“华为杯”第十五届中国研究生

数学建模竞赛

题 目　机场新增卫星厅对中转旅客影响的研究

摘　　要：

本文以最优化原理为理论基础，对航班-登机口的优化分配问题进行研究，并在此基础上评价终端厅对中转旅客的影响程度。针对三个问题，本模型在考虑航班分配失败率最小、被使用登机口比率最小的同时分别以最小化中转旅客总体最短流程时间和总体紧张度为研究目标，建立多目标多约束的登机口分配优化模型，利用生物地理学算法进行步步寻优，但研究中也发现，在最小化中转旅客总体最短流程时间和总体紧张度的过程中，导致分配到临时登机口的旅客人数较多，尤其是旅客人数较多的航班会倾向于分配到临时机位；针对这个现象，本文采用惩罚机制来限制分配到临时机位的航班数量，对比验证了惩罚机制对于航班分配的影响。

问题一：在不考虑中转旅客换乘问题的基础上，建立以航班分配失败率最小和被使用登机口比率最小为目标的双目标多约束优化模型，结合登机口分配规则、飞机指派规则、机体型号等约束条件，利用生物地理学算法对该模型进行求解。结果表明：成功分配登机口的航班比例为 82.18%，被使用登机口数量为 65 个。

问题二：在问题一的基础上，考虑中转旅客的总体最短流程时间，按照以航班分配失败率最小（目标一）、中转旅客总体最短流程时间最小（目标二）以及被使用登机口比率最小（目标三）为优先级，建立多目标多约束的登机口分配优化模型。利用生物地理学算法通过不断的收敛逐渐得到最优解。结果发现，成功分配登机口的航班比例为 77.56%，中转旅客总体最短流程时间为 50350 分钟，被使用登机口数量为 64 个。由于分配到临时登机口的旅客最短流程时间并不计入总体最短流程时间，我们发现在进行目标函数一和二的优化过程中，承载旅客数量较少的航班更容易分配到固定登机口，而承载旅客数量较多的航班倾向于分配到临时登机口，通过计算以上结果，发现共有 1197 人分配到临时登机口，同样会损害航空公司的利益。所以针对问题二本文提出第二种目标函数，即将分配到临时登机口的旅客加入惩罚时间 6 小时计入总最短流程时间，结果发现虽然被使用登机口数量增加为 67 个，总换乘时间增加至 69480 分钟，但是只有 699 个乘客分配至临时登机口。

问题三：在问题二的基础上，考虑中转旅客的换乘时间，按照以航班分配失败率最小（目标一）、中转旅客总体紧张度最小（目标二）以及被使用登机口比率最小（目标三）为优先级，建立多目标多约束的登机口分配优化模型。利用生物地理学算法通过不断的收敛逐渐得到最优解。同样，当临时登机口旅客得惩罚

时间不计入换乘时间，即不影响旅客总体紧张度的情况下，成功分配登机口的航班比例为 79.21%，旅客总体紧张度为 454.33，被使用登机口数量为 66 个，而 1193 个旅客被分配至临时登机口；当临时登机口旅客的惩罚时间计入换乘时间，成功分配登机口的航班比例增加为 82.84%，旅客总体紧张度增加为 607.26，被使用登机口数量保持 66 个，但是只有 726 个乘客分配至临时登机口。

综上所述，本文提出的多目标多约束的登机口分配优化模型能恰当地描述不同目标以及约束条件下航班-登机口地分配问题，而且采用生物地理学算法可快速求解该模型，二者的组合是一种行之有效的登机口分配方案。

关键词：登机口分配，多目标多约束，航班分配失败率、最短流程时间、总体紧张度、被使用登机口比率，生物地理学算法

一、 问题重述

1.1 研究背景

随着我国航空运输的快速发展，急剧增长的客货运输量对民航运输需求不断增加。现有的航站楼旅客的客流量已经达到饱和状态，为了应对不断增加的客流量对机场带来的巨大压力，某机场正在增设卫星厅；虽然卫星厅可以很大程度上的缓解原有航站楼登机口不足的问题，但是对于部分中转旅客而言，会增加其在转场过程中所需要的换乘时间，进而加剧中转旅客的换乘紧张程度，使得旅客对于航空公司和机场的服务满意度降低。

机场资源的有限性是限制枢纽机场发展的重要因素之一，如何提高中转旅客效率已经成为改善枢纽机场中转运行效率的关键性问题，也是其实现自身枢纽战略目标的关键性技术[1-2]。在进行航班换乘时，航班和登机口的衔接指派上也面临着资源的合理调度问题，需要通过技术手段对登机口进行合理指派从而保证有限的登机口资源实现利用最大化，同时考虑旅客的换乘体验[3-4]。对于单纯的航班—登机口优化分配问题已经被很好的解决，而且有非常成熟的产品满足航空公司和机场地勤服务公司的需求。但是如何在优化分配登机口的同时考虑旅客换乘时间最小，这方面的研究还是相对较少。登机口优化分配有两个关键因素需要考虑：一是创建合适的数学优化模型；二是找到合适的算法快速求解该数学模型。在实际应用过程中，创建更加符合实际需求的数学模型和采用行之有效的求解算法是解决优化分配问题的关键。

1.2 已知信息

（1）已知机场的航站楼 T 和航站楼 S 的布局设计、两航站楼登机口的功能属性、不同航站楼登机口间的换乘时间、登机口分配规则、旅客流程等；

（2）已知 2018 年 1 月 19、20、21 号三天的出发、到达航班的旅客数据、航班信息。

1.3 需要解决的问题

基于已知信息建立数学模型，并在给定的航班数据、旅客信息下，求解以下三个问题：

问题一：航班-登机口分配问题。作为分析新建卫星厅对航班影响问题的第一步，首先要建立登机口优化分配数学模型，不考虑中转旅客的换乘问题，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。

问题二：考虑中转旅客最短流程时间的航班-登机口分配问题。本问题是在问题一的基础上加入旅客换乘因素，考虑中转旅客的到达航班和出发航班的飞机所在的登机口的位置对旅客中转流程时间的影响，要求在登机口分配过程中首先考虑航班分配成功数量最大及中转旅客的总体最短流程时间最短，并且在此基础上进一步最小化被使用登机口的数量，进而得到一个多目标优化模型。本问题不考虑换乘旅客乘坐捷运和步行时间。

问题三：考虑中转旅客的换乘时间的航班-登机口分配问题。在登机口分配过程中，新建卫星厅对航班存在的最大影响是中转旅客的换乘时间可能会增加，进而导致部分中转旅客无法及时换乘后续航班；本问题中将换乘旅客的最短流程时间、捷运时间以及步行时间之和作为旅客的换乘时间，旅客的换乘时间转换为旅客换乘紧张度，在进行航班-登机口分配的过程中，首先优化航班分配成功数量最大及中转旅客的总体换乘紧张度最小，并且在此基础上进一步最小化被使用登机口的数量，进而得到一个多目标优化模型。

二、 模型假设

（1） 假设题目中涉及的航班时间均为国际标准时间。

（2） 假设题目中涉及的旅客均为换乘旅客，不考虑始发旅客和终到旅客；

（3） 假设航班的实际到达时间、实际出发时间与航班计划一致，不受天气状况、管制等其他因素干扰；

（4） 设置临时机位，且临时机位的数量不受限制，保证每架飞机均能指派到一个登机口；

（5） 研究时段内的所有航班信息和旅客中转信息是已知的；

（6） 假设旅客中转过程中的换乘时间与登机口位置有关，是相对固定的，不考虑人为随机因素的影响；

（7） 登机口指派是一个连续过程，本文选择一段时间内的飞机进行指派。

三、 符号说明

表 3-1 符号说明

符号	变量说明
F	飞机集合
f_i	飞机编号

C_{f_i}	飞机型号
$C_{f_i}^a$	航班到达类型
$C_{f_i}^d$	航班出发类型
G	登机口集合
g_k	登机口编号
C_{g_k}	登机口机体类别
$C_{g_k}^a$	登机口到达类型
$C_{g_k}^d$	登机口出发类型
$C_{g_k}^{T/S}$	登机口所属的航站楼
T	同一登机口两飞机之间的空档时间，45min
a_{f_i}	飞机 f_i 的到达时刻
d_{f_i}	飞机 f_i 的出发时刻
x_{f_i,f_j}	判断飞机 f_i 和飞机 f_j 的到达、出发航班号是否有旅客换乘
y_{f_i,g_k}	飞机 f_i 是否分配到登机口 g_k
z_{g_k}	判断登机口 g_k 是否被使用
μ_{f_i,f_j}	飞机 f_i 和飞机 f_j 之间的换乘紧张度
$T1_{g_k,g_m}$	登机口 g_k 和 g_m 之间的最短流程时间
$T2_{g_k,g_m}$	登机口 g_k 和 g_m 之间的捷运时间
$T3_{g_k,g_m}$	登机口 g_k 和 g_m 之间的行走时间
m_{g_k,g_m}	登机口 g_k 和 g_m 之间换乘的旅客数量

四、 问题一模型的建立与求解

4.1 问题描述与分析

问题一要求对 20 日到达或 20 日出发的转场飞机和中转旅客信息进行分析，不考虑中转旅客的换乘问题，尽可能多的将航班分配到合适的登机口，并在此基础上最小化被使用登机口的数量。由于同一飞机的到达航班和出发航班都必须分配在同一个登机口进行，所以航班-登机口分配问题等价于飞机-登机口分配。经过数据提取之后，共有 303 驾飞机需要登机口分配，分配到航站楼 T 和卫星厅 S 共 69 个登机口，在本问题中不需要考虑航站楼 T 和卫星厅 S 之间的换乘问题。我们需要针对登机口分配问题进行重新规划来将尽可能多的飞机分配到合适的登机口，并在此基础上最小化被使用登机口的数量。基于问题一的要求，本文建

立一个双目标多约束的登机口分配模型。

（1）目标函数建立

本文先定义以下符号：

F 为飞机集合，f_i 为其中的一架飞机，i 为飞机的编号，$i=1\cdots m$；

C_{f_i} 为飞机的机体类型，区分飞机为窄体机或宽体机：

$$C_{f_i}=\begin{cases}0 & f_i\text{为窄体机}\\ 1 & f_i\text{为宽体机}\end{cases} \tag{1}$$

$C_{f_i}^{a}$ 为飞机到达航班的类型，区分到达航班为国内航班还是国际航班：

$$C_{f_i}^{a}=\begin{cases}0 & f_i\text{的到达航班为国内航班}\\ 1 & f_i\text{的到达航班为国际航班}\end{cases} \tag{2}$$

$C_{f_i}^{d}$ 为飞机出发航班的类型，区分出发航班为国内航班还是国际航班：

$$C_{f_i}^{d}=\begin{cases}0 & f_i\text{的出发航班为国内航班}\\ 1 & f_i\text{的出发航班为国际航班}\end{cases} \tag{3}$$

G 为登机口集合，g_k 为其中的一个登机口，k 为登机口的编号，$k=1\cdots n$；

C_{g_k} 为登机口机体类型，区分登机口为窄体机或宽体机：

$$C_{g_k}=\begin{cases}0 & g_k\text{为窄体机}\\ 1 & g_k\text{为宽体机}\end{cases} \tag{4}$$

$C_{g_k}^{a}$ 为登机口到达类型，区分国内航班和国际航班：

$$C_{g_k}^{a}=\begin{cases}0 & g_k\text{的到达航班为国内航班}\\ 1 & g_k\text{的到达航班为国际航班}\end{cases} \tag{5}$$

$C_{g_k}^{d}$ 为登机口出发类型，区分国内航班和国际航班：

$$C_{g_k}^{d}=\begin{cases}0 & g_k\text{的出发航班为国内航班}\\ 1 & g_k\text{的出发航班为国际航班}\end{cases} \tag{6}$$

T 为前后两架飞机被分配到同一登机口时，两飞机之间的空档间隔时间必须大于 T，在本文中要求 T 为 45 分钟；

a_{f_i} 为飞机 f_i 的到达时刻，d_{f_i} 为飞机 f_i 的出发时刻；

y_{f_i,g_k} 是决策变量，用于判断飞机 f_i 是否分配到登机口 g_k：

$$y_{f_i,g_k}=\begin{cases}0 & \text{如果飞机}f_i\text{未分配到登机口}g_k\\ 1 & \text{如果飞机}f_i\text{被分配到登机口}g_k\end{cases} \tag{7}$$

z_{g_k} 用于判断登机口 g_k 是否被使用：

$$z_{g_k}=\begin{cases}0 & \text{登机口}g_k\text{未被使用}\\ 1 & \text{登机口}g_k\text{被使用}\end{cases} \tag{8}$$

应用上述的符号，结合问题一要求，将尽可能多的航班分配到登机口，由于每架转场飞机的到达和出发航班必须分配在同一登机口进行，所以目标函数可以转化为尽可能多的飞机分配到登机口，可建立如下的目标函数一：

$$\min Z_1 = \frac{m - \sum_{f_i \in F, g_k \in G} y_{f_i,g_k}}{m} \tag{9}$$

其中 m 为飞机的数量，在问题一中共有 303 架飞机，$m=303$；在模型求解的过程中，由于部分飞机可能无法分配到合适的登机口，所以根据题目要求设置临时登机口，且容量无限大，临时登机口并不包含在登机口集合 G 中，本问题中登机口的数量为 69，$n=69$；目标函数 Z_1 表示飞机分配失败率最小，即航班分配到合适登机口的数量最大，符合问题一的题目要求；

根据问题一要求，尽可能最小化被使用登机口的数量，可建立如下的目标函数二：

$$\min Z_2 = \frac{\sum_{g_k \in G} z_{g_k}}{n} \tag{10}$$

z_{g_k} 为决策变量，用于判断登机口 g_k 是否被使用。本问题的要求是在满足航班与登机口功能属性相符的基础上，尽量将飞机分配到同一登机口，所以本文建立的目标函数是被使用登机口比率最小，符合问题一的题目要求。

（2）约束条件

根据登机口分配规则，建立如下约束条件引入到模型中：

1、 机体型号匹配约束：在进行登机口分配时，需要将机体型号相同的航班 f_i 与登机口 g_k 相匹配，在本文中机体型号分为窄体机和宽体机，故有如下约束

$$(C_{f_i} - C_{g_k}) \times y_{f_i,g_k} = 0, \forall f_i \in F, \forall g_k \in G \tag{11}$$

2、 航班到达类型匹配约束：飞机航班到达类型与登机口到达类型相匹配，在本文中航班到达类型分为国际航班和国内航班，故有如下约束：

$$(C^a_{f_i} - C^a_{g_k}) \times y_{f_i,g_k} = 0, \forall f_i \in F, \forall g_k \in G \tag{12}$$

3、 航班出发类型匹配约束：飞机航班出发类型与登机口出发类型相匹配，在本文中航班出发类型分为国际航班和国内航班，故有如下约束：

$$(C^d_{f_i} - C^d_{g_k}) \times y_{f_i,g_k} = 0, \forall f_i \in F, \forall g_k \in G \tag{13}$$

4、 飞机指派约束：转场飞机的到达和出发两个航班必须分配到同一登机口进行，也就是说一个飞机只能分配到同一登机口，不能同时分配到多个登机口或转移登机口：

$$\sum_{g_k \in G} y_{f_i,g_k} = 1, \ \forall f_i \in F, y_{f_i,g_k} \in \{0,1\} \tag{14}$$

5、 登机口指派约束：$y_{f_i,g_k} \times y_{f_j,g_k} = 0$ 时，表示两个飞机没有指派到同一停机位；当 $y_{f_i,g_k} \times y_{f_j,g_k} = 1$ 时，表示飞机 f_i、f_j 指派到同一登机口上；要使得不等式成立，则 $(d_{f_i} - a_{f_j})$ 与 $(d_{f_j} - a_{f_i})$ 中一正一负，而满足这个条件时，可以确保两个飞机占用登机口的时间段没有重合部分。

$$y_{f_i,g_k} \times y_{f_j,g_k} \times (d_{f_i} - a_{f_j}) \times (d_{f_j} - a_{f_i}) < 0 \tag{15}$$

6、 式（6）用于表示分配到同一登机口上的两个飞机应满足一定的空档间隔时间；当 $y_{f_i,g_k} \times y_{f_j,g_k} = 1$ 时，需满足 $a_{f_j} - d_{f_i} \geq T$，即前后两飞机满足空档间

隔时间T，本文为 40 分钟；当$y_{f_i,g_k} \times y_{f_j,g_k} = 0$时，两个飞机不在同一登机口上，不需要满足空档间隔时间，故引用一个足够大的值M，确保不等式的成立。

$$a_{f_j} - d_{f_i} + (1 - y_{f_i,g_k} \times y_{f_j,g_k})M \geq T,\ i < j \tag{16}$$

7、 本模型中，飞机编号、机位编号都为正整数。

$$i, j, k \in Z^+ \tag{17}$$

综上所述，针对问题一我们可以建立如下的双目标登机口分配优化模型：

$$\min Z_1 = \frac{m - \sum_{f_i \in F, g_k \in G} y_{f_i,g_k}}{m}$$

$$\min Z_2 = \frac{\sum_{g_k \in G} z_{g_k}}{n}$$

s.t.

$$(C_{f_i} - C_{g_k}) \times y_{f_i,g_k} = 0$$

$$(C^a_{f_i} - C^a_{g_k}) \times y_{f_i,g_k} = 0$$

$$(C^d_{f_i} - C^d_{g_k}) \times y_{f_i,g_k} = 0$$

$$\sum_{g_k \in G} y_{f_i,g_k} = 1$$

$$y_{f_i,g_k} \times y_{f_j,g_k} \times (d_{f_i} - a_{f_j}) \times (d_{f_j} - a_{f_i}) < 0$$

$$a_{f_j} - d_{f_i} + (1 - y_{f_i,g_k} \times y_{f_j,g_k})M \geq T,\ i < j$$

$$f_i \in F,\ g_k \in G$$

$$i, j, k \in Z^+$$

4.2 模型的求解

登机口分配是一个非线性动态规划问题，含有多个约束条件[5-7]，因此我们提出基于生物地理学算法（Biogeography-Based Optimization BBO）来求解该问题。

（1）BBO 算法

生物地理学算法是基于生物地理学理论的新型智能优化算法，具有良好的收敛性和稳定性[8-9]。在 BBO 算法提出之前已经存在很多优化算法，例如蚁群算法(ACO)、粒子群算法(PSO)、遗传算法(GA)等，这些算法由于种群协作优化的特性被广泛使用。BBO 算法本身也是一种种群智能优化算法，与 GA 算法相比，其迁入率决定了引入的不同个体的比例，而 GA 的基因重组操作所使用的交叉基因片段来自同一个体，这是 BBO 算法与 GA 算法的不同．BBO 算法和 PSO 算法都可以与其邻居进行信息上的交互，但 BBO 算法由于其独特的生物激励机制，它通过群体中个体间的协作和竞争求解复杂的组合优化问题，能更快速高效地找到问题的最优解。BBO 算法通常用适宜度指数(HSI)来描述一个栖息地的种群丰富度，一个栖息地所包含的种群数量与 HSI 成正相关。

BBO 算法的基本流程如下:

1. 初始化 BBO 算法的参数:最大物种数 $S_{\max}$，最大的迁入率 I，最大迁出率 E，最大突变率 $g_{\max}$;

2. 初始化一组栖息地的适宜度向量（SIV），每个向量都对应着一个给定问题的潜在解;

3. 计算某栖息地的适宜度指数（HSI），判断是否满足停止条件，若满足，停止并输出最优解;否则，进行步骤 4;

4. 进行迁移操作，计算迁入率 λ 和迁出率 μ，修正 SIV，重新计算 HSI;

5. 对栖息地执行突变操作，根据变异算子更新物种，重新计算 HSI;

6. 跳转到步骤 3 进行下一次的迭代。

该算法的流程图如图 4-1 所示：

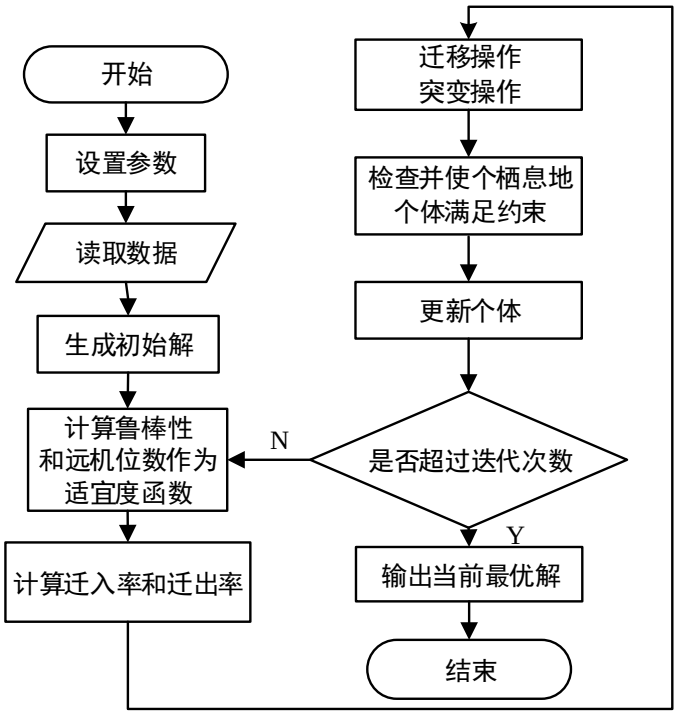

图 4-1 BBO 算法流程图

（2）模型求解过程

该模型涉及到复杂的规划问题，目标函数为双目标多约束优化模型，而根据问题一的要求，两个目标函数的优先顺序为航班分配失败率最小 Z_1，登机口使用比率最小 Z_2。在进行求解的过程中，不断改变两个个目标函数的权重，从而得到一个总体程度最优的解。本模型中增加一个临时登机口，类别不受限制，数量不受限制，为分配失败的飞机提供临时登机口。

1、根据题目中的信息提示我们需要对 InputData.xlsx 进行数据预处理，对 Pucks 附件中的数据提取出 20 日到达或 20 日出发的航班，其中包括 19 日到达 20 日出发的航班信息，同时对 Gates 附件中登机口的字符进行量化标记便于后续处理。

2、建立 aircraft[FA] （FA 为符合条件航班总数）结构体存储航班信息，包括飞机转场记录号（aircraft_rn）、到达航班类型（arrive_type）、出发航班类型（depart_type）、到达航班时刻（arrive_time）、出发航班时刻（depart_time）、飞机型号（aircraft_jx）。建立 gate[DD] （DD 为登机口数量）数组中存储登机口编

号（gate_jp）、登机口到达类型（gate_type）、出发类型（depart_type）、登机口机体类别（gate_jx）。

3、确定不同飞机转场记录号（aircraft_rn）的到达航班类型（arrive_type）、出发航班类型（depart_type）、飞机型号（aircraft_jx）等信息；

4、BBO 算法优化求解

Step1. 将飞机到达航班类型（arrive_type）、出发航班类型（depart_type）、飞机型号（aircraft_jx）等信息与登机口到达类型（gate_type）、出发类型（depart_type）、登机口机体类别（gate_type）一一对应，为每架飞机（aircraft_num）筛选出属性匹配的登机口（gate_num）；

Step2. 随机分配飞机给属性匹配的登机口，确定每个登机口 i 的开放时间（topen[i]），即该登机口前序飞机的出发航班时刻，同时满足 45min 的空档间隔时间；并与未分配登机口的飞机到达航班时刻（arrive_time）相比较，满足要求即可分配给该登机口，否则拒绝分配，搜索其它可分配登机口，若无适合的登机口，则分配到临时登机口；生成初始可行解，完成机位初始分配；

Step3. 对 *Step2* 生成的初始可行解进行有效评估，并带入目标函数 Z_1、Z_2 的计算中，在定义的种群数量范围内进行更新计算。

Step4. 利用 BBO 算法优化种群所有解，在满足约束条件的基础上对登机口多次迭代，经过多次迁移和突变操作，并用上代最优值替换本代最差值，与当前最优解结果比较，判断是否更优。

Step5. 判断是否满足迭代次数，输出当前最优解。

4.3 求解结果与分析

（1）航班分配情况

采用 BBO 算法对模型进行求解，得到航班分配成功数量为 498 个，航班分配成功比例为 82.18%；被使用登机口数量为 65 个，被使用登机口比例为 94.2%。根据航班分配登机口的结果，按宽、窄体机类型分别画出航班分配成功的数量和比例，如图 4-2，4-3 所示。可以看出，宽体机的分配成功率达到 97.96%，窄体机的分配成功率也达到 79.13%。

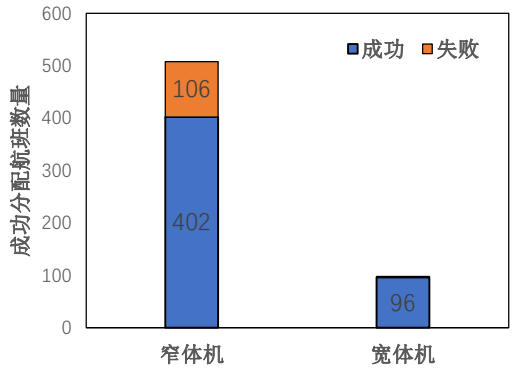

图 4-2 航班分配成功数量

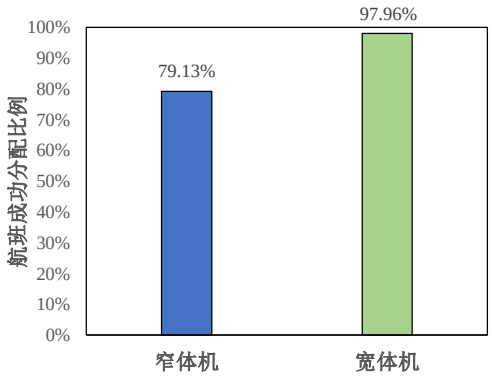

图 4-3 航班分配成功比例

（2）登机口使用情况

T 和 S 终端厅登机口的使用数目和被使用登机口的平均使用率如图 4-4 和 4-5 所示。T 航站楼的被使用登机口数量为 27 个，S 航站楼的被使用登机口数量为 38 个；T 航站楼被使用登机口的平均使用率（64.21%）要略高于 S 航站楼的平均使用率（59.15%）。

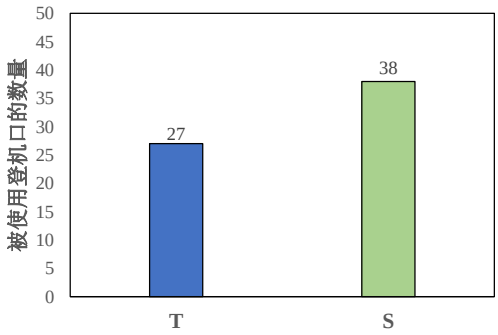

图 4-4 登机口使用数量

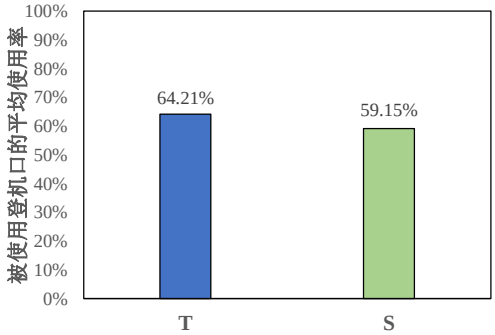

图 4-5 登机口平均使用率

根据问题一的航班分配结果绘制甘特图，直观体现登机口的利用效果，如图 4-6 所示。

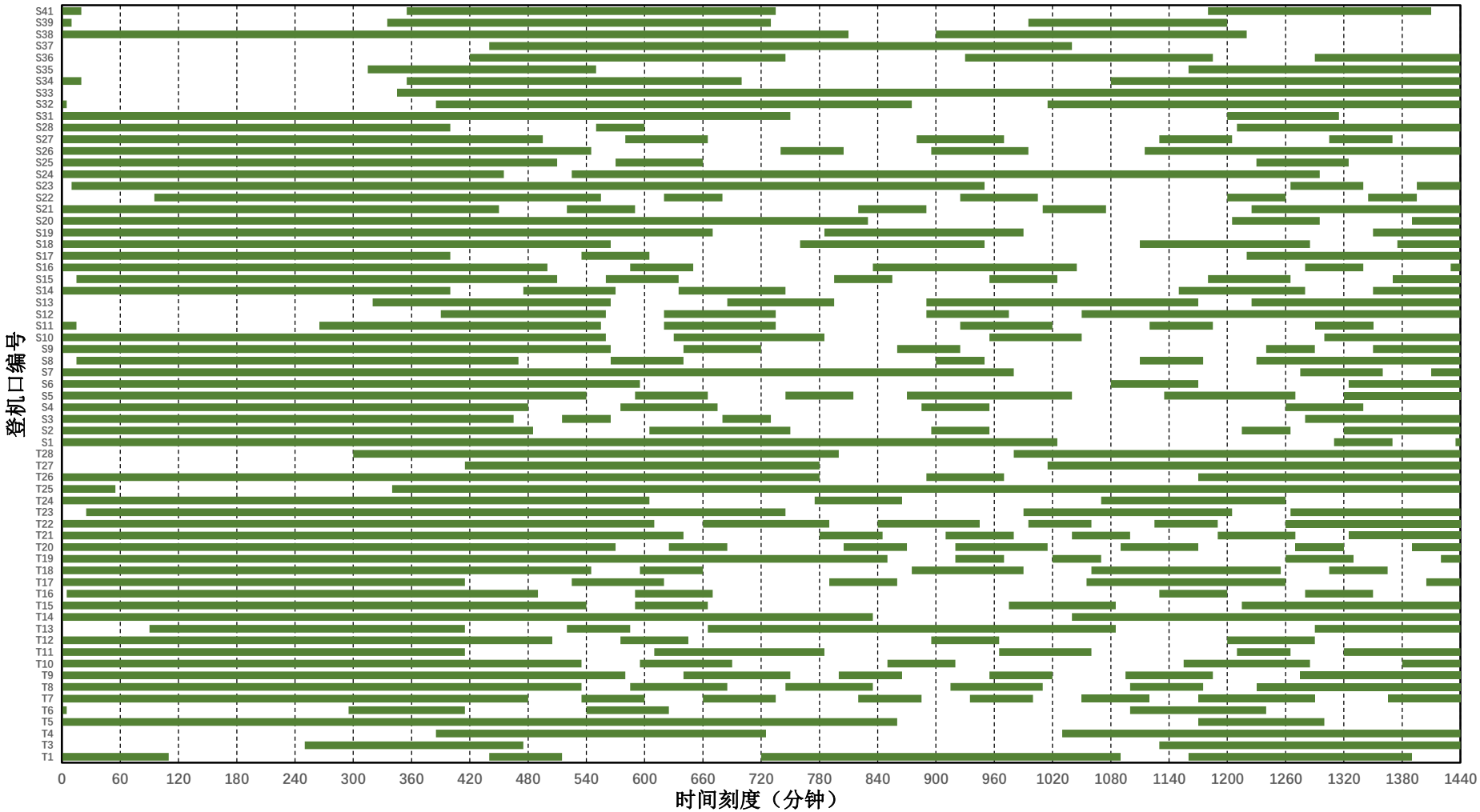

图 4-6 基于问题一航班分配结果的甘特图

五、 问题二模型的建立与求解

5.1 问题描述与分析

考虑中转旅客最短流程时间的航班—登机口分配问题。本问题是在问题一的基础上加入旅客换乘因素：首先要求航班分配成功的数量最大，其次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。其中，中转旅客的总体最短流程时间只包括在不同登机口的转换时间，并不包括捷运时间和旅客的行走时间。

与问题一不同，问题二需要考虑登机口所属航站楼 T 或者卫星厅 S，登机口到达、出发类型等因素都会影响旅客的最短流程时间。所以基于问题二的要求，本文建立多目标多约束登机口指派优化模型。

（1）目标函数的建立

在问题一的符号说明的基础上，本文引入以下符号定义：

$C_{g_k}^{T/S}$ 代表登机口 g_k 所属航站楼：

$$C_{g_k}^{T/S}=\begin{cases}0 & \text{登机口}g_k\text{属于}T\text{航站楼}\\ 1 & \text{登机口}g_k\text{属于}S\text{航站楼}\end{cases} \tag{18}$$

x_{f_i,f_j} 用于决策飞机 f_i 和飞机 f_j 的航班之间是否有中转旅客换乘：

$$x_{f_i,f_j}=\begin{cases}0 & \text{飞机}f_i\text{和}f_j\text{之间无旅客换乘}\\ 1 & \text{飞机}f_i\text{和}f_j\text{之间有旅客换乘}\end{cases} \tag{19}$$

m_{g_k,g_m} 为在登机口 g_k 和登机口 g_m 间换乘的旅客数量；

$T1_{g_k,g_m}$ 为登机口 g_k 和登机口 g_m 间的中转旅客最短流程时间，与登机口 g_k、g_m 的功能属性有关，具体设置如下：

表 5-1 最短流程时间表

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（0）	航站楼 T	15/0	20/1	35/0	40/1
	卫星厅 S	20/1	15/0	40/1	35/0
国际到达（1）	航站楼 T	35/0	40/1	20/0	30/1
	卫星厅 S	40/1	45/2	30/1	20/0

应用上述的符号，结合问题二要求，在分配登机口的时候，保证分配到合适登机口的航班数量最大化，可建立如下的目标函数一：

$$\min Z_1=\frac{m-\sum_{f_i\in F,g_k\in G}y_{f_i,g_k}}{m} \tag{20}$$

目标函数 Z_1 保证航班分配失败率最小，符合问题二的要求；

结合问题二要求，最小化中转旅客的总体最短流程时间，可建立如下的目标函数二：

$$\min Z_2 = \sum_{\substack{f_i \in F \\ f_j \in F \\ i \neq j}} \sum_{\substack{g_k \in G \\ g_m \in G}} y_{f_i,g_k} \times y_{f_j,g_m} \times x_{f_i,f_j} \times T1_{g_k,g_m} \times m_{g_k,g_m} \quad (21)$$

在目标函数Z_2中，首先确定每个飞机分配到了与其属性相符的登机口，进而判断飞机f_i的到达航班与飞机f_j的出发航班是否为中转旅客的到达航班和出发航班及中转的旅客数量；确定到达航班与出发航班分配到的登机口位置，得到中转旅客的最短流程时间。优化目标Z_2使得中转旅客的总体最短流程时间最小。

结合问题二要求，最小化被使用登机口的数量，可建立如下的目标函数三：

$$\min Z_3 = \frac{\sum_{g_k \in G} z_{g_k}}{n} \quad (22)$$

目标函数Z_3表示被使用登机口比率最小，符合问题二的要求。

问题二中的约束条件与问题一中的约束条件一致，所以针对问题二可建立如下的多目标多约束登机口分配优化模型：

$$\min Z_1 = \frac{m - \sum_{f_i \in F, g_k \in G} y_{f_i,g_k}}{m}$$

$$\min Z_2 = \sum_{\substack{f_i \in F \\ f_j \in F \\ i \neq j}} \sum_{\substack{g_k \in G \\ g_m \in G}} y_{f_i,g_k} \times y_{f_j,g_m} \times x_{f_i,f_j} \times T1_{g_k,g_m} \times m_{g_k,g_m}$$

$$\min Z_3 = \frac{\sum_{g_k \in G} z_{g_k}}{n}$$

约束条件：

$$(C_{f_i} - C_{g_k}) \times y_{f_i,g_k} = 0$$

$$(C^a_{f_i} - C^a_{g_k}) \times y_{f_i,g_k} = 0$$

$$(C^d_{f_i} - C^d_{g_k}) \times y_{f_i,g_k} = 0$$

$$\sum_{g_k \in G} y_{f_i,g_k} = 1$$

$$y_{f_i,g_k} \times y_{f_j,g_k} \times (d_{f_i} - a_{f_j}) \times (d_{f_j} - a_{f_i}) < 0$$

$$a_{f_j} - d_{f_i} + (1 - y_{f_i,g_k} \times y_{f_j,g_k}) M \geq T, \quad i < j$$

$$f_i \in F, g_k \in G$$

$$i, j, k \in Z^+$$

5.2 模型的求解

在问题一的基础上，问题二增加了一个优化目标，即旅客的最短流程时间总体最小，属于多目标多约束优化问题，约束条件与问题一相同。根据问题二的说明，首先确定三个目标函数的优先顺序，即分配登机口航班数量最大Z_1、中转旅客流程时间总体最小Z_2以及登机口使用数量最小Z_3。在进行求解的过程中，不断改变三个目标函数的权重，从而得到一个总体程度最优的解。

（1）根据题目中的信息提示我们将转场飞机记录号、旅客记录号进行筛选匹配，对 Pucks 和 Tickets 附件中的数据提取出 20 日到达或 20 日出发的航班和旅客，其中包括 19 日到达 20 日出发的航班和旅票信息，同时对 Gates 附件中登机口的字符进行量化标记便于后续处理。

（2）建立 ticket[NT]（NT 为符合条件的旅客记录号总数）结构体用来存储旅客的机票信息，包括到达航班编号（arrive_num）、出发航班编号（depart_num）、旅客数量（ticket_pn）。建立 aircraft[FA] （FA 为符合条件航班总数）结构体存储航班信息，包括飞机转场记录号（aircraft_rn）、到达航班编号（arrive_num）、出发航班编号（aircraft_depart）、到达航班类型（arrive_type）、出发航班类型（depart_type）、到达航班时刻（arrive_time）、出发航班时刻（depart_time）、飞机型号（aircraft_jx）。建立 gate[DD] （DD 为登机口数量）数组中存储登机口编号（gate_jp）、登机口所属终端厅区域（gate_region）、登机口到达类型（arrive_type）、出发类型（depart_type）、登机口机体类别（gate_jx）等。

（3）分别提取旅客的到达航班编号（arrive_num）、出发航班编号（depart_num），与航班信息中的到达航班编号（arrive_num）、出发航班编号（depart_num）相匹配，进而确定到达航班和出发航班的飞机转场记录号（aircraft_rn）、到达航班类型（arrive_type）、出发航班类型（depart_type）、飞机型号（aircraft_jx）等信息；

（4）BBO 算法优化求解

Step1. 将飞机到达航班类型（arrive_type）、出发航班类型（depart_type）、飞机型号（aircraft_jx）等信息与登机口到达类型（arrive_type）、出发类型（depart_type）、登机口机体类别（gate_jx）一一对应，为每架飞机（aircraft_num）筛选出属性匹配的登机口（gate_num）；

Step2. 随机分配飞机给属性匹配的登机口，确定每个登机口 i 的开放时间（topen[i]），即该登机口前序飞机的出发航班时刻，同时满足 45min 的空档间隔时间；与未分配登机口的飞机的到达航班时刻（arrive_time）相比较，满足要求即可分配给该登机口，否则拒绝分配，搜索其他可分配登机口；从而生成初始可行解；

Step3. 根据旅客的到达航班和出发航班的飞机转场记录号（aircraft_rn），生成初始可行解，完成机位初始分配。确定飞机分配到的登机口编号（gate_num），进而判断登机口所属终端厅（gate_region）；根据 gate_region 计算每位中转旅客的最短流程时间（process_time）并求和；

Step4. 对 *Step3* 生成的初始可行解进行有效评估，并带入目标函数 Z_1、Z_2 和 Z_3 的计算中，在定义的种群数量范围内进行更新计算。

Step5. 利用 BBO 算法优化种群所有解，在满足约束条件的基础上对登机口多次迭代，经过多次迁移和突变操作，并用上代最优值替换本代最差值，与当前最优解结果比较，判断是否更优。

Step6. 判断是否满足迭代次数，输出当前最优解。

需要注意的是，根据模型求解发现，如果将分配到临时机位的旅客换乘时间不计入总换乘时间目标函数中，在目标函数 Z_1 和 Z_2 的优化过程中，承载旅客数量较少的航班会尽量分配至固定登机口，而承载旅客数量较多的航班会尽量分配至临时登机口，导致分配到合适登机口的航班数量较多，但是分配到合适登机口的旅客数量降少；如果将临时机位的旅客换乘时间算作 6 小时作为惩罚时间计入总体最短流程时间的目标函数中，则会降低分配到临时登机口的旅客数量，但是

旅客总体最短流程时间会增加，所以问题二中，我们将临时旅客惩罚时间按照计入和不计入目标函数分别进行模型的求解，具体的求解结果见 5.3。

5.3 求解结果与分析

5.3.1 未考虑临时机位惩罚机制的求解结果

根据问题二所要求的优化目标，该模型针对三个目标进行优化，分别得到航班分配成功的数量为 470 个，总流程时间为 50350 分钟，登机口使用数量为 64 个。该模型将更多的乘客的航班划入了临时登机口以缩短总流程时间，该模型获得了所有模型中最短的总流程时间。但是分配到临时机位的旅客共有 1197 位（共有 2751 人）。

（1）航班分配情况

使用 BBO 算法对所建立模型进行求解，得出航班分配成功数量为 470 个，航班分配成功比例为 77.56%。按宽、窄体机分别画出航班分配成功的数量和比例，如图 5-1 和 5-2 所示。可以看出，窄体机的航班分配成功率达到 74.8%，宽体机的航班分配成功率达到 91.84%。

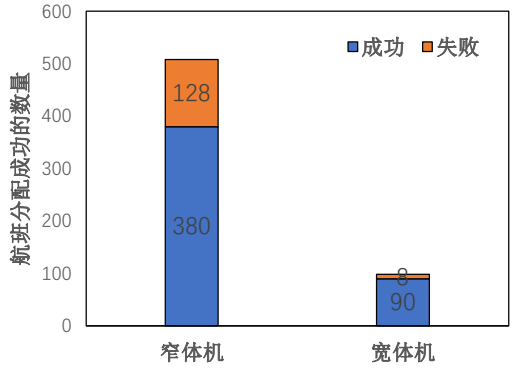

图 5-1 航班分配成功数量

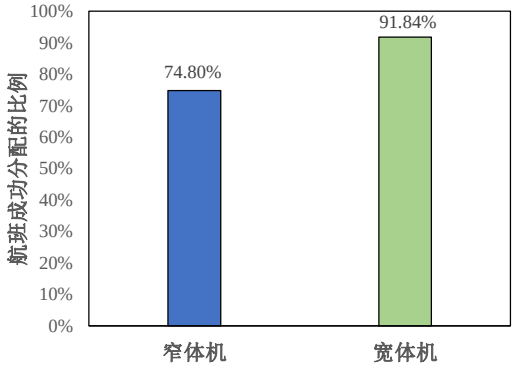

图 5-2 航班分配成功比例

（2）登机口使用情况

被使用登机口数量为 64 个，被使用登机口比例为 92.75%。T 和 S 登机口的使用数目和计算被使用登机口的平均使用率分别如图 5-3 和 5-4 所示。T 航站楼的使用数量为 28 个，S 航站楼的使用数量为 36 个，T 航站楼的平均时间使用率（59.76%）要略高于 S 航站楼的平均时间使用率（56.03%）。

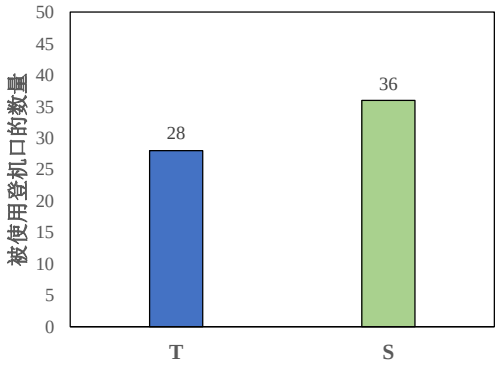

图 5-3 登机口使用数量

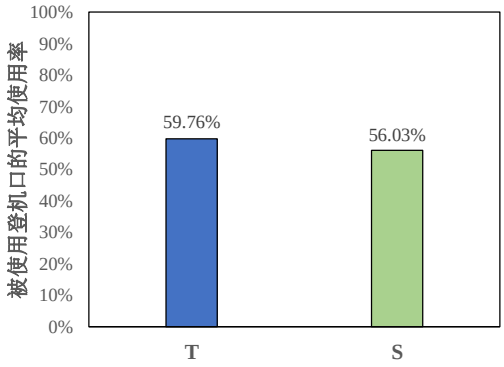

图 5-4 登机口平均使用率

（3）旅客换乘情况

换乘成功意味着航班之间的旅客换乘时间要小于航班之间的时间差。本文根

据航班分配结果，计算得到旅客的换乘时间。假设航班被分配到临时登机口不属于换乘失败的前提下，旅客的换乘失败的人数为 0，比率为 0；但是有 1197 位旅客被分配到临时登机口。旅客换乘时间分布如图 5-5 所示，约 53.54%的乘客能在 1 小时内完成换乘。

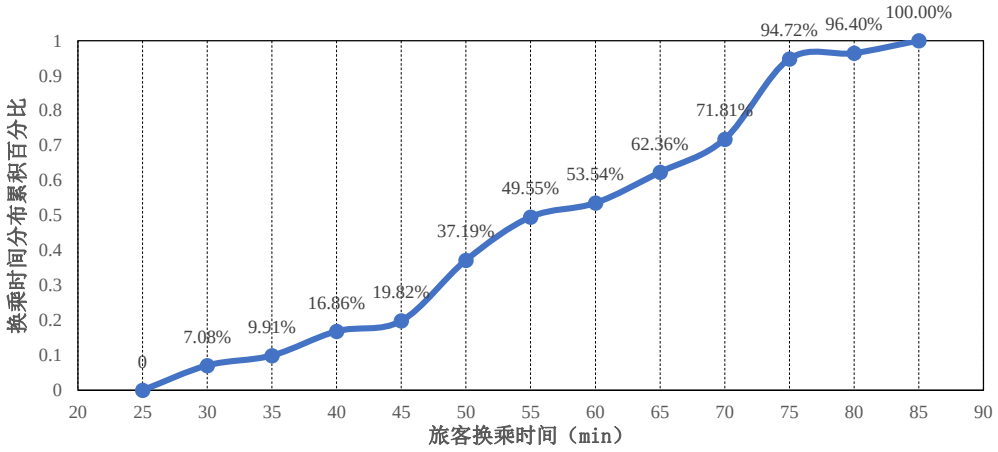

图 5-5 旅客换乘时间分布

（4）旅客紧张度分布情况

旅客紧张度分布如图5-6所示，有87.71%的旅客换乘紧张度都在0.5及以下，说明大部分旅客换乘并不十分紧张。

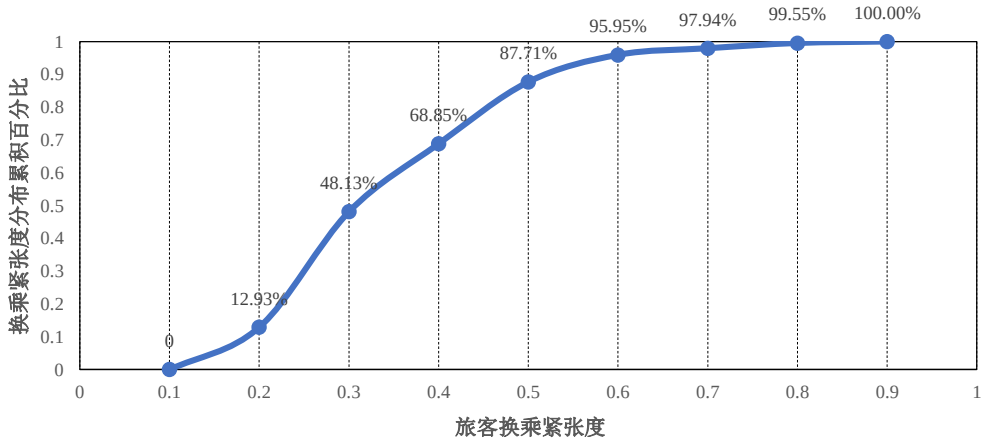

图 5-6 旅客紧张度分布

根据航班分配结果绘制甘特图，直观体现登机口的利用效果，如图 5-7 所示。

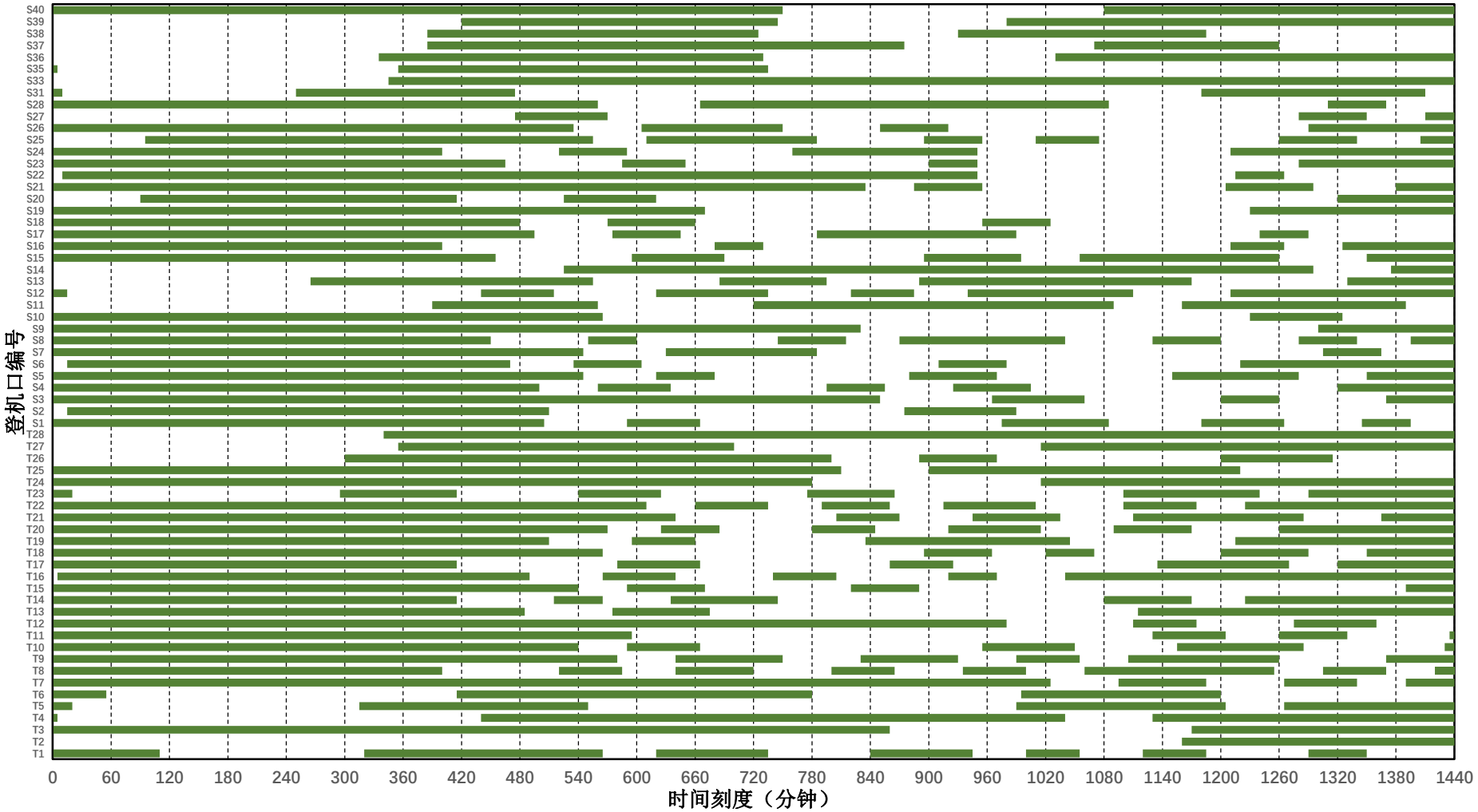

图 5-7 基于问题二航班分配结果的甘特图

5.3.2 考虑临时机位惩罚机制的求解结果

该模型通过对临时登机口的乘客换乘时间增加 6 小时的惩罚机制，保证航班和乘客成功分配登机口的数量增加，但是会延长总体最短流程时间及被使用登机口的数量。航班成功分配数量增加至 490 个，成功分配到固定机位的旅客数从 1554 人增长至 2052 人，说明该模型能够使得航班成功分配率增加；但是旅客总体最短流程时间也增加至 69480 分钟，被使用登机口数量增加至为 67 个，说明模型牺牲了部分旅客的最短流程时间而使得航班分配比例和旅客分配比例增加。在实际应用的过程中，针对不同的情况，可以判断是否选择临时机位惩罚机制来优化航班分配。

（1）航班分配情况

使用 BBO 算法对所建立模型进行求解，得出航班分配成功数量为 490 个，航班分配成功比例为 80.86%。按宽、窄体机分别画出航班分配成功的数量和比例，如图 5-8 和 5-9 所示。可以看出，窄体机的航班分配成功率达到 78.74%，宽体机的航班分配成功率达到 91.84%。

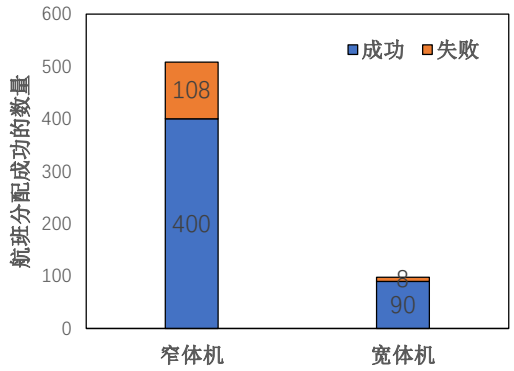

图 5-8 航班分配成功数量

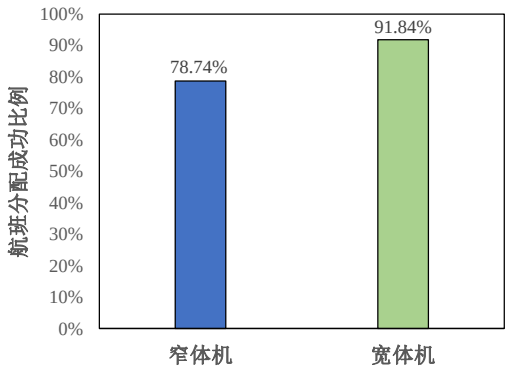

图 5-9 航班分配成功比例

（2）登机口使用情况

被使用登机口数量为 67 个，被使用登机口比例为 97.10%。T 和 S 登机口的使用数目和计算被使用登机口的平均使用率分别如图 5-10 和 5-11 所示。T 航站楼的使用数量为 28 个，S 航站楼的使用数量为 39 个，T 航站楼的平均时间使用率（61.09%）要略高于 S 航站楼的平均时间使用率（58.40%）。

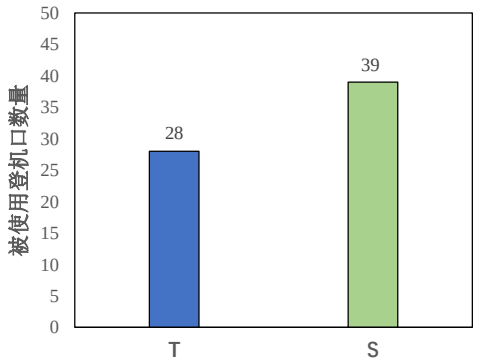

图 5-10 登机口使用数量

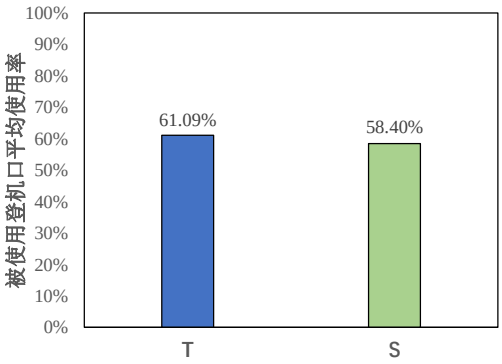

图 5-11 被使用登机口平均使用率

（3）旅客换乘情况

假设航班被分配到临时登机口不属于换乘失败的前提下，旅客的换乘失败的人数为 0，比率为 0；但是有 699 位旅客被分配到临时登机口。旅客换乘时间分

布如图 5-12 所示，约 46.35%的乘客能在 1 小时内完成换乘。

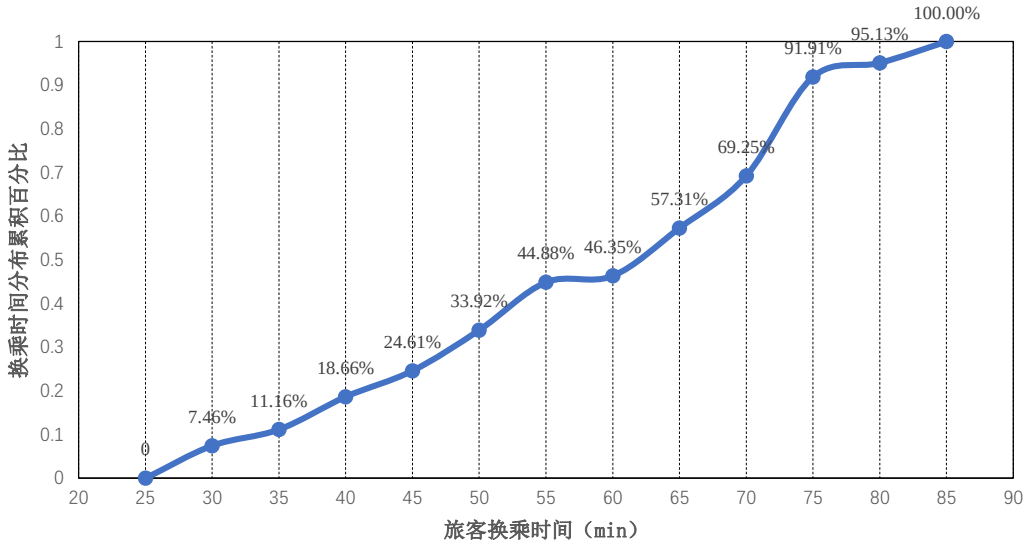

图 5-12 旅客换乘时间分布

（4）旅客紧张度分布情况

旅客紧张度分布如图 5-13 所示，有 83.77%的旅客换乘紧张度都在 0.5 及以下，说明大部分旅客换乘并不十分紧张。

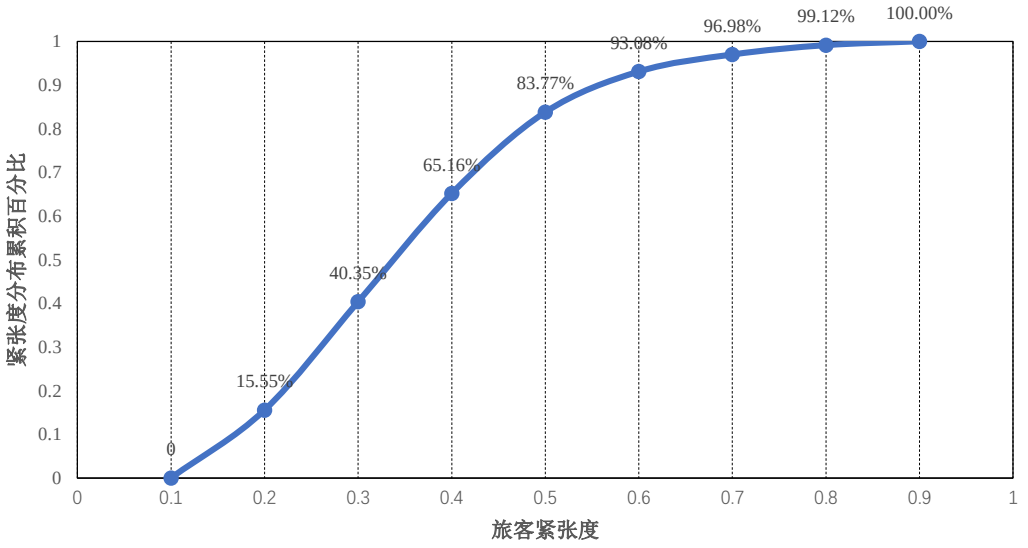

图 5-13 旅客紧张度分布

根据航班分配结果绘制甘特图，直观体现登机口的利用效果，如图 5-14 所示。

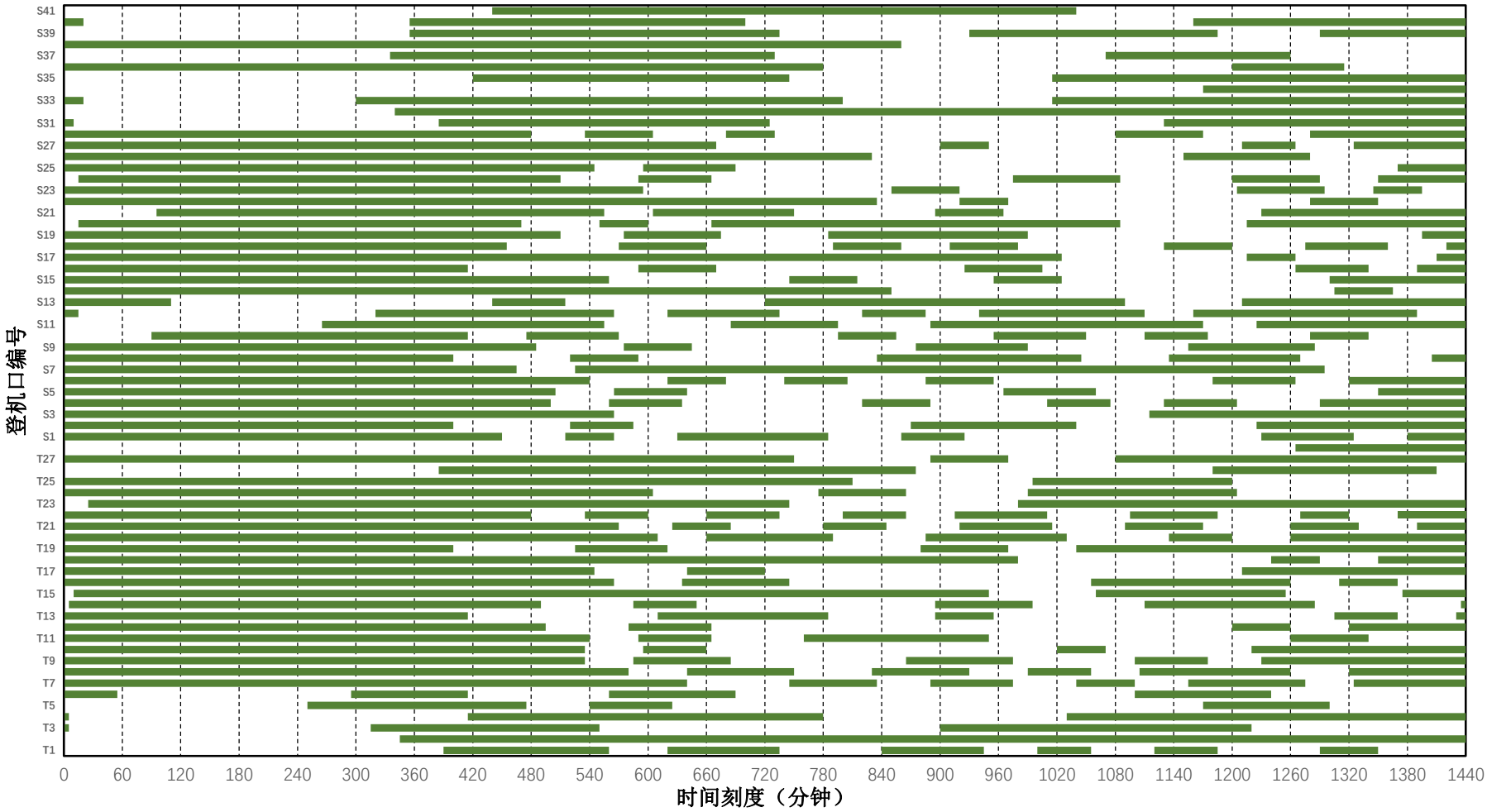

图 5-14 基于问题二航班分配结果的甘特图（加入惩罚机制）

六、 问题三模型的建立与求解

6.1 问题描述与分析

问题三是考虑中转旅客换乘时间的航班-登机口分配问题。新建卫星厅 S 对航班的最大影响是中转旅客换乘时间的可能延长，所以分析中转旅客换乘时间受卫星厅影响机理是具有十分重要的应用价值。本问题是在问题二的基础上加入旅客换乘连接变量，引入“换乘紧张度”参量评价新建卫星厅对旅客换乘时间的影响。问题三首先要求航班分配成功的数量最大，其次要求最小化中转旅客的总体换乘紧张度，并且在此基础上最小化被使用登机口的数量。

问题三种引入了旅客捷运时间和行走时间两个时间参量，与最短流程时间共同组成了旅客的换乘时间。与问题二不同之处在于，问题三中的中转旅客的行走时间与登机口所在终端厅及其区域有关，与问题二相比，问题三的求解过程中增加了一个登机口区域的属性变量。基于问题三的要求，本文建立多目标多约束的登机口指派优化模型。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。

（1）目标函数的建立

在问题一的符号说明的基础上，本文引入以下符号定义：

$T2_{g_k,g_m}$ 为登机口 g_k 和登机口 g_m 间的中转旅客捷运时间，与登机口 g_k、g_m 的功能属性有关（假设 g_k 为旅客到达登机口， g_m 为旅客出发登机口， $k,m\in G$），具体时间根据题目已有条件确定；

$T3_{g_k,g_m}$ 为登机口 g_k 和登机口 g_m 间的中转旅客步行时间，与登机口 g_k、g_m 的功能属性有关（假设 g_k 为旅客到达登机口， g_m 为旅客出发登机口， $k,m\in G$），具体时间根据题目已有条件确定，如下表所示：

表 6-1 旅客步行时间

登机口区域	0	1	2	3	4	5	6
0	10	15	20	25	20	25	25
1	15	10	15	20	15	20	20
2	20	15	10	25	20	25	25
3	25	20	25	10	15	20	20
4	20	15	20	15	10	15	15
5	25	20	25	20	15	10	20
6	25	20	25	20	15	20	10

注：0~6 以此代表登机口的 T-N,T-C,T-S,S-N,S-C,S-S,S-E.

应用上述的符号，结合问题三要求，在分配登机口的时候，保证分配到合适登机口的航班数量最大化，可建立如下的目标函数一：

$$\min Z_1=\frac{m-\sum_{f_i\in F,g_k\in G} y_{f_i,g_k}}{m} \tag{23}$$

目标函数 Z_1 保证未分配到登机口的航班比例最小，符合问题三的要求；

结合问题三要求，最小化中转旅客的换乘紧张度，根据换乘紧张度的定义，得到换

乘旅客在登机口 g_k 和登机口 g_m 之间的换乘时间为 $T1_{g_k,g_m}+T2_{g_k,g_m}+T3_{g_k,g_m}$，航班连接时间为 $d_{f_j}-a_{f_i}$，则换乘紧张度可得如下公式：

$$\mu_{f_i,f_j}=y_{f_i,g_k}\times y_{f_j,g_m}\times x_{f_i,f_j}\times(T1_{g_k,g_m}+T2_{g_k,g_m}+T3_{g_k,g_m})/(d_{f_j}-a_{f_i}) \quad (24)$$

根据问题三的要求，中转旅客的总换乘紧张度最小，可建立如下的目标函数二：

$$\min Z_2=\sum_{\substack{f_i\in F\\ f_j\in F\\ i\neq j}}\sum_{\substack{g_k\in G\\ g_m\in G}} m_{g_k,g_m}\times\mu_{f_i,f_j} \quad (25)$$

在目标函数 Z_2 中，首先确定每个飞机分配到了与其属性相符的登机口，进而判断飞机 f_i 的到达航班与飞机 f_j 的出发航班是否为 tickets 信息中转旅客的到达航班和出发航班所组成的航班对，分别确定到达航班与出发航班分配到的登机口位置，根据航班类型和登机口所在的终端厅及区域，确定中转旅客的最短流程时间、捷运时间和步行时间；根据 tickets 信息确定中转旅客的航班连接时间以及中转旅客数量，进而得到中转旅客的总换乘紧张度。优化目标 Z_2 使得中转旅客的总体换乘紧张度最小。

结合问题二要求，最小化被使用登机口的数量，可建立如下的目标函数三：

$$\min Z_3=\frac{\sum_{g_k\in G} z_{g_k}}{n} \quad (26)$$

目标函数 Z_3 表示登机口使用的比率最小，符合问题二的要求。

问题二中的约束条件与问题一中的约束条件一致，所以针对问题三可建立如下的多目标多约束登机口分配优化模型：

$$\min Z_1=\frac{m-\sum_{f_i\in F,g_k\in G} y_{f_i,g_k}}{m}$$

$$\min Z_2=\sum_{\substack{f_i\in F\\ f_j\in F\\ i\neq j}}\sum_{\substack{g_k\in G\\ g_m\in G}} m_{g_k,g_m}\times\mu_{f_i,f_j}$$

$$\mu_{f_i,f_j}=y_{f_i,g_k}\times y_{f_j,g_m}\times x_{f_i,f_j}\times(T1_{g_k,g_m}+T2_{g_k,g_m}+T3_{g_k,g_m})/(d_{f_j}-a_{f_i})$$

$$\min Z_3=\frac{\sum_{g_k\in G} z_{g_k}}{n}$$

约束条件

$$(C_{f_i}-C_{g_k})\times y_{f_i,g_k}=0$$

$$(C^a_{f_i}-C^a_{g_k})\times y_{f_i,g_k}=0$$

$$(C^d_{f_i}-C^d_{g_k})\times y_{f_i,g_k}=0$$

$$\sum_{g_k\in G} y_{f_i,g_k}=1$$

$$y_{f_i,g_k}\times y_{f_j,g_k}\times(d_{f_i}-a_{f_j})\times(d_{f_j}-a_{f_i})<0$$

$$a_{f_j}-d_{f_i}+(1-y_{f_i,g_k}\times y_{f_j,g_k})M\geq T,\ \ i<j$$

$$f_i\in F,\ g_k\in G$$

$$i,j,k\in Z^+$$

6.2 模型的求解

在问题二的基础上，问题三改变了一个优化目标，即使得中转旅客的总体紧张度最小化，替代问题二中的旅客最短流程时间总体最小，同样属于多目标多约束优化问题，约束条件与问题二一致。根据问题三的说明，确定三个目标函数的有限顺序，即分配登机口航班数量最大、中转旅客总体紧张度最小、登机口使用数量最小。在求解的过程中，不断改变三个目标函数的权重，从而得到一个总体程度最优的解。

（1）根据题目中的信息提示我们将转场飞机记录号、旅客记录号进行筛选匹配，对 Pucks 和 Tickets 附件中的数据提取出 20 日到达或 20 日出发的航班和旅客，其中包括 19 日到达 20 日出发的航班和旅票信息，同时对 Gates 附件中登机口的字符进行量化标记便于后续处理。读取 Weight 数据表中走行时间的权重矩阵。

（2）ticket[NT] 结构体用来存储旅客的机票信息；aircraft[FA] 结构体存储航班信息；建立 gate[DD] 结构体存储登机口信息，具体信息见 5.2。

（3）分别提取旅客的到达航班编号（arrive_num）、出发航班编号（depart_num），与航班信息中的到达航班编号（arrive_num）、出发航班编号（depart_num）相匹配，进而确定到达航班和出发航班的飞机转场记录号（aircraft_rn）、到达航班类型（arrive_type）、出发航班类型（depart_type）、到达航班时刻（arrive_time）、出发航班时刻（depart_time）、飞机型号（aircraft_jx）等信息；

（4）根据旅客的到达航班时刻（arrive_time）、出发航班时刻（depart_time），确定每位中转旅客的航班连接时间（connection_time= depart_time- arrive_time）。

（5）BBO 算法优化求解

Step1. 将飞机到达航班类型（arrive_type）、出发航班类型（depart_type）、飞机型号（aircraft_jx）等信息与登机口到达类型（arrive_type）、出发类型（depart_type）、登机口机体类别（gate_jx）一一对应，为每架飞机（aircraft_num）筛选出属性匹配的登机口（gate_num）；

Step2. 随机分配飞机给属性匹配的登机口，确定每个登机口 i 的开放时间（topen[i]），即该登机口前序飞机的出发航班时刻，同时满足 45min 的空档间隔时间；与未分配登机口的飞机的到达航班时刻（arrive_time）相比较，满足要求即可分配给该登机口，否则拒绝分配，搜索其他可分配登机口；从而生成初始可行解，完成机位初始分配；

Step3. 根据旅客的到达航班和出发航班的飞机转场记录号（aircraft_rn），确定飞机分配到的登机口编号（gate_num），进而判断登机口所属终端厅区域（gate_region）；根据 gate_type 计算每位中转旅客的最短流程时间（processtime）、捷运时间（bus_time）以及行走时间（walk_time），进而确定旅客的换乘时间（transfer_time= process_time+ bus_time+ walk_time）；

Step4. 根据旅客记录号（lvpiao_rn），中转旅客的航班连接时间（connection_time）和换乘时间（transfer_time），计算总体换乘紧张度为单个换乘紧张度乘以人数（urgent_time=transfer_time/connection_time）*ticket_pn。

Step5. 对 *Step2* 生成的初始可行解进行有效评估，结合 *Step3* 和 *Step4* 并带入目标函数 Z_1、Z_2 和 Z_3 的计算中，在定义的种群数量范围内进行更新计算。

Step6. 利用 BBO 算法优化种群所有解，在满足约束条件的基础上对登机口多次迭代，经过多次迁移和突变操作，并用上代最优值替换本代最差值，与当前最优解结果比较，判断是否更优。

Step7. 判断是否满足迭代次数，输出当前最优解。

同样问题三的求解过程中，我们将临时旅客换乘时间按照计入和不计入目标函数 Z_2

分别进行模型的求解，具体的求解结果见 6.3。

6.3 求解结果与分析

6.3.1 未考虑临时机位惩罚机制的求解结果

根据问题三所要求的优化目标，该模型针对三个目标进行优化，分别得到航班分配成功的数量为 480 个，总流程时间为 45433 分钟，登机口使用数量为 66 个。该模型将更多的乘客的航班划入了临时登机口以缩短总换乘时间，但是分配到临时机位的旅客共有 1161 位（共有 2751 人）。

（1）航班分配情况

使用 BBO 算法对所建立模型进行求解，得出航班分配成功数量为 480 个，航班分配成功比例为 79.21%。按宽、窄体机分别画出航班分配成功的数量和比例，如图 6-2 和 6-3 所示。可以看出，窄体机的航班分配成功率达到 75.59%，宽体机的航班分配成功率达到 97.96%。

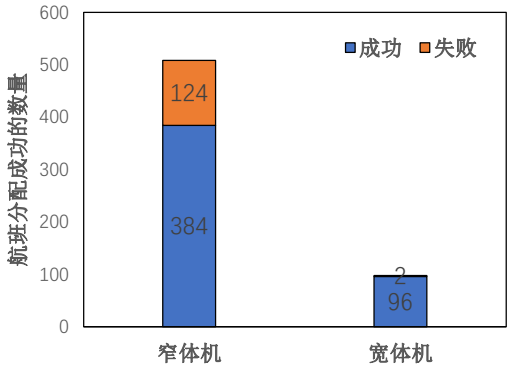

图 6-2 航班分配成功数量

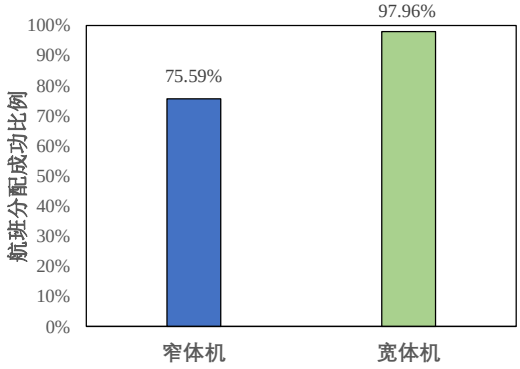

图 6-3 航班分配成功比例

（2）登机口使用情况

被使用登机口数量为 66 个，被使用登机口比例为 95.65%。T 和 S 登机口的使用数目和计算被使用登机口的平均使用率分别如图 6-4 和 6-5 所示。T 航站楼的使用数量为 28 个，S 航站楼的使用数量为 38 个，T 航站楼的平均时间使用率（59.76%）要略高于 S 航站楼的平均时间使用率（56.03%）。

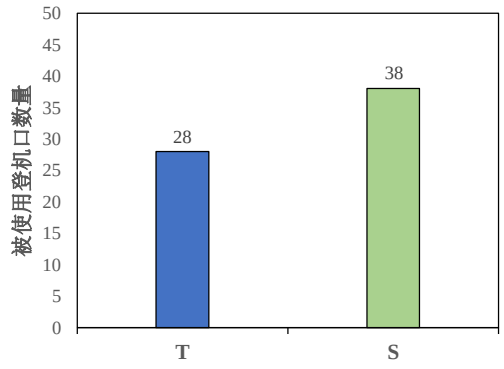

图 6-4 登机口使用数量

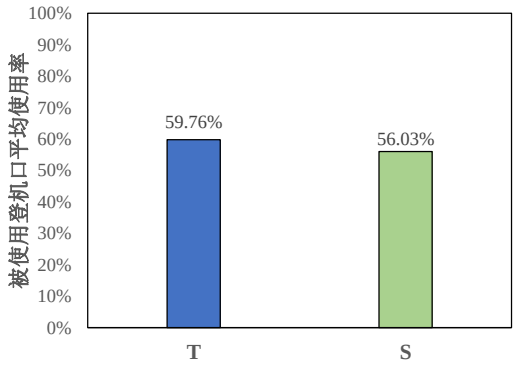

图 6-5 登机口平均使用率

（3）旅客换乘情况

假设航班被分配到临时登机口不属于换乘失败的前提下，旅客的换乘失败的人数为 0，比率为 0；但是有 1161 位旅客被分配到临时登机口。旅客换乘时间分布如图 6-6 所示，约 50.57%的乘客能在 1 小时内完成换乘。

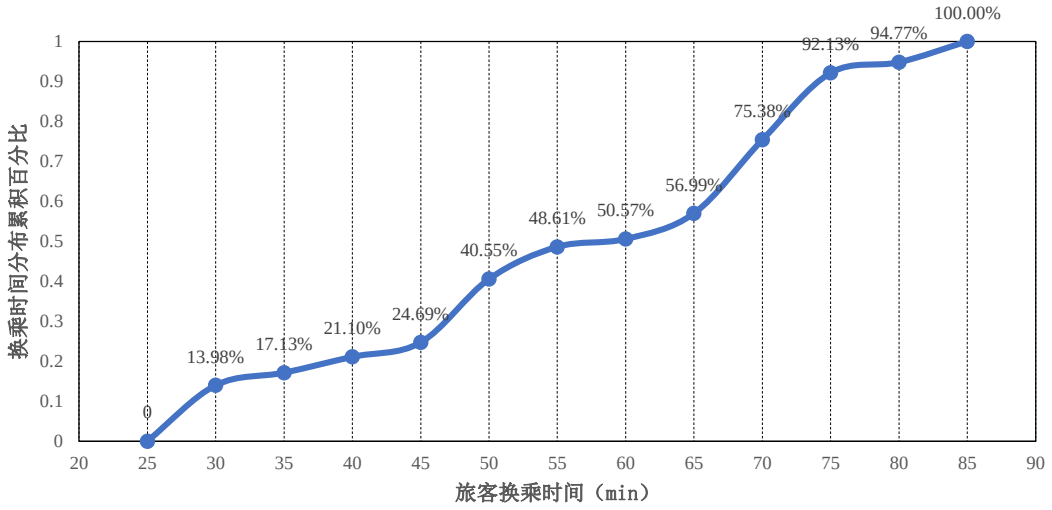

图 6-6 旅客换乘时间分布

（4）旅客紧张度分布情况

旅客紧张度分布如图 6-7 所示，有 86.27%的旅客换乘紧张度都在 0.5 及以下，说明大部分旅客换乘并不十分紧张。

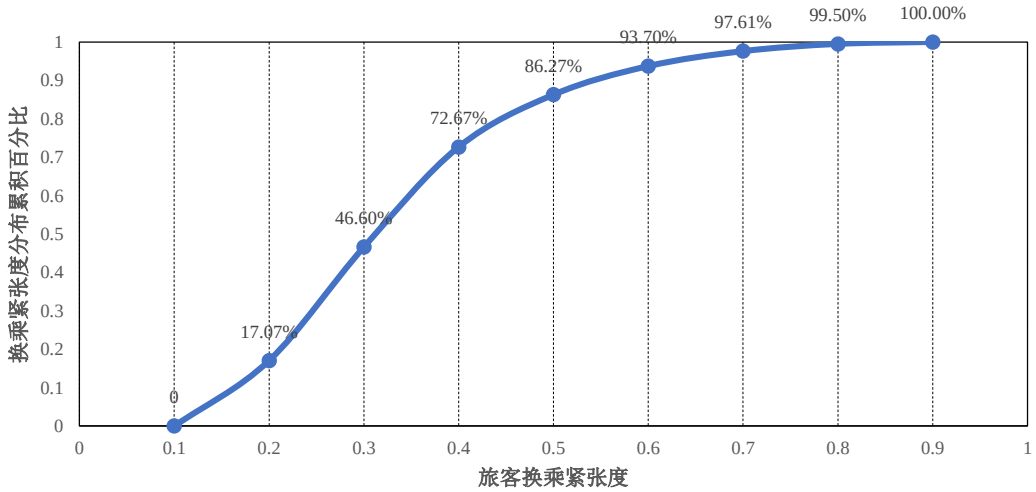

图 6-7 旅客换乘紧张度分布

根据航班分配结果绘制甘特图，直观体现登机口的利用效果，如图 6-8 所示

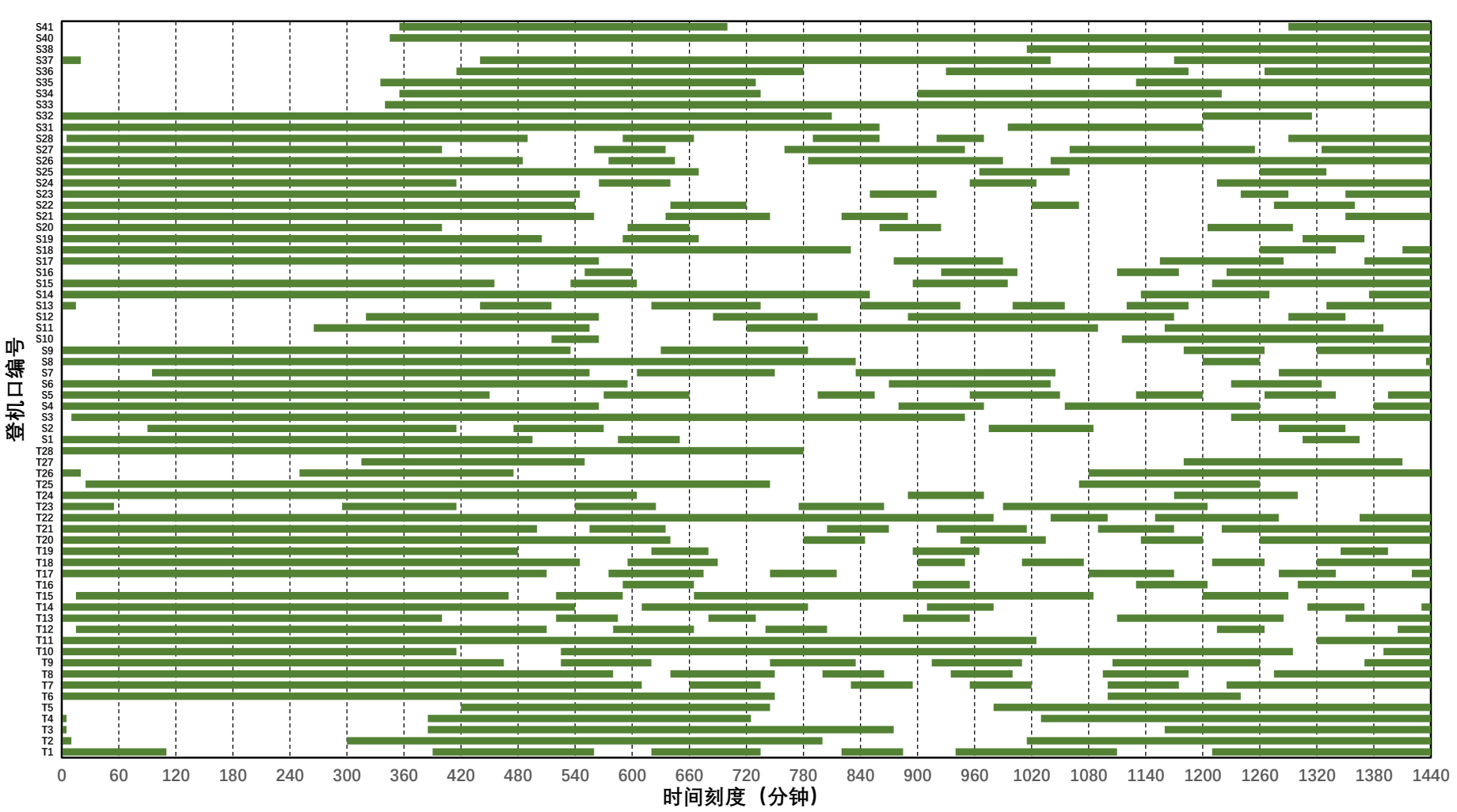

图 6-8 基于问题三航班分配结果的甘特图

6.3.2 考虑临时机位惩罚机制的求解结果

航班成功分配数量增加至 502 个，成功分配到固定机位的旅客数从 1588 人增长至 2025 人，说明该模型能够使得航班和乘客成功分配率增加；但是旅客总体最短流程时间也增加至 68255 分钟，被使用登机口数量保持为 66 个，说明模型牺牲了部分旅客的最短流程时间而使得航班分配比例和旅客分配比例增加。在实际应用的过程中，针对不同的情况，可以判断是否选择临时机位惩罚机制来优化航班分配。

（1） 航班分配情况

使用 BBO 算法对所建立模型进行求解，得出航班分配成功数量为 502 个，航班分配成功比例为 82.84%。按宽、窄体机分别画出航班分配成功的数量和比例，如图 6-9 和 6-10 所示。可以看出，窄体机的航班分配成功率达到 79.45%，宽体机的航班分配成功率达到 100%。

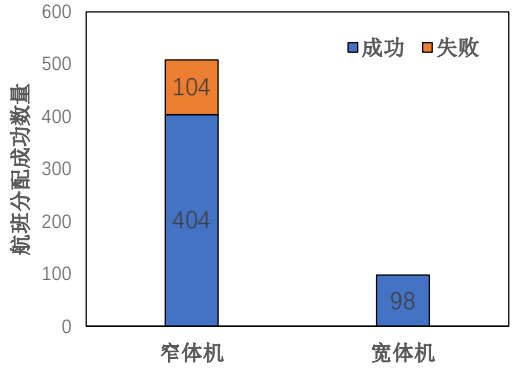

图 6-9 航班分配成功数量

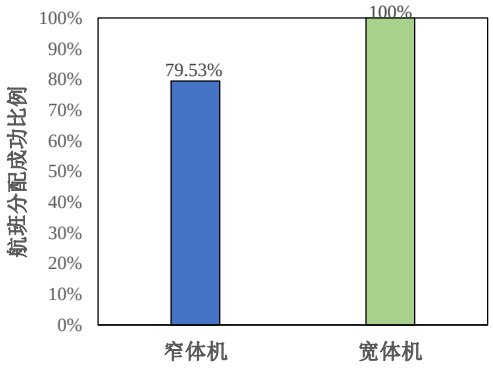

图 6-10 航班分配成功比例

（2）登机口使用情况

被使用登机口数量为 66 个，被使用登机口比例为 95.65%。T 和 S 登机口的使用数目和计算被使用登机口的平均使用率分别如图 6-11 和 6-12 所示。T 航站楼的使用数量为 27 个，S 航站楼的使用数量为 39 个，T 航站楼的平均时间使用率（63.39%）要略高于 S 航站楼的平均时间使用率（58.70%）。

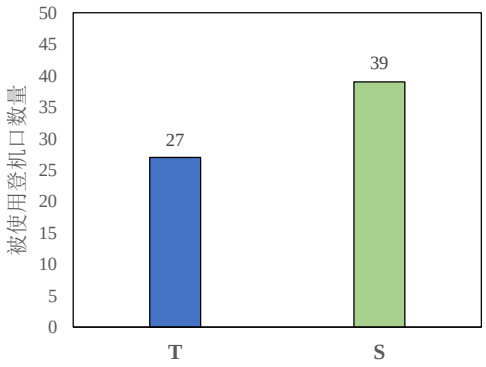

图 6-11 登机口使用数量

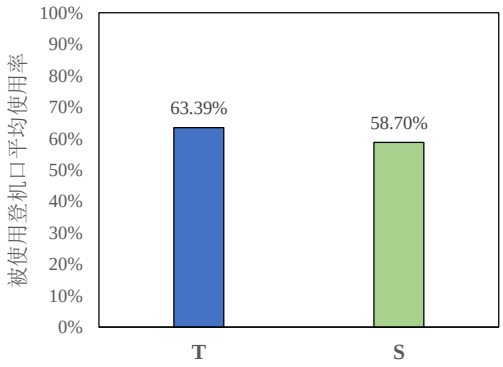

图 6-12 登机口平均使用率

（3）旅客换乘情况

假设航班被分配到临时登机口不属于换乘失败的前提下，旅客的换乘失败的人数为 0，比率为 0；但是有 726 位旅客被分配到临时登机口。旅客换乘时间分布如图 6-13 所示，约 47.65%的乘客能在 1 小时内完成换乘。

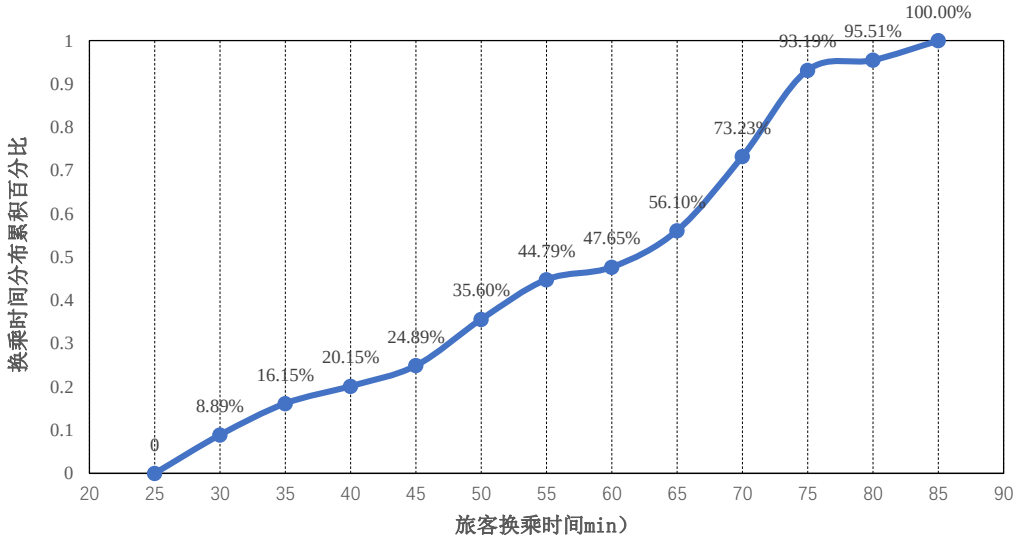

图 6-13 旅客换乘时间分布

（4）旅客紧张度分布情况

旅客紧张度分布如图 6-14 所示，有 85.38%的旅客换乘紧张度都在 0.5 及以下，说明大部分旅客换乘并不十分紧张。

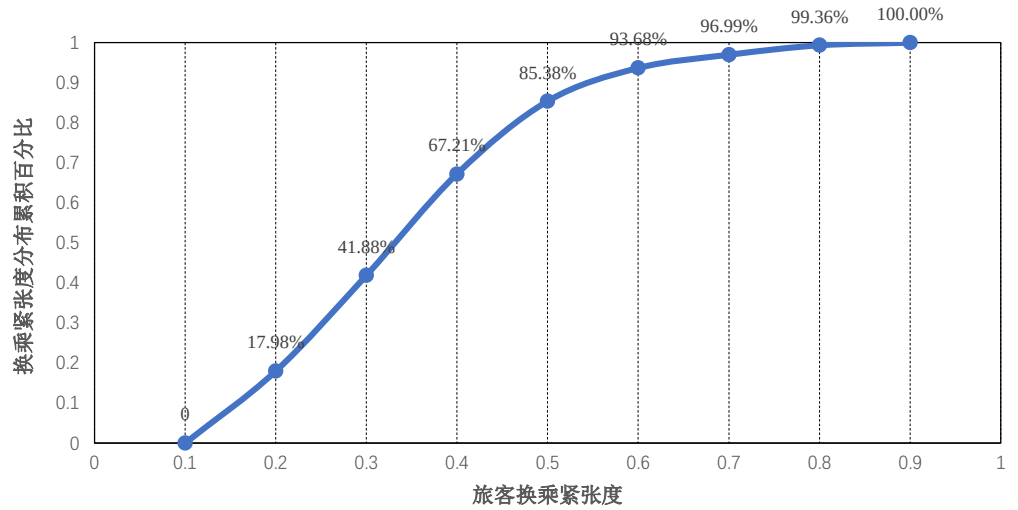

图 6-14 旅客换乘紧张度分布

根据航班分配结果绘制甘特图，直观体现登机口的利用效果，如图 6-15 所示。

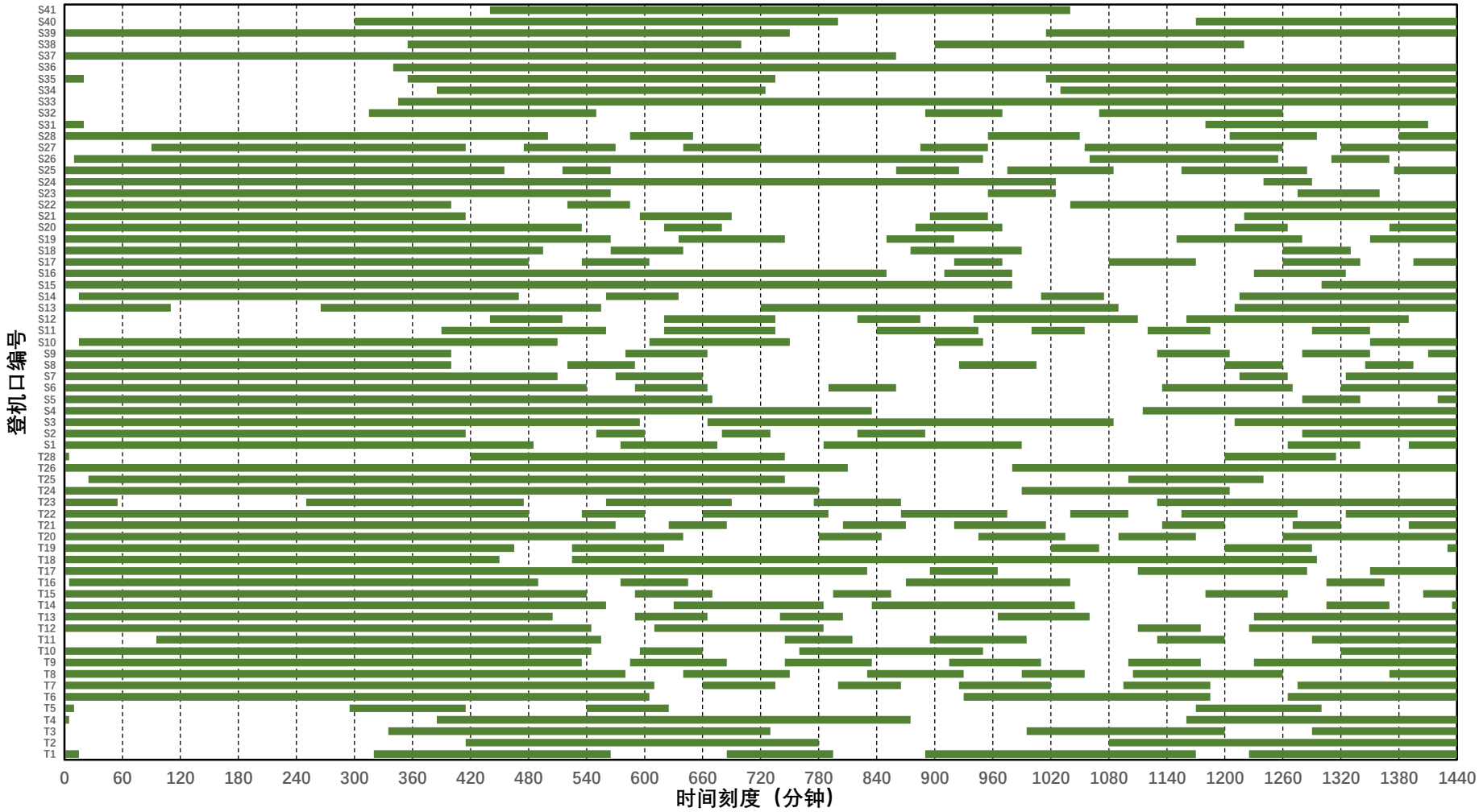

图 6-15 基于问题三航班分配结果的甘特图（加入惩罚机制）

七、模型对比分析

针对三个问题总共获得五个模型，分别是针对问题一的模型、针对问题二、三的未加入对临时机位的惩罚机制的模型、加入对临时机位的惩罚机制的模型。将五个模型的所有求解结果所对应的各类指标进行计算并整理成表格，如表 7-1 所示，结合该表格比较模型之间的差异并分析产生原因：

（1）问题一、二和三模型纵向对比分析

问题一的首要目标是尽可能的分配航班到合适的登机口，而在问题二和问题三未考虑对临时机位进行惩罚的模型中，需要进一步流程时间和乘客换乘因素，因此削弱了尽可能的分配航班到合适的登机口这一首要目标的权重，因此在以上三个模型中，第一个模型的成功分配航班比率是最高的，达到 82.18%；问题二中未考虑对临时机位进行惩罚的模型获得了所有模型中最小的总最短流程时间 50350 分钟；问题三中未考虑对临时机位进行惩罚的模型获得了所有模型中最低的总紧张度 454.33。

（2）问题二模型横向对比分析

问题二中未加入对临时机位惩罚的模型中，总最短流程时间为 50350 分钟，加入惩罚机制后达到 69480 分钟，同时正确换乘人数由 1554 升到 2052，成功分配航班比率由 77.56%升到 80.86%。以总最短流程时间为优化目标之一时，由于目标函数值是以乘客为单位进行累加，因此模型会倾向于将人数较多的航班分配到临时机位，从而导致正确换乘人数的显著减少，考虑到这个问题，将分配到临时机位的航班进行以乘客为单位的时间惩罚项，从而抑制对于临时机位的分配倾向。因此加入惩罚项后，正确换乘人数显著上升，同时成功分配航班比例也显著上升，而登机口使用数量未明显增加，虽然总最短流程时间上升，但能够满足更多乘客的需求。

（3）问题三模型横向对比分析

问题三未加入对临时机位惩罚的模型中，总紧张度为 454.33，加入惩罚机制后达到 607.26，同时换乘人数由 1588 人上升到 2025 人，成功分配航班比率由 79.21%上升到 82.24%。以总紧张度为优化目标之一时，由于跟（3）同样的原因导致正确换乘人数的显著减少。考虑到这个问题，同样将分配到临时机位的航班进行以乘客为单位的时间惩罚项，从而抑制对于临时机位的分配倾向。因此加入惩罚项后，正确换乘人数显著上升，同时成功分配航班比例也显著上升，而登机口使用数量未明显增加，虽然总紧张度上升，但能够满足更多乘客的需求。

表 7-1 模型求解结果对应各类指标对比

		成功分配航班比例	登机口使用数量	登机口平均使用率	正确换乘人数	总最短流程时间(min)	总换乘时间	总紧张度
问题一模型		82.18%	65	**61.25%**	1926	65725	67513	591.11
问题二模型	未加入惩罚项	77.56%	**64**	58.47%	1554	**50350**	**53838**	474.65
	加入惩罚项	80.86%	67	59.51%	**2052**	69480	71549	630.70

问题三模型	未加入惩罚项	79.21%	66	57.96%	1588	53485	54094	**454.33**
	加入惩罚项	**82.84%**	66	60.62%	2025	68255	69417	607.26

八、 模型的总结与评价

8.1 模型总结

三个问题所对应的模型均为多目标多约束模型。三个模型的假设前提、目标和约束条件有所不同，因此适应场景也不同。

1. 问题一模型是以航班分配失败率最小和被使用登机口比率最小为目标的双目标多约束优化模型，该模型对航班分配进行简化，未考虑航班衔接换乘，是目标函数比较简单的规划模型，而由于假设不考虑乘客中转换乘的费用，因此获得的方案很难适用于实际情况。

2. 问题二模型是以航班分配失败率最小（目标一）、中转旅客总体最短流程时间最小（目标二）以及被使用登机口比率最小（目标三）为目标的的多目标多约束的登机口分配优化模型，该模型就是否加入对临时机位的惩罚机制衍生出两个不同的模型，在假设临时机位不属于换乘失败的前提下，不加入对临时机位的惩罚会使模型倾向于将航班分配到临时机位，而尽管总流程时间降低，但大量乘客匹配到临时机位无法正常换乘。对临时机位进行时间惩罚，使得总流程时间和乘客正常换乘数量得以平衡。

3. 问题三模型是以航班分配失败率最小（目标一）、中转旅客总体紧张度最小（目标二）以及被使用登机口比率最小（目标三）为目标的多目标多约束的登机口分配优化模型。该模型同样就是否加入对临时机位的惩罚机制衍生出两个不同的模型。对临时机位进行时间惩罚，使得总体紧张度和乘客正常换乘数量得以平衡。

8.2 模型评价

模型的优点在于：

1. 目标函数定义明确，对多个目标函数进行权重的调整能够获得不同的方案。

2. 主要运用 BBO 算法对模型进行求解，该算法的优势在于适用于解决高维度的，多目标的最优化问题；

模型的缺点在于：

1. 参数引入过多，模型还有简化的余地；

2. 模型目标多为累积指标（如总流程时间、总换乘紧张度），对于平均指标的考虑不足。

3. 模型只用了 BBO 算法进行求解，未和其它启发式算法如遗传算法、模拟退火算法等进行对比验证。

九、 参考文献

[1] Abdelghani B, Mageed A. G, 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

[2] Marinelli M, Dell O M, Sassanelli D. A Metaheuristic Approach to Solve the Flight Gate Assignment Problem [J]. Transportation Research Procedia, 2015, 5:211–220.

[3] Zhang H H, Xue Q W, Jiang Y. Multi-objective gate assignment based on robustness in hub airports[J]. 9,2(2017-2-01), 2017, 9(2):168781401668858.

[4] Yu C, Lau H Y K. Airport gate reassignment based on the optimization of transfer passenger connections[C]//4th International Conference on Traffic and Logistic Engineering (ICTLE 2015). 2015.
[5] Narciso M E, Piera M A. Robust gate assignment procedures from an airport management perspective[J]. Omega, 2015, 50:82–95.
[6] 王笑天, 田勇, 万莉莉,等. 基于列生成算法的停机位指派的鲁棒性研究[J] 武汉理工大学学报：交通科学与工程版, 2015:171-174. DOI:doi:10.3963/j.issn.2095-3844.2015.01.039.
[7] 李军会, 朱金福, 陈欣. 基于航班延误分布的机位鲁棒指派模型[J]. 交通运输工程学报, 2014.
[8] 罗宇骁. 基于 BBO 算法的大型机场滑行道和停机位联合调度研究[D]. 南京航空航天大学, 2016.
[9] 鲁宇明, 王彦超, 刘嘉瑞,等. 一种改进的生物地理学优化算法[J]. 计算机工程与应用, 2016, 52(17):146-151.

附录：

程序算法已模块化，使用 Visual Studio2010 编程平台，操作系统为 Windows 7 旗舰版，处理器为 Inter® Core(TM) i7-4790，主频 3.6GHz，内存 8G。程序共分为两个部分，分别是：

1）C 语言编写主体程序，包括文件读取、生成可行解（模型约束条件：飞机空挡间隔 45min、登机口宽窄类型、国内外到达类型）、评估可行解（区分问题 1，2，3 的关键步骤）、种群更新计算、BBO 算法优化数据；

2）C++针对原始数据预处理以及对 1）中的航班-登机口分配结果进行数据分析验证，包括对提供的数据（InputData.xlsx）中的 Pucks/Tickets/Gates 数据的筛选、匹配和临时机位/最小使用登机口数量/最短换乘时间/最短流程/紧张度/正确换乘人数的对比验证分析。

附录 1、问题的求解

已对于题中三个问题的算法代码进行系统性地整合，对于每个问题的求解，则需要修改对应的评估目标函数优化代码即可。见代码中 void Evaluategatebase(int zq) 已做明确标记。;

数据结构定义：

```
#define GR 250          //定义种群数量
#define DD 69           //停机位总数目
#define FA 303          //飞机总数
#define NT 1703         //旅票数量
#define HCSJ 45         //飞机推出需要的时间
#define NCga 1500       //停机位的迭代次数
#define Alph 0.8        //远机位率
#define MUprob 0.04     //变异的概率
#define Beta 0.5        //blend

double bestgaapr;
double bestgarub;
double bestgahis;
double resultallot[2]={0,0};     //停机位分配数量和临时机位数量
int bestgate[FA];                //停机位最优值
int graphweight[7][7]={0};       //区域权重数据

struct feiji   //飞机信息结构体
{
    char num[10];               //进港航班号
    int intime;                 //记录到达时间
    int outtime;                //记录出发时间
    int jx;                     //记录机型
    int feiji_arrivetype;       //到达类型
    int feiji_departtype;       //出发类型
```

```
} aircraft[FA];

struct tingjiwei //停机位信息结构体
{
      int jp;                    //所属停机坪号
      int jx;                    //停机位可容纳的机型
     int tingji_arrivetype;        //停机位到达类型
     int tingji_departtype;       //停机位出发类型
     int dir;                    //权重大小
} gateh[DD];

struct lvpiao //旅票信息结构体
{
      char rn[10];                       //旅客记录号
      int pn;                            //乘客数
     char lvpiao_arriveplane[10];     //到达航班
     int lvpiao_arrivedate;            //到达日期
     char lvpiao_departplane[10];    //出发航班
     int lvpiao_departdate;            //出发日期
     char lvpiao_trans1num[10];      //转场记录号到达
     char lvpiao_trans2num[10];      //转场记录号出发
     int lvpiao_arrivetype;            //到达类型
     int lvpiao_departtype;            //出发类型
     int lvpiao_arrivetime;            //到达时间
     int lvpiao_departtime;           //出发时间
} ticketn[NT];

struct gtzhongqunjihe //种群结合结构体
{
      double geti[FA]; //定义个体
      double apr;          //定义远机位个数
      double rub;          //定义鲁棒性
      double his;          //定义总栖息地适宜度
}gatebbogroup[GR];     //定义算法种群集合

void Inputgatehs(FILE *fp);
void Inputaircrafts(FILE *fp);
void Inputickets(FILE *fp);
void Inputtransweights(FILE *fp);
void Trantimes();
void Initialfligenerate();
void Gatereassign(int zq);
void Evaluategatebase(int zq);
void Sortgatezhongqun();
```

```
void Bbogateprocess();
void muavailibit(int zq);
int comp(const void *a, const void *b);
bool IsArrayRepeat(int *arr, int len);
double Rubucontmax();
double Rubucontmin();
```

主体程序介绍：

```
//-------------------------------------------------------------
//*************************************************************
//主函数入口信息
//*************************************************************
int main()
{
int i,j,zq,k;
FILE *fp5,*fp6,*fp7,*fp8,*fl;
srand(time(NULL));

fp5 = fopen("gate.csv","at+");
Inputgatehs(fp5);                          //输入登机口信息
fp6 = fopen("aircraft.csv","at+");
Inputaircrafts(fp6);                       //输入飞机信息
fp7 = fopen("tickets-Input time.csv","at+");
Inputickets(fp7);                          //输入旅票信息
fp8 = fopen("trans_weight.csv","at+");    //输入位置权重信息
Inputtransweights(fp8);


fl = fopen("航班分布停机位输出文件_out.txt","a");          //将结果保存在 out.txt 这
个文件里面
fclose(fp5);
fclose(fp6);
fclose(fp7);
fclose(fp8);

//Trantimes();                        //时间十进制（数据预处理已结束）

Initialfligenerate();                  //生成初始航班
Bbogateprocess();                      //停机位的 BBO 过程

fprintf(fl,"%s\n","本次算法中的停机位信息:");

for(j=0;j<FA;j++)
fprintf(fl,"%d -> ",bestgate[j]);
```

```
fprintf(fl,"\n");
fprintf(fl,"%lf %lf %lf %lf\n\n\n",bestgahis,bestgarub,bestgaapr,Alph);

fclose(fl);
return 0;

}

//-------------------------------------------------------------
//*************************************************************
//读取 Gate 文件获得登机口信息
//*************************************************************
void Inputgatehs(FILE *fp)
{
int i;
if(fp==NULL)
{
printf("Sorry,the file is not exist\n");
exit(1);
}
else
{
for(i=0;i<DD;i++)
      fscanf(fp,"%d,%d,%d,%d,%d",&gateh[i].jp,&gateh[i].jx,&gateh[i].tingji_arrivety
pe,&gateh[i].tingji_departtype,&gateh[i].dir); //修改增加了机型，注意
}
}

//-------------------------------------------------------------
//*************************************************************
//读取 Aircraft 文件获得登机口信息
//*************************************************************
void Inputaircrafts(FILE *fp)
{
int i;
if(fp==NULL)
{
printf("Sorry,the file is not exist\n");
exit(1);
}
else
{
char line[1024];
for(i=0;i<FA;i++)
```

```
{
    fgets(line,sizeof(line),fp);
    char *save_ptr;
    char *num,*intime,*outtime,*jx,*artype,*dptype;
    num=strtok_s(line,",",&save_ptr);
    intime=strtok_s(NULL,",",&save_ptr);
    outtime=strtok_s(NULL,",",&save_ptr);
    jx=strtok_s(NULL,",",&save_ptr);
    artype=strtok_s(NULL,",",&save_ptr);
    dptype=strtok_s(NULL,",",&save_ptr);
    strcpy(aircraft[i].num,num);
    std::string txt1 = std::string(intime);
    std::string txt2 = std::string(outtime);
    std::string txt3 = std::string(jx);
    std::string txt4 = std::string(artype);
    std::string txt5 = std::string(dptype);
    aircraft[i].intime=atoi(txt1.c_str());
    aircraft[i].outtime=atoi(txt2.c_str());
    aircraft[i].jx=atoi(txt3.c_str());
    aircraft[i].feiji_arrivetype=atoi(txt4.c_str());
    aircraft[i].feiji_departtype=atoi(txt5.c_str());
}
}
}

//---------------------------------------------------------------
//*****************************************************************
//读取 Tickets 文件获得登机口信息
//*****************************************************************
void Inputickets(FILE *fp)
{
int i;
if(fp==NULL)
{
printf("Sorry,the file is not exist\n");
exit(1);
}
else
{
char line[1024];
for(i=0;i<NT;i++)
{
    fgets(line,sizeof(line),fp);
    char *save_ptr;
```

```
        char *rn,*pn,*apl,*adt,*dpl,*ddt,*tar,*tdr,*atype,*dtype,*atime,*dtime;
        rn=strtok_s(line,",",&save_ptr);
        pn=strtok_s(NULL,",",&save_ptr);
        apl=strtok_s(NULL,",",&save_ptr);
        adt=strtok_s(NULL,",",&save_ptr);
        dpl=strtok_s(NULL,",",&save_ptr);
        ddt=strtok_s(NULL,",",&save_ptr);
        tar=strtok_s(NULL,",",&save_ptr);
        atype=strtok_s(NULL,",",&save_ptr);
        tdr=strtok_s(NULL,",",&save_ptr);
        dtype=strtok_s(NULL,",",&save_ptr);
        atime=strtok_s(NULL,",",&save_ptr);
        dtime=strtok_s(NULL,",",&save_ptr);

        std::string txt1 = std::string(pn);
        std::string txt2 = std::string(adt);
        std::string txt3 = std::string(ddt);
        std::string txt4 = std::string(atype);
        std::string txt5 = std::string(dtype);
        std::string txt6 = std::string(atime);
        std::string txt7 = std::string(dtime);

        strcpy(ticketn[i].rn,rn);
        strcpy(ticketn[i].lvpiao_arriveplane,apl);
        strcpy(ticketn[i].lvpiao_departplane,dpl);
        strcpy(ticketn[i].lvpiao_trans1num,tar);
        strcpy(ticketn[i].lvpiao_trans2num,tdr);

        ticketn[i].pn=atoi(txt1.c_str());
        ticketn[i].lvpiao_arrivedate=atoi(txt2.c_str());
        ticketn[i].lvpiao_departdate=atoi(txt3.c_str());
        ticketn[i].lvpiao_arrivetype=atoi(txt4.c_str());
        ticketn[i].lvpiao_departtype=atoi(txt5.c_str());
        ticketn[i].lvpiao_arrivetime=atoi(txt6.c_str());
        ticketn[i].lvpiao_departtime=atoi(txt7.c_str());

    }
    }
    }

    //-------------------------------------------------------------
    //*************************************************************
    //读取 Trans_Weights 文件获得区域权重信息
    //*************************************************************
```

```
void Inputtransweights(FILE *fp)
{
int i=0;
if(fp==NULL)
{
printf("Sorry,the file is not exist\n");
exit(1);
}
else
{
for(i=0;i<7;i++)
      fscanf(fp,"%d,%d,%d,%d,%d,%d,%d",&graphweight[i][0],&graphweight[i][1],&
graphweight[i][2],&graphweight[i][3],&graphweight[i][4],&graphweight[i][5],&grap
hweight[i][6]);
}
}

//-------------------------------------------------------------
//*************************************************************
//统一标准化时间（19、20、21），已预处理，在此没有调用
//*************************************************************
void Trantimes()
{
int i;
for(i=0;i<FA;i++)
{
//aircraft[i].intime=aircraft[i].intime/100*60+aircraft[i].intime%100;
//aircraft[i].outtime=aircraft[i].outtime/100*60+aircraft[i].outtime%100;
}
}

//-------------------------------------------------------------
//*************************************************************
//初始化生成初始航班
//*************************************************************
void Initialfligenerate()
{
int zq;
for(zq=0;zq<GR;zq++)
{
      Gatereassign(zq);          //生成一组可行解
      Evaluategatebase(zq);    //评价一组可行解
}
Sortgatezhongqun();
```

```
}

//-------------------------------------------------------------
//*****************************************************************
//完成机位分配操作
//*****************************************************************
void Gatereassign(int zq)
{
double psum,pro,u;
double rdnum;
int i,j,bk;
int intim,outtim;
int topen[DD];

memset(topen,0,sizeof(topen));

for(i=0;i<FA;i++)
{

intim=aircraft[i].intime;
outtim=aircraft[i].outtime;
psum=0;

for(j=0;j<DD;j++)
{
      //记录符合条件的登机口（时间间隔、登机口、国内外到达出发类型）
     if((intim>topen[j])&&(aircraft[i].jx==gateh[j].jx)                    &&
((aircraft[i].feiji_arrivetype==gateh[j].tingji_arrivetype)||(gateh[j].tingji_arrivetype==
2))                                                                        &&
((aircraft[i].feiji_departtype==gateh[j].tingji_departtype)||(gateh[j].tingji_departtype=
=2)))
      psum=psum+1;
}
if(psum==0)
{
      gatebbogroup[zq].geti[i]=DD;
}
else
{
      rdnum=(double)(rand()%5000)/5000;
      pro=0;
      for(j=0;j<DD;j++)
      {
            if((intim>topen[j])&&(aircraft[i].jx==gateh[j].jx)              &&
```

```
((aircraft[i].feiji_arrivetype==gateh[j].tingji_arrivetype)||(gateh[j].tingji_arrivetype==
2))                                                                    &&
((aircraft[i].feiji_departtype==gateh[j].tingji_departtype)||(gateh[j].tingji_departtype=
=2))) //
            pro=pro+1/psum;
            if(pro>rdnum)
            {
                  bk=j;
                  break;
            }
      }
      u=(double)rand()/(double)RAND_MAX*0.99;
      gatebbogroup[zq].geti[i]=u+(double)bk;
      if(bk!=DD)//之前是 DD-1
      topen[bk]=outtim+45;//时间间隔

}
//printf("%d,%d,%d\n",aircraft[i].intime,aircraft[i].outtime,gatebbogroup[zq].geti[i]);
}

}

//-------------------------------------------------------------
//*************************************************************
//鲁棒最大值
//*************************************************************
double Rubucontmax()
{
int i,j;
double d;
double dmax=-1000000000;
for(i=0;i<FA-1;i++)
{
      for(j=i+1;j<FA;j++)
      {
      d=(double)aircraft[j].intime-(double)aircraft[i].outtime-2*(double)HCSJ;
      if(d>dmax)
      dmax=d;
      }
}
return dmax;
}

//-------------------------------------------------------------
```

```
//*****************************************************************
//鲁棒最小值
//*****************************************************************
double Rubucontmin()
{
int i,j;
double d;
double dmin=1000000000;
for(i=0;i<FA-1;i++)
{
	for(j=i+1;j<FA;j++)
	{
	d=(double)aircraft[j].intime-(double)aircraft[i].outtime-2*(double)HCSJ;
	if(d<dmin)
	dmin=d;
	}
}
return dmin;
}

//-------------------------------------------------------------
//*****************************************************************
//在一组解中获取停机位分配数量和临时机位数量
//*****************************************************************
int comp(const void *a, const void *b) //  注意 const
{
	return *(int *)a - *(int *)b; //  升序
}

bool IsArrayRepeat(int *arr, int len)
{
	if (!arr || len <= 1)
		return false;


	qsort(arr, len, sizeof(int), comp);
	int count_same=0,count_divr=0;
	for (int i = 0; i < len - 1; i++)
	{
		if (arr[i] == arr[i + 1] && arr[i]==69)//相同元素的判断
			count_same++;
		else if(arr[i] != arr[i + 1] && arr[i]!=69)//种类的判断
			count_divr++;
	}
```

```
	resultallot[0]=count_same+1;
	resultallot[1]=count_divr+1;
	 return false;
}

//-------------------------------------------------------------
//*************************************************************
//评估已生成的一组初始解
//*************************************************************
void Evaluategatebase(int zq)
{
//int i,j,s,n,m;
//int yjw;
//double rbmax=0,rbmin=0,emax=0,emin=0,rbsum=0,rbzs=0,sum=0;
//double ysum=0;
//int recordd[5];
//
//rbmax=Rubucontmax();
//rbmin=Rubucontmin();
//emin=exp((-0.05)*rbmax);
//emax=exp((-0.05)*rbmin);                    //鲁棒性公式
//rbmin=emin;
//rbmax=emax;
//
//sum=0;
//for(j=0;j<DD-1;j++)
//{
//	s=0;
//	for(i=0;i<FA;i++)
//	{
//		if((int)gatebbogroup[zq].geti[i]==j)
//		{
//		recordd[s]=i;
//		s++;
//		}
//	}
//	if(s!=1&&s>0)     //计算每一个机位的鲁棒性值
//	{
//		for(n=0;n<s-1;n++)
//		{
//		for(m=n+1;m<s;m++)
//		{
//                                                rbzs=(double)aircraft[recordd[m]].intime-
(double)aircraft[recordd[n]].outtime-2*(double)HCSJ;
```

```
//              rbsum=rbsum+(exp((-0.05)*rbzs)-rbmin)/(rbmax-rbmin);
//              }
//              }
//      }
//}
//gatebbogroup[zq].rub=rbsum;
//
//yjw=0;
//for(i=0;i<FA;i++)     //计算远机位数量
//{
//      if((int)gatebbogroup[zq].geti[i]==DD-1)
//      yjw++;
//}
//ysum=(double)yjw/(double)FA;
//gatebbogroup[zq].apr=ysum;
////ddddddddddddddddddddddddddddddddddddddddddddddd
//if(gatebbogroup[zq].apr<=Alph)
//gatebbogroup[zq].his=gatebbogroup[zq].rub;
//else
//gatebbogroup[zq].his=gatebbogroup[zq].rub+100*gatebbogroup[zq].apr;

----------------------------------------------问题 1 求解----------------------------------------
-----------
//尽可能多的分配到合适的登机口，最小化被使用登机口的数量目标函数
    int gateacnum[FA]={0};
    memset(gateacnum, 0, sizeof(gateacnum));
    for (int i = 0; i < FA; i++)
        gateacnum[i]=(int)gatebbogroup[zq].geti[i];
    int len = sizeof(gateacnum)/sizeof(int);
    int result=IsArrayRepeat(gateacnum,len);
    double falserate=resultallot[0]/FA;
    double gateuse=resultallot[1]/DD;

    gatebbogroup[zq].his = 1*falserate+0*gateuse;
----------------------------------------------------结束----------------------------------------
------------

----------------------------------------------问题 2 求解----------------------------------------
-----------
//最小化中转旅客的总体流程时间，尽可能多的分配到合适的登机口，最小化被使用登机口的数量目标函数，在同一数量级保持权重分别为 0.5、0.3、0.2.
    int gateacnum[FA]={0};
    memset(gateacnum, 0, sizeof(gateacnum));
    double ysum=0;
```

```
for (int i = 0; i < FA; i++)
    gateacnum[i]=(int)gatebbogroup[zq].geti[i];

int len = sizeof(gateacnum)/sizeof(int);
int result=IsArrayRepeat(gateacnum,len);

double falserate=resultallot[0]/FA;
double gateuse=resultallot[1]/DD;

//总体最短流程时间+捷运时间目标函数
int gate_tr1,gate_tr2;
double all_value_TRANS=0,all_value_CONNECT=0,value_urgent=0;
for (int m = 0; m < NT; m++)
{
    gate_tr1=-1;gate_tr2=-1;
    for (int i = 0; i < FA; i++)
    {
        if(strcmp(aircraft[i].num,ticketn[m].lvpiao_trans1num)==0)
            gate_tr1=(int)gatebbogroup[zq].geti[i];
        if(strcmp(aircraft[i].num,ticketn[m].lvpiao_trans2num)==0)
            gate_tr2=(int)gatebbogroup[zq].geti[i];

        if(gate_tr1!=-1 && gate_tr2!=-1)
            break;//搜索到登机口跳出当前子循环
    }
    //赋初值
    int timecost=0;
    if(ticketn[m].lvpiao_arrivetype==0  &&  ticketn[m].lvpiao_departtype==0)//
到达和出发都是国内
    {
        if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
            timecost=15;
        else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
        {
            if(gate_tr2==69)
                timecost=360;
            else
                timecost=20;
        }
        else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
        {
            if(gate_tr1==69)
                timecost=360;
            else
```

```
                timecost=20;
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=15;
            }
        }
        if(ticketn[m].lvpiao_arrivetype==0  &&  ticketn[m].lvpiao_departtype==1)//
到达是国内，出发是国际
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
                timecost=35;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=40;
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=40;
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=35;
            }
        }
        if(ticketn[m].lvpiao_arrivetype==1  &&  ticketn[m].lvpiao_departtype==0)//
到达是国际，出发是国内
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
                timecost=35;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
```

```
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=40;
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=40;
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=45;
            }
        }
        if(ticketn[m].lvpiao_arrivetype==1  &&  ticketn[m].lvpiao_departtype==1)//
到达和出发都是国际
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
                timecost=20;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=30;
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=30;
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
```

```
                    timecost=20;
                }
            }
            //旅客最短流程时间
            all_value_TRANS+=timecost*ticketn[m].pn;//单个时间消费*乘客数
}
gatebbogroup[zq].his = 0.5*falserate+0.2*gateuse + 0.3*(all_value_TRANS/1000000);
```

----------结束----------

----------问题 3 求解----------

```
//最小化换乘旅客总体紧张度的最小化，尽可能多的分配到合适的登机口，最小化被使用登机口的数量目标函数，在同一数量级保持权重分别为 0.5、0.3、0.2.
    int gateacnum[FA]={0};
    memset(gateacnum, 0, sizeof(gateacnum));
    double ysum=0;
    for (int i = 0; i < FA; i++)
        gateacnum[i]=(int)gatebbogroup[zq].geti[i];

    int len = sizeof(gateacnum)/sizeof(int);
    int result=IsArrayRepeat(gateacnum,len);

    double falserate=resultallot[0]/FA;
    double gateuse=resultallot[1]/DD;

    //ysum = 0.8*falserate+0.2*gateuse;

    //总体最短流程时间+捷运时间目标函数
    int gate_tr1,gate_tr2;
    double all_value_TRANS=0,all_value_CONNECT=0,value_urgent=0;
    for (int m = 0; m < NT; m++)
    {
        gate_tr1=-1;gate_tr2=-1;
        for (int i = 0; i < FA; i++)
        {
            if(strcmp(aircraft[i].num,ticketn[m].lvpiao_trans1num)==0)
                gate_tr1=(int)gatebbogroup[zq].geti[i];
            if(strcmp(aircraft[i].num,ticketn[m].lvpiao_trans2num)==0)
                gate_tr2=(int)gatebbogroup[zq].geti[i];

            if(gate_tr1!=-1 && gate_tr2!=-1)
                break;//搜索到登机口跳出当前子循环
```

```
        }
        //赋初值
        int timecost=0;
        if(ticketn[m].lvpiao_arrivetype==0  &&  ticketn[m].lvpiao_departtype==0)//
到达和出发都是国内
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
                timecost=15;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=20+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=20+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=15;
            }
        }
        if(ticketn[m].lvpiao_arrivetype==0  &&  ticketn[m].lvpiao_departtype==1)//
到达是国内，出发是国际
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
                timecost=35;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=40+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
```

```
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=40+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=35;
            }
        }
        if(ticketn[m].lvpiao_arrivetype==1  &&  ticketn[m].lvpiao_departtype==0)//
到达是国际，出发是国内
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
                timecost=35;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=40+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=40+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=45+16;//加入换乘时间
            }
        }
        if(ticketn[m].lvpiao_arrivetype==1  &&  ticketn[m].lvpiao_departtype==1)//
到达和出发都是国际
        {
            if(gate_tr1<=27 && gate_tr2<=27 && gate_tr1>=0 && gate_tr2>=0)
```

```
                timecost=20;
            else if(gate_tr1<=27 && gate_tr2>27 && gate_tr1>=0)
            {
                if(gate_tr2==69)
                    timecost=360;
                else
                    timecost=30+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2<=27 && gate_tr2>=0)
            {
                if(gate_tr1==69)
                    timecost=360;
                else
                    timecost=30+8;//加入换乘时间
            }
            else if(gate_tr1>27 && gate_tr2>27)
            {
                if(gate_tr1==69 || gate_tr2==69)
                    timecost=360;
                else
                    timecost=20;
            }
        }
        //旅客最短换乘时间
        if(timecost!=0)
        {
            double
compare1=timecost+graphweight[gateh[gate_tr1].dir][gateh[gate_tr2].dir];
            double                          compare2=ticketn[m].lvpiao_departtime-
ticketn[m].lvpiao_arrivetime;
            if(compare1>compare2)
                compare1+=360;      //惩罚机制 6 小时
            else
                compare1=compare1;//不变

            all_value_TRANS+=compare1;        //单个时间消费累积
            all_value_CONNECT+=compare2;     //航班连接时间累积

    value_urgent+=all_value_TRANS/all_value_CONNECT*ticketn[m].pn;//单个紧
张度*人数
        }
    }
    //双目标优化（紧张度和最小化登机口数量）
    gatebbogroup[zq].his = 0.5*falserate+0.2*gateuse + 0.3*(value_urgent/10000);
```

```
}
----------------------------------------------------结束----------------------------------------
-------------

//-------------------------------------------------------------
//*****************************************************************
//种群更新计算
//*****************************************************************
void Sortgatezhongqun()
{
int i,j,k,bk;
double gezhan[FA];
double hiszhan,rubzhan,aprzhan;

for(i=0;i<GR;i++)
{
	bk=i;
	for(j=i+1;j<GR;j++)
	{
		if(gatebbogroup[j].his<gatebbogroup[bk].his)
		bk=j;
	}
	if(bk!=i)
	{
		for(k=0;k<FA;k++)
		{
			gezhan[k]=gatebbogroup[bk].geti[k];
			hiszhan=gatebbogroup[bk].his;
			rubzhan=gatebbogroup[bk].rub;
			aprzhan=gatebbogroup[bk].apr;
		}
		for(k=0;k<FA;k++)
		{
			gatebbogroup[bk].geti[k]=gatebbogroup[i].geti[k];
			gatebbogroup[bk].his=gatebbogroup[i].his;
			gatebbogroup[bk].rub=gatebbogroup[i].rub;
			gatebbogroup[bk].apr=gatebbogroup[i].apr;
		}
		for(k=0;k<FA;k++)
		{
			gatebbogroup[i].geti[k]=gezhan[k];
			gatebbogroup[i].his=hiszhan;
			gatebbogroup[i].apr=aprzhan;
			gatebbogroup[i].rub=rubzhan;
```

```
            }
        }
}
}

//-----------------------------------------------------------
//***************************************************************
//停机位的 BBO 过程
//***************************************************************
void Bbogateprocess()
{
FILE *f3;     //-------------------------------------------------------------------------
int NC=0;
int i,j,selectindex;
double rdnumsum,rdnum,musum,select,check;

struct zuiyou1
{
double getie[FA];
double rube;
double hise;
double apre;
}elite1[2];//每一代最优值存储结构体

struct gtzhongqunjihe zcbbog[GR];

double mu[GR],lambda[GR];   //定义迁入率和迁出率

f3 = fopen("最优化目标值收敛值.txt","a");       //-----------------------------------------
-------------------------------------------
musum=0;
for(i=0;i<GR;i++)
{
      mu[i]=(double)(GR+1-(i+1))/(double)(GR+1);
      lambda[i]=1-mu[i];
      musum=musum+mu[i];
}

while(NC++<=NCga)
{
for(i=0;i<2;i++)   //将最优的几个值存入
{
      for(j=0;j<FA;j++)
      elite1[i].getie[j]=gatebbogroup[i].geti[j];
```

```
        elite1[i].hise=gatebbogroup[i].his;
        elite1[i].apre=gatebbogroup[i].apr;
        elite1[i].rube=gatebbogroup[i].rub;
}

for(i=0;i<GR;i++)     //迁移的过程
{
        for(j=0;j<FA;j++)
        {
                rdnum=(double)(rand()%1000)/1000;
                if(rdnum<lambda[i])
                {
                        rdnum=(double)(rand()%1000)/1000;
                        rdnumsum=rdnum*musum;
                        select=mu[0];
                        selectindex=0;
                        while((rdnumsum>select)&&(selectindex<GR))
                        {
                                selectindex++;
                                select=select+mu[selectindex];
                        }
                        zcbbog[i].geti[j]=Beta*gatebbogroup[i].geti[j]+(1-
Beta)*gatebbogroup[selectindex].geti[j];
                }
                else
                zcbbog[i].geti[j]=gatebbogroup[i].geti[j];
        }
}

for(i=0;i<GR;i++)    //突变操作
{
        for(j=0;j<FA;j++)
        {
                rdnum=(double)(rand()%1000)/1000;
                if(rdnum<MUprob)
                zcbbog[i].geti[j]=(double)rand()/(double)RAND_MAX*((double)DD-1.01);
        }
}

for(i=0;i<GR;i++)
{
        for(j=0;j<FA;j++)
        gatebbogroup[i].geti[j]=zcbbog[i].geti[j];
        gatebbogroup[i].his=zcbbog[i].his;
```

```
}

for(i=0;i<GR;i++)
{
	muavailibit(i);
	Evaluategatebase(i);
}
Sortgatezhongqun();

for(i=0;i<2;i++)   //用上代最优值换本代最差值
{
	for(j=0;j<FA;j++)
	gatebbogroup[GR-2+i].geti[j]=elite1[i].getie[j];
	gatebbogroup[GR-2+i].his=elite1[i].hise;
	gatebbogroup[GR-2+i].rub=elite1[i].rube;
	gatebbogroup[GR-2+i].apr=elite1[i].apre;
}

Sortgatezhongqun();
printf("%0.10lf\n",gatebbogroup[0].his);
+fprintf(f3,"%0.10lf\n",gatebbogroup[0].his);	//------------------------------
}


for(j=0;j<FA;j++)
bestgate[j]=gatebbogroup[0].geti[j];
bestgahis=gatebbogroup[0].his;
bestgarub=gatebbogroup[0].rub;
bestgaapr=gatebbogroup[0].apr;

fclose(f3);//.....................................................
}

//-------------------------------------------------------------
//*****************************************************************
//迁移突变可行性处理
//*****************************************************************
void muavailibit(int zq)
{
double psum,pro;
double rdnum,jw,u;
int i=0,j=0,bk=0,jwt;
int intim,outtim;
double topen[DD];
```

```
double zcgeti[FA]={0};
int check,check1;

memset(topen,0,sizeof(topen));

for(i=0;i<FA;i++)
{

intim=aircraft[i].intime;
outtim=aircraft[i].outtime;
jw=gatebbogroup[zq].geti[i];
jwt=(int)jw;

if((intim>topen[j])&&(aircraft[i].jx==gateh[j].jx)                    &&
((aircraft[i].feiji_arrivetype==gateh[j].tingji_arrivetype)||(gateh[j].tingji_arrivetype==
2))                                                                   &&
((aircraft[i].feiji_departtype==gateh[j].tingji_departtype)||(gateh[j].tingji_departtype=
=2))) //
{
      zcgeti[i]=jw;
      if(jwt!=DD)//之前是 DD-1
      topen[jwt]=outtim;
}
else
{
      psum=0;
      for(j=0;j<DD;j++)
      {
          if((intim>topen[j])&&(aircraft[i].jx==gateh[j].jx)              &&
((aircraft[i].feiji_arrivetype==gateh[j].tingji_arrivetype)||(gateh[j].tingji_arrivetype==
2))                                                                   &&
((aircraft[i].feiji_departtype==gateh[j].tingji_departtype)||(gateh[j].tingji_departtype=
=2))) //
            psum=psum+1;
      }
      if(psum==0)
      {
      zcgeti[i]=DD;
      }
      else
      {
            rdnum=(double)(rand()%5000)/5000;
            pro=0;
            for(j=0;j<DD;j++)
```

```
            {
                if((intim>topen[j])&&(aircraft[i].jx==gateh[j].jx)                    &&
((aircraft[i].feiji_arrivetype==gateh[j].tingji_arrivetype)||(gateh[j].tingji_arrivetype==
2))                                                                                   &&
((aircraft[i].feiji_departtype==gateh[j].tingji_departtype)||(gateh[j].tingji_departtype=
=2))) //
                pro=pro+1/psum;
                if(pro>rdnum)
                {
                    bk=j;
                    break;
                }
            }
            u=(double)rand()/(double)RAND_MAX*0.99;
            zcgeti[i]=(double)bk+u;
            if(bk!=DD)//之前是 DD-1
            topen[bk]=outtim+45;
        }
}
```

附录 2、数据预处理

数据结构定义：

```
double resultallot[2]={0,0};            //停机位分配数量和临时机位数量
int graphweight[7][7]={0};              //区域权重数据

struct tingjiwei //停机位信息结构体
{
    int dir;                    //权重大小
} gateh[69];

struct lvpiao //旅票信息结构体
{
    int lvpiao_arrivetime;              //到达时间
    int lvpiao_departtime;             //出发时间
} ticketn[1703];

typedef     CArray<CStringArray*,CStringArray*>     CMy2Array;
CMy2Array my2Array; //  子数组，待插入到主数组 my2Array,每次都要 new 一下
CStringArray *PucksString = new CStringArray;// Pucks 子数组
CStringArray *TicketsString = new CStringArray;//Tickets 子数组
CStringArray *GatesString = new CStringArray;//Gates 子数组
CStringArray *ShapesString = new CStringArray;//Shapes 子数组
```

```
CString ST_ALLOW[69];//可允许的登机口到达和出发类型
CString ARRIVE_INFO[3]={};
CString DEPART_INFO[3]={};

ofstream DATA_GATES;//GATES 数据
ofstream DATA_AIRCRAFTS;//飞行器数据
ofstream DATA_TICKETS;//旅票数据
ofstream DATA_URGENCY;//紧张度
```

主体程序介绍：

```
//-------------------------------------------------------------
//*************************************************************
//原始数据预处理，读取 Pucks/Tickets/Gates 进行筛选匹配处理成问题求解的 csv 文件。
//*************************************************************
void CFL_PlanningDlg::OnBnClickedPredata()
{
    m_pThread = AfxBeginThread(PreData3,this,THREAD_PRIORITY_NORMAL,0,CREATE_SUSPENDED);//进度条线程
    m_pThread->ResumeThread();
}

UINT CFL_PlanningDlg::PreData1(LPVOID pParam)
{
    CFL_PlanningDlg *pDlg = (CFL_PlanningDlg *)pParam;
    DATA_AIRCRAFTS.open("C:/Users/hl_to/Desktop/Aircraft.csv",ios::app);
    //DATA_GATES.open("C:/Users/Administrator/Desktop/Gates 处理.csv",ios::app);

    CString PKNUM,ARRIVE_DATE,ARRIVE_TIME,ARRIVE_TYPE,PLANE_TYPE,SHAPE_FLAG,DEPART_DATE,DEPART_TIME,DEPART_TYPE;
    CString csv,tin,tout;
    int TIN=0,TOUT=0;
    int Anal_FlightNUM=0;
    for (int count = 0; count < PucksString->GetCount(); count++ )
    {
        ARRIVE_DATE=pDlg->DevideString(PucksString->GetAt(count),',',1);
        ARRIVE_DATE=ARRIVE_DATE.Left(2);

        DEPART_DATE=pDlg->DevideString(PucksString->GetAt(count),',',6);
        DEPART_DATE=DEPART_DATE.Left(2);
```

```
        //筛选 20 日到达或 20 日出发的航班和旅客,其中包括 19 日到达 20 日出发的飞机
        if((ARRIVE_DATE=="19" && DEPART_DATE=="20") || (ARRIVE_DATE=="20"))
        {
            PKNUM=pDlg->DevideString(PucksString->GetAt(count),',',0);
            ARRIVE_TIME=pDlg->DevideString(PucksString->GetAt(count),',',2);
            ARRIVE_TYPE=pDlg->DevideString(PucksString->GetAt(count),',',4);

            PLANE_TYPE=pDlg->DevideString(PucksString->GetAt(count),',',5);
            SHAPE_FLAG=pDlg->GetShapeType(PLANE_TYPE);
            if(SHAPE_FLAG=="W")
                SHAPE_FLAG="1";
            if(SHAPE_FLAG=="N")
                SHAPE_FLAG="0";

            DEPART_TIME=pDlg->DevideString(PucksString->GetAt(count),',',7);
            DEPART_TYPE=pDlg->DevideString(PucksString->GetAt(count),',',9);

            if(ARRIVE_DATE=="19" && DEPART_DATE=="20")
            {
                TIN=pDlg->TimeConversion1(ARRIVE_TIME);
                TOUT=pDlg->TimeConversion1(DEPART_TIME)+1440;
            }
            else if(ARRIVE_DATE=="20" && DEPART_DATE=="20")
            {
                TIN=pDlg->TimeConversion1(ARRIVE_TIME)+1440;
                TOUT=pDlg->TimeConversion1(DEPART_TIME)+1440;
            }
            else if(ARRIVE_DATE=="20" && DEPART_DATE=="21")
            {
                TIN=pDlg->TimeConversion1(ARRIVE_TIME)+1440;
                TOUT=pDlg->TimeConversion1(DEPART_TIME)+2880;
            }

            if(ARRIVE_TYPE=="I")
                ARRIVE_TYPE="1";
            else if(ARRIVE_TYPE=="D")
                ARRIVE_TYPE="0";


            if(DEPART_TYPE=="I")
                DEPART_TYPE="1";
            else if(DEPART_TYPE=="D")
```

```
                    DEPART_TYPE="0";


                tin.Format("%d",TIN);
                tout.Format("%d",TOUT);

    csv=PKNUM+","+tin+","+tout+","+SHAPE_FLAG+","+ARRIVE_TYPE+","+D
EPART_TYPE+","+"\n";
                DATA_AIRCRAFTS.write(csv,csv.GetLength());

                Anal_FlightNUM++;
            }
        }
        DATA_AIRCRAFTS.close();
        AfxMessageBox("");

        return 0;
}

UINT CFL_PlanningDlg::PreData2(LPVOID pParam)
{
        CFL_PlanningDlg *pDlg = (CFL_PlanningDlg *)pParam;
        //DATA_AIRCRAFTS.open("C:/Users/Administrator/Desktop/Aircrafts          处
理.csv",ios::app);
        DATA_GATES.open("C:/Users/Administrator/Desktop/Gate.csv",ios::app);

       CString  GATENUM,SHAPE_TYPE,ARRIVE_TYPE,DEPART_TYPE,REGION;
        CString csv;
        for (int count = 0; count < GatesString->GetCount(); count++ )
        {
            GATENUM=pDlg->DevideString(GatesString->GetAt(count),',',0);
            SHAPE_TYPE=pDlg->DevideString(GatesString->GetAt(count),',',5);
            if(SHAPE_TYPE=="W")
                SHAPE_TYPE="1";
            if(SHAPE_TYPE=="N")
                SHAPE_TYPE="0";

            ARRIVE_TYPE=pDlg->DevideString(GatesString->GetAt(count),',',3);
            DEPART_TYPE=pDlg->DevideString(GatesString->GetAt(count),',',4);
            if(ARRIVE_TYPE=="I")
                ARRIVE_TYPE="1";
            else if(ARRIVE_TYPE=="D")
                ARRIVE_TYPE="0";
            else if(ARRIVE_TYPE=="DI")
```

```
            ARRIVE_TYPE="2";

        if(DEPART_TYPE=="I")
            DEPART_TYPE="1";
        else if(DEPART_TYPE=="D")
            DEPART_TYPE="0";
        else if(DEPART_TYPE=="DI")
            DEPART_TYPE="2";

        //REGION=pDlg->DevideString(GatesString->GetAt(count),',',2);


    csv=GATENUM+","+SHAPE_TYPE+","+ARRIVE_TYPE+","+DEPART_TYP
E+","+"\n";
        DATA_GATES.write(csv,csv.GetLength());
    }
    DATA_GATES.close();
    AfxMessageBox("");

    return 0;
}

UINT CFL_PlanningDlg::PreData3(LPVOID pParam)
{
    CFL_PlanningDlg *pDlg = (CFL_PlanningDlg *)pParam;
    //DATA_AIRCRAFTS.open("C:/Users/hl_to/Desktop/Aircraft.csv",ios::app);
    //DATA_GATES.open("C:/Users/Administrator/Desktop/Gates                处
理.csv",ios::app);
    DATA_TICKETS.open("C:/Users/hl_to/Desktop/Tickets.csv",ios::app);

    CString
TNUM,ARRIVE_DATE,ARRIVE_PLANE,DEPART_DATE,DEPART_PLANE,PAS
SENGERNUM;
    CString csv,tin,tout;

    for (int count = 0; count < TicketsString->GetCount(); count++ )
    {
        ARRIVE_DATE=pDlg->DevideString(TicketsString->GetAt(count),',',3);
        ARRIVE_DATE=ARRIVE_DATE.Left(2);
        DEPART_DATE=pDlg->DevideString(TicketsString->GetAt(count),',',5);
        DEPART_DATE=DEPART_DATE.Left(2);

        //筛选 20 日到达或 20 日出发的航班和旅客,其中包括 19 日到达 20 日出
发的飞机
```

```
        if((ARRIVE_DATE=="19" && DEPART_DATE=="20") || (ARRIVE_DATE=="20"))
        {
            TNUM=pDlg->DevideString(TicketsString->GetAt(count),',',0);

    PASSENGERNUM=pDlg->DevideString(TicketsString->GetAt(count),',',1);

    ARRIVE_PLANE=pDlg->DevideString(TicketsString->GetAt(count),',',2);

    DEPART_PLANE=pDlg->DevideString(TicketsString->GetAt(count),',',4);

            //获取飞机转场记录号、到达类型、出发类型、到达时间、出发时间
            int result1 = pDlg->GetTicketsInfoFromPucks(ARRIVE_DATE,ARRIVE_PLANE,0);
            int result2 = pDlg->GetTicketsInfoFromPucks(DEPART_DATE,DEPART_PLANE,1);

            if(result1==1 && result2==1)//筛选到的旅客车票信息
            {
                CString time1,time2;
                if(ARRIVE_DATE=="19")

    time1.Format("%d",pDlg->TimeConversion1(ARRIVE_INFO[2]));
                else if(ARRIVE_DATE=="20")

    time1.Format("%d",pDlg->TimeConversion1(ARRIVE_INFO[2])+1440);

                if(DEPART_DATE=="20")

    time2.Format("%d",pDlg->TimeConversion1(DEPART_INFO[2])+1440);
                else if(DEPART_DATE=="21")

    time2.Format("%d",pDlg->TimeConversion1(DEPART_INFO[2])+2880);


    csv=TNUM+","+PASSENGERNUM+","+ARRIVE_PLANE+","+ARRIVE_DATE+","+DEPART_PLANE+","+DEPART_DATE+","+ARRIVE_INFO[0]+","+ARRIVE_INFO[1]+","+DEPART_INFO[0]+","+DEPART_INFO[1]+","+time1+","+time2+","+"\n";
                DATA_TICKETS.write(csv,csv.GetLength());
            }
        }
    }
    DATA_TICKETS.close();
```

```
    AfxMessageBox("");

    return 0;
}

int CFL_PlanningDlg::GetTicketsInfoFromPucks(CString str1,CString str2,int FLAG)
{
    CString arriveplane,departplane,arrivedate,departdate;
    if(FLAG==0)
    {
        int num=0;
        for (int count = 0; count < PucksString->GetCount(); count++ )
        {
            arrivedate=DevideString(PucksString->GetAt(count),',',1);
            arrivedate=arrivedate.Left(2);
            arriveplane=DevideString(PucksString->GetAt(count),',',3);

            if(arrivedate==str1 && arriveplane==str2)
            {
                ARRIVE_INFO[0]=DevideString(PucksString->GetAt(count),',',0);
                ARRIVE_INFO[1]=DevideString(PucksString->GetAt(count),',',4);
                if(ARRIVE_INFO[1]=="I")
                    ARRIVE_INFO[1]="1";
                else if(ARRIVE_INFO[1]=="D")
                    ARRIVE_INFO[1]="0";

                ARRIVE_INFO[2]=DevideString(PucksString->GetAt(count),',',2);
                return 1;
            }
        }
        return 0;
    }
    else if(FLAG==1)
    {
        int num=0;
        for (int count = 0; count < PucksString->GetCount(); count++ )
        {
            departdate=DevideString(PucksString->GetAt(count),',',6);
            departdate=departdate.Left(2);
            departplane=DevideString(PucksString->GetAt(count),',',8);

            if(departdate==str1 && departplane==str2)
            {
                DEPART_INFO[0]=DevideString(PucksString->GetAt(count),',',0);
```

```
                DEPART_INFO[1]=DevideString(PucksString->GetAt(count),',',9);
                if(DEPART_INFO[1]=="I")
                    DEPART_INFO[1]="1";
                else if(DEPART_INFO[1]=="D")
                    DEPART_INFO[1]="0";

                DEPART_INFO[2]=DevideString(PucksString->GetAt(count),',',7);
                return 1;
            }
        }
        return 0;
    }
}

//-------------------------------------------------------------
//*************************************************************
//最短换乘时间验证，对比未加惩罚机制和加惩罚机制的对比分析。
//*************************************************************

void CFL_PlanningDlg::OnBnClickedTimecertify2()
{
    m_pThread =
AfxBeginThread(PreDataEnd,this,THREAD_PRIORITY_NORMAL,0,CREATE_SU
SPENDED);//进度条线程
    m_pThread->ResumeThread();
}

UINT CFL_PlanningDlg::PreDataEnd(LPVOID pParam)
{
    CFL_PlanningDlg *pDlg = (CFL_PlanningDlg *)pParam;
    FILE *fp,*fp2,*fp3;
    fp = fopen("trans_weight.csv","at+");
    Inputtransweights(fp);
    fp2 = fopen("gate.csv","at+");
    Inputgatehs(fp2);                    //输入登机口信息
    fp3 = fopen("tickets-Input time.csv","at+");
    Inputickets(fp3);                    //输入旅票信息
    fclose(fp);
    fclose(fp2);
    fclose(fp3);

    CString str=_T("");
    CSVRead read01(_T("ticket.csv"));
```

```
while(read01.file.ReadString(str))
{
    TicketsString->Add(str);
}

str=_T("");
CSVRead read02(_T("aircraft.csv"));

while(read02.file.ReadString(str))
{
    PucksString->Add(str);
}

int a[303]={0};
DATA_URGENCY.open("C:/Users/hl_to/Desktop/Urgency.csv",ios::app);
//第一问结果
//CString csv="61 -> 36 -> 17 -> 68 -> 2 -> 55 -> 3 -> 23 -> 60 -> 22 -> 21 -> 37
-> 63 -> 68 -> 12 -> 19 -> 38 -> 29 -> 50 -> 6 -> 39 -> 54 -> 4 -> 34 -> 20 -> 5 -> 69
-> 43 -> 47 -> 48 -> 49 -> 41 -> 13 -> 69 -> 7 -> 28 -> 15 -> 46 -> 14 -> 69 -> 35 -> 8
-> 69 -> 9 -> 44 -> 69 -> 69 -> 52 -> 11 -> 53 -> 42 -> 18 -> 51 -> 16 -> 30 -> 31 ->
45 -> 69 -> 24 -> 10 -> 33 -> 32 -> 67 -> 39 -> 69 -> 22 -> 26 -> 1 -> 38 -> 59 -> 65
-> 68 -> 69 -> 27 -> 68 -> 69 -> 68 -> 27 -> 40 -> 3 -> 69 -> 60 -> 69 -> 0 -> 58 -> 69
-> 15 -> 69 -> 69 -> 37 -> 13 -> 11 -> 44 -> 28 -> 20 -> 45 -> 23 -> 12 -> 69 -> 14 ->
22 -> 50 -> 10 -> 54 -> 49 -> 69 -> 33 -> 47 -> 8 -> 9 -> 48 -> 31 -> 69 -> 35 -> 46 ->
69 -> 18 -> 69 -> 52 -> 69 -> 69 -> 53 -> 38 -> 0 -> 6 -> 69 -> 16 -> 69 -> 34 -> 37 ->
7 -> 69 -> 21 -> 20 -> 41 -> 15 -> 40 -> 39 -> 9 -> 47 -> 8 -> 10 -> 22 -> 6 -> 32 -> 48
-> 11 -> 21 -> 19 -> 0 -> 45 -> 69 -> 7 -> 15 -> 38 -> 54 -> 46 -> 69 -> 69 -> 43 -> 52
-> 55 -> 29 -> 20 -> 40 -> 69 -> 27 -> 9 -> 35 -> 53 -> 24 -> 34 -> 12 -> 6 -> 11 -> 19
-> 21 -> 31 -> 4 -> 8 -> 0 -> 69 -> 51 -> 69 -> 69 -> 50 -> 69 -> 42 -> 33 -> 69 -> 68
-> 7 -> 69 -> 22 -> 69 -> 2 -> 38 -> 44 -> 5 -> 69 -> 26 -> 13 -> 27 -> 69 -> 52 -> 69
-> 69 -> 35 -> 54 -> 68 -> 55 -> 64 -> 19 -> 21 -> 23 -> 8 -> 7 -> 16 -> 49 -> 9 -> 6 ->
69 -> 44 -> 38 -> 51 -> 59 -> 48 -> 20 -> 69 -> 32 -> 69 -> 13 -> 61 -> 0 -> 69 -> 60
-> 69 -> 69 -> 29 -> 1 -> 69 -> 69 -> 12 -> 36 -> 3 -> 37 -> 69 -> 46 -> 39 -> 30 -> 17
-> 43 -> 18 -> 40 -> 47 -> 8 -> 31 -> 55 -> 21 -> 20 -> 6 -> 53 -> 22 -> 11 -> 19 -> 44
-> 69 -> 69 -> 42 -> 15 -> 14 -> 38 -> 33 -> 58 -> 16 -> 41 -> 54 -> 7 -> 29 -> 17 ->
46 -> 35 -> 69 -> 69 -> 10 -> 48 -> 12 -> 45 -> 21 -> 69 -> 32 -> 36 -> 49 -> 13 -> 19
-> 55 -> 6 -> 11 -> 69 -> 37 -> 28 -> 15 -> 53";
//第二问结果
//CString csv="2 -> 11 -> 6 -> 24 -> 23 -> 36 -> 4 -> 5 -> 62 -> 22 -> 21 -> 50 ->
3 -> 58 -> 44 -> 20 -> 0 -> 48 -> 31 -> 19 -> 39 -> 7 -> 69 -> 14 -> 69 -> 67 -> 69 ->
37 -> 12 -> 18 -> 43 -> 10 -> 35 -> 69 -> 8 -> 51 -> 16 -> 53 -> 45 -> 69 -> 34 -> 69
-> 69 -> 28 -> 13 -> 69 -> 69 -> 9 -> 42 -> 55 -> 46 -> 32 -> 30 -> 17 -> 49 -> 29 ->
33 -> 69 -> 69 -> 15 -> 47 -> 52 -> 58 -> 40 -> 69 -> 22 -> 25 -> 4 -> 0 -> 63 -> 27 ->
60 -> 69 -> 62 -> 26 -> 69 -> 64 -> 65 -> 38 -> 5 -> 69 -> 66 -> 69 -> 39 -> 3 -> 69 ->
```

54 -> 69 -> 69 -> 13 -> 51 -> 7 -> 47 -> 41 -> 69 -> 33 -> 22 -> 35 -> 69 -> 31 -> 69 -> 15 -> 45 -> 44 -> 12 -> 69 -> 16 -> 50 -> 69 -> 28 -> 14 -> 9 -> 69 -> 42 -> 18 -> 69 -> 53 -> 69 -> 52 -> 69 -> 69 -> 32 -> 0 -> 39 -> 19 -> 69 -> 34 -> 69 -> 13 -> 7 -> 8 -> 69 -> 21 -> 69 -> 55 -> 43 -> 40 -> 38 -> 15 -> 35 -> 69 -> 51 -> 22 -> 19 -> 44 -> 21 -> 31 -> 7 -> 20 -> 39 -> 14 -> 69 -> 8 -> 18 -> 0 -> 53 -> 16 -> 69 -> 69 -> 35 -> 29 -> 32 -> 48 -> 69 -> 40 -> 69 -> 25 -> 52 -> 17 -> 42 -> 24 -> 50 -> 33 -> 21 -> 15 -> 19 -> 69 -> 31 -> 65 -> 7 -> 39 -> 20 -> 9 -> 69 -> 69 -> 45 -> 69 -> 30 -> 28 -> 69 -> 66 -> 8 -> 69 -> 4 -> 69 -> 5 -> 0 -> 52 -> 26 -> 69 -> 23 -> 17 -> 63 -> 69 -> 15 -> 69 -> 69 -> 42 -> 7 -> 64 -> 13 -> 67 -> 19 -> 6 -> 22 -> 21 -> 8 -> 11 -> 20 -> 12 -> 0 -> 69 -> 10 -> 69 -> 35 -> 3 -> 16 -> 69 -> 69 -> 32 -> 69 -> 9 -> 1 -> 38 -> 69 -> 2 -> 69 -> 69 -> 28 -> 58 -> 69 -> 69 -> 30 -> 17 -> 25 -> 48 -> 69 -> 43 -> 39 -> 51 -> 49 -> 18 -> 33 -> 21 -> 13 -> 69 -> 46 -> 37 -> 44 -> 19 -> 10 -> 52 -> 4 -> 6 -> 69 -> 11 -> 69 -> 69 -> 35 -> 54 -> 50 -> 0 -> 53 -> 22 -> 36 -> 34 -> 7 -> 55 -> 47 -> 16 -> 31 -> 43 -> 69 -> 40 -> 28 -> 17 -> 42 -> 32 -> 20 -> 8 -> 30 -> 41 -> 48 -> 14 -> 6 -> 35 -> 69 -> 52 -> 69 -> 54 -> 7 -> 9 -> 10";

//第三问结果

//CString csv="58 -> 21 -> 10 -> 59 -> 27 -> 45 -> 25 -> 22 -> 2 -> 64 -> 6 -> 8 -> 3 -> 1 -> 28 -> 19 -> 0 -> 35 -> 20 -> 69 -> 40 -> 54 -> 23 -> 13 -> 69 -> 5 -> 69 -> 44 -> 53 -> 16 -> 12 -> 33 -> 32 -> 69 -> 7 -> 47 -> 51 -> 36 -> 18 -> 69 -> 17 -> 69 -> 69 -> 46 -> 9 -> 69 -> 69 -> 49 -> 42 -> 48 -> 52 -> 50 -> 41 -> 31 -> 30 -> 11 -> 14 -> 69 -> 24 -> 55 -> 29 -> 34 -> 25 -> 38 -> 69 -> 22 -> 1 -> 26 -> 39 -> 62 -> 60 -> 67 -> 69 -> 61 -> 68 -> 69 -> 2 -> 3 -> 0 -> 63 -> 69 -> 4 -> 69 -> 40 -> 64 -> 69 -> 29 -> 69 -> 69 -> 37 -> 14 -> 12 -> 8 -> 9 -> 69 -> 42 -> 22 -> 43 -> 20 -> 54 -> 69 -> 51 -> 32 -> 53 -> 16 -> 69 -> 11 -> 28 -> 69 -> 15 -> 46 -> 55 -> 69 -> 17 -> 47 -> 69 -> 34 -> 69 -> 13 -> 69 -> 69 -> 18 -> 0 -> 40 -> 69 -> 69 -> 36 -> 69 -> 48 -> 49 -> 7 -> 69 -> 6 -> 69 -> 14 -> 12 -> 39 -> 38 -> 11 -> 16 -> 8 -> 54 -> 22 -> 19 -> 53 -> 55 -> 32 -> 7 -> 20 -> 0 -> 48 -> 6 -> 69 -> 34 -> 40 -> 50 -> 47 -> 69 -> 69 -> 33 -> 44 -> 31 -> 12 -> 69 -> 39 -> 69 -> 23 -> 15 -> 18 -> 42 -> 61 -> 17 -> 13 -> 8 -> 55 -> 20 -> 69 -> 43 -> 63 -> 7 -> 0 -> 19 -> 32 -> 6 -> 69 -> 51 -> 69 -> 52 -> 29 -> 69 -> 4 -> 69 -> 69 -> 22 -> 69 -> 58 -> 40 -> 17 -> 1 -> 69 -> 65 -> 49 -> 3 -> 21 -> 53 -> 69 -> 69 -> 31 -> 54 -> 24 -> 16 -> 25 -> 20 -> 7 -> 5 -> 6 -> 8 -> 43 -> 12 -> 37 -> 40 -> 69 -> 15 -> 69 -> 32 -> 62 -> 41 -> 19 -> 69 -> 21 -> 69 -> 44 -> 2 -> 38 -> 69 -> 64 -> 23 -> 69 -> 36 -> 26 -> 69 -> 69 -> 35 -> 14 -> 59 -> 47 -> 69 -> 17 -> 0 -> 42 -> 11 -> 51 -> 20 -> 6 -> 43 -> 69 -> 30 -> 33 -> 50 -> 19 -> 52 -> 45 -> 63 -> 32 -> 69 -> 49 -> 7 -> 69 -> 16 -> 29 -> 34 -> 39 -> 55 -> 68 -> 15 -> 28 -> 46 -> 13 -> 36 -> 10 -> 17 -> 54 -> 69 -> 40 -> 18 -> 50 -> 12 -> 48 -> 21 -> 8 -> 44 -> 41 -> 31 -> 9 -> 69 -> 32 -> 69 -> 11 -> 69 -> 45 -> 16 -> 13 -> 35";

//第二问结果惩罚机制

//CString csv="65 -> 17 -> 44 -> 24 -> 63 -> 53 -> 60 -> 5 -> 3 -> 67 -> 19 -> 34 -> 2 -> 58 -> 11 -> 6 -> 40 -> 49 -> 31 -> 20 -> 39 -> 29 -> 23 -> 10 -> 21 -> 26 -> 69 -> 30 -> 36 -> 46 -> 35 -> 50 -> 28 -> 69 -> 7 -> 18 -> 12 -> 9 -> 55 -> 69 -> 16 -> 8 -> 69 -> 32 -> 43 -> 69 -> 69 -> 33 -> 45 -> 42 -> 54 -> 52 -> 41 -> 15 -> 14 -> 51 -> 47 -> 69 -> 22 -> 13 -> 37 -> 48 -> 4 -> 38 -> 69 -> 5 -> 60 -> 2 -> 39 -> 64 -> 59 ->

1 -> 69 -> 66 -> 67 -> 69 -> 25 -> 58 -> 0 -> 3 -> 69 -> 62 -> 69 -> 40 -> 68 -> 69 -> 37 -> 69 -> 69 -> 28 -> 35 -> 29 -> 18 -> 34 -> 21 -> 55 -> 4 -> 47 -> 69 -> 31 -> 5 -> 32 -> 45 -> 36 -> 46 -> 69 -> 11 -> 13 -> 8 -> 51 -> 43 -> 10 -> 69 -> 52 -> 9 -> 69 -> 48 -> 69 -> 12 -> 69 -> 69 -> 33 -> 0 -> 39 -> 20 -> 69 -> 28 -> 69 -> 15 -> 16 -> 7 -> 69 -> 21 -> 19 -> 47 -> 55 -> 38 -> 40 -> 33 -> 42 -> 6 -> 10 -> 23 -> 20 -> 46 -> 45 -> 37 -> 21 -> 69 -> 39 -> 31 -> 69 -> 7 -> 35 -> 0 -> 50 -> 28 -> 8 -> 69 -> 29 -> 36 -> 18 -> 33 -> 19 -> 38 -> 6 -> 26 -> 12 -> 48 -> 13 -> 2 -> 54 -> 45 -> 21 -> 49 -> 20 -> 69 -> 43 -> 66 -> 69 -> 39 -> 69 -> 37 -> 69 -> 69 -> 42 -> 69 -> 32 -> 51 -> 69 -> 22 -> 7 -> 69 -> 23 -> 69 -> 24 -> 0 -> 31 -> 62 -> 69 -> 60 -> 9 -> 3 -> 6 -> 18 -> 69 -> 69 -> 15 -> 14 -> 64 -> 55 -> 26 -> 20 -> 21 -> 5 -> 8 -> 7 -> 37 -> 13 -> 30 -> 0 -> 69 -> 31 -> 69 -> 45 -> 58 -> 35 -> 19 -> 69 -> 53 -> 6 -> 36 -> 67 -> 39 -> 69 -> 61 -> 4 -> 69 -> 33 -> 25 -> 69 -> 69 -> 11 -> 51 -> 63 -> 50 -> 69 -> 54 -> 40 -> 16 -> 44 -> 47 -> 9 -> 38 -> 29 -> 8 -> 48 -> 28 -> 17 -> 19 -> 20 -> 10 -> 27 -> 43 -> 21 -> 45 -> 69 -> 69 -> 37 -> 49 -> 55 -> 0 -> 31 -> 66 -> 42 -> 41 -> 12 -> 15 -> 11 -> 33 -> 7 -> 54 -> 6 -> 69 -> 50 -> 51 -> 17 -> 32 -> 69 -> 21 -> 52 -> 14 -> 28 -> 43 -> 20 -> 46 -> 69 -> 35 -> 69 -> 44 -> 45 -> 12 -> 13";

//第三问结果惩罚机制

CString csv="64 -> 42 -> 51 -> 25 -> 23 -> 16 -> 58 -> 22 -> 27 -> 62 -> 6 -> 18 -> 3 -> 4 -> 45 -> 19 -> 40 -> 31 -> 55 -> 20 -> 0 -> 36 -> 5 -> 14 -> 21 -> 66 -> 69 -> 46 -> 28 -> 34 -> 35 -> 30 -> 17 -> 69 -> 7 -> 49 -> 29 -> 47 -> 44 -> 69 -> 11 -> 8 -> 69 -> 12 -> 48 -> 69 -> 69 -> 33 -> 52 -> 13 -> 32 -> 9 -> 43 -> 50 -> 53 -> 37 -> 41 -> 69 -> 24 -> 15 -> 54 -> 10 -> 22 -> 40 -> 69 -> 4 -> 67 -> 59 -> 0 -> 2 -> 63 -> 60 -> 69 -> 62 -> 65 -> 69 -> 3 -> 61 -> 38 -> 1 -> 69 -> 27 -> 69 -> 39 -> 68 -> 69 -> 54 -> 69 -> 69 -> 52 -> 35 -> 49 -> 18 -> 17 -> 21 -> 44 -> 4 -> 29 -> 69 -> 41 -> 22 -> 45 -> 34 -> 15 -> 28 -> 69 -> 36 -> 55 -> 8 -> 12 -> 14 -> 33 -> 69 -> 48 -> 9 -> 69 -> 37 -> 69 -> 11 -> 69 -> 69 -> 47 -> 38 -> 39 -> 20 -> 69 -> 13 -> 69 -> 46 -> 54 -> 7 -> 69 -> 6 -> 21 -> 30 -> 29 -> 0 -> 40 -> 12 -> 10 -> 8 -> 9 -> 22 -> 19 -> 28 -> 33 -> 14 -> 6 -> 20 -> 39 -> 29 -> 69 -> 7 -> 13 -> 38 -> 46 -> 52 -> 21 -> 69 -> 15 -> 45 -> 47 -> 54 -> 69 -> 0 -> 69 -> 59 -> 48 -> 16 -> 10 -> 65 -> 37 -> 43 -> 8 -> 44 -> 20 -> 6 -> 35 -> 5 -> 69 -> 39 -> 19 -> 55 -> 69 -> 69 -> 50 -> 69 -> 12 -> 52 -> 69 -> 25 -> 7 -> 69 -> 23 -> 69 -> 2 -> 38 -> 41 -> 66 -> 69 -> 62 -> 18 -> 61 -> 21 -> 49 -> 69 -> 69 -> 54 -> 53 -> 59 -> 44 -> 1 -> 19 -> 6 -> 24 -> 8 -> 7 -> 11 -> 16 -> 31 -> 38 -> 69 -> 36 -> 69 -> 10 -> 22 -> 33 -> 20 -> 69 -> 46 -> 21 -> 52 -> 3 -> 39 -> 69 -> 67 -> 4 -> 69 -> 14 -> 58 -> 69 -> 69 -> 35 -> 18 -> 27 -> 55 -> 69 -> 47 -> 40 -> 30 -> 34 -> 41 -> 48 -> 0 -> 11 -> 8 -> 12 -> 43 -> 51 -> 19 -> 45 -> 44 -> 5 -> 28 -> 20 -> 50 -> 6 -> 69 -> 32 -> 36 -> 29 -> 38 -> 10 -> 2 -> 42 -> 15 -> 13 -> 53 -> 33 -> 9 -> 54 -> 34 -> 21 -> 69 -> 35 -> 16 -> 37 -> 46 -> 69 -> 7 -> 47 -> 52 -> 55 -> 28 -> 20 -> 44 -> 69 -> 14 -> 69 -> 36 -> 32 -> 18 -> 13";

```
for (int count = 0; count < 303; count++ )
    a[count]=atoi(pDlg->DevideString(csv,'->',count));

int t1=a[3];
int t2=a[302];
```

```
    //
    CString NumInfo,arriveplane,departplane;
    int arrivetype=0,departtype=0,passengernum=0,num=0;
    double
all_value_TRANS=0,all_value_CONNECT=0,value_urgent=0,passennum=0;
    for(int count=290;count<TicketsString->GetCount();count++)
    {
        arriveplane=pDlg->DevideString(TicketsString->GetAt(count),',',6);
        int gate_tr1=pDlg->GetNumFromAircraft(arriveplane);
        departplane=pDlg->DevideString(TicketsString->GetAt(count),',',8);
        int gate_tr2=pDlg->GetNumFromAircraft(departplane);

        if(gate_tr1!=-1 && gate_tr2!=-1)//旅票匹配好
        {
            arrivetype=atoi(pDlg->DevideString(TicketsString->GetAt(count),',',7));
            departtype=atoi(pDlg->DevideString(TicketsString->GetAt(count),',',9));

    passengernum=atoi(pDlg->DevideString(TicketsString->GetAt(count),',',1));

            int timecost=0;
            if(arrivetype==0 && departtype==0)//到达和出发都是国内
            {
                if(a[gate_tr1]<=27 && a[gate_tr2]<=27 && a[gate_tr1]>=0 &&
a[gate_tr2]>=0)
                    timecost=15;
                else if(a[gate_tr1]<=27 && a[gate_tr2]>27 && a[gate_tr1]>=0)
                {
                    if(a[gate_tr2]==69)
                    {

                        timecost=0;
                    }
                    else
                        timecost=20+8;
                }
                else if(a[gate_tr1]>27 && a[gate_tr2]<=27 && a[gate_tr2]>=0)
                {
                    if(a[gate_tr1]==69)
                    {

                        timecost=0;
                    }
                    else
                        timecost=20+8;
```

```
                }
                else if(a[gate_tr1]>27 && a[gate_tr2]>27)
                {
                    if(a[gate_tr1]==69 || a[gate_tr2]==69)
                    {

                        timecost=0;
                    }
                    else
                        timecost=15;
                }
            }
            if(arrivetype==0 && departtype==1)//到达是国内，出发是国际
            {
                if(a[gate_tr1]<=27 && a[gate_tr2]<=27 && a[gate_tr1]>=0 &&
a[gate_tr2]>=0)
                    timecost=35;
                else if(a[gate_tr1]<=27 && a[gate_tr2]>27 && a[gate_tr1]>=0)
                {
                    if(a[gate_tr2]==69)
                    {

                        timecost=0;
                    }
                    else
                        timecost=40+8;
                }
                else if(a[gate_tr1]>27 && a[gate_tr2]<=27 && a[gate_tr2]>=0)
                {
                    if(a[gate_tr1]==69)
                    {

                        timecost=0;
                    }
                    else
                        timecost=40+8;
                }
                else if(a[gate_tr1]>27 && a[gate_tr2]>27)
                {
                    if(a[gate_tr1]==69 || a[gate_tr2]==69)
                    {

                        timecost=0;
                    }
```

```
                else
                    timecost=35;
            }
        }
        if(arrivetype==1 && departtype==0)//到达是国际，出发是国内
        {
            if(a[gate_tr1]<=27 && a[gate_tr2]<=27 && a[gate_tr1]>=0 &&
a[gate_tr2]>=0)
                timecost=35;
            else if(a[gate_tr1]<=27 && a[gate_tr2]>27 && a[gate_tr1]>=0)
            {
                if(a[gate_tr2]==69)
                {

                    timecost=0;
                }
                else
                    timecost=40+8;
            }
            else if(a[gate_tr1]>27 && a[gate_tr2]<=27 && a[gate_tr2]>=0)
            {
                if(a[gate_tr1]==69)
                {

                    timecost=0;
                }
                else
                    timecost=40+8;
            }
            else if(a[gate_tr1]>27 && a[gate_tr2]>27)
            {
                if(a[gate_tr1]==69 || a[gate_tr2]==69)
                {

                    timecost=0;
                }
                else
                    timecost=45+16;
            }
        }
        if(arrivetype==1 && departtype==1)//到达和出发都是国际
        {
            if(a[gate_tr1]<=27 && a[gate_tr2]<=27 && a[gate_tr1]>=0 &&
a[gate_tr2]>=0)
```

```
                timecost=20;
            else if(a[gate_tr1]<=27 && a[gate_tr2]>27 && a[gate_tr1]>=0)
            {
                if(a[gate_tr2]==69)
                {
                    timecost=0;

                }
                else
                    timecost=30+8;
            }
            else if(a[gate_tr1]>27 && a[gate_tr2]<=27 && a[gate_tr2]>=0)
            {
                if(a[gate_tr1]==69)
                {

                    timecost=0;
                }
                else
                    timecost=30+8;
            }
            else if(a[gate_tr1]>27 && a[gate_tr2]>27)
            {
                if(a[gate_tr1]==69 || a[gate_tr2]==69)
                {

                    timecost=0;
                }
                else
                    timecost=20;
            }
        }
        if(timecost!=0)
        {
            int code1=a[gate_tr1];
            int code2=a[gate_tr2];
            double
compare1=timecost+graphweight[gateh[code1].dir][gateh[code2].dir];
            double                  compare2=ticketn[count].lvpiao_departtime-
ticketn[count].lvpiao_arrivetime;
            if(compare1>compare2)
                compare1+=360;      //惩罚机制 6 小时
            else
                compare1=compare1;//不变
```

```
                    all_value_TRANS+=compare1;        //单个时间消费累积
                    all_value_CONNECT+=compare2;     //航班连接时间累积

    value_urgent+=all_value_TRANS/all_value_CONNECT*passengernum;// 单 个紧张度*人数
                    passennum+=passengernum;


                    for(int k=0;k<passengernum;k++)
                    {
                        CString csvinput,csvinput1="",csvinput2="",csvinput3="";
                        csvinput1.Format("%.3f",compare1/compare2);
                        csvinput2.Format("%.3f",compare1);
                        csvinput3.Format("%.3f",compare2);
                        csvinput=csvinput1+","+csvinput2+","+csvinput3+","+"\n";
                        DATA_URGENCY.write(csvinput,csvinput.GetLength());
                    }

                }
                //else
                    //passennum+=passengernum;
            }
            //

            NumInfo.Format("%d",count);
            pDlg->SetDlgItemTextA(IDC_EDIT1,NumInfo);
        }
        DATA_URGENCY.close();
        NumInfo.Format("%.2f,%.2f,%.2f",value_urgent,all_value_TRANS,passennum);
        AfxMessageBox(NumInfo);

        return 0;
}

}

for(i=0;i<FA;i++)
gatebbogroup[zq].geti[i]=zcgeti[i];
}
```