“华为杯”第十五届中国研究生数学建模竞赛

学　　校	国防科技大学
参赛队号	**18900020072**
队员姓名	**1.赵啸宇**
	2.顾　伟
	3.王钊辉

“华为杯”第十五届中国研究生

数学建模竞赛

题 目　　增设卫星厅的登机口分配问题

摘　　要：

机场合理编制登机口分配计划可以降低运营成本、提升乘客满意度。针对新建卫星厅背景下的登机口分配问题，本文采用 0-1 整数规划模型，考虑了增设登机口的航班分配问题、考虑中转流程时间的登机口分配问题、考虑换成紧张度的登机口分配问题，并分别建立模型求解。

针对问题一，在最大化航班停靠数的基础上，同时考虑最小化被使用登机口的数量。对于双目标优化问题，引入参数简化为单目标优化，结合登机口和航班属性、航班间是否冲突等约束构造登机口分配的 0-1 整数模型。经过数据预处理，共有 303 架次飞机需要进行登机口的调配，通过使用 CPLEX，2s 内可以求得模型最优解。总计可为最多 256 架次飞机安排登机口，共使用 65 个登机口，剩余 47 架次飞机前往临时停机坪，安排的成功率为 88.48%。

针对问题二，在最大化航班停靠数的基础上，同时考虑卫星楼对中转旅客的影响，加入了中转流程时间这一因素。在第一问的基础上增加了记录航班停靠位置的决策变量，同时优化目标增加了对旅客总体流程时间的优化。通过 CPLEX 求解，我们在在 80 秒左右可以求得问题的最优解，旅客中转流程总时间最小为 50025 分钟。

针对问题三，该问题仍然以最大化航班停靠数为首要目标，同时考虑了旅客的捷运时间和行走时间，并定义了旅客总体换乘紧张度作为次要目标，在这两个目标的基础上最小化被使用登机口数量。结合问题二的求解思路，该问题的规模和求解复杂性显著增大，采用 CPLEX 求解器无法在有限时间内求得最优解。为了加快求解速度，我们采用 CPLEX 快速地生成一个可行解，结合禁忌搜索进行优化，极大的提升了优化过程。在成功安排 256 架次飞机，使用 65 个登机口的基础上，使用 CPLEX 在 240 分钟内，求得乘客总的换乘紧张度为 474.45，CPLEX 结合禁忌搜索 732 秒求得乘客总的紧张度 478.06。

针对登机口分配的三个问题，我们建立了三个整数规划模型，并采用 CPLEX 和元启发式算法进行求解。前两个问题求得了最优解，第三个问题在一定时间内求得了满意解。模型实际应用价值高，针对大规模和复杂度比较高的问题，模型求解还需要进一步研究。

关键词：登机口指派 0-1 整数规划 禁忌搜索算法 CPLEX

目录

一、问题重述

近年来，随着中国经济的迅猛发展，民用航空运输事业快速向前迈进。自2006 年来，保持持续稳步增长的状态，中国已经跻身于世界民航大国的行列[1]。在整个航空运输网络系统中，机场作为重要的航线网络交汇点，逐步成为限制航空运输交通流量的瓶颈，特别是繁忙机场的拥堵、航班延误等问题越来越引起人们的广泛关注。为适应我国现代化民航运输业的高速发展，迫切要求机场运营者采用更为科学合理的理论和方法，来帮助其更有效地利用机场中的关键资源。停机位是机场运行管理中的关键资源之一，停机位分配是否合理直接影响到整个机场的运行效率，停机位的优化分配在保障机场安全顺畅、高效运行中发挥着十分重要的作用。

机场停机位分配问题 GAP（Gate Assignment Problem）是指，在考虑机型大小、停机位大小、航班时刻等因素的情况下。对未来一个时间段(一般为 3~4 小时范围内的到港或离港航班指定适宜的登机口。保证航班正点不延误，为旅客上下飞机提供登机门。停机位分配恰当与否，不仅对提高航空器的利用率关系密切，而且对保证航班计划的实现、降低运输成本、为旅客提供优良的服务影响重大。为航班分配停机位包括航班占用停机位时间和占用具体停机位两项内容。这既与航空器种类、到港和离港密集程度有关，又与机场设备设施和机位分配方法有关。合理编制停机位分配计划是机场生产指挥中心完成作业任务的核心工作之一[2]。

题目要求在优化分配登机口的同时考虑最小化旅客行走时间、降低旅客换乘紧张度，此方向更加注重乘客对航空公司的满意度，更符合服务型航空公司“以人为本”的发展方向，但在此方面学界研究有限，市场上产品一般也不具备此功能。目前对于 GAP 问题的研究已经有很多成果[3]，但问题的优化目标多集中于降低停机位开放成本、减小乘客行走距离[4]、减少停机位航班冲突等方面。[5-6]

二、问题分析

根据主要任务可见，本题是按照分步设计的思路。在考虑增设卫星厅之后，首先完成航班-登机口分配以满足飞机停靠的基本需求，在此基础上最小化使用停机位的数量以减小成本，然后从换乘旅客的角度出发，尽可能的规避新建卫星厅对中转旅客带来的影响。最后将新建卫星厅的交通因素纳入考虑，对旅客换乘舒适度进行综合考量。

问题一只考虑飞机-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。问题一种需要考虑的约束包括机型与登机口的属性匹配和同一登机口两飞机之间的最小空挡间隔。问题一中变量规模较小，考虑的约束复杂性较低，可以采用整数规划建模，并采用精确求解算法进行求解。

问题二考虑了中转旅客最短流程时间。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，且同样要求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，因此，只需判断旅客的到达航班与出发航班各自停在航站楼还是卫星厅，再根据航站楼与卫星厅的最短流程时间表匹配流程时间。问题二复杂性比问题一有所提高，可以尝试精确求解和启发式方式进行求解。

问题三，考虑中转旅客的换乘时间。在问题二的基础上更加细化，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。影响换乘时间的因素众多，主要有航班的到港离港特点、飞机停靠位置、是否需出入境手续等。影响旅客换乘时间的因素比问题二有明显的增加，在每个乘客到达类型和离开类型确定的基础上，旅客登机口区域有 7 种选择，换乘时间有 49 种选择。同时，每架次航班停靠位置的变化，不仅仅影响一个乘客的换乘时间，因此问题的复杂性和求解规模较大。采用精确求解方法在有限时间内得到最优解比较困难，需要尝试采用启发式方法在合理时间内求得满意的可行解。

三、问题假设及符号说明

3.1 问题假设

（1）临时机位

分配不到固定登机口的飞机停靠在机场的临时机位，在此假定临时机位数量无限制。因临时机位无停靠特性，因此停靠于临时机位的航班统一算做入港不离港，临时机位的旅客统一看作只入港不换乘，在问题二、问题三中不予以考虑。

（2）问题分析时段

飞机转场计划和中转旅客信息包含 2018 年 1 月 19、20、21 三天，在此仅对 20 日到达或 20 日出发的航班和旅客进行分析。从飞机转场计划表和中转旅客信息表中，分别剔除于 19 日、21 日当天到达且离场的飞机及旅客信息，与 19 日到达 21 日离港的飞机及旅客信息。20 日到达或者离开的旅客，如果到达或者离开其中一架次航班不在上述考虑内的，这些旅客在问题三种不考虑。

（3）优化目标优先等级

问题一中各目标的优先级为：最大化匹配固定登机口>最小化临时机位。

问题二中各目标的优先级为：最大化匹配固定登机口>最小化中转旅客流程时间>最小化临时机位。

问题三中各目标的优先级为：最大化匹配固定登机口>最小化中转旅客总体紧张度>尽最小化临时机位。

（4）换乘失败

换乘失败的定义为：旅客乘坐到达航班与换乘航班均停靠在固定登机口，但旅客在机场中转过程中，到达换乘航班登机口的时间晚于换乘航班的出发时间。中转旅客失败则换乘时间设置为 6 小时，换乘失败会对航空公司带来巨大损失并造成负面影响，因此在考虑旅客中转时，将换乘失败的旅客作为惩罚项加入模型中。问题二中旅客最小中转时间小于航班之间的最小连接时间，因此惩罚因素只在问题三种考虑。

（5）飞机属性匹配

飞机属性匹配严格按照题目中的要求，其中窄机型不能停靠在宽型机型的停机口。

（6）问题二乘客换乘流程时间

问题二只考虑旅客换乘过程中的流程时间，不考虑捷运和步行时间。判断旅客是否能及时换乘时也不考虑这些因素。旅客流程时间最长为 45 分钟，小于航班间最小连接时间，因此问题二不考虑旅客能否换乘成功。

3.2 符号说明

符号	说明
I	20 日当天到达与 20 日当天离开的总航班数
J	航站楼与卫星厅的登机口总数目
J_T	航站楼的登机口总数目
x_{ij}	0-1 变量，航班 i 是否停靠在登机口 j
y_i	0-1 变量，登机口 j 是否开放
w_{ij}	航班 i 与登机口 j 的属性是否匹配
$\rho_{i_1 i_2}$	两个航班 $i1$ 与 $i2$ 是否冲突
μ 、λ	多目标处理参数
k	换乘乘客编号
T_i	0-1 变量，航班 i 是否停靠在航站楼 T
S_i	0-1 变量，航班 i 是否停靠在卫星厅 S
T_a^k	0-1 变量，旅客 k 到达的航班 a 是否在航站楼 T
Tran	中转流程时间
f_{im}	0-1 变量，航班 i 停靠在登机口区域 m
p_{kmn}	乘客 k 从登机口区域 m 中转到登机口区域 n
$tran_{mn}$	从区域 m 到区域 n 的最小中转时间
$tranMAX_k$	乘客 k 的航班连接时间
T_m	区域 m 的起始编号
M	航站楼与卫星厅划分的区域总数量
f_{km}^a	0-1 变量，乘客 k 到达航班是否处于区域 m
f_{kn}^l	0-1 变量，乘客 k 出发航班是否处于区域 n

四、数据预处理

4.1 飞机数据处理

问题一针对的是飞机的转场问题，在给出的原始数据(表 1)中，飞机转场计划信息包含 2018 年 1 月 19、20、21 三天总计 753 条，但竞赛只需要对 20 日到达或 20 日出发的航班和旅客进行分析。因此，我们首先对飞机转场记录进行筛选，总计 303 条。为了便于下一步分析，我们记每条记录为一架飞机，包含两个航班，从 0-302 依次编号，并进行一些等价变换，如表 2 所示：

表 1 原始数据表

转场记录号	到达日期	到达时刻	到达航班	到达类型	出发航班	出发日期	出发时刻	出发类型	飞机型号

PK001	19-Jan-18	10:00	NV6294	D	NV3118	19-Jan-18	12:40	I	321

表 2 处理后的专场记录表（完整表格见附件）

飞机编号	转场记录号	到达时间	到达航班	到达类型	出发航班	出发时间	出发类型	飞机型号	飞机类型
0	PK151	-210	NV6117	1	NV866	605	0	33E	W
1	PK178	-110	GN945	1	GN0476	580	0	73H	N
2	PK185	-70	NV617	1	NV6616	535	0	321	N
3	PK268	640	NV303	1	NV303	750	0	320	N
4	PK269	645	NV643	1	NV6732	725	0	320	N

为了便于计算时间间隔，在此以问题开始时间（2018 年 1 月 20 日 00 时 00 分）为基准时间，以分钟为粒度，将数据中标准时间格式的时间进行转化，动作在基准时间之前则为负，动作在基准时间之后则为正。

对航班入境出境类型及飞机的宽窄体类型进行处理。国内航班（类型 D）记为 0，国际航班（类型 I）记为 1。结合机型数据表，该航班执行飞机为宽机则为 W，该航班执行飞机为窄机则为 N。

如表 2 中飞机编号为 0 的记录，其到达时间为-210，即表示该飞机在 2018 年 1 月 19 日 20 时 30 分到达，为国际航班。出发时间为 605，即表示该飞机在 2018 年 1 月 20 日 10 时 05 分出发，出发类型为国内航班。执飞飞机为宽体机。

4.2 旅客数据处理

问题二和问题三是在问题一的基础上构造而成的，针对的也是 20 日的旅客，因此我们选出所搭乘的到达航班和出发航班均在问题一中出现的旅客，共计 2751 人次。

问题二、三考虑的流程时间、换乘紧张度等均是以旅客为主体。因此我们需要对旅客记录进行处理，搭乘同一架次航班（相同时间和相同航班号）到达同时搭乘同一架次航班离开的进行合并，筛选后旅客原始记录如表三所示，处理后的旅客数据如下（表 3）所示：

表 3 处理后的旅客数据表（完整表格见附件）

旅客记录号	乘客数	到达航班	到达日期	出发航班	出发日期
T1356	2	NV673	19-Jan-18	NV664	20-Jan-18
T1357	2	NV673	19-Jan-18	NV664	20-Jan-18
T1358	2	NV673	19-Jan-18	NV664	20-Jan-18
T1359	2	NV673	19-Jan-18	NV664	20-Jan-18
T1360	2	NV673	19-Jan-18	NV664	20-Jan-18
T1361	2	NV673	19-Jan-18	NV320	20-Jan-18
T1362	2	NV673	19-Jan-18	NV320	20-Jan-18

从表 3 中我们可以看到前 5 条记录的旅客均是搭乘同一次航班，又搭乘了同一架航班离开，可以进行合并，合并后结果如表 4 所示。

表 4 合并后的旅客数据表（完整表格见附件）

数量	到达飞机	出发飞机
10	192	189
7	192	192
1	192	190
2	193	189
1	193	188

以第一行为例，该行表示有 10 个旅客乘坐了飞机 192（对应表二中的 19 号到达的航班 NV673）到达，并且乘坐了飞机 189（对应表二中的 20 号出发的航班 NV664）离开。

五、问题建模与求解

5.1 登机口分配问题

5.1.1 登机口分配问题分析

问题一的求解目标是在满足同一登机口飞机之间的空挡间隔时间的基础上，尽可能多地分配航班到合适的登机口，并在此基础上最大化被使用登机口的数量。由于本问题不用考虑中转旅客的影响，实质上是航班与登机口的停靠匹配问题，问题的复杂性相对较小。因此，我们选择 0-1 整数规划模型进行求解。在求解过程中需要考虑的约束有：

（1）同一个登机口同一时间最多只能停靠一架飞机。

（2）飞机停靠的登机口属性必须相互匹配，包括飞机到达属性（国内/国际）、飞机离开属性（国内/国际）、机型（宽体机/窄体机）。

（3）分配在同一登机口的两飞机之间的空挡间隔时间必须大于等于 45 分钟。

优化目标是在最大化航班停靠登机口数的基础上，最小化被使用登机口的数量。该问题是一个双目标优化的问题，最大化航班停靠数为最高优化目标，为了方便求解，我们将这两个目标合并为一个目标，将第二个目标取反转换为最大化目标，并乘以一个较小的权重，以达到双目标同时求解。

5.1.2 登机口分配的 0-1 整数规划模型

模型中定义的决策变量，其中 x_{ij} 和 y_j 都为 0-1 变量。

（1） x_{ij} :航班 i 停靠在登机口 j，若停靠则为 1，不停靠则为 0。

（2） y_j :登机口 j 是否开放，若开放则为 1，不开放则为 0。

为了方便求解，在此引入了两个变量 w_{ij} 和 $\rho_{i_1 i_2}$。w_{ij} 表示航班i与登机口j属性是否匹配，$arriveType$代表到达类型为国内/国际；$leaveType$表示处罚类型为国内/国际；$flightType$表示飞机型号为宽体/窄体。若三项均一致，则 w_{ij} 为 1，否则为 0，处理的伪代码为：

w_{ij}:

for $flight\ i$ **from** 1 **to** I

  for $gate\ j$ **from** 1 **to** J

    If$(flight_i.arriveType == gate_j.arriveType$ &&

      $flight_i.leaveType == gate_j.leaveType$ &&

      $flight_i.flightType == gate_j.flightType)$

        $w_{ij} = 1$

  end

end

$\rho_{i_1 i_2}$ 表示航班i_1和航班i_2是否冲突。$startTime$表示飞机入港时间，$leaveTime$表示飞机出港时间，$tranTime$为空档时间，即两架占用同一登机口的飞机，出入间隔必须大于空档时间，在此取 45 分钟。若前一航班的出港时间与后一航班的入港时间不满足空档时间，则判断两航班有冲突，即 $\rho_{i_1 i_2}$ 为 0；若前一航班的出港时间与后一航班的入港时间间隔大于空档时间，则两航班无冲突，即 $\rho_{i_1 i_2}$ 为 1。

$\rho_{i_1 i_2}$:

if $(flight_{i_1}.leaveTime + tranTime < flight_{i_2}.startTime$ or

  $flight_{i_1}.arriveTime - tranTime < flight_{i_2}.leaveTime)$

  $\rho_{i_1 i_2} = 0$

else

  $\rho_{i_1 i_2} = 1$

$$\max \sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} - \mu \sum_{j=1}^{J} y_j \qquad (1)$$

$$\sum_{j=1}^{J} x_{ij} = 1 \quad (1 \le i \le I) \qquad (2)$$

$$x_{ij} \le w_{ij} \qquad (3)$$

$$x_{i_1 j} + x_{i_2 j} \le 2 - \rho_{i_1 i_2} (i_1 \ne i_2) \qquad (4)$$

$$x_{ij} \le y_j \qquad (5)$$

$$x_{ij} = 0 \text{ or } 1, \ \ y_j = 0 \text{ or } 1 \ \ (1 \le i \le I, 1 \le j \le J) \qquad (6)$$

公式（1）为问题一的优化目标，$\sum_{i=1}^{I}\sum_{j=1}^{J}x_{ij}$ 表示最大化航班停靠登机口数，优化目标$\sum_{j=1}^{J}y_j$ 表示开启的登机口数量，将$\sum_{j=1}^{J}y_j$ 取反与目标$\sum_{i=1}^{I}\sum_{j=1}^{J}x_{ij}$ 一致以获得最小化登机口数量，并乘以一个较小的权重系数μ，在求解过程中，我们设置$\mu=0.01$。

公式（2）表示每个航班只能停靠在一个登机口；公式（3）表示每个航班必须与停靠的登机口属性匹配；公式（4）表示保证一个登机口只能同时停靠同时停靠一架飞机。当航班i_1和航班i_2冲突时，即$\rho_{i_1 i_2}=1$时，航班i_1和i_2不能停靠在通一个登机口j，即$x_{i_1 j}$和$x_{i_2 j}$不能同时为1；公式（5）表示航班必须停靠在开放的登机口。公式（6）规定了x_{ij}和y_j都为0-1变量。

5.1.3 模型求解

从问题一的模型可知，该优化问题决策变量包括了303×69个0-1变量x_{ij}和69个0-1变量y_j，共计20976个0-1变量。并且问题中的约束条件和优化目标相对简单，因此可以采用CPLEX进行编程求解。CPLEX是IBM公司开发的用于数学规划求解器。目前，超过1000所大学、1000多个公司和政府机构正在使用CPLEX。CPLEX优化程序提供了解决实际的大型优化问题所需的能力，它提供灵活的高性能优化程序，解决线性规划、二次方程规划和混合正数规划等问题，几乎帮助解决了每个行业的规划和计划问题。

在本问题的解决中，我们采用JAVA语言调用CPLEX的API进行求解。在配置为CORE i7 7500U，主频2.7GHZ，4GB内存的电脑上，CPLEX可以在2s内求得最优解。

5.1.4 登机口分配问题求解结果

针对目标一，利用上述方法进行求解，在参与分配的303架次飞机中，我们总计可以为最多256架次飞机（即512架次航班）安排合适登机口，成功率为84.48%。剩余47架次飞机前往临时停机坪。其中，宽体飞机安排了49架次（共49架次），窄体飞机安排了207架次（共254架次），未安排上固定登机口的47架次均为窄体飞机执飞。

针对目标二，本结果共使用65个登机口，未使用的登机口一个位于航站楼T，三个位于卫星厅S。

具体航班和登机口的对应结果如下表所示：

表5 航班登机口对应表（完整表格见附件）

飞机	登机口	飞机转场记录号	宽窄类型	到达时间	出发时间	到达类型	离开类型
0	4	PK151	W	-210	605	1	0
2	7	PK185	N	-70	535	1	0

5	21	PK270	N	660	735	1	0
6	7	PK278	N	745	835	1	0
7	6	PK285	N	800	865	1	0

以第二行为例，飞机 2 是窄体飞机，对应转场记录号为 PK185，其到达航班为国际航班，到达时间为-70（19 日 22：50），出发航班为国内航班，离开时间为 535（20 日 8：55）。

对登机口分配模型的求解结果进行可视化处理，相关图表如下所示。

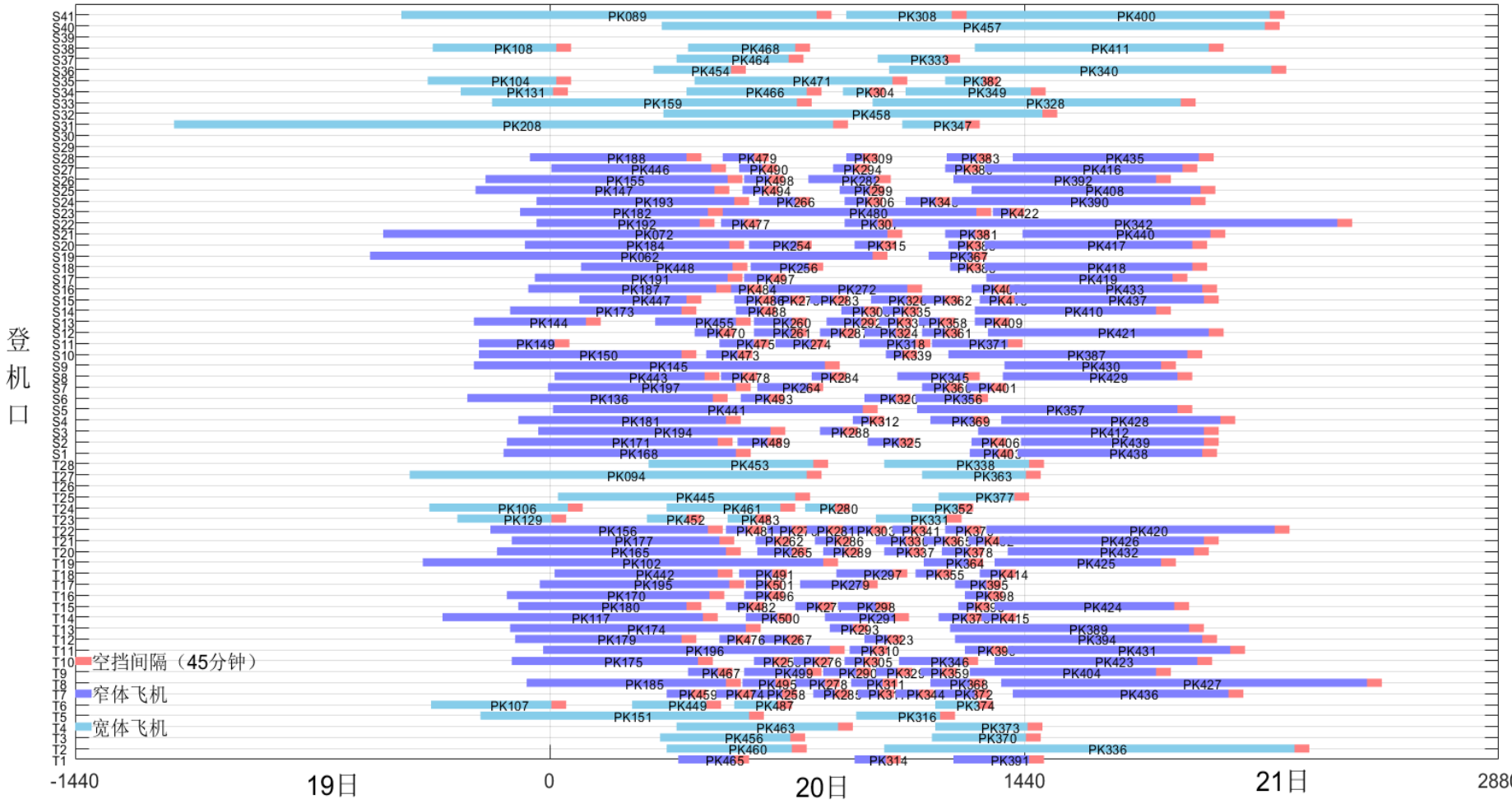
登机口
空挡间隔（45分钟）
窄体飞机
宽体飞机
-1440
19日
0
20日
1440
21日
2880
12

图 1 问题一登机口占用甘特图

图 1 为问题一的登机口占用时间甘特图。图中无交叉的登机口占用计划，可知结果计划满足问题约束。图中横坐标为时间，取值范围为-1440（19 日 0 时 0 分）——2880（22 日 0 时 0 分），基准时间为 0（20 日 0 时 0 分）。纵坐标为航站楼和卫星厅的各登机口。可以看出登机口被占用的时间长短差异很大。

1、短时间的停靠多集中于 20 日 10 点~15 点之间，且多为窄体飞机。在数据预处理环节可知窄体飞机大多执飞国内航班，可以理解为国内航班在早上执飞完第一航段后到本机场进行转场，并于不久后执飞下一航段。

2、窄体飞机的到达和出发比较集中，集中到达均为每日的晚间，集中出发均在每日的早上，这也很大程度上符合多数窄体飞机执飞国内航班这一特点。

3、横跨两日的停机位占用中，占用 19~20 日的航班无论从数量上还是占用长度上都少于占用 20~21 日的航班。

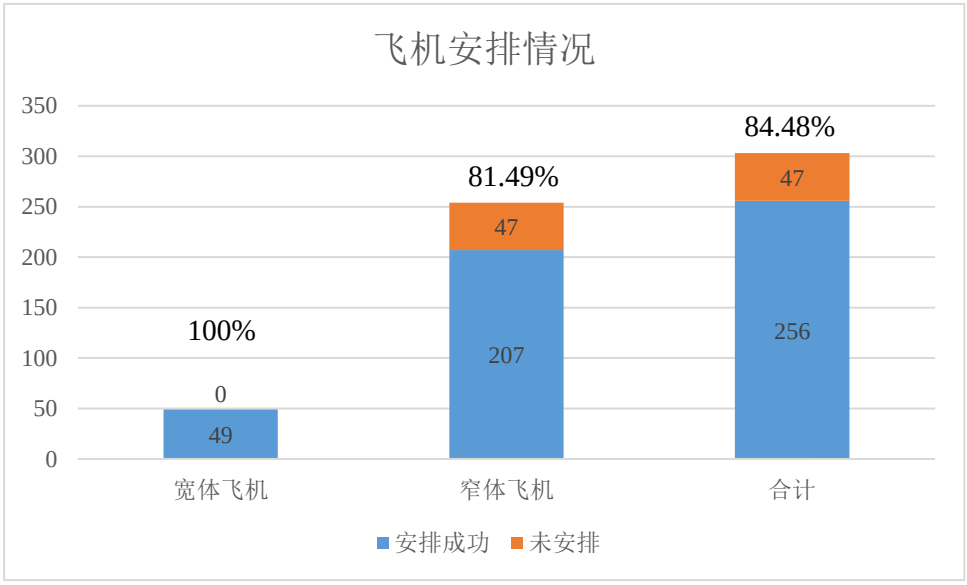

图 2 问题一飞机安排情况图

图 2 为飞机安排情况柱状图，由结果可知，宽体飞机均被安排到了固定登机口，而窄体飞机有约 19%未安排到固定登机口，转到了临时机位。登机口分配的总体成功率为 84.48%。

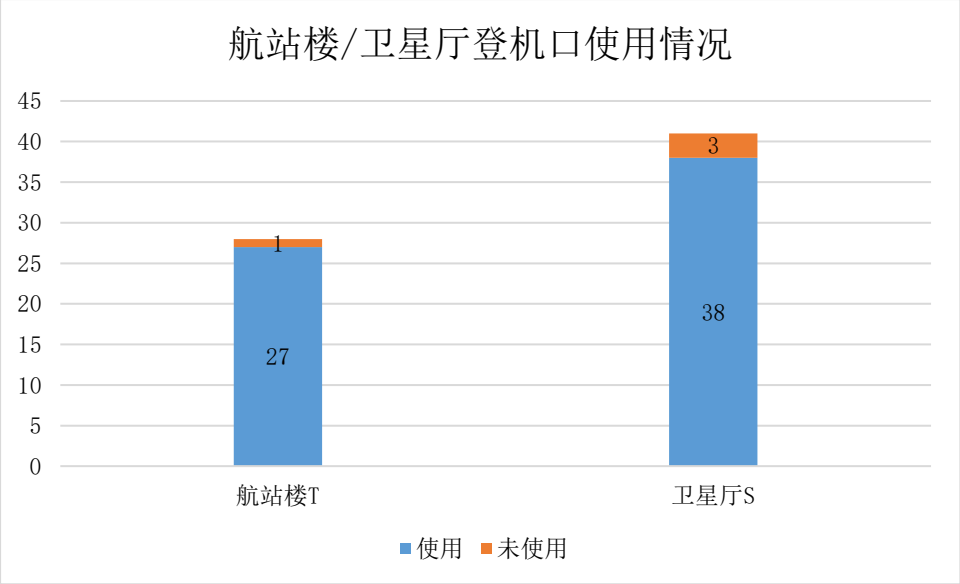

图 3 问题一航站楼/卫星厅登机口使用情况图

图 3 展示了航站楼与卫星厅的整体使用情况，由结果分析得知，航站楼的 28 个登机口中只有一个未开放，而卫星厅的 41 个登机口中有 3 个未开放。

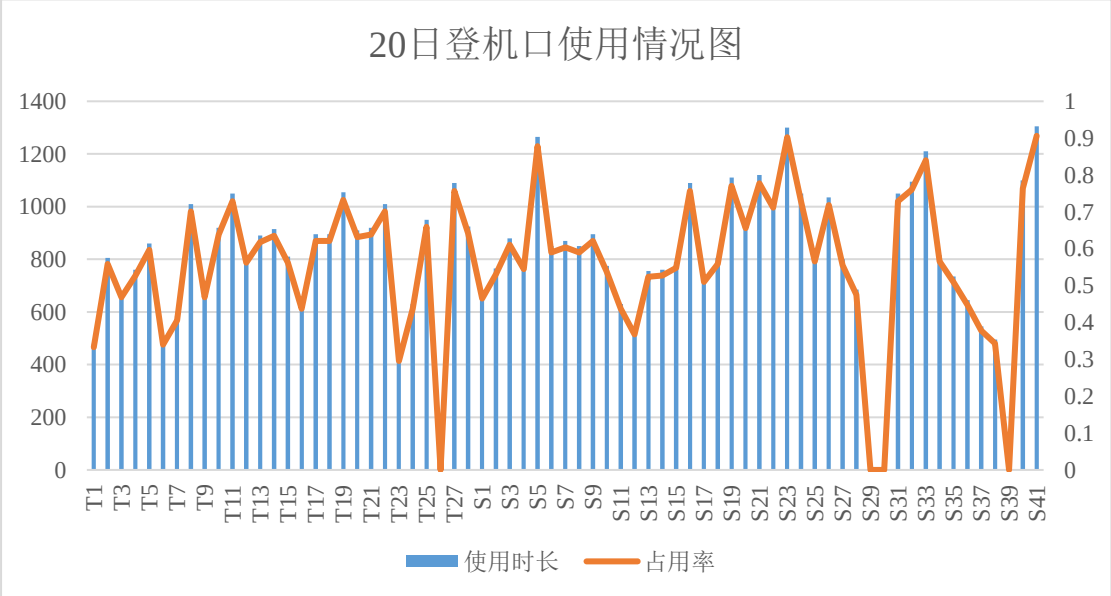

图 4 20 日登机口使用情况图

图 4 为 20 日的登机口使用率图，登机口的使用情况差距十分明显，如 S23 与 S41 的占用率超过 90%，而 T6、T23、S37、S38 等使用率均低于 40%。未开放的登机口为 T26、S29、S30、S39。

5.2 考虑中转流程时间的登机口分配问题

5.2.1 考虑中转流程时间的问题分析

问题二是在问题一的基础上考虑了中转旅客的流程时间，在加入旅客换乘的因素下，最小化中转旅客的总体流程时间。旅客的中转流程时间见表 5。

从表中可以看出，旅客的中转流程时间与到达类型、出发类型以及航班到达和出发的位置有关。因此在问题一的基础上，不仅要考虑到航班停靠的位置是否产生冲突，还需要考虑停靠位置对于旅客中转时间的影响。对于每一个乘客，到达类型和出发类型是确定的，只需要确定旅客到达和出发的航班的位置，即可确定旅客中转的时间。为了解决问题二，我们在第一问的基础上增加了记录航班停靠位置的决策变量，同时优化目标增加了对旅客总体流程时间的优化。通过CPLEX 求解，我们在较短的时间之内求得了问题的最优解。

表 6 中转流程时间表

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	20	35	40
	卫星厅 S	20	15	40	35
国际到达（I）	航站楼 T	35	40	20	30
	卫星厅 S	40	45	30	20

5.2.2 基于停靠位置的登机口分配模型

模型中定义的决策变量如下：

（1）x_{ij}:航班i停靠在登机口j，若停靠则为 1，不停靠则为 0。

（2）y_j:登机口j是否开放，若开放则为 1，不开放则为 0。

（3）T_i:航班i停靠在航站楼 T，若停靠则为 1，不停靠则为 0。

（4）S_i:航班i停靠在卫星厅 S，若停靠则为 1，不停靠则为 0。

其中x_{ij}、y_j、T_i和S_i都为 0-1 变量。

$$\min \sum_{k=1}^{K}(T_a^k T_l^k Tran_{TT} + T_a^k S_l^k Tran_{TS} + S_a^k T_l^k Tran_{ST} + S_a^k S_l^k Tran_{SS}) - \lambda \sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij} + \mu \sum_{j=1}^{J} y_j \tag{7}$$

$$\sum_{j=1}^{J} x_{ij} = 1 \quad (1 \le i \le I) \tag{8}$$

$$x_{i_1 j} + x_{i_2 j} \le 2 - \rho_{i_1 i_2} \quad (i_1 \ne i_2) \tag{9}$$

$$T_i = \sum_{j=1}^{J_T} x_{ij} \quad (1 \le i \le I) \tag{10}$$

$$S_i = \sum_{j=J_T+1}^{J} x_{ij} \quad (1 \le i \le I) \tag{11}$$

$$x_{ij} \le y_j \tag{12}$$

$$x_{ij} \le w_{ij} \tag{13}$$

$$x_{ij} = 0 \text{ or } 1,\ y_j = 0 \text{ or } 1\ (1 \le i \le m, 1 \le j \le n) \tag{14}$$

公式（7）代表代表问题二的优化目标，其中$\sum_{p=1}^{P}(T_a^k T_l^k Tran_{TT} + T_a^k S_l^k Tran_{TS} + S_a^k T_l^k Tran_{ST} + S_a^k S_l^k Tran_{SS})$表示最小化乘客的流程时间。其中$a$表示乘客到达的航班编号，$l$表示乘客离开的航班编号，$T_a^k$表示乘客 k 到达的航班是否在航站楼，$S_a^k$表示乘客 k 到达的航班是否在卫星厅，$T_l^k$表示乘客 k 离开的航班是否在航站楼，$S_l^k$表示乘客 k 离开的航班是否在卫星厅。$Tran_{TT}$、$Tran_{TS}$、$Tran_{ST}$、$Tran_{SS}$分别为航站楼到航站楼、航站楼到卫星厅、卫星厅到航站楼、卫星厅到卫星厅的流程时间。优化目标$\sum_{j=1}^{J} y_j$表示最小化登机口数量，该目标优先级最低，乘以一个较小的权重μ。优化目标$\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ij}$表示最大化航班停靠登机口数，在求解过程中优先级最高，我们将其取反与其他目标一致，并乘以权重λ。在本模型中，我们设置$\mu = 0.01$，$\lambda = 100000$。

公式（8）表示每个航班只能停靠在一个登机口；公式（9）表示当航班i_1和航班i_2冲突时，即$\rho_{i_1 i_2} = 1$时，航班i_1和i_2不能停靠在通一个登机口 j，即$x_{i_1 j}$和$x_{i_2 j}$不能同时为 1；公式（10）表示航班i是否在航站楼，J_T表示航站楼登机口数量；公式（11）表示航班i是否在卫星厅；公式（12）表示航班必须停靠在开放的登机口；公式（13）表示每个航班必须与停靠的登机口属性匹配。公式（14）规定了x_{ij}和y_j都为 0-1 变量。

5.2.3 模型求解

问题二中的模型优化变量包括了 303×69 个 0-1 变量x_{ij}、69 个 0-1 变量y_j、303 个 0-1 变量T_i和 303 个 0-1 变量S_i，共计 21582 个 0-1 变量。决策变量、优化目标和约束条件相对于第一问复杂性有所增加，但是规模不大。因此，我们先尝试用 JAVA 调用 CPLEX 求解器 API 进行求解。求解器在问题一相同配置的电脑中，经过 83.70s 可以求得最优解。

5.2.4 考虑中转流程时间的求解结果

问题二的第一目标依旧是尽可能多地安排航班，因此结果同样为 256 架次飞机安排在合适登机口，剩余 47 架次飞机前往临时停机坪。

修改目标函数后，目标一得到的结果为 256 架次。

针对目标二，根据原始数据计算得出，20 日中转旅客共有 2751 人次。在 2751 人次旅客中，有 1685 人次分配到了固定登机口，1066 人次分配到了临时停机坪，

分配成功率为 61.25%。分配到固定登机口的 1685 人次旅客全部换乘成功，换乘失败率为 0，中转流程时间共计 50025 分钟。

目标三，本结果共使用 65 个登机口，4 个未使用登机口均为卫星厅 S 的登机口。

具体航班和登机口的对应结果样例如下表所示：

表 7 问题二航班登机口对应结果样例表

飞机	登机口	飞机转场记录号	宽窄类型	到达时间	出发时间	到达类型	离开类型
0	23	PK151	W	-210	605	1	0
3	6	PK268	N	640	750	1	0
5	21	PK270	N	660	735	1	0
6	7	PK278	N	745	835	1	0
7	6	PK285	N	800	865	1	0

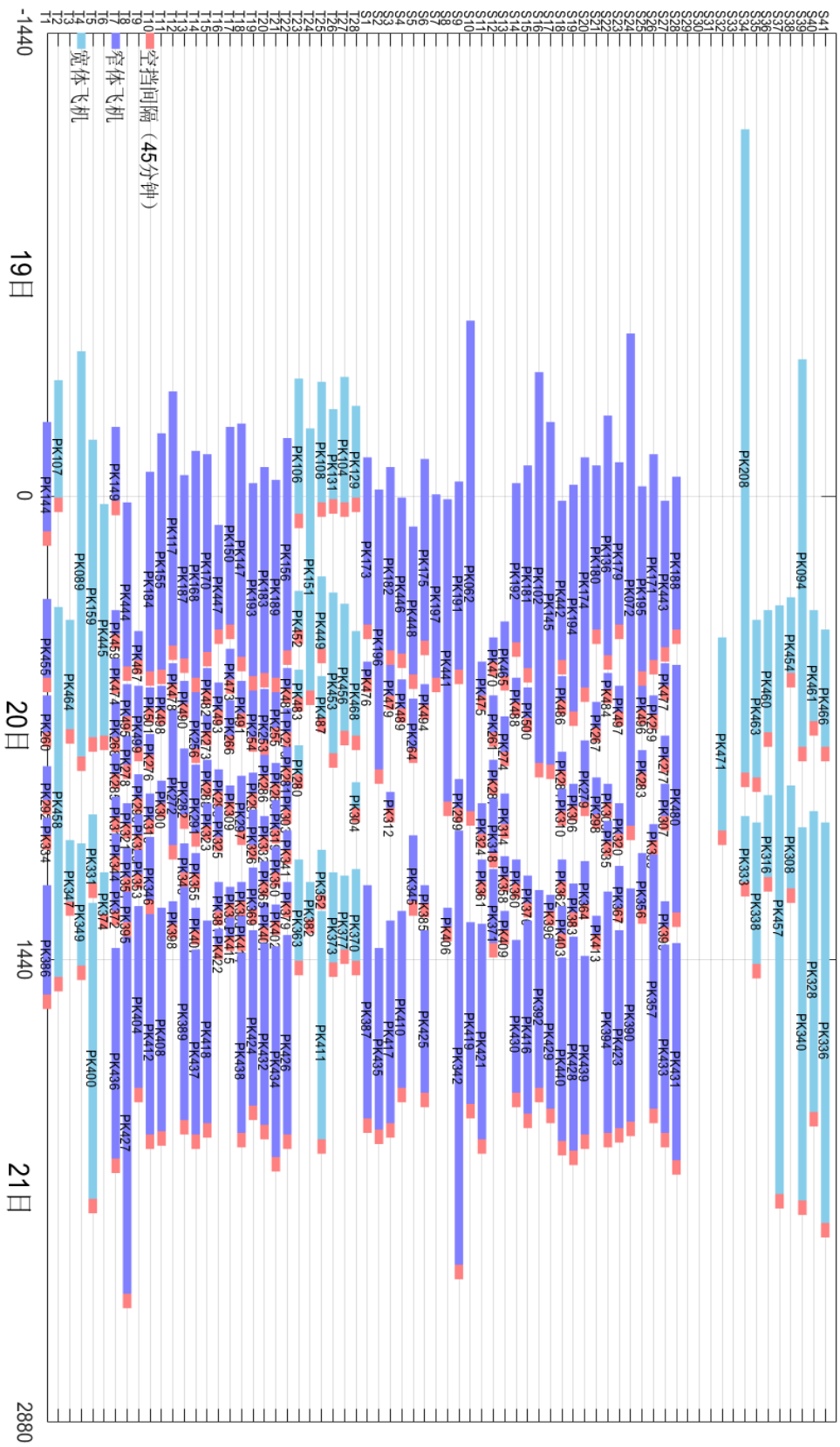

图 5 问题二登机口占用甘特图

图 5 是考虑中转旅客后的结果甘特图，与单一的登机口分配甘特图相比，可以看出短时间停留的中转航班分布更加的集中，且主要集中于航站楼 T，从结果可知，如此安排可以最小化总体旅客流程时间。

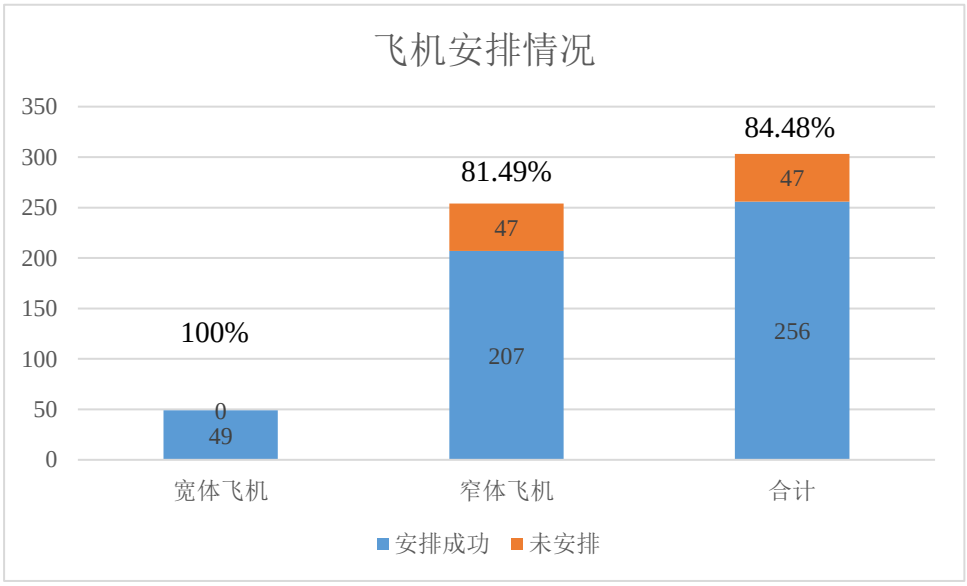

图 6 问题二飞机安排情况图

图 6 为问题二飞机安排情况柱状图，由结果可知，宽体飞机均被安排到了固定登机口，而窄体飞机有约 19%未安排到固定登机口，转到了临时机位。登机口分配的总体成功率为 84.48%。

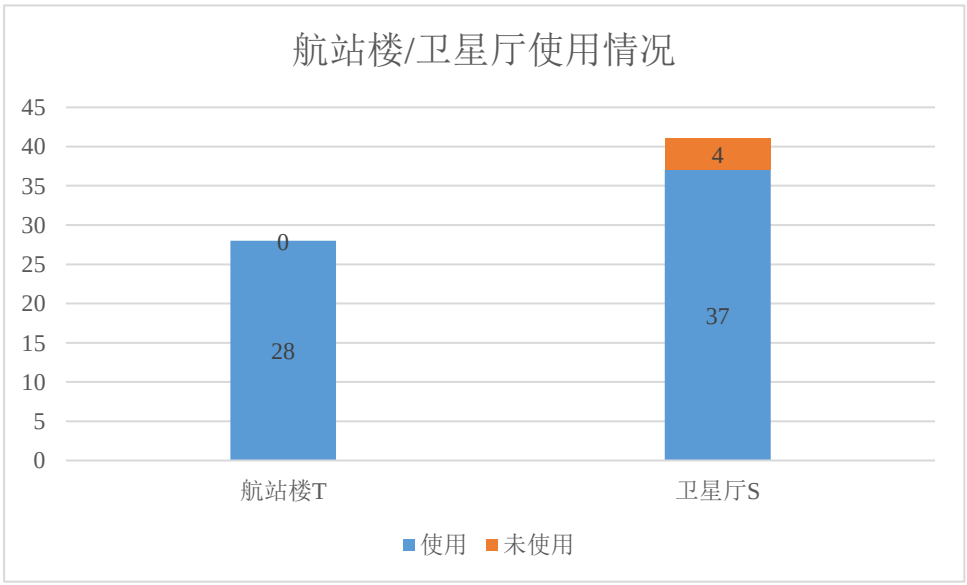

图 7 问题二航站楼/卫星厅使用情况图

图 7 展示了航站楼与卫星厅的整体使用情况，该结果与问题一的结果有所差异，航站楼的 28 个登机口中全部开放，而卫星厅的 41 个登机口中有 4 个未开放。

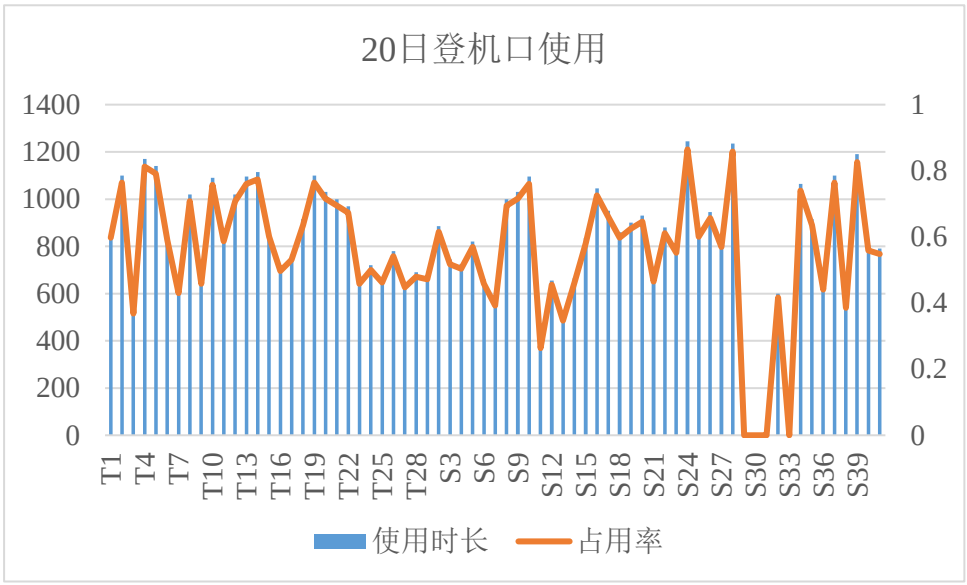

图 8 问题二 20 日登机口使用图

图 4 为 20 日的登机口使用图，登机口的使用率峰值略低于问题一，如 S24、S28、S39 的占用率超过 80%，而 S11、S13、S38 等使用率均低于 40%。未开放的登机口为 S29、S30、S31、S39。

成功分配的 1685 人次旅客中转流程时间共计 50025 分钟，流程时间分布如下图所示：

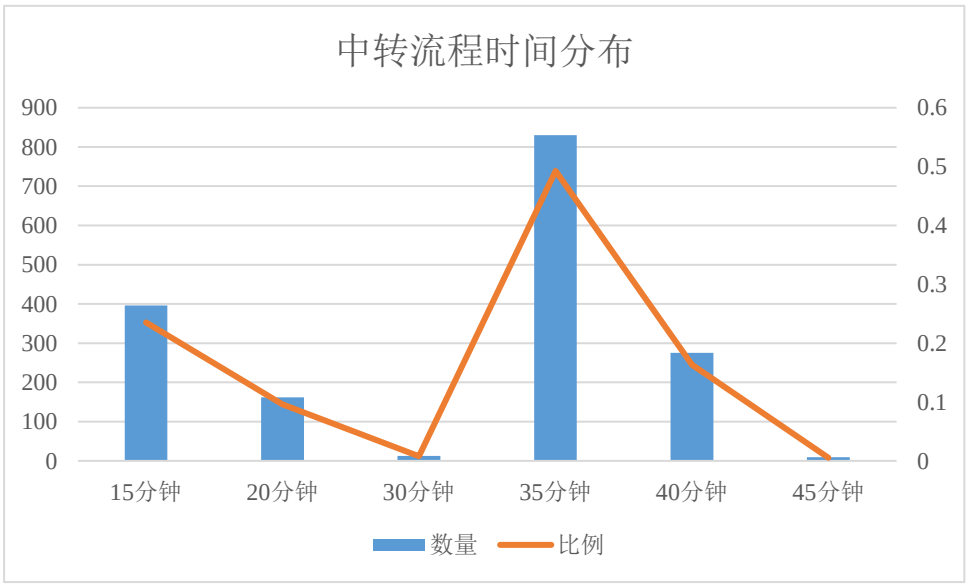

图 9 中转流程时间分布图

从中转流程时间分布图中可以得出，中转流程时间比例最多的为 35 分钟，15 分钟次之，之后是 40 分钟与 20 分钟，占据比例较少的是 30 分钟与 40 分钟，该结果与原始数据的旅客特点较为符合。

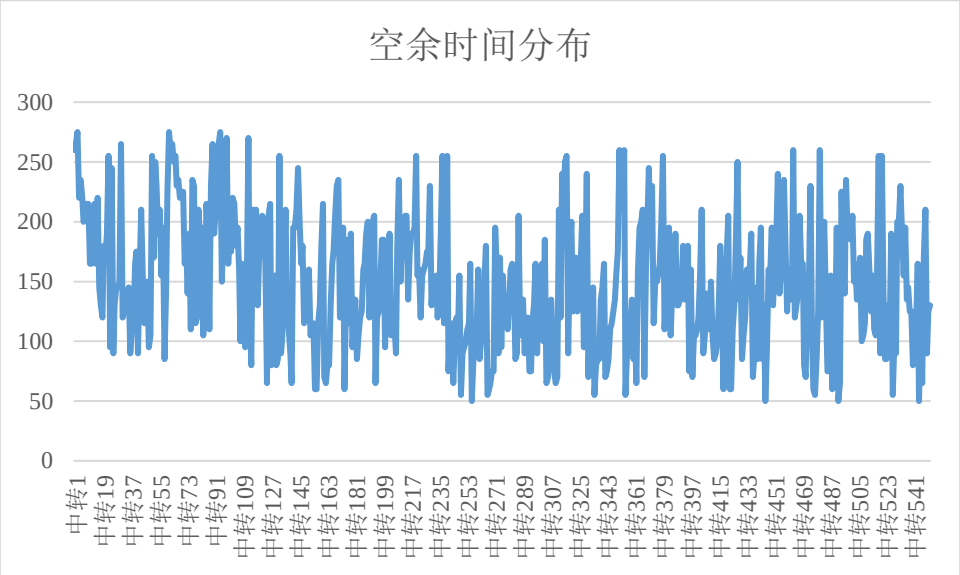

图 10 中转空余时间分布图

乘客换乘航班出发时间减去旅客到达机场的时间为可用中转时间。空余时间为可用中转时间与中转流程时间之差，由图可以看出，空余时间均大于 0，可知所有旅客均有充足的时间进行换乘，无换乘失败的旅客。

5.3 考虑换乘紧张度的登机口分配问题

5.3.1 考虑换乘紧张度的问题分析

问题三在前两问的基础上考虑了旅客的捷运时间和行走时间，并定义了旅客的换乘紧张度：旅客换乘时间与航班连接时间的比值。其中旅客换乘时间为最短流程时间、捷运时间和行走时间的总和。航班连接时间为旅客后一班航班出发时间与前一班航班到达时间的差。

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

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

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

模型求解过程中需要确定每一个旅客的换乘时间和航班连接时间。因为旅客的到达和出发航班是确定的，所以旅客的航班连接时间可以直接确定。旅客换乘时间不仅与旅客前后航班是否在航站楼或卫星厅有关，还与航班所处航站楼或卫星厅的位置有关。

已知一位旅客，我们可以确定该旅客到达和出发航班的类型，即国内或者国际。旅客的登机口区域一共有 T-North、T-Center、T-South、S-North、S-Center、S-South 和 S-East7 种类型，因此，旅客换乘时间一共有 49 种选择。根据每种选择，我们可以对照行走时间表、最短流程时间表算出旅客需要的最短中转时间：

$$\text{最短中转时间} = \text{行走时间} + \text{最短流程时间} + \text{捷运次数} \times \text{单次捷运时间}$$

因此，在前两问的基础上，我们对每个航班增加了 7 个变量记录航班停靠的登机口区域。对每一位旅客增加了 49 个变量记录旅客最短中转时间的组合。由于问题规模的增大和复杂性的提高，求解难度增大，采用 CPLEX 在有限时间内

求得最优解困难度增加。

5.3.2 基于乘客的登机口分配问题模型

模型中定义的决策变量：

（1）x_{ij}:航班i停靠在登机口j，若停靠则为1，不停靠则为0。

（2）y_j:登机口j是否开放，若开放则为1，不开放则为0。

（3）f_{im}：航班i停靠在登机口区域m，若停靠则为1，不停靠则为0。

（4）p_{kmn}：乘客k从登机口区域m中转到登机口区域n，若中转则为1，不中转则为0。

其中x_{ij}、y_j、f_{im}、p_{kmn}均为0-1变量。

$$\min\sum_{k=1}^{K}\sum_{m=1}^{M}\sum_{n=1}^{M}(p_{kmn}\bullet tran_{mn})/tranMAX_k-\lambda\sum_{i=1}^{I}\sum_{\mathrm{j}=1}^{J}x_{ij}+\mu\sum_{j=1}^{J}y_j \quad (15)$$

$$\sum_{\mathrm{j}=1}^{J}x_{ij}=1\ \ (1\le i\le m) \quad (16)$$

$$x_{i_1j}+x_{i_2j}\le 2-\rho_{i_1i_2}\ \ (i_1\ne i_2) \quad (17)$$

$$x_{ij}\le y_j \quad (18)$$

$$x_{ij}\le w_{ij} \quad (19)$$

$$f_{im}=\sum_{j=T_m}^{T_{m+1}}x_{ij} \quad (20)$$

$$p_{kmn}\ge f_{km}^{a}+f_{kn}^{l}-1 \quad (21)$$

$$x_{ij}=0\ or\ 1,\ \ y_j=0\ or\ 1\ \ (1\le i\le m,1\le j\le n) \quad (22)$$

公式（15）表示问题三的优化总目标，$tran_{mn}$表示从区域m到区域n的最小中转时间，$tranMAX_k$表示乘客k的航班连接时间。T_m表示区域m的起始编号。f_{km}^{a}表示乘客k到达航班是否处于区域m，f_{kn}^{l}表示乘客k出发航班是否处于区域n。三个目标分别如下，$\sum_{k=1}^{K}\sum_{m=1}^{M}\sum_{n=1}^{M}(p_{kmn}\bullet tran_{mn})/tranMAX_k$表示旅客换乘

紧张度之和。优化目标$\sum_{i=1}^{I}\sum_{j=1}^{J}x_{ij}$表示航班停靠登机口数，在求解过程中，优先级最高，我们将其取反与其他目标一致，并乘以权重λ。优化目标$\sum_{j=1}^{J}y_j$表示登机口数量，该目标优先级最低，乘以一个较小的权重μ。在本模型中，我们设置$\mu=0.01$，$\lambda=100000$。

公式（16）表示每个航班只能停靠在一个登机口；公式（17）表示当航班i_1和航班i_2冲突时，即$\rho_{i_1 i_2}=1$时，航班i_1和i_2不能停靠在通一个登机口j，即$x_{i_1 j}$和$x_{i_2 j}$不能同时为 1；公式（18）表示航班必须停靠在开放的登机口；公式（18）表示每个航班必须与停靠的登机口属性匹配；公式（20）表示若航班 i 停靠在 m 所属区域的登机口，则航班 i 停靠的区域为 m；公式（21）表示当乘客 k 到达航班停靠 m，离开航班停靠 n 时，乘客的中转时间为区域 m 到区域 n 的中转时间。

5.3.3 CPLEX 求解结果

问题三首先采用 CPLEX 求解，因问题规模过大，经过十几小时仍未求出精确解。

在此我们给出 CPLEX 求解时长为 240 分钟的结果。问题三的第一目标依旧是尽可能多地安排航班，因此结果仍为 256 架次飞机安排在合适登机口，剩余 47 架次飞机前往临时停机坪，旅客换乘紧张度为 474.45，共使用 65 个登机口。

换乘失败的定义为：旅客乘坐到达航班与换乘航班均停靠在固定登机口，但旅客在机场中转过程中，到达换乘航班登机口的时间晚于换乘航班的出发时间。经过统计，结果中换乘失败率为 0。

具体航班和登机口的对应结果如下表所示：

表 8 问题三航班登机口对应结果表

飞机	登机口	飞机转场记录号	宽窄类型	到达时间	出发时间	到达类型	离开类型
0	23	PK151	W	-210	605	1	0
1	8	PK178	N	-110	580	1	0
3	8	PK268	N	640	750	1	0
4	6	PK269	N	645	725	1	0
5	21	PK270	N	660	735	1	0

问题三的飞机安排情况和航站楼与卫星厅的整体使用情况与问题二在总体数量上相同，都是宽体飞机均被安排到了固定登机口，而窄体飞机有约 19%未安排到固定登机口，转到了临时机位。登机口分配的总体成功率为 84.48%。航站楼的 28 个登机口全部开放，而卫星厅的 41 个登机口中有 4 个未开放。虽然总体情况相同，但是，飞机的具体安排是不同的（前往了不同登机口），卫星厅具体开放的登机口也是不同的（具体数据见附件结果）。

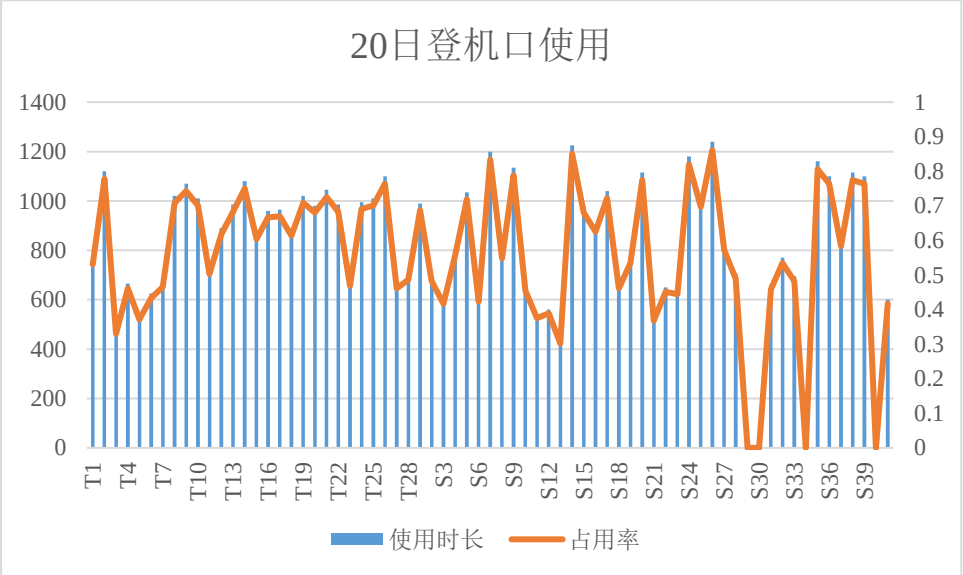

图 11 问题三 20 日登机口使用图

从 20 日的登机口使用图上可以看出，S26 的使用率最高达 86.11%，在所有开放的登机口中使用率最小的是 S13，使用率为 30.2%，未开放的登机口为 S29、S30、S34、S40。总体而言，登机口的使用率较前两问更加均匀，波动更小，说明第三问的优化目标在实际中更有意义。

在 2751 人次旅客中，有 1684 人次分配到了登机口，1067 人次分配到了临时停机坪，分配成功率为 61.21%。成功分配的 1684 人次旅客的换乘时间分布和换乘紧张度分别如图 11 和图 12 所示：

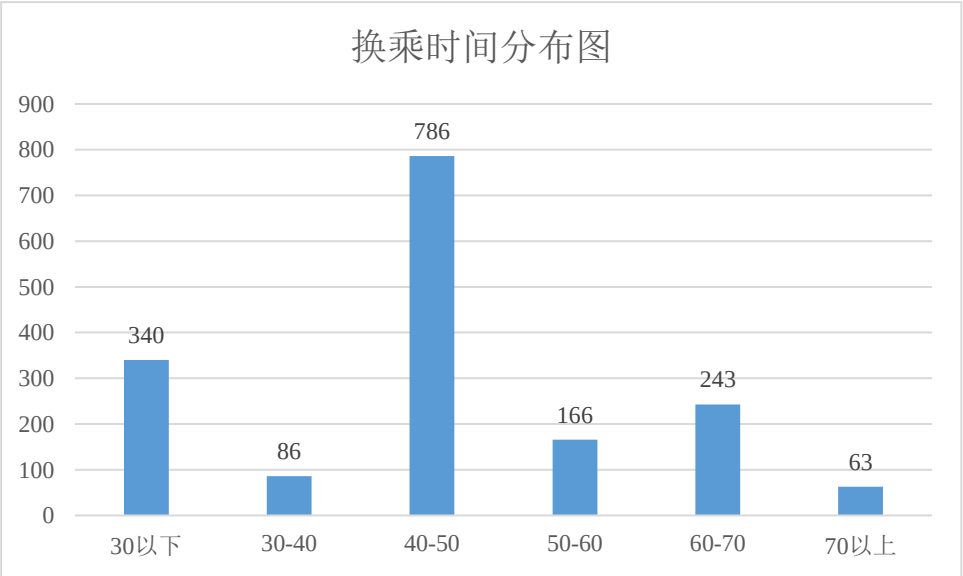

图 12 换乘时间分布图

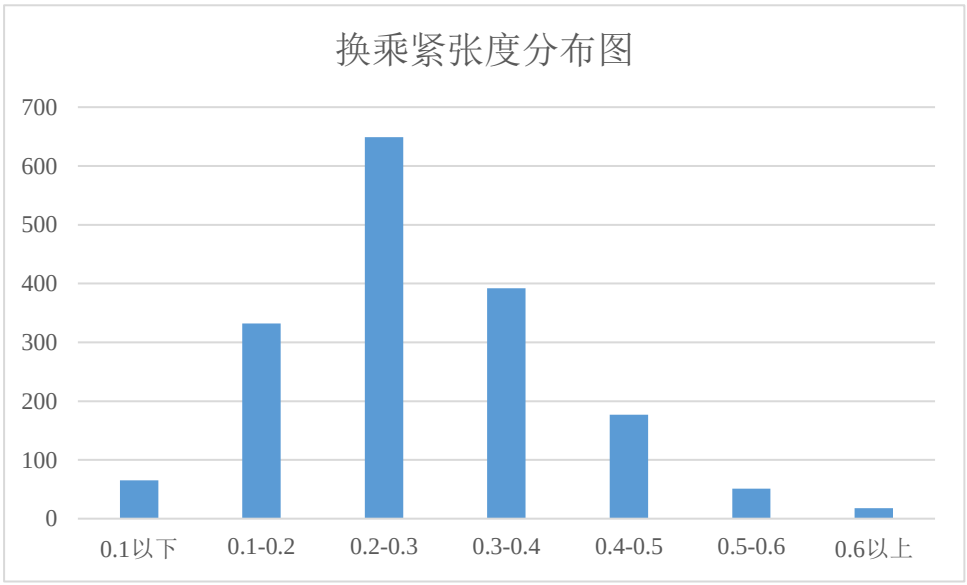

图 13 换乘紧张度分布图

从换乘紧张度来看，大部分旅客的换乘紧张度在 0.2-0.3 左右，95.9%的旅客的换乘紧张度在 0.5 以下，换乘紧张度较为合理。

从换成时间和换乘紧张度的分布来看，该模型得到的结果较为合理，有利于提高旅客的满意度。

图 15 是问题三结果的甘特图，可以清楚地看到，我们的结果满足时间间隔要求，各架次飞机间无冲突，是一个可行解。

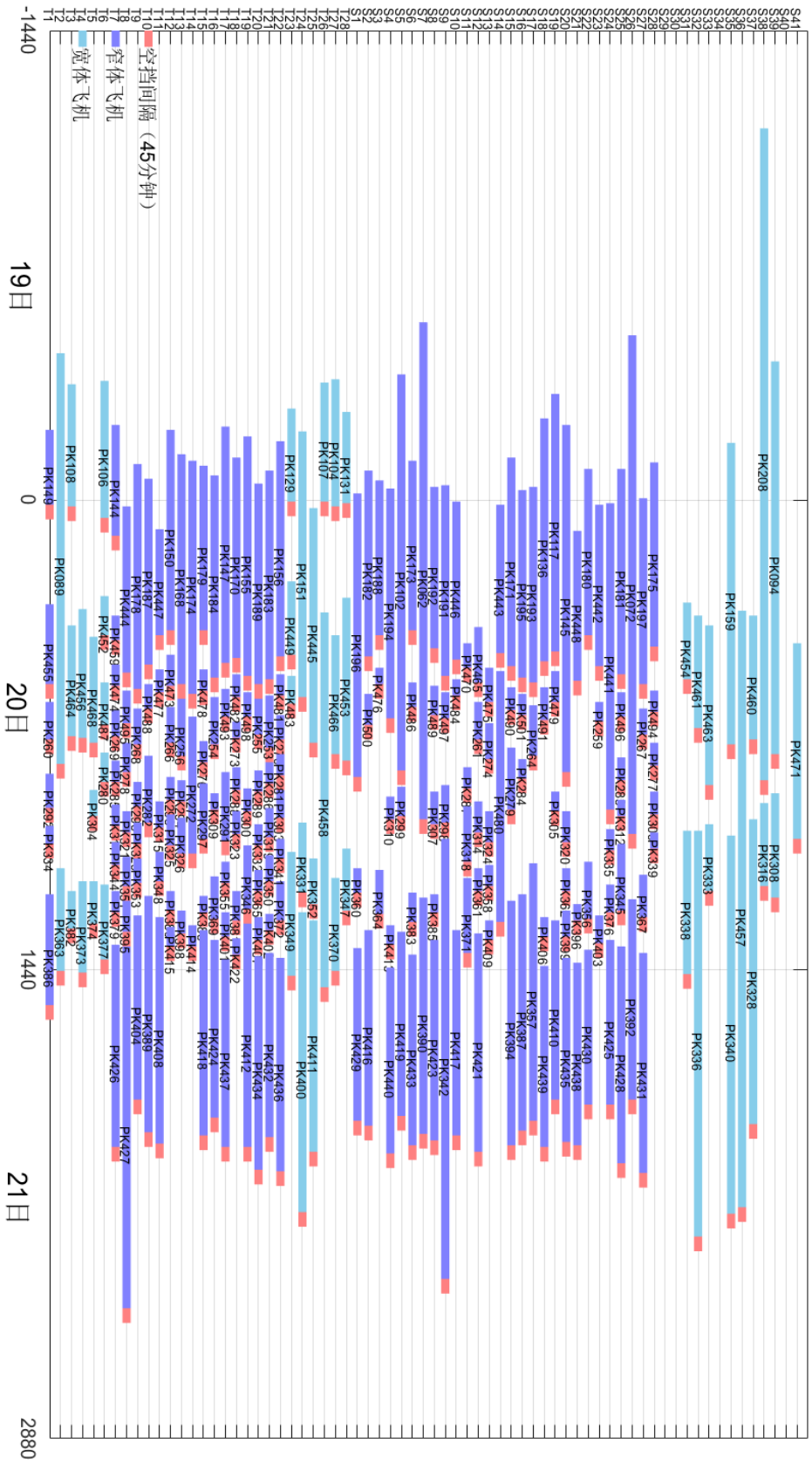

图 14 问题三登机口占用甘特图

5.3.4 结合禁忌搜索求解

停机位调度问题本身就是一个 NP-Hard 问题，当问题规模增加时，无法在短时间内求得最优解。问题三问题规模和求解复杂性增大，采用 CPLEX 求解器无法在有限时间内求得最优解。我们尝试用 CPLEX 求解 8 个小时，可以得到一个比较好的可行解。但是，在实际生活中，机场停机位调度需要在较短时间内给出结果。通过几个小时，甚至几十个小时去求得最优解显然不可取的，因此，我们尝试将智能优化方法与 CPLEX 求解结合，期望在短时间内求得比较好的可行解。

禁忌搜索是一种现代启发式算法，是一个跳脱出最优解的搜索方法。迄今为止，禁忌搜索算法在组合优化、生产调度、机器学习、电路设计等领域取得了很大的成功。在本问题中，我们采用 CPLEX 快速地为禁忌搜索生成一个可行解，然后采用禁忌搜索进行优化。这种结合的好处是，CPLEX 可以快速的在满足航班安排数最少的条件下求得一个初始解。禁忌搜索可以在固定航班数安排数量的基础上，快速对计算结果进行优化。求解过程的伪代码如下：

$TabuSearch$:

$s \leftarrow solutionGeneratedByCPLEX()$
$While\ ite\ <\ Iteration_{max}$
  $updateTabuList()$
  $innerExchangeOperation(s)$:
    $for\ flight_i\ and\ flight_j\ in\ s, i \neq j$
      $s' \leftarrow exchange(i,j)$
      $evaluation(s')$
  $outerExchangeOperation(s)$:
    $for\ flight_i\ out\ s\ and\ flight_j\ in\ s, i \neq j$
      $s' \leftarrow exchange(i,j)$
      $evaluation(s')$
  $chooseMinS(s')\ and\ operation\ not\ in\ tabuList$
end

（1）初始解的生成

算法采用 CPLEX 求解 0-1 整数规划模型求得问题三的一个满足最大化航班停靠数的可行解。算法运行时间为 720s。运行结果如下：

（2）禁忌表和藐视规则

当算法执行一次交换操作，即某架次航班 i 通过交换停靠登机口 j，记为操作O_{ij}。则 i 在 n 次迭代以内不能从登机口 j 移出。禁忌表长度 $n=10+10\rho$。ρ 为 0 到 1 的随机数。当处于禁忌表中的操作执行时，可以得到比历史最优解更好的解，则该操作不受禁忌表的限制。每次迭代之后，每项操作的禁忌表长度减 1。

（3）邻域生成

内交换操作：内交换操作定义为已安排航班之间的交换。交换已安排的两个航班停靠的登机口，得到一个新的解。交换的两个航班的登机口属性要相互满足，属于同一个登机口的两个航班不进行交换（交换得到的是相同解）。

外交换操作：外交换操作定义为移出一个已安排航班，插入一个未安排成功的航班。插入航班选择的登机口首先需要满足属性要求

（4）解的评价

解的评价为问题三种的求解目标，即最短中转旅客换乘时间。因为领域采用交换操作生成，因此可以保证解中有最大的飞机停靠数量。但是，通过交换，当前解可能变为不可行解。因此对解进行评价时，对不可行解进行惩罚。因此解的评价函数为：$f(s)=\sum_{i=1}^{n}T_i^t/T_i^c+10000\rho$。$T_i^t$ 为乘客 i 的换乘时间，T_i^c 为乘客 i 的航班连接时间，ρ 为解中冲突的航班数。

（5）移动选择

每一次迭代之后，选择邻域中评价值最小的解。

5.3.5 CPLEX 结合禁忌搜索求解结果

使用 CPLEX 生成初始解，该初始解满足飞机停靠登机口数量最大化。结合禁忌进行优化，禁忌搜索 50 代内结果未搜索到更好的解时停止运行。CPLEX 与禁忌搜索运行时间共计 732s。

该结果为 256 架次飞机安排在合适登机口，剩余 47 架次飞机前往临时停机坪，旅客换乘紧张度为 478.06，共使用 65 个登机口，换乘失败率为 0。虽然求解结果劣于 CPLEX 经过 240 分钟的求解结果，但在较短时间内取得了比较满意的解，具有更好的实用性，符合实际需求。

六、总结

针对飞机停机位调度问题，我们建立了 0-1 整数规划模型。在建立模型的过程中，为了模型的简洁和方便求解，我们对于一些复杂约束包括停机位的属性匹配和空挡间隔时间进行了预处理，将这些约束变成了 0-1 变量，加入了模型里，简化的模型同时提高了模型求解速度。0-1 整数规划模型对问题描述直观，方便 CPLEX 等求解器进行求解。

对于问题一，我们考虑了登机口的属性匹配和空挡间隔时间，优化目标为最大化航班停靠登机口数量，同时将登机口开放数量乘以一个较小的负数加入求解目标，作为优化的第二目标。CPLEX 可以在 2s 内得到最优解。总计可为最多 256 架次飞机安排登机口，共使用 65 个登机口。

对于问题二，我们在问题一的基础上，对每个航班增加了两个变量记录飞机停靠的位置，从而确定旅客的最短流程时间。问题二的规模和复杂性都有所增加，但是针对我们建立的模型，也可以在一分钟左右求得最优解，旅客中转流程总时间最小为 50025 分钟。

对于问题三，我们对每个航班增加了 7 个变量记录航班停靠的登机口区域。对每一位旅客增加了 49 个变量记录旅客最短中转时间的组合。同时航班停靠的区域会影响许多旅客的转乘紧张度。问题的复杂性和求解规模显著增大。采用 CPLEX 需要花费几个小时甚至十几个小时进行求解，而且无法保证求得问题的最优解，因此在本问题的求解中我们也给出了元启发式算法的求解策略，可以大大减少问题求解的时间，在实际应用中有更具有价值。使用 CPLEX 四小时求得乘客总的换乘紧张度为 474.45（随着事件增多可以取得更优解），CPLEX 结合禁忌搜索 732 秒求得乘客总的紧张度 478.06。

本模型也存在一些缺点：一是模型的扩展性较差。当实际问题的约束发生变化时，需要对问题进行重新建模。二是当问题规模和求解的复杂性较大时，模型求解难度比较大。无法在有限的时间内求得问题的最优解。接下在大规模问题的求解算法上，还需要进一步的研究。

七、参考文献

[1]卫东选. 基于运行安全的机场停机位分配问题研究[D]. 南京航空航天大学, 2010.
[2]文军, 孙宏, 徐杰,等. 基于排序算法的机场停机位分配问题研究[J]. 系统工程, 2004, 22(7):102-106.
[3]Bouras A, Ghaleb M A, Suryahatmaja U S, et al. The Airport Gate Assignment Problem: A Survey[J]. The Scientific World Journal,2014,(2014-11-20), 2014, 2014(6):9165-9172.
[4]Lim A, Rodrigues B, Zhu Y. Airport Gate Scheduling with Time Windows.[J]. Artificial Intelligence Review, 2005, 24(1):5-31.
[5]Yan S, Huo C M. Optimization of multiple objective gate assignments[J]. Transportation Research Part A, 2001, 35(5):413-432.
[6]Liu S, Chen W, Liu J. Optimizing airport gate assignment with operational safety constraints[C]// International Conference on Automation and Computing. IEEE, 2014:61-66.

八、附录

8.1 JAVA 调用 CPLEX API 方法：

首先从官网下载 CPLEX 安装包进行安装。安装成功后，在 eclipse 中新建一个 java 项目，右键单击项目，build path，add libraries。然后选择 user library， 刚开始里面并没有 CPLEX 库，继续点击 user libraries j 将 cplex 库添加进来。选择导入外部 jar, 选择 cplex 安装目录中的 lib 文件夹，将 cplex.jar 添加进去即可，这个项目就完成了 cplex 的调用。

8.2 代码

8.2.1 问题一、问题二代码

```
//定义数据
class Data{
	int flightNum;
	int parkNum;
	List<Flight> flightList;
	List<Deport> deportList;
	List<Passenger> passengerList;
	int[][] isTypeMatching;
	int[][] isConflict;
	int[][][][][] tranTime =
{{{{{15,0},{20,1}},{{20,1},{15,0}}},{{{35,0},{40,1}},{{40,1},{35,0}}}},{{{{35,0},{40,1}},{{40,1},{45
,2}}},{{{20,0},{30,1}},{{30,1},{20,0}}}}};;}
//定义航班
class Flight{
	String name;
	int startTime;
	int endTime;
	int arriveType;
	int leaveType;
	char planeType;
}
//定义登机口
class Deport{
	String name;
	char location;
	String area;
	int arriveType;
	int leaveType;
	char planeType;
}
//定义乘客信息
class Passenger{
	int passengerNum;
	int arriveFlight;
```

```
    int leaveFlight;
}
public class AGAPSchedule2 {

    Data data;                              //定义类Data的对象
    IloCplex model;                         //定义cplex内部类的对象
    public IloNumVar[][] xVar;          //x[i][j]表示航班i选择登机口j
    public IloNumVar[] yVar;
    public IloNumVar[] flightLocation1;
    public IloNumVar[] flightLocation2;
    double cost;                    //目标值object
    public AGAPSchedule2(Data data) {
        this.data = data;
    }

    public void build_model() throws IloException{
        //model
        model = new IloCplex();

        //variables
        xVar = new IloNumVar[data.flightNum][data.parkNum];
        yVar = new IloNumVar[data.parkNum];
        flightLocation1 = new IloNumVar[data.flightNum];
        flightLocation2 = new IloNumVar[data.flightNum];
        //定义变量的类型和取值范围
        for(int i = 0; i < data.flightNum; i++){
            for(int j = 0; j < data.parkNum; j++){
                xVar[i][j] = model.numVar(0, 1, IloNumVarType.Int, "xVar"+i+","+j);
            }
        }
        for(int i = 0; i < data.parkNum; i++){
            yVar[i] = model.numVar(0, 1, IloNumVarType.Int, "yVar"+i);
        }
        for(int i = 0; i < data.flightNum; i++){
            flightLocation1[i] = model.numVar(0, 1, IloNumVarType.Int, "flightALocation1"+i);
        }
        for(int i = 0; i < data.flightNum; i++){
            flightLocation2[i] = model.numVar(0, 1, IloNumVarType.Int, "flightALocation2"+i);
        }

        //优化目标
        IloNumExpr obj1 = model.numExpr();
        for(int i = 0; i < data.flightNum; i++){
            for(int j = 0; j < data.parkNum; j++){
```

```
                    obj1 = model.sum(obj1, xVar[i][j]);
               }
          }
          IloNumExpr obj2 = model.numExpr();
          for(int i = 0; i < data.parkNum; i++){
               obj2 = model.sum(obj2, yVar[i]);
          }
          //........问题1优化目标..............
          //model.addMaximize(model.sum(obj1, model.prod(-0.01, obj2)));
          //.............................
          IloNumExpr obj3 = model.numExpr();
          for(int i = 0; i < data.passengerList.size(); i++){
               Passenger p = data.passengerList.get(i);
               int pArrive = p.arriveFlight;
               int pLeave = p.leaveFlight;
               IloNumExpr exp= model.numExpr();
               exp = model.sum(exp,model.prod(model.prod(flightLocation1[pArrive],
flightLocation1[pLeave]),data.tranTime[1-data.flightList.get(pArrive).arriveType][1-
data.flightList.get(pLeave).leaveType][0][0][0]));
               exp = model.sum(exp,model.prod(model.prod(flightLocation1[pArrive],
flightLocation2[pLeave]),data.tranTime[1-data.flightList.get(pArrive).arriveType][1-
data.flightList.get(pLeave).leaveType][0][1][0]));
               exp = model.sum(exp,model.prod(model.prod(flightLocation2[pArrive],
flightLocation1[pLeave]),data.tranTime[1-data.flightList.get(pArrive).arriveType][1-
data.flightList.get(pLeave).leaveType][1][0][0]));
               exp = model.sum(exp,model.prod(model.prod(flightLocation2[pArrive],
flightLocation2[pLeave]),data.tranTime[1-data.flightList.get(pArrive).arriveType][1-
data.flightList.get(pLeave).leaveType][1][1][0]));
               obj3 = model.sum(obj3, model.prod(exp, p.passengerNum));
          }
          //.......问题2优化目标........
          model.addMinimize(model.sum(model.sum(obj3, model.prod(0.01, obj2)),model.prod(100000,
model.diff(256, obj1))));

          //约束1：属性匹配
          for(int i = 0; i < data.flightNum; i++){
               for(int j = 0; j < data.parkNum; j++){
                    if(data.isTypeMatching[i][j] == 0)
                         model.addEq(xVar[i][j], 0);
               }
          }
          //约束2：每个航班只能选择一个登机口
          for(int i = 0; i < data.flightNum; i++){
               IloNumExpr exp1 = model.numExpr();
```

```
                for(int j = 0; j < data.parkNum; j++){
                    exp1 = model.sum(exp1, xVar[i][j]);
                }
                model.addLe(exp1, 1);
            }
            //约束3：间隔时间小于45分钟
            for(int i = 0; i < data.flightNum; i++){
                for(int j = 0; j < data.flightNum; j++){
                    if(i == j)
                        continue;
                    for(int k = 0; k < data.parkNum; k++){
                        if(data.isTypeMatching[i][k] == 1 && data.isTypeMatching[j][k] == 1 &&
data.isConflict[i][j] == 1)
                            model.addLe(model.sum(xVar[i][k], xVar[j][k]), 1);
                    }
                }
            }
            //约束4：只能选择开放的登记口
            for(int i = 0; i < data.flightNum; i++){
                for(int j = 0; j < data.parkNum; j++){
                    model.addLe(xVar[i][j], yVar[j]);
                }
            }
            //约束5：记录每个航班的位置匹配情况
            for(int i = 0; i < data.flightNum; i++){
                IloNumExpr exp2 = model.numExpr();
                IloNumExpr exp3 = model.numExpr();
                for(int j = 0; j < 28; j++){
                    exp2 = model.sum(exp2, xVar[i][j]);
                }
                for(int j = 28; j < data.parkNum; j++){
                    exp3 = model.sum(exp3, xVar[i][j]);
                }
                model.addEq(flightLocation1[i], exp2);
                model.addEq(flightLocation2[i], exp3);
            }
        }
        public static void process_AGAPSchedule(Data data) throws FileNotFoundException{
            String line = null;
            String[] substr = null;
            Scanner cin = new Scanner(new BufferedReader(new
FileReader("D://data//AGAP//flight.csv")));  //读取文件
            line = cin.nextLine();//读取一行
            //初始化参数
```

```
data.flightNum = 303;
data.parkNum = 69;
data.deportList = new ArrayList<Deport>();
data.flightList = new ArrayList<Flight>();
data.passengerList = new ArrayList<Passenger>();
data.isTypeMatching = new int[data.flightNum][data.parkNum];
data.isConflict = new int[data.flightNum][data.flightNum];
while(cin.hasNextLine()){
     line = cin.nextLine();
     substr = line.split((","));
     Flight flight = new Flight();
     flight.name = substr[0];
     if(substr[1].charAt(0) == '-'){
          String number = substr[1].substring(1);
          flight.startTime = -1 * Integer.parseInt(number);
     }
     else
          flight.startTime = Integer.parseInt(substr[1]);
     flight.arriveType = Integer.parseInt(substr[3]);
     flight.endTime = Integer.parseInt(substr[5]);
     flight.leaveType = Integer.parseInt(substr[7]);
     flight.planeType = substr[10].charAt(0);
     data.flightList.add(flight);
}
cin.close();//关闭流
cin = new Scanner(new BufferedReader(new FileReader("D://data//AGAP//InputData.csv")));
cin.nextLine();
while(cin.hasNextLine()){
     line = cin.nextLine();
     substr = line.split((","));
     Deport deport = new Deport();
     deport.name = substr[0];
     deport.location = substr[1].charAt(0);
     deport.area = substr[2];
     deport.arriveType = Integer.parseInt(substr[3]);
     deport.leaveType = Integer.parseInt(substr[4]);
     deport.planeType = substr[5].charAt(0);
     data.deportList.add(deport);
}
cin.close();

cin = new Scanner(new BufferedReader(new FileReader("D://data//AGAP//passenger.csv")));
cin.nextLine();
while(cin.hasNextLine()){
```

```
                line = cin.nextLine();
                substr = line.split((","));
                Passenger passenger = new Passenger();
                passenger.passengerNum = Integer.parseInt(substr[0]);
                passenger.arriveFlight = Integer.parseInt(substr[1]);
                passenger.leaveFlight = Integer.parseInt(substr[2]);
                data.passengerList.add(passenger);
            }
            cin.close();
            for(int i = 0; i < data.flightList.size(); i++){
                for(int j = 0; j < data.deportList.size(); j++){
                    if((data.flightList.get(i).arriveType == data.deportList.get(j).arriveType ||
data.deportList.get(j).arriveType == 2) &&
                                    (data.flightList.get(i).leaveType ==
data.deportList.get(j).leaveType || data.deportList.get(j).leaveType == 2) &&
                                        data.flightList.get(i).planeType ==
data.deportList.get(j).planeType)
                        data.isTypeMatching[i][j] = 1;
                }
            }
            for(int i = 0; i < data.flightList.size(); i++){
                for(int j = 0; j < data.flightList.size(); j++){
                    if(i == j)
                        continue;
                    Flight flight1 = data.flightList.get(i);
                    Flight flight2 = data.flightList.get(j);
                    data.isConflict[i][j] = 1;
                    if(flight1.endTime + 45 <= flight2.startTime)
                        data.isConflict[i][j] = 0;
                    if(flight1.startTime - 45 >= flight2.endTime)
                        data.isConflict[i][j] = 0;
                }
            }
        }
        public static void main(String[] args) throws Exception {
            Data data = new Data();
            process_AGAPSchedule(data);
            System.out.println("input succesfully");
            System.out.println("cplex procedure############################");
            AGAPSchedule2 cplex = new AGAPSchedule2(data);
            cplex.build_model();
            double cplex_time1 = System.nanoTime();
            cplex.solve();
            double cplex_time2 = System.nanoTime();
```

```
            double cplex_time = (cplex_time2 - cplex_time1) / 1e9;//求解时间，单位s
            System.out.println("cplex_time " + cplex_time + " bestcost " + cplex.cost);
      }
}
```

8.2.2 问题三：数据和模型部分

```
//定义数据
class Data{
      int flightNum;
      int parkNum;
      List<Flight> flightList;
      List<Deport> deportList;
      List<Passenger> passengerList;
      int[][] isTypeMatching;
      int[][] isConflict;
      int[][][][] tranTime =
{{{{15,28},{28,15}},{{35,48},{48,35}}},{{{35,48},{48,61}},{{20,38},{38,20}}}};;
      int[][] tranbTimeArea =
{{10,15,20,25,20,25,25},{15,10,15,20,15,20,20},{20,15,10,25,20,25,25},{25,20,25,10,15,20,20},{20,15,
20,15,10,15,15},
                  {25,20,25,20,15,10,20},{25,20,25,20,15,20,10}};
      int[][][][] tranTimeSum;
      int[][] isAreaMatching;
}

public void build_model() throws IloException{
            //model
            model = new IloCplex();
            //variables
            xVar = new IloNumVar[data.flightNum][data.parkNum];
            yVar = new IloNumVar[data.parkNum];
            flightArea = new IloNumVar[7][data.flightNum];
            flightAreaPassenger = new IloNumVar[data.passengerList.size()][7][7];
            //定义变量的类型和取值范围
            for(int i = 0; i < data.flightNum; i++){
                  for(int j = 0; j < data.parkNum; j++){
                        xVar[i][j] = model.numVar(0, 1, IloNumVarType.Int, "xVar"+i+","+j);
                  }
            }
            for(int i = 0; i < data.parkNum; i++){
                  yVar[i] = model.numVar(0, 1, IloNumVarType.Int, "yVar"+i);
            }
            for(int i = 0; i < 7; i++){
                  for(int j = 0; j < data.flightNum; j++){
                        flightArea[i][j] = model.numVar(0, 1, IloNumVarType.Int, "flightArea"+i+","+j);
```

```
                }
            }
            for(int p = 0; p < data.passengerList.size(); p++){
                for(int i = 0; i < 7; i++){
                    for(int j = 0; j < 7; j++){
                        flightAreaPassenger[p][i][j] = model.numVar(0, 1, IloNumVarType.Int,
"flightAreaPassenger"+p+","+i+","+j);
                    }
                }
            }
            //优化目标
            IloNumExpr obj1 = model.numExpr();
            for(int i = 0; i < data.flightNum; i++){
                for(int j = 0; j < data.parkNum; j++){
                    obj1 = model.sum(obj1, xVar[i][j]);
                }
            }
            IloNumExpr obj2 = model.numExpr();
            for(int i = 0; i < data.parkNum; i++){
                obj2 = model.sum(obj2, yVar[i]);
            }
            //model.addMaximize(model.sum(obj1, model.prod(-0.01, obj2)));

            IloNumExpr tranTimeDegreeSum = model.numExpr();
            for(int i = 0; i < data.passengerList.size(); i++){
                Passenger p = data.passengerList.get(i);
                int pArrive = p.arriveFlight;
                int pLeave = p.leaveFlight;
                IloNumExpr exp= model.numExpr();
                IloNumExpr expSum= model.numExpr();
                for(int m = 0; m < 7; m++){
                    for(int n = 0; n < 7; n++){
                        if(data.isAreaMatching[pArrive][m] == 1 && data.isAreaMatching[pLeave][n]
== 1){
                            expSum = model.sum(expSum, model.prod(flightAreaPassenger[i][m][n],
data.tranTimeSum[1-data.flightList.get(pArrive).arriveType][1-
data.flightList.get(pLeave).leaveType][m][n]));
                        }
                    }
                }
                tranTimeDegreeSum = model.sum(tranTimeDegreeSum, model.prod(model.prod(model.sum(exp,
expSum), 1.0 / p.connectTime), p.passengerNum));
            }
            model.addMinimize(model.sum(model.sum(model.prod(10,tranTimeDegreeSum), model.prod(0.01,
```

```
obj2)),model.prod(100000, model.diff(256, obj1))));

            //约束1：属性匹配
            for(int i = 0; i < data.flightNum; i++){
                 for(int j = 0; j < data.parkNum; j++){
                      if(data.isTypeMatching[i][j] == 0)
                           model.addEq(xVar[i][j], 0);
                 }
            }
            //约束2：每个航班只能选择一个登机口
            for(int i = 0; i < data.flightNum; i++){
                 IloNumExpr exp1 = model.numExpr();
                 for(int j = 0; j < data.parkNum; j++){
                      exp1 = model.sum(exp1, xVar[i][j]);
                 }
                 model.addLe(exp1, 1);
            }
            //约束3：间隔时间小于45分钟
            for(int i = 0; i < data.flightNum; i++){
                 for(int j = 0; j < data.flightNum; j++){
                      if(i == j)
                           continue;
                      for(int k = 0; k < data.parkNum; k++){
                           if(data.isTypeMatching[i][k] == 1 && data.isTypeMatching[j][k] == 1 &&
data.isConflict[i][j] == 1)
                                model.addLe(model.sum(xVar[i][k], xVar[j][k]), 1);
                      }
                 }
            }
            //约束4：只能选择开放的登记口
            for(int i = 0; i < data.flightNum; i++){
                 for(int j = 0; j < data.parkNum; j++){
                      model.addLe(xVar[i][j], yVar[j]);
                 }
            }
            //约束5：记录每个航班的区域匹配情况
            for(int i = 0; i < data.flightNum; i++){
                 IloNumExpr exp1 = model.numExpr();
                 IloNumExpr exp2 = model.numExpr();
                 IloNumExpr exp3 = model.numExpr();
                 IloNumExpr exp4 = model.numExpr();
                 IloNumExpr exp5 = model.numExpr();
                 IloNumExpr exp6 = model.numExpr();
                 IloNumExpr exp7 = model.numExpr();
```

```
                for(int j = 0; j < 9; j++){
                    exp1 = model.sum(exp1, xVar[i][j]);
                }
                for(int j = 9; j < 19; j++){
                    exp2 = model.sum(exp2, xVar[i][j]);
                }
                for(int j = 19; j < 28; j++){
                    exp3 = model.sum(exp3, xVar[i][j]);
                }
                for(int j = 28; j < 38; j++){
                    exp4 = model.sum(exp4, xVar[i][j]);
                }
                for(int j = 38; j < 48; j++){
                    exp5 = model.sum(exp5, xVar[i][j]);
                }
                for(int j = 48; j < 58; j++){
                    exp6 = model.sum(exp6, xVar[i][j]);
                }
                for(int j = 58; j < data.parkNum; j++){
                    exp7 = model.sum(exp7, xVar[i][j]);
                }
                model.addEq(flightArea[0][i], exp1);
                model.addEq(flightArea[1][i], exp2);
                model.addEq(flightArea[2][i], exp3);
                model.addEq(flightArea[3][i], exp4);
                model.addEq(flightArea[4][i], exp5);
                model.addEq(flightArea[5][i], exp6);
                model.addEq(flightArea[6][i], exp7);
            }
            //约束6：确定每个旅客的转换时间
            for(int p = 0; p < data.passengerList.size(); p++){
                Passenger passenger = data.passengerList.get(p);
                int pArrive = passenger.arriveFlight;
                int pLeave = passenger.leaveFlight;
                for(int i = 0; i < 7; i++){
                    for(int j = 0; j < 7; j++){
                        if(data.isAreaMatching[pArrive][i] == 1 && data.isAreaMatching[pLeave][j]
== 1){
                            model.addLe(model.diff(model.sum(flightArea[i][pArrive],
flightArea[j][pLeave]),flightAreaPassenger[p][i][j]), 1);
                        }
                    }
                }
            }
```

```
    }
```

8.2.3 问题三：禁忌搜索算法实现部分

```
public int[] search(int[] solution){
        //记录历史最优解
        int[] bestSolution = new int[data.flightNum];
        for(int i = 0; i < data.flightNum; i++){
            bestSolution[i] = solution[i];
        }
        double bestCost = evaluation(solution);
        int iteration = 0;
        int[][] tabuList = new int[data.flightNum][data.parkNum];
        List<List<Integer>> departList = new ArrayList<List<Integer>>();
        for(int i = 0; i < data.parkNum; i++){
            List<Integer> list = new ArrayList<Integer>();
            departList.add(list);
        }
        for(int i = 0; i < data.flightNum; i++){
            if(solution[i] >= 0)
                departList.get(solution[i]).add(i);
        }
        double cost = bestCost;
        while(iteration < 100){
            //更新禁忌表
            for(int i = 0; i < data.flightNum; i++){
                for(int j = 0; j < data.parkNum; j++){
                    if(tabuList[i][j] > 0)
                        tabuList[i][j] = tabuList[i][j] - 1;
                }
            }
            //解外交换操作
            int exchangeA = 0;
            int exchangeB = 0;
            int exchangePark = 0;
            boolean isinnerExchange = false;
            double minCostChange = 1000000000;
            Random random = new Random();
            for(int i = 0; i < data.flightNum; i++){
                if(solution[i] < 0){
                    for(int j = 0; j < data.parkNum; j++){
                        if(data.isTypeMatching[i][j] == 0)
                            continue;
                        for(int p = 0; p < departList.get(j).size(); p++){
                            int q =  departList.get(j).get(p);
                            solution[q] = -1;
```

```
                    solution[i] = j;
                    double costChange = evaluation(solution);

                    if(costChange - cost < minCostChange){
                        if((tabuList[i][j] > 0) && costChange > bestCost)
                            continue;
                        exchangeA = i;
                        exchangeB = q;
                        exchangePark = j;
                        minCostChange = costChange - cost;
                    }
                    solution[q] = j;
                    solution[i] = -1;
                }
            }
        }
    }
    //解内交换操作
    for(int i = 0; i < data.flightNum; i++){
        if(solution[i] >= 0){
            for(int j = 0; j < data.parkNum; j++){
                if( solution[i] == j)
                    continue;
                if(data.isTypeMatching[i][j] == 0)
                    continue;
                for(int p = 0; p < departList.get(j).size(); p++){
                    int q =  departList.get(j).get(p);
                    if(data.isTypeMatching[q][solution[i]] == 0 )
                        continue;
                    solution[q] = solution[i];
                    solution[i] = j;
                    double costChange = evaluation(solution);
                    if(costChange - cost < minCostChange){
                        if((tabuList[i][j] > 0) && costChange > bestCost)
                            continue;
                        exchangeA = i;
                        exchangeB = q;
                        isinnerExchange = true;
                        minCostChange = costChange - cost;
                    }
                    solution[i] = solution[q];
                    solution[q] = j;
                }
            }
```

```
                    }
               }
               if(isinnerExchange){
                    for(int i = 0;i < departList.get(solution[exchangeA]).size();i++){
                         if(departList.get(solution[exchangeA]).get(i) == exchangeA)
                              departList.get(solution[exchangeA]).remove(i);
                    }
                    departList.get(solution[exchangeA]).add(exchangeB);
                    for(int i = 0;i < departList.get(solution[exchangeB]).size();i++){
                         if(departList.get(solution[exchangeB]).get(i) == exchangeB)
                              departList.get(solution[exchangeB]).remove(i);
                    }
                    departList.get(solution[exchangeB]).add(exchangeA);
                    int index = solution[exchangeA];
                    solution[exchangeA] = solution[exchangeB];
                    solution[exchangeB] = index;
               }
               else{
                    solution[exchangeB] = -1;
                    solution[exchangeA] = exchangePark;
                    for(int i = 0;i < departList.get(exchangePark).size();i++){
                         if(departList.get(exchangePark).get(i) == exchangeB)
                              departList.get(exchangePark).remove(i);
                    }
                    departList.get(exchangePark).add(exchangeA);
               }
               cost = cost + minCostChange;
               tabuList[exchangeA][exchangePark] = random.nextInt(10)+10;
               //tabuList[exchangeB][exchangePark] = 5;
               //更新最优
               if(cost < bestCost){
                    for(int i = 0; i < data.flightNum; i++){
                         bestSolution[i] = solution[i];
                    }
                    bestCost = cost;
               }

               cost = evaluation(solution);
               System.out.println(cost);
               iteration++;
          }
          return bestSolution;
     }
```