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	北京航空航天大学
参赛队号	**21100060045**
队员姓名	**1.于晓飞**
	2.王飞龙
	3.王巧家

中国研究生创新实践系列大赛
“华为杯”第十八届中国研究生
数学建模竞赛

题 目 航空公司机组优化排班问题

摘 要：

航空公司机场排班问题是一个传统的组合优化问题，在复杂多变的航空组织运营环境下，为航空公司高效、合理地制定机组排班任务对于实现资源高效配置，提高运输效率以及降低运营成本尤为重要。对于本题设定情景的机场排班问题，本文首先采用线性加权的方法对多目标的目标函数进行处理，构建混合整数规划模型来精确描述问题，并设计启发式算法来对模型进行求解，最终编程实现算法完成了给定算例的求解和结果的分析。

对于问题一，本文首先对目标①④⑦进行线性加权处理，针对加权后的目标函数和问题一对应的 3 条约束条件建立混合整数规划模型；针对模型提出贪婪初始解构造和LS(Local Search)解优化的思路。其中贪婪初始解构造需要注意极端情况下，某些航班人员可以乘坐其他航班回到基地。在 LS 解优化设计过程中，本文创新性地针对航班规划的问题特点设计了机组人员“解编”和“组编”的启发式优化思想，通过机组人员的解编和组编来完成机组人员在不同航班环之间的互换，从而实现不同航班环的破坏和修复的过程。最终通过编写代码实现算法，得到的结果如下：A 套数据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、替补资格使用次数分别为 4、202、0 和 48；B 套数据相应的结果为 7631、6323、0 和 3806。

对于问题二，本文建立模型二对模型一进行补充，在模型一的基础之上增加执勤相关约束和目标的考量。模型二首先对目标①②④⑤⑦进行线性加权处理，针对加权后的目标函数和问题二对应的 5 条约束对于模型一进行扩充，增加模型一中混合整数规划模型的约束和目标函数；在初始解构建过程中需要额外增加对于执勤相关约束的考量；在 LS 优化的过程中，除了需要考量执勤的相关因素之外，对每次局部搜索的成本还需要考量新增加的执勤成本。最终通过编写代码实现算法，得到的结果如下：A 套数据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、替补资格使用次数分别为 73、133、0 和 68；机组利用率为 96.57%；机组人员的最大、平均和最小执勤天数分别为 1.83、1.08 和 1.58；总体执勤成本为 28.51 万元。B 套数据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、替补资格使用次数分别为 9162、4792、0 和 2850；机组利用率达到 97.81%；最大、平均和最小执勤天数分别为 6.67、0.75 和 1.60；总体执勤成本为 1011.23 万元。

对于问题三，本文建立模型三对模型二进行补充，在模型二的基础之上新增任务环相关约束和目标。模型三首先对目标①②③④⑤⑥⑦进行线性加权处理，针对加权后的目标函数和问题三对应的 3 条约束对于模型二进行扩充，增加模型二中混合整数规划模型的约束和目标函数；在初始解构建过程中需要额外增加对于任务环相关约束的考量；在 LS 优

化的过程中，除了需要考量任务环的相关因素之外，对每次局部搜索的成本还需要考量新增加的任务环成本。最终通过编写代码实现算法，得到的结果如下：A 套数据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、替补资格使用次数分别为 4、202、0 和 48；机组利用率为 100%；机组人员的最大、平均和最小执勤天数分别为 2.33、1.08 和 1.76；总体执勤成本为 43.39 万元；总体任务环成本为 0.08 万元。B 套数据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、替补资格使用次数分别为 6836、7118、0 和 4391；机组利用率达到 99.47%；机组人员的最大、平均和最小执勤天数分别为 3.58、0.75 和 1.62；总体执勤成本为 1448.05 万元；总体任务环成本为 5.12 万元。

针对机组优化排班问题，本文构建的启发式算法框架思路简洁，并创新性地提出了基于 LS 的 “解编和组编”优化思路，具有较强的适应性和推广性；构建的算法能在短时间内取得较为理想的解方案，实用性较强，可以为之后的机场排班算法提供借鉴意义。对于更大规模和复杂的问题，模型求解的精确性还需要进一步研究。

关键词：多目标优化；混合整数规划模型；贪婪构造；LS 算法；解编；组编；模拟退火算法

目录

1. 问题重述

1.1 问题背景

民航业是航空运输的重要组成部分，其发展水平在一定程度上反映国家和地区的现代化水平、经济结构等。随着航空市场规模逐步扩大，航班数量和机组人员数量迅速增加，航空公司面临着高效规划航班任务和管理各项资源的巨大压力[1]。因此，航空公司在复杂多变的航空组织运营环境下，合理制定机组排班任务显得尤为重要，从而实现资源高效调配，运输效率提高以及运营成本降低。

编排机组任务时，需要遵循以下约束规则：

1. 每个机组人员初始从基地出发并最终回到基地；
2. 每个机组人员的下一航段的起飞机场必须和上一航段的到达机场一致；
3. 每个机组人员相邻两个航段之间的连接时间不小于 MinCT 分钟；
4. 每个机组人员每天至多只能执行一个执勤；
5. 每次执勤的飞行时间最多不超过 MaxBlk 分钟；
6. 每次执勤的时长最多不超过 MaxDP 分钟；
7. 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；
8. 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；
9. 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；
10. 每个机组人员相邻两个任务环之间至少有 MinVacDay 天休息；
11. 每个机组人员连续执勤天数不超过 MaxSuccOn 天；
12. 每趟航班最多乘机人数为 5 人。

将机场排班问题抽象出一个网络结构，如下图 1.1 所示。其中机组人员在规划初始均位于基地，如图中的基地 1 和基地 2，之后机组人员之间进行任意组合对任意可行航班环如图中 1-3-4-5-1 进行服务，最终航班规划结束后所有机组人员均要回到基地。

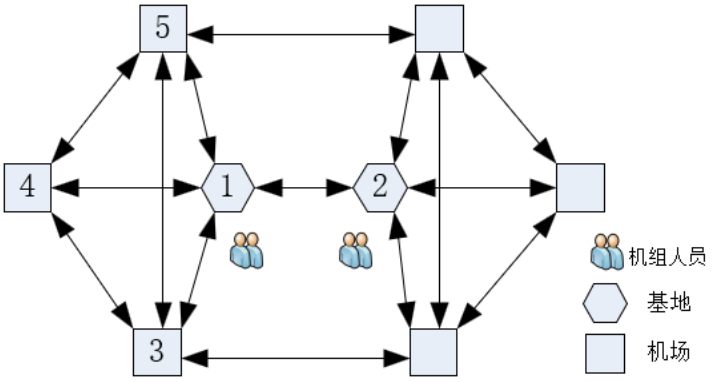


图 1.1 机场排班计划网络图

1.2 待解决的问题

本题目的宗旨是在明确表达飞行时间、执勤时间、休息时间等约束的基础上，建立机组排班线性优化模型，并在给定的两组数据集下，求解如下问题：

问题一：在限制条件 1~3 的约束下，依编号次序考虑以下优化目标：

①.尽可能多的航班满足机组配置;

④.尽可能少的总体乘机次数；

⑦.尽可能少使用替补资格。

建立线性规划模型给航班分配机组人员，设计算法程序按所给数据运行，输出求解结果。

问题二：引进执勤概念。假定每个机组人员的每单位小时执勤成本给定，在限制条件 1~8 的约束下，依编号次序考虑以下优化目标：

①.尽可能多的航班满足机组配置;

②.机组人员的总体执勤成本最低；

④.尽可能少的总体乘机次数；

⑤.机组人员之间的执勤时长尽可能平衡；

⑦.尽可能少使用替补资格。

建立线性规划模型给航班分配机组人员，设计算法程序按所给数据运行，输出求解结果。

问题三：编制排班计划。假定每个机组人员的每单位小时任务环成本给定，在限制条件 1~11 的约束下，依编号次序考虑以下优化目标：

①.尽可能多的航班满足机组配置;

②.机组人员的总体执勤成本最低；

③.机组人员的总体任务环成本最低；

④.尽可能少的总体乘机次数；

⑤.机组人员之间的执勤时长尽可能平衡；

⑥.机组人员之间的任务环时长尽可能平衡；

⑦.尽可能少使用替补资格。

建立线性规划模型给航班分配机组人员，设计算法程序按所给数据运行，输出求解结果。

下表 1.1 列出三个问题需要求解的指标。

表 1.1 结果指标

编号	结果指标	问题一	问题二	问题三
1	不满足机组配置航班数	√	√	√
2	满足机组配置航班数	√	√	√
3	机组人员总体乘机次数	√	√	√
4	替补资格使用次数	√	√	√
5	机组总体利用率		√	√
6	最小/平均/最大 一次执勤飞行时长		√	√
7	最小/平均/最大 一次执勤时长		√	√
8	最小/平均/最大 机组人员执勤天数		√	√
9	一/二/三/四天 任务环数量分布			√
10	总体执勤成本（万元）		√	√
11	总体任务环成本（万元）			√
12	程序运行分钟数	√	√	√

2. 模型假设及符号说明

2.1 模型假设

(1) 每个机组人员有一个固定的基地，用所在机场表达；

(2) 假设机组人员有且仅有一个主要资格，也可以具备其他替补资格。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；

(3) 航班都是唯一的，当航班应用于机组人员排班时也叫航段；

(4) 每个航班有给定的最低机组资格配置，用格式 C<数字>F<数字>表示，代表正副机长的数量，只有满足最低机组资格配置才能起飞；

(5) 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；

(6) 机组人员之间可以任意组合；

(7) 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；

(8) 不满足最低机组资格配置的航班不能配置任何机组人员；

(9) 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。

2.2 符号说明

符号	说明
P	机场位置集合
N	总共需要的机组人员数，取值 2
N^c	正机长数，取值 1
N^f	副机长数，取值 1
F	航班集合
Dt_j	$j \in F$，航班 j 的出发时间
DS_j	$j \in F$，航班 j 的出发机场
At_j	$j \in F$，航班 j 的到达时间
AS_j	$j \in F$，航班 j 的到达机场
E	员工集合
G	员工执勤集合
H	员工任务环集合
M	执勤员工总数
T_{ij}	$i \in E$， $j \in G$，员工 i 第 j 次执勤开始时间
A_{ij}	$i \in E$， $j \in G$，员工 i 第 j 次执勤结束时间
Tp_{ij}	$i \in E$， $j \in H$，员工 i 第 j 个任务环开始日期时间
Ap_{ij}	$i \in E$， $j \in H$，员工 i 第 j 个任务环结束日期时间
CP_i	$i \in E$，员工 i 是否可担任正机长，取值 0，1
FO_i	$i \in E$，员工 i 是否可担任副机长，取值 0，1
Dh_i	$i \in E$，员工 i 是否允许乘机，取值 0，1

BS_i	$i \in E$，员工 i 的基地
Ft_{ij}	$i \in E$，员工 i 第 j 次执勤飞行时间
DT_i	$i \in E$，员工 i 的执勤总时间
DC_i	$i \in E$，员工 i 的单位小时执勤成本
Pt_{ij}	$i \in E$，员工 i 第 j 个任务环时长
PT_i	$i \in E$，员工 i 的任务环总时间
PC_i	$i \in E$，员工 i 的单位小时任务环成本
x_i	$i \in F$，航班 i 的机组配置是否满足要求
y_{ijda}	$i \in E$，$j \in G$，$d,a \in P$，员工 i 在航班 j 是否乘机从 d 机场前往 a 机场，取值 0，1
z_{ijda}	$i \in E$，$j \in G$，$d,a \in P$，员工 i 是否使用替补资格服务航班 j 从 d 机场前往 a 机场，取值 0，1
k_{ijda}	$i \in E$，$j \in G$，$d,a \in P$，员工 i 是否不使用替补资格服务航班 j 从 d 机场前往 a 机场，取值 0，1

2.3 数据说明

赛事共提供两套数据用于模型检测，其基本信息如下表 2.1 所示，A 套数据的开始时间为 2021 年 8 月 11 日至 2021 年 8 月 25 日；B 套数据的开始时间为 2021 年 8 月 1 日至 2021 年 8 月 31 日。可以直观地看出 A 套数据量明显小于 B 套数据，在后续进行模型验证与计算的时候需要保证优先运行 A 套数据的前提下，再进行 B 套数据的检测。

表 2.1 数据说明表

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

3. 问题一的建模与求解

3.1 问题一的描述与分析

根据主要任务可见，本题满足分布设计的思路。由于机组排班问题的特殊性，对于每个机组成员而言，一次完整的排班计划，是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；任务环是由一连串的执勤和休息时间组成；而执勤又是由一连串航班和间隔连接构成。具体的隶属关系可以用下图 3.1 表示。

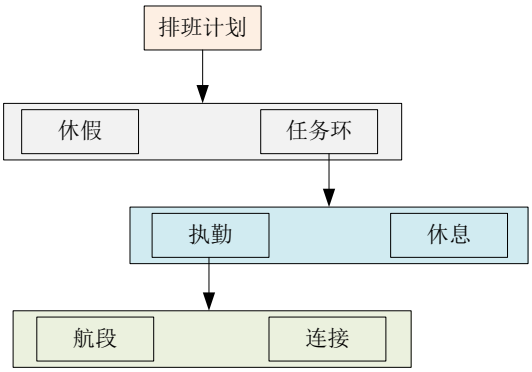


图 3.1 机组排班任务分解图

问题一是合理地为航班分配机组人员，保证一个航班能够达到最低的机组资格配置执行飞行任务，即满足正机长与副机长的人员数目需求。在此基础上实现尽可能多的航班满足机组配置、尽可能少的总体乘机次数和尽可能少使用替补资格的优化目标。

3.2 问题一模型的建立

（1）确定目标函数：航班机组人员的分配过程中，要尽可能使得更多的航班满足机组配置，从而达到航班执行飞行任务的基本要求，而且减少机组人员总体的乘机次数，以及尽可能少的使用正机长替补副机长执行飞行任务。根据上述目标进行建模，得到如下目标函数：

目标函数 1：最大化航班满足机组配置，

$$\max \sum_{i \in F} x_i \tag{3.1}$$

目标函数 2：最小化总体乘机次数，

$$\min \sum_{i \in E} \sum_{j \in F} \sum_{d=DS_j} \sum_{a=AS_j} y_{ijda} \tag{3.2}$$

目标函数 3：最小化替补资格使用次数，

$$\min \sum_{i \in E} \sum_{j \in F} \sum_{d=DS_j} \sum_{a=AS_j} z_{ijda} \tag{3.3}$$

其中，x_i表示航班i的机组配置是否满足要求，如式（3.4）所示；y_{ijda}表示员工i在航班j是否乘机从d机场前往a机场，如式（3.5）所示；z_{ijda}表示员工i是否使用替补资格服务航班j并乘机从d机场前往a机场，如式（3.6）所示。

$$x_i=\begin{cases}1, & \text{航班 } i \text{ 的机组配置满足要求}\\ 0, & \text{航班 } i \text{ 的机组配置不满足要求}\end{cases} \tag{3.4}$$

$$y_{ijda}=\begin{cases}1, & \text{员工 } i \text{ 在航班 } j \text{ 乘机从 } d \text{ 机场前往 } a \text{ 机场}\\ 0, & \text{员工 } i \text{ 在航班 } j \text{ 未乘机从 } d \text{ 机场前往 } a \text{ 机场}\end{cases} \tag{3.5}$$

$$z_{ijda}=\begin{cases}1, & \text{员工 } i\text{使用替补资格服务航班 } j\text{并乘机从 } d\text{机场前往 } a\text{机场}\\ 0, & \text{员工 } i\text{未使用替补资格服务航班 } j\text{并乘机从 } d\text{机场前往 } a\text{机场}\end{cases} \tag{3.6}$$

（2）确定模型的约束条件，该问题的约束如下：

约束 1：航班能够执行飞行任务的基本要求是其机组人员达到最低的配置标准，所以满足机组人员配置的航班满足最低的配置标准，

$$\sum_{i_1\in E}\sum_{\substack{i_2\in E\\ i_1\neq i_2}}(CP_{i_1}\cdot K_{i_1jda}+FO_{i_2}\cdot K_{i_2jda}+CP_{i_2}\cdot Z_{i_2jda})+N\cdot(1-x_j)=N,\ j\in F, d=DS_j, a=AS_j \tag{3.7}$$

约束 2：机组人员配置的航班需要满足正机长的配置要求，

$$\sum_{i\in E}CP_i\cdot K_{ijda}+N^c(1-x_j)=1, j\in F, d=DS_j, a=AS_j \tag{3.8}$$

约束 3：机组人员配置的航班需要满足副机长的配置要求，

$$\sum_{i\in E}(FO_i\cdot K_{ijda}+CP_i\cdot Z_{ijda})+N^f(1-x_j)=1,\ j\in F, d=DS_j, a=AS_j \tag{3.9}$$

约束 4：每个机组人员初始从基地出发，

$$\sum_{j\in F}\sum_{a=AS_j}Dh_i\cdot(y_{ijd_1a_1}+z_{ijd_1a_1}+k_{ijd_1a_1}), i\in E, d_1=BS_i \tag{3.10}$$

约束 5：每个机组人员终点回到基地，

$$\sum_{j\in F}\sum_{d=DS_j}Dh_i\cdot(y_{ijd_2a_2}+z_{ijd_2a_2}+k_{ijd_2a_2}), i\in E, a_2=BS_i \tag{3.11}$$

约束 6：保证每个机组人员下一航段起飞机场与上一航段到达机场一致，

$$Dh_i\cdot(y_{ij_1d_1a}+z_{ij_1d_1a}+k_{ij_1d_1a})=Dh_i\cdot(y_{ij_2d_2a}+z_{ij_2d_2a}+k_{ij_2d_2a}),$$

$$i\in E, j_1\in F, j_2\in F, j_1\neq j_2, d_1=DS_{j_1}, d_2=AS_{j_2}, a=AS_{j_1} \tag{3.12}$$

约束 7：保证每个机组人员相邻两个航段的连接时间不小于 MinCT 分钟，

$$\left[Dh_i\cdot(y_{ij_2d_2a}+z_{ij_2d_2a}+k_{ij_2d_2a})\cdot At_{j_2}-Dh_i\cdot(y_{ij_1d_1a}+z_{ij_1d_1a}+k_{ij_1d_1a})\cdot Dt_{j_1}\right]$$

$$+MinCT\left[2-(x_{j_1}+x_{j_2})\right]\geq MinCT \tag{3.13}$$

3.3 问题一模型的求解

机组排班问题考虑的实际因素众多，导致模型的复杂程度增加、算例规模庞大。为了突出模型的实用性，而不一味追求花费大量时间求解一个精确解，本文针对上述混合整数规划模型，直接设计启发式算法进行相应的求解工作。整体求解思路如下图 3.2 所示。

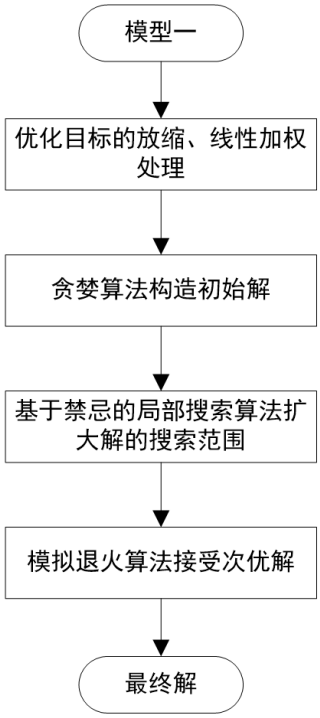


图 3.2 模型一求解思路

上述建立的机组排班模型存在三个优化目标，此类多目标决策问题是目前数学建模中比较普遍的一类问题，要求满足多个目标函数最优与决策变量的线性约束条件或非线性约束条件下进行求解[2]。多目标决策问题的处理方式有主要目标法、线性加权法、分层序列法、步骤法（Stem 法），本文将采用线性加权法。线性加权法的特点主要是实现将多个目标函数通过线性加权的方式集成到单个目标函数，那么问题转化为一般性的线性规划类问题，从而降低模型复杂度，提高模型求解效率。

本文使用线性加权处理多目标函数的具体转化方法如下：

步骤 1：对各个目标函数进行缩放处理。对于①目标，将每次优化得到的满足机组配置的航班数$\sum_{i\in F_n} x_i$除以总的航班数$|F_n|$作为①的目标函数；对于④目标，将每次优化得到的所有机组人员的所有乘机次数除以航班数$|F_n|$*机组人员数$|E_n|$作为④的目标函数；对于⑦目标，将每次优化得到的所有机组人员的所有替补次数除以航班数$|F_n|$*机组人员数$|E_n|$作为⑦的目标函数；

步骤 2：对于缩放后的目标函数进行加权处理，权重分配按照题目中所给的顺序进行依次递减分配；由于问题一种只考虑上述三个目标，所以按照 5，3，2 的比例分别作为这三个目标的不同权重，之后将上述缩放之后的目标进行加权作为新的目标函数，将多目标问题转化为单目标问题进行求解。

根据现有通用的算法，可以采用两阶段的方法来进行求解机组排班问题。第一阶段首先枚举出所有可行的航班环，第二阶段将机组人员进行航班环的指派。这种思路优点在于思路简单易行，但是缺点明显，首先不能充分利用机组人员，某些机组人员可能出现闲置的情况；其次这种思路不能让机组人员在航班途中进行交叉组编服务后续航段，缩小了寻优空间，缺少灵活性。使用枚举算法求解机场排版问题的思路可以如下表 3.1 所示。

表 3.1 基于枚举算法的机场排班求解

```
1：输入航班排班表
2：初始化首个航班 i=1，下一个航班 j=i+1
3：for 依次遍历首个航班 i
4：      j = j+1;
5：      if 航班 j 的出发时间与 i 的到达站相同 then
6：          if 航班 j 的出发时间与航班 i 的达到时间差大于 40min，且同一天 then
7：              选出航班 j 的集合 Table，航班 i 的飞行时长 Ft1，航班 j 的出发与 i 的到达时间差 Ft2
8：              if Ft1≤MaxBlk && Ft1+Ft2≤MaxDP then
9：                  i=j
10：                 return
11：             else
12：                 输出到执勤航班序列中
13：             end if
14：         else
15：             判断 j 的后一个航班，j=j+1
16：         end if
17：     else 判断 j 的后一个航班，j=j+1
18：     end if
19：输出所有执勤航班
```

基于枚举算法处理上述所说的问题之外，最大的问题在于当问题规模变大时，在枚举可行航班环的时候需要耗费的时间过长，因此对于大规模算例并不是优质的解法。因此本文设计如下启发式算法进行构造和优化求解。

3.3.1 贪婪构造初始解

本文首先基于贪婪的思想构造初始解，具体流程如下图 3.3 所示。在构造初始解时在每一个机组人员基地将所有的机组人员进行划分，按照最低资格配置划分出最大的机组人员组合数，若最后剩余机组人员不能满足最低资格配置，则将剩余的机组人员划归到最后一组满足最低资格配置的机组人员之中。

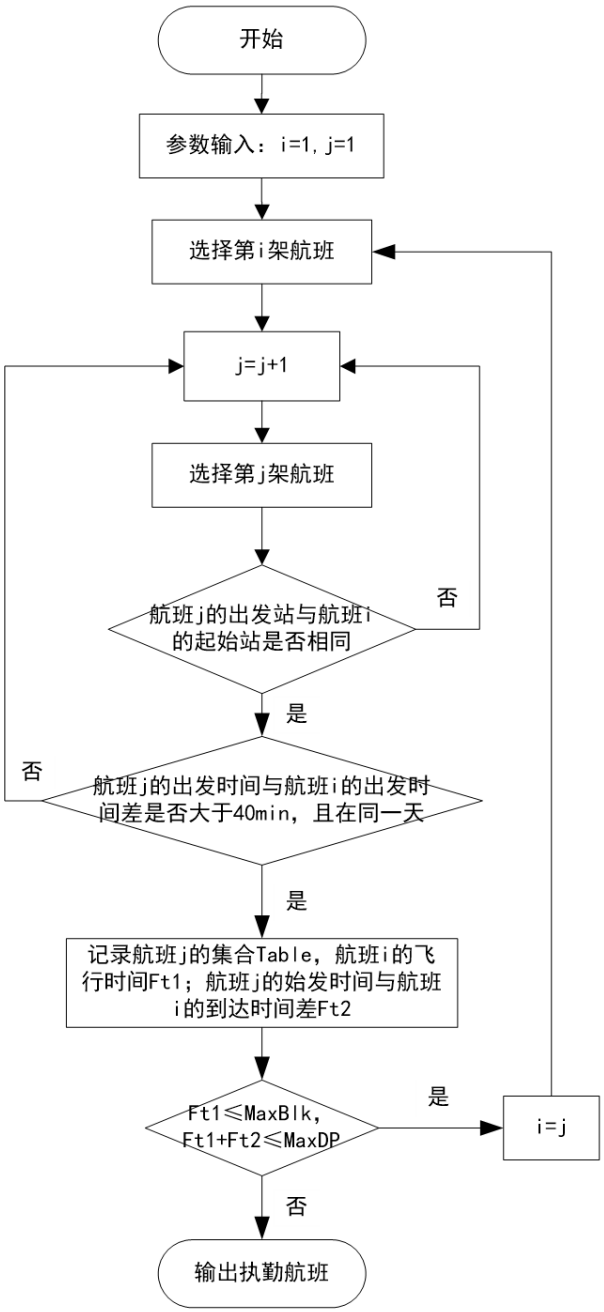


图 3.3 贪婪算法构造初始解的流程图

在划分完机组人员之后，对于每一个划分的机组人员组成的小组，进行贪婪构造航班环。贪婪构造的思路如下：选择从当前场站可以出发的航班之中，某到达机场所剩余的航班数量最多的航班作为机组人员服务的航班，在向后寻找的过程中，若在某个机场的某一个航班可以返回到机组人员的基地，则将之记录下来，并且只需要记录最后一个可以返回机组人员基地的机场位置即可。当机组人员向后服务到某个机场之后，没有可以继续服务的航班，则首先判断从当前机场是否可以回到基地，若不能，则寻找之前记录的最后一个可以回到基地的机场，机组人员从可以返回基地的机场出发，返回基地。上述为基于贪婪思想的一个最低配置机组人员的航班环构建过程。在构建完一个航班环之后，将此航班环服务完的航班进行记录，在后续航班环构建时，只能服务之前没有服务过的航班。将所有的最低配置机组人员均进行上述贪婪构建之后形成初始解。

初始解构造极端情况：若在构造初始解的过程中，某个航班环中没有可行航班可以回到机组人员的基地，则机组人员在最后一个完成航班的机场等待。在所有航班环全部构建完毕之后，将不能返回基地的机组人员用其他已经构建完成的航班环进行提供“乘机”服务，由其他航班环将这些机组人员带回基地。

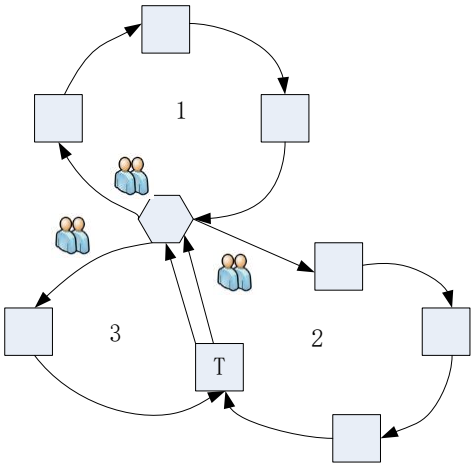


图 3.4 初始解构建示意图

初始解解构建如上图 3.4 所示，其中航班环 1 和航班环 2 可以正常构造完毕，但是航班环 3 在机场 *T* 时没有剩余航班可以回到基地，也没有其余可以服务的航班，则航班环 3 中的机组人员只能乘坐航班 2 的飞机从机场 T 返回到基地。

3.3.2 基于 LS 的优化算法

本文基于带禁忌(Tabu Search)规则的局部搜索(Local Search)算法和模拟退火(Simulated Annealing)算法来设计初始解的优化思路。

局部搜索是解决最优化问题的一种启发式算法。对于计算起来非常复杂的最优化问题，要找到最优解需要的时间随问题规模呈指数增长，因此诞生了各种启发式算法来退而求其次寻找次优解，是一种近似算法(Approximate algorithms)。以最小化函数值目标为例，局部搜素算法的基本思想是在当前点的邻域内搜索，找到邻域内的最优解。将当前函数值与邻域内最优解的函数值作比较，如果当前函数值更小，则停止搜索，输出当前解作为局部最优解；否则，将该取到邻域内最优解设为当前点解，重复进行邻域搜索。标记已经解得的局部最优解或求解过程，并在进一步的迭代中避开这些局部最优解或求解过程。本文将使用基于禁忌规则的局部搜索算法解决的机组排班问题，具体思路如下：

为了顺利应用 LS 的思想来求解机场排班问题，本文提出两个新的概念：“机组人员组编”和“机组人员解编”，其具体概念如下所示。

机组人员组编过程：

步骤 1：在初始解航班环之中随机选择两个航班环 A、B；

步骤 2：若 A、B 有共同机场，则选择两个航班环中到达时间最早的公共机场 T，在 T 机场进行两个航班环机组人员的组编结合，组编结合的规则如下图 3.4 所示：选择航班总数少的航班环 A，将 A 航班环中的多余机组人员组编到 B 航班环；若 A 航班环中的机组人员为最低配置，则直接将 A 公共机场之后的航段取消，让 A 航班环的机组人员从公共机场开始即和 B 航班环进行组编；若 A 航班环中有多于的机组人员，则不取消 A 航班环后

续航段，如下图 3.6 所示，将 A 航班环公共机场前的航段复制一份作为新航班环 C，航班环 C 为 A 航班环多余机组人员的航班环，其中在 A、B 公共机场前的航段均为“乘机”航段，之后为飞行服务航段；

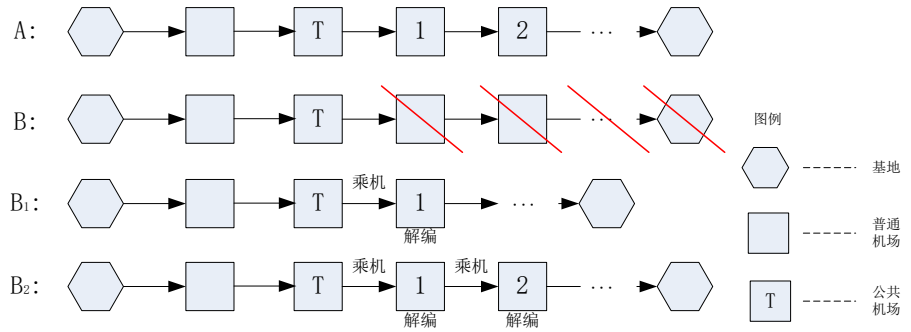


图 3.4 有公共机场 T

步骤 3：若 A、B 没有共同机场，则判断 A、B 是否具有相同的基地，若有，则选择基地为公共机场；否则，重新选择用于机组人员混编的航班环，如下图 3.5 所示。组编之后需要计算减少的效益 Bt_1（减少的航班数和其他成本）。

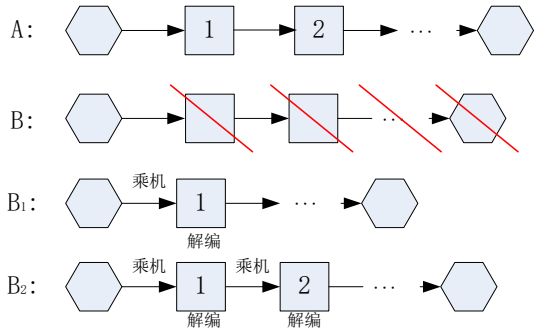


图 3.5 无公共机场，但有共同基地

机组人员解编过程：

对于进行机组人员组编的航班环 A 或者新建的航班环 C，从 A、B 组编的位置开始（基地\公共机场）向后遍历 B 航班环中的每一个机场，计算在 B 航班环之后的每一个机场将新组编的机组人员进行解编独立服务所能增加的效益 Bt_2（增加的航班数\增加的乘机次数\增加的使用替补资格的次数上述三者均要考虑）。

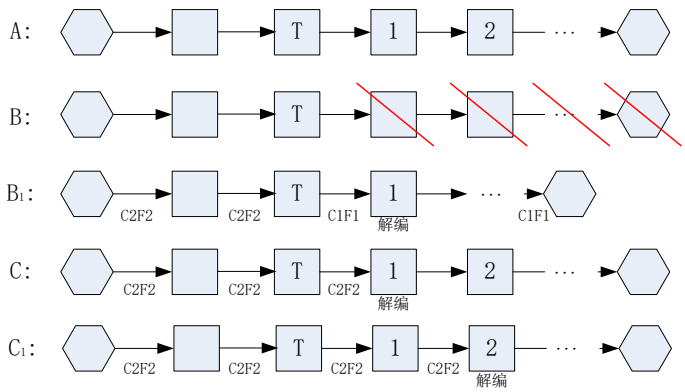


图 3.6 有公共机场 T，航班环 B 有多余机组人员

在减少的效益 Bt_1 确定的情况下，增加的效益 Bt_2 的值越大，则此次优化的效果越明显，所以根据完全贪婪的思想，选择 Bt_2 最大的解编机场位置将航班环 B 新增的机组人员进行解编。解编之后，对于航班环 A/C 中的机组人员，从组编机场开始到当前解编机场的航段都为“乘机”航段，从解编位置之后的航段为服务航段。为了避免陷入局部最优解，依概率 50%选择效益 Bt_2 最大的解编位置进行解编，依概率 50%选择效益 Bt_2 次大的解编位置进行解编。

本文将上述一次解编和一次组编过程称为一次 LS 优化过程。基于 LS 的优化算法如下表 3.2 所示。

表 3.2 基于 LS 的优化算法

1：输入初始解 $\boldsymbol{S_0}$
2：初始化外部循环迭代次数 *outNum*，外循环计数变量 *i*，内循环迭代次数 *inNum*，内循环计数变量 *j*，全局最优解 *bestSol*=$\boldsymbol{S_0}$，局部最优解 *locSol*=$\boldsymbol{S_0}$，次优解 *wosSol*=$\boldsymbol{S_0}$
3：**while** i++ < *outNum*, **do**
4：　　j = 0;
5：　　**while** j++ < *inNum*, **do**
6：　　随机选择 *locSol* 中的任意两个航班环 A、B
7：　　**if** A、B 有公共机场 T, **then**
8：　　以 T 为机组人员重组点对 A、B 进行一次组编和解编优化操作
9：　　**else if** A、B 有公共基地 D, **then**
10：　　以 D 为机组人员重组点对 A、B 进行一次组编和解编优化操作
11：　　**else**
12：　　重新选择用于优化的航班环 A、B
13：　　**continue**;
14：　　**end If**
15：　　**if** *f*(*locSol*)>*f*(*bestSol*), **then**
16：　　更新 *bestSol*=*locSol*
17：　　**end If**
18：　　**if** 达到模拟退火接受标准, **then**
19：　　*locSol*=*wosSol*;
20：　　**else**

21：　　*localSol=bestSol*;
22：　　**end If**
23：　**end while**
24：　更新模拟退火参数相关参数
25：**end while**
26：输出最优解的目标函数值 *f*(*bestSol*)

局部搜索的缺点在于，太过于对某一局部区域以及其邻域的搜索，导致一叶障目。为了找到全局最优解，禁忌搜索就是对于找到的一部分局部最优解，有意识地避开它，从而或得更多的搜索区域[3]。对于组编邻域操作算子，本文引入禁忌(TS)规则来防止重复搜索。每一个禁忌元素的设计内容如下：一个禁忌元素=<航班环 idx, 机场 idx, 汇入的机组人员数 num>。根据算例规模大小，本文将禁忌表的长度设置为[5, 10]之间的任意数字；同时本文采用先进先出(FIFO)的解禁方式来动态更新禁忌表中的禁忌元素。

同时本文采用模拟退火的方式来选择次优解进行后续的优化，借此来跳出局部最优解。为了增加寻优空间的广度和防止陷入局部最优解，本文同样使用模拟退火机制来接受退化的解方案[4]。

接受退化解的概率如下式（3.14）所示，

$$e^{-(f(S')-f(S_{inc}))/tempr} \tag{3.14}$$

其中 *tempr* 表示当前模拟退火的温度， $f(S')$表示当前解的目标函数值，$f(S_{inc})$表示当前内部临时最优解的目标函数值；

模拟退火的初始温度值设定如下式（3.15）所示，

$$tempr = -tp * f(S_0) / \ln(0.5) \tag{3.15}$$

其中，*tp* 表示温度初始化参数，本文取值为 0.005[5]， $f(S_0)$表示表示初始解的目前函数值。在每一次迭代结束之后，均运用模拟退火机制计算接受概率 *ARatio*，若 *ARatio* 大于随机概率 *RRatio*，则接受退化的解方案作为下一次优化的输入项，同时将此次优化的粒子的适应度减小为 0，提高粒子在下一轮优化之中的使用概率。如果连续 5000 次迭代没有对最优解进行更新，则采用下式（3.16）重新更新模拟退火的初始温度，

$$tempr = -tp * f(S_{inc}) / \ln(0.5) \tag{3.16}$$

其中，$f(S_{inc})$表示当前内部临时最优解的目标函数值，并且对粒子群序列的最优算子的位置进行重置。

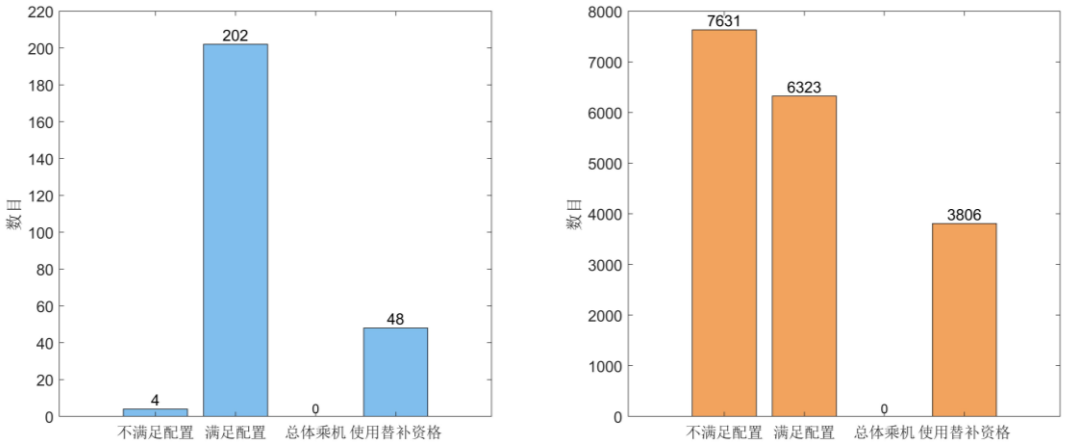
3.4 问题一模型的结果

根据上述设计的启发式算法编写程序，对问题一模型进行求解，分别运行 A、B 两套检测数据得到如下图 3.7 所示的结果。

A 数据结果见图 3.7(a)：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、替补资格使用次数分别为 4、202、0 和 48。从结果可以看出，本套数据的仅有 4 个航班不满足机组人员配置，98%的航班能够满足最低的机组人员配置；以及 A 数据的程序运行时间较短，仅为 0.01 分钟。

B 数据结果见图 3.7(b)：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、替补资格使用次数分别为 7631、6323、0 和 3806。该套数据中不能满足机

组人员配置的航班数目较多，超过 50%；由于 B 套数据的量较大，程序运行共花费 0.5 分钟。



(a) A 数据 **(b) B 数据**

图 3.7 模型一结果

4. 问题二的模型与求解

4.1 问题二的描述与分析

问题二是在问题一的基础上引进执勤的概念，执勤由一连串航段（飞行或乘机）和间隔连接时间组成。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。在问题一的约束条件下，初步的确定出航班的排班计划，该问将在前一问的基础上将排班计划中加入执勤的目标与约束，实现执勤任务的划分，如下图 4.1 所示。同时，在初始解构建过程中需要额外增加对于执勤相关约束的考量；在 LS 优化的过程中，除了也需要考量执勤的相关因素之外，对每次局部搜索的成本还需要考量新增加的执勤成本。

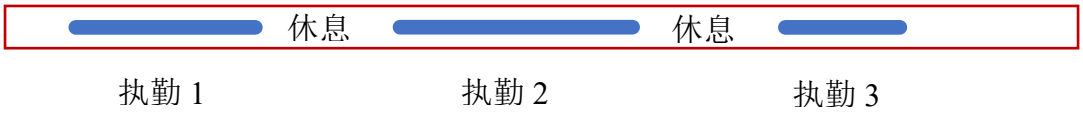


图 4.1 排班计划引入执勤概念，划分执勤任务

4.2 问题二的模型的建立

（1）确立目标函数：该问当中引进执勤的概念，在保证问题一模型的优化目标的同时需要使得机组人员的总体执勤成本最低以及机组人员之间的执勤时长尽可能平衡。

目标函数 1：最小化机组人员总体执勤成本，

$$\min \sum_{i \in E} DT_i \cdot DC_i \tag{4.1}$$

目标函数 2：机组人员的执勤时间尽可能平衡。为了将“时间尽可能平衡”这一优化指标进行量化处理，本文引入方差的概念，使得机组人员间的执勤时间的波动幅度最小，即转换成满足机组人员执勤时间的方差最小这一目标，

$$\min \frac{\sum_{i \in E} (DT_i - \overline{DT})}{M} \tag{4.2}$$

（2）确定模型的约束条件，在保证问题一约束的条件下，需要加入如下执勤相关的约束规则：

约束 1：每个机组人员每天至多只能执行一个执勤，

$$\lceil T_{i,j+1} \rceil - \lfloor A_{ij} \rfloor \geq 0, i \in E, j \in G \tag{4.3}$$

该约束涉及到在一天的时间内，不能同时出现第 j 次执勤的结束和第 j+1 次执勤的开始。为满足该约束条件，本题用“⌊⌋”和“⌈⌉”符号分别表示对日期时间进行向下和向上取整运算。例如，某一执勤的开始时间为“8 月 20 日 7：30”，向上取整后得到 20，向下取整后得到 21。对日期时间经过此类运算，可以保证得到整数的值（本题数据不涉及跨月份的航班，可以确保上述运算规则的正确性）。

约束 2：每次执勤的飞行时间最多不超过 MaxBlk 分钟，

$$Ft_{ij} \leq MaxBlk, i \in E, j \in G \tag{4.4}$$

约束 3：每次执勤的时长最多不超过 MaxDP 分钟，

$$A_{ij} - T_{ij} \leq MaxDP, i \in E, j \in G \tag{4.5}$$

约束 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，此约束在模型一中已经表示，见(3.12)。

约束 5：每个机组人员的相邻两个执勤之间的休息时间不小于 MinRest 分钟，

$$T_{i,j+1} - A_{ij}, i \in E, j \in G \tag{4.6}$$

4.3 问题二模型的求解

模型二是在模型一的基础之上添加了执勤相关的考量，在上述模型的算法框架之中，只需要在贪婪构造和 LS 优化时同样对执勤相关的约束进行考量即可完成模型二模型算法的构建工作。具体操作如下图 4.2 所示。

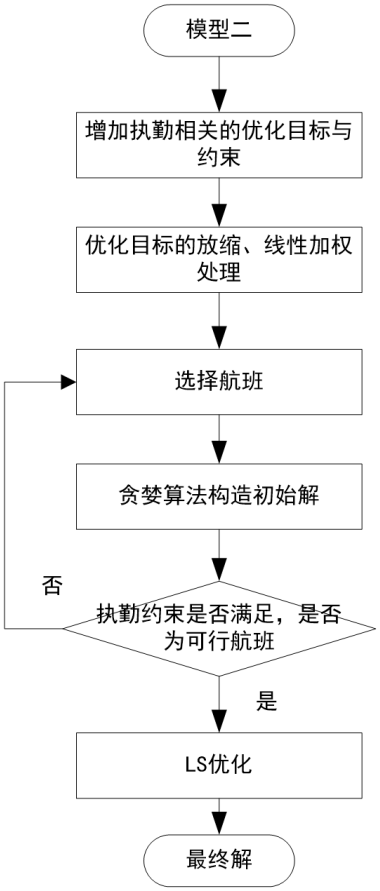


图 4.2 模型二求解思路

在贪婪构造初始解时，在判断某个航班的是否为可行航班时，除了需要对上述模型一中的连接时间约束进行判断，还需要考察每次执勤的时长不超过 MaxDP 分钟，每次执勤的飞行时长不超过 MaxBlk 分钟和相邻两个执勤之间的休息时间不小于 MinRest 分钟。若上述执勤相关的约束不满足，则说明当前选定的航班不满足要求，需要重新选择其他的航班继续构造航班环。

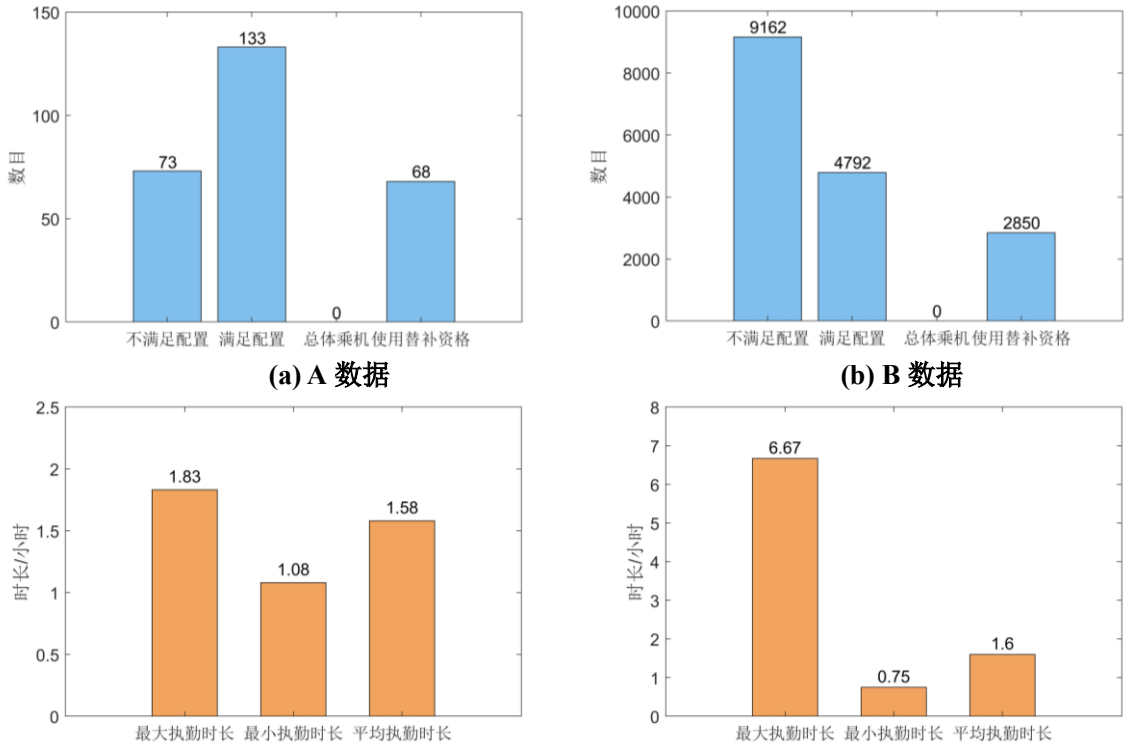
在使用 LS 进行优化时，除了需要考虑模型一种的目标①④⑦，还需要添加考虑机组人员的总体执勤成本②和机组人员之间的执勤时长平衡度（用所有执勤是时长的方差来表示）⑤。即在每一次组编机组人员和解编机组人员操作时，当对组编和解编过程的成本进行衡量时，需要综合考虑上述 5 个目标，采用加权的方式将上述 5 个目标融合为一个目标，权重分别采用 4、2、2、1、1，每一次操作都选择使得综合效益最大的操作作为最终操作来进行优化。

4.4 问题二模型的结果

对问题二模型进行求解，分别运行 A、B 两套检测数据得到如下图 4.2 所示的结果。

A 数据结果分析：图 4.2(a)表明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、替补资格使用次数分别为 73、133、0 和 68。从该结果可以看出，不满足机组人员配置的航班较多，超过 50%；机组利用率为 96.57%；从图 4.2(c)中看出机组人员的最大、平均和最小执勤天数分别为 1.83、1.08 和 1.58；并计算得到总体执勤成本为 28.51 万元；以及 A 数据的程序运行时间较短，仅为 0.01 分钟。

B 数据结果分析：图 4.2(b)表明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、替补资格使用次数分别为 9162、4792、0 和 2850；机组利用率达到 97.81%；从图 4.2(d)中看出机组人员的最大、平均和最小执勤天数分别为 6.67、0.75 和 1.60；并计算得到总体执勤成本为 1011.23 万元；该问题下程序运行共花费 0.3 分钟。



(a) A 数据 **(b) B 数据**

(c) A 数据 **(d) B 数据**

图 4.3 模型二结果

5. 问题三的模型与求解

5.1 问题三的描述与分析

问题三是在前两问的基础上引进任务环的概念，编制排班计划。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。在模型构建的过程中需要加入任务环的优化目标与相关约束；在初始解构建过程中需要额外增加对于任务环相关约束的考量；在 LS 优化的过程中，除了也需要考量任务环的相关因素之外，对每次局部搜索的成本还需要考量新增加的任务环成本

5.2 问题三的模型的建立

（1）确立目标函数：该问当中引进任务环的概念，再保证前两问模型的优化目标的同时需要使得机组人员的总体任务环成本最低以及机组人员之间的任务环时长尽可能平衡。

目标函数 1：最小化机组人员总体任务环成本，

$$\min \sum_{i \in E} PT_i \cdot PC_i \tag{5.1}$$

目标函数 2：机组人员的任务环时长尽可能平衡。该优化目标与第二个问中的优化目标 2 类似，这里同样使用机组人员的任务环时长的方差进行量化处理，

$$\min \frac{\sum_{i \in E} (PT_i - \overline{PT})}{M} \tag{5.2}$$

（2）确定模型的约束条件，在保证问题一约束的条件下，需要加入如下执勤相关的约束规则：

约束 1：每个机组人员每个排班周期的任务环总时长不超过 MaxTAFB 分钟，

$$\sum_{i \in E} \sum_{j \in H} Pt_{ij} \le MaxTAFB \tag{5.3}$$

约束 2：每个机组人员相邻两个任务环之间至少有 MinVacDay 天休息，

$$\left\lceil Tp_{i,j+1} \right\rceil - \left\lfloor Ap_{ij} \right\rfloor \ge MinVacDay, i \in E, j \in H \tag{5.4}$$

同问题二中的约束 1，这里同样使用“⌊⌋”和“⌈⌉”符号分别表示对日期时间进行向下和向上取整运算。例如，某一任务环的开始时间为“8 月 20 日 7：30”，向上取整后得到 20，向下取整后得到 21。对日期时间经过此类运算，可以保证得到整数的值，进而与 MinVacDay 进行比较。

约束 3：每个机组人员连续执勤天数不超过 MaxSuccOn 天。

$$\sum_{i \in E} \sum_{j \in H} Pt_{ij} \le MaxSuccOn * 24 * 60 \tag{5.5}$$

5.3 问题三模型的求解

模型三是在模型一和模型二的基础之上进一步添加了任务环相关的考量，在上述模型二的算法框架之中，只需要在贪婪构造和 LS 优化时同样对任务环相关的约束进行考量即可完成模型三模型算法的构建工作。具体操作如下图 5.1 所示。

在贪婪构造初始解时，在判断某个航班的是否为可行航班时，除了需要对上述模型一和模型二中的连接时间和执勤时间约束进行判断，还需要考察每个排班周期的任务环总时长不超过 MaxTAFB 分钟，相邻两个任务环之间至少有 MinVacDay 天休息和连续执勤天数不超过 MaxSuccOn 天，本文将连续执勤天数进行简单处理，直接设定每个任务环的时间不超过 MaxSuccOn 天。若上述任务环相关的约束不满足，则说明当前选定的航班不满足要求，需要重新选择其他的航班继续构造航班环。

在使用 LS 进行优化时，除了需要考虑模型一和模型二中的目标①④⑦②⑤，还需要添加考虑机组人员的总体任务环成本③和机组人员之间的任务环时长平衡度(用所有执勤是时长的方差来表示) ⑥。即在每一次组编机组人员和解编机组人员操作时，当对组编和解编过程的成本进行衡量时，需要综合考虑上述 7 个目标，采用加权的方式将上述 7 个目标融合为一个目标，权重分别采用 3、2、1.5、1.5、1、0.5、0.5，每一次操作都选择使得综合效益最大的操作作为最终操作来进行优化。

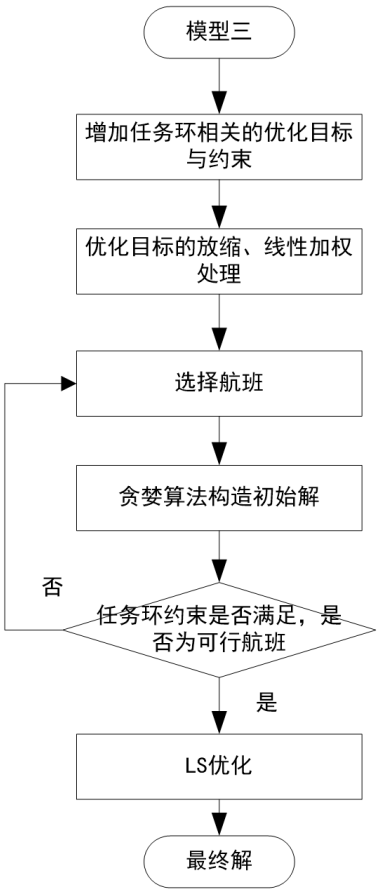


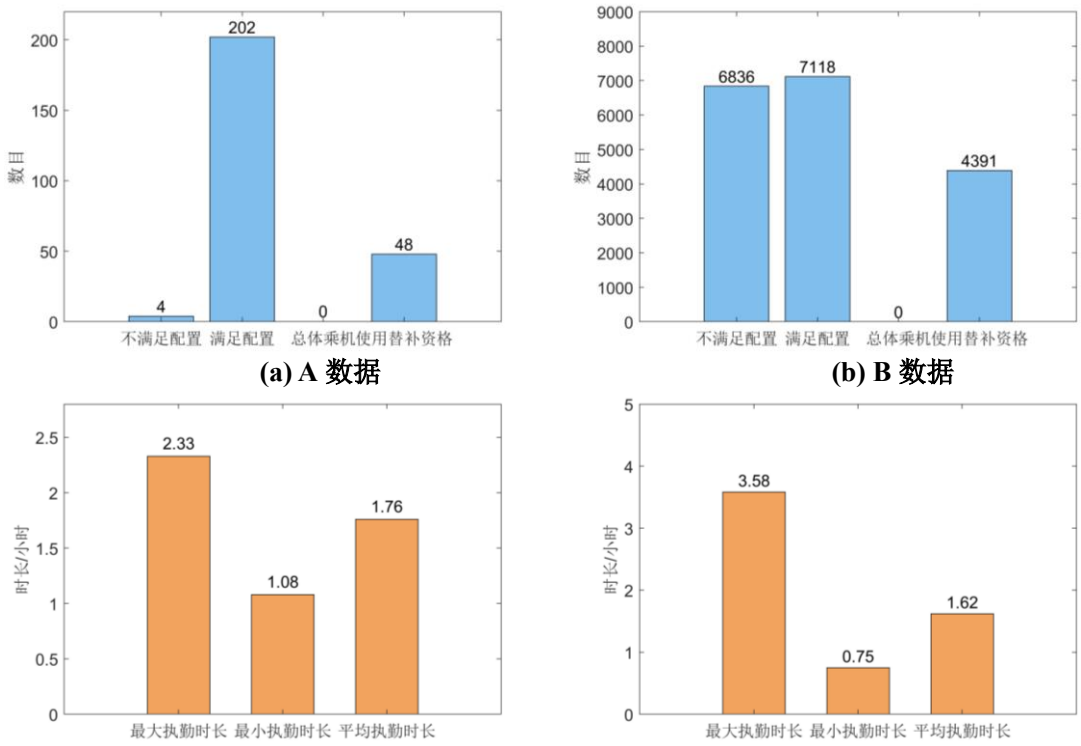
图 5.1 模型三求解思路

5.4 问题三模型的结果

对问题三模型进行求解，分别运行 A、B 两套检测数据得到如下图 5.2 所示的结果。

A 数据结果分析：图 5.2(a)表明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、替补资格使用次数分别为 4、202、0 和 48。从图 5.2(c)可以看出，仅有 4 个航班不满足机组的人员配置；机组利用率为 100%；从图中看出机组人员的最大、平均和最小执勤天数分别为 2.33、1.08 和 1.76；并计算得到总体执勤成本为 43.39 万元；总体任务环成本为 0.08 万元；以及 A 数据的程序运行时间为 0.01 分钟。

B 数据结果分析：图 5.2(b)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、替补资格使用次数分别为 6836、7118、0 和 4391；机组利用率达到 99.47%；从图 5.2(d)中看出机组人员的最大、平均和最小执勤天数分别为 3.58、0.75 和 1.62；并计算得到总体执勤成本为 1448.05 万元；总体任务环成本为 5.12 万元；该问题下程序运行共花费 0.33 分钟。



(a) A 数据　(b) B 数据

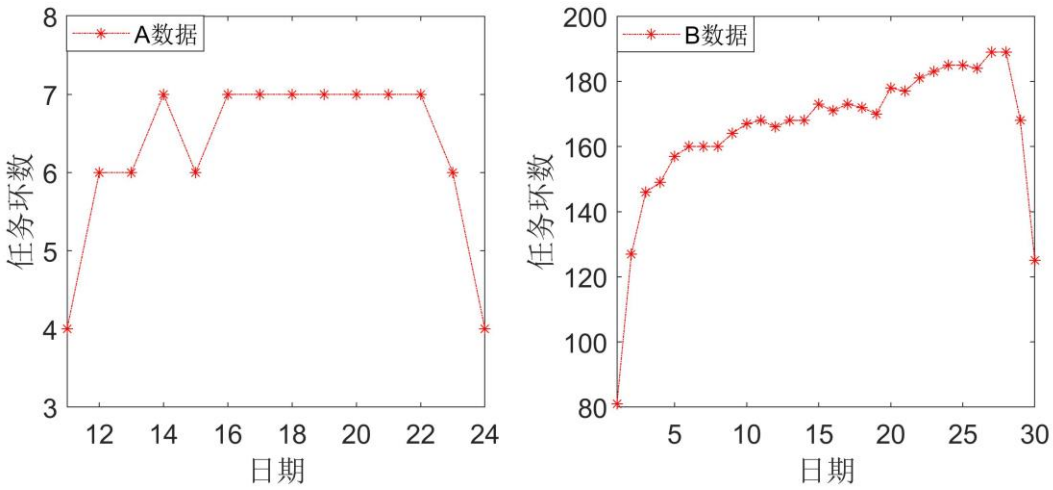
(c) A 数据　(d) B 数据

图 5.2 模型三结果

表 5.1 A、B 数据周期内的任务环数目

日期	A 任务环数	B 任务环数	日期	A 任务环数	B 任务环数
1	—	81	16	7	171
2	—	127	17	7	173
3	—	146	18	7	172
4	—	149	19	7	170
5	—	157	20	7	178
6	—	160	21	7	177
7	—	160	22	7	181
8	—	160	23	6	183
9	—	164	24	4	185
10	—	167	25	—	185
11	4	168	26	—	184
12	6	166	27	—	189
13	6	168	28	—	189
14	7	168	29	—	168

15	6	173	30	—	125



(a) (b)

图 5.3 任务环分布图

A 套数据和 B 套数据的任务环数量见表 5.1，分布如图 5.3 所示，两套数据的任务环数量分布较为相似，在各自计算周期的起点、终点位置数量分布较少，中间阶段的数量分布较为均衡。

6. 结论

高效合理的机组排班计划将极大提高航空公司运行效率，节省运营成本。本题将航空公司实际的排班过程进行抽象，使得问题之间环环相扣。

问题一，本文首先采用线性加权的方法对多目标的目标函数进行处理，之后构建了混合整数规划模型来精确描述问题。针对模型提出贪婪初始解构造和 LS 优化思路，在 LS 优化设计的过程中针对该问题创新性的提出“解编”和“组编”的启发式优化思想，使解的效果更加理想；

问题二，加入新的优化目标和约束条件，在初始解和 LS 优化过程中需要将执勤的相关影响因素纳入考量，局部搜索成本加入执勤的成本；

问题三，在前两问的基础上再次增加优化目标与约束，在初始解和 LS 优化过程中需要将任务环的相关影响因素纳入考量，局部搜索成本加入任务环的成本。

用设计的算法对三个问题进行求解，表 6.1 和表 6.2 分别列出了 A 套数据和 B 套数据的检测指标结果，图 6.1 给出了检测两套数据的程序运行时间。A 套数据量较小，均在 0.01 分钟内完成；B 套数据量较大，程序运行时间在 0.3-0.5 分钟之间。综合计算结果来看，本文构建的模型，启发式的优化算法在解决机组排班问题上表现较好，并且具有较强的实用性。

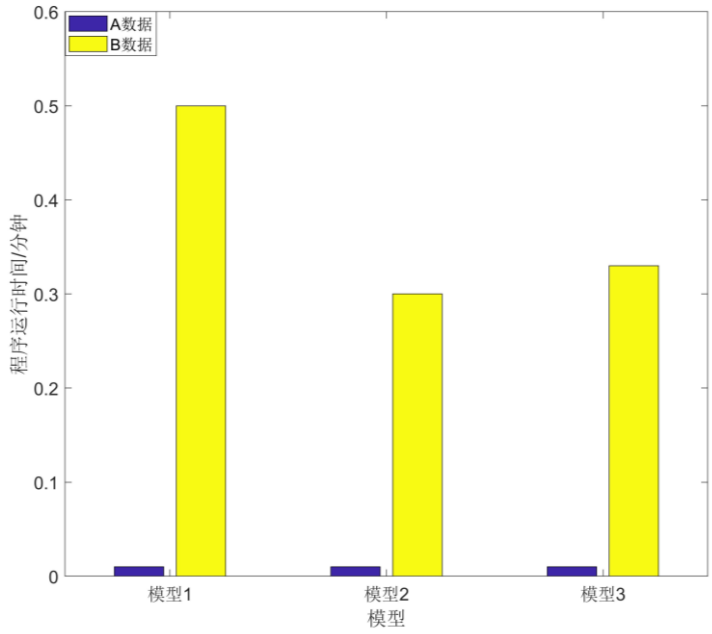


图 6.1 程序运行时间对比图

表 6.1 A 套数据指标结果

编号	结果指标	问题一	问题二	问题三
1	不满足机组配置航班数	4	73	4
2	满足机组配置航班数	202	133	202
3	机组人员总体乘机次数	0	0	0

4	替补资格使用次数	48	68	48
5	机组总体利用率		96.57%	100%
6	最小/平均/最大 一次执勤飞行时长		1.83	2.33
7	最小/平均/最大 一次执勤时长		1.08	1.08
8	最小/平均/最大 机组人员执勤天数		1.58	1.76
9	一/二/三/四天 任务环数量分布			表 5.1、图 5.3(a)
10	总体执勤成本（万元）		28.508	43.39
11	总体任务环成本（万元）			0.08
12	程序运行分钟数	0.01	0.01	0.01

表 6.2 B 套数据指标结果

编号	结果指标	问题一	问题二	问题三
1	不满足机组配置航班数	7631	9162	6836
2	满足机组配置航班数	6323	4792	7118
3	机组人员总体乘机次数	0	0	0
4	替补资格使用次数	3806	2850	4391
5	机组总体利用率		97.81%	99.47%
6	最小/平均/最大 一次执勤飞行时长		6.67	3.58
7	最小/平均/最大 一次执勤时长		0.75	0.75
8	最小/平均/最大 机组人员执勤天数		1.6	1.62
9	一/二/三/四天 任务环数量分布			表 5.1、图 5.3(b)
10	总体执勤成本（万元）		1011.23	1448.05
11	总体任务环成本（万元）			5.12
12	程序运行分钟数	0.5	0.3	0.33

7. 模型优缺点与展望

7.1 模型优势

本文所构架的启发式算法框架思路简洁，具有较强的适应性和推广性；

本文构建的算法能在较短时间内取得较为理想的解方案，实用性较强；

本文对于机场排班提出了创新性的“解编和组编”的优化思路，可以为之后的机场排班算法提供借鉴意义。

7.2 模型劣势

本文将机组人员连续执勤天数直接看做是一次任务环的天数，这样计算相对于真实的连续执勤天数可能会超出 MaxSuccOn 天一些时间；

本文使用线性加权方法处理多目标函数，权重的选取主观因素较大，对于优化结果的影响较大；

本文算法精确性和求解质量仍有较大的改进空间。

7.3 展望和完善

本模型可以进一步使用 CPLEX 进行求解，还可以基于列生成和 BCP 算法设计精确算法进行求解；

对于启发式算法方面，模型的优化算法还可以设计更加复杂的求解算子，如 2-opt 算子来进行优化，能够达到更优的求解效果；

还可以对启发式算法添加自适应的机制，动态调整不同算子的选择使用的概率，或者动态调整禁忌搜索禁忌表的长度，使用历史信息更优效率地进行寻优。

8. 参考文献

[1] 向杜兵. 航空机组排班计划一体化优化研究[D].北京交通大学,2020.

[2] 肖晓伟, 肖迪, 林锦国,等. 多目标优化问题的研究概述[J]. 计算机应用研究, 2011, 28(003):805-808,827.

[3] 彭正超,胡晓兵,周韶武,殷鸣,李彦儒.基于改进粒子群-禁忌搜索算法的FMS布局优化[J].组合机床与自动化加工技术,2021(06):159-163.

[4] 刘忠慧,陈建宇,宋国杰,闵帆.基于模拟退火法的概念集构造算法[J].模式识别与人工智能,2021,34(08):723-732.

[5] Parragh S.N., Cordeau J.F. Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows[J]. Computers & Operations Research, 2017, 83:28-44.

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

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

[8] 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.

9. 附录

9.1 算法头文件【header.h】

```
#pragma once
#include<iostream>
#include<vector>
#include<map>
#include<unordered_map>
#include<algorithm>
#include<cmath>
#include<fstream>
#include<ctime>
#include <iomanip>


//航班结构体
class Flight
{
public:
    struct tm Dt;           //航班的出发时间
    char Ds[3];             //航班的出发机场
    struct tm At;           //航班的到达时间
    char As[3];             //航班的到达机场
    int fIdx;               //航班编号

    struct tm dutyBeg;      //当前航班所属的执勤的开始时间
    double dutyTime;        //当前航班所属的执勤的总时长
    double dutyFlyTime;     //当前航班所属的执勤的飞行时长
    double ringTime;        //当前航班所属的单个任务环时长
    double tolRingTime;     //当前航班所属的所有任务环时长
};

//机组人员结构体
class Crew
{
public:
    int Cp;             //员工是否可以担任正机长，0-1
    int Fo;             //员工是否可以担任副机长，0-1
```

```
36.     int Dh;             //员工是否允许乘机，0-1
37.     char Bs[3];         //员工的基地
38.     double Dc;          //员工的单位小时执勤成本
39.     double Pc;          //员工的单位小时任务环成本
40.     int cIdx;           //员工编号
41. };
42.
43. //执勤结构体
44. class Duty
45. {
46. public:
47.     std::vector<Flight> dutys;
48.
49. };
50.
51. //航班环结构体
52. class FltRing
53. {
54. public:
55.     std::vector<Flight> flts;                 //航班集合
56.     std::vector<std::vector<Crew>> crws;        //机组人员集合
57.     std::vector<bool> svRec;                  //记录机组人员在每个航班是否服务 1，还是乘机
    0
58.     int dutyNum;                                //记录总的执勤次数
59.     double tolDutyTime;                         //总执勤时长
60.     double tolFlyTime;                          //总飞行时长
61.     std::vector<double> dutyTimeRec;                  //记录所有次执勤的时长
62.     std::vector<double> flyTimeRec;                   //记录所有次执勤的飞行时长
63.     int ringNum;                                //记录总的任务环数量
64.
65.     double AccuRingTime;                        //记录所有任务环的时间
66.     std::vector<double> betweenRings;         //记录所有任务环之间的休假时间
67.     std::unordered_map<int, int> datRing;         //记录不同天的任务环数量
68.
69.
70.     int lastPortIdx = -1;                       //记录最后一个可以返回基地的航班在 flts 中
    的索引
71.     Flight lastFlight;                          //记录最后一个可以返回基地的航班
72. };
73.
74. //解结构体
75. class Solution
```

```
{
public:
    std::vector<FltRing> fRings;  //所有的航班环
    std::vector<Flight> unDone;   //所有未服务的航班集合
    std::vector<Flight> Done; //所有已经服务的航班集合

    Solution(std::vector<Flight>& SetF)
    {
        unDone = SetF;
    }
};

extern std::vector<Flight> SetF;  //所有航班的集合
extern std::vector<Crew> SetC;        //所有员工的集合


//读取数据
void InputData();

//贪婪构造初始解
void InitCrt(Solution &initSol);

//计算每个机场到其他机场之间的航班数量
void FillFltNum();

//输出结果
    void Output(Solution & Sol);
```

9.2 算法函数文件【func.cpp】

```
#define _CRT_SECURE_NO_WARNINGS
#include"header.h"

using namespace std;


vector<int> P;            //所有机场的位置集合
int N = 2;               //每个航班总共需要的机组人数
int Nc = 1;              //每个航班总共需要的正机长人数
int Nf = 1;              //每个航班总共共需要的副机长人数
int FltNum;              //全部的航班数
```

```
12. int crwNum;              //全部的机组人员数
13.
14. double MinCT = 40;       //航段之间最小的连接时间 40min
15. double MaxBlk = 600;     //一次执勤飞行时长最多不超过 600min
16. double MaxDP = 720;      //执勤时长最多不超过 720min
17. double MinRest = 660;    //相邻执勤之间的休息时间不少于 660min
18. int MaxDH = 5;           //每趟航班最多的乘机人数 5 人
19. double MaxTAFB = 14400; //排班周期单个机组人员的任务环总时长不超过 14400
20. int MaxSuccOn = 4;       //连续执勤天数不超过 4 天
21. int MinVacDay = 2;       //相邻两个任务环之间至少休息 2 天
22.
23.
24. vector<Flight> SetF;  //所有航班的集合
25. vector<Crew> SetC;        //所有员工的集合
26. unordered_map<string, int> g_FltNum;  //每个机场到其他机场之间的航班数量
27. clock_t sTime = clock();
28.
29. ofstream output("E:/桌面文件/研究生/华为数学建模竞赛/F/newOut/B-3-out.txt");
30.
31. //读取数据
32. void InputData()
33. {
34.     ifstream input1("E:/桌面文件/研究生/华为数学建模竞赛/F/B-Flight.txt");
35.     ifstream input2("E:/桌面文件/研究生/华为数学建模竞赛/F/B-Crew.txt");
36.     if (!input1 || !input2 || !output)
37.     {
38.         cout << "打开输入/输出文件失败！" << endl;
39.         exit(-1);
40.     }
41.
42.     input1 >> FltNum;
43.     input2 >> crwNum;
44.     SetF.resize(FltNum, Flight());
45.     SetC.resize(crwNum, Crew());
46.
47.     for (int i = 0; i < FltNum; ++i)
48.     {//读入机场数据
49.         tm tempT;
50.         input1 >> tempT.tm_mon >> tempT.tm_mday >> tempT.tm_year >> tempT.tm_hour >> te
    mpT.tm_min;
51.         SetF[i].Dt = tempT;
52.         SetF[i].Dt.tm_sec = 0;
```

```
53.         SetF[i].Dt.tm_year -= 1900;
54.         SetF[i].Dt.tm_mon -= 1;
55.         SetF[i].Dt.tm_isdst = -1;
56.         input1 >> SetF[i].Ds;
57.         input1 >> tempT.tm_mon >> tempT.tm_mday >> tempT.tm_year >> tempT.tm_hour >> te
    mpT.tm_min;
58.         SetF[i].At = tempT;
59.         SetF[i].At.tm_sec = 0;
60.         SetF[i].At.tm_year -= 1900;
61.         SetF[i].At.tm_mon -= 1;
62.         SetF[i].At.tm_isdst = -1;
63.         input1 >> SetF[i].As;
64.         SetF[i].fIdx = i;
65.     }
66.
67.     for (int i = 0; i < crwNum; ++i)
68.     {
69.         input2 >> SetC[i].Cp >> SetC[i].Fo >> SetC[i].Dh >> SetC[i].Bs >> SetC[i].Dc >>
     SetC[i].Pc;
70.         SetC[i].cIdx = i;
71.     }
72.
73.     input1.close();
74.     input2.close();
75. }
76.
77. //计算每个机场到其他机场之间的航班数量
78. void FillFltNum()
79. {
80.     for (int i = 0; i < FltNum; ++i)
81.     {
82.         g_FltNum[SetF[i].Ds] += 1;
83.         g_FltNum[SetF[i].As] += 1;
84.
85.     }
86. }
87.
88. //判断两个航班的离开时间是否在同一天
89. bool JgeSameDay(Flight &f1, Flight &f2)
90. {
91.     if (f1.At.tm_year != f2.At.tm_year)
92.         return false;
```

```
93.     else if (f1.At.tm_mon != f2.At.tm_mon)
94.         return false;
95.     else if (f1.At.tm_mday != f2.At.tm_mday)
96.         return false;
97.     else
98.         return true;//同一年，同一月，同一天
99. }
100.
101. //计算相邻两个航班直接的出发时间和离开时间之差,f1 为前一个航班，f2 为后一个航班
102. double CalDiffTime(Flight &f1, Flight &f2)
103. {
104.     //时间加减
105.     time_t t2 = mktime(&f2.Dt);
106.     time_t t1 = mktime(&f1.At);
107.
108.     double t = difftime(t2, t1);//double difftime( time_t timeEnd, time_t timeStart );
109.     return t/60;
110. }
111.
112. //计算某个航班的飞行时间
113. double CalFltTime(Flight &f)
114. {
115.     //时间加减
116.     time_t t2 = mktime(&f.At);
117.     time_t t1 = mktime(&f.Dt);
118.
119.     double t = difftime(t2, t1);//double difftime( time_t timeEnd, time_t timeStart );
120.     return t/60;
121. }
122.
123. //寻找最低资格配置机组人员组合
124. bool FindCrewPair(vector<Crew> &qulified, vector<Crew> &remainCrews)
125. {
126.     if (remainCrews.size() < N) return false;
127.
128.     bool CpFlag = false,    //标记是否有正机长
129.         FoFlag = false;     //标记是否有副机长
130.     for (int i = 0; i <(int)remainCrews.size(); ++i)
131.     {
132.         if (remainCrews[i].Cp && !remainCrews[i].Fo && !CpFlag)
```

```
133.         {//先找一个没有副机长替补资格的正机长
134.             CpFlag = true;
135.             qulified.push_back(remainCrews[i]);
136.             remainCrews.erase(remainCrews.begin() + i);
137.         }
138.         else if (!remainCrews[i].Cp && !FoFlag)
139.         {//找一个副机长，不使用替补资格
140.             FoFlag = true;
141.             qulified.push_back(remainCrews[i]);
142.             remainCrews.erase(remainCrews.begin() + i);
143.         }
144.         if (CpFlag && FoFlag)
145.             break;
146.     }
147.     if (!CpFlag)
148.     {//找有副机长替补资格的正机长
149.         for (int i = 0; i < (int)remainCrews.size(); ++i)
150.         {
151.             if (remainCrews[i].Cp && !CpFlag)
152.             {
153.                 CpFlag = true;
154.                 qulified.push_back(remainCrews[i]);
155.                 remainCrews.erase(remainCrews.begin() + i);
156.             }
157.         }
158.     }
159.     if (!CpFlag) return false;
160.
161.     //判断是否需要使用正机长的替补资格
162.     if (!FoFlag)
163.     {//需要使用正机长的替补资格
164.         for (int i = 0; i < (int)remainCrews.size(); ++i)
165.         {
166.             if (remainCrews[i].Cp && remainCrews[i].Fo)
167.             {
168.                 FoFlag = true;
169.                 qulified.push_back(remainCrews[i]);
170.                 remainCrews.erase(remainCrews.begin() + i);
171.             }
172.             if (FoFlag)
173.                 break;
174.         }
```

```
175.     }
176.     if (!FoFlag) return false;
177.
178.     return true;
179. }
180.
181. //判断某个航班相对于上个航班是不是可行航班，满足执勤时间要求，满足任务环时间要求
182. bool JgeFlt(Flight &lastFlt, Flight &preFlt, bool base, bool newDuty, bool newRing)
183. {
184.     if (base) return true;
185.
186.     //判断 连接时间 是否>=MinCT
187.     double diffTime1 = INT_MAX,      //判断连接时间
188.         diffTime2 = INT_MAX,         //判断执勤时间
189.         diffTime3 = INT_MAX;         //判断任务环时间
190.
191.     if (!base)//若不是从基地出发的第一个航班，则有前一个航班 laseFlt，不是新的执勤
192.         diffTime1 = CalDiffTime(lastFlt, preFlt);
193.     else if (!base && newDuty)
194.         diffTime2 = CalDiffTime(lastFlt, preFlt);
195.     else if(!base && newRing)
196.         diffTime3 = CalDiffTime(lastFlt, preFlt);
197.
198.     if (diffTime1 < 0 || diffTime2 < 0 || diffTime3 < 0)
199.         return false;
200.
201.     if (!newDuty && !newRing && diffTime1 < MinCT)           //连接时间满足要求
202.         return false;
203.     if (newDuty && diffTime2 < MinRest)                      //执勤时间满足要求
204.         return false;
205.     if (newRing && diffTime3 < MinVacDay * 24 * 60)          //任务环时间满足要求
206.         return false;
207.
208.     return true;
209. }
210.
211. //计算某个机场的航班数量
212. int CalFltNum(char* port, vector<Flight> &remainFlight)
213. {
214.     //int res = 0;
215.     //for (int i = 0; i < (int)remainFlight.size(); ++i)
216.     //{
```

```
217.     //  if (!strcmp(port, remainFlight[i].Ds))
218.     //      ++res;
219.     //}
220.     //return res;
221.     return g_FltNum[port];
222. }
223.
224. //寻找飞行时间最长的航班            航班数量最多的机场
225. int FindFlight(char* prePort, vector<Flight> &remainFlight, Flight & lastFlt, bool bas
    e, bool newDuty, bool newRing)//最后 3 个参数表示是否从基地出发,是否新的执勤，是否新的任务
    环
226. {
227.     //double minTime = 0;        //最大航行时间
228.     double maxFNum = 0; //最大航班数量
229.     int resFlt = -1;
230.     for (int i = 0; i < (int)remainFlight.size(); ++i)
231.     {
232.         if (!strcmp(prePort, remainFlight[i].Ds))
233.         {
234.             bool fltFlag = JgeFlt(lastFlt, remainFlight[i], base, newDuty, newRing);//
    判断相邻两个航班之间的连接时间是否>=MinCT
235.
236.             if (fltFlag)
237.             {//计算航班数量
238.                 //double fltTime = CalFltTime(remainFlight[i]);
239.                 int fltNum = CalFltNum(remainFlight[i].Ds, remainFlight);
240.                 //if (fltTime > minTime)
241.
242.                 //添加随机因素
243.                 int randR = rand() % 101;
244.                 if (fltNum > maxFNum )
245.                 {
246.                     /*if (base)
247.                     {
248.                         resFlt = i;
249.                         maxFNum = fltNum;
250.                     }*/
251.                     //else if(strcmp(remainFlight[i].As, lastFlt.Ds) != 0)
252.                     //{
253.                     if (resFlt == -1)
254.                     {
255.                         resFlt = i;
```

```
256.                         maxFNum = fltNum;
257.                     }
258.                     else
259.                     {
260.                         if (randR > 50)
261.                         {
262.                             resFlt = i;
263.                             maxFNum = fltNum;
264.                         }
265.                     }
266.                     //}
267.                 }
268.             }
269.
270.         }
271.     }
272.     return resFlt;
273. }
274.
275. //判断某个机场是否有可行航班可以使得当前航班环返回基地
276. bool JgeToBase(char *prePort, char *base, Flight &lastFlt, vector<Flight> &remainFligh
    t, Flight &resFlt, bool newDuty, bool newRing)
277. {
278.     for (int i = 0; i < (int)remainFlight.size(); ++i)
279.     {
280.         if (!strcmp(prePort, remainFlight[i].Ds) && !strcmp(base, remainFlight[i].As)

281.             && JgeFlt(lastFlt,remainFlight[i],0, newDuty, newRing))
282.         {//若有，则选择第一个可以返回基地的航班
283.             resFlt = remainFlight[i];
284.             return true;
285.         }
286.     }
287.     return false;
288. }
289.
290. //在已经构建的航班环中选择一个可以在某机场接纳机组人员的航班环，假设乘机也需要满足最小连接时间
    约束
291. int BackFltRing(char *prePort, char *preBase, Flight &preFlt, FltRing &preRing, bool n
    ewDuty, bool newRing)
292. {
293.     for (int i = 0; i < (int)preRing.flts.size(); ++i)
```

```
294.     {
295.         if (!strcmp(preRing.flts[i].Ds, prePort) && JgeFlt(preFlt, preRing.flts[i], 0, newDuty, newRing)
296.             && !strcmp(preBase, preRing.flts[0].Ds))//基地相同
297.             return i;
298.     }
299.     return -1;
300. }
301.
302. //判断从前一个航班当当前选定的航班是不是新的执勤
303. bool JgeNewDuty(bool base, Flight &lastFlt, Flight &preFlt)
304. {
305.     if (base) return true;
306.
307.     int newDuty = false;
308.
309.     if (lastFlt.dutyBeg.tm_mday != preFlt.dutyBeg.tm_mday)//若不是同一天出发的，则是newDuty
310.         newDuty = true;
311.     else if (lastFlt.dutyTime + CalDiffTime(lastFlt, preFlt) + CalFltTime(preFlt) > MaxDP)//执勤时间判断
312.         newDuty = true;
313.     else if (lastFlt.dutyFlyTime + CalFltTime(preFlt) > MaxBlk)//执勤飞行时间判断
314.         newDuty = true;
315.     else
316.         newDuty = false;
317.     return newDuty;
318. }
319.
320. //判断从前一个航班当当前选定的航班是不是新的任务环
321. bool JgeNewRing(bool base, Flight &lastFlt, Flight &preFlt)
322. {
323.     if (base) return true;
324.
325.     int newRing = false;
326.
327.     if (lastFlt.ringTime + CalDiffTime(lastFlt, preFlt) + CalFltTime(preFlt) > MaxSuccOn*24*60)//连续执勤天数判断
328.         newRing = true;
329.
330.
331.     return newRing;
```

```
}


//贪婪构造初始解
void InitCrt(Solution &initSol)
{
    //遍历所有机组人员，划分不同基地的机组人员
    vector<vector<Crew>> partCrews(1);

    char *preBs = SetC[0].Bs;
    partCrews.back().push_back(SetC[0]);
    for (int i = 1; i < crwNum; ++i)
    {
        if (!strcmp(SetC[i].Bs, preBs))
            partCrews.back().push_back(SetC[i]);
        else        //假设机组人员的基地顺序排列
        {
            partCrews.push_back(vector<Crew>());
            preBs = SetC[i].Bs;
            partCrews.back().push_back(SetC[i]);
        }
    }
    if (partCrews.size() > 1)
    {
        vector<Crew> t = partCrews[0];
        partCrews[0] = partCrews[1];
        partCrews[1] = t;
    }

    //对于每一个基地的机组人员新建航班环
    for (int i = 0; i < (int)partCrews.size(); ++i)
    {
        //判断当前剩余机组人员是否能够达到最低资格配置；
        //若能，则找到用于新建航班环的机组人员；若不能将剩余机组人员加到上一个航班环中
        vector<Crew> preCrews,
            notUsedCrws;                //记录最终没有用上的机组人员
        bool flag = FindCrewPair(preCrews, partCrews[i]);
        char *base = preCrews[0].Bs;
        while (flag)
        {//若找到合适配置的机组人员，新开辟一个航班环
            initSol.fRings.push_back(FltRing());
            //贪心往每一个航班环后插入航班，直到不能插入为止
```

```
374.             Flight tf;
375.             int preFlt = FindFlight(preCrews[0].Bs, initSol.unDone, tf, 1,1,1); //寻找
    到达机场航班数最大的航班
376.
377.             while (preFlt != -1)
378.             {
379.                 Flight tFlt = initSol.unDone[preFlt], lastFlt;
380.                 //判断从前一个航班到当前选定的航班是不是新的任务环
381.                 bool newRing = false, newDuty = false;
382.                 if (initSol.fRings.back().flts.empty())
383.                 {
384.                     Flight tf;
385.                     newRing = JgeNewRing(1, tf, tFlt);
386.                 }
387.                 else
388.                     newRing = JgeNewRing(0, initSol.fRings.back().flts.back(), tFlt);

389.                 if (!newRing)
390.                 {
391.                     //判断从前一个航班到当前选定的航班是不是新的执勤
392.                     if (initSol.fRings.back().flts.empty())
393.                     {
394.                         Flight tf;
395.                         newDuty = JgeNewDuty(1, tf, tFlt);
396.                     }
397.                     else
398.                         newDuty = JgeNewDuty(0, initSol.fRings.back().flts.back(), tFl
    t);
399.                 }
400.                 else
401.                     newDuty = true;//新任务环肯定对应新的执勤
402.
403.                 tFlt.dutyBeg = newDuty ? tFlt.Dt : lastFlt.Dt;
404.                 double flyTime = CalFltTime(tFlt);
405.                 tFlt.dutyFlyTime += newDuty ? flyTime : initSol.fRings.back().flts.bac
    k().dutyFlyTime + flyTime;
406.                 tFlt.dutyTime += newDuty ? flyTime : initSol.fRings.back().flts.back()
    .dutyTime + flyTime + CalDiffTime(lastFlt, tFlt);
407.                 tFlt.ringTime += newRing ? flyTime : initSol.fRings.back().flts.back()
    .dutyTime + flyTime + CalDiffTime(lastFlt, tFlt);
408.                 tFlt.tolRingTime += initSol.fRings.back().flts.empty() ? flyTime : las
    tFlt.dutyTime + flyTime + CalDiffTime(lastFlt, tFlt);
```

```
409.                 if (tFlt.ringTime > MaxTAFB || tFlt.dutyFlyTime > MaxBlk || tFlt.dutyT
    ime > MaxDP)
410.                 {//判断最大航班环的时间
411.                     preFlt = -1;
412.                     initSol.fRings.back().AccuRingTime += initSol.fRings.back().flts.b
    ack().ringTime;
413.                     goto LABEL;
414.                 }
415.                 else
416.                 {
417.                     if (!initSol.fRings.back().flts.empty())
418.                     {
419.                         initSol.fRings.back().dutyNum += newDuty ? 1 : 0;
420.                         initSol.fRings.back().tolDutyTime += newDuty ? initSol.fRings.
    back().flts.back().dutyTime : 0;
421.                         initSol.fRings.back().tolFlyTime += newDuty ? initSol.fRings.b
    ack().flts.back().dutyFlyTime : 0;
422.                         if (newDuty)
423.                         {
424.                             initSol.fRings.back().dutyTimeRec.push_back(initSol.fRings
    .back().flts.back().dutyTime);
425.                             initSol.fRings.back().flyTimeRec.push_back(initSol.fRings.
    back().flts.back().dutyFlyTime);
426.                         }
427.                         initSol.fRings.back().ringNum += newRing ? 1 : 0;
428.                         if(newRing)
429.                             initSol.fRings.back().betweenRings.push_back(CalDiffTime(i
    nitSol.fRings.back().flts.back(),tFlt));
430.                         initSol.fRings.back().AccuRingTime += newRing ? initSol.fRings
    .back().flts.back().ringTime : 0;
431.
432.                         if (tFlt.Dt.tm_mday != initSol.fRings.back().flts.back().Dt.tm
    _mday)
433.                             initSol.fRings.back().datRing[initSol.fRings.back().flts.b
    ack().Dt.tm_mday] += 1;
434.                     }
435.                     initSol.fRings.back().flts.push_back(tFlt);
436.                     initSol.fRings.back().svRec.push_back(1);
437.                     //initSol.Done.push_back(tFlt);
438.                     initSol.unDone.erase(initSol.unDone.begin() + preFlt);
439.                     --g_FltNum[tFlt.Ds];
```

```
440.                     initSol.fRings.back().crws.push_back(preCrews);             //当前
    航班的机组人员配置
441.
442.                     lastFlt = initSol.fRings.back().flts.back();
443.
444.                     if (JgeToBase(tFlt.As, base, lastFlt, initSol.unDone, initSol.fRin
    gs.back().lastFlight, newDuty, newRing))
445.                     {//更新最后一个可以返回基地的航班 lastPort, lastFlight
446.                         initSol.fRings.back().lastPortIdx = (int)initSol.fRings.back()
    .flts.size() - 1;
447.                         initSol.fRings.back().lastFlight.dutyFlyTime += initSol.fRings
    .back().flts.back().dutyFlyTime + flyTime;
448.                         initSol.fRings.back().lastFlight.dutyTime += initSol.fRings.ba
    ck().flts.back().dutyTime + flyTime + CalDiffTime(initSol.fRings.back().flts.back(), tF
    lt);
449.                         initSol.fRings.back().lastFlight.ringTime +=  initSol.fRings.b
    ack().flts.back().dutyTime + flyTime + CalDiffTime(initSol.fRings.back().flts.back(), t
    Flt);
450.                         initSol.fRings.back().lastFlight.tolRingTime += initSol.fRings
    .back().flts.back().dutyTime + flyTime + CalDiffTime(initSol.fRings.back().flts.back(),
     tFlt);
451.
452.                     }
453.
454.                     preFlt = FindFlight(lastFlt.As, initSol.unDone, lastFlt, 0, newDut
    y, newRing);//寻找航行时间最大的航班
455.                 }
456.
457.                 LABEL:
458.                 if (preFlt == -1)
459.                 {//航班环构造结束，判断机组人员是否在基地
460.                     int lastPos = initSol.fRings.back().lastPortIdx;
461.                     if (strcmp(initSol.fRings.back().flts.back().As, base) != 0 && las
    tPos != -1)
462.                     {//若不在，则选择最后一个可以返回基地的航班让机组人员返回基地，更新
    initSol.unDone 和 initSol.Done
463.                         Flight lastFlt = initSol.fRings.back().lastFlight;
464.                         if (lastPos == initSol.fRings.back().flts.size() - 1)
465.                         {//直接回基地就行
466.                             initSol.fRings.back().flts.push_back(lastFlt);
467.                             initSol.fRings.back().crws.push_back(initSol.fRings.back()
    .crws.back());
```

```
468.                                 initSol.fRings.back().svRec.push_back(1);
469.
470.                             }
471.                             else
472.                             {
473.                                 vector<Flight> delFlts, recFlts;
474.                                 vector<vector<Crew>> recCrws;
475.                                 vector<bool> recSv;
476.                                 delFlts.assign(initSol.fRings.back().flts.begin() + lastPo
    s + 1, initSol.fRings.back().flts.end());
477.                                 initSol.unDone.insert(initSol.unDone.end(), delFlts.begin(
    ), delFlts.end());
478.                                 //更新航班环、员工环和服务环
479.                                 recFlts.assign(initSol.fRings.back().flts.begin(), initSol
    .fRings.back().flts.begin() + lastPos + 1);
480.                                 initSol.fRings.back().flts = recFlts;
481.                                 recCrws.assign(initSol.fRings.back().crws.begin(), initSol
    .fRings.back().crws.begin() + lastPos + 2);
482.                                 initSol.fRings.back().crws = recCrws;
483.                                 recSv.assign(initSol.fRings.back().svRec.begin(), initSol.
    fRings.back().svRec.begin() + lastPos + 2);
484.                                 initSol.fRings.back().svRec = recSv;
485.
486.                                 initSol.fRings.back().flts.push_back(lastFlt);
487.
488.                             }
489.                         }
490.                         else if (lastPos == -1)
491.                         {//若没有航班可以让当前机组人员做任务回基地，则选择其他航班乘机回基地
492.                             int j = 0;
493.                             for (; j < initSol.fRings.size() - 1; ++j)
494.                             {
495.                                 char *preBase = initSol.fRings.back().flts[0].Ds;
496.                                 int pos = BackFltRing(initSol.fRings.back().flts.back().As
    , preBase, initSol.fRings.back().flts.back(), initSol.fRings[j], newDuty, newRing);
497.                                 if (pos != -1)
498.                                 {//让当前机组人员在 pos 位置之后乘坐 initSol.fRings[j]的飞机返
    回基地
499.                                     vector<Flight> recFlts;
500.                                     vector<vector<Crew>> recCrws;
501.                                     vector<bool> recSv;
```

```
502.                                 recFlts.assign(initSol.fRings[j].flts.begin()+pos, ini
    tSol.fRings[j].flts.end());
503.
504.                                 initSol.fRings.back().flts.insert(initSol.fRings.back(
    ).flts.end(), recFlts.begin(), recFlts.end());
505.                                 int tNum = (int)initSol.fRings[j].flts.size() - pos;
506.                                 initSol.fRings.back().crws.insert(initSol.fRings.back(
    ).crws.end(), tNum, initSol.fRings.back().crws[0]);
507.                                 initSol.fRings.back().svRec.insert(initSol.fRings.back
    ().svRec.end(), tNum, 0);
508.
509.
510.                                 break;
511.                             }
512.                         }
513.                         if (j == initSol.fRings.size() - 1)
514.                         {//当前航班环直接判定无效
515.                             initSol.unDone.insert(initSol.unDone.end(), initSol.fRings
    .back().flts.begin(), initSol.fRings.back().flts.end());
516.                             notUsedCrws.insert(notUsedCrws.end(), initSol.fRings.back(
    ).crws[0].begin(), initSol.fRings.back().crws[0].end());
517.                             initSol.fRings.pop_back();
518.                         }
519.
520.                     }
521.                     initSol.fRings.back().AccuRingTime += initSol.fRings.back().flts.b
    ack().ringTime;
522.
523.                 }
524.             }
525.             if (initSol.fRings.back().flts.empty())
526.             {
527.                 initSol.fRings.pop_back();
528.                 break;
529.             }
530.
531.             preCrews.clear();
532.             flag = FindCrewPair(preCrews, partCrews[i]);
533.         }
534.
535.         //若剩余机组人员不能满足最低配置，将剩余机组人员全部加入到最后一个航班环中
536.         if (!partCrews[i].empty())
```

```
537.         {
538.             for (int j = 0; j < (int)initSol.fRings.back().crws.size(); ++j)
539.                 initSol.fRings.back().crws[j].insert(initSol.fRings.back().crws[j].end
    (), partCrews[i].begin(), partCrews[i].end());
540.         }
541.         //若有 notUsedCrews,则将他们分组分别随着开始航班环依次乘机
542.         if (!notUsedCrws.empty())
543.         {
544.             //分组
545.             vector<vector<Crew>> grp(1);
546.             for (int j = 0; j < notUsedCrws.size(); ++j)
547.             {
548.                 if (grp.back().size() < MaxDH)
549.                     grp.back().push_back(notUsedCrws[j]);
550.
551.                 else
552.                 {
553.                     grp.push_back(vector<Crew>());
554.                     grp.back().push_back(notUsedCrws[j]);
555.                 }
556.             }
557.             //乘机
558.             for (int j = 0; j < (int)grp.size(); ++j)
559.             {
560.                 for (int k = 0; k < (int)initSol.fRings[j].crws.size(); ++k)
561.                     initSol.fRings[j].crws[k].insert(initSol.fRings[j].crws[k].end(),
    grp[j].begin(), grp[j].end());
562.             }
563.         }
564.
565.     }
566.
567.
568. }
569.
570. //计算某个航班环的航班段数量
571. int CalFltNum(vector<bool> &svRec)
572. {
573.     int rec = 0;
574.     for (const auto & c : svRec)
575.     {
576.         if (c == 1)
```

```
577.             ++rec;
578.     }
579.     return rec;
580. }
581.
582. //一次解编和组编优化
583. void OneOpt(Solution & Sol)
584. {
585.     //随机选择两个航班环
586.     vector<int> tSet;
587.     for (int i = 0; i < Sol.fRings.size(); ++i)
588.         tSet.push_back(i);
589.     random_shuffle(tSet.begin(), tSet.end());
590.     int ring1 = tSet[0], ring2 = tSet[1];
591.
592.     int fNum1 = CalFltNum(Sol.fRings[ring1].svRec),
593.         fNum2 = CalFltNum(Sol.fRings[ring2].svRec);
594.     //判断两个
595.
596.
597. }
598.
599. //输出结果
600. void Output(Solution & Sol)
601. {
602.     cout << "不满足机组配置的航班数为：" << (int)Sol.unDone.size() << endl;
603.     cout << "满足机组配置的航班数为：" << FltNum - (int)Sol.unDone.size() << endl;
604.     output << "不满足机组配置的航班数为：" << (int)Sol.unDone.size() << endl;
605.     output << "满足机组配置的航班数为：" << FltNum - (int)Sol.unDone.size() << endl;
606.     int followNum = 0;
607.     for (int i = 0; i < (int)Sol.fRings.size(); ++i)
608.     {
609.         for (int j = 0; j < (int)Sol.fRings[i].svRec.size(); ++j)
610.         {
611.             if (Sol.fRings[i].svRec[j] == 0)
612.                 ++followNum;
613.         }
614.     }
615.     cout << "整体乘机次数为：" << followNum << endl;
616.     output << "整体乘机次数为：" << followNum << endl;
617.
618.     int rep = 0;
```

```
619.     for (int i = 0; i < (int)Sol.fRings.size(); ++i)
620.     {
621.         for (int j = 0; j < (int)Sol.fRings[i].crws.size(); ++j)
622.         {
623.             if (Sol.fRings[i].crws[j][1].Cp == 1)
624.                 ++rep;
625.         }
626.     }
627.     cout << "整体替补次数为: " << rep << endl;
628.     output << "整体替补次数为: " << rep << endl;
629.
630.     double totalDutyCost = 0;
631.     for (int i = 0; i < Sol.fRings.size(); ++i)
632.     {
633.         totalDutyCost += Sol.fRings[i].tolDutyTime/60 *Sol.fRings[i].crws[0][0].Dc;
634.         totalDutyCost += Sol.fRings[i].tolDutyTime / 60 *Sol.fRings[i].crws[0][1].Dc;
635.
636.     }
637.     cout << "总体执勤成本为: " << totalDutyCost/10000 << endl;
638.     output << "总体执勤成本为: " << totalDutyCost/10000 << endl;
639.
640.     double maxDutyTime = 0, minDutyTime = INT_MAX, tolDutyTime = 0, tolFlyTime = 0, to
    lNum = 0, aveDutyTime = 0;
641.     for (int i = 0; i < Sol.fRings.size(); ++i)
642.     {
643.         double preMax = *max_element(Sol.fRings[i].dutyTimeRec.begin(), Sol.fRings[i].
    dutyTimeRec.end());
644.         if (preMax > maxDutyTime)
645.             maxDutyTime = preMax;
646.
647.         double preMin = *min_element(Sol.fRings[i].flyTimeRec.begin(), Sol.fRings[i].f
    lyTimeRec.end());
648.         if (preMin < minDutyTime)
649.             minDutyTime = preMin;
650.
651.         for (int j = 0; j < Sol.fRings[i].dutyTimeRec.size(); ++j)
652.             tolDutyTime += Sol.fRings[i].dutyTimeRec[j];
653.
654.         tolNum += Sol.fRings[i].dutyNum;
655.     }
656.     for (int i = 0; i < Sol.fRings.size(); ++i)
```

```
657.     {
658.         for (int j = 0; j < Sol.fRings[i].flts.size()-1; ++j)
659.             tolFlyTime += Sol.fRings[i].svRec[j] * CalFltTime(Sol.fRings[i].flts[j]);

660.     }
661.
662.     double vacTime = 0;
663.     for (int i = 0; i < Sol.fRings.size(); ++i)
664.     {
665.         for (int j = 0; j < Sol.fRings[i].betweenRings.size(); ++j)
666.             vacTime += Sol.fRings[i].betweenRings[j];
667.     }
668.
669.     //计算所有任务环的时间
670.     double tolRingTime = 0;
671.     for (int i = 0; i < Sol.fRings.size(); ++i)
672.         tolRingTime += Sol.fRings[i].AccuRingTime;
673.
674.
675.     cout << fixed << setprecision(2) << "最大一次执勤飞行时长为:
    " << maxDutyTime/60 << endl;
676.     cout << fixed << setprecision(2) << "最小一次执勤飞行时长为:
    " << minDutyTime/60 << endl;
677.     cout << fixed << setprecision(2) << "平均一次执勤飞行时长为:
    " << tolDutyTime/ tolNum /60 << endl;
678.     cout << fixed << setprecision(2) << "机组总体利用率为:
    " << (tolFlyTime) * 100 / (tolDutyTime) << "%" << endl;
679.     cout << fixed << setprecision(2) << "总体任务环成本为:
    " << tolRingTime * 20 / 60 / 10000 << endl;
680.
681.     output << fixed << setprecision(2) << "最大一次执勤飞行时长为:
    " << maxDutyTime/60 << endl;
682.     output << fixed << setprecision(2) << "最小一次执勤飞行时长为:
    " << minDutyTime/60 << endl;
683.     output << fixed << setprecision(2) << "平均一次执勤飞行时长为:
    " << tolDutyTime/ tolNum /60 << endl;
684.     output << fixed << setprecision(2) << "机组总体利用率为:
    " << (tolFlyTime) * 100 / (tolDutyTime) << "%" << endl;
685.     output << fixed << setprecision(2) << "总体任务环成本为:
    " << tolRingTime * 20 / 60 / 10000 << endl;
686.
687.     clock_t eTime = clock();
```

```
688.     cout << "程序运行时间为:
    " << fixed << setprecision(2) << double((eTime - sTime) / CLOCKS_PER_SEC)/60.0 << endl;

689.     output << "程序运行时间为:
    " << fixed << setprecision(2) << double((eTime - sTime) / CLOCKS_PER_SEC)/60.0 << endl;

690.
691.
692.     //处理任务环分布
693.     unordered_map<int, int> rec;
694.     for (int i = 0; i < Sol.fRings.size(); ++i)
695.     {
696.         unordered_map<int, int>::iterator ite = Sol.fRings[i].datRing.begin();
697.         for (int j = 0; j < Sol.fRings[i].datRing.size(); ++j)
698.         {
699.             rec[ite->first] += 1;
700.             ++ite;
701.         }
702.     }
703.     cout << "不同天数的任务环的数量分布:" << endl;
704.     output << "不同天数的任务环的数量分布:" << endl;
705.     unordered_map<int, int>::iterator ite = rec.begin();
706.     while(ite != rec.end())
707.     {
708.         cout << "第 " << ite->first << " 天的任务环数量为: " << ite->second << endl;
709.         output << "第 " << ite->first << " 天的任务环数量为:
    " << ite->second << endl;
710.         ++ite;
711.     }
712.
713.     //输出没有完成的航班
714.     output << "\n\n 时间的格式为月-日-年-时-分" << endl;
715.     for (int i = 0; i < Sol.unDone.size(); ++i)
716.     {
717.         output << "出发时间:
    "<< Sol.unDone[i].Dt.tm_mon+1 << "\t" << Sol.unDone[i].Dt.tm_mday << "\t" << Sol.unDone
    [i].Dt.tm_year+1900 << "\t" << Sol.unDone[i].Dt.tm_hour << "\t" << Sol.unDone[i].Dt.tm_
    min << endl;
718.         output << "到达时间:
    "<< Sol.unDone[i].At.tm_mon+1 << "\t" << Sol.unDone[i].At.tm_mday << "\t" << Sol.unDone
    [i].At.tm_year+1900 << "\t" << Sol.unDone[i].At.tm_hour << "\t" << Sol.unDone[i].At.tm_
    min << endl;
```

```
719.         output << "出发机场：" << Sol.unDone[i].Ds << endl;
720.         output << "到达机场：" << Sol.unDone[i].As << endl;
721.     }
722.     //输出所有航班信息
723.     output << "\n\n 所有已经完成的航班信息" << endl;
724.     output << "机组人员信息包括：机组人员数量、是否机长，是否副机长，是否替补" << endl;
725.     for (int i = 0; i < Sol.fRings.size(); ++i)
726.     {
727.         output << Sol.fRings[i].crws[0].size() << endl;
728.         for (int j = 0; j < Sol.fRings[i].crws[0].size(); ++j)
729.         {
730.             output << Sol.fRings[i].crws[0][j].Cp << "\t" << Sol.fRings[i].crws[0][j].
    Fo << "\t" << (j == Sol.fRings[i].crws[0].size()-
    1 ? Sol.fRings[i].crws[0][j].Cp:0) << endl;
731.         }
732.     }
733.     output << "\n\n 航班信息包括：航班起飞时间，起飞机场、到达时间、到达机场" << endl;
734.     for (int i = 0; i < Sol.fRings.size(); ++i)
735.     {
736.         for (int j = 0; j < Sol.fRings[i].flts.size(); ++j)
737.         {
738.             output << Sol.fRings[i].flts[j].Dt.tm_mon+1 << "\t" << Sol.fRings[i].flts[
    j].Dt.tm_mday
739.                 << "\t" << Sol.fRings[i].flts[j].Dt.tm_year+1900 << "\t" << Sol.fRings
    [i].flts[j].Dt.tm_hour << "\t"
740.                 << Sol.fRings[i].flts[j].Dt.tm_min << "\t" << Sol.fRings[i].flts[j].Ds
     << "\t" <<
741.
742.                 Sol.fRings[i].flts[j].At.tm_mon+1 << "\t" << Sol.fRings[i].flts[j].At.
    tm_mday
743.                 << "\t" << Sol.fRings[i].flts[j].At.tm_year+1900 << "\t" << Sol.fRings
    [i].flts[j].At.tm_hour << "\t"
744.                 << Sol.fRings[i].flts[j].At.tm_min << "\t" << Sol.fRings[i].flts[j].As
     << endl;
745.         }
746.         output << endl << endl;
747.
748.     }
    749. }
```

9.3 算法主函数文件【main.cpp】

```cpp
#include"header.h"

using namespace std;


int main()
{
    //读取数据
    InputData();
    FillFltNum();

    Solution initSol(SetF); //初始解

    //贪婪加随机构造解
    InitCrt(initSol);

    //输出结果
    Output(initSol);

    return 0;
}
```