“华为杯”第十五届中国研究生

数学建模竞赛

题　目　　中转航班调度：从 MILP 模型到启发式算法

摘　要

近年来航空业和旅游业蓬勃发展，越来越多的人选择乘坐飞机出行。2017 年，民航行业完成旅客运输量 55156 万人次，比上年增长 13.0%[1]。面对日益增长的乘客人数，航空公司通过对航站楼进行扩建并增设卫星厅以缓解原有航站楼登机口不足的压力。但是，在中转航班调度问题中，学术界和工业界对同时优化登机口的分配和最小化旅客行走时间的研究有限[2-5]。本文提出了线性规划模型和启发式算法，针对一个具体的航站楼扩增案例进行评估，并给出在不同的优化目标下，航班-登机口分配方案。

对于问题一，我们针对飞机和登机口可能的组合设计了 01 规划模型，最大化成功安排的航班数目，减少临时登机口的使用，并且在此基础上最小化使用的登机口的数目。我们使用 Python+CPLEX 优化器求解该模型，得到最多可以成功安排 512 次航班，最少使用 65 个登机口。我们基于经典的区间调度算法，设计线性时间复杂度的贪婪算法，能够成功使用 66 个登机口安排 510 次航班。

对于问题二，我们拓展问题一的模型并设计了混合整数线性规划（MILP）模型，以最小化旅客的流程时间。由于旅客的流程时间取决于诸多因素，例如航站楼、登机口属性等等，模型的变量和约束的数目较大，我们压缩了变量使用的状态以减少变量的个数，加速模型的求解。我们保持成功安排航班的数目为 512 次，得到最小的旅客流程时间的总和为 55490 分钟，共需要 66 个登机口。由于求解 MILP 模型对求解时间和计算资源的要求较高，我们基于模拟退火算法设计启发式算法快速求解问题二，保持成功安排航班的数目为 510 次，得到旅客流程时间的总和为 62075 分钟，共需要 66 个登机口。

对于问题三，我们沿用问题二的 MILP 模型，在不扩大模型的规模的同时，修改约束以考虑旅客行走时间和捷运时间的影响，修改目标函数来最小化旅客的换乘紧张度。模型三保持成功安排航班的数目为 512 次，得到的最小换乘紧张度为 531.38，需要 67 个登机口。类似的，我们拓展模拟退火启发式算法快速求解问题三，保持成功安排航班的数目为 510 次，得到换乘紧张度为 607.35，共需要 66 个登机口。

目　录

1 问题描述

由于旅行业的快速发展，某航空公司在某机场的现有航站楼 T 的旅客流量已达饱和状态，为了应对未来的发展，现正增设卫星厅 S。但引入卫星厅后，虽然可以缓解原有航站楼登机口不足的压力，对中转旅客的航班衔接显然具有一定的负面影响。本题希望参赛选手建立数学模型，优化分配登机口，分析中转旅客的换乘紧张程度，为航空公司航班规划的调整提供参考依据。

飞机在机场廊桥（登机口）的一次停靠通常由一对航班（到达航班和出发航班，也叫“转场”）来标识。所谓的中转旅客就是从到达航班换乘到由同一架或不同架飞机执行的出发航班的旅客。

航站楼 T 和卫星厅 S 的布局设计如下图所示。T 具有完整的国际机场航站功能，包括出发、到达、出入境和候机。卫星厅 S 是航站楼 T 的延伸，可以候机，没有出入境功能。为叙述方便起见，我们统称航站楼 T 和卫星厅 S 为终端厅。航站楼 T 有 28 个登机口，卫星厅 S 有 41 个登机口。

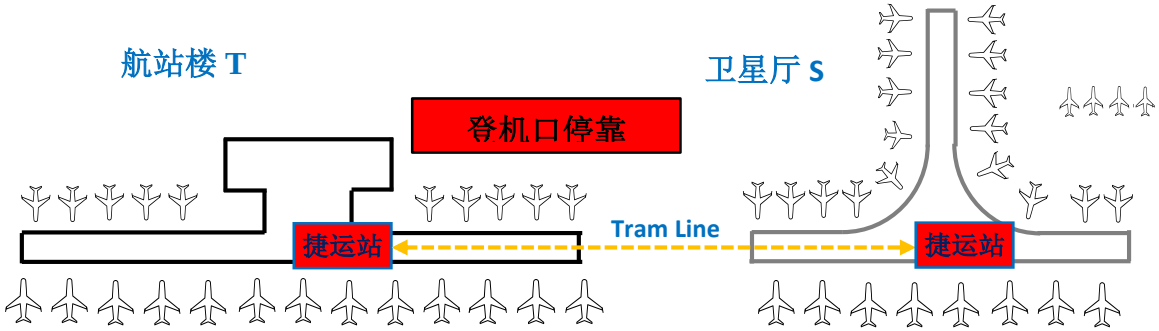

图 1-1 卫星厅 S 相对于航站楼 T 示意图

登机口属于固定机位，配置有相应的设备，方便飞机停靠时的各种技术操作。航班-登机口的分配需要考虑如下规则：

a) T 和 S 的所有登机口统筹规划分配；
b) 每个登机口的国内/国际、到达/出发、宽体机/ 窄体机等功能属性事先给定，不能改变，具体配置详见数据部分。飞机转场计划（见数据部分）里的航班只能分配到与之属性相吻合的登机口；
c) 每架飞机转场的到达和出发两个航班必须分配在同一登机口进行，其间不能挪移别处；
d) 分配在同一登机口的两飞机之间的空挡间隔时间必须大于等于 45 分钟；
e) 机场另有简易临时机位，供分配不到固定登机口的飞机停靠。假定临时机位数量无限制。

旅客流程可以按始发旅客、终到旅客和中转旅客分类规范。本文只考虑中转旅客，中转旅客从前一航班的到达至后一航班的出发之间的流程，按国内（D）和国际（I）、航站楼（T）和卫星厅（S）组合成 16 种不同的场景。

本文需要根据上述规则求解一下三个问题：

问题一：本题只考虑航班-登机口分配。作为分析新建卫星厅对航班影响问题的第一步，首先要建立数学优化模型，尽可能多地分配航班到合适的登机口，并

且在此基础上最小化被使用登机口的数量。本问题不需要考虑中转旅客的换乘，但要求把建立的数学模型进行编程，求最优解。

问题二：考虑中转旅客最短流程时间。本问题是在问题一的基础上加入旅客换乘因素，要求最小化中转旅客的总体最短流程时间，并且在此基础上最小化被使用登机口的数量。本题不考虑旅客乘坐捷运和步行时间，但也要求编程并求最优解。

问题三：考虑中转旅客的换乘时间。如前所述，新建卫星厅对航班的最大影响是中转旅客换乘时间的可能延长。因此，数学模型最终需要考虑换乘旅客总体紧张度的最小化，并且在此基础上最小化被使用登机口的数量。本问题可以在问题二的基础上细化，引入旅客换乘连接变量，并把中转旅客的换乘紧张度作为目标函数的首要因素。和前面两个问题一样，本问题也要求把建立的数学模型进行编程，并求最优解。换乘紧张度定义：

$$\text{换乘紧张度} = \frac{\textit{旅客换乘时间}}{\textit{航班连接时间}}$$

$$\text{顾客换乘时间} = \text{最短流程时间} + \text{捷运时间} + \text{行走时间}$$

$$\text{航班连接时间} = \text{后一航班出发时间} - \text{前一航班到达时间}$$

2 模型假设

根据题目描述，提出如下假设：

1. 假设所有航班都按照飞机转场记录按时起飞和降落，不存在飞机延误。
2. 本模型只考虑 20 日到达或 20 日出发的航班和旅客，超过此范围的不在模型及算法的考虑范围之内，也不在评价指标内考虑。
3. 旅客流程可以按始发旅客、终到旅客和中转旅客分类，本模型只考虑中转旅客。
4. 假设航站楼 T 和卫星厅 S 的登机口全都统筹规划分配。
5. 假设每个登机口的国内/国际、到达/出发、宽体机/窄体机等功能属性事先给定，不能再次改变，且飞机转场计划里的航班只能分配到与之属性相吻合的登机口。此外，窄体机不能停靠在宽体机的机位。
6. 假设每架飞机转场的到达和出发两个航班必须分配在同一登机口进行，其间不能再挪移别处。
7. 假设分配在同一登机口的两飞机之间的空挡间隔时间必须大于等于 45 分钟。
8. 假设机场设有临时机位，供分配不到固定登机口的飞机停靠，且临时机位数量无限制。如果有飞机选择临时机位进行停靠，该飞机上的旅客在后续的处理中被忽略。
9. 假设每个中转旅客的最短流程时间都根据国内（D）和国际（I）、航站楼（T）和卫星厅（S）组合成的 16 种不同的场景所决定，其中，换乘失败是指没有足够的换乘时间，在处理这些旅客时，假设其换乘时间是 6 小时。

3 符号定义及数据预处理

3.1 符号定义

表 3-1 模型使用的符号及其含义

符号	含义
常量	
A	飞机集合
a	某一飞机，$a \in A$
$Size_a$	a飞机的机型种类（N/W）
S	终端厅 S 的登机口集合
S_{North}	处于终端厅 S 北边的登机口集合，$S_{North} \subseteq S$
S_{South}	处于终端厅 S 南边的登机口集合，$S_{South} \subseteq S$
S_{Center}	处于终端厅 S 中间的登机口集合，$S_{Center} \subseteq S$
S_{East}	处于终端厅 S 东边的登机口集合，$S_{East} \subseteq S$
s	终端厅 S 的某一个登机口，$s \in S$
T	终端厅 T 的登机口集合
T_{North}	处于终端厅 T 北边的登机口集合，$T_{North} \subseteq T$
T_{South}	处于终端厅 T 南边的登机口集合，$T_{South} \subseteq T$
T_{Center}	处于终端厅 T 中间的登机口集合 $T_{Center} \subseteq T$
t	终端厅 T 的某一个登机口，$t \in T$
P	临时停靠机位集合
j	某一登机口（包含临时停靠机位），$j \in S \cup T \cup P$
$Area_j$	j 登机口所处的区域(North/Center/South/East)
$ArriveType_j$	j 登机口允许到达类型集合($\{I\}/\{D\}/\{D,I\}$，I表示国际航班，D表示国内航班)
$LeaveType_j$	j 登机口允许出发类型集合($\{I\}/\{D\}/\{D,I\}$)
$PlaneSize_j$	j 登机口允许停靠的飞机类型($\{N\}/\{W\}/\{N,W\}$，N表示窄机型，W表示宽机型)
$Arrive_a$	a飞机降落的航班
$Leave_a$	a飞机起飞的航班

$Time_{Arrive_a}$	a飞机降落航班的时间
$ConnectTime_{Arrive_{a_1}Leave_{a_2}}$ $= Time_{Leave_{a_2}} - Time_{Arrive_{a1}}$	乘坐a_1飞机降落航班，转乘a_2飞机起飞航班的航班连接时间
$Type_{Arrive_a}$	a飞机降落航班的类型(I/D)
$Type_{Leave_a}$	a飞机起飞航班的类型(I/D)
$Num_{Arrive_{a_1}Leave_{a_2}}$	乘坐a_1飞机降落航班，转乘a_2飞机起飞航班的乘客数目
$CostTime_k$	表示选择k方案所需的最短流程时间，$k = 0,1,2 \dots 16$
$CostAllTime_k$	表示选择k方案所需的总转运时间，$k = 0,1,2 \dots 196$
0-1 变量	
x_{aj}	是否将a飞机安排至j登机口，1 表示是
z_j	是否使用过j登机口，1 表示是
$y_{Arrive_{a_1}Leave_{a_2}k}$	乘坐a_1飞机降落航班，转乘a_2飞机起飞航班是否对应k方案所需的最短流程时间，1 表示是
$CanTransfer_{Arrive_{a_1}Leave_{a_2}}$	乘坐a_1飞机降落航班，转乘a_2飞机起飞航班是否能够成功转乘，1 表示成功
整数变量	
$ProcessTime_{Arrive_{a_1}Leave_{a_2}}$	乘坐a_1飞机降落航班，转乘a_2飞机起飞航班所花费的最短流程时间
$TravelTime_{Arrive_{a_1}Leave_{a_2}}$	乘坐a_1飞机降落航班，转乘a_2飞机起飞航班所花费的旅客换乘时间
模型结果	
$MaxFlightNum$	最多可以安排在固定登机位的航班数量
$MinProcessTime$	最少的中转旅客的总体流程时间（分钟）
$MinTense$	最少的换乘旅客总体紧张度

3.2 数据预处理

首先根据题目要求，我们只考虑 20 日到达或 20 日出发的航班和旅客，经过筛选，总共需要对 303 架飞机，606 个航班进行航班-登机口方案的分配。此外，为了统一模型的表达，我们给临时停靠位也赋予了允许到达类型、允许出发类型和停靠的飞机类型的属性，由于临时停靠为对于允许到达类型，允许出发类型和停靠的飞机类型没有限制，我们设定：

$$\forall j \in P: ArriveType_j = \{D, I\}$$
$$\forall j \in P: LeaveType_j = \{D, I\}$$

$$\forall j \in P: PlaneSize_j = \{N, W\}$$

此外，我们对于流程时间的表格进行了方案的编码，统计出了$CostTime_k(k = 1,2,\dots 16)$的值 $CostTime = [0,15,20,35,40,20,15,40,35,35,40,20,30,40,45,30,20]$。当$k = 0$时意味着，到达和出发的航班中至少有一个航班被分配至临时停靠机位，此时$CostTime_0 = 0$，意味着不再计算这部分转乘旅客的流程时间。

表 3-2 $CostTime$编码规则

到达 \ 出发		国内出发（D）		国际出发（I）	
		航站楼 T	卫星厅 S	航站楼 T	卫星厅 S
国内到达（D）	航站楼 T	15/0 (k=1)	20/1 (k=2)	35/0 (k=3)	40/1 (k=4)
	卫星厅 S	20/1 (k=5)	15/0 (k=6)	40/1 (k=7)	35/0 (k=8)
国际到达（I）	航站楼 T	35/0 (k=9)	40/1 (k=10)	20/0 (k=11)	30/1 (k=12)
	卫星厅 S	40/1 (k=13)	45/2 (k=14)	30/1 (k=15)	20/0 (k=16)

同时为了计算紧张度，考虑捷运时间和行走时间，本文又设计了$CostAllTime_k$来表示选择k方案所需的总转运时间，由于需要考虑登机口在航站楼所处的位置，即(S_{North} /S_{South}/ S_{Center}/ S_{East}/ T_{North}/ T_{South}/ T_{Center})，所以k的取值范围为$0,1,2 \dots 196$。当$k = 0$时意味着，到达和出发的航班中至少有一个航班被分配至临时停靠机位，此时$CostAllTime_k = 0$，意味着不再计算这部分转乘旅客的总转运时间。

通过将旅客所需的时间编码，可以将变量需要表达的状态进行压缩。原本需要四个变量：到达航班，到达航班的登机口，出发航班，出发航班的登机口才能够确定旅客需要的时间，现在只需要找出其对应的k值即可。压缩状态可以减少变量的数量，加速模型的求解，同时可以将问题二和问题三用同样的模型表达。

4 问题一模型求解及算法设计

4.1 问题定义

问题一：本题只考虑航班-登机口分配。作为分析新建卫星厅对航班影响问题的第一步，首先要建立数学优化模型，尽可能多地分配航班到合适的登机口，并且在此基础上最小化被使用登机口的数量。本问题不需要考虑中转旅客的换乘。

问题的求解的优先级为：先确定最多能够分配多少个航班到固定登机位，在确定了这个解之后，确定最少需要使用多少固定登机位。

4.2 数学模型建立及求解

4.2.1 数学模型的建立

针对问题一，我们建立了两阶段的 0-1 规划数学模型，首先将尽可能多地分配航班到合适的登机口作为目标函数，并进行数学模型的求解。当数学模型求解完之后，我们将之前求出来的最多能够分配的航班数量作为一个约束条件，改变目标函数为最小化登机口的数量。此时能够保证所求出的航班-登机口分配方案是尽可能多地分配航班到合适的登机口并最小化被使用登机口数量的最优解。具体构建过程如下。

- **第一阶段数学模型：**

(1) 目标函数

在第一阶段求解的过程中，我们需要最大化安排至固定登机口的航班数量，因此目标函数如下所示，其中x_{aj}表示是否将 a 飞机安排至 j 登机口，1 表示是：

$$\max 2 * \sum_{a \in A} \sum_{j \in S \cup T} x_{aj} \tag{4-1}$$

(2) 约束条件

约束 1：保证任一个飞机都分配至并仅分配至 1 个登机口(包含临时登机口)

$$\forall a \in A: \sum_{j \in S \cup T \cup P} x_{aj} = 1 \tag{4-2}$$

约束 2：保证任一个飞机都分配到属性相同的登机口

$$\forall a \in A, \forall j \in S \cup T \cup P:$$

$$x_{aj} = 0, if\ Type_{Arrive_a} \notin ArriveType_j \tag{4-3}$$

$$x_{aj} = 0, if\ Type_{Arrive_a} \notin ArriveType_j \tag{4-4}$$

$$x_{aj} = 0, if\ Size_a \notin PlaneSize_j \tag{4-5}$$

约束 2.1 保证的是a飞机到达航班种类满足j登机口到达航班的种类要求，约束 2.2 保证的是a飞机出发航班种类满足j登机口出发航班的种类要求，约束 2.3 保证的是a飞机的机型满足j登机口的机型要求。如果有任意一个约束不满足，说明飞机和登机口属性不匹配，该a飞机不能停靠在该j登机口。

约束 3：分配在同一登机口的两飞机之间的空挡间隔时间必须大于等于 45 分钟

$$\forall a_1, a_2 \in A, \forall j \in S \cup T:$$

$$x_{a_1 j} + x_{a_2 j} \leq 1,$$
$$if\ Time_{Arrive_{a_1}} \leq Time_{Arrive_{a_2}}\ and\ Time_{Leave_{a_1}} + 45 > Time_{Arrive_{a_2}} \tag{4-6}$$

当a_1飞机的起飞时间和a_2的降落时间不满足间隔 45 分钟间隔的关系时，它们不能分配至同一固定登机口，即$x_{a_1 j} + x_{a_2 j} \neq 2$。换言之，如果两架飞机占用的时间间隔有交集，则不能停靠在一个登机口。

- **第二阶段数学模型：**

在求解完第一阶段模型后，我们可以得出其优化目标的取值，并将其定义为$MaxFlightNum$，即最多可以安排在固定登机位的航班数量。在第二阶段数学模型中，我们改变目标函数，将最小化被使用登机口的数量作为目标函数，同时增添约束，保证最后安排到固定登机口的航班数量等于$MaxFlightNum$。

(1) 目标函数

在第二阶段求解的过程中，我们需要最小化被使用登机口的数量，因此目标函数如下所示，其中z_j表示是否使用过j登机口，1 表示是：

$$\min \sum_{j \in S \cup T} z_j \tag{4-7}$$

(2) 约束条件

保持阶段一的约束 1-3，同时新增：

约束 4：保证安排至固定登机口的航班数量是之前阶段一求解出来的目标函数值

$$2 * \sum_{a \in A} \sum_{j \in S \cup T} x_{aj} = MaxFlightNum \tag{4-8}$$

约束 5：任意一个固定登机口，如果至少有一个飞机被分配到该登机口，那么说明该登机口被使用过

$$\forall j \in S \cup T: Mz_j \geq \sum_{a \in A} x_{aj} \tag{4-9}$$

其中，M是一个充分大的正整数，保证任意一个x_{aj}取值为 1 的时候z_j都等于 1。

4.2.2 数学模型的求解

本文使用 CPLEX 优化器，用编程语言表达出上述线性规划并进行求解。IBM ILOG CPLEX 优化器提供了用于求解线性规划（LP）和相关问题的 C、C++、Java 和 Python 库。具体而言，它求解线性或二次约束优化问题，其中，要优化的目标可表达为线性函数或凸二次函数。模型中的变量可声明为连续变量，或进一步约束为只能具有整数值。问题一的模型为 01 整数规划模型，**共有 21279 个整数变量，有 1009063 个约束**。

在求解第一阶段数学模型时，我们通过 Python 调用 CPLEX 求得$\boldsymbol{MaxFlightNum = 512}$，即成功安排了 512 次航班，调度 256 架飞机至固定停机位，安排了 94 次航班，47 架飞机去临时停靠机位。模型二求解出来的结果为，

最少使用的登机口数量为 65 个。具体结果展示如下：

- 给出成功分配到登机口的航班数量和比例，按宽、窄体机分别画柱状图

表 4-1 问题一 CPLEX 求解成功分配到登机口的航班数量和比例

成功分配到登机口的航班数量	256*2=512
成功分配到登机口的航班比例	256/303=84.49%
成功分配到登机口的宽体机航班数量	49*2=98
宽体机航班总数量	49*2=98
宽体机分配成功比例	49/49=100%
成功分配到登机口的窄体机航班数量	207*2=414
窄体机航班总数量	254*2=508
窄体机分配成功比例	207/254=81.50%

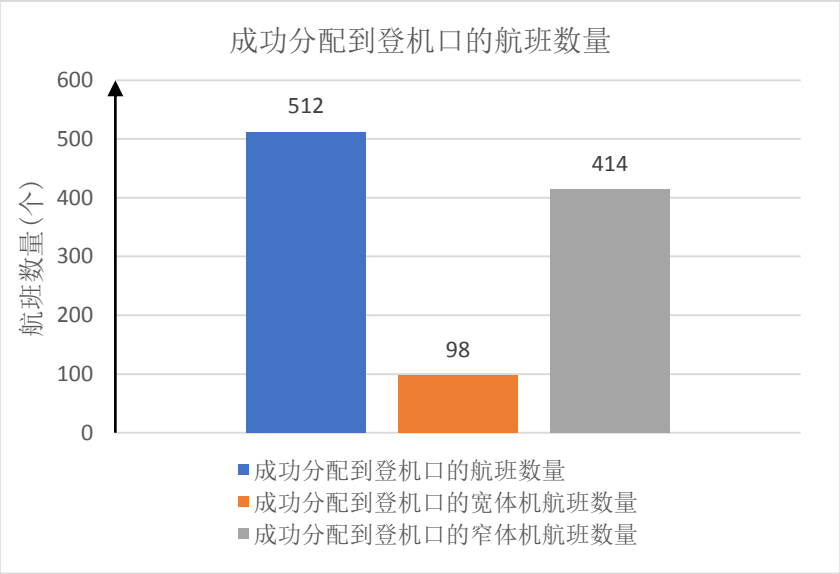

图 4-1 问题一 CPLEX 求解成功分配到登机口的航班数量柱状图

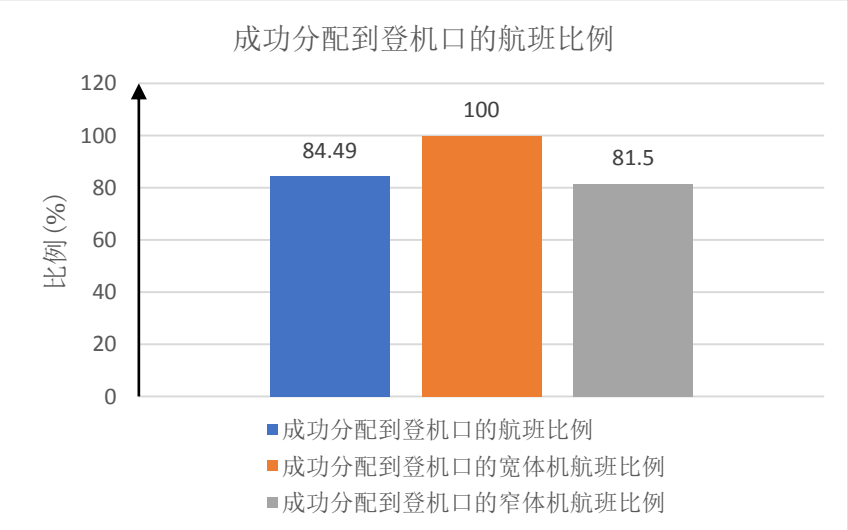

图 4-2 问题一 CPLEX 求解成功分配到登机口的航班比例柱状图

- 给出 T 和 S 登机口的使用数目和被使用登机口的平均使用率，要求画柱状图

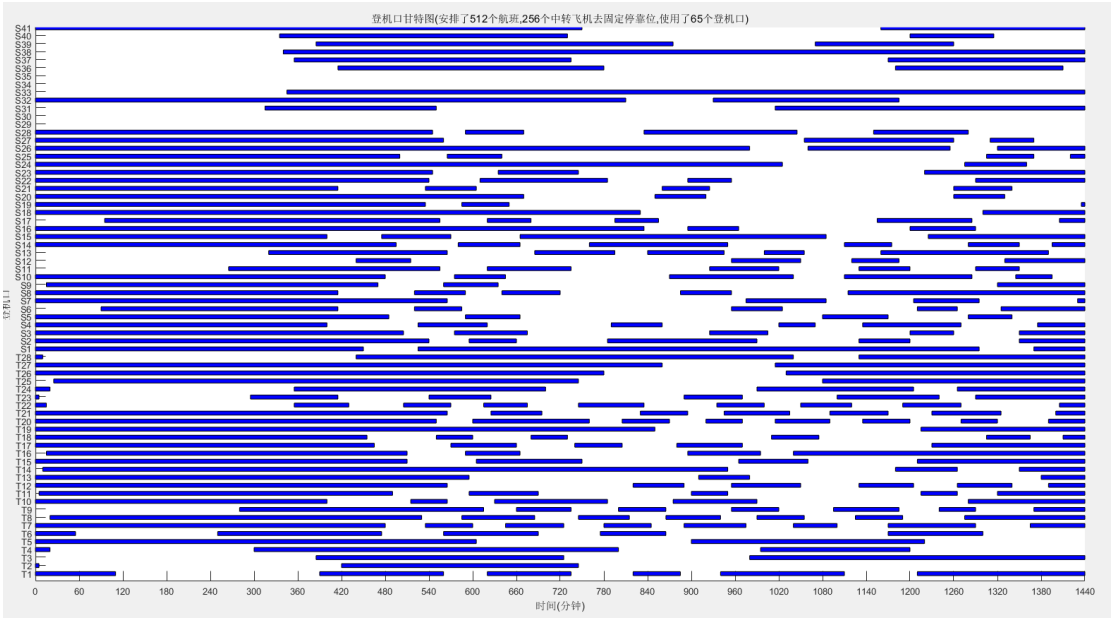

图 4-3 问题一 CPLEX 求解各个登机口甘特图

上图是根据航班-登机口分配方案画出的各个对应登机口的甘特图，根据图片我们发现，在保证成功分配到登机口的航班数量为 512 的情况下，最少使用的登机口数量是 65，不使用的登机口数量为 4 (S_{29}，S_{30}，S_{34}，S_{35})，具体的数据信息显示如下：

表 4-2 问题一 CPLEX 求解登机口的使用数目和被使用登机口的平均使用率数据

T 航站楼使用的登机口数目	28
T 航站楼使用登机口的平均使用率	61.36%
S 航站楼使用的登机口数目	37
S 航站楼使用登机口的平均使用率	58.51%

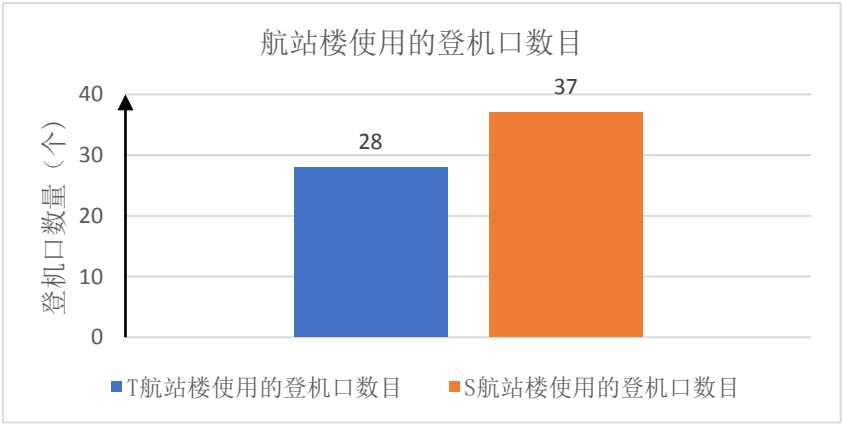

图 4-4 问题一 CPLEX 求解航站楼使用的登机口数目柱状图

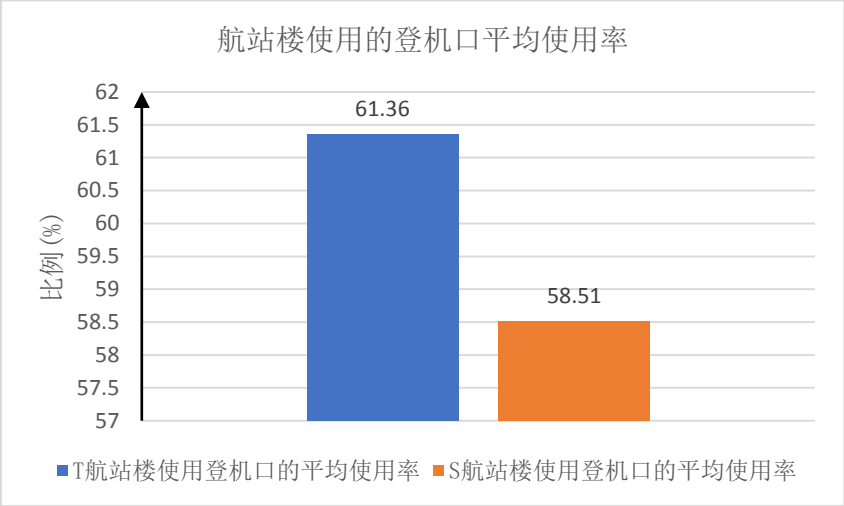

图 4-5 问题一 CPLEX 求解航站楼使用的登机口平均使用率柱状图

4.3 贪婪算法的设计及求解

4.3.1 贪婪算法的设计

问题一的目标在于尽可能多地分配航班到合适的登机口，减少分配到临时停机位的航班数量。我们可以对问题做如下抽象：对于每一架飞机，将其从降落到起飞到起飞后 45 分钟这一段时间看作一个线段，对于每一个登机口，将其在 8 月 20 号一整天的时间看作一个线段，我们的任务就是尽可能地安排飞机，将登机口的时间填满，如果有无法安排的飞机则将其安排到临时停机位。该问题的一种特殊情况是仅存在一个登机口，此时问题简化为经典的区间调度问题[6]。

经典的区间调度问题的一个例子是排课问题：给定若干课程的起止时间，问在一个教室内最多可以排多少门课程。对于排课问题，可以证明如下的贪心算法可以取得最优解：

a. 选择**结束时间**最早的课程，将该课程排入课表
b. 从课程集合中删除于该课程时间冲突的其他课程
c. 重复 a 到 b 直到课程集合为空

与排课问题相比，问题一有如下几个不同点：第一，存在多个可以进行调度的登机口；第二，飞机只能安排到属性相同的登机口。因此，从贪心算法出发，我们使用如下的贪婪算法：

a. 按照**结束时间**对所有飞机排序，该结束时间等于飞机起飞的时间加 45 分钟的空档间隔时间
b. 选取结束时间最早的飞机，根据该飞机的属性求出候选登机口集合，该飞机可以安排在候选登机口集合中的任意一个登机口
c. 按照**优先级**对候选登机口排序，选择出优先级最低的一个子集
d. 在该子集中按照**忙碌时间**对候选登机口排序，选择出忙碌时间最大的登机口，该忙碌时间等于该登机口已经安排的最后一架飞机的结束时间
e. 重复 b 到 d 直到飞机集合为空

首先，该贪婪算法对所有飞机按照结束时间排序，保证先离开的飞机先被调度，该调度模式与经典的区间调度问题的调度模式一致。

然后，算法在候选登机口中选取优先级最低的登机口。登机口的优先级由该登机口的属性决定，属性越灵活的登机口其优先级越高，例如：对于降落和起飞航班都要求是国内航班的登机口，其优先级较低；而对于允许国内和国际航班降落和起飞的登机口，其优先级较高。先调度优先级低的登机口的目的在于**预留更多的资源留给后面的飞机**，因为优先级更高的登机口更灵活，能停靠的飞机的种类更丰富。

接下来，在相同优先级的候选登机口中，选取忙碌时间最大的登机口，以此尽量减少使用登机口的数量。

该贪婪算法的时间复杂度为$O(|A|\log|A| + |A|(|T| + |S|))$。其中，$O(|A|\log|A|)$用于对飞机进行排序，$O(|A|(|T| + |S|))$用于枚举每一架飞机和每一个登机口的组合。该算法复杂度为线性。

该算法最终需要调度 50 架飞机（100 次航班）至临时停机位，共使用 66 个登机口。使用模型求解出的问题一的最优解为调度 47（94 次航班）架飞机至临时停机位，共使用 65 个登机口。

步骤 b，c 和 d 对于优化问题一都有显著影响，并且选取登机口说考虑要素的顺序也有重要的影响。如果先选取忙碌时间最大的登机口，再考虑优先级的话，该算法会调度 73 架飞机至临时停机位；如果不按照结束时间对飞机排序，则需要调度 84 架飞机至临时停机位。

4.3.2 贪婪算法的求解

本文通过设计贪婪算法，最终求得$\boldsymbol{MaxFlightNum = 510}$，即成功安排了 510 次航班，255 个飞机至固定停机位，**最少使用的登机口数量为 66 个**，具体结果显示如下：

- 给出成功分配到登机口的航班数量和比例，按宽、窄体机分别画柱状图。

表格 4-3 问题一贪婪算法求解成功分配到登机口的航班数量和比例

成功分配到登机口的航班数量	255*2=510
成功分配到登机口的航班比例	255/303=84.16%
成功分配到登机口的宽体机航班数量	49*2=98
宽体机航班总数量	49*2=98
宽体机分配成功比例	49/49=100%
成功分配到登机口的窄体机航班数量	206*2=412
窄体机航班总数量	254*2=508
窄体机分配成功比例	206/254=81.10%

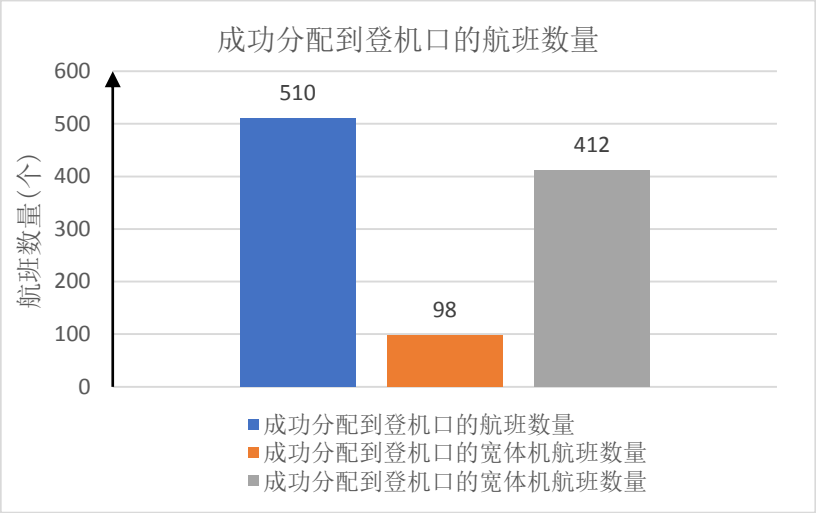

图 4-6 问题一贪婪算法求解成功分配到登机口的航班数量柱状图

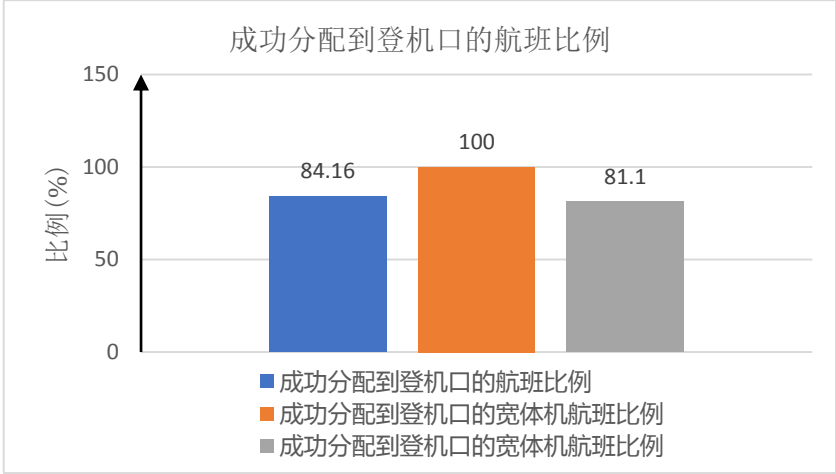

图 4-7 问题一贪婪算法求解成功分配到登机口的航班比例柱状图

- 给出 T 和 S 登机口的使用数目和被使用登机口的平均使用率，要求画柱状图。

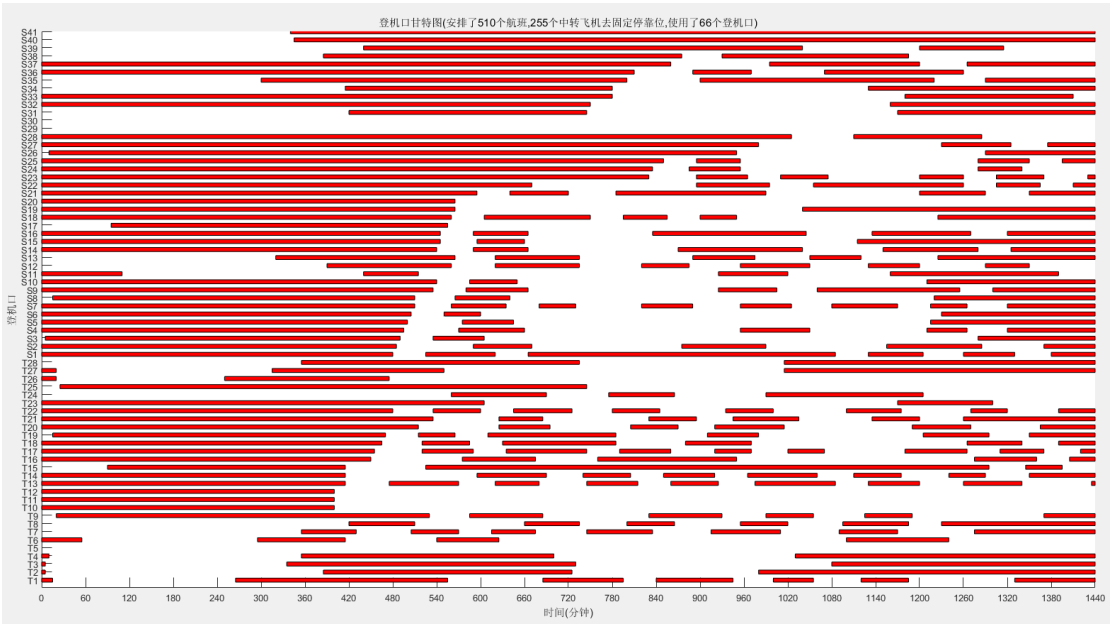

图 4-8 问题一贪婪算法求解各个登机口甘特图

上图是根据航班-登机口分配方案画出的各个对应登机口的甘特图，根据图片我们发现，在保证成功分配到登机口的航班数量为 510 的情况下，最少使用的登机口数量是 66，不使用的登机口数量为 3 (T_5，S_{29}，S_{30})，具体的数据信息显示如下：

表 4-4 问题一贪婪算法求解登机口的使用数目和被使用登机口的平均使用率数据

T 航站楼使用的登机口数目	27
T 航站楼使用登机口的平均使用率	51.39%
S 航站楼使用的登机口数目	39
S 航站楼使用登机口的平均使用率	63.55%

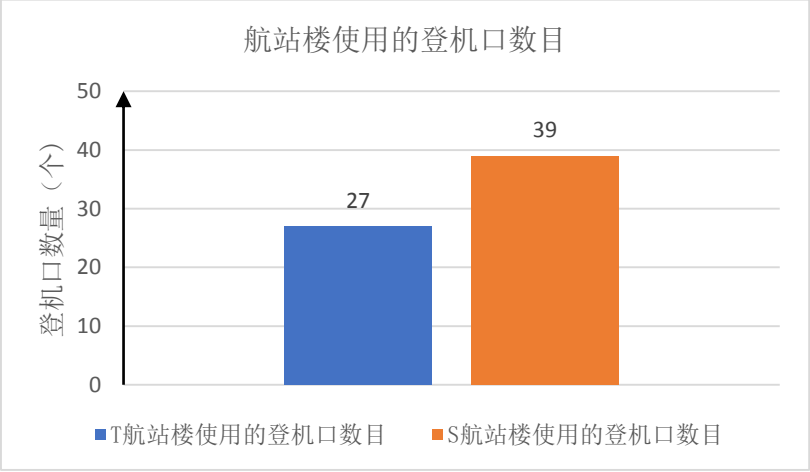

图 4-9 问题一贪婪算法求解航站楼使用的登机口数目柱状图

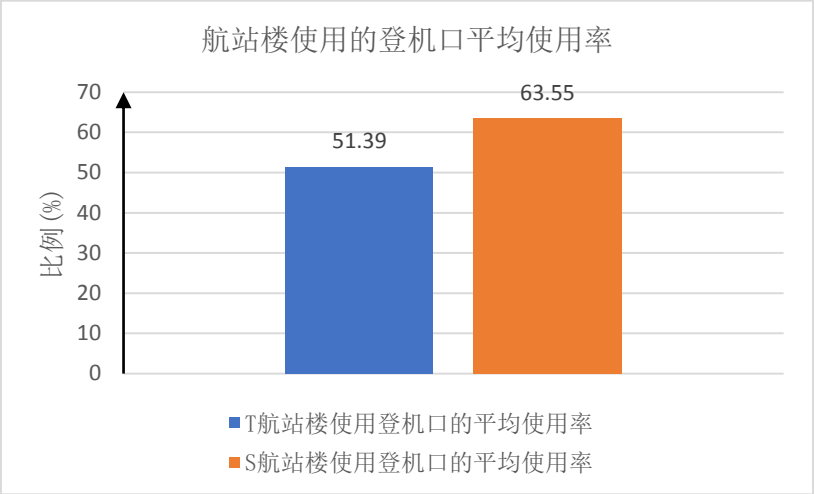

图 4-10 问题一贪婪算法求解航站楼使用的登机口平均使用率柱状图

4.4 数据分析及结论

根据 CPLEX 求解器求解出数学模型的结果以及贪婪算法求解出的结果，我们获得如下数据结果：

表 4-5 问题一 CPLEX 求解器求解结果与贪婪算法求解结果分析

	CPLEX 求解结果	贪婪算法求解结果
成功分配到登机口的航班数量	**512**	510
成功分配到登机口的航班比例	84.49%	84.16%
成功分配到登机口的宽体机航班数量	98	98
宽体机航班总数量	98	98
宽体机分配成功比例	100%	100%
成功分配到登机口的窄体机航班数量	414	412
窄体机航班总数量	508	508
窄体机分配成功比例	81.50%	81.10%
T 航站楼使用的登机口数目	28	27
T 航站楼使用登机口的平均使用率	61.36%	51.39%
S 航站楼使用的登机口数目	37	39
S 航站楼使用登机口的平均使用率	58.51%	63.55%
总共使用的登机口数量	**65**	66
求解时间	9.39 秒	**<1 秒**

根据上述表格我们可以发现下述结论：

- CPLEX 求解器求解出来的航班-登机口分配方案优于贪婪算法求解出的航班-登机口分配方案。这是由于 CPLEX 求解器在求解的过程中不断地采用整数规划求解算法(分支定界和割平面)等算法去探寻最优解，而我们设计的贪婪算法只能保证局部最优解。但是贪婪算法的求解结果与 CPLEX 求解器的求解结果是非常相近的，也证明了我们贪婪算法的有效性。
- 贪婪算法求解的时间远小于 CPLEX 求解结果。当问题规模进一步扩大，无法用 CPLEX 求解器一下子进行问题的求解时，建议使用贪婪算法获得一个局部最优解，并将该局部最优解作为 CPLEX 求解器的初始输入，可以大大节约 CPLEX 的求解时间。
- 此外，通过对于数据的进一步分析，我们可以发现宽体机的分配比例是远大

于窄体机的分配比例的。这是由于窄机型航班占整个航班比例的 83%，但是能够容纳窄机型的登机口只占总登机口的 65%，因此在之后的登机口建设中，可以**多建设能够容纳窄机型的登机口**，或者将**一部分属性为宽机型的登机口重建为窄机型**以提高窄体机分配成功比例。

最终选择用 CPLEX 求解器求解出的航班-登机口方案作为最终航班-登机口方案，具体调度方案见附件。

5 问题二模型求解及算法设计

5.1 问题的定义

问题二：考虑中转旅客最短流程时间。本问题是在问题一的基础上加入旅客换乘因素，要求最小化中转旅客的总体最短流程时间，并且在此基础上最小化被使用登机口的数量。本题不考虑旅客乘坐捷运和步行时间。

问题的求解的优先级为，先确定最多能够分配多少个航班到固定登机位；在确定了这个解之后，需要确定最小化中转旅客的总体最短流程时间；在保证最多分配航班数目，最小化中转旅客的总体流程时间后，最小化被使用登机口的数量。

5.2 数学模型的建立及求解

5.2.1 数学模型的建立

针对问题二，我们同样设计了两阶段数学模型，复用问题一求解出的最多分配航班数目，先建立第一阶段数学模型最小化中转旅客的总体最短流程时间。当第一阶段数学模型求解完之后，我们将之前求出来的最小化中转旅客的总体最短流程时间作为一个约束条件，改变目标函数为最小化登机口的数量。此时能够保证题目所要求的最优解。具体构建过程如下：

- **第一阶段数学模型：**

(1) 目标函数

在第一阶段求解的过程中，我们需要最小化中转旅客的总体最短流程时间，因此目标函数可设为如下所示，其中$Num_{Arrive_{a_1}Leave_{a_2}}$表示乘坐$a_1$飞机降落航班，转乘$a_2$飞机起飞航班的乘客数目，属于事先处理出来的数据，为常量；$ProcessTime_{Arrive_{a_1}Leave_{a_2}}$表示乘坐$a_1$飞机降落航班，转乘$a_2$飞机起飞航班所花费的最短流程时间，为决策变量；$y_{Arrive_{a_1}Leave_{a_2}k}$ 表示乘坐a_1飞机降落航班，转乘a_2飞机起飞航班是否对应 k 方案所需的最短流程时间，1 表示是：

$$\min \sum_{a_1 \in A} \sum_{a_2 \in A} Num_{Arrive_{a_1}Leave_{a_2}} * ProcessTime_{Arrive_{a_1}Leave_{a_2}} \tag{5-1}$$

(2) 约束条件

保持问题一的约束 1-5，同时新增：

约束 6：保证任一个航班换乘都分配至一种流程时间方案

$$\forall a_1 \in A, \forall a_2 \in A: \sum_{k=0}^{16} y_{Arrive_{a_1}Leave_{a_2}k} = 1 \tag{5-2}$$

约束 7：任意一次航班换乘中有一架或两架飞机被分配至临时停靠机位时，选择$k = 0$的方案，即$y_{Arrive_{a_1}Leave_{a_2}0} = 1$

$$\forall a_1 \in A, \forall a_2 \in A:$$

$$\sum_{j\in P} x_{a_1 j} \le y_{Arrive_{a_1} Leave_{a_2} 0} \tag{5-3}$$

$$\sum_{j\in P} x_{a_2 j} \le y_{Arrive_{a_1} Leave_{a_2} 0} \tag{5-4}$$

$$y_{Arrive_{a_1} Leave_{a_2} 0} \le \sum_{j\in P} x_{a_1 j} + \sum_{j\in P} x_{a_2 j} \tag{5-5}$$

约束 8：根据$x_{a_1 j}$，$x_{a_1 j}$，$Type_{Arrive_{a_1}}$和$Type_{Leave_{a_2}}$的，建立关于$y_{Arrive_{a_1} Leave_{a_2} k}$的约束：

$$\forall a_1 \in A, \forall a_2 \in A, \forall j_1 \in S \cup T, \forall j_2 \in S \cup T:$$

$k = 1$的方案：

$$x_{a_1 j} + x_{a_2 j} - y_{Arrive_{a_1} Leave_{a_2} 1} \le 1$$
$$if\ j_1 \in T, j_2 \in T\ , Type_{Arrive_{a_1}} = D\ and\ Type_{Leave_{a_2}} = D \tag{5-6}$$

$k = 2$的方案：

$$x_{a_1 j} + x_{a_2 j} - y_{Arrive_{a_1} Leave_{a_2} 2} \le 1$$
$$if\ j_1 \in T, j_2 \in S\ , Type_{Arrive_{a_1}} = D\ and\ Type_{Leave_{a_2}} = D \tag{5-7}$$

…

$k = 16$的方案：

$$x_{a_1 j} + x_{a_2 j} - y_{Arrive_{a_1} Leave_{a_2} 16} \le 1$$
$$if\ j_1 \in S, j_2 \in S\ , Type_{Arrive_{a_1}} = I\ and\ Type_{Leave_{a_2}} = I \tag{5-22}$$

其中，$k = 1$的方案代表降落航班和出发航班都属于国内航班，且都停靠在 T 航站楼所对应的流程时间。

约束 9：a_1飞机降落航班的时间加上对应的最短流程时间小于a_2飞机起飞航班的时间，才算换乘成功，$CanTransfer_{Arrive_{a_1} Leave_{a_2}}$表示乘坐$a_1$飞机降落航班，转乘$a_2$飞机起飞航班是否转乘成功，1 表示失败；此外，如果$y_{Arrive_{a_1} Leave_{a_2} 0} = 1$，意味着以$k = 0$的方案换乘成功，即$CanTransfer_{Arrive_{a_1} Leave_{a_2}} = 1$，防止在目标函数中重复计算由于飞机停入临时停靠位带来的损失。

$$\forall a_1 \in A, \forall a_2 \in A:$$

$$\sum_{k=0}^{16} y_{Arrive_{a_1}Leave_{a_2}k} * CostTime_k + M * CanTransfer_{Arrive_{a_1}Leave_{a_2}} \leq M + Time_{Leave_{a_2}} - Time_{Arrive_{a_1}} \tag{5-23}$$

$$CanTransfer_{Arrive_{a_1}Leave_{a_2}} \geq y_{Arrive_{a_1}Leave_{a_2}0} \tag{5-24}$$

M是一个充分大的正整数。

约束 10：计算$ProcessTime_{Arrive_{a_1}Leave_{a_2}}$。换乘成功时该时间取决于$k$，失败时定义为 6 小时

$$\forall a_1 \in A, \forall a_2 \in A:$$

代表换乘成功的情况：

$$ProcessTime_{Arrive_{a_1}Leave_{a_2}} - \sum_{k=0}^{16} y_{Arrive_{a_1}Leave_{a_2}k} * CostTime_k + CanTransfer_{Arrive_{a_1}Leave_{a_2}} \geq 1 \tag{5-25}$$

代表换乘失败的情况：

$$ProcessTime_{Arrive_{a_1}Leave_{a_2}} + 6 * 60 * CanTransfer_{Arrive_{a_1}Leave_{a_2}} \geq 6 * 60 \tag{5-26}$$

如果换乘成功，即$CanTransfer_{Arrive_{a_1}Leave_{a_2}} = 1$：

$$ProcessTime_{Arrive_{a_1}Leave_{a_2}} = \sum_{k=0}^{16} y_{Arrive_{a_1}Leave_{a_2}k} * CostTime_k$$

否则$ProcessTime_{Arrive_{a_1}Leave_{a_2}} = 6 * 60$分钟。

- **第二阶段数学模型：**

在求解完第一阶段模型后，我们可以得出其优化目标的值，并将其定义为$MinProcessTime$，即最少的中转旅客的总体流程时间。在第二阶段数学模型中，我们改变目标函数，将最小化被使用登机口的数量作为目标函数，同时增添约束，保证最后安排到固定登机口的航班数量等于$MaxFlightNum$且最少的中转旅客的总体流程时间为$MinProcessTime$。

(1) 目标函数

在第二阶段求解的过程中，我们需要最小化被使用登机口的数量，因此目标函数可设为如下所示，其中z_j表示是否使用过j登机口，1 表示是：

$$\min \sum_{j \in S \cup T} z_j \tag{5-27}$$

(2) 约束条件

保持阶段一的约束 1-10，同时新增：

约束 11：保证中转旅客的总体流程时间是之前阶段一求解出来的目标函数

$$\sum_{a_1 \in A} \sum_{a_2 \in A} Num_{Arrive_{a_1} Leave_{a_2}} * ProcessTime_{Arrive_{a_1} Leave_{a_2}} = MinProcessTime \tag{5-28}$$

5.2.2 数学模型的求解

本文通过 Python 调用 CPLEX 的方法，用编程语言表达出上述算法并进行求解。求解结果如下。问题二我们建立的数学模型是整数规划模型，**有 36849 个整数变量，有 5103490 个约束**，因此求解起来非常的困难。

在求解第一阶段数学模型时，我们通过 Python 调用 CPLEX 求得 $\boldsymbol{MaxFlightNum = 512}$，即成功安排了 512 个航班，256 个飞机至固定停机位，$\boldsymbol{MinProcessTime = 55490}$。阶段二求解出来的结果为，**最少使用的登机口数量为 66 个**。具体结果显示如下：

- 成功分配到登机口的航班数量和比例，按宽、窄体机分别画柱状图。

表 5-1 问题二 CPLEX 求解成功分配到登机口的航班数量和比例

成功分配到登机口的航班数量	256*2=512
成功分配到登机口的航班比例	256/303=84.49%
成功分配到登机口的宽体机航班数量	49*2=98
宽体机航班总数量	49*2=98
宽体机分配成功比例	49/49=100%
成功分配到登机口的窄体机航班数量	207*2=414
窄体机航班总数量	254*2=508
窄体机分配成功比例	207/254=81.50%

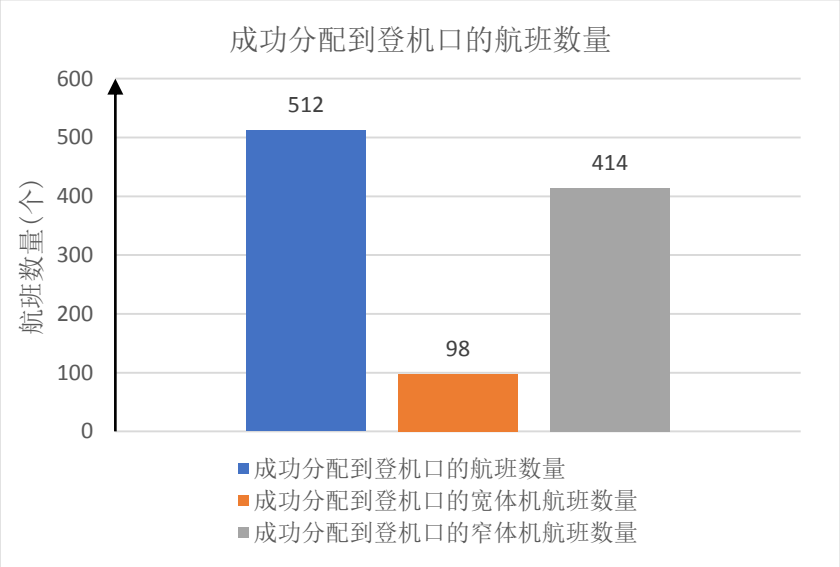

图 5-1 问题二 CPLEX 求解成功分配到登机口的航班数量柱状图

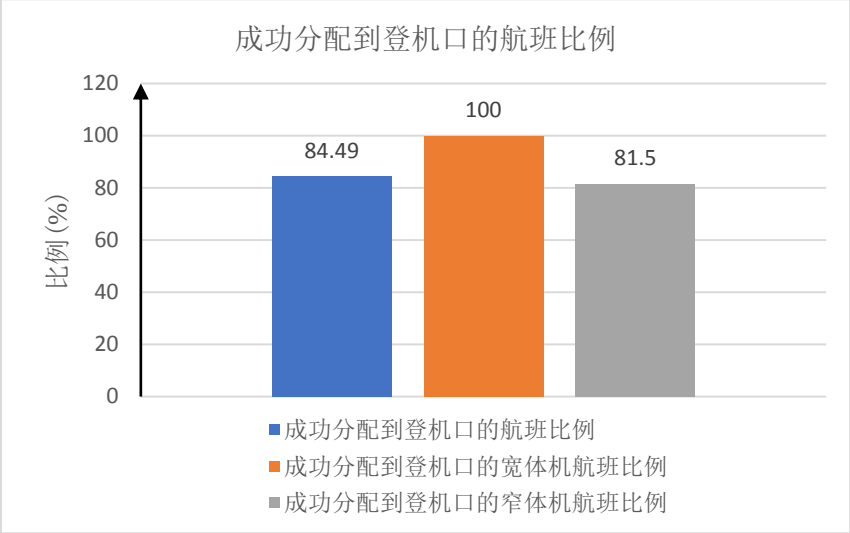

图 5-2 问题二 CPLEX 求解成功分配到登机口的航班比例柱状图

- 给出 T 和 S 登机口的使用数目和被使用登机口的平均使用率，要求画柱状图。

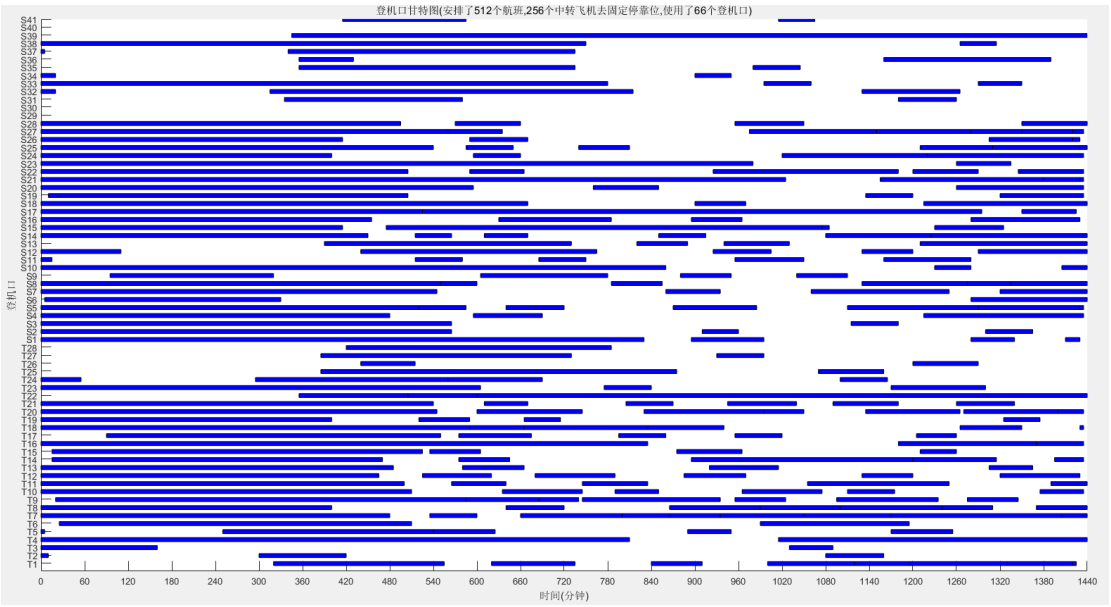

图 5-3 问题二 CPLEX 求解各个登机口甘特图

上图是根据航班-登机口分配方案画出的各个对应登机口的甘特图，根据图片我们发现，在保证成功分配到登机口的航班数量为 512 的情况下，最少旅客中转时间为$MinProcessTime = 55490$的情况下，最少使用的登机口数量是 66，不使用的登机口数量为 3 (S_{29}，S_{30}，S_{40})，具体的数据信息显示如下：

表 5-2 问题二 CPLEX 求解登机口的使用数目和被使用登机口的平均使用率数据

T 航站楼使用的登机口数目	28
T 航站楼使用登机口的平均使用率	58.61%
S 航站楼使用的登机口数目	38
S 航站楼使用登机口的平均使用率	59.43%

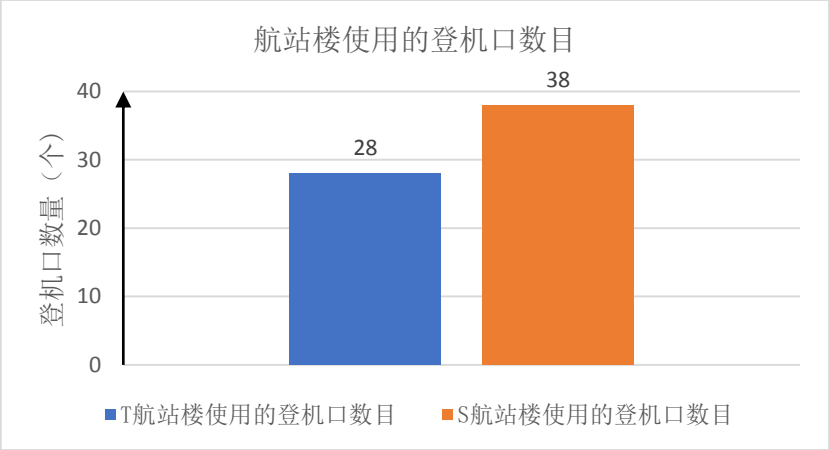

图 5-4 问题二 CPLEX 求解航站楼使用的登机口数目柱状图

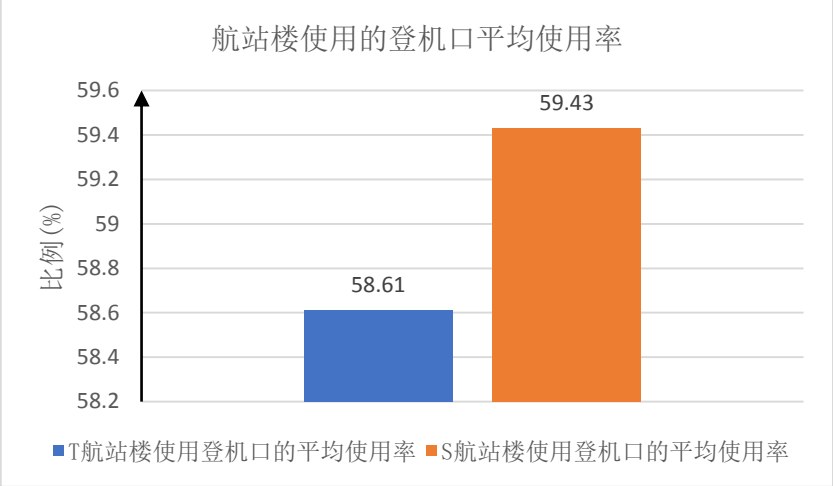

图 5-5 问题二 CPLEX 求解航站楼使用的登机口平均使用率柱状图

- 给出换乘失败旅客数量和比率

表 5-3 问题二 CPLEX 求解换乘失败旅客数量和比率

换乘失败旅客数量	0
换乘失败旅客比例	0

注：因临时停靠位损失的旅客不算做换乘失败

5.3 模拟退火算法的设计及求解

5.3.1 模拟退火算法的设计

相比较问题一，问题二的算法设计存在如下挑战：第一，优化的目标为流程时间，而每位旅客的流程时间与两次航班有关；第二，需要在问题一的基础上优化，即不能够为了减少流程时间而增加停靠在临时停机位的飞机数量。

在使用启发式算法求解问题一的过程中，我们发现最终有多种将一架飞机安排到一个登机口的方案。换言之，如果画出问题一的调度方案的甘特图，我们发现可以将两个登机口的调度结果交换，而不影响停靠在临时停机位的飞机数量。例如，对于两个属性相同的登机口 1 和 2，如果可以将飞机 a，b 安排在登机口 1，将飞机 c，d，e 安排在登机口 2，那么一定可以将 c，d，e 安排在登机口 1，将 a，b 安排在登机口 2，原因在于飞机的属性和登机口的属性是吻合的，同时飞机之间不存在时间上的冲突。

受此启发，我们基于模拟退火算法设计算法二，思路类似于使用模拟退火算

法解决 TSP 问题[7]：我们不断交换相同属性的两个登机口的调度方案，并且以一定的概率接受该新的调度方案。算法流程如下：

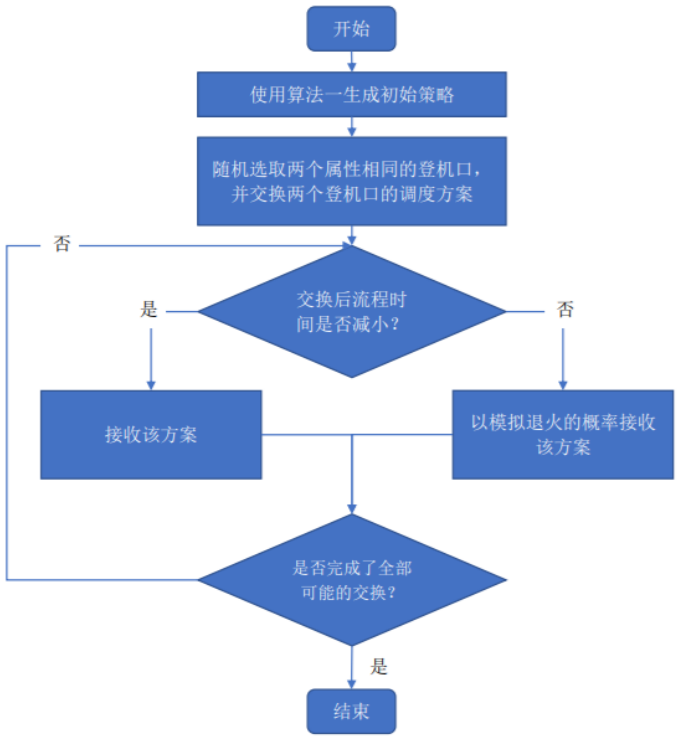

图 5-6 模拟退火算法流程图

在模拟退火算法中，有两个主要参数会影响算法的执行结果：初始温度T_0和降温速度r。如果初始温度较大，降温速度较慢，则算法有更高的概率会选取流程时间增大的解；如果初始温度较低，降温速度较快，则只有在流程时间减小的情况下算法才会更新调度方案。下表展示了不同的初始温度和降温速度对问题二求解结果的影响。

表 5-4 参数对模拟退火算法性能的影响

r \ T_0	10	100	1000	10000	100000
0.99	62075	62075	65155	64565	64565
0.94	62075	62075	65265	64565	64565
0.89	62075	62075	65040	64565	64565
0.84	62075	62075	62080	64565	65285
0.79	62075	62075	62080	64565	64565

可以看出，在不同的参数组合下，算法二均收敛到了局部最优值 62075 分钟。尽管算法二求出的流程时间小于模型二求解出的流程时间（55490 分钟），但是相比起使用 CPLEX 求解混合整数规划模型，算法二的运行时间和所需要的计算存储资源都大大减少了。CPLEX 求解模型在使用 32 个线程的情况下所需要的时间超过 48 个小时，而算法二在单线程下的运行时间仅需要不到 1 分钟；CPLEX 求解模型需要使用超过 20GB 的内存，而算法二所占用的内存不超过 1MB。

5.3.2 模拟退火算法的求解

本文通过设计模拟退火算法，最终求得$\boldsymbol{MaxFlightNum = 510}$，即成功安排了 510 个航班，255 个飞机至固定停机位，$\boldsymbol{MinProcessTime = 62075}$，**最少使用的登机口数量为 66 个**。具体结果显示如下：

- 成功分配到登机口的航班数量和比例，按宽、窄体机分别画柱状图。

表 5-5 问题二模拟退火求解成功分配到登机口的航班数量和比例

成功分配到登机口的航班数量	255*2=510
成功分配到登机口的航班比例	255/303=84.16%
成功分配到登机口的宽体机航班数量	49*2=98
宽体机航班总数量	49*2=98
宽体机分配成功比例	49/49=100%
成功分配到登机口的窄体机航班数量	206*2=412
窄体机航班总数量	254*2=508
窄体机分配成功比例	206/254=81.10%

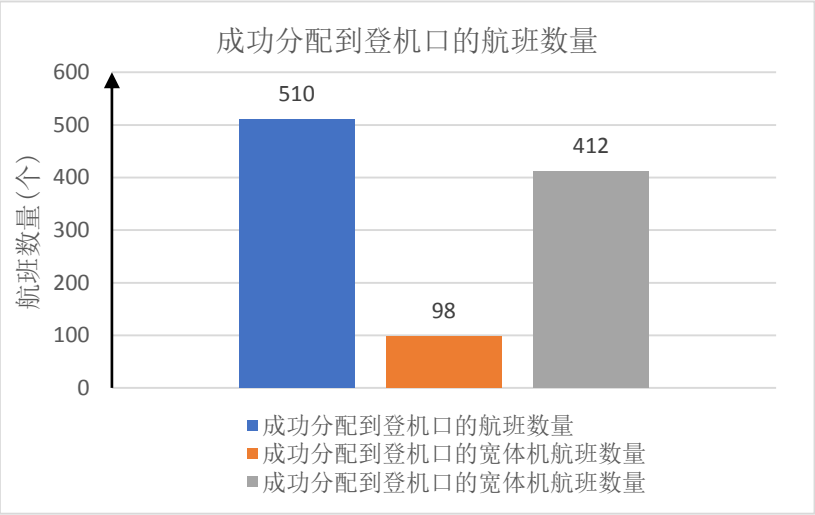

图 5-7 问题二模拟退火求解成功分配到登机口的航班数量柱状图

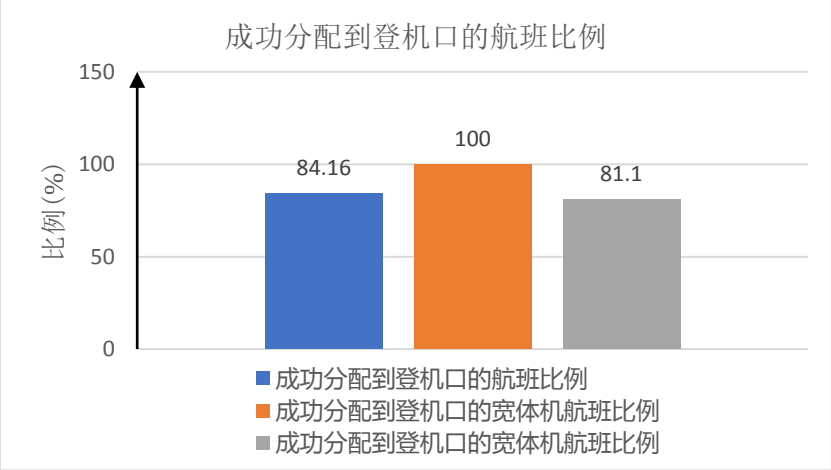

图 5-8 问题二模拟退火求解成功分配到登机口的航班比例柱状图

- 给出 T 和 S 登机口的使用数目和被使用登机口的平均使用率，要求画柱状图。

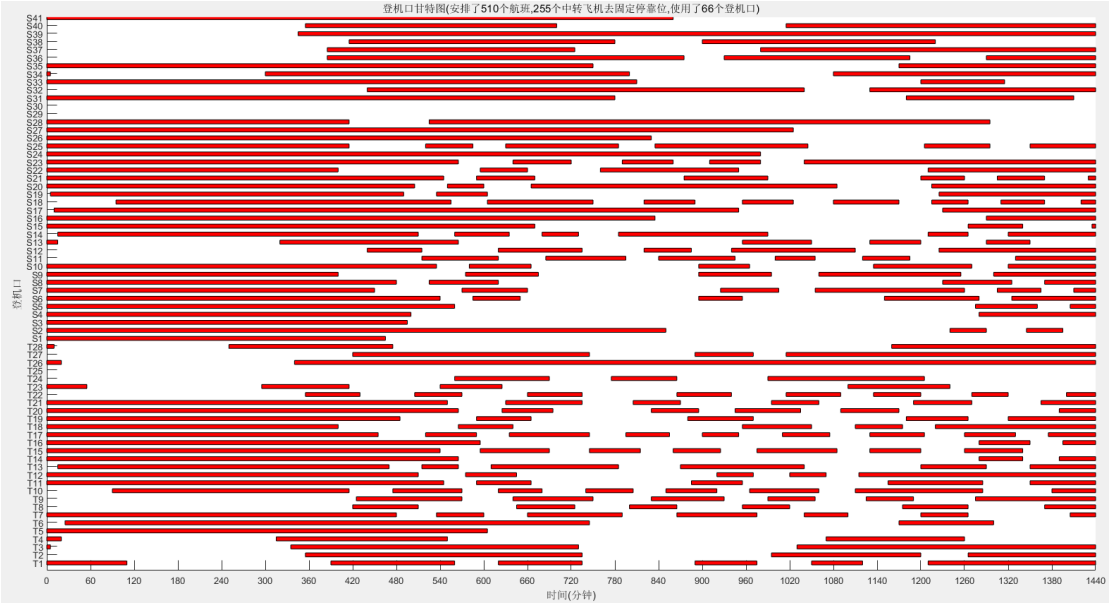

图 5-9 问题二模拟退火求解各个登机口甘特图

上图是根据航班-登机口分配方案画出的各个对应登机口的甘特图，根据图片我们发现，在保证成功分配到登机口的航班数量为 510 的情况下，最少旅客中转时间为$MinProcessTime = 62075$的情况下，最少使用的登机口数量是 66，不使用的登机口数量为 3 (T_{25}，S_{30}，S_{40})，具体的数据信息显示如下：

表 5-6 问题二模拟退火求解登机口的使用数目和被使用登机口的平均使用率数据

T 航站楼使用的登机口数目	27
T 航站楼使用登机口的平均使用率	54.06%
S 航站楼使用的登机口数目	39
S 航站楼使用登机口的平均使用率	60.57%

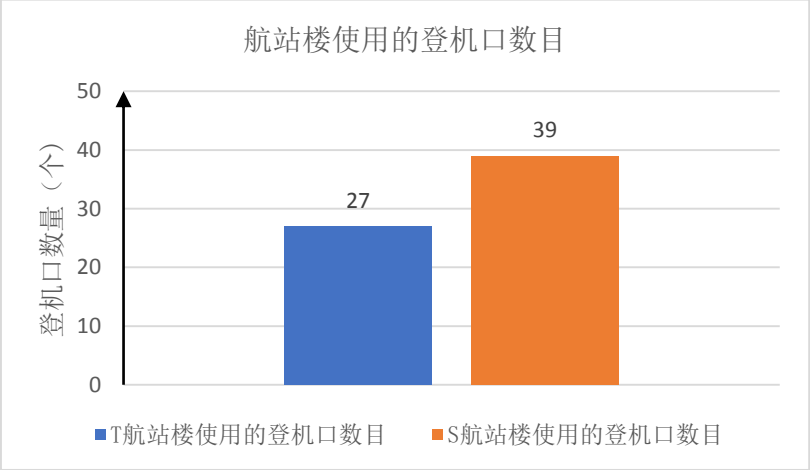

图 5-10 问题二模拟退火求解航站楼使用的登机口数目柱状图

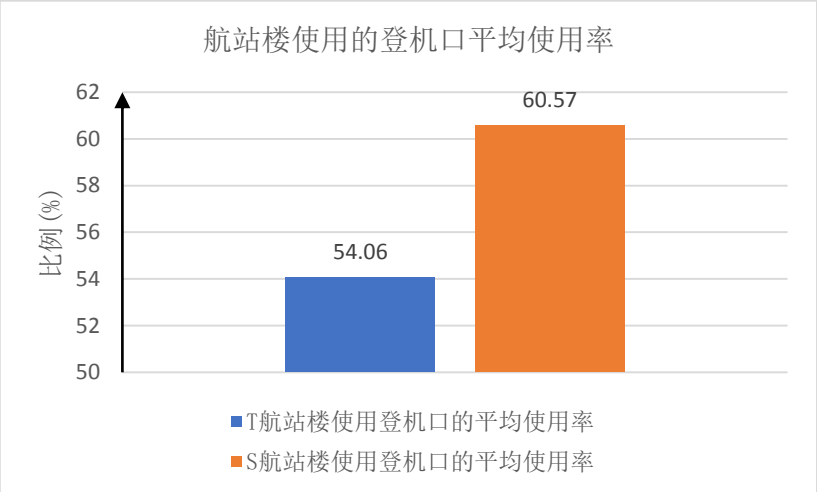

图 5-11 问题二模拟退火求解航站楼使用的登机口平均使用率柱状图

- 给出换乘失败旅客数量和比率

表 5-7 问题二模拟退火求解换乘失败旅客数量和比率

换乘失败旅客数量	0
换乘失败旅客比例	0

注：因临时停靠位损失的旅客不算做换乘失败

5.4 数据分析及结论

根据 CPLEX 求解器求解出数学模型的结果以及模拟退火算法求解出的结果，我们获得如下数据结果：

表 5-8 问题二 CPLEX 求解器求解结果与模拟退火算法求解结果分析

	CPLEX 求解结果	贪婪算法求解结果
成功分配到登机口的航班数量	**512**	510
成功分配到登机口的航班比例	84.49%	84.16%
成功分配到登机口的宽体机航班数量	98	98
宽体机航班总数量	98	98
宽体机分配成功比例	100%	100%
成功分配到登机口的窄体机航班数量	414	412
窄体机航班总数量	508	508
窄体机分配成功比例	81.50%	81.10%
最小旅客换乘时间	**55490**	62075
T 航站楼使用的登机口数目	28	27
T 航站楼使用登机口的平均使用率	58.61%	54.06%
S 航站楼使用的登机口数目	38	39
S 航站楼使用登机口的平均使用率	59.43%	60.57%
总共使用的登机口数量	66	66
求解时间	>48 小时	**<5 分钟**

根据上述表格我们可以发现，下述结论：

- CPLEX 求解器求解出来的航班-登机口分配方案优于模拟退火算法求解出的航班-登机口分配方案，在第一优化目标和第二优化目标上都优于，第三优化目标相同，但是这两者的 gap 都在 11%，并且第一优化目标相差得 gap 在 1%以内，证明模拟退火算法求解该类航班调度问题的有效性。

- 模拟退火算法求解的时间远小于 CPLEX 求解结果。在问题二的约束下，CPLEX 求解器在有限的时间内已经求解不出最优解了，因此我们建议先采用当问题规模较大时，采用模拟退火算法来进行求解。

最终选择用 CPLEX 求解器求解出的航班-登机口方案作为最终航班-登机口方案，具体调度方案见附件。

6 问题三模型求解及算法设计

6.1 问题的定义

问题三：考虑中转旅客的换乘时间。如前所述，新建卫星厅对航班的最大影响是中转旅客换乘时间的可能延长。因此，数学模型最终需要考虑换乘旅客总体紧张度的最小化，并且在此基础上最小化被使用登机口的数量。本问题可以在问题二的基础上细化，引入旅客换乘连接变量，并把中转旅客的换乘紧张度作为目标函数的首要因素。

问题的求解的优先级为，先确定最多能够分配多少个航班到固定登机位；在确定了这个解之后，需要确定最小化换乘旅客总体紧张度；在保证最多分配航班数目，最小化换乘旅客总体紧张度后，最小化被使用登机口的数量。

6.2 数学模型的建立及求解

6.2.1 数学模型的建立

针对问题二，我们同样设计了两阶段数学模型，复用问题一求解出的最多分配航班数目，先建立第一阶段数学模型最小化换乘旅客总体紧张度。当第一阶段数学模型求解完之后，我们将之前求出来的最小化换乘旅客总体紧张度作为一个约束条件，改变目标函数为最小化登机口的数量。此时能够保证题目所要求的最优解。具体构建过程如下：

- **第一阶段数学模型：**

(1) 目标函数

在第一阶段求解的过程中，我们需要最小化换乘旅客总体紧张度，因此目标函数可设为如下所示，其中$ConnectTime_{Arrive_{a_1}Leave_{a_2}}$表示乘坐$a_1$飞机降落航班，转乘$a_2$飞机起飞航班的航班连接时间，是常量；$TravelTime_{Arrive_{a_1}Leave_{a_2}}$表示乘坐$a_1$飞机降落航班，转乘$a_2$飞机起飞航班所花费的旅客换乘时间：

$$\min \sum_{a_1 \in A} \sum_{a_2 \in A} Num_{Arrive_{a_1}Leave_{a_2}} * \frac{TravelTime_{Arrive_{a_1}Leave_{a_2}}}{ConnectTime_{Arrive_{a_1}Leave_{a_2}}} \tag{6-1}$$

(2) 约束条件

保持问题二的约束 1-10，同时修改：

约束 6：保证任一个航班换乘都分配至一种方案

$$\forall a_1 \in A, \forall a_2 \in A: \sum_{k=0}^{196} y_{Arrive_{a_1}Leave_{a_2}k} = 1 \tag{6-2}$$

注意此处与模型二的区别在于k的取值范围，即总时间的取值范围大大增加了。

约束 8：根据$x_{a_1 j}$，$x_{a_1 j}$，$Type_{Arrive_{a_1}}$和$Type_{Leave_{a_2}}$，建立关于$y_{Arrive_{a_1}Leave_{a_2}k}$的约束：

$$\forall a_1 \in A, \forall a_2 \in A, \forall j_1 \in S \cup T, \forall j_2 \in S \cup T:$$

$k = 1$的方案：

$$x_{a_1 j} + x_{a_2 j} - y_{Arrive_{a_1} Leave_{a_2} 1} \leq 1$$

$$if\ j_1 \in T_{North}, j_2 \in T_{North}, Type_{Arrive_{a_1}} = D\ and\ Type_{Leave_{a_2}} = D \tag{6-3}$$

$k = 2$的方案：

$$x_{a_1 j} + x_{a_2 j} - y_{Arrive_{a_1} Leave_{a_2} 2} \leq 1$$

$$if\ j_1 \in T_{North}\ and\ j_2 \in T_{Center}, Type_{Arrive_{a_1}} = D\ and\ Type_{Leave_{a_2}} = D \tag{6-4}$$

…

$k = 196$的方案：

$$x_{a_1 j} + x_{a_2 j} - y_{Arrive_{a_1} Leave_{a_2} 196} \leq 1$$

$$if\ j_1 \in \mathrm{S}_{East}\ \text{and}\ j_2 \in \mathrm{S}_{East}, Type_{Arrive_{a_1}} = I\ and\ Type_{Leave_{a_2}} = I \tag{6-198}$$

相比较模型二，模型三还需要考虑登机口在航站楼所处的位置以及捷运。

约束 9：a_1飞机降落航班的时间加上对应的中转旅客换乘时间小于a_2飞机起飞航班的时间，才算换乘成功。

$$\forall a_1 \in A, \forall a_2 \in A:$$

$$\sum_{k=0}^{196} y_{Arrive_{a_1} Leave_{a_2} k} * CostAllTime_k + M * CanTransfer_{Arrive_{a_1} Leave_{a_2}} \leq M + Time_{Leave_{a_2}} - Time_{Arrive_{a_1}} \tag{6-199}$$

$$CanTransfer_{Arrive_{a_1} Leave_{a_2}} \geq y_{Arrive_{a_1} Leave_{a_2} 0} \tag{6-200}$$

M是一个充分大的正整数。

约束 10：计算$TravelTime_{Arrive_{a_1} Leave_{a_2}}$

$$\forall a_1 \in A, \forall a_2 \in A:$$

代表换乘成功的情况：

$$TravelTime_{Arrive_{a_1} Leave_{a_2}} - \sum_{k=0}^{196} y_{Arrive_{a_1} Leave_{a_2} k} * CostAllTime_k + CanTransfer_{Arrive_{a_1} Leave_{a_2}} \geq 1 \tag{6-201}$$

代表换乘失败的情况：

$$TravelTime_{Arrive_{a_1} Leave_{a_2}} + 6 * 60 * CanTransfer_{Arrive_{a_1} Leave_{a_2}} \geq 6 * 60 \tag{6-202}$$

如果换乘成功，即$CanTransfer_{Arrive_{a_1} Leave_{a_2}} = 1$，那么$TravelTime_{Arrive_{a_1} Leave_{a_2}} = \sum_{k=0}^{196} y_{Arrive_{a_1} Leave_{a_2} k} * CostAllTime_k$，否则$TravelTime_{Arrive_{a_1} Leave_{a_2}} = 6 * 60$分钟。

- **第二阶段数学模型：**

在求解完第一阶段模型后，我们可以得出其目标函数，并将其定义为$MinTense$，即最少的换乘旅客总体紧张度。在第二阶段数学模型中，我们改变目标函数，将最小化被使用登机口的数量作为目标函数，同时增添约束，保证最后安排到固定登机口的航班数量等于$MaxFlightNum$且最少的换乘旅客总体紧张度为$MinTense$。

(1) 目标函数

在第二阶段求解的过程中，我们需要最小化被使用登机口的数量，因此目标函数可设为如下所示，其中z_j表示是否使用过 j 登机口，1 表示是：

$$\min \sum_{j \in S \cup T} z_j \tag{6-203}$$

(2) 约束条件

保持阶段一的约束 1-10，同时新增：

约束 11：保证中转旅客的总体流程时间是之前阶段一求解出来的目标函数

$$\sum_{a_1 \in A} \sum_{a_2 \in A} Num_{Arrive_{a_1} Leave_{a_2}} * \frac{TravelTime_{Arrive_{a_1} Leave_{a_2}}}{ConnectTime_{Arrive_{a_1} Leave_{a_2}}} = MinTense \tag{6-204}$$

6.2.2 数学模型的求解

本文通过 Python 调用 CPLEX 的方法，用编程语言表达出上述算法并进行求解。求解结果如下。问题三我们建立的数学模型是整数规划模型，与问题二数学模型规模一样，**有 192549 个变量，有 5103490 个约束**，注意到模型三虽然考虑的因素增加了，但是约束的数量并没有变多！

在求解第一阶段数学模型时，我们通过 Python 调用 CPLEX 求得 $\boldsymbol{MaxFlightNum = 512}$，即成功安排了 512 个航班，256 个飞机至固定停机位，换乘旅客总体紧张度$\boldsymbol{MinTense = 531.382}$。阶段二求解出来的结果为，**最少使用的登机口数量为 67 个**。具体结果显示如下：

- 成功分配到登机口的航班数量和比例，按宽、窄体机分别画柱状图。

表 6-1 问题三 CPLEX 求解成功分配到登机口的航班数量和比例

成功分配到登机口的航班数量	256*2=512
成功分配到登机口的航班比例	256/303=84.49%
成功分配到登机口的宽体机航班数量	49*2=98
宽体机航班总数量	49*2=98
宽体机分配成功比例	49/49=100%
成功分配到登机口的窄体机航班数量	207*2=414
窄体机航班总数量	254*2=508
窄体机分配成功比例	207/254=81.50%

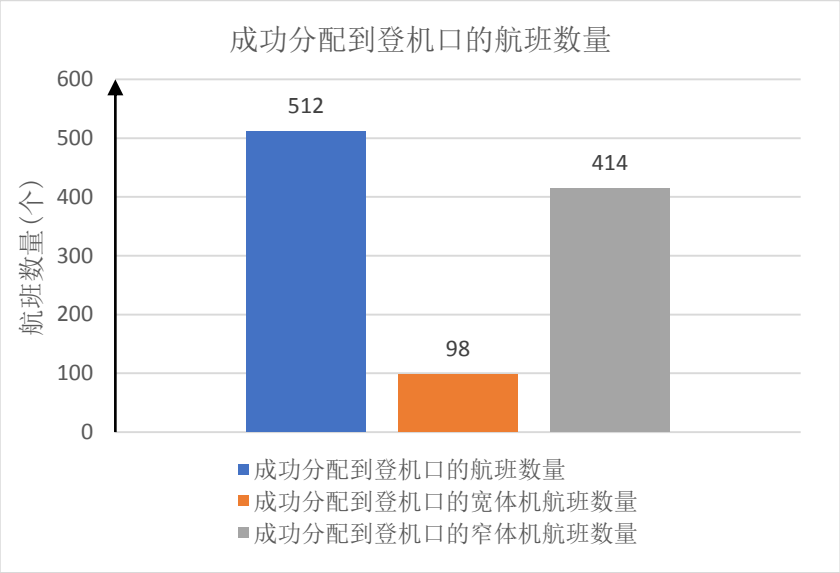

图 6-1 问题三 CPLEX 求解成功分配到登机口的航班数量柱状图

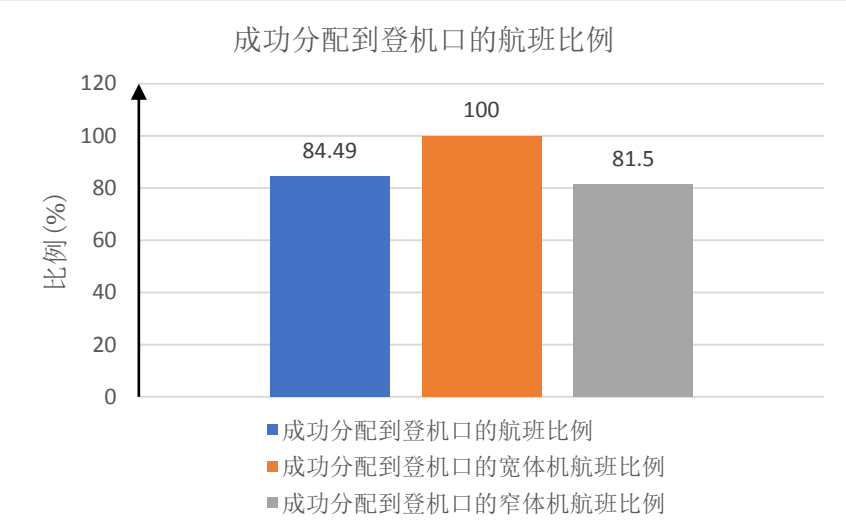

图 6-2 问题三 CPLEX 求解成功分配到登机口的航班比例柱状图

- 给出 T 和 S 登机口的使用数目和被使用登机口的平均使用率，要求画柱状图。

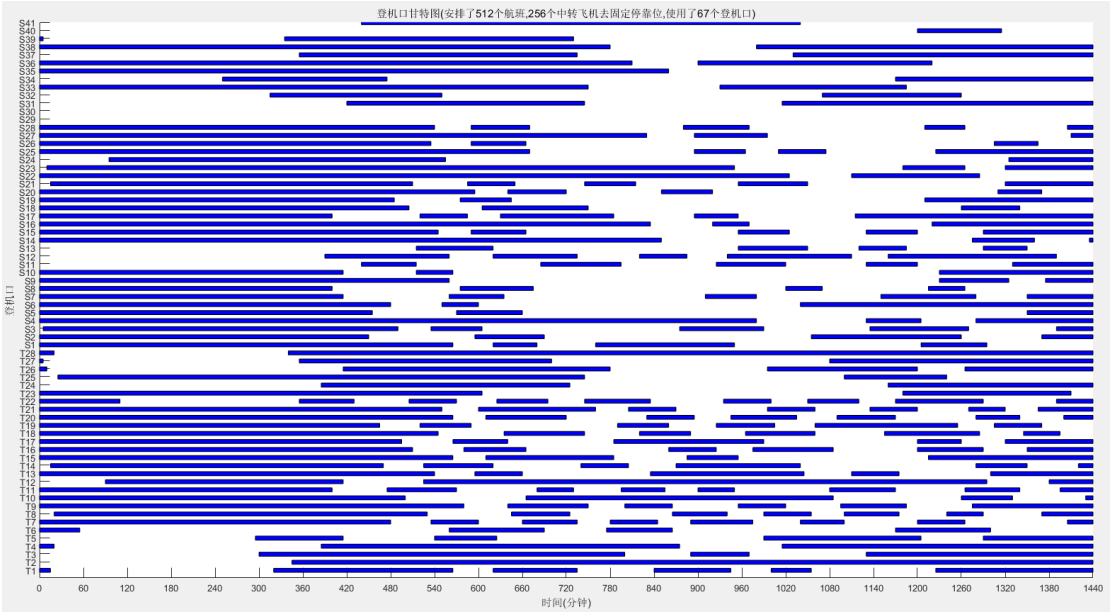

图 6-3 问题三 CPLEX 求解各个登机口甘特图

上图是根据航班-登机口分配方案画出的各个对应登机口的甘特图，根据图片我们发现，在保证成功分配到登机口的航班数量为 512 的情况下，最小换乘旅客总体紧张度为$MinTense = 531.382$的情况下，最少使用的登机口数量是 67，不使用的登机口数量为 2 (S_{29}，S_{30})，具体的数据信息显示如下：

表 6-2 问题三 CPLEX 求解登机口的使用数目和被使用登机口的平均使用率数据

T 航站楼使用的登机口数目	28
T 航站楼使用登机口的平均使用率	62.08%
S 航站楼使用的登机口数目	39
S 航站楼使用登机口的平均使用率	55.15%

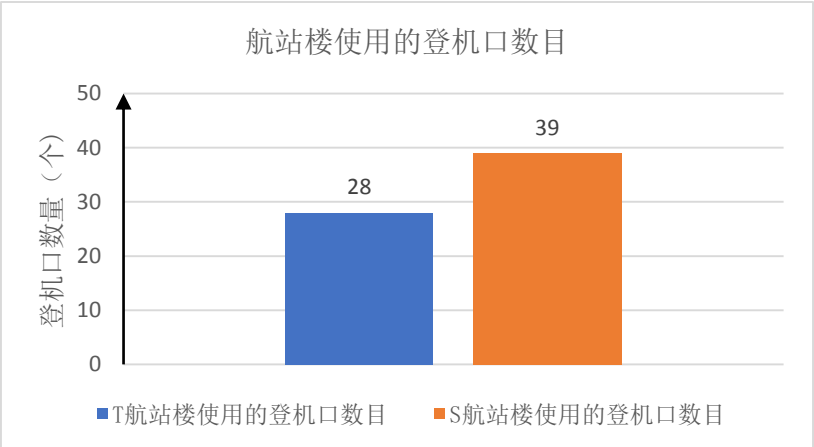

图 6-4 问题三 CPLEX 求解航站楼使用的登机口数目柱状图

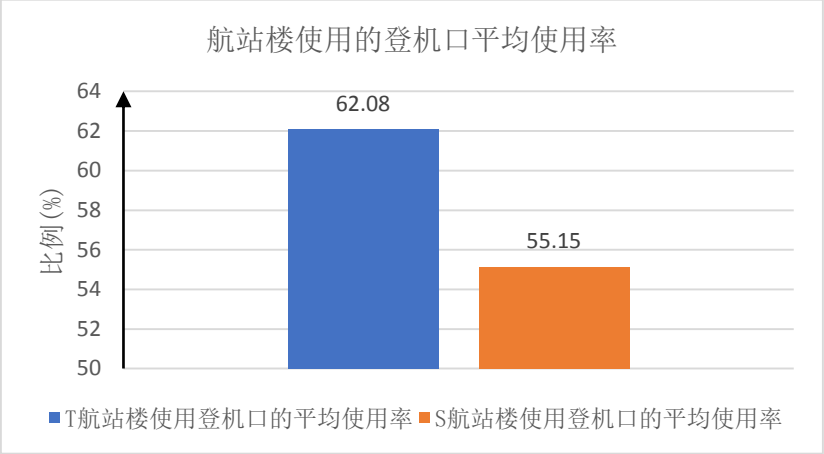

图 6-5 问题三 CPLEX 求解航站楼使用的登机口平均使用率柱状图

- 给出换乘失败旅客数量和比率

表 6-3 问题三 CPLEX 求解换乘失败旅客数量和比率

换乘失败旅客数量	0
换乘失败旅客比例	0

注：因临时停靠位损失的旅客不算做换乘失败

- 总体旅客换乘时间分布图

根据模型求出的航班-登机口分配方案获得：

表 6-4 问题三 CPLEX 求解总体旅客换乘时间分布

换乘时间段	人数	比例	换乘时间段	人数	比例
20-25	31	0.018	55-60	41	0.024
25-30	222	0.130	60-65	177	0.104
30-35	7	0.004	65-70	269	0.158
35-40	72	0.042	70-75	117	0.069
40-45	57	0.033	75-80	29	0.017
45-50	587	0.344	80-85	9	0.005
50-55	88	0.052			

画图得：

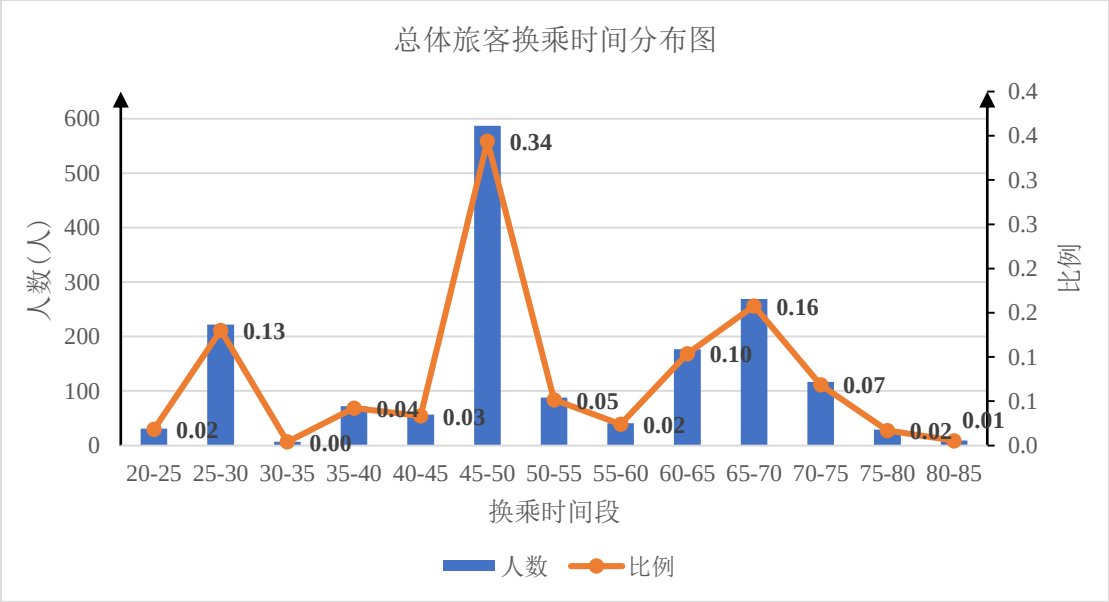

图 6-6 问题三 CPLEX 求解总体旅客换乘时间分布图

- 总体旅客换乘紧张度分布图

根据模型求出的航班-登机口分配方案获得：

表 6-5 问题三 CPLEX 求解总体旅客换乘紧张度分布

换乘紧张度段	人数	比例
0.1-0.2	266	0.15592
0.2-0.3	654	0.383353
0.3-0.4	480	0.28136
0.4-0.5	186	0.109027
0.5-0.6	72	0.042204
0.6-0.7	30	0.017585
0.7-0.8	18	0.010551

画图得：

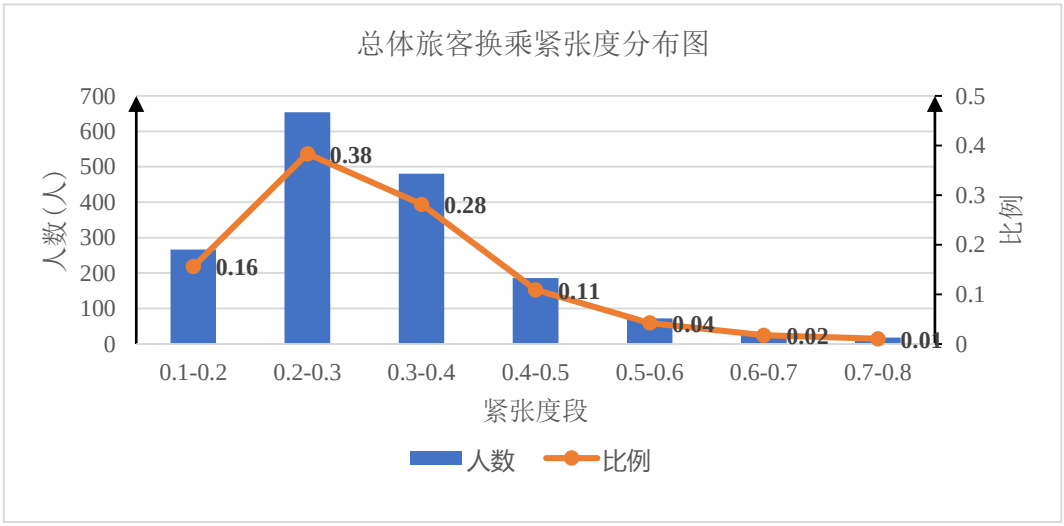

图 6-7 问题三 CPLEX 求解总体旅客换乘紧张度分布

6.3 模拟退火算法的设计及求解

6.3.1 模拟退火算法的设计

问题三与问题二的区别在于对于时间的求解，问题二仅仅需要求解流程时间，而问题三还需要考虑换成捷运的时间以及步行时间。因此，在算法三中我们改变求解时间的方式，依旧采用模拟退火算法，首先根据算法一产生一个调度方案，并将登机口按照属性分类；然后随机选取两个属性相同的登机口，并交换两个登机口的调度方案；如果交换后的调度方案的紧张度减小，则接收新方案；否则，以模拟退火的概率接收该新方案。具体实现细节与算法二一致，在此不予赘述。

在问题三中，相对于求解规划模型，模拟退火算法在运行时间和所需要的计算资源方面的优势更加明显，原因在于模型三的约束条件的个数大于模型二的约束条件的个数。

6.3.2 模拟退火算法的求解

本文通过设计模拟退火算法，最终求得$\boldsymbol{MaxFlightNum = 510}$，即成功安排了 510 个航班，255 个飞机至固定停机位，换乘旅客总体紧张度$\boldsymbol{MinTense = 607.35}$。阶段二求解出来的结果为，**最少使用的登机口数量为 66 个**。具体结果显示如下：

- 成功分配到登机口的航班数量和比例，按宽、窄体机分别画柱状图。

表 6-6 问题三模拟退火求解成功分配到登机口的航班数量和比例

成功分配到登机口的航班数量	255*2=510
成功分配到登机口的航班比例	255/303=84.16%
成功分配到登机口的宽体机航班数量	49*2=98
宽体机航班总数量	49*2=98
宽体机分配成功比例	49/49=100%
成功分配到登机口的窄体机航班数量	206*2=412
窄体机航班总数量	254*2=508
窄体机分配成功比例	206/254=81.10%

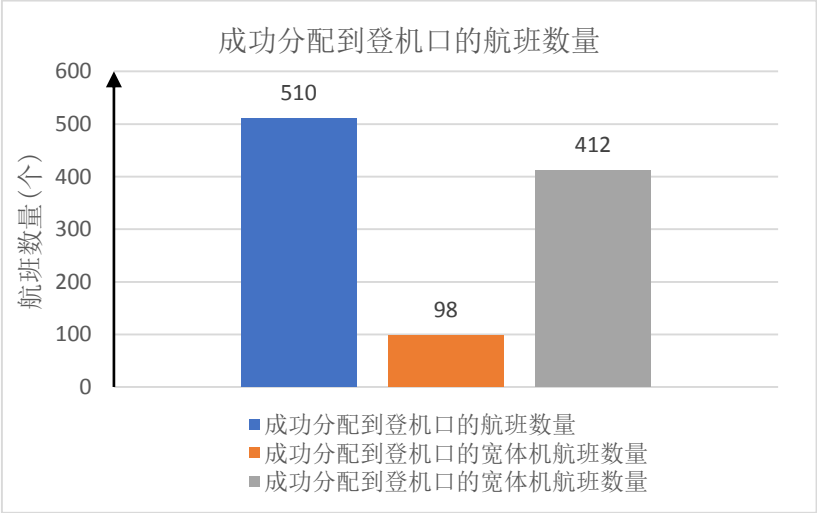

图 6-8 问题三模拟退火求解成功分配到登机口的航班数量柱状图

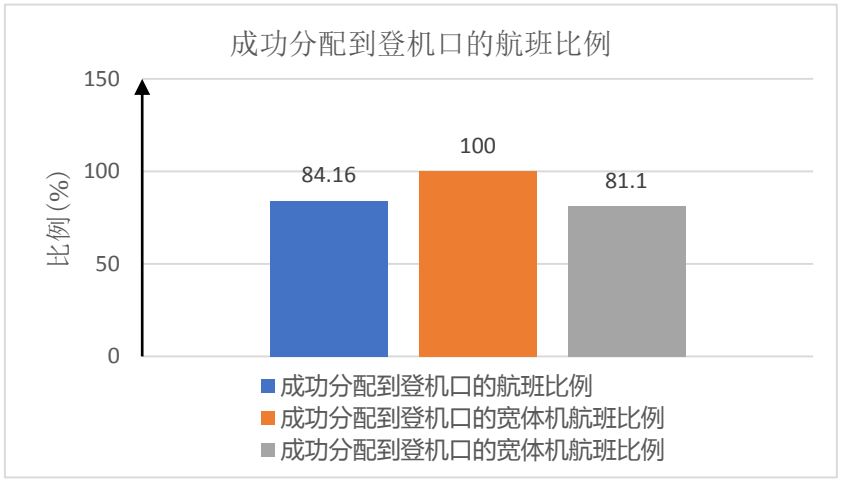

图 6-9 问题三模拟退火求解成功分配到登机口的航班比例柱状图

- 给出 T 和 S 登机口的使用数目和被使用登机口的平均使用率，要求画柱状图。

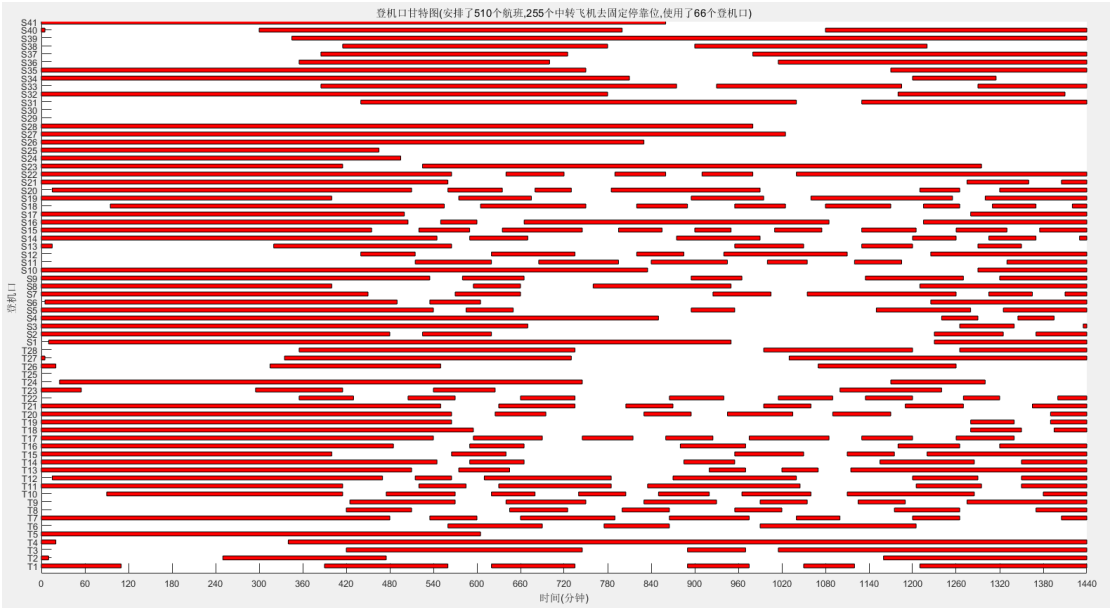

图 6-10 问题三模拟退火求解各个登机口甘特图

上图是根据航班-登机口分配方案画出的各个对应登机口的甘特图，根据图片我们发现，在保证成功分配到登机口的航班数量为 510 的情况下，$MinTense = 607.35$的情况下，最少使用的登机口数量是 66，不使用的登机口数量为 3 (T_{25}，S_{29}，S_{30})，具体的数据信息显示如下：

表格 6-7 问题三模拟退火求解登机口的使用数目和被使用登机口的平均使用率数据

T 航站楼使用的登机口数目	27
T 航站楼使用登机口的平均使用率	54.08%
S 航站楼使用的登机口数目	39
S 航站楼使用登机口的平均使用率	60.56%

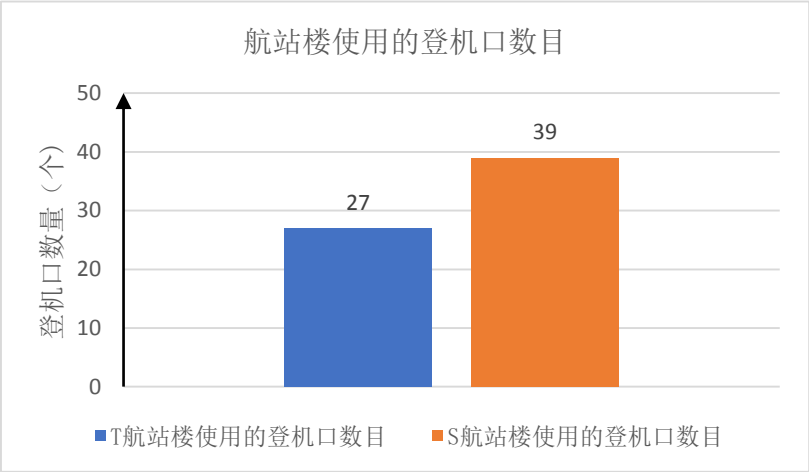

图 6-11 问题三模拟退火求解航站楼使用的登机口数目柱状图

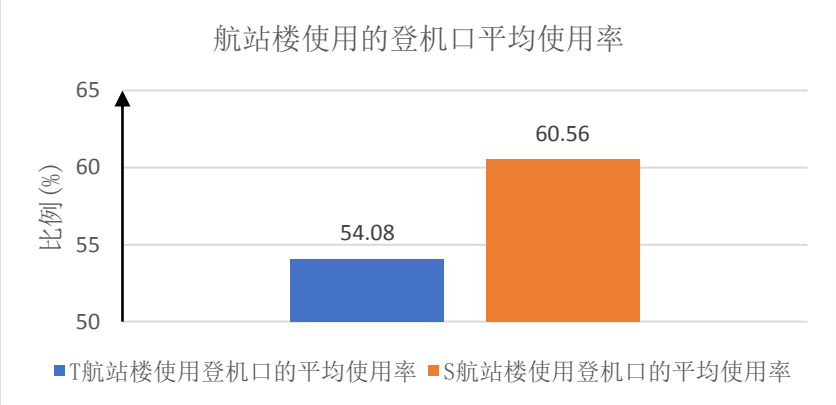

图 6-12 问题三模拟退火求解航站楼使用的登机口平均使用率柱状图

- 给出换乘失败旅客数量和比率

表 6-8 问题三模拟退火求解换乘失败旅客数量和比率

换乘失败旅客数量	0
换乘失败旅客比例	0

注：因临时停靠位损失的旅客不算做换乘失败

- 总体旅客换乘时间分布图

根据模型求出的航班-登机口分配方案获得：

表 6-9 问题三模拟退火求解总体旅客换乘时间分布

换乘时间段	人数	比例	换乘时间段	人数	比例
20-25	16	0.008	55-60	40	0.021
25-30	239	0.126	60-65	110	0.058
30-35	138	0.073	65-70	342	0.180
35-40	43	0.023	70-75	189	0.100
40-45	132	0.070	75-80	50	0.026
45-50	466	0.246	80-85	25	0.013
50-55	108	0.057			

画图得：

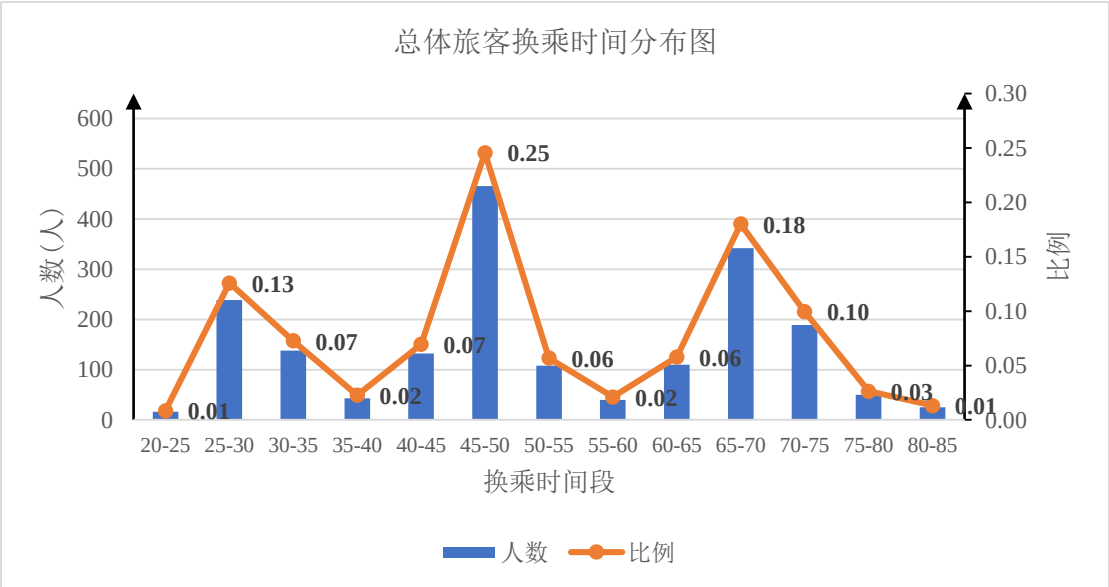

图 6-13 问题三模拟退火求解总体旅客换乘时间分布图

● 总体旅客换乘紧张度分布图

根据模型求出的航班-登机口分配方案获得：

表 6-10 问题三模拟退火求解总体旅客换乘紧张度分布

换乘紧张度段	人数	比例
0.1-0.2	379	0.200
0.2-0.3	566	0.298
0.3-0.4	514	0.271
0.4-0.5	234	0.123
0.5-0.6	141	0.074
0.6-0.7	49	0.026
0.7-0.8	11	0.006
0.8-0.9	4	0.002

画图得：

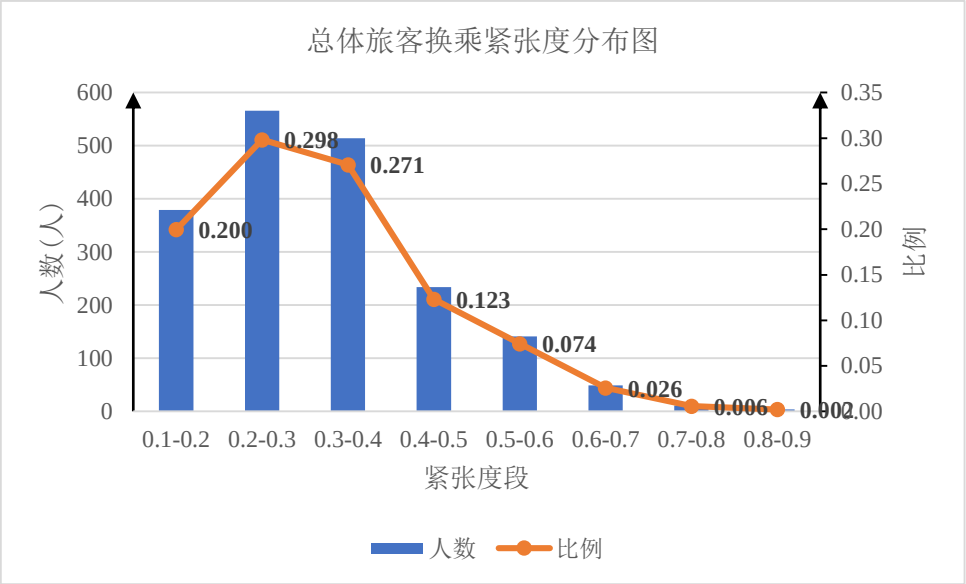

图 6-14 问题三模拟退火求解总体旅客换乘紧张度分布

6.4 数据分析及结论

根据 CPLEX 求解器求解出数学模型的结果以及模拟退火算法求解出的结果，我们获得如下数据结果：

表 6-11 问题三 CPLEX 求解器求解结果与模拟退火算法求解结果分析

	CPLEX 求解结果	模拟退火算法求解结果
成功分配到登机口的航班数量	**512**	510
成功分配到登机口的航班比例	84.49%	84.16%
成功分配到登机口的宽体机航班数量	98	98
宽体机航班总数量	98	98
宽体机分配成功比例	100%	100%
成功分配到登机口的窄体机航班数量	414	412
窄体机航班总数量	508	508
窄体机分配成功比例	81.50%	81.10%
最少的中转旅客的换乘紧张度	**531.382**	607.35
T 航站楼使用的登机口数目	28	27
T 航站楼使用登机口的平均使用率	62.08%	54.08%
S 航站楼使用的登机口数目	39	39
S 航站楼使用登机口的平均使用率	55.15%	60.56%
总共使用的登机口数量	67	**66**
求解时间	>48 小时	**<5 分钟**

根据上述表格我们可以发现，下述结论：

- CPLEX 求解器求解出来的航班-登机口分配方案优于模拟退火算法求解出的航班-登机口分配方案，在第一优化目标和第二优化目标上都优于，第三优化

目标劣于模拟退火算法的求解，但是这两者的 gap 都在 12%，并且第一优化目标相差得 gap 在 1%以内，证明模拟退火算法求解该类航班调度问题的有效性。

- 模拟退火算法求解的时间远小于 CPLEX 求解结果。类似，在问题三的约束下，CPLEX 求解器在有限的时间内已经求解不出最优解了，因此我们建议先采用当问题规模较大时，采用模拟退火算法来进行求解。

最终选择用 CPLEX 求解器求解出的航班-登机口方案作为最终航班-登机口方案，具体调度方案见附件。

7 总结分析

第一问我们建立了整数规划模型，解决了基于最大化成功分配航班到合适的登机口的第一优化目标下，并且在此基础上最小化被使用登机口的数量。为了提高求解效率，在利用 CPLEX 求解器的基础上又设计了贪婪算法进行求解。在求解第二问的时候，我们对第一问的模型进行扩展，引入新的决策变量先保证最大化成功分配航班到合适的登机口，在保证最小化旅客的流程时间的情况下最小化被使用登机口的数量。由于数学模型决策变量较多，用 CPLEX 求解器求解的时候需要很多的时间(>48 小时)和内存，我们又基于问题二设计了模拟退火算法，能够用特别少的时间(<5 分钟)就求出来一个较优的分配方案。在求解问题三的时候，我们对数据进行了预处理，因此通过改变输入就可以复用问题二的算法和模型，来先保证最大化成功分配航班到合适的登机口，在保证最小化旅客的换乘紧张度的情况下最小化被使用登机口的数量。

- **问题一到三结论如下：**

问题一：成功安排了 512 个航班，最少使用 66 个登机口；

问题二：成功安排了 512 个航班，最少旅客中转时间是 55490 分钟，最少使用 66 个登机口；

问题三：成功安排了 512 个航班，最少旅客换乘紧张度是 531.382，最少使用 67 个登机口。

- **问题一到三总结如下：**

CPLEX 求解数学模型的优点在于易于理解，描述准确简洁，针对性强，与实际联系紧密，具有很好的通用性和推广性，利用 CPLEX 求解器进行求解时，只要给足够多的时间和计算资源，就能保证计算出全局最优解。但是，其缺点在于引入的变量过多，尤其是很多的 0-1 整数变量，在用 CPLEX 求解的时候需要较长的求解时间和内存资源，当机场需要快速决策时，CPLEX 求解算法将会疲于招架。

模拟退火启发式算法的有点在于快速求取局部最优解，算法结构简单，但是缺点在于，它是启发式算法，不能保证求取出来的结果就是全局最优解，而且该算法受参数(初始温度和退火速度)影响，同时具有随机性。

因此我们**建议**，当问题规模进一步扩大，无法用 CPLEX 求解器及时进行问题的求解时，可以使用模拟退火算法获得一个局部最优解，并将该局部最优解作为 CPLEX 求解器的初始输入，可以大大节约 CPLEX 的求解时间。

- **关于机场建设的指导性意见**

宽体机的分配比例是远大于窄体机的分配比例的。这是由于窄机型航班占整个航班比例的 83%，但是能够容纳窄机型的登机口只占总登机口的 65%，因此在之后的登机口建设中，可以**多建设能够容纳窄机型的登机口**，或者将**一部分属性为宽机型的登机口重建为窄机型**以提高窄体机分配成功比例。

问题二和问题三的旅客换乘失败率都是 0，对于旅客数据的处理我们可以发现，任意需要换乘的旅客，其在换乘的时候，换乘航班的链接时间都是大于他在机场中转所需要花费的时间，这也是符合现实认知的。

在求解多目标问题时，**机场建设方一定要明确首要优化目标和次要优化目标**。当改变优化的目标函数顺序后，解也会发生变化。例如问题一和问题三，问题一的第二优化目标是最小化登机口的使用数量，而问题三的第二优化目标是最小化

旅客紧张度，第三优化目标才是最小化登机口的使用数量，此时它们的航班-登机口调度方案就会出现较大的区别。

基于数学模型算法和摸拟退火算法相关的分析，建议机场建设方如果需要快速决策航班-分配方案的时候，建议采用模拟退火算法；当有足够多的决策时间和计算资源时，建议采用 CPLEX 求解算法来求解。

参考文献

[1] 中国民用航空局, "2017 年民航行业发展统计公报", < http://www.caac.gov.cn/XXGK/XXGK/TJSJ/201805/t20180521_188131.html >, 2018 05

[2] Hemchand Kochukuttan and Sergey Shebalov, New Gate Planning Optimizer Perfects Gate Assignment Process, Ascend Magazine, https://www.sabreairlinesolutions.com/pdfs/Plan_Ahead.pdf

[3] Abdelghani Bouras, Mageed A. Ghaleb, Umar S. Suryahatmaja, and Ahmed M. Salem, The Airport Gate Assignment Problem: A Survey, The Scientific World Journal, Volume 2014, http://dx.doi.org/10.1155/2014/923859

[4] Dong Zhang, Diego Klabjan, Optimization for gate re-assignment, Transportation Research Part B: Methodological, Volume 95, January 2017

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

[6] Wikipedia contributors, 'Interval scheduling', Wikipedia, The Free Encyclopedia, 26 July 2018, 11:04 UTC, <https://en.wikipedia.org/w/index.php?title=Interval_scheduling&oldid=852061237> [accessed 18 September 2018]

[7] 史峰, 王辉, 郁磊, 等. MATLAB 智能算法 30 个案例分析[J]. 2011.

附录 1：调度方案

下表展示了问题一、二、三最终的调度方案。其中：

问题一：成功安排了 512 个航班，最少使用 66 个登机口；

问题二：成功安排了 512 个航班，最少旅客中转时间是 55490 分钟，最少使用 66 个登机口；

问题三：成功安排了 512 个航班，最少旅客换乘紧张度是 531.382，最少使用 67 个登机口。

飞机转场记录	问题一登机口	问题二登机口	问题三登机口	飞机转场记录	问题一登机口	问题二登机口	问题三登机口
PK062	S26	S23	S4	PK351	T9	T9	T9
PK072	S24	S21	S22	PK352	T23	T24	T25
PK089	S32	T4	S36	PK353	临时	T8	T8
PK094	T26	S33	S38	PK354	临时	临时	临时
PK102	S18	S1	S27	PK355	S14	S5	T13
PK104	T24	S34	T28	PK356	S10	T10	S22
PK106	T6	T24	T6	PK357	S8	S3	S17
PK107	T2	S37	T27	PK358	S12	T1	S13
PK108	T4	S32	T4	PK359	T8	临时	临时
PK112	临时	临时	临时	PK360	T12	T12	S4
PK117	T17	T12	T19	PK361	S11	S12	S11
PK129	T23	T5	S39	PK362	S2	S8	S15
PK131	T28	T2	T26	PK363	T28	S32	T3
PK136	S14	S28	T17	PK364	S4	S19	S3
PK142	临时	临时	临时	PK365	T20	T20	T21
PK144	T1	S12	T22	PK366	临时	临时	临时
PK145	S16	T16	S16	PK367	S28	S27	S7
PK147	S25	T11	T10	PK368	临时	临时	临时
PK148	临时	临时	临时	PK369	S17	S21	T18
PK149	T22	S11	T1	PK370	S41	S36	T24
PK150	T10	T19	T11	PK371	S13	S11	S12
PK151	T5	T23	T23	PK372	T7	T7	T22
PK155	S2	S25	T13	PK373	S37	T23	S34
PK156	T7	T7	T7	PK374	T6	T5	T6
PK159	S41	S38	S33	PK375	临时	临时	临时
PK165	临时	临时	临时	PK376	T14	T16	S23
PK168	T12	S2	T15	PK377	S36	S31	T23
PK170	S5	T13	S19	PK378	T22	临时	临时
PK171	T15	T10	T16	PK379	临时	T22	T7
PK173	S15	S24	S8	PK380	S3	S22	T17
PK174	T13	S20	S20	PK381	S16	T14	T16
PK175	S1	S14	S2	PK382	S40	T26	S40

PK177	临时	临时	临时	PK383	S7	T17	S1
PK178	临时	T8	T9	PK384	临时	临时	临时
PK179	S4	S15	S17	PK385	S6	S25	S28
PK180	S21	S5	S10	PK386	T1	S13	临时
PK181	S19	S4	S26	PK387	T15	T15	S19
PK182	S10	S17	S6	PK388	T11	S18	S8
PK183	T20	T20	T21	PK389	T19	S4	T15
PK184	S23	S22	S15	PK390	S23	S24	S16
PK185	临时	临时	临时	PK391	临时	临时	T1
PK186	临时	临时	临时	PK392	S15	S14	S25
PK187	S3	S26	S18	PK393	临时	临时	临时
PK188	S8	S27	S7	PK394	T17	S15	S10
PK189	T21	T21	T20	PK395	T21	S10	S9
PK190	临时	临时	临时	PK396	T9	T8	T8
PK191	S22	S16	S28	PK397	临时	T21	临时
PK192	T18	S8	S5	PK398	S20	S20	T10
PK193	S27	S18	S9	PK399	S21	S23	S18
PK194	S20	S7	S25	PK400	T24	S38	T26
PK195	S28	T18	T18	PK401	T12	T18	T11
PK196	T19	S3	S14	PK402	T20	T20	T21
PK197	S7	S10	S1	PK403	S24	S8	S14
PK208	T27	T3	S35	PK404	T8	T9	T9
PK253	T20	T20	T21	PK405	临时	临时	临时
PK254	T15	S9	S18	PK406	S5	S6	T20
PK255	临时	T21	T20	PK407	S14	S1	T14
PK256	S22	S14	T15	PK408	T10	S16	S4
PK257	T22	临时	临时	PK409	S11	S12	S13
PK258	临时	临时	临时	PK410	S22	S5	S15
PK259	S17	S15	S1	PK411	T23	S33	T5
PK260	S11	T1	T1	PK412	S18	S2	T13
PK261	T1	S13	S12	PK413	T18	T13	S26
PK262	临时	临时	临时	PK414	S25	S26	T19
PK263	T21	临时	T22	PK415	S27	S25	S20
PK264	T10	S16	S17	PK416	T11	S7	S23
PK265	临时	临时	临时	PK417	S9	S19	S21
PK266	S23	T10	T18	PK418	S26	T12	T17
PK267	S8	S5	S20	PK419	S6	T19	S24
PK268	临时	T8	T9	PK420	临时	临时	临时
PK269	T7	T22	T8	PK421	S12	T1	S11
PK270	T9	T7	T7	PK422	S10	S22	T18
PK271	临时	临时	临时	PK423	T14	S28	S7
PK272	S15	T19	T10	PK424	S2	S17	T16
PK273	T18	T12	T11	PK425	S3	S27	S5

PK274	S13	S11	S11	PK426	T7	临时	T21
PK275	临时	临时	临时	PK427	T9	T8	T8
PK276	T17	S25	T14	PK428	S1	T16	S2
PK277	T8	T11	S21	PK429	S4	T10	S9
PK278	T22	T9	T22	PK430	T13	S21	T12
PK279	S14	S20	S1	PK431	T12	T11	S3
PK280	T6	T23	T6	PK432	T20	T22	T22
PK281	T7	T22	T7	PK433	S14	T14	T11
PK282	S2	S8	T17	PK434	T21	T20	T20
PK283	S4	T10	T19	PK435	S17	S10	S28
PK284	S17	T17	T11	PK436	T22	T7	T7
PK285	T9	T7	T9	PK437	T18	S1	S27
PK286	T20	T21	T21	PK438	S25	S26	T14
PK287	T1	S13	S12	PK439	S7	T18	T10
PK288	T12	S15	T18	PK440	S19	S8	S14
PK289	T21	T20	T20	PK441	T14	S19	S23
PK290	临时	临时	临时	PK442	T16	T14	S21
PK291	S28	T18	T13	PK443	S9	T15	T14
PK292	S13	T1	T1	PK444	T8	T9	T8
PK293	S20	S14	S20	PK445	T25	T6	T25
PK294	S21	S7	T16	PK446	T11	S6	S3
PK295	临时	临时	临时	PK447	S6	T17	T12
PK296	T8	T8	T8	PK448	S17	S9	S24
PK297	S10	S5	T14	PK449	T6	T5	S34
PK298	T10	T15	S3	PK450	S11	临时	临时
PK299	T17	S9	S28	PK451	T9	临时	临时
PK300	S8	T12	T15	PK452	T23	T24	T5
PK301	临时	临时	临时	PK453	T4	T2	T3
PK302	临时	临时	临时	PK454	S31	S32	S32
PK303	T7	T22	T7	PK455	S13	T1	T1
PK304	T23	T5	T3	PK456	S40	S31	S39
PK305	S22	S16	S17	PK457	S38	S37	T28
PK306	S16	S1	S25	PK458	S33	S39	T2
PK307	T16	T14	S27	PK459	T22	T22	T22
PK308	T5	S34	S36	PK460	S37	S36	S37
PK309	T11	S18	T11	PK461	T24	S35	T27
PK310	T13	S2	S7	PK462	临时	临时	临时
PK311	临时	临时	临时	PK463	S39	T27	T4
PK312	T20	T13	S16	PK464	T3	T25	T24
PK313	临时	临时	临时	PK465	T1	S13	S12
PK314	S11	S12	S11	PK466	S36	S41	T26
PK315	S3	S22	T19	PK467	临时	临时	临时
PK316	S32	T27	S33	PK468	T2	T28	S31

PK317	T22	T7	T22	PK469	临时	临时	临时
PK318	T1	S13	S12	PK470	S12	S12	S11
PK319	T21	T21	T20	PK471	T28	T26	S41
PK320	T12	T17	S21	PK472	临时	临时	临时
PK321	T9	T9	T9	PK473	S15	S15	T11
PK322	临时	临时	临时	PK474	T22	T22	T22
PK323	S6	S28	S15	PK475	临时	S11	S13
PK324	S12	S11	S13	PK476	T10	S14	S10
PK325	T15	T10	T18	PK477	S8	T19	T19
PK326	S7	S27	T16	PK478	S6	S5	S17
PK327	临时	临时	临时	PK479	S4	T12	T14
PK328	T3	S35	S38	PK480	S1	S17	T12
PK329	T8	T8	T8	PK481	T7	T7	T7
PK330	临时	临时	临时	PK482	S21	T15	S3
PK331	T24	T6	T5	PK483	T23	T5	T5
PK332	临时	T20	T21	PK484	T18	S8	S6
PK333	T4	S33	T26	PK485	临时	临时	临时
PK334	S13	T1	T1	PK486	S9	S27	S7
PK335	T18	S15	S25	PK487	T6	T24	T6
PK336	S31	T4	S31	PK488	S25	T11	T17
PK337	T20	临时	临时	PK489	T17	S28	S5
PK338	T27	S41	T4	PK490	S10	T14	S19
PK339	S4	S24	S8	PK491	S3	T17	S8
PK340	T26	T3	S37	PK492	临时	临时	临时
PK341	T7	T22	T7	PK493	S14	T13	T16
PK342	T16	S9	S6	PK494	S19	S25	S21
PK343	临时	临时	临时	PK495	T8	T9	临时
PK344	T22	T7	T22	PK496	S5	S22	S26
PK345	S27	T11	S2	PK497	S28	S26	S28
PK346	S26	S7	T19	PK498	T16	T18	S15
PK347	S39	T25	S32	PK499	临时	临时	临时
PK348	S5	S14	T11	PK500	T11	S4	S2
PK349	T25	T2	T27	PK501	S2	S24	T13
PK350	T21	T21	T20				

附录 2：部分代码

完整的代码长度超过 2600 行，出于篇幅考虑，附录仅包含数学模型（plane_model.py）和启发式算法（algorithm.py）的描述，其余的代码见附件。

plane_model.py

```
from __future__ import print_function
import cplex
import data
import mip_starts
from sys import argv

if len(argv) == 2:
    PROBLEM = str(argv[1])
else:
    PROBLEM = '1.0'
print('\n *** Solving problem {}... *** \n'.format(PROBLEM))

ANS = {
    '1.0': 47,
    '1.1': 65
}

data = data.Data()

"""
Linear programming model based on planes
"""
problem = cplex.Cplex()

"""
Objective function:
maximize the number of parked planes, which is equal to minimizing
the number of planes assigned to the temporary gate
"""
problem.objective.set_sense(problem.objective.sense.minimize)

"""
Variables:
x_i_j: assign plane i to gate j
"""
names = []
x_num = 0
xt_num = 0
# assigned to a normal gate
for i in range(1, data.const['TotalPlane'] + 1):
    for j in range(1, data.const['TotalGate'] + 1):
        names.append('x_{}_{}'.format(i, j))
        x_num += 1
# assigned to the temporary gate
for i in range(1, data.const['TotalPlane'] + 1):
    names.append('x_{}_0'.format(i))
    xt_num += 1

if PROBLEM == '1.1':
    # z_j: whether to use gate j
    z_num = 0
    for j in range(1, data.const['TotalGate'] + 1):
        names.append('z_{}'.format(j))
        z_num += 1
if PROBLEM == '2.0' or PROBLEM == '3.0':
    # a i1 i2 k: transfer from plane i to plane j using cost k
    a_num = 0
    for (i1, i2) in data.const['TransferNumber']:
        # NOTE: starts from 0
```

```
        for k in range(0, data.const['TotalCost'][PROBLEM] + 1):
            names.append('a_{}_{}_{}'.format(i1, i2, k))
            a_num += 1
    # y_i1_i2: whether successfully transfer from plane i to plane j
    y_num = 0
    for (i1, i2) in data.const['TransferNumber']:
        names.append('y_{}_{}'.format(i1, i2))
        y_num += 1
    # cost_i1_i2: cost to transfer from plane i to plane j
    cost_num = 0
    for (i1, i2) in data.const['TransferNumber']:
        names.append('cost_{}_{}'.format(i1, i2))
        cost_num += 1

"""
Objective function
"""
if PROBLEM == '1.0':
    objective = [0] * x_num + [1] * xt_num
    # bounds
    lower_bounds = [0] * (x_num + xt_num)
    upper_bounds = [1] * (x_num + xt_num)
    # binary variables
    types = ['B'] * (x_num + xt_num)
elif PROBLEM == '1.1':
    objective = [0] * (x_num + xt_num) + [1] * z_num
    # bounds
    lower_bounds = [0] * (x_num + xt_num + z_num)
    upper_bounds = [1] * (x_num + xt_num + z_num)
    # binary variables
    types = ['B'] * (x_num + xt_num + z_num)
elif PROBLEM == '2.0' or PROBLEM == '3.0':
    if PROBLEM == '2.0':
        objective = [0] * (x_num + xt_num + a_num + y_num)
        for (i1, i2) in data.const['TransferNumber']:
            objective.append(data.const['TransferNumber'][(i1, i2)])
    else:
        objective = [0] * (x_num + xt_num + a_num + y_num)
        for (i1, i2) in data.const['TransferNumber']:
            objective.append(1.0 * data.const['TransferNumber'][(i1, i2)] / data.const['ConnectTime'][(i1,
i2)])
    # bounds
    lower_bounds = [0] * (x_num + xt_num + a_num + y_num + cost_num)
    upper_bounds = [1] * (x_num + xt_num + a_num + y_num) + [10 * 24 * 60] * cost_num
    # some are binary
    types = ['B'] * (x_num + xt_num + a_num + y_num) + ['I'] * cost_num
else:
    raise NameError('No such problem: {}'.format(PROBLEM))
# add variables
problem.variables.add(obj = objective, lb = lower_bounds,
    ub = upper_bounds,
    names = names,
    types = types)

"""
Constraints
"""
constraints = []
constraint_names = []
constraint_senses = []
rhs = []

# arrival constraint
for i in range(1, data.const['TotalPlane'] + 1):
    c = [[], []]
    for j in range(1, data.const['TotalGate'] + 1):
        c[0].append('x_{}_{}'.format(i, j))
        c[1].append(1)
    c[0].append('x_{}_0'.format(i))
```

```
        c[1].append(1)
        constraint_names.append('c_{}'.format(len(constraints)))
        constraints.append(c)
        rhs.append(1)
        constraint_senses.append('E')

    def add_zero_x_constraint(i, j):
        c = [['x_{}_{}'.format(i, j)], [1]]
        constraint_names.append('c_{}'.format(len(constraints)))
        constraints.append(c)
        rhs.append(0)
        constraint_senses.append('E')
        return

    # identical constraint
    for i in range(1, data.const['TotalPlane'] + 1):
        plane_info = data.filtered_plane_dict[i]
        for j in range(1, data.const['TotalGate'] + 1):
            gate_info = data.filtered_gate_dict[j]
            # same size, arrive type and leave type
            if plane_info['size'] != gate_info['size']:
                add_zero_x_constraint(i, j)
                continue
            if (gate_info['arrive_type'] != 'DI' and
            plane_info['arrive_type'] != gate_info['arrive_type']):
                add_zero_x_constraint(i, j)
                continue
            if (gate_info['leave_type'] != 'DI' and
            plane_info['leave_type'] != gate_info['leave_type']):
                add_zero_x_constraint(i, j)
                continue

    def can_same_gate(p1, p2):
        at1 = p1['arrive_time_int']
        at2 = p2['arrive_time_int']
        lt1 = p1['leave_time_int']
        lt2 = p2['leave_time_int']
        if at2 - lt1 >= 45: return True
        if at1 - lt2 >= 45: return True
        return False

    # gap constraint
    for j in range(1, data.const['TotalGate'] + 1):
        for i1 in range(1, data.const['TotalPlane'] + 1):
            for i2 in range(i1 + 1, data.const['TotalPlane'] + 1):
                p1 = data.filtered_plane_dict[i1]
                p2 = data.filtered_plane_dict[i2]
                if not can_same_gate(p1, p2):
                    c = [['x_{}_{}'.format(i1, j), 'x_{}_{}'.format(i2, j)], [1, 1]]
                    constraint_names.append('c_{}'.format(len(constraints)))
                    constraints.append(c)
                    rhs.append(1)
                    constraint_senses.append('L')

    if PROBLEM == '1.1':
        M = data.const['TotalPlane'] + 1
        # gate usage constraint
        for j in range(1, data.const['TotalGate'] + 1):
            c = [[], []]
            for i in range(1, data.const['TotalPlane'] + 1):
                c[0].append('x_{}_{}'.format(i, j))
                c[1].append(1)
            c[0].append('z_{}'.format(j))
            c[1].append(-M)
            constraint_names.append('c_{}'.format(len(constraints)))
            constraints.append(c)
            rhs.append(0)
            constraint_senses.append('L')
```

```
    # gate number constraint
    c = [[], []]
    for i in range(1, data.const['TotalPlane'] + 1):
        c[0].append('x_{}_0'.format(i))
        c[1].append(1)
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(ANS['1.0'])
    constraint_senses.append('E')

if PROBLEM == '2.0' or PROBLEM == '3.0':
    # gate number constraint
    c = [[], []]
    for i in range(1, data.const['TotalPlane'] + 1):
        c[0].append('x_{}_0'.format(i))
        c[1].append(1)
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(ANS['1.0'])
    constraint_senses.append('E')

    # constraint between (x_i1_j1, x_i2_j2) and a_i1_i2_k
    for (i1, i2) in data.const['TransferNumber']:
        p1 = data.filtered_plane_dict[i1]
        p2 = data.filtered_plane_dict[i2]
        if i1 == i2:
            for j in range(1, data.const['TotalGate'] + 1):
                g = data.filtered_gate_dict[j]
                if PROBLEM == '2.0':
                    key = (p1['arrive_type'], g['hall'], p1['leave_type'], g['hall'])
                elif PROBLEM == '3.0':
                    key = (p1['arrive_type'], g['hall'], g['area'],
                           p1['leave_type'], g['hall'], g['area'])
                k = data.const['k'][PROBLEM][key]
                c = [['x_{}_{}'.format(i1, j), 'a_{}_{}_{}'.format(i1, i2, k)],
                    [2, -1]]
                constraint_names.append('c_{}'.format(len(constraints)))
                constraints.append(c)
                rhs.append(1)
                constraint_senses.append('L')
            c = [['x_{}_0'.format(i1), 'a_{}_{}_0'.format(i1, i2)], [1, -1]]
            constraint names.append('c {}'.format(len(constraints)))
            constraints.append(c)
            rhs.append(0)
            constraint_senses.append('L')
            continue

        for j1 in range(1, data.const['TotalGate'] + 1):
            g1 = data.filtered_gate_dict[j1]
            for j2 in range(1, data.const['TotalGate'] + 1):
                g2 = data.filtered_gate_dict[j2]
                if PROBLEM == '2.0':
                    key = (p1['arrive_type'], g1['hall'], p2['leave_type'], g2['hall'])
                elif PROBLEM == '3.0':
                    key = (p1['arrive_type'], g1['hall'], g1['area'],
                           p2['leave_type'], g2['hall'], g2['area'])
                k = data.const['k'][PROBLEM][key]
                c = [['x_{}_{}'.format(i1, j1), 'x_{}_{}'.format(i2, j2),
                    'a_{}_{}_{}'.format(i1, i2, k)],
                    [1, 1, -1]]
                constraint_names.append('c_{}'.format(len(constraints)))
                constraints.append(c)
                rhs.append(1)
                constraint_senses.append('L')
        # x_i1_0 <= a_i1_i2_0
        c = [['x_{}_0'.format(i1), 'a_{}_{}_0'.format(i1, i2)], [1, -1]]
        constraint_names.append('c_{}'.format(len(constraints)))
        constraints.append(c)
        rhs.append(0)
```

```
    constraint_senses.append('L')
    c = [['x_{}_0'.format(i2), 'a_{}_{}_0'.format(i1, i2)], [1, -1]]
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(0)
    constraint_senses.append('L')

# force some k that a_i1_i2_k = 1
for (i1, i2) in data.const['TransferNumber']:
    c = [[], []]
    # NOTE: starts from 0
    for k in range(0, data.const['TotalCost'][PROBLEM] + 1):
        c[0].append('a_{}_{}_{}'.format(i1, i2, k))
        c[1].append(1)
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(1)
    constraint_senses.append('E')

# constraint between y_i1_i2 and a_i1_i2_k
M = 10 * 24 * 60
for (i1, i2) in data.const['TransferNumber']:
    p1 = data.filtered_plane_dict[i1]
    p2 = data.filtered_plane_dict[i2]
    arrive_time = p1['arrive_time_int']
    leave_time = p2['leave_time_int']
    diff = leave_time - arrive_time
    c = [[], []]
    for k in range(1, data.const['TotalCost'][PROBLEM] + 1):
        c[0].append('a_{}_{}_{}'.format(i1, i2, k))
        c[1].append(data.const['Cost'][PROBLEM][k])
    c[0].append('y_{}_{}'.format(i1, i2))
    c[1].append(M)
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(M + diff)
    constraint_senses.append('L')
    # y_i1_i2 >= a_i1_i2_0
    c = [['y_{}_{}'.format(i1, i2), 'a_{}_{}_0'.format(i1, i2)], [1, -1]]
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(0)
    constraint_senses.append('G')

# constraint between y_i1_i2 and cost_i1_i2
for (i1, i2) in data.const['TransferNumber']:
    c = [['cost_{}_{}'.format(i1, i2)], [1]]
    for k in range(1, data.const['TotalCost'][PROBLEM] + 1):
        c[0].append('a_{}_{}_{}'.format(i1, i2, k))
        c[1].append(-data.const['Cost'][PROBLEM][k])
    c[0].append('y_{}_{}'.format(i1, i2))
    c[1].append(1)
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(1)
    constraint_senses.append('G')
    c = [['cost_{}_{}'.format(i1, i2), 'y_{}_{}'.format(i1, i2)], [1, 60 * 6]]
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(60 * 6)
    constraint_senses.append('G')

# NOTE: hack
for (i1, i2) in data.const['TransferNumber']:
    c = [['y_{}_{}'.format(i1, i2)], [1]]
    constraint_names.append('c_{}'.format(len(constraints)))
    constraints.append(c)
    rhs.append(1)
    constraint_senses.append('E')
```

```
problem.parameters.timelimit.set(5 * 3600)  # 5 hours
problem.linear_constraints.add(lin_expr = constraints, senses = constraint_senses,
    rhs = rhs,
    names = constraint_names)

if PROBLEM == '2.0':
    problem.MIP_starts.add(mip_starts.assign2,
        problem.MIP_starts.effort_level.solve_MIP)
elif PROBLEM == '3.0':
    problem.MIP_starts.add(mip_starts.assign3,
        problem.MIP_starts.effort_level.solve_MIP)

print('solving...')
problem.solve()

# check answers
obj = {
    '1.0': 0
}
assign = {}

for i in range(1, data.const['TotalPlane'] + 1):
    for j in range(1, data.const['TotalGate'] + 1):
        v = problem.solution.get_values('x_{}_{}'.format(i, j))
        if int(v + 0.5) == 1:
            print('land plane {} at {}'.format(i, j))
            assign[i] = j
    v = problem.solution.get_values('x_{}_0'.format(i))
    if int(v + 0.5) == 1:
        print('land plane {} at the temporary gate'.format(i))
        obj['1.0'] += 1
        assign[i] = 0

if PROBLEM == '1.1':
    obj['1.1'] = 0
    for j in range(1, data.const['TotalGate'] + 1):
        v = problem.solution.get_values('z_{}'.format(j))
        if int(v + 0.5) == 1:
            print('use gate {}'.format(j))
            obj['1.1'] += 1

if PROBLEM == '2.0':
    obj['2.0'] = 0
    for (i1, i2) in data.const['TransferNumber']:
        v = problem.solution.get_values('cost_{}_{}'.format(i1, i2))
        v = int(v + 0.5)
        print('assign 1 person from plane {} to plane {} costs {} min'.format(i1, i2, v))
        obj['2.0'] += v

print('obj = {}'.format(obj))
print('assign = {}'.format(assign))
```

algorithm.py

```
"""
    This file contains Solver class for solving problem 1 to 3.
    There are mainly three handlers for three problems respectfully:
     - solve1():
        solve1 function applies extended interval scheduling algorithm to
        arrange planes to different categories of gates, and further
        assign planes to each gate.
        See 4.3 for more details.
     - solve2_swap():
        Base on the arrangement of solve1 function, solver2_swap function
        adopts Simulate Anneal concept to keep swapping planes of two
        different gates trying to reduce total procedure time.
        See 5.3 for more details.
     - solve3_cooling()
```

```
        solve3_cooling function supplements the evaluation function by
        walking time and metro time and keeps other logic unchanged.
        See 6.3 for more details.
"""
import data
import checker


class Solver:
    def __init__(self):
        self.data = data.Data()
        self.plane_dict = self.data.copy_plane_dict()
        self.plane_list = self.data.copy_plane_list()
        self.gate_cato_list = self.data.copy_gate_cato_list()
        self.gate_dict = self.data.copy_gate_dict()

    def mini_temp(self, plane_dict=None, gate_cato_list=None):
        if plane_dict is None:
            plane_dict = self.plane_dict
        if gate_cato_list is None:
            gate_cato_list = self.gate_cato_list
        temp = []
        idx_list = plane_dict.keys()
        idx_list.sort(key=lambda idx: (plane_dict[idx]['arrive_time_int'],
                                       plane_dict[idx]['leave_time_int']),
                      reverse=True)
        for gate_cato in gate_cato_list:
            gate_cato['plane_queue'] = []
            gate_cato['end_time'] = []
            if len(gate_cato['gate_list']) > 0:
                gate_cato['plane_queue'].append([])
                gate_cato['end_time'].append(1e9)
        for plane_idx in idx_list:
            plane_info = plane_dict[plane_idx]
            tar_cato = None
            tar_queue = None
            tar_endt = None
            for i in range(len(gate_cato_list)):
                gate_cato = gate_cato_list[i]
                cato = gate_cato['cato']
                end_time = gate_cato['end_time']
                # check if plane match the category
                if plane_info['arrive_type'] not in cato[0]:
                    continue
                if plane_info['leave_type'] not in cato[1]:
                    continue
                if plane_info['size'] != cato[2]:
                    continue
                for j in range(len(end_time)):
                    if plane_info['leave_time_int'] <= end_time[j]:
                        if tar_cato is None or tar_endt > end_time[j]:
                            tar_cato = i
                            tar_queue = j
                            tar_endt = end_time[j]
                # gate category is arranged with increasing priority
                # found one and break
                if tar_cato is not None:
                    break
            if tar_cato is None:
                # move to temporary areat
                temp.append(plane_idx)
            else:
                gate_cato_list[tar_cato]['plane_queue'][tar_queue].append(plane_idx)
                gate_cato_list[tar_cato]['end_time'][tar_queue] = plane_info['arrive_time_int'] - 45
                if tar_endt == 1e9:
                    if len(gate_cato_list[tar_cato]['plane_queue']) \
                            < len(gate_cato_list[tar_cato]['gate_list']):
                        # add new queue
                        gate_cato_list[tar_cato]['plane_queue'].append([])
```

```
                gate_cato_list[tar_cato]['end_time'].append(1e9)
    print("Un-arranged: " + str(len(temp)))
    return temp

def solve1(self):
    arrangement = {}
    temp = self.mini_temp()
    for idx in temp:
        arrangement[idx] = 0
    for gate_cato in self.gate_cato_list:
        for i in range(len(gate_cato['plane_queue'])):
            for plane_idx in gate_cato['plane_queue'][i]:
                arrangement[plane_idx] = gate_cato['gate_list'][i]
    return arrangement

def solve2_swap(self, T=100, r=0.99):
    from math import exp
    from random import random

    arrangement = {}
    c = checker.Checker()

    # initial arrangement
    queue2gate = {gate_cato: {} for gate_cato in range(len(self.gate_cato_list))}
    temp = self.mini_temp()
    for idx in temp:
        arrangement[idx] = 0
    for i in range(len(self.gate_cato_list)):
        gate_cato = self.gate_cato_list[i]
        for j in range(len(gate_cato['plane_queue'])):
            queue2gate[i][j] = j
            for plane_idx in gate_cato['plane_queue'][j]:
                arrangement[plane_idx] = gate_cato['gate_list'][j]

    tar_proc_time = c.get_proc_time(arrangement)
    old_proc_time = None
    while old_proc_time is None or old_proc_time > tar_proc_time:
        old_proc_time = tar_proc_time
        for i in range(len(self.gate_cato_list)):
            queue = self.gate_cato_list[i]['plane_queue']
            gate = self.gate_cato_list[i]['gate_list']
            for q1 in range(len(queue)):
                for q2 in range(q1 + 1, len(queue)):
                    g1 = queue2gate[i][q1]
                    g2 = queue2gate[i][q2]
                    new_arrangement = dict(arrangement)
                    # swap g1 and g2
                    for plane_idx in queue[q1]:
                        new_arrangement[plane_idx] = gate[g2]
                    for plane_idx in queue[q2]:
                        new_arrangement[plane_idx] = gate[g1]
                    new_proc_time = c.get_proc_time(new_arrangement)
                    if new_proc_time < tar_proc_time:
                        print('proc time: {} -> {}'.format(tar_proc_time, new_proc_time))
                        tar_proc_time = new_proc_time
                        arrangement = dict(new_arrangement)
                        queue2gate[i][q1] = g2
                        queue2gate[i][q2] = g1
                        break
                    elif new_proc_time > tar_proc_time:
                        dE = tar_proc_time - new_proc_time
                        if exp(dE / T) > random():
                            print('{} cooling: proc time: {} -> {}'.format(exp(dE / T),
                                                                           tar_proc_time,
                                                                           new_proc_time))
                            tar_proc_time = new_proc_time
                            arrangement = dict(new_arrangement)
                            queue2gate[i][q1] = g2
                            queue2gate[i][q2] = g1
```

```
                        break
                    T *= r
                if old_proc_time > tar_proc_time: break
            if old_proc_time > tar_proc_time: break
        return arrangement

    def solve3_cooling(self):
        from math import exp
        from random import random

        arrangement = {}
        c = checker.Checker()

        # initial arrangement
        queue2gate = {gate_cato: {} for gate_cato in range(len(self.gate_cato_list))}
        temp = self.mini_temp()
        for idx in temp:
            arrangement[idx] = 0
        for i in range(len(self.gate_cato_list)):
            gate_cato = self.gate_cato_list[i]
            for j in range(len(gate_cato['plane_queue'])):
                queue2gate[i][j] = j
                for plane_idx in gate_cato['plane_queue'][j]:
                    arrangement[plane_idx] = gate_cato['gate_list'][j]

        tar_tense = c.get_all_time(arrangement)
        old_tense = None
        T = 1
        r = 0.01
        while old_tense is None or old_tense > tar_tense:
            old_tense = tar_tense
            for i in range(len(self.gate_cato_list)):
                queue = self.gate_cato_list[i]['plane_queue']
                gate = self.gate_cato_list[i]['gate_list']
                for q1 in range(len(queue)):
                    for q2 in range(q1 + 1, len(queue)):
                        g1 = queue2gate[i][q1]
                        g2 = queue2gate[i][q2]
                        new_arrangement = dict(arrangement)
                        # swap g1 and g2
                        for plane_idx in queue[q1]:
                            new arrangement[plane idx] = gate[g2]
                        for plane_idx in queue[q2]:
                            new_arrangement[plane_idx] = gate[g1]
                        new_tense = c.get_all_time(new_arrangement)
                        if new_tense < tar_tense:
                            print('tense: {} -> {}'.format(tar_tense, new_tense))
                            tar_tense = new_tense
                            arrangement = dict(new_arrangement)
                            queue2gate[i][q1] = g2
                            queue2gate[i][q2] = g1
                            break
                        elif new_tense > tar_tense:
                            dE = tar_tense - new_tense
                            if T == 0: continue
                            if exp(dE / T) > random():
                                print('{} cooling: tense: {} -> {}'.format(exp(dE / T), tar_tense, new_tense))
                                tar_tense = new_tense
                                arrangement = dict(new_arrangement)
                                queue2gate[i][q1] = g2
                                queue2gate[i][q2] = g1
                                break
                            T *= r
                if old_tense > tar_tense: break
            if old_tense > tar_tense: break
        return arrangement


if __name__ == '__main__':
```

```
c = checker.Checker()

def print_temp(arrangement):
    c.check(arrangement)
    cnt = 0
    for i in arrangement:
        if arrangement[i] == 0:
            cnt += 1
    print(cnt)

solver = Solver()

# p1
arrangement = solver.solve1()
print(arrangement)
print_temp(arrangement)

# p2
arrangement = solver.solve2_swap()
print(arrangement)
print(c.get_proc_time(arrangement))

# p3
arrangement = solver.solve3_cooling()
print(arrangement)
print(c.get_all_time(arrangement))
```