“华为杯”第十五届中国研究生

数学建模竞赛

学　　校	东南大学
参赛队号	18102860509
队员姓名	1.曾嘉鑫
	2.李文超
	3.孙叶青

“华为杯”第十五届中国研究生

数学建模竞赛

题 目　　航班-登机口优化分配问题

摘　　要：

为了应对旅游业的快速发展,某航空公司需要在现有的航站楼 T 以外，增设一个卫星厅 S。引入 S 后，虽然可以缓解原有航站楼登机口不足的压力，但对中转旅客的航班衔接显然具有一定的负面影响。本文通过建立数学模型，考虑各种可能被影响的因素，综合分析了航班-登机口的优化分配问题。

针对问题一，我们首先根据固有属性对航班和登机口分别进行分类，以**集合**元素为自变量构造函数来讨论，十分简明易懂。我们以登机口的“**通用度**”为指标，创造性地将分配过程从逻辑上分为**三轮**，来保证在最大化分配的基础上尽可能地节约被使用的登机口资源。针对单个登机口的选取问题，我们参考操作系统的“**高优先权优先调度算法**”提出了“**非抢占式动态优先级算法**”，选取在等待降落的飞机到达时间之前最晚释放的登机口，来保证所有开启的登机口的高利用率。若已开启的登机口都停有飞机时，才会开启新的登机口。最终 303 架飞机中成功分配了 249 架，仅 54 架飞机需要停放在简易临时机位上，成功率达到了 82.18%。69 个登机口中仍有 4 个未曾投入使用，使用率为 94.20%。

针对问题二，类似于问题一中**分类讨论**的思想，我们根据到达类型和出发类型也对有效旅客信息进行分类。我们借助 **0-1 规划**的思想，引入了两个 0-1 变量 x 和 y 来刻画某类记录的总流程时间。此外，又加上限制条件：乘坐同一航班到达的旅客从同一登机口下飞机以及同一航班出发的旅客从同一登机口上飞机。利用 Lingo 软件得到了理想化（所有旅客均中转成功）的最优解 $T_{流}=87575\,\min$，再结合第一问的实际情况，调整参数，依据“**非抢占式动态优先级算法**”进行优化，这时 2751 名有效旅客中有 2268 名换乘成功，483 名换乘失败，换乘成功率为 82.44%。最终得到实际的较优解 $T_{流}=76920\,\min$。

针对问题三，考虑到对于特定的一条旅客记录，其换乘时间由中转方案唯一确定，因此我们构造了 28 阶**换乘耗时矩阵** K，其元素就代表每种中转方案的换乘时间；我们又构造了 28 阶**门选矩阵** W，它仅有一个元素为 1，其他都为 0，表

示该旅客记录选中了相应的中转方案。我们又定义了一个**取出函数** *C*（*W*）来得到该条旅客记录的中转时间。由此，我们可以构建出优化模型，又加上限制条件：乘坐同一航班到达的旅客从同一登机口下飞机以及同一航班出发的旅客从同一登机口上飞机。利用 Lingo 软件得到了理想化的最优解 *Tensity* = 668.73。考虑到现实因素需要加以调整，调整后可以达到的实际较优解为 *Tensity* = 805.82。

关键词： 登机口调度，优化调度，机场规划，旅客满意度

目录

一、问题重述

1.1 问题背景

随着人民生活水平的提高，旅游业快速发展，某航空公司在某机场的现有航站楼 T 的旅客流量已达饱和状态，现正增设卫星厅 S。引入卫星厅后，虽然可以缓解原有航站楼 T 登机口不足的压力，但对中转旅客的航班衔接显然具有一定的负面影响。

飞机在机场廊桥（登机口）的一次停靠通常由一对航班（到达航班和出发航班，也叫“转场”）来标识，如下图所示。航班-登机口分配就是把这样的航班对分配到合适的登机口。所谓的中转旅客就是从到达航班换乘到由同一架或不同架飞机执行的出发航班的旅客。

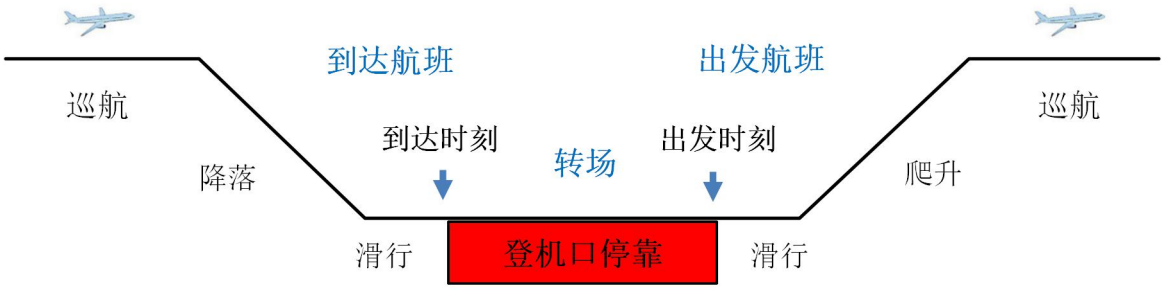

图 **1** 航班及其停靠示意图

单纯的航班-登机口的优化分配问题已经被很好地解决，Sabre Airline Solutions® 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。但在优化分配登机口的同时考虑最小化旅客行走时间，学界研究有限，市场上产品一般也不具备此一功能。

1.2 细节描述

1.2.1 机场布局

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

T 和 S 之间有捷运线相通，可以快速往来运送国内、国际旅客。假定旅客无需等待，随时可以发车，单程一次需要 8 分钟。

航站楼 T 有 28 个登机口，卫星厅 S 有 41 个登机口。其配置见数据部分。

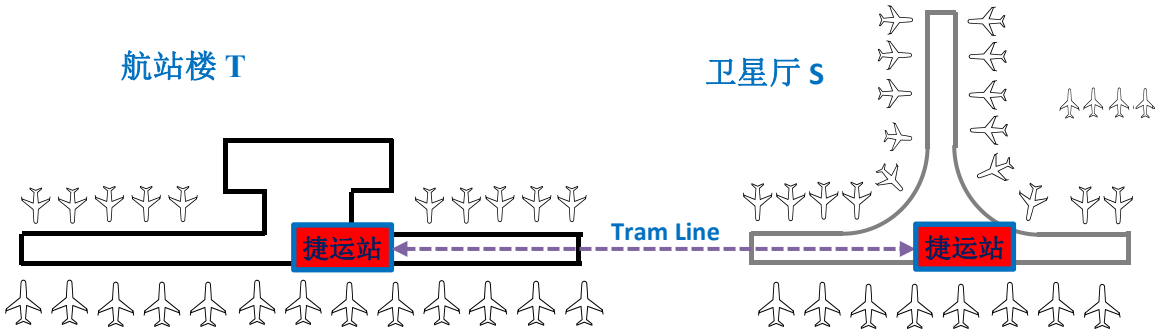

图 2 卫星厅 S 相对于航站楼 T 示意图

1.2.2 登机口分配

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

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

注：本题数据中使用到的宽窄飞机型号分别有：

- 宽体机（Wide-body）：332, 333, 33E, 33H, 33L, 773
- 窄体机（Narrow-body）：319, 320, 321, 323, 325, 738, 73A, 73E, 73H, 73L。

1.2.3 旅客流程

旅客流程可以按始发旅客、终到旅客和中转旅客分类规范。但由于新建卫星厅对始发旅客和终到旅客影响甚微，故不在本赛题研究范围内。中转旅客从前一航班的到达至后一航班的出发之间的流程，按国内（D）和国际（I）、航站楼（T）和卫星厅（S）组合成 16 种不同的场景。这些场景的最短流程时间和捷运乘坐次数由下表给出，其中每一格的第一个数是最短流程时间（分钟），第二个数是捷运乘坐次数。捷运时间和旅客行走时间不计入最短流程时间。

例：某旅客乘坐国际航班到达卫星厅 S，然后换乘国内航班从卫星厅 S 登机，其最短流程时间是 45 分钟，需乘坐捷运 2 次（因为需要去航站楼 T 办理入境手续）。

表 1 旅客流程图

出发 / 到达		国内出发（D）		国际出发（I）	
		航站楼 T	卫星厅 S	航站楼 T	卫星厅 S
国内到达（D）	航站楼 T	15/0	20/1	35/0	40/1
	卫星厅 S	20/1	15/0	40/1	35/0
国际到达（I）	航站楼 T	35/0	40/1	20/0	30/1
	卫星厅 S	40/1	45/2	30/1	20/0

1.3 待解决的问题

真实的航班-登机口分配调度方案可能受到各种制约因素的影响，但作为评估，我们可以不妨假设其是在一定约束条件下的优化方案。本题要求就以下三种情形对航班-登机口分配问题建立数学模型，优化分配登机口，分析中转旅客的换乘紧张程度，为航空公司航班规划的调整提供参考依据。仅考虑航站楼 T 和卫星厅 S 同时使用的情形。

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

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

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

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

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

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

其中，行走时间由下列表格查找（单位：分钟。捷运乘坐时间需另行计算）

表 2 行走时间表

登机口区域	T-North	T-Center	T-South	S-North	S-Center	S-South	S-East
T-North	10	15	20	25	20	25	25
T-Center		10	15	20	15	20	20
T-South			10	25	20	25	25
S-North				10	15	20	20
S-Center					10	15	15
S-South						10	20
S-East							10

二、问题分析

问题一分析：本题只考虑航班-登机口分配，需要匹配的条件有到达类型、出发类型和机型，因此不妨根据这 3 个固有属性将航班分为 $2*2*2=8$ 种，将登机口分为 10 种（由表格数据易得）。因此第一步，我们只需考虑针对某一种类型的航班选出合适的那类登机口，缩小选择范围；第二步，对于每一架等待降落

的飞机，优先选取已经开启的登机口（已经有飞机转场），其次选取尚未开启的登机口，所有合适的登机口都停有飞机时，最后选取简易临时机位。其中，当存在多个已经开启的登机口时，针对单个登机口的选取问题，我们参考操作系统的“**高优先权优先调度算法**”提出了“**非抢占式动态优先级算法**”，选取在等待降落的飞机到达时间之前最晚释放的登机口，来保证所有开启的登机口的高利用率。

问题二分析：本题考虑中转旅客最短流程时间，需要考虑的因素优先级从高到低依次为：成功分配到登机口位的航班数量；总最短流程时间；被使用的登机口数量。我们首先根据每条旅客记录的航班到达和出发类型将其分为 4 类，分别表示出这 4 类的流程时间，希望总流程时间，即它们的和最小。某类记录的流程时间问题实质上是一个 **0-1 规划问题**，我们可以引入两个 0-1 变量 x 和 y 来刻画某类记录的总流程时间。此外，还需引入函数来刻画乘坐同一航班到达的旅客从同一登机口下飞机以及同一航班出发的旅客从同一登机口上飞机。由此，我们可以得到一个最优解，之后还需结合实际情况基于登机口资源剩余情况和“非抢占式动态优先级算法”对结果进行优化。

问题三分析：考虑中转旅客的换乘紧张度。对于一名特定的旅客而言，其航班连接时间为定值，其流程时间、捷运时间以及行走时间都由航班到达的登机口和出发的登机口唯一确定，因此可以构造**换乘耗时矩阵** K，其元素就代表每种中转方案的换乘时间。对于每一条旅客记录，也可以构造一个**门选矩阵** W，它仅有一个元素为 1，其他都为 0，表示该旅客记录选中了相应的中转方案。我们又定义了一个**取出函数** $C(W)$ 来得到该条旅客记录的换乘时间。由此，我们可以构建出优化模型。此外，对于乘坐同一航班到达的旅客以及乘坐同一航班出发的旅客，需要分别从同一个登机口到达和出发。

三、模型假设

（1）对于同一个登机口，第一架飞机出发 45 分钟后，就可以投入后续的分配过程，即这个登机口资源被释放。

（2）对于一个登机口，同一时刻只能停放一架飞机，不存在多架飞机排队出发的情况。

（3）对于一架飞机的转场，从到达到出发都只用到一个登机口，不能在登机口之间或登机口和简易临时机位之间挪动。

（4）简易临时机位的数量无上限。

（5）只考虑 20 日到达或 20 日出发的航班和旅客。

（6）新建卫星厅对始发旅客和终到旅客的影响可以被忽略不计。

（7）所有航班准时到达和出发。

（8）对于每一名旅客，其流程时间都是最短流程时间。

（9）旅客中转换乘期间不会发生意外，导致换乘时间的增加。

（10）旅客不会临时调整航班。

（11）登机口不会临时出现故障，所有登机口在 20 日的 24 小时都保持正常运转。

四、数据预处理

（1）将既不是 20 日到达也非 20 日出发的航班数据过滤。

（2）对航班和登机口分别按照其固有属性分类，分别得到每一类的数目和飞机总数。

（3）对每一条旅客记录，增补其航班的到达类型和出发类型，即是“国内”还是“国际”。

（4）统计出有效旅客记录的数目。

（5）统计出有效旅客数目。

五、问题一求解

5.1 模型准备

本题只考虑航班-登机口分配，要求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。航班和登机口都有三种固有属性，即到达类型、出发类型和机型。对于某个特定的航班，只能在三项条件都满足的那类登机口中选择最优的方案。因此不妨根据这 3 种固有属性分别对航班和登机口进行分类。航班可以分为$2*2*2=8$类，不妨记为：*D_I_W*，*D_I_N*,*I_D_W*，*I_D_N*，*D_D_W*，*D_D_N*，*I_I_W*，*I_I_N*。其中第一个字母代表航班的到达类型，第二个字母代表航班的出发类型，第三个字母代表航班的机型。

例：*D_I_W* 表示国内到达，国际出发且机型为宽型机的航班。

易由附件数据得，登机口可以分为 10 类，不妨记为：*I_I_W*，*I_I_N*，*D_D_W*，*D_D_N*，*I_DI_W*，*D_DI_N*，*DI_D_N*，*DI_I_W*，*DI_DI_W*，*DI_DI_N*。其中第一个字母代表登机口的到达类型，第二个字母代表登机口的出发类型，第三个字母代表登机口能接受的机型。

例：*I_I_W* 表示该登机口适用于国内到达，国内出发且机型为宽型机的航班。*DI_D_N* 表示该登机口适用于国内或国际到达，国内出发且机型为窄机型的航班。

设航班类型集合 F = {*D_I_W*，*D_I_N*,*I_D_W*，*I_D_N*，*D_D_W*，*D_D_N*，*I_I_W*，*I_I_N*}，设登机口类型集合 G = {*I_I_W*，*I_I_N*，*D_D_W*，*D_D_N*，*I_DI_W*，*D_DI_N*，*DI_D_N*，*DI_I_W*，*DI_DI_W*，*DI_DI_N*}。

我们首先定义某类航班总数函数 *Fa(f)*，$f \in F$ 和某类登机口总数函数 *Ga(g)*，$g \in G$。则可由数据预处理结果得到表 3 和表 4。

表 3 航班总数函数 *Fa(f)* 映射表

自变量 *f*	该类航班总数 *Fa(f)*
D_I_W	5
D_I_N	45
I_D_W	5
I_D_N	36
D_D_W	0
D_D_N	145
I_I_W	39
I_I_N	28

表 4 登机口总数函数 *Ga(g)*映射表

自变量 *g*	该类登机口总数 *Ga(g)*
I_I_W	17
I_I_N	4
D_D_W	2
D_D_N	35
I_DI_W	1
D_DI_N	2
DI_D_N	2
DI_I_W	1
DI_DI_W	3
DI_DI_N	2

然后我们定义一个以登机口类型为自变量的通用度函数 *U(g)*，$g \in G$ 来描述登机口到达和出发类型的通用程度，即登机口匹配不同航班的能力。出发类型和到达类型都唯一的登机口可以视作“专用登机口”，它的通用度为 0；出发类型和到达类型都不限的登机口可以视作“万能登机口”，它的通用度为 2；介于两者之间的登机口，通用度为 1。即设：

$$U(g)=\begin{cases}0, g \in \{I\_I\_W, I\_I\_N, D\_D\_W, D\_D\_N\} \\ 1, g \in \{I\_DI\_W, D\_DI\_N, DI\_D\_N, DI\_I\_W\} \\ 2, g \in \{DI\_DI\_W, DI\_DI\_N\}\end{cases}$$

为了尽可能多地分配航班到合适的登机口，并且最小化投入使用的登机口数量，我们将分配过程分为三轮，分别依次考虑 *U(g)*取不同值时，登机口的分配情况。

由此，我们再定义一个以航班类型类型为自变量的分配结果函数 $R_i(f), f \in F$ 来表示第 i 轮 f 类型的航班成功分配的数量。若 $\sum_{1}^{i} R_i(f) = Fa(f), (f \in F, i \in \{1,2\})$，则表示 f 类航班在第 i 轮中分配成功；若 $\sum_{1}^{i} R_i(f) < Fa(f), (f \in F, i \in \{1,2\})$，则表示 f 类航班在第 i 轮中分配失败，需进入第 $i+1$ 轮分配。航班-登机口分配流程如图 3 所示。

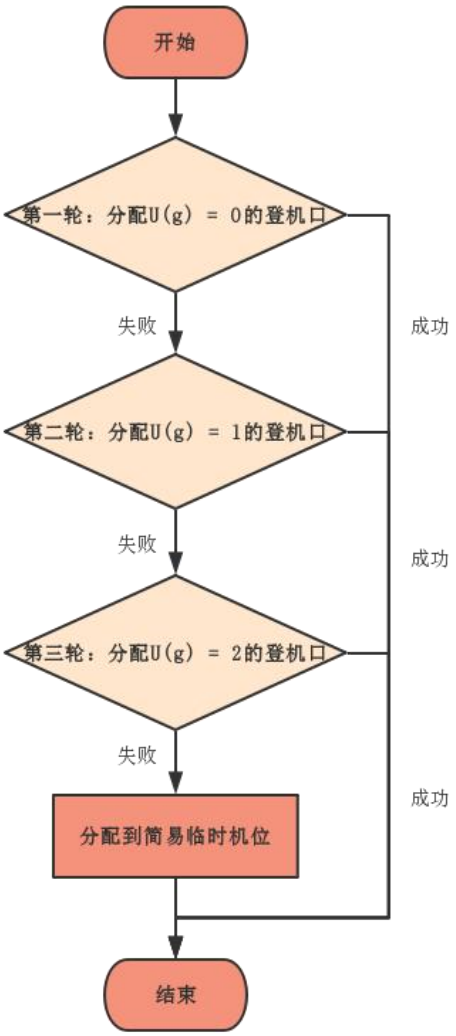

图 3 航班-登机口分配流程

5.2 符号说明

符号	说明
$Fa(f)$	f 类型航班的总数
$Ga(g)$	g 类型登机口的总数
$U(g)$	g 类型登机口的通用程度
$R_1(f)$	第一轮分配中 f 类型航班成功分配的数目
$R_2(f)$	第二轮分配中 f 类型航班成功分配的数目
$R_3(f)$	第三轮分配中 f 类型航班成功分配的数目
$S_1(g)$	第一轮分配中 g 类型航班投入使用的数目

$S_2(g)$	第二轮分配中 g 类型航班投入使用的数目
$S_3(g)$	第三轮分配中 g 类型航班投入使用的数目
$T_{\text{i释放}}$	登机口 i 被释放的时间
$T_{\text{j到达}}$	飞机 j 到达登机口的时间
$T_{\text{j出发}}$	飞机 j 从登机口出发的时间
$P(i)$	登机口 i 的优先函数

5.3 模型的建立和求解

5.3.1 分配原则

（1）分配先后顺序

为了降低开启新登机口的概率，在每一轮的分配中，对于每一架等待降落的飞机，都优先选取已经开启的登机口，即该登机口已经有飞机转场过；其次再选取未开启的登机口。若一轮分配完毕后，仍有飞机没有分配到登机口，则进入下一轮分配。

（2）前两轮已开启登机口的选取原则：非抢占式动态优先级算法

首先，我们给登机口和飞机分别编号：

终端厅 T 的 28 个登机口 T1-T28 依次编号为 1-28；

终端厅 S 的 41 个登机口 S1-S41 依次编号为 29-69；

20 日在该机场起、落的飞机共 303 架，按照到达时间从早到晚的顺序编号为 1-303。

设 $T_{\text{j到达}}$ 表示飞机 $j(j\in\{1,2,...,303\})$到达登机口的时间，$T_{\text{j出发}}$ 表示飞机 j 从登机口出发的时间。

对于登机口 $i(i\in\{1,2,...,69\})$，假如上一架出发的飞机为 j，则定义 $T_{\text{i释放}}$ 为登机口 i 被释放的时间，即下一架飞机可以降落的最早时间。不难得到：

$$T_{\text{i释放}} = T_{\text{j出发}} + 45\text{min}$$

针对每一个登机口 $i(i\in\{1,2,...,69\})$，定义优先函数:

$$P(i) = T_{k\text{到达}} - T_{i\text{释放}}$$ 。

其中，$T_{k\text{到达}}$ 是当前等待降落的飞机 $k(k\in\{1,2,...,303\})$的到达时间。

k 将根据优先函数 $P(i)$的值来选取已开启登机口，值越小，优先级越高；值为负，排除该登机口 i。 因此选取 $P(i)$值为最小的非负数的 i。

这样我们可以得到基于非抢占式动态优先级算法的数学模型：

目标函数 $$\max \sum_{f\in F} R_1(f) + R_2(f) + R_3(f)$$

$$\text{s.t.}\begin{cases}\min\sum_{g\in G}S_1(g)+S_2(g)+S_3(g)\\ \forall g\in G, S_1(g)+S_2(g)+S_3(g)\le Ga(g)\\ \forall f\in F, R_1(f)+R_2(f)+R_3(f)\le Fa(f)\\ \forall i\in\{1,2,3\}, S_i(g)\ge 0\\ \forall i\in\{1,2,3\}, R_i(f)\ge 0\\ \forall k\in \text{某类航班}\subseteq F, i\in \text{某类登机口}\subseteq G, T_{k\text{到达}}-T_{i\text{释放}}\ge 0\end{cases}$$

其中，航班 k 与登机口 i 类型匹配。

5.3.2 前两轮分配结果

第一轮仅考虑 $U(g)=0$ 的专用登机口，第二轮仅考虑 $U(g)=1$ 的登机口，这两轮的分配结果如表 5 所示。

表 5 前两轮分配结果

航班类型 f	$R_1(f)$	$R_2(f)$	$R_1(f)+R_2(f)$	$Fa(f)$	是否成功
D_I_W	0	4	4	5	否
D_I_N	0	11	11	45	否
I_D_W	0	2	2	5	否
I_D_N	0	12	12	36	否
D_D_W	0	0	0	0	是
D_D_N	145	0	145	145	是
I_I_W	39	0	39	39	是
I_I_N	20	0	20	28	否

由表知，*D_D_W* 类型的航班不存在，因此不需要考虑 *D_D_W* 类型航班的分配；此外，有 2 种类型的航班在第一轮就分配成功：*D_D_N*，*I_I_W*；有 5 种类型的航班需要进入第三轮分配：*D_I_W*，*D_I_N*，*I_D_W*，*I_D_N*，*I_I_N*。

5.3.3 第三轮分配

仅考虑 $U(g)=2$ 的万能登机口，$Ga(DI\_DI\_W)=3$，$Ga(DI\_DI\_N)=2$。*DI_DI_W* 型登机口只能分配给 5 种航班类型中的 *ID_W* 和 *DI_W*，对于这两种类型的航班，分别分配出 1 个登机口资源，则：

$R_1(I\_D\_W)+R_2(I\_D\_W)+R_3(I\_D\_W)=5=Fa(I\_D\_W)$，分配成功；

$R_1(D\_I\_W)+R_2(D\_I\_W)+R_3(D\_I\_W)=5=Fa(D\_I\_W)$，分配成功。

将 2 个 *DI_DI_N* 型登机口分配给 *D_I_N*，*I_D_N* 和 *I_I_N* 这 3 种类型的航班，一共有 6 种情况，在不同情况下 $R_3(D\_I\_N)+R_3(I\_D\_N)+R_3(I\_I\_N)$ 的值如表 6 所示。

表 6 不同情况下成功的航班总数

$R_3(D\_I\_N)+R_3(I\_D\_N)+R_3(I\_I\_N)$		第一个 *DI_DI_N* 登机口		
		D_I_N	*I_D_N*	*I_I_N*
第二个 *DI_DI_N* 登机口	*D_I_N*	11	**12**	10
	I_D_N		10	10
	I_I_N			6

将 2 个 *DI_DI_N* 型登机口分别分配给 *D_I_N* 和 *I_D_N* 这两种类型的航班时，$R_3(D\_I\_N)+R_3(I\_D\_N)+R_3(I\_I\_N)$ 达到最大，为 12，因此选择这种方案。

由此我们可以得到三轮航班分配结果如表 7 所示。

表 7 三轮航班分配结果

航班类型	$R_1(f)$	$R_2(f)$	$R_3(f)$	$R_3(f)+R_3(f)+R_3(f)$	$Fa(f)$	需要简易临时机位的个数
D_I_W	0	4	1	5	5	0
D_I_N	0	11	6	17	45	28
I_D_W	0	2	3	5	5	0
I_D_N	0	12	6	18	36	18
D_D_W	0	0	0	0	0	0
D_D_N	145	0	0	145	145	0
I_I_W	39	0	0	39	39	0
I_I_N	20	0	0	20	28	8

“需要简易临时机位的个数”为 0 的航班就是三轮之后成功分配的类型。303 架飞机中成功分配了 249 架，仅 54 架飞机需要停放在简易临时机位上，成功率达到了 82.18%。

三轮登机口资源分配情况如表 8 所示。

表 8 三轮登机口资源分配情况

登机口类型	$S_1(g)$	$S_2(g)$	$S_3(g)$	$S_1(g)+S_2(g)+S_3(g)$	*Ga(g)*	剩余资源
I_I_W	17	0	0	17	17	0
I_I_N	4	0	0	4	4	0
D_D_W	0	0	0	0	2	2
D_D_N	34	0	0	34	35	1
I_DI_W	0	1	0	1	1	0
D_DI_N	0	2	0	2	2	0
DI_D_N	0	2	0	2	2	0
DI_I_W	0	1	0	1	1	0
DI_DI_W	0	0	2	2	3	1
DI_DI_N	0	0	2	2	2	0

“剩余资源”表示该类型登机口未开启的数目。69 个登机口中仍有 4 个未曾投入使用，使用率为 94.20%。

5.4 答案提交

（a）成功分配到登机口的航班的数量统计图：见图 3 和图 4。

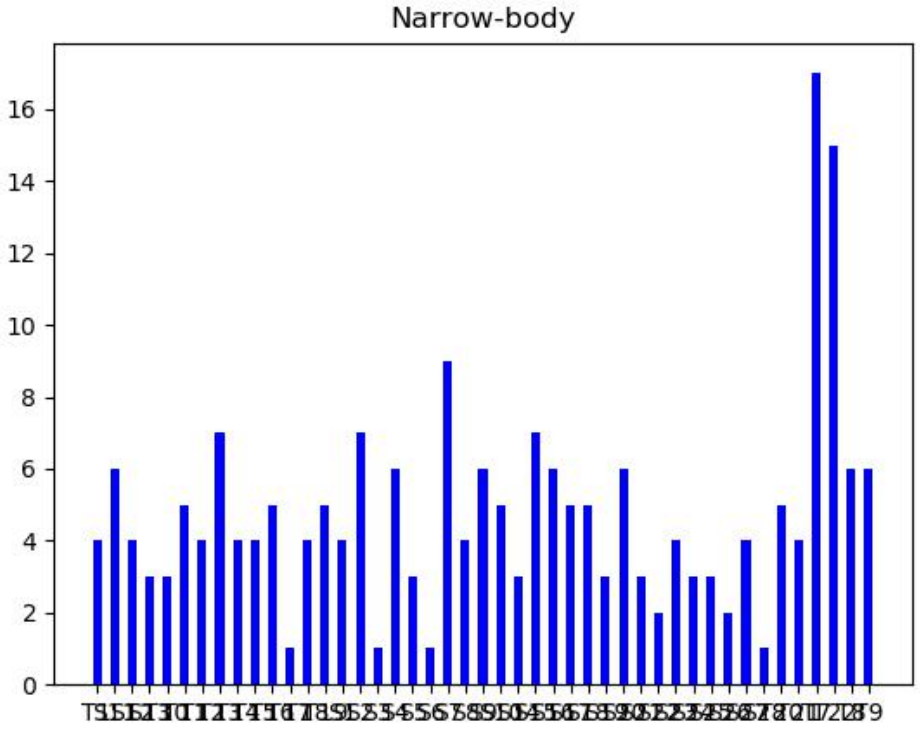

图 3 成功分配的窄体机航班数量统计图

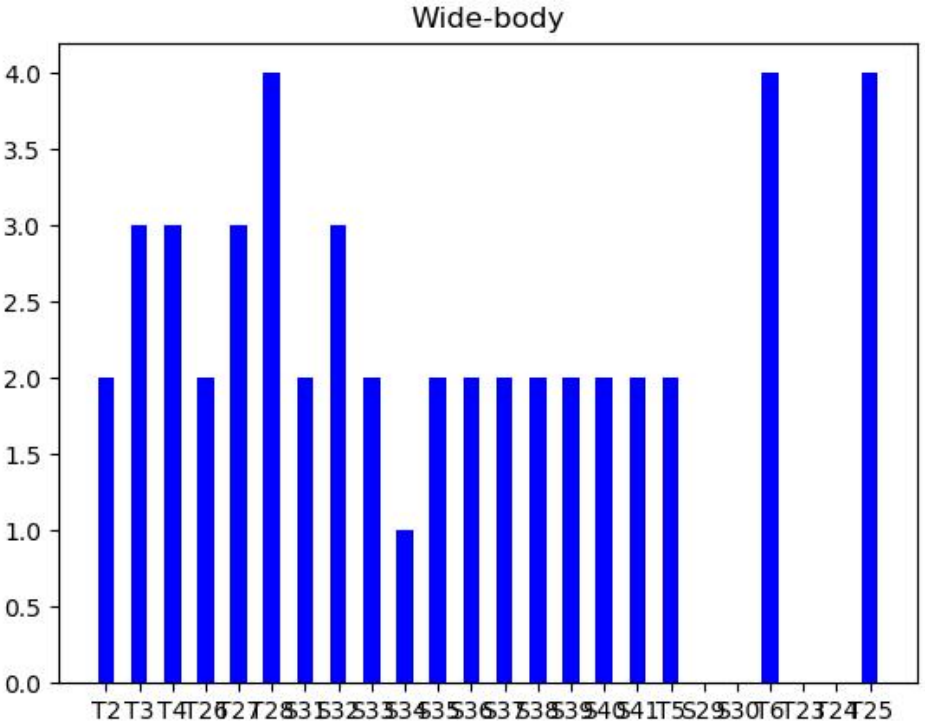

图 4 成功分配的宽体机航班数量统计图

（b）登机口的使用情况展示：横坐标的时间单位为分钟，纵坐标上没有出现的登机口表示没有被开启使用，彩色线条表示该段时间有飞机在该登机口停放。因篇幅所限，仅展示 10 种类型的登机口中 *D_D_N* 型和 *I_I_W* 型的使用情况，见图 5 和图 6。

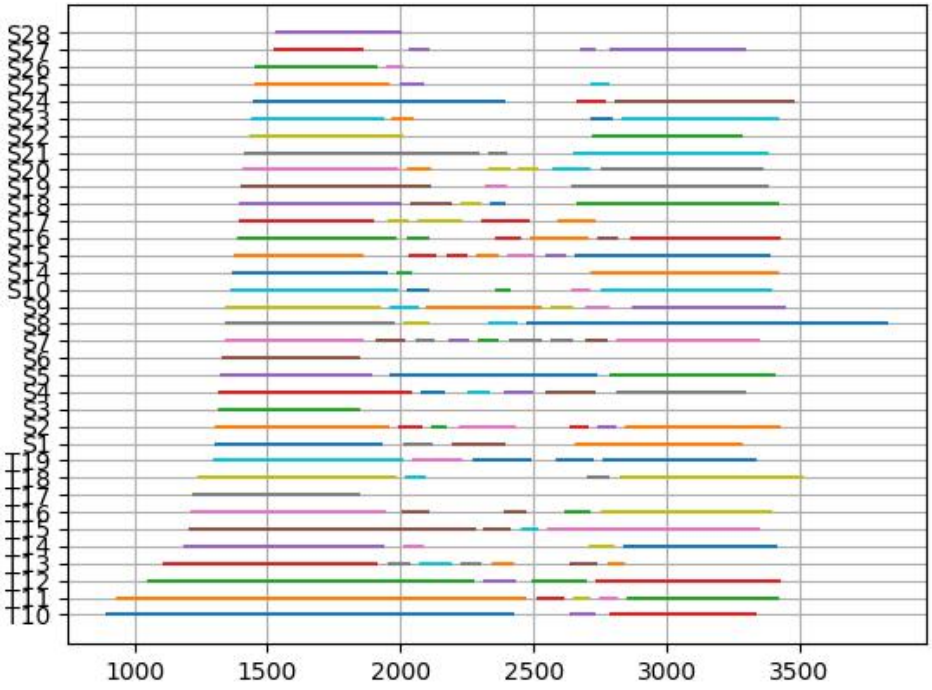

图 5 *D_D_N* 型登机口的使用情况

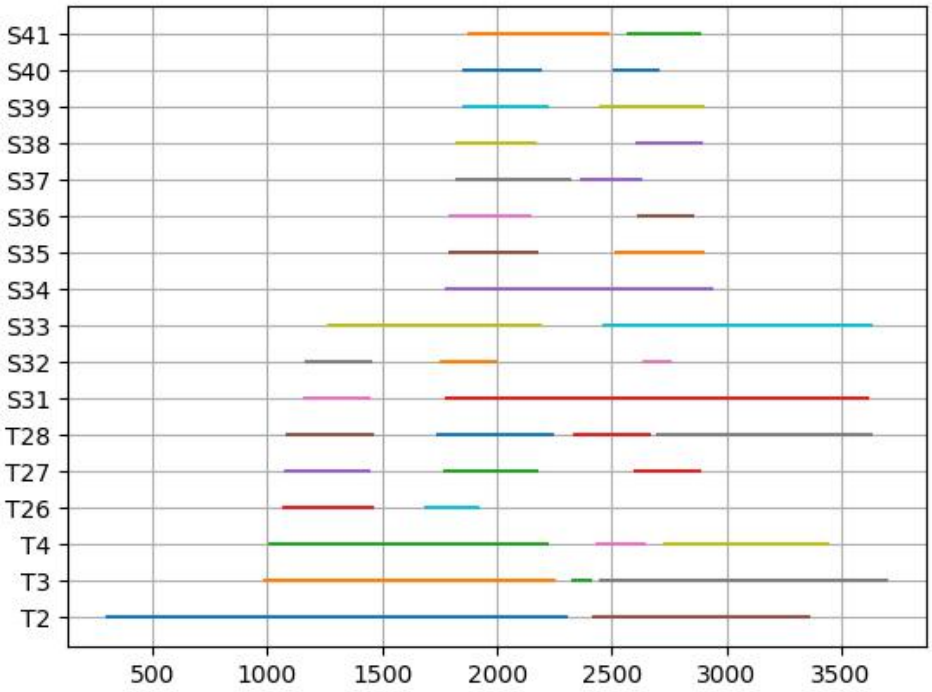

图 6 *I_I_W* 型登机口的使用情况

六、问题二求解

6.1 模型准备

由数据预处理结果知，一共有 1649 条有效旅客记录，首先我们给这 1649 条旅客记录编号：1-1649。

然后根据每条旅客记录的航班到达和出发类型定义 4 个集合：*DD*，*DI*，*II*，*ID*；其元素就是符合集合类型的旅客记录。

例：集合 *DI* 表示其中的旅客记录都是国内到达，国际出发类型的。

考虑到乘坐同一航班到达的旅客只可能在同一个登机口下飞机，乘坐同一航班出发的旅客也只可能在同一个登机口上飞机。因此，我们对有效的旅客信息定义两个集合：*A* 表示 20 日到达的航班班次去掉“NV”后的数字集合；*L* 表示 20 日出发的航班班次去掉“NV”后的数字集合。

设第 $i(i\in\{1,2,...,1649\})$条旅客记录下，到达航班班次去掉“NV”后的数字为 a_i，出发航班班次去掉“NV”后的数字为l_i，则有$a_i \in A, l_i \in L$ 。

6.2 符号说明

符号	说明
$t_{DD流}$	DD 集合所有旅客记录的总流程时间
$t_{DI流}$	DI 集合所有旅客记录的总流程时间
$t_{II流}$	II 集合所有旅客记录的总流程时间
$t_{ID流}$	ID 集合所有旅客记录的总流程时间
N_i	第 *i* 条旅客记录下的旅客数量
$T_{流}$	所有中转旅客的总流程时间
A	20 日到达的航班班次去掉“NV”后的数字集合
L	20 日出发的航班班次去掉“NV”后的数字集合
a_i	第 *i* 条旅客记录下的到达航班班次去掉“NV”后的数字
l_i	第 *i* 条旅客记录下的出发航班班次去掉“NV”后的数字

6.3 模型的建立和求解

这是一个典型的 0-1 问题，对于第 $i(i\in\{1,2,...,1649\})$条旅客记录，我们引入两个 0-1 变量x_i和y_i，设：

$x_i=\begin{cases}0,\text{在终端厅S到达1}\\1,\text{在终端厅T到达}\end{cases}$ 以及 $y_i=\begin{cases}0,\text{从终端厅S出发1}\\1,\text{从终端厅T出发}\end{cases}$，则有：

$$t_{DD流}=\sum_{i\in DD}[x_iy_i*15+x_i(1-y_i)*20+(1-x_i)y_i*20+(1-x_i)(1-y_i)*15]*N_i,$$

$$t_{DI流}=\sum_{i\in DI}[x_iy_i*35+x_i(1-y_i)*40+(1-x_i)y_i*40+(1-x_i)(1-y_i)*35]*N_i,$$

$$t_{II流}=\sum_{i\in II}[x_iy_i*35+x_i(1-y_i)*40+(1-x_i)y_i*40+(1-x_i)(1-y_i)*45]*N_i,$$

$$t_{ID流}=\sum_{i\in ID}[x_iy_i*20+x_i(1-y_i)*30+(1-x_i)y_i*30+(1-x_i)(1-y_i)*20]*N_i,$$

设 $T_{流}$ 为所有中转旅客的总流程时间，则我们可以得到理想化的数学模型：

目标函数：$\min T_{流}=t_{DD流}+t_{DI流}+t_{II流}+t_{ID流}$

$$s.t.\begin{cases}\forall i\in\{1,2,...,1649\},x_i\in\{0,1\}\\\forall i\in\{1,2,...,1649\},y_i\in\{0,1\}\\\text{若}\forall i,j\in\{1,2,...,1649\},i\neq j,a_i=a_j,\text{则}x_i=x_j\\\text{若}\forall i,j\in\{1,2,...,1649\},i\neq j,l_i=l_j,\text{则}y_i=y_j\end{cases}$$

借助 Lingo 软件，我们得到理想化情况下的最优解为 $T_{流}=87575\min$。考虑到现实因素需要加以调整，例如：若对于 DD 型旅客记录，最优方案的总流程时间最小的登机口在终端厅 T，而 T 中符合条件的登机口资源不足时，则根据“非抢占式动态优先级算法”为之选择终端厅 S 中符合要求的登机口。经过调整，实际情况下，2751 名有效旅客中有 2268 名换乘成功，483 名换乘失败，换乘成功率为 82.44%。可以达到的 $T_{流}=76920\min$。

七、问题三求解

7.1 符号说明

符号	说明
Tensity	换乘紧张度
$T_{换}$	所有中转旅客的换乘时间
$T_{流}$	所有中转旅客的最短流程时间

$T_{运}$	所有中转旅客的捷运时间
$T_{走}$	所有中转旅客的行走时间
$T_{i换}$	第 i 条旅客记录下的换乘时间
$T_{i连}$	第 i 条旅客记录下的航班连接时间，即后一航班出发时间与前一航班到达时间的差值
K	换乘耗时矩阵
W	门选矩阵
$C（W）$	取出函数
a_i	第 i 条旅客记录下的到达航班班次去掉“NV”后的数字
l_i	第 i 条旅客记录下的出发航班班次去掉“NV”后的数字
N_i	第 i 条旅客记录下的旅客数量

7.2 模型的建立和求解

“表 1 旅客流程图”的数据部分是一张 4*4 的表格，提供了旅客的流程时间和捷运时间信息；“表 2 行走时间表”的数据部分是一张 7*7 的表格，提供了旅客的行走时间信息。并且已知 $T_{换}=T_{流}+T_{运}+T_{走}$，因此不妨把两张表中的信息整合到一起，这就构成了一张 28*28 的大表格——“旅客换乘时间表”，其左上角 7*7 的“国内到达（D）- 国内出发（D）”部分的换乘时间表如表 9 所示。

表 9 *DD* 型旅客换乘时间表

登机口区域		国内出发（D）						
		T-North	T-Center	T-South	S-North	S-Center	S-South	S-East
国内到达（D）	T-North	25	30	35	53	48	53	53
	T-Center	30	25	**30**	48	43	48	58
	T-South	35	30	25	53	48	45	53
	S-North	53	48	53	25	30	35	35
	S-Center	48	43	48	30	25	30	30
	S-South	53	48	45	35	30	25	35
	S-East	53	58	53	35	30	35	25

由此，我们可以定义一个 28 阶的换乘耗时矩阵 K，K_{mn} 就是 28*28 的旅客换乘时间表上第 m 行第 n 列的值，其物理意义为各种中转方案的换乘时间。对于第 i 条旅客记录，我们定义一个 28 阶的门选矩阵 W_i，它的 $28*28=784$ 个元素中仅有一个元素为 1，其他都为 0。若 W_{imn} 为 1，则表示第 i 条旅客记录选中了 28*28 的旅客换乘时间表上第 m 行第 n 列的中转方案。

再引入一个取出函数 $C(W)$,其自变量为一个门选矩阵 W，因变量为一个数值，对应法则为：取出 K 中对应于 W 中非 0 元素位置的值，即该中转方案的换乘时间，因此其函数值域就是旅客换乘时间表上所有数据的集合的非空子集.

例：若对于第 i 条旅客记录的门选矩阵 W_i，其所有元素中仅 W_{i23} 为 1，则由表 9 知 $C(W_i)=T_{\text{i换}}=30$。

由此，我们可以得到基于最小化换乘旅客总体紧张度的模型：

$$\text{目标函数}\quad \min Tensity=\sum_{i=1}^{1649}\frac{C(W_i)*N_i}{T_{\text{i连}}}$$

$$s.t.\begin{cases}\forall i\in\{1,2,...,1649\},\sum_{s=1}^{28}\sum_{t=1}^{28}W_{ist}=1\\ \text{若}\forall i,j\in\{1,2,...,1649\},i\neq j,a_i=a_j,\text{则}\sum_{t=1}^{28}W_{ist}=\sum_{t=1}^{28}W_{jst},s\in\{1,2,...,28\}\\ \text{若}\forall i,j\in\{1,2,...,1649\},i\neq j,l_i=l_j,\text{则}\sum_{s=1}^{28}W_{ist}=\sum_{s=1}^{28}W_{jst},t\in\{1,2,...,28\}\end{cases}$$

借助 Lingo 软件，我们得到理想情况下的最优解为 *Tensity* = 668.73。考虑到现实因素需要加以调整，调整后可以达到的较优解为 *Tensity* = 805.82。

7. 3 答案提交

（e）总体旅客换乘紧张度分布图：见图 7。

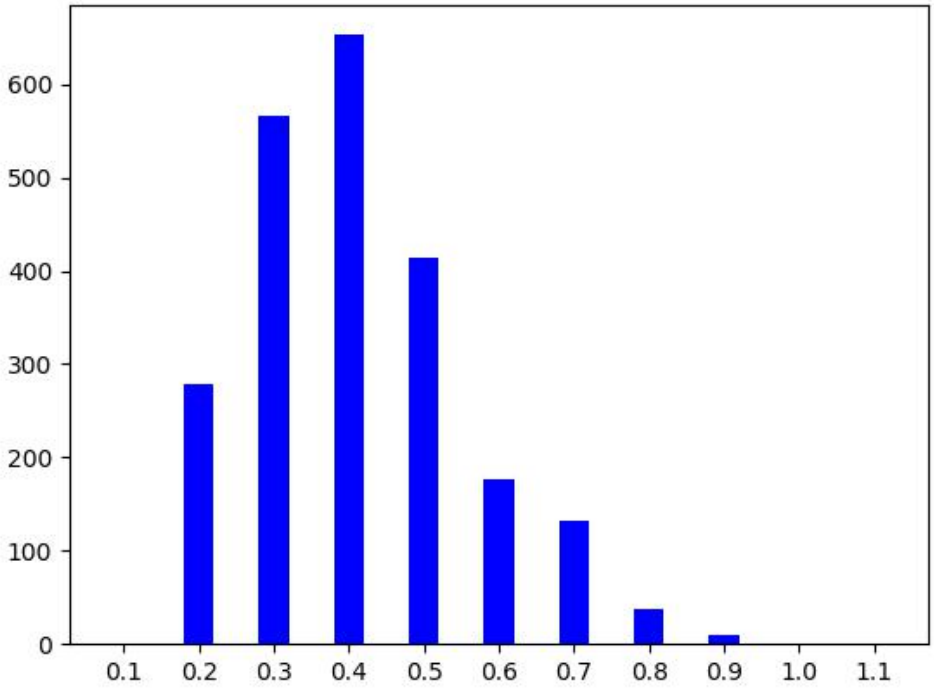

图 7 总体旅客换乘紧张度分布图

八、创新点总结

（1）问题一巧妙地借助集合对航班和登机口进行三轮次的分类讨论，简洁清晰。
（2）问题一以操作系统的“高优先权优先调度算法”为蓝本提出了“非抢占式动态优先级算法”，在尽可能多地分配航班的基础上最小化投入使用的登机口资源。
（3）问题二利用了 0-1 规划的思想，在问题一的基础上引入了两个 0-1 变量进行优化。
（4）问题三创新地提出了换乘耗时矩阵 K，门选矩阵 W 和取出函数 $C（W）$的概念，极大地化简了问题的复杂度。
（5）模型简单明了，易于理解，可操作性强。

九、参考文献

[1] Abdelghani Bouras, Mageed A. Ghaleb, Umar S. Suryahatmaja, and Ahmed M. Salem. The Airport Gate Assignment Problem: A Survey[J]. *The Scientific World Journal*(2014): 27-55, 2014. http://dx.doi.org/10.1155/2014/923859.
[2] 汤小丹，梁红兵，哲凤屏，汤子瀛．计算机操作系统(第三版)[M]．西安:西安电子科技大学出版社, 91-94, 2011.
[3] 王志清．民航旅客运输便捷工程及其流程优化方法研究[D]．南京：南京航天航空大学, 2006: 35-89.
[4] 曹燕．民航运输旅客流程的优化研究[D]．南京：南京航天航空大学, 2005: 5-16.

十、附录一：程序和数据结构流程图

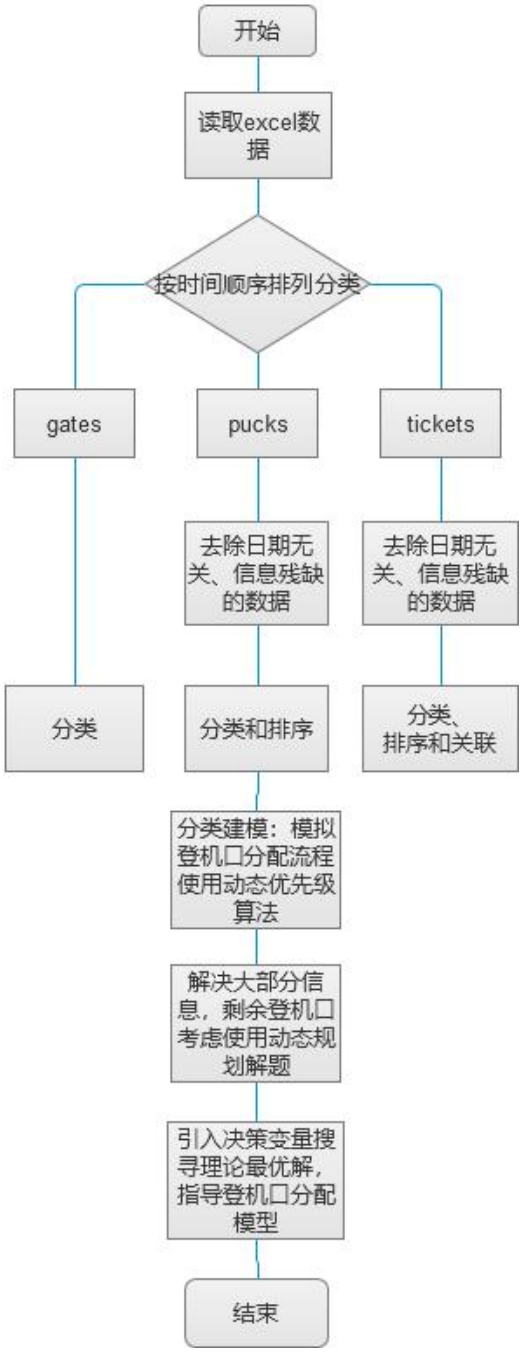

图 8 程序和数据结构流程图

十一、附录二：部分 Python 代码展示

（1）提取 excel 数据

```
def extractGate(path):
    workbook = xlrd.open_workbook(path)
```

```
    sheet = workbook.sheet_by_name('Gates')
    gates = []
    gates_matrix = []
    for i in range(sheet.nrows):
        if i != 0:
            data = sheet.row_values(i)
            gate                                                    =
Gate(data[0],data[1],data[2],data[3],data[4],data[5])
            gates.append(gate)
            gates_matrix.append(data)

    return gates,gates_matrix
```

（2）计算笛卡尔积并保存

```
def saveAsMul(path,matrix_1,matrix_2):
    mapper = []
    for x in itertools.product(matrix_1, matrix_2):
        y = x[0] + x[1]
        mapper.append(y)
    np.savetxt(path,mapper,delimiter=',',fmt='%s')
```

（3）预处理，对数据分类排序

```
def PreOperation_Gates(path):
    gates, gates_matrix = extractExcel.extractGate(path)
    gate_time = np.zeros(gates_matrix.__len__())
    I_I_N_gates = []  # 4
    I_I_W_gates = []  # 17
    I_DI_W_gates = []  # 1

    D_D_N_gates = []  # 35
    D_D_W_gates = []  # 2
    D_DI_N_gates = []  # 2

    DI_DI_W_gates = []  # 3
    DI_DI_N_gates = []  # 2
    DI_D_N_gates = []  # 2
    DI_I_W_gates = []  # 1
    for gate in gates_matrix:
        if gate[3] == 'I' and gate[4] == 'I' and gate[5] == 'N':
            I_I_N_gates.append(gate)
        if gate[3] == 'I' and gate[4] == 'I' and gate[5] == 'W':
            I_I_W_gates.append(gate)
        if gate[3] == 'I' and gate[4] == 'D, I' and gate[5] == 'W':
            I_DI_W_gates.append(gate)
```

```
        if gate[3] == 'D' and gate[4] == 'D' and gate[5] == 'N':
            D_D_N_gates.append(gate)
        if gate[3] == 'D' and gate[4] == 'D' and gate[5] == 'W':
            D_D_W_gates.append(gate)
        if gate[3] == 'D' and gate[4] == 'D, I' and gate[5] == 'N':
            D_DI_N_gates.append(gate)
        if gate[3] == 'D, I' and gate[4] == 'D, I' and gate[5] == 'W':
            DI_DI_W_gates.append(gate)
        if gate[3] == 'D, I' and gate[4] == 'D, I' and gate[5] == 'N':
            DI_DI_N_gates.append(gate)
        if gate[3] == 'D, I' and gate[4] == 'D' and gate[5] == 'N':
            DI_D_N_gates.append(gate)
        if gate[3] == 'D, I' and gate[4] == 'I' and gate[5] == 'W':
            DI_I_W_gates.append(gate)
    return I_I_N_gates,I_I_W_gates,I_DI_W_gates,D_D_N_gates,\

D_D_W_gates,D_DI_N_gates,DI_DI_W_gates,DI_DI_N_gates,DI_D_N_gates,DI_
I_W_gates
```

（4） 动态优先权算法

```
def recommend_First_random_Lates_Second(planes, gatesList, dicFormula,
dicFormulaReverse, answers):
    # 记录 gate 释放时间
    gatesTime = np.zeros(gatesList.__len__())
    # 记录找到 gate 标记
    planeFlag = np.zeros(planes.__len__())
    # 记录找到 bestGate 标记
    planeBestFlag = np.zeros(planes.__len__())
    resultGate = []
    resultArriveTime = []
    resultLeaveTime = []
    for i in range(planes.__len__()):
        flag = 0
        gateNum = 0  # 记录选择的 gate 的 number
        bestWeight = -1  # 记录优先权
        bestFlag = 0
        # 第一步 确定是否会失败
        # 第二步 确定是否有最优解(对应的登机口且空闲值最小)
        # 第三步 确定是否次优解(空闲值最小)
        for j in range(gatesTime.__len__()):
            if planes[i][1] * 24 * 60 + planes[i][2] >= gatesTime[j]:
                # 登机口独占 动态优先权算法
                flag = 1  # 找到至少一个
                if dicFormula.__contains__(planes[i][0]) and \
```

```
                        ((answers[dicFormula[planes[i][0]]] == 0 and
gatesList[j][1] == 'T') \
                         or (answers[dicFormula[planes[i][0]]] == 1 and
gatesList[j][1] == 'S')):
                    # 存在解对应的类型,则使用另外一个循环
                    bestFlag = 1
                    break
                if bestWeight < gatesTime[j]:  # 比较优先权
                    bestWeight = gatesTime[j]
                    gateNum = j
        if bestFlag == 1:
            # 进入此循环后重新记录 bestWeight
            bestWeight = -1
            for j in range(gatesTime.__len__()):
                if planes[i][1] * 24 * 60 + planes[i][2] >= gatesTime[j]:
                    # 找同一类型
                    if dicFormula.__contains__(planes[i][0]) and \
                            ((answers[dicFormula[planes[i][0]]] == 0
and gatesList[j][1] == 'T') or (
                                    answers[dicFormula[planes[i][0]]]
== 1 and gatesList[j][1] == 'S')):
                        # 找最繁忙的 gate
                        if bestWeight < gatesTime[j]:  # 比较优先权
                            bestWeight = gatesTime[j]
                            gateNum = j
        planeFlag[i] = flag
        planeBestFlag[i] = bestFlag
        # 不管有没有找到最优解都要更新时间
        if flag == 1:
            gatesTime[gateNum] = planes[i][6] * 24 * 60 + planes[i][7]
+ 45
            resultArriveTime.append(planes[i][1]  *  24  *  60  +
planes[i][2])
            resultLeaveTime.append(planes[i][6] * 24 * 60 + planes[i][7])
            resultGate.append(gatesList[gateNum][0])
        else:
            resultArriveTime.append(-1)
            resultLeaveTime.append(-1)
            resultGate.append('fail')
    return planeFlag, planeBestFlag, resultGate, resultArriveTime,
resultLeaveTime
```