	19	20
总计	-	206	206	1	14	412	426

方案 2-B 中，机组人员的任务执行的总体情况如表 5-4，从结果来看：465 位机组人员都执行了任务，一共完成 28275 次任务，包括 27752 次飞行任务，50 次替补任务，与此同时产生 523 次乘机任务。

表 5-4　机组人员任务执行总体结果统计（B 套数据）

机组人员总数	执行任务的员工数	未执行任务的员工数	执行任务的员工人数占比	未执行任务的员工人数占比	飞行任务次数	乘机任务次数	替补次数	任务总次数
465	465	0	100.00%	0.00%	27752	523	50	28275

另外，我们还考虑了 124 位同时具有正副机长资格的员工任务执行情况，详见表 5-5。他们一共完成 12195 次任务，其中包含 12144 次飞行任务，占总任务次数的 99.58%，且在飞行任务中，执行替补 50 次，占飞行任务的 0.41%，以及 51 次乘机任务，占总任务次数的 0.42%，有理由认为结果满足了在机组排班过程中优先安排主要资格的目标。且容易得到一位员工最少执行 73 次任务，最多执行 116 次任务，人均执行约 98 次任务。

表 5-5 具有正副机长资格机组人员任务执行表（B 套数据）

具有替补资格的员工数	执行任务的员工数	飞行任务次数	飞行任务次数频率	乘机任务次数	乘机任务频率	替补次数	替补频率(替补/飞行任务次数)	任务总次数
124	124	12144	99.58%	51	0.42%	50	0.41%	12195

2）A 组具有副机长资格的机长执勤时间和飞行时间

在 A 组员工中，具有副机长资格的员工可能同时具有正机长资格，如图 5-3 所示，前面执勤时间高的员工同时拥有正副机长资格，相较只有副机长资格的员工，安排的执勤时间（4500min 左右）和飞行时间（3500min 左右）更长，但是执勤之间的休息时间也会相应安排的更长一些。

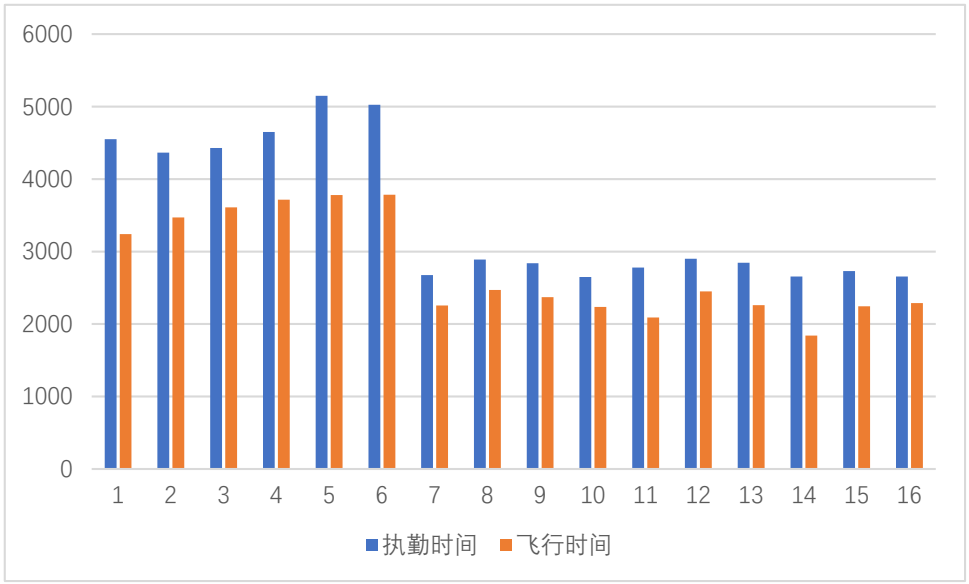

图 5-3　A 组机长执勤和飞行时间图

3）B 组具有双重机长资格的机长执勤时间和飞行时间

B 组员工人数更多，因此只分析具有双重机长几个的员工执勤和飞行时间。可以发现在 B 组中，机长执勤时间与员工所在基地位置相关。在基地 HOM 的机长中，双重机长资格的机长较少，因此基本都拥有较高执勤时间和飞行时间；而在基地 TGD 的机长中，双重机长资格的员工较多，因此会分成部分，一部分员工执勤和飞行时间较高，而另一部分则较低。但从总体而言，当机场员工飞行时间较高时，都会相应安排更多的休息时间。图 5-4 反映 B 组机长执勤和飞行时间情况。

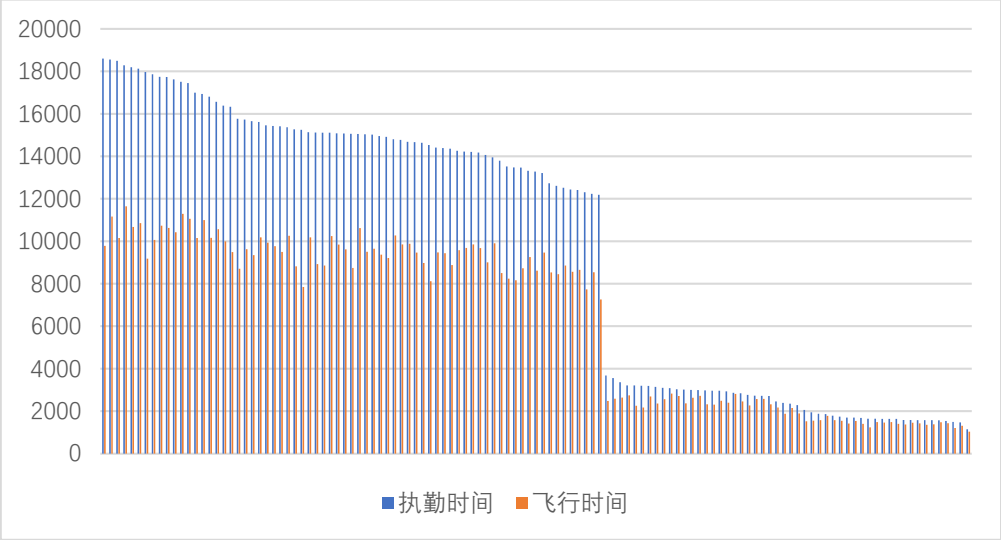

图 5-4　B 组机长执勤和飞行时间图

以机场人员 A0013 为例，在 8/11/2021-8/18/2021 的值班情况如图 5-5 所示。可以看出在 A 组员工值班期间，上班时间在早上八点到下午五点之间，每次执勤值一到两个班，在 A 组较为宽松的航班环境下，机场人员的总体值班时间不紧凑，上下班时间更加分散，而且在平时会出现休息整天的情况。

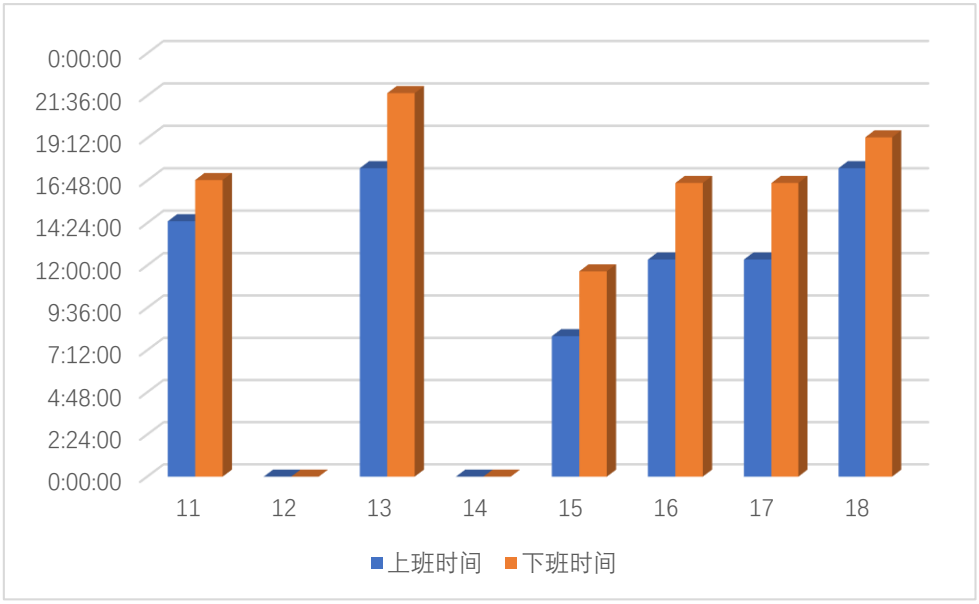

图 5-5　某 A 组机场员工一周上下班时间

以机场人员 B0139 为例，在 8/19/2021-8/26/2021 的值班情况如图 5-6 所示：可以看出在 B 组员工值班期间，上班时间多在早上六点到在早上八点之间，每次执勤在是个小时左右，在 B 组更为紧凑的航班环境下，机场人员的总体值班时间更多，上下班时间和平常人员时间相差不大，而且休息和工作时间平衡的会更好更合理。

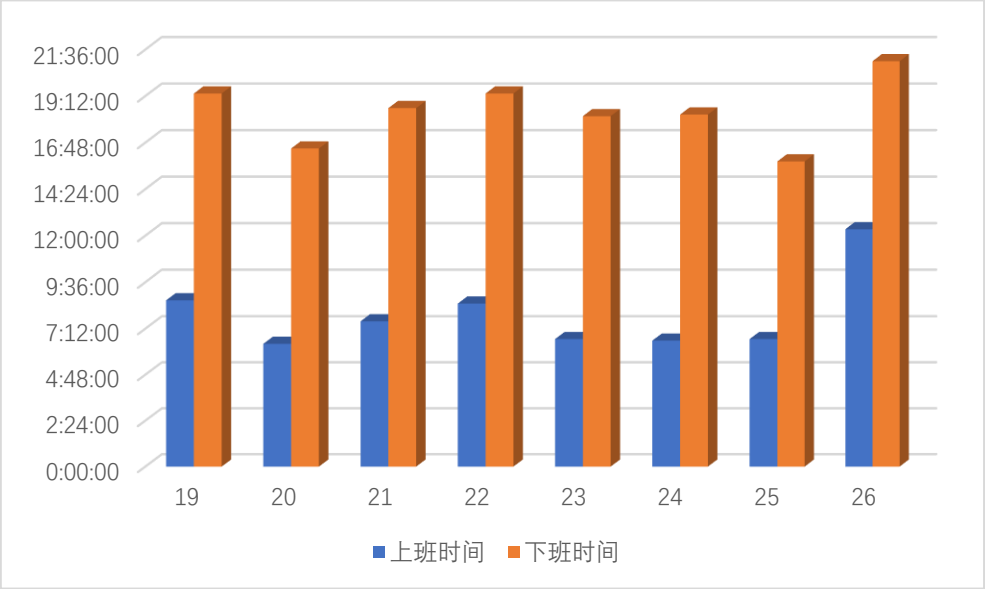

图 5-6　某 B 组机场员工一周上下班时间

4）机场航班起飞结果分析

与问题一相同， A 套数据机场航班全部满足，接下来只需要分析 B 套数据的机场航班情况。在 B 套数据中，有 12 个机场的航班全部能够按照计划起飞，其他机场的航班起飞比例比问题一有所下降，其中 XXJ 机场的起飞比例最低，为 91.30%。

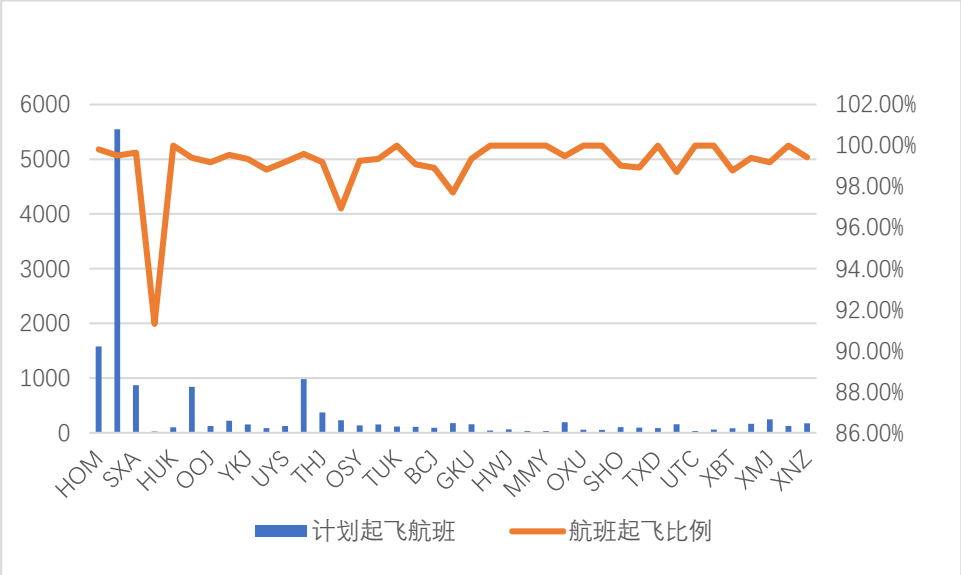

图 5-7　机场航班起飞

从数量上来看，见图 5-8，航班不能起飞的数量最多是 TGD 机场的 35 次，不能起飞的数量比问题一多 8 次。其余机场有少部分的航班控制在 5 次左右，大部分机场不能起飞的航班为 1 次或者 0 次。

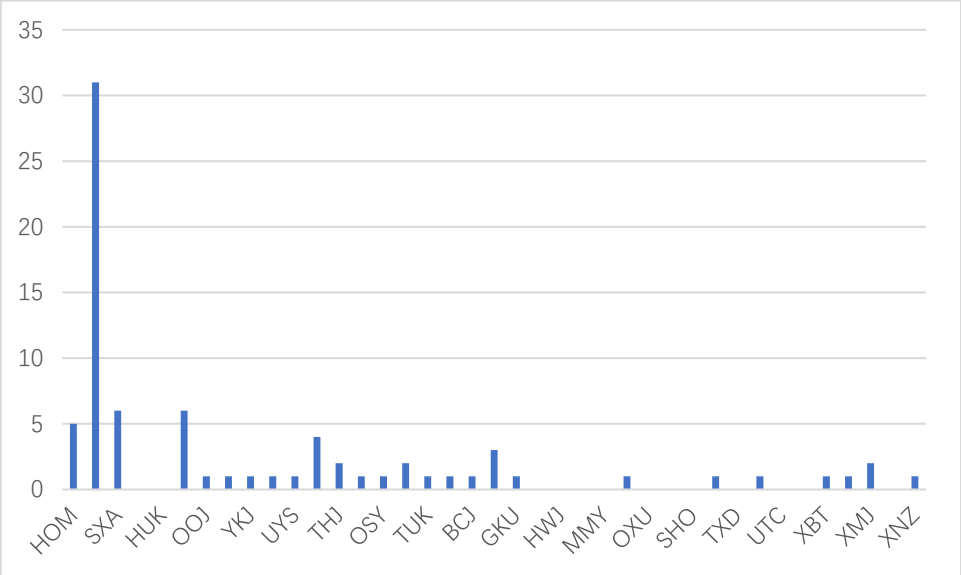

图 5-8　机场航班不能起飞数量

5）机场航班到达结果分析

在 B 套数据中，有 13 个机场的航班能按照计划到达，航班到达的比例基本控制在 98%以上，整体上比问题一的到达比例下降，其中 GKS 机场的到达比例与问题一的结果一样最低，为 98.54%。

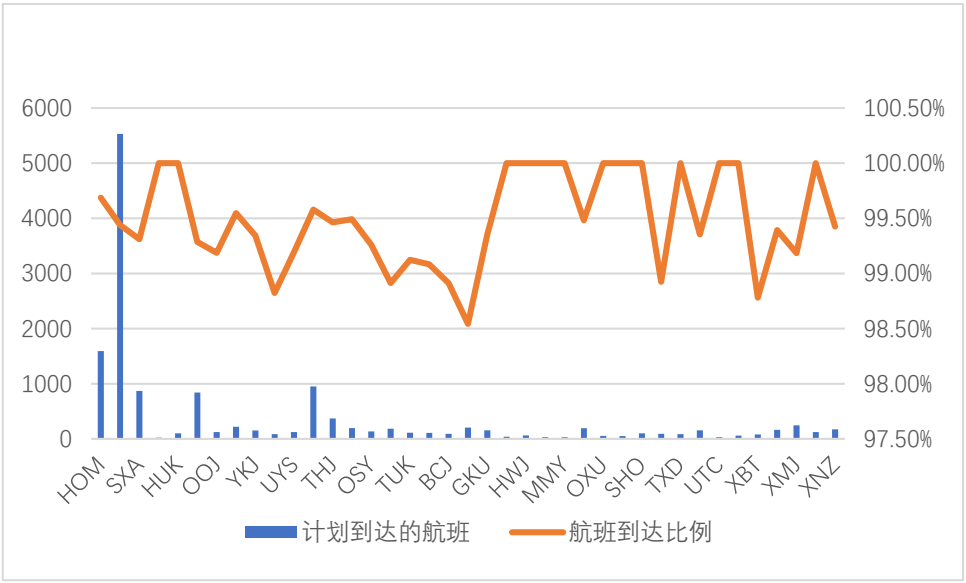

图 5-9　机场航班到达

从数量上来看，见图 5-10，大部分不能按计划到达的航班集中在一个机场---TGD，4 个机场不能到达的航班在 5 个左右，其他机场不能到达的航班数量基本控制在 4 次以下。

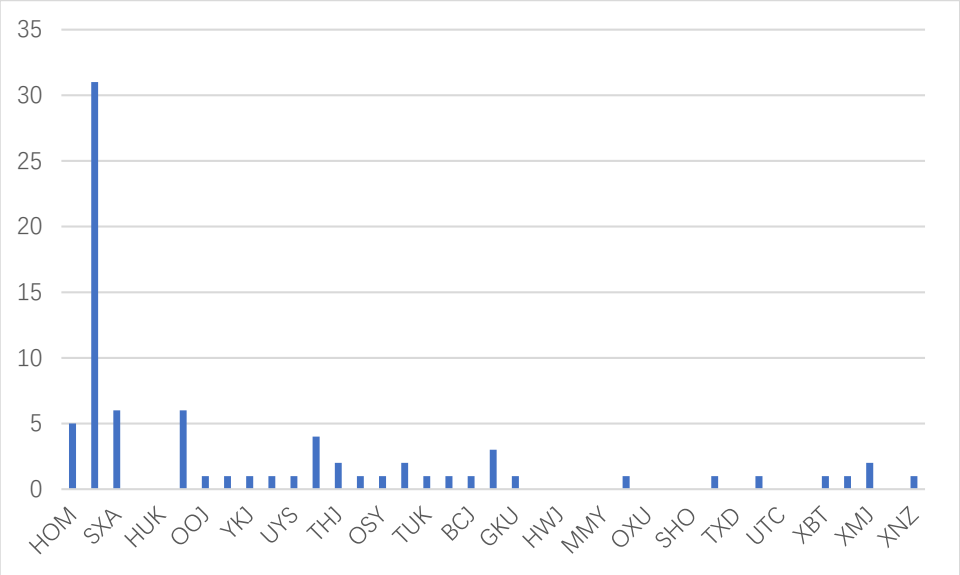

图 5-10　机场航班不能到达数量

6 问题三建模与求解

6.1 问题分析

问题三解决机组人员的任务环设计问题，新增两个优化目标：首先是尽可能降低机组人员的总体任务环成本，即在满足各个航班起飞条件的前提下，降低机组人员包含飞行时长，休息时长在内的任务时长；其次是各个机组人员之间的任务环时长尽可能平衡因此我们通过最小化机组人员任务环时长的极差来实现任务环时长平衡。

本文建模过程通过 01 变量表示机组人员在不同日期是否处于休假状态，把日期分为不同的任务环链。机组人员的任务环包括一系列的执勤过程。在任务环设计过程中需要考虑到成环约束，包括任务环总时长，连续执勤天数等，同时两个任务环之间还需要满足休假日期约束，最后需要保证任务环开始和结束时，机组人员需要在自己基地。

针对目标一，尽可能降低机组人员的总体任务环成本。本文通过计算机组人员在执行任务环时生成的成本，并进行相应优化处理，在保证所有航班起飞数量最大化的前提下，最小化总的任务环成本。

针对目标二，各个机组人员之间的任务环时长尽可能平衡。本文通过对不同员工的任务环时长进行计算和比较，以极差为评判标准对任务环设计进行优化，在首先满足最大的航班机组设置的基础上，使得机组人员的任务环时长尽可能平衡。

6.2 模型建立

1）新增目标函数：

目标一：机组人员的总体任务环成本最低

$$Max\sum_{k\epsilon K}\left(\bar{p}^{k}\sum_{l\in L}\boldsymbol{g}_{l}^{k}\right)$$

目标二：机组人员之间的任务环时长尽可能平衡

$$Min\left(Max\left(\sum_{l\in L}\boldsymbol{g}_{l}^{k}\right)-Min\left(\sum_{l\in L}\boldsymbol{g}_{l}^{k}\right)\right)$$

2）新增决策变量：

$$h_{nl}^{k}=\begin{cases}1，\text{员工}k\text{在第}n\text{天时处于第}l\text{个任务环上}\\0，\text{员工}k\text{在第}n\text{天时不处于第}l\text{个任务环上}\end{cases}$$

3）新增约束条件：

a）员工k的任务环是一段连续的时间：

$$\sum_{n\in N} h_{nl}^{k}\left(1-h_{n-1,l}^{k}\right)\le 1\ \ \forall l\in L,k\in K$$
$$\sum_{n\in N} h_{n-1,l}^{k}\left(1-h_{n,l}^{k}\right)\le 1\ \ \forall l\in L,k\in K \qquad (6-1)$$

b）员工k每天只在一个任务环上：

$$\sum_{l\in L} h_{nl}^{k}=1\ \ \forall n\in N,k\in K \qquad (6-2)$$

c）每个任务环里的第一次执勤航班的出发机场和最后一次执勤航班的达到机场如下：

$$d_l^k=\sum_{n\in N} h_{nl}^{k}\left(1-h_{n-1,l}^{k}\right)\left(\sum_{i\in I} x_{ij}^{k}b_{jn}(1-b_{in})d_j\right)$$
$$d_l^{k'}=\sum_{n\in N} h_{nl}^{k}\left(1-h_{n+1,l}^{k}\right)\left(\sum_{j\in I} x_{ij}^{k}b_{in}\left(1-b_{jn}\right)d_i'\right)\ ,\forall l\in L,k\in K \qquad (6-3)$$

d）每个任务环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：

$$\begin{cases} d_l^k(d_l^k-\overline{d^k})=0 \\ d_l^{k'}(d_l^{k'}-\overline{d^k})=0 \end{cases},\forall k\epsilon\ K\ ,l\in L \qquad (6-4)$$

e)机组人员在一个任务环的总时长表示如下：

$$g_l^k=\sum_{n\in N} h_{nl}^{k}\left(1-h_{n+1,l}^{k}\right)\left(\sum_{i\in I}\sum_{j\in I} x_{ij}^{k}b_{in}\left(1-b_{jn}\right)t_i'\right)$$
$$-\sum_{n\in N} h_{nl}^{k}\left(1-h_{n-1,l}^{k}\right)\left(\sum_{i\in I}\sum_{j\in I} x_{ij}^{k}b_{jn}(1-b_{in})t_j\right)\ ,\forall l\in L,k\in K \qquad (6-5)$$

f)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：

$$\sum_{l\in L} g_l^k\le MaxTAFB,\ \forall k\epsilon\ K \qquad (6-6)$$

g）每个机组人员相邻两个任务环之间至少有 MinVacDay 天休息：

$$\sum_{n\in N} h_{nl}^{k}\left(1-h_{n-1,l}^{k}\right)n-\sum_{n\in N} h_{n-1,l-1}^{k}\left(1-h_{n,l-1}^{k}\right)n$$
$$\ge MaxSuccOn\ \ \forall k\epsilon\ K\ ,l\epsilon\ L \qquad (6-7)$$

h）每个机组人员连续执勤天数不超过 MaxSuccOn 天：

$$\sum_{n\in N} u_{n-1}^{k}(1-u_n^k)n-\sum_{n\in N} u_n^k\left(1-u_{n-1}^{k}\right)n$$
$$\le MaxSuccOn\ \ \forall k\epsilon\ K\ ,l\epsilon\ L \qquad (6-8)$$

综上，**针对问题三建立的数学模型如下:**

目标① $Max\sum_{i\in I} a_i$

目标② $Max\sum_{k\epsilon K}(p^k\sum_{n\in N}m_n^k)$

目标③ $Max\sum_{k\epsilon K}\left(\bar{p}^k\sum_{l\in L}g_l^k\right)$

目标④ $Min\sum_{j\in I}\sum_{i\in I}\sum_{k\in K}x_{ij}^k r_j^k$

目标⑤ $Min\left(Max(\sum_{n\in N}m_n^k)-Min(\sum_{n\in N}m_n^k)\right)$

目标⑥ $Min\left(Max\left(\sum_{l\in L}g_l^k\right)-Min\left(\sum_{l\in L}g_l^k\right)\right)$

目标⑦ $Min\sum_{j\in I}\sum_{i\in I}\sum_{k\in K}x_{ij}^k f_j^k C^k$

$$
\text{s.t.}\begin{cases}
\sum_{i\epsilon I} x_{ij}^k = 1 \ \ \forall j\epsilon I, k \in K \\
\sum_{j\epsilon I} x_{ij}^k = 1 \ \ \forall i\epsilon I, k \in K \\
x_{ij}^k\left(a_j - 1\right) = 0 \quad \forall i, j\epsilon I, k \in K \\
x_{i0}^k\left(d_i' - \overline{d^k}\right) = 0 \ \ \forall i\epsilon I, k \in K \\
x_{0j}^k\left(d_j - \overline{d^k}\right) = 0 \ \ \forall j\epsilon I, k \in K \\
\left(d_i' - d_j\right)x_{ij}^k = 0 \ \ \forall i, j\epsilon I/\{0\}, k \in K \\
x_{ij}^k\left(t_i' + MinCT - t_j\right) \le 0 \ \ \forall i, j\epsilon I, k \in K \\
\sum_{i\epsilon I, k\epsilon K} x_{ij}^k \le 2 + MaxDH \ \ \forall j\epsilon I \\
c_i^k + f_i^k + r_i^k = 1 \ \ \forall i\epsilon I/\{0\}, k \in K \\
c_i^k(C^k - 1) = 0 \ \ \forall i\epsilon I, k \in K \\
f_i^k(F^k - 1) = 0 \ \ \forall i\epsilon I, k \in K \\
\sum_{i\epsilon I, k\epsilon K} x_{ij}^k c_j^k = 1 \ \ \forall j\epsilon I \\
\sum_{i\epsilon I, k\epsilon K} x_{ij}^k f_j^k = 1 \ \ \forall j\epsilon I \\
x_{ij}^k\left(e_j' - e_i'\right)\left(t_i' + MinRest - t_j\right) \le 0 \ \ \forall i, j\epsilon I, k \in K \\
\begin{cases}
(u_n^k - 1)\left(\sum_{j\epsilon I}\sum_{i\epsilon I} x_{ij}^k\, b_{jn}\right) \\
u_n^k - \sum_{j\epsilon I}\sum_{i\epsilon I} x_{ij}^k\, b_{jn} \le 0
\end{cases}, \ \forall k\epsilon \ K \ , n\epsilon \ N \\
u_n^k \sum_{j\epsilon I}\sum_{i\epsilon I}\left(x_{ij}^k b_{jn}\left(t_j' - t_j\right)\left(c_j^k + f_j^k\right)\right) \le MaxBlk, \ \forall k\epsilon \ K \ , n\epsilon \ N \\
m_n^k = u_n^k \sum_{j\epsilon I}\sum_{i\epsilon I}\left(x_{ij}^k b_{jn}\left(t_j' - t_j + b_{in}\left(t_j - t_i'\right)\right)\right), \ \forall k\epsilon \ K \ , n\epsilon \ N \\
m_n^k \le MaxDP, \ \forall k\epsilon \ K \ , n\epsilon \ N \\
\sum_{n\epsilon N} h_{nl}^k\left(1 - h_{n-1,l}^k\right) \le 1 \ \ \forall l \in L, k \in K \\
\sum_{n\epsilon N} h_{n-1,l}^k\left(1 - h_{n,l}^k\right) \le 1 \ \ \forall l \in L, k \in K \\
\sum_{l\epsilon L} h_{nl}^k = 1 \ \ \forall n \in N, k \in K \\
d_{lf}^k = \sum_{n\epsilon N} h_{nl}^k(1 - h_{n-1,l}^k)\left(\sum_{i\epsilon I} x_{ij}^k b_{jn}(1 - b_{in})d_j\right) \ \ \forall k\epsilon \ K \ , l \in L \\
d_{lf}^{k\prime} = \sum_{n\epsilon N} h_{nl}^k\left(1 - h_{n+1,l}^k\right)\left(\sum_{j\epsilon I} x_{ij}^k b_{in}\left(1 - b_{jn}\right)d_i'\right)\forall k\epsilon \ K \ , l \in L \\
d_{lf}^k\left(d_{lf}^k - \overline{d^k}\right) = 0 \ \ \forall k\epsilon \ K \ , l \in L \\
d_{lf}^{k'}\left(d_{lf}^{k'} - \overline{d^k}\right) = 0 \ \ \forall k\epsilon \ K \ , l \in L \\
g_l^k = \sum_{n\epsilon N} h_{nl}^k\left(1 - h_{n+1,l}^k\right)\left(\sum_{i\epsilon I}\sum_{j\epsilon I} x_{ij}^k b_{in}\left(1 - b_{jn}\right)t_i'\right) \\
- \sum_{n\epsilon N} h_{nl}^k\left(1 - h_{n-1,l}^k\right)\left(\sum_{i\epsilon I}\sum_{j\epsilon I} x_{ij}^k b_{jn}(1 - b_{in})t_j\right) \ \ \forall k\epsilon \ K \ , l \in L \\
\sum_{l\epsilon L} g_l^k \le MaxTAFB, \ \forall k\epsilon \ K \\
\sum_{n\epsilon N} h_{nl}^k\left(1 - h_{n-1,l}^k\right)n - \sum_{n\epsilon N} h_{n-1,l-1}^k\left(1 - h_{n,l-1}^k\right)n \ge MaxSuccOn \ \ \forall k\epsilon \ K \ , l\epsilon \ L \\
\sum_{n\epsilon N} u_{n-1}^k(1 - u_n^k)n - \sum_{n\epsilon N} u_n^k(1 - u_{n-1}^k)n \le MaxSuccOn \ \ \forall k\epsilon \ K \ , l\epsilon \ L
\end{cases}
$$

6.3 算法求解

模型建立完成后，采用构造法完成员工与航班的匹配，设计启发式算法对该问题进行求解，在满足约束集的条件下尽量满足目标函数。

算法步骤如下：

Step1：初始化航班和员工并将航班排序。按照航班的出发时间和到达时间从小到大排序；同时对航班的合理性进行判断，剔除不合理的航班（不合理的航班包括：①正副机长无法在航班起飞前通过任何途径到达；②正副机长完成航班后在一个排班周期内无法返回基地）。

Step2：判断当前的时间节点，若开始安排新一天的航班，则说明进入下一天，需更新飞行员的执勤状态，并根据飞行员已安排的航班，考虑是否休假或者强制休息。

Step3：给航班分配副机长。先遍历只具有副机长资格的员工，再遍历同时具有正副机长资格的员工。若没有满足执勤、飞行时间约束、任务环约束以及地点约束的副机长，则进入 Step5；分配成功进入 Step4。

Step4：给航班分配正机长。先遍历同时具有正副机长资格的员工，再遍历只有正机长资格的员工。若没有满足执勤、飞行时间约束、任务环约束以及地点约束的正机长，则进入 Step5；若分配成功，则进入 Step2，继续分配下一班航班，否则进入 Step6。

Step5：在更早的航班中加入搭乘人员，执行乘机任务到达指定机场。若找到对应职务员工，进入 Step3；否则寻找失败，将当前航班剔除，进入 Step2，继续分配下一班航班。

Step6：员工返回基地。遍历所有员工所在位置，判断是否在基地，搭乘最晚航班返回基地，若不能没有返回航班，则取消相应员工上次回到基地之前的航班，并重新搭乘其他航班返回基地。

Step7：所有航班匹配完成，算法结束。

6.4 结果分析

利用以上的算法分别运行 A 套数据和 B 套数据，对应得出最优的航班机组人员分配方案，以下称为方案 3-A 和方案 3-B。表 6-1 给出了关键指标统计结果。

表 6-1 关键指标结果统计表 1-1

数据 \ 指标	不满足机组配置航班数	满足机组配置航班数	机组人员总体乘机次数	替补资格使用次数	机组总体利用率	最小/最大/平均执勤飞行时长（时）		
A 套数据	60	146	12	0	78.85%	75	520	315
B 套数据	9067	4818	1820	0	68.30%	45	555	280

表 6-1 关键指标结果统计表 1-2

数据 \ 指标	最小/最大/平均执勤执勤时长（时）			最小/最大/平均机组人员执勤天数			一/二/三/四天任务环数量分布	总体执勤成本（万元）	总体任务环成本（万元）	程序运行时间（分）
A 套数据	75	695	399	1	8	4	0/32/9/1	6.934	39.48	0.2
B 套数据	45	720	411	0	11	7	0/339/205/109	1351.248	200.4580	31.573

从计算结果可以看出，在 A 套数据中，146 趟航班满足机组配置，航班机组配置成功率为 70.87%，其中机组人员总体乘机次数为 12 次，替补 0 次，机组总体利用率达 78.85%，执勤的最小、最大和平均飞行时长分别为 75h、520h、315h，执勤的最小、最大和平均执勤时长分别为 75h、695h、399h，执勤的最小、最大和平均执勤天数分别为 1 天、8 天、4 天，第一、二、三、四天的任务环数量分别是 0、32、9、1 个，一共产生 6.934 万元的执勤成本和 39.48 万元的任务环成本。

在 B 套数据中，9067 趟航班满足机组配置，航班机组配置成功率为 65.30%，其中机组人员总体乘机次数为 1820 次，替补 0 次，机组总体利用率达 68.30%，执勤的最小、最大和平均飞行时长分别为 75h、520h、315h，执勤的最小、最大和平均飞行时长分别为 45h、555h、280h，执勤的最小、最大和平均执勤时长分别为 45h、720h、411h，执勤的最小、最大和平均执勤天数分别为 0 天、11 天、7 天，第一、二、三、四天的任务环数量分别是 0、339、205、109 个，一共产生 1351.248 万元的执勤成本和 200.4580 万元的任务环成本。

1）机组人员分配结果分析

方案 3-A 中，机组人员的任务执行的总体情况如表 6-2，从结果来看：21 位机组人员都执行任务，一共完成 426 次任务，包括 292 次飞行任务，0 次替补任务，与此同时产生 12 次乘机任务。

表 6-2　机组人员任务执行总体结果统计（A 套数据）

机组人员总数	执行任务的员工数	未执行任务的员工数	执行任务的员工人数占比	未执行任务的员工人数占比	飞行任务次数	替补次数	乘机任务次数	任务总次数
21	21	0	100.00%	0.00%	292	0	12	426

机组人员的任务执行的具体情况如表 6-3 所示，一位员工最少执行 2 次任务，最多执行 25 次任务，人均执行约 14 次任务。

表 6-3　机组人员任务执行结果统计（A 套数据）

工号 EmpNo	是否具有替补资格	正机长 Captain	副机长 FirstOfficer	替补 Substitute	乘机 Deadhead	飞行次数	任务总次数（飞行+乘机）
A0001	N	1	0	-	1	1	2
A0002	N	1	0	-	1	1	2
A0003	N	3	0	-	2	3	5
A0004	N	1	0	-	1	1	2
A0005	Y	24	0	0	0	24	24
A0006	Y	25	0	0	0	25	25
A0007	Y	25	0	0	0	25	25
A0008	Y	22	0	0	0	22	22
A0009	Y	23	0	0	0	23	23
A0010	Y	16	0	0	0	16	16
A0011	N	5	0	-	1	5	6
A0012	N	0	5	-	0	5	5
A0013	N	0	24	-	0	24	24
A0014	N	0	19	-	1	19	20
A0015	N	0	17	-	0	17	17
A0016	N	0	12	-	0	12	12
A0017	N	0	23	-	1	23	24
A0018	N	0	20	-	1	20	21
A0019	N	0	1	-	1	1	2
A0020	N	0	15	-	1	15	16
A0021	N	0	10	-	1	10	11
总计	-	146	146	0	12	292	304

方案 3-B 中，机组人员的任务执行的总体情况如表 6-4，从结果来看：465 位机组人员都执行了任务，一共完成 28275 次任务，包括 27752 次飞行任务，50 次替补任务，与此同时产生 523 次乘机任务。且容易得到一位员工最少执行 1 次任务，最多执行 50 次任务，人均执行约 22 次任务。

表 6-4　机组人员任务执行总体结果统计（B 套数据）

机组人员总数	执行任务的员工数	未执行任务的员工数	执行任务的员工人数占比	未执行任务的员工人数占比	飞行任务次数	乘机任务次数	替补次数	任务总次数
465	463	2	99.57%	0.43%	27752	523	12	28275

另外，我们还考虑了 124 位同时具有正副机长资格的员工任务执行情况，具体统计情况见表 6-5。他们一共完成 3935 次任务，其中包含 3887 次飞行任务，占总任务次数的 98.78%，且在飞行任务中，执行了替补 12 次，占飞行任务的 0.31%；以及 48 次乘机任务，占总任务次数的 1.22%，有理由认为结果满足了在

机组排班过程中优先安排主要资格的目标。且容易得到一位员工最少执行 14 次任务，最多执行 50 次任务，人均执行约 32 次任务。

表 6-5　具有正副机长资格机组人员任务执行表（B 套数据）

具有替补资格的员工数	执行任务的员工数	飞行任务次数	飞行任务次数频率	乘机任务次数	乘机任务频率	替补次数	替补频率（替补/飞行任务次数）	任务总次数
124	124	3887	98.78%	48	1.22%	12	0.31%	3935

2）机长执勤时间和飞行时间结果分析

1.　A 组具有副机长资格的机长执勤时间和飞行时间

在 A 组员工中，具有副机长资格的员工可能同时具有正机长资格，如图 6-1 所示，前面执勤时间高的员工同时拥有正副机长资格，相较只有副机长资格的员工，安排的执勤时间（3000min 左右）和飞行时间（2500min 左右）更长，但是执勤之间的休息时间也会相应安排的更长一些。

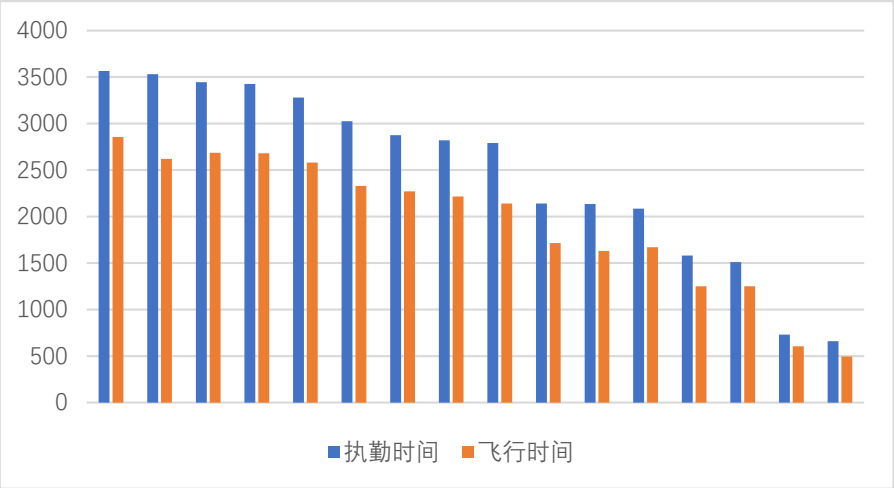

图 6-1　A 组具有副机长资格的机长执勤与飞行时间图

2.　B 组具有双重机长资格的机长执勤时间和飞行时间

B 组员工人数更多，在此只分析具有双重机长几个的员工执勤和飞行时间作简单说明，结果见图 6-2。可以发现在 B 组中，机长执勤时间与员工所在基地位置相关。这一点和问题二中结论相同。但是在问题三中出现新的现象，机长执勤时间分段效果不明显。在连续执勤时间约束下，不同能力和基地的员工最终的执勤时间差距缩小。

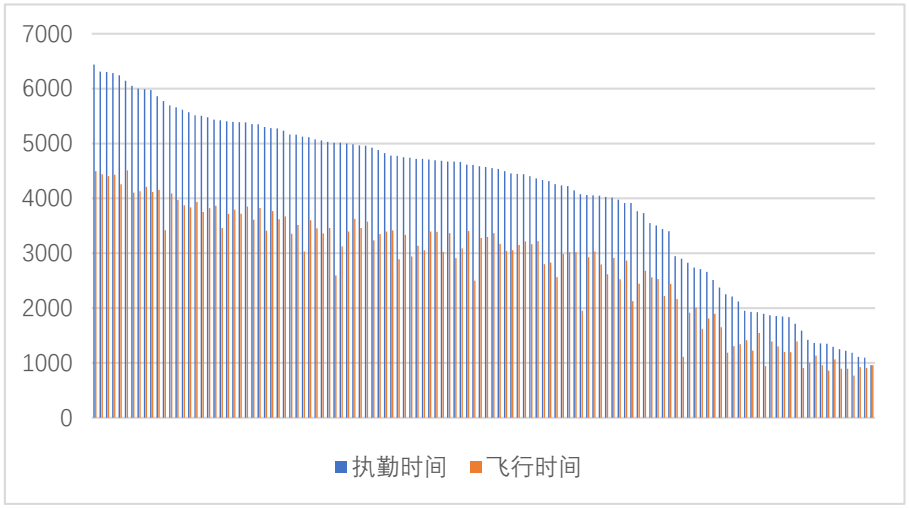

图 6-2　B 组具有双重机长资格的机长执勤与飞行时间图

3）机长任务环时间结果分析

1.　A 组具有副机长资格的机长任务环时间

A 组具有副机长资格的机长任务环时间分布如图 6-3 所示，可以发现在问题三的优化过程中，不同机长的总任务环时长相差不大，总体维持在 12000min 左右。即使员工同时拥有正机长资格，总的任务时长不会因此明显增加或减少。

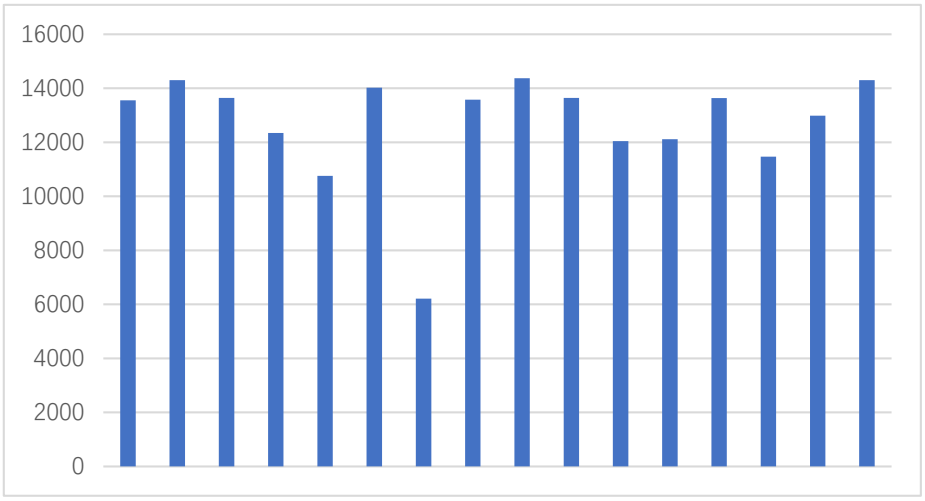

图 6-3　A 组具有副机长资格的机长任务环时间分布图

2.　B 组具有副机长资格的机长任务环时间

B 组具有副机长资格的机长任务环时间分布如图 6-4 所示，可以发现在问题三的优化过程中，不同机长的总任务环时长相差不大，总体维持在 13000min 左右，和 A 组相比任务时长相应增加。主要原因在于航班数量较多，每个机组人员需要耗费更多时间进行正常工作，并因此很难出现任务环时长极高或极低的情形。

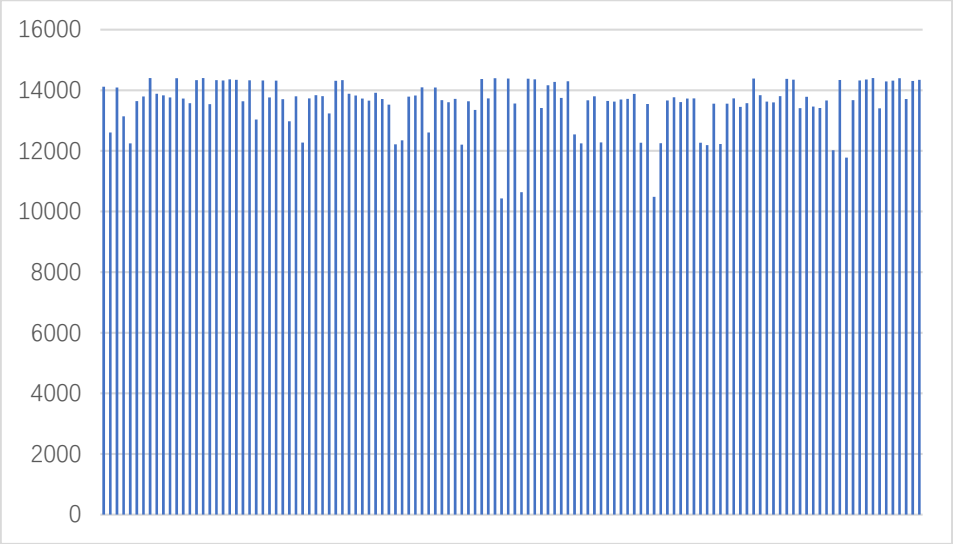

图 6-4　B 组具有副机长资格的机长任务环时间分布图

4）机场航班起飞和到达结果分析

与问题一和二不同的是，问题三中 A 套数据和 B 套数据均有航班不能起飞的情况，为了方便与问题一和二对比，我们接下来只分析 B 套数据的结果，A 套数据具体见附件。在 B 套数据中，机场的航班起飞和到达平均占比 35%左右，也就是说，在整个排班计划中，由于需要考虑员工的休假情况，会使得航班的起飞比例大幅度下降。

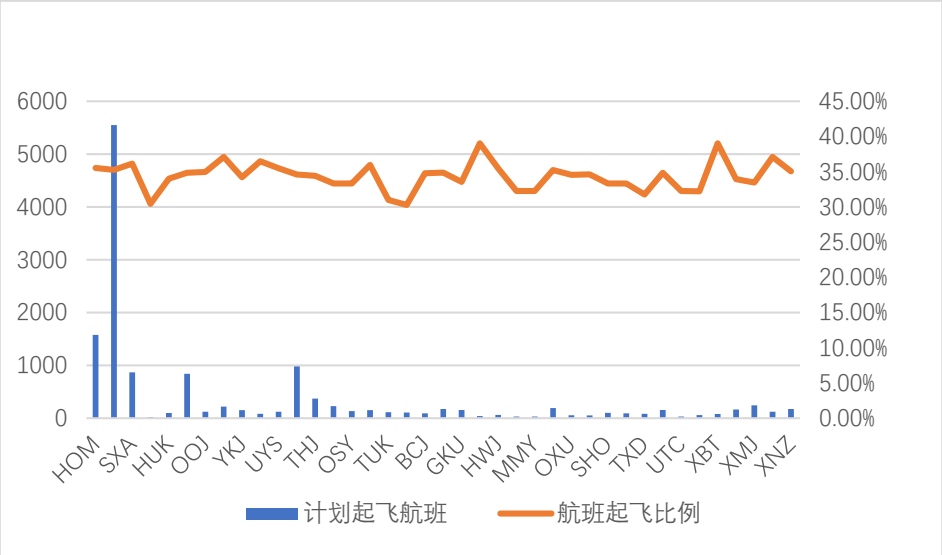

图 6-5　机场航班起飞

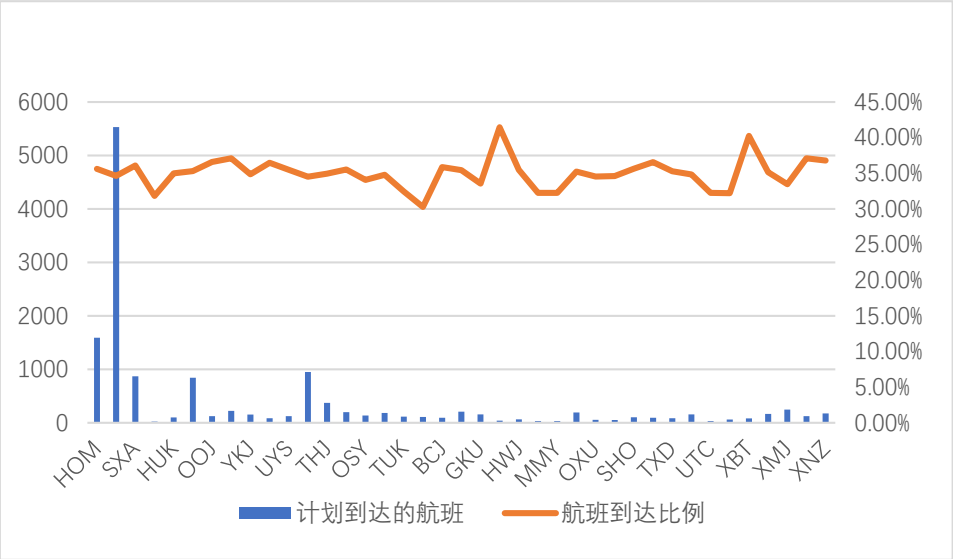

图 6-6　机场航班到达

在图 6-7 和 6-8 中，航班不能起飞和到达的数量最多是 TGD 机场，大幅度超过了问题一和二的数量。其他 HOM 机场不能起飞和到达航班的数量为 1000 次，SXA，FBX 和 NOU 机场不能起飞和到达的次数在 500 次左右，大部分机场不能起飞和到达的航班在 100 次及以下。

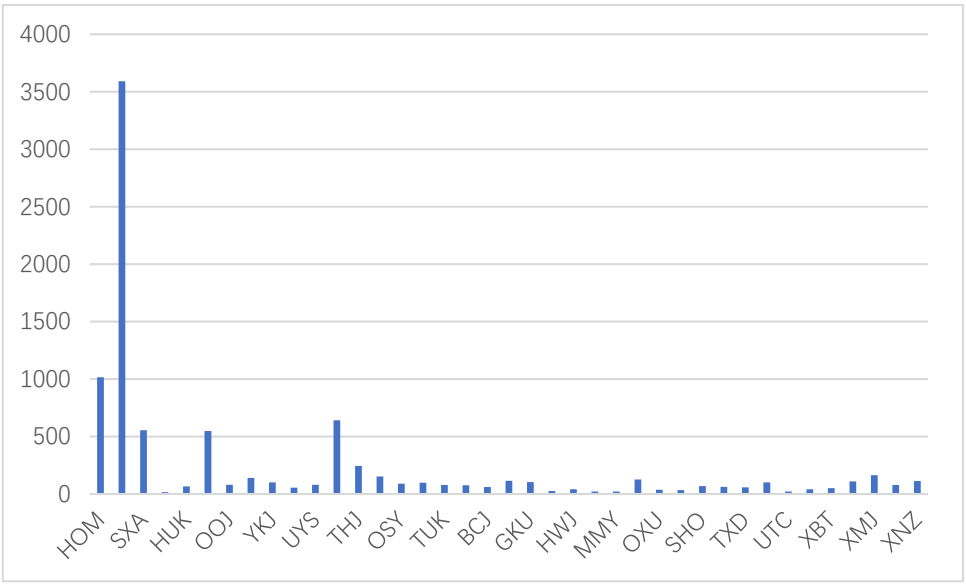

图 6-7　机场航班不能起飞数量

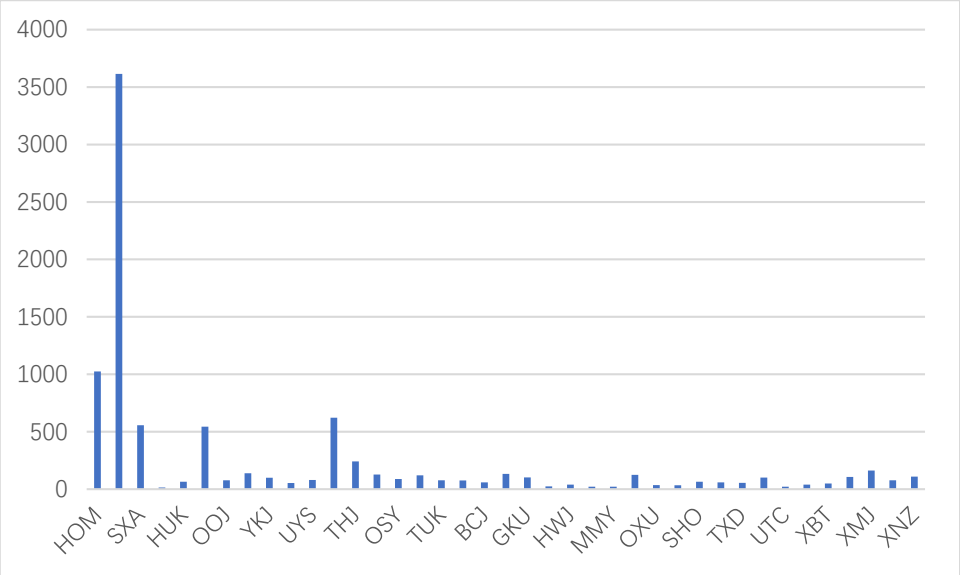
4000
3500
3000
2500
2000
1500
1000
500
0
HOM
SXA
HUK
OOJ
YKJ
UYS
THJ
OSY
TUK
BCJ
GKU
HWJ
MMY
OXU
SHO
TXD
UTC
XBT
XMJ
XNZ

图 6-8　机场航班到达

7 总结

本文考虑三个场景下的航班机组优化排班问题，并评估机组人员休假对航班排班的影响。在研究问题中，问题一是对机组人员与航班的匹配分配问题，问题二是考虑执勤因素对机组人员与航班匹配的影响；由于每个机组人员每天只能进行一次执勤，因此问题二是在问题一上进行航班环的切割；问题三加入任务环的因素，在问题二的基础上再加一层，考虑周期性的排班问题。因此，问题一与二是并列关系，而问题三是递进关系。

在这三个问题中，问题一只需要考虑航班与人员的匹配问题，目标是尽可能多的航班满足机组配置，然后尽可能少的总体乘机次数，最后尽可能少使用替补资格。问题二加入执勤因素，在问题一上以天为单位进行切割，在考虑问题一目标的基础上，还需要考虑机组人员的总体执勤成本最低和机组人员之间的执勤时长尽可能平衡。问题三考虑周期性排班，在前两个问题的基础上考虑机组人员的总体任务环成本最低和机组人员之间的任务环时长尽可能平衡。

本文针对该航班与机组人员的匹配问题建立了多目标下的整数规划模型，并采用构造法和领域搜索算法对 A、B 两套数据进行求解。针对问题一，运行 A 套数据成功分配 13 位机组人员执行任务（乘机任务 8 次，替补 0 次），满足 206 趟航班的最低机组配置；运行 B 套数据成功分配 360 位机组人员执行任务（乘机任务 374 次，替补 0 次），13885 趟航班中有 13850 趟航班满足最低机组配置，航班机组配置成功率达 99.75%。

针对问题二，考虑执勤因素的影响，运行 A 套数据，成功分配 21 位机组人员执行任务（乘机任务 14 次，替补 0 次），206 趟航班全部满足最低机组配置，机组总体利用率达 79.64%，执勤的最小、最大和平均飞行时长分别为 75h、490h、219h，执勤的最小、最大和平均执勤时长分别为 75h、705h、275h，执勤的最小、最大和平均执勤天数分别为 2 天、15 天、9 天，一共产生 53.344 万元的执勤成本；运行 B 套数据，成功分配 465 位机组人员执行任务（乘机任务 524 次，替补 0 次），13885 趟航班中有 13807 趟航班满足最低机组配置，航班机组配置成功率达 99.44%，机组总体利用率达 63.02%，执勤的最小、最大和平均飞行时长分别为 45h、560h、236h，执勤的最小、最大和平均执勤时长分别为 45h、709h、375h，执勤的最小、最大和平均执勤天数分别为 8 天、31 天、24 天，一共产生 4110.192 万元的执勤成本。

针对问题三，考虑周期性排班，运行 A 套数据，成功分配 21 位机组人员执行任务（乘机任务 12 次，替补 0 次），206 趟航班中有 146 趟航班满足最低机组配置，航班机组配置成功率为 70.87%，机组总体利用率达 78.85%，执勤的最小、最大和平均飞行时长分别为 75h、520h、315h，执勤的最小、最大和平均执勤时长分别为 75h、695h、399h，执勤的最小、最大和平均执勤天数分别为 1 天、8 天、4 天，第一、二、三、四天的任务环数量分别是 0、32、9、1 个，一共产生 6.934 万元的执勤成本和 39.48 万元的任务环成本；运行 B 套数据，成功分配 463 位机组人员执行任务（乘机任务 1820 次，替补 0 次），13885 趟航班中有 9067 趟航班满足最低机组配置，航班机组配置成功率达 65.30%，机组总体利用率达 68.30%，执勤的最小、最大和平均飞行时长分别为 45h、555h、280h，执勤的最

小、最大和平均执勤时长分别为 45h、720h、411h，执勤的最小、最大和平均执勤天数分别为 0 天、11 天、7 天，第一、二、三、四天的任务环数量分别是 0、339、205、109 个，一共产生 1351.248 万元的执勤成本和 200.4580 万元的任务环成本。

本文建立的数学模型对现实中不同情境下的机组排班问题有很强的参考意义。由于本问题考虑的约束比较多，航班情况比较复杂，模型构建存在一些可以改进的地方，比如模型没有考虑其他机场出发的机组人员情况。本文提出构造法和领域搜索算法，先后运行 A、B 两套数据，求解得到比较满意的局部最优解，但考虑到算法时间和复杂度的限制，本文求解算法在实际中仍具有一定的指导意义。

参考文献

[1] Frédéric Quesnel, Guy Desaulniers, François Soumis (2019) Improving Air Crew Rostering by Considering Crew Preferences in the Crew Pairing Problem. Transportation Science, Published online in Articles in Advance 18 Oct 2019.

[2] Vahid Zeighami, François Soumis (2019) Combining Benders' Decomposition and Column Generation for Integrated Crew, Pairing and Personalized Crew Assignment Problems. Transportation Science, Published online in Articles in Advance 02 Aug 2019. https://doi.org/10.1287/trsc.2019.0892

[3] Mohamed Haouari, Farah Zeghal Mansour, Hanif D. Sherali (2019) A New Compact Frmulation Fr the Daily Crew Pairing Problem. Transportation Science 53(3): 811-828.

[4] Shuo Liu, Wenhua Chen, Jiyin Liu, Optimizing airport gate assignment with operational safety constraints, 2014 20th International Conference on Automation and Computing.

[5] Saeed Saemi, Alireza Rashidi Komijan, Reza Tavakkoli-Moghaddam and Mohammad Fallah, A new mathematical model to cover crew pairing and rostering problems simultaneously, Journal of Engg. Research Vol. 9 No. (2) June 2021 pp. 218-233.

[6] Xiaodong Luo, Yogesh Dashora, Tina Shaw (2015) Airline Crew Augmentation: Decades of Improvements from Sabre. Interfaces 45(5): 409-424.

附　录

核心代码、数据结构及说明：

```
struct Crew {
    int Degree[3];//机组人员的飞行资格，0表示乘机资格、1表示副机长资格、2表示正机长资格
    int Base;//机组人员的基地
    int DutyCostPerHr;//每小时执勤成本
    int ParingCostPerHr;//每小时任务环成本
    int CurrentStn;//完成当前航班后所在机场
    int lastFlight;//刚完成的航班，初始为-1，后续为个航班号
    int CurrentStatus;//当前工作状态，0表示休息状态, 1表示工作中状态，2表示休假状态
    int CurrentDutyTime;//当前已执勤时长
    int CurrentFlyTime;//当前已飞行时长
    int CurrentPairingDay;//当前已连续安排任务环的时长，用于判断是否优先安排休假
    int TotalDutyTime; //总执勤时长
    int TotalPairingTime; //总任务环时间
    int ConDutyDay;//连续执勤天数
    int ConRestDay;//连续休息时长
};
Crew CrewPara[CrewNum];//机组人员属性
struct Flight {
    int FltNum; //航班号
    int DptrDate;//航班起飞日期
    int DptrTime;//航班起飞时间
    int ArrvDate;//航班到达日期
    int ArrvTime;//航班到达时间
    int DptrStn;//航班出发机场
    int ArrvStn;//航班到达机场
    int RelaDptrTime;//航班相对出发时间（以第一班航班为起始时间进行转化）
    int RelaArrvTime;//航班相对到达时间（以第一班航班为起始时间进行转化）
    int LegTime;//航班飞行时长
    int isFly; //航班是否确定起飞
};
Flight FlightPara[FlightNum];//航班属性
int FlightCrewNum[FlightNum][3];//某一航班的搭乘数量，0表示乘机、1表示副机长、2表示正机长
int StnCrewNum[StnNum]; //初始化人员分布，用于确定哪些机场是基地
int StnFlow[StnNum][StnNum]; //机场之间的航班流量
int FlightCanFly[FlightNum];//飞机在初始化时是否一定被剔除
int FlightBackP[FlightNum][2];//航班优先分给哪个基地的机组人员(最多两个基地)
int CrewDayStatus[CrewNum][32];//机组人员每一天的状态（最多为31天）
vector<vector<int>> CrewFlight(CrewNum); //机组人员分配的航班链
```

```
vector<vector<int>> CrewDegree(CrewNum); //机组人员在该航班的身份，0表示乘机、1表示副机长、2表示正机长，与CrewFlight对应
vector<vector<int>> CrewStatus(CrewNum);// 机组人员完成该航班之后的状态，0表示休息、 1表示继续工作、2表示休假
vector<vector<int>> CrewConDutyDay(CrewNum);//记录上一阶段的连续执勤时间，用于回溯
vector<vector<int>> CrewConPairingDay(CrewNum);//记录上一阶段的连续任务环时间，用于回溯
vector<int> Captain; //只有正机长资格的机组人员集合
vector<int> FirstOfficer; //只有副机长资格的机组人员集合
vector<int> CaptainFirstOfficer; //同时拥有正机长和副机长资格的机组人员集合
```

问题一：

```
// Problem1.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <time.h>
#include <ctime>
#include <string>
#include <string.h>
#include <algorithm>
#include <vector>

using namespace std;
#define MAX_VAL 999999

clock_t startTime, endTime;//记录运行时间
double bestTime;//最佳时间
double cutting_time = 1800;//终止条件
string inFile_A = "F:/华为比赛/华为建模比赛/2021 年 F 题/机组排班 Data A-Crew.csv";
string inFile_B = "F:/华为比赛/华为建模比赛/2021 年 F 题/机组排班 Data A-Flight.csv";
string UncoveredFlights = "F:/华为比赛/华为建模比赛/2021 年 F 题/UncoveredFlights_A.csv";
string CrewRosters = "F:/华为比赛/华为建模比赛/2021 年 F 题/CrewRosters_A.csv";
const int MinCT = 40;

//数据 A
const int startDate = 11;
const int FlightNum = 206;//航班数量
const int CrewNum = 21;//机组人员
```

```
const int StnNum = 7;
//数据 B
//const int startDate = 1;
//const int FlightNum = 13954;//航班数量
//const int CrewNum = 465;//机组人员
//const int StnNum = 39;
int FlightCrewNum[FlightNum][3];//0, 1, 2,1 表示副机长，2 表示机长
int FlightCanFly[FlightNum];//飞机是否能起飞
int StnCrewNum[StnNum];
int FlightCNum[FlightNum];//飞机数量
int StnFlow[StnNum][StnNum];
int FlightBackP[FlightNum][2];//回来的可能性
vector<vector<int>> CrewFlight(CrewNum);
vector<vector<int>> CrewDegree(CrewNum);
vector<int> Captain;
vector<int> FirstOfficer;
vector<int> CaptainFirstOfficer;

struct Crew {
	int Degree[3];//等级
	int Base;//出发基地
	int DutyCostPerHr;//每小时执勤成本
	int ParingCostPerHr;//任务环成本
	int CurrentStn;//当前所在机场
	int lastFlight;//上一航班
};
Crew CrewPara[CrewNum];//机组人员属性

struct Flight{
	int FltNum;
	int DptrDate;//航班起飞日期
	int DptrTime;//航班起飞时间
	int ArrvDate;//航班到达日期
	int ArrvTime;//航班到达时间
	int DptrStn;//航班出发机场
	int ArrvStn;//航班到达机场
	int RelaDptrTime;//航班相对出发时间
	int RelaArrvTime;//航班相对到达时间
	int LegTime;//航班持续时长
	int isFly;
	//最低驾驶配置都是 C1F1
};
Flight FlightPara[FlightNum];//航班属性
```

```
vector<vector<string>> read_csv(string file) {
	int i, j;
	ifstream inFile(file, ios::in);
	string lineStr;
	vector<vector<string>> strArray;
	while (getline(inFile, lineStr))
	{
		// 存成二维表结构
		stringstream ss(lineStr);
		string str;
		vector<string> lineArray;
		// 按照逗号分隔
		while (getline(ss, str, ','))
			lineArray.push_back(str);
		strArray.push_back(lineArray);
	}
	return strArray;
}

void split(const string& s, vector<int>& sv, const char flag = ' ') {
	sv.clear();
	istringstream iss(s);
	string temp;

	while (getline(iss, temp, flag)) {
		sv.push_back(stoi(temp));
	}
	return;
}

void exchangeFlight(int f1, int f2) {
	int DptrDate = FlightPara[f1].DptrDate;//航班起飞日期
	int DptrTime = FlightPara[f1].DptrTime;//航班起飞时间
	int ArrvDate = FlightPara[f1].ArrvDate;//航班到达日期
	int ArrvTime = FlightPara[f1].ArrvTime;//航班到达时间
	int DptrStn = FlightPara[f1].DptrStn;//航班出发机场
	int ArrvStn = FlightPara[f1].ArrvStn;//航班到达机场
	int RelaDptrTime = FlightPara[f1].RelaDptrTime;//航班相对出发时间
	int RelaArrvTime = FlightPara[f1].RelaArrvTime;//航班相对到达时间
	int LegTime = FlightPara[f1].LegTime;//航班持续时长
	int FltNum = FlightPara[f1].FltNum;
	//f1->f2
	FlightPara[f1].DptrDate = FlightPara[f2].DptrDate;
	FlightPara[f1].DptrTime = FlightPara[f2].DptrTime;
```

```
        FlightPara[f1].ArrvDate = FlightPara[f2].ArrvDate;
        FlightPara[f1].ArrvTime = FlightPara[f2].ArrvTime;
        FlightPara[f1].DptrStn = FlightPara[f2].DptrStn;
        FlightPara[f1].ArrvStn = FlightPara[f2].ArrvStn;
        FlightPara[f1].RelaDptrTime = FlightPara[f2].RelaDptrTime;
        FlightPara[f1].RelaArrvTime = FlightPara[f2].RelaArrvTime;
        FlightPara[f1].LegTime = FlightPara[f2].LegTime;
        FlightPara[f1].FltNum = FlightPara[f2].FltNum;
        //f2->f1
        FlightPara[f2].DptrDate = DptrDate;
        FlightPara[f2].DptrTime = DptrTime;
        FlightPara[f2].ArrvDate = ArrvDate;
        FlightPara[f2].ArrvTime = ArrvTime;
        FlightPara[f2].DptrStn = DptrStn;
        FlightPara[f2].ArrvStn = ArrvStn;
        FlightPara[f2].RelaDptrTime = RelaDptrTime;
        FlightPara[f2].RelaArrvTime = RelaArrvTime;
        FlightPara[f2].LegTime = LegTime;
        FlightPara[f2].FltNum = FltNum;
}

int checkStartFlight(int f) {
        int i, ff;
        if (StnCrewNum[FlightPara[f].DptrStn] != 0) {//判断是否在起点了，是则 return
                return 1;//找到了
        }

        for (i = 0; i < f; i++) {//没找到则搜寻
                ff = i;
                if (FlightPara[ff].ArrvStn == FlightPara[f].DptrStn && FlightPara[f].RelaDptrTime -
FlightPara[ff].RelaArrvTime >= MinCT) {//如果满足要求
                        if (checkStartFlight(ff) == 1) {
                                return 1;
                        }//搜寻 ff
                }
        }

        return -1;
}

int checkEndFlight(int f) {
        int i, ff;

        if (StnCrewNum[FlightPara[f].ArrvStn] != 0) {//判断是否在起点了，是则 return
```

```
            FlightCNum[f] += 2;
            return 1;//找到了
        }

        for (i = f + 1; i < FlightNum; i++) {
            ff = i;
            if (FlightPara[ff].DptrStn == FlightPara[f].ArrvStn && FlightPara[ff].RelaDptrTime -
FlightPara[f].RelaArrvTime >= MinCT) {//如果满足时间要求
                if (checkEndFlight(ff) == 1) {
                    return 1;
                }//搜寻 ff
            }
        }
        return -1;
}

void read_initial(string inFile_1, string inFile_2) {
        int i, j, k, date, time, f, l;
        //读取 inFile_A
        vector<vector<string>> data_A;
        data_A = read_csv(inFile_1);
        //读取 inFile_B
        vector<vector<string>> data_B;
        data_B = read_csv(inFile_2);

        for (i = 0; i < StnNum; i++) {
            StnCrewNum[i] = 0;
        }

        for (i = 0; i < CrewNum; i++) {
            CrewPara[i].Degree[2] = atoi(data_A[i + 1][1].c_str());
            CrewPara[i].Degree[1] = atoi(data_A[i + 1][2].c_str());
            CrewPara[i].Base = atoi(data_A[i + 1][4].c_str());
            CrewPara[i].DutyCostPerHr = atoi(data_A[i + 1][5].c_str());
            CrewPara[i].ParingCostPerHr = atoi(data_A[i + 1][6].c_str());
            CrewPara[i].CurrentStn = CrewPara[i].Base;//初始在 Base
            CrewPara[i].lastFlight = -1;//上一航班
            if (CrewPara[i].Degree[2] == 1 && CrewPara[i].Degree[1] == 1) {
                CaptainFirstOfficer.push_back(i);
            }
            else if (CrewPara[i].Degree[2] == 1) {
                Captain.push_back(i);
            }
            else if (CrewPara[i].Degree[1] == 1) {
```

```
                FirstOfficer.push_back(i);
            }
            StnCrewNum[CrewPara[i].Base]++;
        }

        for (i = 0; i < StnNum; i++) {
            for (j = 0; j < StnNum; j++) {
                StnFlow[i][j] = 0;
            }
        }

        for (i = 0; i < FlightNum; i++) {
            vector<int> sv;
            split(data_B[i + 1][1], sv, '/');
            FlightPara[i].DptrDate = sv[1];
            split(data_B[i + 1][2], sv, ':');
            FlightPara[i].DptrTime = sv[0] * 60 + sv[1];
            split(data_B[i + 1][4], sv, '/');
            FlightPara[i].ArrvDate = sv[1];
            split(data_B[i + 1][5], sv, ':');
            FlightPara[i].ArrvTime = sv[0] * 60 + sv[1];
            FlightPara[i].DptrStn = atoi(data_B[i + 1][3].c_str());
            FlightPara[i].ArrvStn = atoi(data_B[i + 1][6].c_str());
            FlightPara[i].RelaDptrTime = (FlightPara[i].DptrDate - startDate) * 24 * 60 +
FlightPara[i].DptrTime;
            FlightPara[i].RelaArrvTime = (FlightPara[i].ArrvDate - startDate) * 24 * 60 +
FlightPara[i].ArrvTime;
            FlightPara[i].LegTime = (FlightPara[i].ArrvDate - FlightPara[i].DptrDate) * 24 * 60 +
(FlightPara[i].ArrvTime - FlightPara[i].DptrTime);
            FlightCrewNum[i][0] = FlightCrewNum[i][1] = FlightCrewNum[i][2] = 0;
            FlightCNum[i] = 0;
            FlightPara[i].isFly = 0;//初始状态，都没有起飞
            FlightPara[i].FltNum = atoi(data_B[i + 1][0].substr(2, data_B[i + 1][0].size() - 1).c_str());
            StnFlow[FlightPara[i].DptrStn][FlightPara[i].ArrvStn]++;
            FlightBackP[i][0] = FlightBackP[i][1] = 0;//回来的可能性都为 0
            FlightCanFly[i] = 1;//初始化都能起飞
        }

        //对航班进行排序
        for (i = 0; i < FlightNum - 1; i++) {
            for (j = i + 1; j < FlightNum; j++) {
                if (FlightPara[i].RelaDptrTime > FlightPara[j].RelaDptrTime) {
                    exchangeFlight(i, j);
                }
```

```
                else if (FlightPara[i].RelaDptrTime == FlightPara[j].RelaDptrTime) {
                    if (FlightPara[i].RelaArrvTime > FlightPara[j].RelaArrvTime) {
                        exchangeFlight(i, j);
                    }
                }
            }
        }

        //对于每一个航班判断能否起飞
        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            if (StnCrewNum[FlightPara[i].DptrStn] == 0) {
                f = checkStartFlight(i);
                if (f == -1) {
                    FlightCanFly[i] = 0;
                }
            }
        }

        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            if (StnCrewNum[FlightPara[i].ArrvStn] == 0) {
                f = checkEndFlight(i);
                if (f == -1) {
                    FlightCanFly[i] = 0;
                }
            }
        }

        int dt, ar;
        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            ar = FlightPara[i].ArrvStn;
            FlightBackP[i][0] = StnFlow[ar][0];
            FlightBackP[i][1] = StnFlow[ar][1];
        }
    }

    int find_Captain(int f) {
        int i, j, l, c, best_c;
        best_c = -1;
        for (i = 0; i < CaptainFirstOfficer.size(); i++) {
            //飞行员当前所在地与航班出发点相同，飞行员有正机长和副机长资格
            c = CaptainFirstOfficer[i];
            if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                if (CrewPara[c].lastFlight != -1) {//判断时间是否可行
                    l = CrewPara[c].lastFlight;
```

```
                    if (FlightPara[f].RelaDptrTime < (FlightPara[l].RelaArrvTime + MinCT)) {
                        continue;
                    }
                }
                if (best_c == -1) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                    if (CrewFlight[c].size() > CrewFlight[best_c].size()) {
                        best_c = c;
                    }
                }
            }
        }
        if (best_c == -1) {
            for (i = 0; i < Captain.size(); i++) {
                //飞行员当前所在地与航班出发点相同，飞行员只有正机长资格
                c = Captain[i];
                if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                    if (CrewPara[c].lastFlight != -1) {//判断时间是否可行
                        l = CrewPara[c].lastFlight;
                        if (FlightPara[f].RelaDptrTime < (FlightPara[l].RelaArrvTime + MinCT)) {
                            continue;
                        }
                    }
                    if (best_c == -1) {
                        best_c = c;
                    }
                    else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                        best_c = c;
                    }
                    else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                        if (CrewFlight[c].size() > CrewFlight[best_c].size()) {
                            best_c = c;
                        }
                    }
```

```
                }
            }
        }
        return best_c;//没有找到可用机长
}

int find_FirstOfficer(int f) {
        int i, j, l, c, best_c;
        best_c = -1;
        for (i = 0; i < FirstOfficer.size(); i++) {//飞行员只有副机长资格
            //飞行员当前所在地与航班出发点相同
            c = FirstOfficer[i];
            if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                if (CrewPara[c].lastFlight != -1) {//判断时间是否可行
                    l = CrewPara[c].lastFlight;
                    if (FlightPara[f].RelaDptrTime < (FlightPara[l].RelaArrvTime + MinCT)) {
                        continue;
                    }
                }
                if (best_c == -1) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                    if (CrewFlight[c].size() > CrewFlight[best_c].size()) {
                        best_c = c;
                    }
                }
            }
        }
        if (best_c == -1) {
            for (i = 0; i < CaptainFirstOfficer.size(); i++) {
                //飞行员当前所在地与航班出发点相同，飞行员有正机长和副机长资格
                c = CaptainFirstOfficer[i];
                if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                    if (CrewPara[c].lastFlight != -1) {//判断时间是否可行
                        l = CrewPara[c].lastFlight;
                        if (FlightPara[f].RelaDptrTime < (FlightPara[l].RelaArrvTime + MinCT))
{
                            continue;
```

```
                    }
                }
                if (best_c == -1) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                    if (CrewFlight[c].size() > CrewFlight[best_c].size()) {
                        best_c = c;
                    }
                }
            }
        }
    }
    return best_c;//没有找到可用副机长
}

void updateCrew(int c, int f, int degree) {
    CrewFlight[c].push_back(f);
    CrewDegree[c].push_back(degree);
    CrewPara[c].CurrentStn = FlightPara[f].ArrvStn;//更新当前位置
    CrewPara[c].lastFlight = f;
    FlightCrewNum[f][degree] += 1;
}

int timeIsOk(int c, Flight f) {//判断时间是否够
    int l;
    //若飞行员处于没有工作过的状态则直接可行
    if (CrewPara[c].lastFlight != -1) {//若飞行员工作过
        l = CrewPara[c].lastFlight;
        if (f.RelaDptrTime < FlightPara[l].RelaArrvTime + MinCT) {
            return -1;
        }
    }
    return c;
}

int calculateCrew(int crew, int flight1, int flight2, int degree) {
    int i, j, l;
    int cost = 0;
```

```
    if (degree == 1 && CrewPara[crew].Degree[2] == 1) {//触发替补
        cost += 100000;
    }
    return cost;
}

int repair(int f, int degree) {
    int i, j, isflight, best_c, best_f;
    int flag, f_c;
    int best_f_c = 9999999;
    best_c = best_f = -1;
    for (i = 0; i < f; i++) {//对于所有比该航班要早的航班
        //如果该航班还坐得下，且该航班已经满足机组配置，且落地地点是该机场
        if (FlightCrewNum[i][0] < 5 && FlightPara[i].isFly == 1 && FlightPara[i].ArrvStn ==
FlightPara[f].DptrStn) {
            //航班可以在下一航班起飞前到达
            if (FlightPara[i].RelaArrvTime + MinCT <= FlightPara[f].RelaDptrTime) {//航班到
达时间和地点符合
                for (j = 0; j < CrewNum; j++) {//找一个职级满足要求的，且在该航班出发
机场
                    if (CrewPara[j].Degree[degree] == 1 && FlightPara[i].DptrStn ==
CrewPara[j].CurrentStn) {//这里有点问题，需要判断等级
                        flag = timeIsOk(j, FlightPara[i]);//确定不会超执勤时间
                        if (flag != -1) {//找到该人，对该人进行评价，是否合适
                            f_c = calculateCrew(j, i, f, degree);//j 乘坐 i 去开 f
                             //cout << f_c << endl;
                            if (f_c < best_f_c) {//挑选最小的
                                best_c = j;
                                best_f = i;
                                best_f_c = f_c;
                            }
                        }
                    }
                }
            }
        }
    }

    if (best_c != -1 && best_f != -1) {
        if (FlightPara[f].DptrDate > FlightPara[best_f].DptrDate) {
            updateCrew(best_c, best_f, 0);//使其坐过来，飞完休息
        }
        else {
            updateCrew(best_c, best_f, 0);//使其坐过来，飞完休息
```

```
            }

            return best_c;
        }
        return -1;
}

void clearFlight(int c, int flag) {//删除航班
        int l = 0, f;
        while (l < flag) {//回溯的次数
            CrewFlight[c].pop_back();
            CrewDegree[c].pop_back();
            f = CrewPara[c].lastFlight;
            CrewPara[c].CurrentStn = FlightPara[f].DptrStn;//更新当前位置
            if (CrewFlight[c].size() > 0) {
                CrewPara[c].lastFlight = CrewFlight[c][CrewFlight[c].size() - 1];
            }
            else {//还未出发
                CrewPara[c].lastFlight = -1;
            }
            l++;
        }
}

void random_initial() {
        int i, j, k;
        int c, f, flag = 0;
        int currentDate = startDate;
        for (i = 0; i < FlightNum; i++) {
            if (FlightCanFly[i] == 1) {
                f = c = -1;
                flag = 1;
                if (FlightCrewNum[i][1] == 0) {//寻找副机长
                    f = find_FirstOfficer(i);//寻找副机长
                    if (f == -1) {//如果没有找到副机长
                        f = repair(i, 1);//从其他地方调一个副机长过来
                        flag = 2;//经历了修复过，需要多回溯一次
                    }
                    if (f != -1) {
                        updateCrew(f, i, 1);//找到了则更新该飞行员的航班
                        FlightCrewNum[i][1] = 1;
                    }
                }
                if (f != -1 && FlightCrewNum[i][2] == 0) {//寻找到副机长了，则可以开始寻找
```

```
机长
                    c = find_Captain(i);//寻找机长
                    if (c == -1) {//如果没有找到机长
                        c = repair(i, 2);//从其他地方调一个机长过来
                    }
                    if (c != -1) {
                        updateCrew(c, i, 2);//找到了则更新
                        FlightCrewNum[i][2] = 1;
                    }
                    else {
                        clearFlight(f, flag);//找不到需要回溯
                    }
                }

                if (c != -1) {//配置成功
                    FlightPara[i].isFly = 1;
                }
            }
        }
    }

int main()
{
    int i, j;
    srand((unsigned)time(NULL));
    read_initial(inFile_A, inFile_B);//读取数据
    startTime = clock();//计时开始
    endTime = startTime;
    random_initial();
    endTime = clock();
    cout << (double)(endTime - startTime) / CLOCKS_PER_SEC << endl;
    return 0;
}
```

问题二：

```
// Problem2.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <time.h>
#include <ctime>
```

```
#include <string>
#include <string.h>
#include <algorithm>
#include <vector>

using namespace std;
#define MAX_VAL 999999

clock_t startTime, endTime;//记录运行时间
double bestTime;//最佳时间
double cutting_time = 1800;//终止条件
string inFile_A = "F:/华为比赛/华为建模比赛/2021 年 F 题/机组排班 Data B-Crew.csv";
string inFile_B = "F:/华为比赛/华为建模比赛/2021 年 F 题/机组排班 Data B-Flight.csv";
string UncoveredFlights = "F:/华为比赛/华为建模比赛/2021 年 F 题/UncoveredFlights_B.csv";
string CrewRosters = "F:/华为比赛/华为建模比赛/2021 年 F 题/CrewRosters_B.csv";
const int MaxDH = 5;//最多乘机人数
const int MinCT = 40;
const int MaxBlk = 600;
const int MaxDP = 720;
const int MinRest = 660;
const int MaxTAFB = 14400;
const int MaxSuccOn = 4;//4 天
const int MinVacDay = 2;//2 天

//数据 A
//const int startDate = 11;
//const int FlightNum = 206;//航班数量
//const int CrewNum = 21;//机组人员
//const int StnNum = 7;
//数据 B
const int startDate = 1;
const int FlightNum = 13954;//航班数量
const int CrewNum = 465;//机组人员
const int StnNum = 39;
int FlightCrewNum[FlightNum][3];//0, 1, 2,1 表示副机长，2 表示机长
int StnFlow[StnNum][StnNum];
int StnCrewNum[StnNum];
int FlightCanFly[FlightNum];//飞机是否能起飞
int FlightBackP[FlightNum][2];//回来的可能性
int CrewDutyTime[CrewNum];
int CrewFlyTime[CrewNum];
vector<vector<int>> CrewFlight(CrewNum);
vector<vector<int>> CrewDegree(CrewNum);
vector<vector<int>> CrewStatus(CrewNum);//状态
```

```
vector<int> Captain;
vector<int> FirstOfficer;
vector<int> CaptainFirstOfficer;

struct Crew {
	int Degree[3];//等级
	int Base;//出发基地
	int DutyCostPerHr;//每小时执勤成本
	int ParingCostPerHr;//任务环成本
	int CurrentStn;//当前所在机场
	int lastFlight;//上一航班
	int CurrentStatus;//0：休息状态, 1：工作状态， 2：休假状态
	int CurrentDutyTime;//当前执勤时长
	int CurrentFlyTime;//当前飞行时长
	int TotalDutyTime;
	int ConDutyDay;//当前连续执勤天数
	int ConPairingTime;//当前任务环时间，不能大于 14400
};
Crew CrewPara[CrewNum];//机组人员属性

struct Flight {
	int FltNum;
	int DptrDate;//航班起飞日期
	int DptrTime;//航班起飞时间
	int ArrvDate;//航班到达日期
	int ArrvTime;//航班到达时间
	int DptrStn;//航班出发机场
	int ArrvStn;//航班到达机场
	int RelaDptrTime;//航班相对出发时间
	int RelaArrvTime;//航班相对到达时间
	int LegTime;//航班持续时长
	int isFly;
	//最低驾驶配置都是 C1F1
};
Flight FlightPara[FlightNum];//航班属性

vector<vector<string>> read_csv(string file) {
	int i, j;
	ifstream inFile(file, ios::in);
	string lineStr;
	vector<vector<string>> strArray;
	while (getline(inFile, lineStr))
	{
		// 存成二维表结构
```

```
            stringstream ss(lineStr);
            string str;
            vector<string> lineArray;
            // 按照逗号分隔
            while (getline(ss, str, ','))
                lineArray.push_back(str);
            strArray.push_back(lineArray);
        }
        return strArray;
}

void split(const string& s, vector<int>& sv, const char flag = ' ') {
        sv.clear();
        istringstream iss(s);
        string temp;

        while (getline(iss, temp, flag)) {
            sv.push_back(stoi(temp));
        }
        return;
}

void exchangeFlight(int f1, int f2) {
        int DptrDate = FlightPara[f1].DptrDate;//航班起飞日期
        int DptrTime = FlightPara[f1].DptrTime;//航班起飞时间
        int ArrvDate = FlightPara[f1].ArrvDate;//航班到达日期
        int ArrvTime = FlightPara[f1].ArrvTime;//航班到达时间
        int DptrStn = FlightPara[f1].DptrStn;//航班出发机场
        int ArrvStn = FlightPara[f1].ArrvStn;//航班到达机场
        int RelaDptrTime = FlightPara[f1].RelaDptrTime;//航班相对出发时间
        int RelaArrvTime = FlightPara[f1].RelaArrvTime;//航班相对到达时间
        int LegTime = FlightPara[f1].LegTime;//航班持续时长
        int FltNum = FlightPara[f1].FltNum;
        FlightPara[f1].DptrDate = FlightPara[f2].DptrDate;
        FlightPara[f1].DptrTime = FlightPara[f2].DptrTime;
        FlightPara[f1].ArrvDate = FlightPara[f2].ArrvDate;
        FlightPara[f1].ArrvTime = FlightPara[f2].ArrvTime;
        FlightPara[f1].DptrStn = FlightPara[f2].DptrStn;
        FlightPara[f1].ArrvStn = FlightPara[f2].ArrvStn;
        FlightPara[f1].RelaDptrTime = FlightPara[f2].RelaDptrTime;
        FlightPara[f1].RelaArrvTime = FlightPara[f2].RelaArrvTime;
        FlightPara[f1].LegTime = FlightPara[f2].LegTime;
        FlightPara[f1].FltNum = FlightPara[f2].FltNum;
        FlightPara[f2].DptrDate = DptrDate;
```

```
        FlightPara[f2].DptrTime = DptrTime;
        FlightPara[f2].ArrvDate = ArrvDate;
        FlightPara[f2].ArrvTime = ArrvTime;
        FlightPara[f2].DptrStn = DptrStn;
        FlightPara[f2].ArrvStn = ArrvStn;
        FlightPara[f2].RelaDptrTime = RelaDptrTime;
        FlightPara[f2].RelaArrvTime = RelaArrvTime;
        FlightPara[f2].LegTime = LegTime;
        FlightPara[f2].FltNum = FltNum;
}

int checkStartFlight(int f) {
        int i, ff;
        if (StnCrewNum[FlightPara[f].DptrStn] != 0) {//判断是否在起点了，是则 return
                return 1;//找到了
        }
        for (i = 0; i < f; i++) {//没找到则搜寻
                ff = i;
                if (FlightPara[ff].ArrvStn == FlightPara[f].DptrStn && FlightPara[f].RelaDptrTime -
FlightPara[ff].RelaArrvTime >= MinCT) {//如果满足要求
                        if (checkStartFlight(ff) == 1) {
                                return 1;
                        }//搜寻 ff
                }
        }
        return -1;
}

int checkEndFlight(int f) {
        int i, ff;

        if (StnCrewNum[FlightPara[f].ArrvStn] != 0) {//判断是否在起点了，是则 return
                return 1;//找到了
        }

        for (i = f + 1; i < FlightNum; i++) {
                ff = i;
                if (FlightPara[ff].DptrStn == FlightPara[f].ArrvStn && FlightPara[ff].RelaDptrTime -
FlightPara[f].RelaArrvTime >= MinCT) {//如果满足时间要求
                        if (checkEndFlight(ff) == 1) {
                                return 1;
                        }//搜寻 ff
                }
        }
```

```
    return -1;
}

void read_initial(string inFile_1, string inFile_2) {
    int i, j, k, date, time, l;
    int dayFlightNum[32];
    for (i = 1; i < 32; i++) {
        dayFlightNum[i] = 0;
    }
    //读取 inFile_A
    vector<vector<string>> data_A;
    data_A = read_csv(inFile_1);
    //读取 inFile_B
    vector<vector<string>> data_B;
    data_B = read_csv(inFile_2);

    for (i = 0; i < CrewNum; i++) {
        CrewPara[i].Degree[2] = atoi(data_A[i + 1][1].c_str());
        CrewPara[i].Degree[1] = atoi(data_A[i + 1][2].c_str());
        CrewPara[i].Base = atoi(data_A[i + 1][4].c_str());
        CrewPara[i].DutyCostPerHr = atoi(data_A[i + 1][5].c_str());
        CrewPara[i].ParingCostPerHr = atoi(data_A[i + 1][6].c_str());
        CrewPara[i].CurrentStn = CrewPara[i].Base;//初始在 Base
        CrewPara[i].lastFlight = -1;//上一航班
        CrewPara[i].CurrentStatus = 1;//初始为工作状态
        CrewPara[i].CurrentDutyTime = 0;
        CrewPara[i].CurrentFlyTime = 0;
        CrewPara[i].TotalDutyTime = 0;
        CrewPara[i].ConDutyDay = 0;//当前连续执勤天数
        CrewPara[i].ConPairingTime = 0;//当前任务环时间，不能大于 14400
        CrewDutyTime[i] = 0;
        CrewFlyTime[i] = 0;

        if (CrewPara[i].Degree[2] == 1 && CrewPara[i].Degree[1] == 1) {
            CaptainFirstOfficer.push_back(i);
        }
        else if (CrewPara[i].Degree[2] == 1) {
            Captain.push_back(i);
        }
        else if (CrewPara[i].Degree[1] == 1) {
            FirstOfficer.push_back(i);
        }
        StnCrewNum[CrewPara[i].Base]++;
    }
```

```
    for (i = 0; i < StnNum; i++) {
        for (j = 0; j < StnNum; j++) {
            StnFlow[i][j] = 0;
        }
    }

    for (i = 0; i < FlightNum; i++) {
        vector<int> sv;
        split(data_B[i + 1][1], sv, '/');
        FlightPara[i].DptrDate = sv[1];
        split(data_B[i + 1][2], sv, ':');
        FlightPara[i].DptrTime = sv[0] * 60 + sv[1];
        split(data_B[i + 1][4], sv, '/');
        FlightPara[i].ArrvDate = sv[1];
        split(data_B[i + 1][5], sv, ':');
        FlightPara[i].ArrvTime = sv[0] * 60 + sv[1];
        FlightPara[i].DptrStn = atoi(data_B[i + 1][3].c_str());
        FlightPara[i].ArrvStn = atoi(data_B[i + 1][6].c_str());
        FlightPara[i].RelaDptrTime = (FlightPara[i].DptrDate - startDate) * 24 * 60 +
FlightPara[i].DptrTime;
        FlightPara[i].RelaArrvTime = (FlightPara[i].ArrvDate - startDate) * 24 * 60 +
FlightPara[i].ArrvTime;
        FlightPara[i].LegTime = (FlightPara[i].ArrvDate - FlightPara[i].DptrDate) * 24 * 60 +
(FlightPara[i].ArrvTime - FlightPara[i].DptrTime);
        FlightCrewNum[i][0] = FlightCrewNum[i][1] = FlightCrewNum[i][2] = 0;
        FlightPara[i].isFly = 0;//初始状态，都不能起飞
        FlightPara[i].FltNum = atoi(data_B[i + 1][0].substr(2, data_B[i + 1][0].size() - 1).c_str());
        StnFlow[FlightPara[i].DptrStn][FlightPara[i].ArrvStn]++;
        dayFlightNum[FlightPara[i].DptrDate]++;
        FlightBackP[i][0] = FlightBackP[i][1] = 0;//回来的可能性都为 0
        FlightCanFly[i] = 1;//初始化都能起飞
    }

    //对航班进行排序
    for (i = 0; i < FlightNum - 1; i++) {
        for (j = i + 1; j < FlightNum; j++) {
            if (FlightPara[i].RelaDptrTime > FlightPara[j].RelaDptrTime) {
                exchangeFlight(i, j);
            }
            else if (FlightPara[i].RelaDptrTime == FlightPara[j].RelaDptrTime) {
                if (FlightPara[i].RelaArrvTime > FlightPara[j].RelaArrvTime) {
                    exchangeFlight(i, j);
                }
```

```
                }
            }
        }

        //对于每一个航班判断能否起飞
        int f;
        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            if (StnCrewNum[FlightPara[i].DptrStn] == 0) {
                f = checkStartFlight(i);
                if (f == -1) {
                    FlightCanFly[i] = 0;
                }
            }
        }

        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            if (StnCrewNum[FlightPara[i].ArrvStn] == 0) {
                f = checkEndFlight(i);
                if (f == -1) {
                    FlightCanFly[i] = 0;
                }
            }
        }

        int dt, ar;
        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            ar = FlightPara[i].ArrvStn;
            FlightBackP[i][0] = StnFlow[ar][0];
            FlightBackP[i][1] = StnFlow[ar][1];
        }
    }

    int timeIsOk(int c, Flight f) {//判断时间是否够
        int l;
        //若飞行员处于没有工作过的状态则直接可行
        if (CrewPara[c].lastFlight != -1) {//若飞行员工作过
            l = CrewPara[c].lastFlight;
            if (CrewPara[c].CurrentStatus == 1) {//当前为工作状态
                //判断飞行时间不超过 MaxBlk, 执勤时间不超过 MaxDP，满足连接时间
                if ((f.RelaDptrTime < FlightPara[l].RelaArrvTime + MinCT) ||
    (CrewPara[c].CurrentFlyTime + f.LegTime > MaxBlk) || (CrewPara[c].CurrentDutyTime +
    (f.RelaArrvTime - FlightPara[l].RelaArrvTime) > MaxDP)) {
                    return -1;
                }
```

```
            }
            else {//当前处于休息状态，当天的第一班航班，判断是否休息足够长的时间
                if (f.RelaDptrTime < FlightPara[l].RelaArrvTime + MinRest) {
                    return -1;
                }
            }
        }
        return c;
}

int find_Captain(int f) {
        int i, j, l, c, best_c;
        int flag = 0;
        best_c = -1;
        for (i = 0; i < CaptainFirstOfficer.size(); i++) {
            //飞行员当前所在地与航班出发点相同，飞行员有正机长和副机长资格
            c = CaptainFirstOfficer[i];
            if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                flag = timeIsOk(c, FlightPara[f]);
                if (flag != -1) {
                    if (best_c == -1) {
                        best_c = c;
                    }
                    else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                        best_c = c;
                    }
                    else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                        if (CrewPara[c].TotalDutyTime < CrewPara[best_c].TotalDutyTime) {
                            best_c = c;
                        }
                    }
                }
            }
        }
        if (best_c == -1) {
            for (i = 0; i < Captain.size(); i++) {
                //飞行员当前所在地与航班出发点相同，飞行员只有正机长资格
                c = Captain[i];
                if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                    flag = timeIsOk(c, FlightPara[f]);
                    if (flag != -1) {
                        if (best_c == -1) {
```

```
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                    if (CrewPara[c].TotalDutyTime < CrewPara[best_c].TotalDutyTime) {
                        best_c = c;
                    }
                }
            }
        }
    }

    return best_c;//没有找到可用机长
}

int find_FirstOfficer(int f) {
    int i, j, l, c, best_c;
    int flag = 0;
    best_c = -1;
    for (i = 0; i < FirstOfficer.size(); i++) {//飞行员只有副机长资格
                                               //飞行员当前所在地与航班出发点相同
        c = FirstOfficer[i];
        if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {//判断是否在当前机场
            flag = timeIsOk(c, FlightPara[f]);
            if (flag != -1) {
                if (best_c == -1) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] > FlightBackP[f][CrewPara[best_c].Base]) {
                    best_c = c;
                }
                else if (FlightBackP[f][CrewPara[c].Base] == FlightBackP[f][CrewPara[best_c].Base]) {
                    if (CrewPara[c].TotalDutyTime < CrewPara[best_c].TotalDutyTime) {
                        best_c = c;
                    }
```

```
                    }
                }
            }
        }
        if (best_c == -1) {
            for (i = 0; i < CaptainFirstOfficer.size(); i++) {
                //飞行员当前所在地与航班出发点相同，飞行员有正机长和副机长资格
                c = CaptainFirstOfficer[i];
                if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                    flag = timeIsOk(c, FlightPara[f]);
                    if (flag != -1) {
                        if (best_c == -1) {
                            best_c = c;
                        }
                        else if (CrewPara[c].TotalDutyTime < CrewPara[best_c].TotalDutyTime)
{
                            best_c = c;
                        }
                    }
                }
            }
        }
        return best_c;//没有找到可用副机长
}

void updateCrew(int c, int f, int degree, int status) {
        CrewFlight[c].push_back(f);
        CrewDegree[c].push_back(degree);
        CrewStatus[c].push_back(status);
        if (status == 0) {//中间航班
            if (CrewPara[c].CurrentStatus == 0 || CrewPara[c].lastFlight == -1) {//第一天从基地起
飞或者刚休息完
                CrewPara[c].TotalDutyTime += FlightPara[f].LegTime;
            }
            else {//处于执勤状态下
                CrewPara[c].TotalDutyTime += FlightPara[f].RelaArrvTime -
FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
            }
            CrewPara[c].CurrentDutyTime = 0;
            CrewPara[c].CurrentFlyTime = 0;
        }
        else {
            if (CrewPara[c].CurrentStatus == 0 || CrewPara[c].lastFlight == -1) {//第一天从基地起
飞或者刚休息完
```

```
                CrewPara[c].TotalDutyTime += FlightPara[f].LegTime;
                CrewPara[c].CurrentDutyTime += FlightPara[f].LegTime;
            }
            else {//处于执勤状态下
                CrewPara[c].TotalDutyTime += FlightPara[f].RelaArrvTime -
FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
                CrewPara[c].CurrentDutyTime += FlightPara[f].RelaArrvTime -
FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
            }
            CrewPara[c].CurrentFlyTime += FlightPara[f].LegTime;
        }
        CrewPara[c].CurrentStatus = status;//进入工作状态
        CrewPara[c].CurrentStn = FlightPara[f].ArrvStn;//更新当前位置
        CrewPara[c].lastFlight = f;
        FlightCrewNum[f][degree] += 1;

}

int calculateCrew(int crew, int flight1, int flight2, int degree) {
        int i, j, l;
        int cost = 0;
        if (CrewPara[crew].lastFlight != -1) {//对于不是首次飞行的
            l = CrewPara[crew].lastFlight;
            if (FlightPara[flight2].DptrDate == FlightPara[flight1].DptrDate) {//同一天
                if (CrewPara[crew].CurrentDutyTime + FlightPara[flight2].RelaArrvTime -
FlightPara[l].RelaArrvTime > MaxDP) {//超过最大执勤时间
                    return 9999999;
                }
                if (CrewPara[crew].CurrentFlyTime + FlightPara[flight2].LegTime +
FlightPara[flight1].LegTime > MaxBlk) {//超过最大飞行时间
                    return 9999999;
                }
            }
            else {
                if (FlightPara[flight2].RelaDptrTime - FlightPara[flight1].RelaArrvTime < MinRest)
{//休息时间不够
                    return 9999999;
                }
            }
        }
        else if (CrewPara[crew].lastFlight == -1) {//对于首次飞行的
            if (FlightPara[flight2].DptrDate == FlightPara[flight1].DptrDate) {//同一天
                if (CrewPara[crew].CurrentDutyTime + FlightPara[flight2].RelaArrvTime -
FlightPara[flight1].RelaDptrTime > MaxDP) {//超过最大执勤时间
```

```
                    return 9999999;
                }
                if (CrewPara[crew].CurrentFlyTime + FlightPara[flight2].LegTime +
FlightPara[flight1].LegTime > MaxBlk) {//超过最大飞行时间
                    return 9999999;
                }
            }
            else {
                if (FlightPara[flight2].RelaDptrTime - FlightPara[flight1].RelaArrvTime < MinRest)
{//休息时间不够
                    return 9999999;
                }
            }
        }

        if (degree == 1 && CrewPara[crew].Degree[2] == 1) {//触发替补
            cost += 100000;
        }

        cost += CrewPara[crew].TotalDutyTime;//执勤时间越少越可能被选中

        return cost;
}

int repair(int f, int degree) {
        int i, j, best_c, best_f, flag, f_c, best_f_c;
        best_f_c = 9999999;
        best_c = best_f = -1;
        flag = 0;
        for (i = 0; i < f; i++) {//对于所有比该航班要早的航班
            //如果该航班还坐得下，且该航班已经满足机组配置，且落地地点是该机场
            if (FlightCrewNum[i][0] < 5 && FlightPara[i].isFly == 1 && FlightPara[i].ArrvStn ==
FlightPara[f].DptrStn) {
                //航班可以在下一航班起飞前到达
                if (FlightPara[i].RelaArrvTime + MinCT <= FlightPara[f].RelaDptrTime) {//航班到
达时间和地点符合
                    for (j = 0; j < CrewNum; j++) {//找一个职级满足要求的，且在该航班出发
机场
                        if (CrewPara[j].Degree[degree] == 1 && FlightPara[i].DptrStn ==
CrewPara[j].CurrentStn) {//这里有点问题，需要判断等级
                            flag = timeIsOk(j, FlightPara[i]);//确定不会超执勤时间
                            if (flag != -1) {//找到该人，对该人进行评价，是否合适
                                f_c = calculateCrew(j, i, f, degree);//j 乘坐 i 去开 f
                                //cout << f_c << endl;
```

```
                                    if (f_c < best_f_c) {//挑选最小的
                                        best_c = j;
                                        best_f = i;
                                        best_f_c = f_c;
                                    }
                                }
                            }
                        }
                    }
                }

                if (best_c != -1 && best_f != -1) {
                    if (FlightPara[f].DptrDate > FlightPara[best_f].DptrDate) {
                        updateCrew(best_c, best_f, 0, 0);//使其坐过来，飞完休息
                    }
                    else {
                        updateCrew(best_c, best_f, 0, 1);//使其坐过来，飞完休息
                    }

                    return best_c;
                }
                return -1;
            }

            void clearFlight(int c, int flag) {//删除航班
                int l = 0, f;
                while (l < flag) {//回溯的次数
                    CrewFlight[c].pop_back();
                    CrewDegree[c].pop_back();
                    CrewStatus[c].pop_back();
                    f = CrewPara[c].lastFlight;
                    CrewPara[c].CurrentStn = FlightPara[f].DptrStn;//更新当前位置
                    if (CrewFlight[c].size() > 0) {
                        CrewPara[c].CurrentFlyTime = CrewPara[c].CurrentFlyTime -
            FlightPara[f].LegTime;
                        CrewPara[c].CurrentStatus = CrewStatus[c][CrewStatus[c].size() - 1];
                        CrewPara[c].lastFlight = CrewFlight[c][CrewFlight[c].size() - 1];
                        if (CrewPara[c].CurrentStatus == 0) {
                            CrewPara[c].CurrentDutyTime = 0;
                        }
                        else {
                            CrewPara[c].CurrentDutyTime = CrewPara[c].CurrentDutyTime -
            (FlightPara[f].RelaArrvTime - FlightPara[CrewPara[c].lastFlight].RelaArrvTime);
```

```
                }

            }
            else {//还未出发
                CrewPara[c].lastFlight = -1;
                CrewPara[c].CurrentDutyTime = 0;
                CrewPara[c].CurrentFlyTime = 0;
                CrewPara[c].CurrentStatus = 1;
            }
            l++;
        }
    }

    void updateCrewStatus() {//每当新的一天
        int i;
        for (i = 0; i < CrewNum; i++) {
            if (CrewFlight[i].size() > 0) {
                CrewPara[i].CurrentDutyTime = 0;
                CrewPara[i].CurrentFlyTime = 0;
                if (CrewPara[i].CurrentStatus != 2) {//处于非休假状态
                    CrewPara[i].CurrentStatus = 0;//当前状态为休息状态
                    CrewStatus[i][CrewStatus[i].size() - 1] = 0;//休息状态
                }
            }
        }
    }

    void random_initial() {
        int i, j, k;
        int c, f, flag = 0;
        int currentDate = startDate;
        for (i = 0; i < FlightNum; i++) {
            if (FlightCanFly[i] == 1) {
                f = c = -1;
                if (FlightPara[i].DptrDate > currentDate) {//若到了第二天，更新状态

                    updateCrewStatus();
                    currentDate = FlightPara[i].DptrDate;
                }

                flag = 1;
                if (FlightCrewNum[i][1] == 0) {//寻找副机长
                    f = find_FirstOfficer(i);//寻找副机长
                    if (f == -1) {//如果没有找到副机长
```

```
                    f = repair(i, 1);//从其他地方调一个副机长过来
                    flag = 2;//经历了修复过，需要多回溯一次
                }
                if (f != -1) {
                    updateCrew(f, i, 1, 1);//找到了则更新该飞行员的航班
                    FlightCrewNum[i][1] = 1;
                }
            }
            if (f != -1 && FlightCrewNum[i][2] == 0) {//寻找到副机长了，则可以开始寻找
机长
                c = find_Captain(i);//寻找机长
                if (c == -1) {//如果没有找到机长
                    c = repair(i, 2);//从其他地方调一个机长过来
                }
                if (c != -1) {
                    updateCrew(c, i, 2, 1);//找到了则更新
                    FlightCrewNum[i][2] = 1;
                }
                else {
                    clearFlight(f, flag);//找不到需要回溯
                }
            }

            if (c != -1) {//配置成功
                FlightPara[i].isFly = 1;
            }
        }
    }
}

int main()
{
    int i, j;
    srand((unsigned)time(NULL));
    read_initial(inFile_A, inFile_B);//读取数据
    startTime = clock();//计时开始
    endTime = startTime;
    random_initial();
    endTime = clock();
    cout << (double)(endTime - startTime) / CLOCKS_PER_SEC << endl;
    return 0;
}
```

问题三：

```
// Problem3.cpp : 定义控制台应用程序的入口点。
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <time.h>
#include <ctime>
#include <string>
#include <string.h>
#include <algorithm>
#include <vector>

using namespace std;
#define MAX_VAL 999999

clock_t startTime, endTime;//记录运行时间
double bestTime;//最佳时间
double cutting_time = 1800;//终止条件
string inFile_A = "F:/华为比赛/华为建模比赛/2021 年 F 题/机组排班 Data B-Crew.csv";
string inFile_B = "F:/华为比赛/华为建模比赛/2021 年 F 题/机组排班 Data B-Flight.csv";
string UncoveredFlights = "F:/华为比赛/华为建模比赛/2021 年 F 题/UncoveredFlights_B.csv";
string CrewRosters = "F:/华为比赛/华为建模比赛/2021 年 F 题/CrewRosters_B.csv";
const int MaxDH = 5;//最多乘机人数
const int MinCT = 40;
const int MaxBlk = 600;
const int MaxDP = 720;
const int MinRest = 660;
const int MaxTAFB = 14400;
const int MaxSuccOn = 4;//4 天
const int MinVacDay = 2;//2 天

//数据 A
//const int startDate = 11;
//const int FlightNum = 206;//航班数量
//const int CrewNum = 21;//机组人员
//const int StnNum = 7;
//数据 B
const int startDate = 1;
const int FlightNum = 13954;//航班数量
const int CrewNum = 465;//机组人员
const int StnNum = 39;
```

```
int currentDate = startDate;
int FlightCrewNum[FlightNum][3];//0, 1, 2,1 表示副机长，2 表示机长
int StnCrewNum[StnNum];
int StnFlow[StnNum][StnNum];
int FlightCanFly[FlightNum];//飞机是否能起飞
int FlightBackP[FlightNum][2];//回来的可能性
int CrewDayStatus[CrewNum][32];//31 天
int CrewParingTime[CrewNum];
int CrewDutyTime[CrewNum];
int CrewFlyTime[CrewNum];
int l1, l2, l3, l4;
vector<vector<int>> CrewFlight(CrewNum);
vector<vector<int>> CrewDegree(CrewNum);
vector<vector<int>> CrewStatus(CrewNum);//0：休息状态, 1：工作状态， 2：休假状态
vector<vector<int>> CrewConDutyDay(CrewNum);//记录上一阶段的连续执勤时间
vector<vector<int>> CrewConPairingDay(CrewNum);//记录上一阶段的连续执勤时间
vector<int> Captain;
vector<int> FirstOfficer;
vector<int> CaptainFirstOfficer;

struct Crew {
    int Degree[3];//等级
    int Base;//出发基地
    int DutyCostPerHr;//每小时执勤成本
    int ParingCostPerHr;//任务环成本
    int CurrentStn;//当前所在机场
    int lastFlight;//上一航班
    int CurrentStatus;//0：休息状态, 1：工作状态， 2：休假状态
    int CurrentDutyTime;//当前执勤时长
    int CurrentFlyTime;//当前飞行时长
    int CurrentPairingDay;//当前任务环时长
    int TotalDutyTime;
    int TotalPairingTime; //任务环时间，总的不能大于 14400
    int ConDutyDay;//连续执勤天数
    int ConRestDay;//连续休息时长
};
Crew CrewPara[CrewNum];//机组人员属性

struct Flight {
    int FltNum;
    int DptrDate;//航班起飞日期
    int DptrTime;//航班起飞时间
    int ArrvDate;//航班到达日期
    int ArrvTime;//航班到达时间
```

```
    int DptrStn;//航班出发机场
    int ArrvStn;//航班到达机场
    int RelaDptrTime;//航班相对出发时间
    int RelaArrvTime;//航班相对到达时间
    int LegTime;//航班持续时长
    int isFly;
    //最低驾驶配置都是 C1F1
};
Flight FlightPara[FlightNum];//航班属性

vector<vector<string>> read_csv(string file) {
    int i, j;
    ifstream inFile(file, ios::in);
    string lineStr;
    vector<vector<string>> strArray;
    while (getline(inFile, lineStr))
    {
        // 存成二维表结构
        stringstream ss(lineStr);
        string str;
        vector<string> lineArray;
        // 按照逗号分隔
        while (getline(ss, str, ','))
            lineArray.push_back(str);
        strArray.push_back(lineArray);
    }
    return strArray;
}

void split(const string& s, vector<int>& sv, const char flag = ' ') {
    sv.clear();
    istringstream iss(s);
    string temp;

    while (getline(iss, temp, flag)) {
        sv.push_back(stoi(temp));
    }
    return;
}

void exchangeFlight(int f1, int f2) {
    int DptrDate = FlightPara[f1].DptrDate;//航班起飞日期
    int DptrTime = FlightPara[f1].DptrTime;//航班起飞时间
    int ArrvDate = FlightPara[f1].ArrvDate;//航班到达日期
```

```
    int ArrvTime = FlightPara[f1].ArrvTime;//航班到达时间
    int DptrStn = FlightPara[f1].DptrStn;//航班出发机场
    int ArrvStn = FlightPara[f1].ArrvStn;//航班到达机场
    int RelaDptrTime = FlightPara[f1].RelaDptrTime;//航班相对出发时间
    int RelaArrvTime = FlightPara[f1].RelaArrvTime;//航班相对到达时间
    int LegTime = FlightPara[f1].LegTime;//航班持续时长
    int FltNum = FlightPara[f1].FltNum;
    FlightPara[f1].DptrDate = FlightPara[f2].DptrDate;
    FlightPara[f1].DptrTime = FlightPara[f2].DptrTime;
    FlightPara[f1].ArrvDate = FlightPara[f2].ArrvDate;
    FlightPara[f1].ArrvTime = FlightPara[f2].ArrvTime;
    FlightPara[f1].DptrStn = FlightPara[f2].DptrStn;
    FlightPara[f1].ArrvStn = FlightPara[f2].ArrvStn;
    FlightPara[f1].RelaDptrTime = FlightPara[f2].RelaDptrTime;
    FlightPara[f1].RelaArrvTime = FlightPara[f2].RelaArrvTime;
    FlightPara[f1].LegTime = FlightPara[f2].LegTime;
    FlightPara[f1].FltNum = FlightPara[f2].FltNum;
    FlightPara[f2].DptrDate = DptrDate;
    FlightPara[f2].DptrTime = DptrTime;
    FlightPara[f2].ArrvDate = ArrvDate;
    FlightPara[f2].ArrvTime = ArrvTime;
    FlightPara[f2].DptrStn = DptrStn;
    FlightPara[f2].ArrvStn = ArrvStn;
    FlightPara[f2].RelaDptrTime = RelaDptrTime;
    FlightPara[f2].RelaArrvTime = RelaArrvTime;
    FlightPara[f2].LegTime = LegTime;
    FlightPara[f2].FltNum = FltNum;
}

int checkStartFlight(int f) {
    int i, ff;
    if (StnCrewNum[FlightPara[f].DptrStn] != 0) {//判断是否在起点了，是则 return
        return 1;//找到了
    }

    for (i = 0; i < f; i++) {//没找到则搜寻
        ff = i;
        if (FlightPara[ff].ArrvStn == FlightPara[f].DptrStn && FlightPara[f].RelaDptrTime -
FlightPara[ff].RelaArrvTime >= MinCT) {//如果满足要求
            if (checkStartFlight(ff) == 1) {
                return 1;
            }//搜寻 ff
        }
    }
```

```
    return -1;
}

int checkEndFlight(int f) {
    int i, ff;

    if (StnCrewNum[FlightPara[f].ArrvStn] != 0) {//判断是否在起点了，是则 return
        return 1;//找到了
    }

    for (i = f + 1; i < FlightNum; i++) {
        ff = i;
        if (FlightPara[ff].DptrStn == FlightPara[f].ArrvStn && FlightPara[ff].RelaDptrTime -
FlightPara[f].RelaArrvTime >= MinCT) {//如果满足时间要求
            if (checkEndFlight(ff) == 1) {
                return 1;
            }//搜寻 ff
        }
    }
    return -1;
}

void read_initial(string inFile_1, string inFile_2) {
    int i, j, k, date, time, l;
    int dayFlightNum[32];
    for (i = 1; i < 32; i++) {
        dayFlightNum[i] = 0;
    }
    //读取 inFile_A
    vector<vector<string>> data_A;
    data_A = read_csv(inFile_1);
    //读取 inFile_B
    vector<vector<string>> data_B;
    data_B = read_csv(inFile_2);

    for (i = 0; i < CrewNum; i++) {
        CrewPara[i].Degree[2] = atoi(data_A[i + 1][1].c_str());
        CrewPara[i].Degree[1] = atoi(data_A[i + 1][2].c_str());
        CrewPara[i].Base = atoi(data_A[i + 1][4].c_str());
        CrewPara[i].DutyCostPerHr = atoi(data_A[i + 1][5].c_str());
        CrewPara[i].ParingCostPerHr = atoi(data_A[i + 1][6].c_str());
        CrewPara[i].CurrentStn = CrewPara[i].Base;//初始在 Base
        CrewPara[i].lastFlight = -1;//上一航班
```

```
        CrewPara[i].CurrentStatus = 1;//初始为工作状态
        CrewPara[i].CurrentDutyTime = 0;
        CrewPara[i].CurrentFlyTime = 0;
        CrewPara[i].CurrentPairingDay = 0;//当前任务环时间
        CrewPara[i].TotalDutyTime = 0;
        CrewPara[i].ConDutyDay = 0;//当前连续执勤天数
        CrewPara[i].ConRestDay = 0;
        CrewPara[i].TotalPairingTime = 0;//任务环时间，不能大于 14400
        CrewParingTime[CrewNum] = 0;
        CrewDutyTime[CrewNum] = 0;
        CrewFlyTime[CrewNum] = 0;

        if (CrewPara[i].Degree[2] == 1 && CrewPara[i].Degree[1] == 1) {
            CaptainFirstOfficer.push_back(i);
        }
        else if (CrewPara[i].Degree[2] == 1) {
            Captain.push_back(i);
        }
        else if (CrewPara[i].Degree[1] == 1) {
            FirstOfficer.push_back(i);
        }

        for (j = 0; j < 32; j++) {
            CrewDayStatus[i][j] = 0;//一开始都是休息状态，不知道今天能不能上班
        }
        StnCrewNum[CrewPara[i].Base]++;
    }

    for (i = 0; i < StnNum; i++) {
        for (j = 0; j < StnNum; j++) {
            StnFlow[i][j] = 0;
        }
    }

    for (i = 0; i < FlightNum; i++) {
        vector<int> sv;
        split(data_B[i + 1][1], sv, '/');
        FlightPara[i].DptrDate = sv[1];
        split(data_B[i + 1][2], sv, ':');
        FlightPara[i].DptrTime = sv[0] * 60 + sv[1];
        split(data_B[i + 1][4], sv, '/');
        FlightPara[i].ArrvDate = sv[1];
        split(data_B[i + 1][5], sv, ':');
        FlightPara[i].ArrvTime = sv[0] * 60 + sv[1];
```

```
        FlightPara[i].DptrStn = atoi(data_B[i + 1][3].c_str());
        FlightPara[i].ArrvStn = atoi(data_B[i + 1][6].c_str());
        FlightPara[i].RelaDptrTime = (FlightPara[i].DptrDate - startDate) * 24 * 60 +
FlightPara[i].DptrTime;
        FlightPara[i].RelaArrvTime = (FlightPara[i].ArrvDate - startDate) * 24 * 60 +
FlightPara[i].ArrvTime;
        FlightPara[i].LegTime = (FlightPara[i].ArrvDate - FlightPara[i].DptrDate) * 24 * 60 +
(FlightPara[i].ArrvTime - FlightPara[i].DptrTime);
        FlightCrewNum[i][0] = FlightCrewNum[i][1] = FlightCrewNum[i][2] = 0;
        FlightPara[i].isFly = 0;//初始状态，都不能起飞
        FlightPara[i].FltNum = atoi(data_B[i + 1][0].substr(2, data_B[i + 1][0].size() - 1).c_str());
        StnFlow[FlightPara[i].DptrStn][FlightPara[i].ArrvStn]++;
        dayFlightNum[FlightPara[i].DptrDate]++;
        FlightBackP[i][0] = FlightBackP[i][1] = 0;//回来的可能性都为 0
        FlightCanFly[i] = 1;//初始化都能起飞
    }

    //对航班进行排序
    for (i = 0; i < FlightNum - 1; i++) {
        for (j = i + 1; j < FlightNum; j++) {
            if (FlightPara[i].RelaDptrTime > FlightPara[j].RelaDptrTime) {
                exchangeFlight(i, j);
            }
            else if (FlightPara[i].RelaDptrTime == FlightPara[j].RelaDptrTime) {
                if (FlightPara[i].RelaArrvTime > FlightPara[j].RelaArrvTime) {
                    exchangeFlight(i, j);
                }
            }
        }
    }

    //对于每一个航班判断能否起飞
    int f;
    for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
        if (StnCrewNum[FlightPara[i].DptrStn] == 0) {
            f = checkStartFlight(i);
            if (f == -1) {
                FlightCanFly[i] = 0;
            }
        }
    }

    for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
        if (StnCrewNum[FlightPara[i].ArrvStn] == 0) {
```

```
                f = checkEndFlight(i);
                if (f == -1) {
                    FlightCanFly[i] = 0;
                }
            }
        }
        int dt, ar;
        for (i = 0; i < FlightNum; i++) {//能找到从基地出发的就停止
            ar = FlightPara[i].ArrvStn;
            FlightBackP[i][0] = StnFlow[ar][0];
            FlightBackP[i][1] = StnFlow[ar][1];
        }
    }

    int timeIsOk(int c, Flight f) {//判断时间是否够，加入任务环
        int l;
        //若飞行员处于没有工作过的状态则直接可行
        if (CrewPara[c].lastFlight != -1) {//若飞行员工作过
            l = CrewPara[c].lastFlight;
            //任务环超过 MaxTAFB，或者当天强制休息了
            if (CrewDayStatus[c][f.DptrDate - startDate] == 2 || CrewPara[c].TotalPairingTime +
    (f.RelaArrvTime - FlightPara[l].RelaArrvTime) > MaxTAFB) {
                return -1;
            }
            //若当天可用
            if (CrewPara[c].CurrentStatus == 1) {//当前为工作状态
                //判断飞行时间超过 MaxBlk, 执勤时间超过 MaxDP，不满足连接时间
                if ((f.RelaDptrTime < FlightPara[l].RelaArrvTime + MinCT) ||
    (CrewPara[c].CurrentFlyTime + f.LegTime > MaxBlk) || (CrewPara[c].CurrentDutyTime +
    (f.RelaArrvTime - FlightPara[l].RelaArrvTime) > MaxDP)) {
                    return -1;
                }
            }
            else {//当前处于休息状态，当天的第一班航班，判断是否休息足够长的时间
                if (f.RelaDptrTime < FlightPara[l].RelaArrvTime + MinRest) {
                    return -1;
                }
            }
        }
        return c;
    }

    int find_Captain(int f) {
        int i, j, l, c, best_c, best_delta, delta;
```

```
    int flag = 0;
    best_c = -1;
    best_delta = 999999;
    for (i = 0; i < CaptainFirstOfficer.size(); i++) {
        //飞行员当前所在地与航班出发点相同，飞行员有正机长和副机长资格
        c = CaptainFirstOfficer[i];
        if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
            flag = timeIsOk(c, FlightPara[f]);
            if (flag != -1) {
                if (best_c == -1) {
                    best_c = c;
                }
                else if (CrewPara[c].TotalPairingTime > CrewPara[best_c].TotalPairingTime)
{
                    best_c = c;
                }
            }
        }
    }
    if (best_c == -1) {
        for (i = 0; i < Captain.size(); i++) {
            //飞行员当前所在地与航班出发点相同，飞行员只有正机长资格
            c = Captain[i];
            if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                flag = timeIsOk(c, FlightPara[f]);
                if (flag != -1) {
                    if (best_c == -1) {
                        best_c = c;
                    }
                    else if (CrewPara[c].TotalPairingTime >
CrewPara[best_c].TotalPairingTime) {
                        best_c = c;
                    }
                }
            }
        }
    }

    return best_c;//没有找到可用机长
}

int find_FirstOfficer(int f) {
    int i, j, l, c, best_c, delta, best_delta;
    int flag = 0;
```

```
    best_c = -1;
    best_delta = 999999;
    for (i = 0; i < FirstOfficer.size(); i++) {//飞行员只有副机长资格
        //飞行员当前所在地与航班出发点相同
        c = FirstOfficer[i];
        if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {//判断是否在当前机场
            flag = timeIsOk(c, FlightPara[f]);
            if (flag != -1) {
                if (best_c == -1) {
                    best_c = c;
                }
                else if (CrewPara[c].TotalPairingTime > CrewPara[best_c].TotalPairingTime)
{
                    best_c = c;
                }
            }
        }
    }
    if (best_c == -1) {
        for (i = 0; i < CaptainFirstOfficer.size(); i++) {
            //飞行员当前所在地与航班出发点相同，飞行员有正机长和副机长资格
            c = CaptainFirstOfficer[i];
            if (CrewPara[c].CurrentStn == FlightPara[f].DptrStn) {
                flag = timeIsOk(c, FlightPara[f]);
                if (flag != -1) {
                    if (best_c == -1) {
                        best_c = c;
                    }
                    else if (CrewPara[c].TotalPairingTime >
CrewPara[best_c].TotalPairingTime) {
                        best_c = c;
                    }
                }
            }
        }
    }

    return best_c;//没有找到可用副机长
}

void updateCrew(int c, int f, int degree, int status) {//更新，如何更新
    CrewFlight[c].push_back(f);
    CrewDegree[c].push_back(degree);
    CrewDayStatus[c][FlightPara[f].DptrDate - startDate] = 1;//该天安排了航班就要算执勤
```

```
//计算执勤时间
if (status == 0) {//如果这是今天最后一班航班
    if (CrewPara[c].CurrentStatus == 0 || CrewPara[c].CurrentStatus == 2 || CrewPara[c].lastFlight == -1) {//第一天从基地起飞或者刚休息完
        CrewPara[c].TotalDutyTime += FlightPara[f].LegTime;
    }
    else {//处于执勤状态下
        CrewPara[c].TotalDutyTime += FlightPara[f].RelaArrvTime - FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
    }
    CrewPara[c].CurrentDutyTime = 0;
    CrewPara[c].CurrentFlyTime = 0;
}
else {//如果不知道是不是最后一班航班
    if (CrewPara[c].CurrentStatus == 0 || CrewPara[c].CurrentStatus == 2 || CrewPara[c].lastFlight == -1) {//第一天从基地起飞或者刚休息完
        CrewPara[c].TotalDutyTime += FlightPara[f].LegTime;
        CrewPara[c].CurrentDutyTime += FlightPara[f].LegTime;
    }
    else {//处于执勤状态下
        CrewPara[c].TotalDutyTime += FlightPara[f].RelaArrvTime - FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
        CrewPara[c].CurrentDutyTime += FlightPara[f].RelaArrvTime - FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
    }
    CrewPara[c].CurrentFlyTime += FlightPara[f].LegTime;
}
if (CrewPara[c].lastFlight == -1) {//如果是第一次工作
    CrewPara[c].ConDutyDay++;//执勤天数+1
    CrewPara[c].CurrentPairingDay++;//任务环天数+1
}
else if (CrewPara[c].lastFlight != -1) {//如果不是第一次工作
    if (FlightPara[f].DptrDate == FlightPara[CrewPara[c].lastFlight].DptrDate + 1) {//连续的一天
        CrewPara[c].ConDutyDay++;//执勤天数+1
    }
    else if (FlightPara[f].DptrDate > FlightPara[CrewPara[c].lastFlight].DptrDate + 1) {//跳了一天
        CrewPara[c].ConDutyDay = 0;//执勤天数重新计算
        CrewPara[c].ConDutyDay++;//执勤天数+1
    }
    if (CrewStatus[c][CrewStatus[c].size() - 1] == 2) {
        CrewPara[c].CurrentPairingDay++;//任务环天数+1
    }
```

```
            else {
                CrewPara[c].CurrentPairingDay += FlightPara[f].DptrDate -
FlightPara[CrewPara[c].lastFlight].DptrDate;//任务环天数+1
            }

        }
        //计算任务环时间
        if (CrewPara[c].CurrentStatus == 2 || CrewPara[c].lastFlight == -1) {//如果是刚休假完
            CrewPara[c].TotalPairingTime += FlightPara[f].LegTime;
        }
        else {
            CrewPara[c].TotalPairingTime += FlightPara[f].RelaArrvTime -
FlightPara[CrewPara[c].lastFlight].RelaArrvTime;
        }
        CrewStatus[c].push_back(status);//当天结束后的状态
        CrewConDutyDay[c].push_back(CrewPara[c].ConDutyDay);
        CrewConPairingDay[c].push_back(CrewPara[c].CurrentPairingDay);
        CrewPara[c].CurrentStatus = status;//当前工作状态
        CrewPara[c].CurrentStn = FlightPara[f].ArrvStn;//更新当前位置
        CrewPara[c].lastFlight = f;
        FlightCrewNum[f][degree] += 1;
}

int calculateCrew(int crew, int flight1, int flight2, int degree) {
        int i, j, l;
        int cost = 0;
        if (CrewPara[crew].lastFlight != -1) {//对于不是首次飞行的
            l = CrewPara[crew].lastFlight;
            if (FlightPara[flight2].DptrDate == FlightPara[flight1].DptrDate) {//同一天
                if (CrewPara[crew].CurrentDutyTime + FlightPara[flight2].RelaArrvTime -
FlightPara[l].RelaArrvTime > MaxDP) {//超过最大执勤时间
                    return 9999999;
                }
                if (CrewPara[crew].CurrentFlyTime + FlightPara[flight2].LegTime +
FlightPara[flight1].LegTime > MaxBlk) {//超过最大飞行时间
                    return 9999999;
                }
            }
            else {
                if (FlightPara[flight2].RelaDptrTime - FlightPara[flight1].RelaArrvTime < MinRest)
{//休息时间不够
                    return 9999999;
                }
            }
```

```
            if (CrewPara[crew].TotalPairingTime + FlightPara[flight2].RelaArrvTime -
FlightPara[l].RelaArrvTime > MaxTAFB) {
                return 9999999;
            }
        }
        else if (CrewPara[crew].lastFlight == -1) {//对于首次飞行的
            if (FlightPara[flight2].DptrDate == FlightPara[flight1].DptrDate) {//同一天
                if (CrewPara[crew].CurrentDutyTime + FlightPara[flight2].RelaArrvTime -
FlightPara[flight1].RelaDptrTime > MaxDP) {//超过最大执勤时间
                    return 9999999;
                }
                if (CrewPara[crew].CurrentFlyTime + FlightPara[flight2].LegTime +
FlightPara[flight1].LegTime > MaxBlk) {//超过最大飞行时间
                    return 9999999;
                }
            }
            else {
                if (FlightPara[flight2].RelaDptrTime - FlightPara[flight1].RelaArrvTime < MinRest)
{//休息时间不够
                    return 9999999;
                }
            }
            if (CrewPara[crew].TotalPairingTime + FlightPara[flight2].RelaArrvTime -
FlightPara[flight1].RelaArrvTime > MaxTAFB) {
                return 9999999;
            }
        }

        if (degree == 1 && CrewPara[crew].Degree[2] == 1) {//触发替补
            cost += 100000;
        }

        cost += CrewPara[crew].TotalDutyTime;//执勤时间越少越可能被选中

        return cost;
}

int repair(int f, int degree) {
        int i, j, best_c, best_f, flag, f_c, best_f_c;
        best_f_c = 9999999;
        best_c = best_f = -1;
        flag = 0;
        for (i = 0; i < f; i++) {//对于所有比该航班要早的航班
            //如果该航班还坐得下，且该航班已经满足机组配置，且落地地点是该机场
```

```
        if (FlightCrewNum[i][0] < 5 && FlightPara[i].isFly == 1 && FlightPara[i].ArrvStn ==
FlightPara[f].DptrStn) {
            //航班可以在下一航班起飞前到达
            if (FlightPara[i].RelaArrvTime + MinCT <= FlightPara[f].RelaDptrTime) {//航班到
达时间和地点符合
                for (j = 0; j < CrewNum; j++) {//找一个职级满足要求的，且在该航班出发
机场
                    if ((CrewPara[j].Degree[degree] == 1) && FlightPara[i].DptrStn ==
CrewPara[j].CurrentStn) {//这里有点问题，需要判断等级
                        flag = timeIsOk(j, FlightPara[i]);//确定不会超执勤时间
                        if (flag != -1) {//找到该人，对该人进行评价，是否合适
                            f_c = calculateCrew(j, i, f, degree);//j 乘坐 i 去开 f
                            //cout << f_c << endl;
                            if (f_c < best_f_c) {//挑选最小的
                                best_c = j;
                                best_f = i;
                                best_f_c = f_c;
                            }
                        }
                    }
                }
            }
        }

        if (best_c != -1 && best_f != -1) {
            if (FlightPara[f].DptrDate > FlightPara[best_f].DptrDate) {
                updateCrew(best_c, best_f, 0, 0);//使其坐过来，飞完休息
            }
            else {
                updateCrew(best_c, best_f, 0, 1);//使其坐过来，飞完休息
            }

            return best_c;
        }
        return -1;
}

void clearFlight(int c, int flag) {//删除航班,需针对任务环进行改进
        int l = 0, f;
        while (l < flag) {//回溯的次数
            CrewFlight[c].pop_back();
            CrewDegree[c].pop_back();
            CrewStatus[c].pop_back();
```

```
            CrewConDutyDay[c].pop_back();
            CrewConPairingDay[c].pop_back();
            f = CrewPara[c].lastFlight;
            CrewPara[c].CurrentStn = FlightPara[f].DptrStn;//更新当前位置
            if (CrewFlight[c].size() > 0) {
                CrewPara[c].CurrentFlyTime = CrewPara[c].CurrentFlyTime -
FlightPara[f].LegTime;
                CrewPara[c].CurrentStatus = CrewStatus[c][CrewStatus[c].size() - 1];
                CrewPara[c].lastFlight = CrewFlight[c][CrewFlight[c].size() - 1];
                CrewPara[c].ConDutyDay = CrewConDutyDay[c][CrewConDutyDay[c].size() - 1];
                CrewPara[c].CurrentPairingDay =
CrewConPairingDay[c][CrewConPairingDay[c].size() - 1];
                if (CrewPara[c].CurrentStatus == 0) {
                    CrewPara[c].CurrentDutyTime = 0;
                }
                else {
                    CrewPara[c].CurrentDutyTime = CrewPara[c].CurrentDutyTime -
(FlightPara[f].RelaArrvTime - FlightPara[CrewPara[c].lastFlight].RelaArrvTime);
                }
                if (CrewStatus[c][CrewStatus[c].size() - 1] == 2) {//休假状态
                    CrewPara[c].TotalPairingTime = CrewPara[c].TotalPairingTime -
FlightPara[f].LegTime;
                }
                else {//非休假状态
                    CrewPara[c].TotalPairingTime = CrewPara[c].TotalPairingTime -
(FlightPara[f].RelaArrvTime - FlightPara[CrewPara[c].lastFlight].RelaArrvTime);
                }
            }
            else {//还未出发
                CrewPara[c].lastFlight = -1;
                CrewPara[c].CurrentDutyTime = 0;
                CrewPara[c].CurrentFlyTime = 0;
                CrewPara[c].CurrentStatus = 1;
                CrewPara[c].ConDutyDay = 0;
                CrewPara[c].TotalPairingTime = 0;
                CrewPara[c].CurrentPairingDay = 0;
            }
            l++;
        }
    }

    void updateCrewStatus() {//每当新的一天
        int i, status;
        //需要判断哪些人休息
```

```
    for (i = 0; i < CrewNum; i++) {
        if (CrewFlight[i].size() > 0) {//对于有航班的
            if (CrewPara[i].CurrentPairingDay > 1 && CrewPara[i].CurrentStn ==
CrewPara[i].Base) {//直接休假
                if (CrewPara[i].CurrentPairingDay == 1) {
                    l1++;
                }
                else if (CrewPara[i].CurrentPairingDay == 2) {
                    l2++;
                }
                else if (CrewPara[i].CurrentPairingDay == 3) {
                    l3++;
                }
                else if (CrewPara[i].CurrentPairingDay == 4) {
                    l4++;
                }
                CrewDayStatus[i][currentDate - startDate] = 2;
                CrewDayStatus[i][currentDate - startDate + 1] = 2;//休假两天
                CrewPara[i].CurrentStatus = 2;//改为休假模式
                CrewStatus[i][CrewStatus[i].size() - 1] = 2;
                CrewPara[i].CurrentPairingDay = 0;//当前任务环
                CrewPara[i].ConDutyDay = 0;//执勤天数清零
            }
            else if (CrewPara[i].ConDutyDay == MaxSuccOn) {//连续执勤已达 4 天
                CrewDayStatus[i][currentDate - startDate] = 2;//休息一天
                CrewPara[i].CurrentStatus = 0;//改为休假模式
                CrewPara[i].ConDutyDay = 0;//执勤天数清零
            }

            CrewPara[i].CurrentDutyTime = 0;
            CrewPara[i].CurrentFlyTime = 0;
            CrewPara[i].CurrentStatus = 0;//当前状态为休息状态
            if (CrewStatus[i][CrewStatus[i].size() - 1] != 2) {
                CrewStatus[i][CrewStatus[i].size() - 1] = 0;//休息状态
            }
        }
    }
}

void random_initial() {
    int i, j, k;
    int c, f, flag = 0;
    l1 = l2 = l3 = l4 = 0;
    for (i = 0; i < FlightNum; i++) {
```

```
            f = c = -1;
            if (FlightPara[i].DptrDate > currentDate) {//若到了第二天，更新状态
                updateCrewStatus();
                currentDate = FlightPara[i].DptrDate;
            }
            flag = 1;
            if (FlightCrewNum[i][1] == 0) {//寻找副机长
                f = find_FirstOfficer(i);//寻找副机长
                if (f == -1) {//如果没有找到副机长
                    f = repair(i, 1);//从其他地方调一个副机长过来
                    flag = 2;//经历了修复过，需要多回溯一次
                }
                if (f != -1) {
                    updateCrew(f, i, 1, 1);//找到了则更新该飞行员的航班
                    FlightCrewNum[i][1] = 1;
                }
            }
            if (f != -1 && FlightCrewNum[i][2] == 0) {//寻找到副机长了，则可以开始寻找机长
                c = find_Captain(i);//寻找机长
                if (c == -1) {//如果没有找到机长
                    c = repair(i, 2);//从其他地方调一个机长过来
                }
                if (c != -1) {
                    updateCrew(c, i, 2, 1);//找到了则更新
                    FlightCrewNum[i][2] = 1;
                }
                else {
                    clearFlight(f, flag);//找不到需要回溯
                }
            }

            if (c != -1) {//配置成功
                FlightPara[i].isFly = 1;
            }
        }
    }

    int main()
    {
        int i, j;
        srand((unsigned)time(NULL));
        read_initial(inFile_A, inFile_B);//读取数据
        startTime = clock();//计时开始
        endTime = startTime;
```

```
    random_initial();
    endTime = clock();
    cout << (double)(endTime - startTime) / CLOCKS_PER_SEC << endl;
    return 0;
}
```