“华为杯”第十五届中国研究生数学建模竞赛

学　　校	上海师范大学
参赛队号	18102700047
队员姓名	1.刘盼攀
	2.徐　奥
	3.李　想

“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评估方法

摘　　要：

本文研究了机场新增卫星厅后的航班-登机口优化问题，中转旅客流程时间最短问题/总体紧张度最小问题和登机口的使用量最小问题的多级目标规划问题，使得机场在尽量减少开支的情况下，尽量满足旅客的需求.

针对问题一，我们求解的一级目标是尽可能多的分配航班到合适的登机口，二级目标是使登机口的使用数量最小，在根据一些条件的约束下（包括机体类别的约束，停机规则的约束，登机口规则的约束以及时间的约束），我们得到了航班-登机口优化问题的模型.并且根据模拟退火算法，利用 C++得到如表 1 所示结论，并且我们又用 matlab 画出成功分配到登机口的航班数量和比例对比图，T 和 S 登机口的使用数目和被使用登机口的平均使用率对比图，如图 6.3 和图 6.4 所示.

表 1 问题一的结论

临时机位使用数量	被使用的登机口的数量	分配到登机口的航班数	登机口使用比例	窄体机数量	窄体机数量的使用比率
84	67	442	0.72459	366	0.717647
宽体机数量	宽体机数量的使用比率	T 登机口使用数量	S 登机口使用数量	T 登机口的平均使用率	S 登机口的平均使用率
76	0.76	28	39	0.15062	0.16699

针对问题二，我们是在问题一的基础上，考虑了旅客的换乘时间（不考虑捷运时间和步行时间），仍然是以尽可能多的分配航班到合适的登机口为一级目标，但二级目标改成中转旅客流程时间最短，三级目标为登机口的使用数量最小；并且约束条件同问题一. 因此我们同样也利用模拟退火算法和 C++求得如表 2 所示结论.并且我们又用 matlab 画出成功分配到登机口的航班数量和比例对比图，T 和 S 登机口的使用数目和被使用登机口的平均使用率对比图，如图 6.5 和 6.6 所示.

表 2 问题二的结论

临时机位使用数量	中转旅客总流程时间	被使用的登机口的数量	分配到登机口的航班数	登机口使用比例	窄体机数量	窄体机数量的使用比率
87	45850	67	468	0.728772	388	0.723388
宽体机数量	宽体机数量的使用比率	T 登机口使用数量	S 登机口使用数量	T 登机口的平均使用率	S 登机口的平均使用率	换乘旅客数量和失败率
80	0.754717	28	39	0.136105	0.153577	均为 0

针对问题三，我们是在问题一的基础上，并对问题二进行细化，加入捷运时间和步行时间，并且仍然以尽可能多的分配航班到合适的登机口为一级目标，而二级成中转旅客的总体紧张度最小，三级目标仍为登机口的使用数量最小；并且约束条件同问题一.因此我们同样也利用模拟退火算法和 C++求得如表 3 所示结论；并且我们又用 matlab 画出成功分配到登机口的航班数量和比例对比图，T 和 S 登机口的使用数目和被使用登机口的平均使用率对比图，如图 6.7 和 6.8 所示.总体旅客换乘时间分布图和总体旅客紧张度分布图如图 6.9 所示.

表 3 问题三的结论

临时机位使用数量	中转旅客总紧张度	被使用的登机口的数量	分配到登机口的航班数	登机口使用比例	窄体机数量	窄体机数量的使用比率
91	814.799	67	460	0.716511	380	0.708955
宽体机数量	宽体机数量的使用比率	T 登机口使用数量	S 登机口使用数量	T 登机口的平均使用率	S 登机口的平均使用率	换乘旅客数量和失败率
80	0.754717	28	39	0.125203	0.158712	均为 0

本文建立出的模型并没有过多的理想化，而是很贴近现实，从而本文的模型具有很大的实用性，可以广泛的运用.

关键词：航班-登机口优化；多级目标规划；模拟退火算法；

一、问题重述

1.1 问题背景

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

飞机在机场廊桥（登机口）的一次停靠通常由一对航班（到达航班和出发航班，也叫“转场”）来标识. 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. 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.

1.2 问题提出

登机口分配问题建立数学优化模型，并通过求解这些模型，进行数据评估分析. 本题的所有建模和数据分析都是针对航站楼 T 和卫星厅 S 同时使用的情形. 需要解决的问题如下：

问题一：航班-登机口分配最优问题

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

问题二：中转旅客流程时间最短问题

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

问题三：中转旅客的换乘时间总体紧张度的最小问题

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. 本题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.

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

旅客换乘时间=最短流程时间+捷运时间+行走时间

航班连接时间=后一航班出发时间-前一航班到达时间

其中，不同终端厅之间的最短流程时间（单位：分钟）和坐捷运次数（单位：次）如表 1.1 所示.

表 1.1 不同终端厅之间的最短流程时间和坐捷运次数

到达 \ 出发		国内出发（D）		国际出发（I）	
		航站楼 T	卫星厅 S	航站楼 T	卫星厅 S
国内到达（D）	航站楼 T	15/0	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

登机口区域之间的行走时间如表 1.2 所示（单位：分钟，捷运乘坐时间需另行计算）

表 1.2 登机口区域之间的行走时间

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

本文主要研究航班-登机口优化分配问题，首先只考虑尽可能多的把航班分配到合适的登机口，其次考虑旅客换乘的总体流程时间，最后进一步的考虑旅客换乘总体紧张度，使得登机口的使用数量最小.设置出既使机场的设施使用率最高，减少浪费，又秉承顾客是上帝的思想，使得尽可能的使旅客能够顺利登机并且不需要很慌忙的模型.下面我们对每个问题做出具体的分析.

2.1 问题一的分析

对于问题一，我们只需要考虑航班-登机口的最优分配问题，即给飞机尽可能多的分配到合适的登机口（即尽可能少的把飞机分配到临时机位），然后使登机口的使用数量最少.对于飞机的停靠，有很多种类型的约束，包括

机体类别（N/W）的约束：332,333,33E,33H,33L,773 是宽体机（W）,319,320, 321, 323, 325, 738, 73A, 73E, 73H, 73L 是窄体机（N），对应飞机型号的飞机只能停在对应的机场；

停机规则的约束：一架飞机只能停在一个登机口；

登机口规则的约束：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；一个登机口同个时刻只能服务一架飞机；出发类型和到达类型与登机口的类型必须匹配；

时间的约束：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；

根据这些约束和目标，我们就可以得到相应的模型.

2.2 问题二的分析

对于问题二，我们又考虑中转旅客最短流程时间.根据题设的要求，我们要在问题一的基础上，主要求出所有中转旅客的换乘时间（不考虑乘坐捷运和步行时间）之和，而旅客的换乘时间跟旅客所乘坐飞机的机体类别（N/W）、停靠的登机口区域（$T-North, T-Center, T-South, S-North, S-Center, S-South, S-East$）和是否换乘成功等有关，所以关键是要判断出飞机停在哪个登机口区域，然后就可以计算出总换乘时间，从而就可以得到目标函数.而问题二在问题

一的基础上并没有增加或者减少约束条件，从而约束条件还是问题一的约束条件，进而得到相应的模型.

2.3 问题三的分析

对于问题三，我们是在问题二的基础上进行细化，并且考虑乘坐捷运时间和步行时间，计算出乘客的总体换乘时间.所以关键问题还是需要判断出旅客所乘坐飞机的机体类别（N/W）、飞机停靠的登机口区域（$T-North, T-Center,$ $T-South, S-North, S-Center, S-South, S\text{-}East$) 和是否换乘成功等，从而求出换乘时间（包括流程时间、行走时间和捷运时间），并且计算出旅客乘坐的相应两趟航班之间的连接时间，从而求出总体紧张度.

我们将三个问题的分析过程表示成图 2.1.

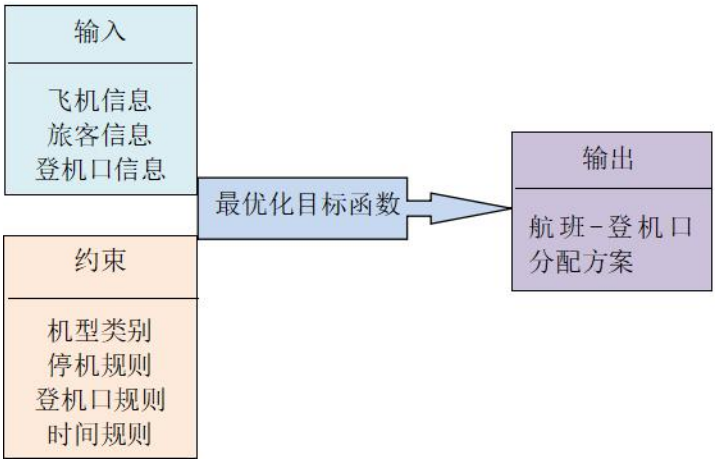


图 2.1 问题的分析过程

三、模型假设

1、假设旅客上、下飞机时间忽略不计；
2、假设旅客达到捷运站时就可以搭上捷运，并且立即发车；
3、假设临时机位数量无限；
4、在问题二中假设旅客的换乘时间中不考虑旅客乘坐捷运时间和步行时间；
5、在问题二、三中假设换乘失败的旅客的换乘时间相当于 6 小时；
6、在问题二、三中假设航班号无对应记录的旅客忽略不计；

四、符号说明

1、集合的定义

符号	符号的意义
N	到达至离开机场的日期与 20 日有重叠的飞机所构成的集合
K	机场可利用的登机口所构成的集合

K_i	由于类型的限制，无法接待飞机i的登机口所构成的集合$(i \in N)$
P	20 日到达或离开机场且能在 Pucks 中检索到对应航班号的所有旅客组所构成的集合

2、参数的定义

符号	符号的意义
n	集合N的元素的个数$\lvert N \rvert$
m	机场可利用的登机口的总数$\lvert K \rvert$,这里等于 69
$m+1$	临时机位的编号，本文中称之为登机口$m+1$
T_i	飞机i的到达时刻$(i \in N)$
T_i'	飞机i的出发时刻$(i \in N)$
β	同一个登机口的飞机的出发时刻与下一架飞机的到达时刻之间的缓冲时间，这里为 45 分钟
M	一个足够大的数,这里取 1000
a_p	第p组旅客的到达航班对应的飞机号（$p \in P$）
d_p	第p组旅客的出发航班对应的飞机号（$p \in P$）
D_i	$\begin{cases}1,\text{飞机}i\text{到达类型是D}\\0,\text{飞机}i\text{到达类型是}I\end{cases}$（$i \in N$）
D_i'	$\begin{cases}1,\text{飞机}i\text{出发类型为}D\\0,\text{飞机}i\text{出发类型为}I\end{cases}$（$i \in N$）
F	最短流程时间矩阵，为 4 阶方阵
n_p	第p组旅客的人数（$p \in P$）
M	登机口区域之间的行走时间矩阵，为 7 阶方阵
F'	登机口区域的捷运次数矩阵，为 4 阶方阵
N'	$N \cup (\bigcup_{p \in P} a_p) \cup (\bigcup_{p \in P} d_p)$，即到达至离开机场的日期与 20 日有重叠的飞机和 20 日到达或离开机场且能在 Pucks 中检索到对应航班号的所有旅客组所乘坐的前后两趟飞机所构成的集合

3、变量的定义

符号	符号的意义
$y_{i,k}$	$\begin{cases}1, & \text{飞机}i\text{停在登机口}k\ (i\in N,k=1,2,\cdots,m+1)\\0, & \text{其他}\end{cases}$
Y	$(y_{i,k})_{n\times(m+1)}$，是以 $y_{i,k}$ 为元素的 $n\times(m+1)$ 矩阵
$z_{i,j,k}$	$\begin{cases}1, & \text{飞机}i\text{和}j\text{均被分配到登机口}k\text{且飞机}j\text{紧接在}i\text{之后}\ (i,j\in N,k\in K)\\0, & \text{其他}\end{cases}$
σ_i	飞机 i 的到达类型，是一个 4 维向量 (DT,DS,IT,IS) $(i\in N)$
σ_i'	飞机 i 的出发类型，是一个 4 维向量 (DT,DS,IT,IS) $(i\in N)$
t_p	第 p 组旅客的流程时间
S_p	$\begin{cases}1, & T_{a_p}+t_p\le T'_{d_p}\text{（即问题二中第}p\text{组旅客换乘成功},p\in P)\\0, & \text{其他}\end{cases}$
γ_i	飞机 i 所在的登机口区域，是一个 7 维 0-1 向量 (TN,TC,TS,SN,SC,SS,SE)
w_p	第 p 组旅客的步行时间 $(p\in P)$
t'_p	第 p 组旅客的捷运时间 $(p\in P)$
r_p	第 p 组旅客的换乘时间，即 $r_p=t_p+t'_p+w_p$ $(p\in P)$
S'_p	$\begin{cases}1, & T_{a_p}+r_p\le T'_{d_p}\text{(问题三中的第}p\text{组旅客换乘成功，}p\in P)\\0, & \text{其他}\end{cases}$
C_p	第 p 组旅客的换乘紧张度 $(p\in P)$
o	问题二中换乘失败的旅客数量
o'	问题三中换乘失败的旅客数量
b	问题二中换乘失败的旅客比率
b'	问题三中换乘失败的旅客比率

注：（1）$DT=\begin{cases}1,\text{飞机到达类型（或出发类型）是}D\text{且停在航站台}T\\0,\text{其他}\end{cases}$；

$$DS=\begin{cases}1,\ \text{飞机到达类型（或出发类型）是}D\text{且停在卫星厅}S\\0,\ \text{其他}\end{cases};$$

$$IT=\begin{cases}1,\ \text{飞机到达类型（或出发类型）是}I\text{且停在航站楼}T\\0,\ \text{其他}\end{cases};$$

$$IS=\begin{cases}1,\ \text{飞机到达类型（或出发类型）是}I\text{且停在卫星厅}S\\0,\ \text{其他}\end{cases};$$

（2）

$$TN=\begin{cases}1,\text{飞机停在登机口区域}T-North\\0,\text{其他}\end{cases};$$

$$TC=\begin{cases}1,\text{飞机停在登机口区域}T-Center\\0,\text{其他}\end{cases};$$

$$TS=\begin{cases}1,\text{飞机停在登机口区域}T-South\\0,\text{其他}\end{cases};$$

$$SN=\begin{cases}1,\text{飞机停在登机口区域}S-North\\0,\text{其他}\end{cases};$$

$$SC=\begin{cases}1,\text{飞机停在登机口区域}S-Center\\0,\text{其他}\end{cases};$$

$$SS=\begin{cases}1,\text{飞机停在登机口区域}S-South\\0,\text{其他}\end{cases};$$

$$SE=\begin{cases}1,\text{飞机停在登机口区域}S-East\\0,\text{其他}\end{cases}.$$

（3）如果飞机停在临时机位，则向量$\sigma_i,\sigma_i',\gamma_i$均为零向量；

五、模型的建立

5.1 问题一模型的建立

由于本模型考虑航班-登机口分配问题，故我们不需要考虑旅客，只需要考虑尽可能多的把航班分配到合适的登机口，再使登机口的使用数量最小化，这是多级目标规划问题.下面我们来建立问题一模型.

5.1.1 目标函数

如果记

$$y_{i,k}=\begin{cases}1,\ \text{飞机}i\text{停在登机口}k(i\in N,k=1,2,\cdots,m+1)\\0,\ \text{其他}\end{cases},$$

要使尽可能多的航班分配到合适的登机口，从而只能尽可能少的分配到临时机位，即临时机位上的飞机的总和最少，因此该模型的一级目标函数为

$$\min \sum_{i \in N} y_{i,m+1}. \tag{5.1}$$

而在此基础上，由于临时机位是简易的，只有在飞机分配不到合适的登机口的时候才能使用，因此，要使登机口被使用的数量最少，我们需要引进符号函数 sgn ，其中

$$\operatorname{sgn} f(x) = \begin{cases} 1, f(x) > 0 \\ 0, f(x) = 0 \\ -1, f(x) < 0 \end{cases}.$$

而同一个时刻，每个登机口最多只能服务一架飞机，每架飞机也只能停在一个登机口，从而该模型的二级目标函数为

$$\min \sum_{k=1}^{m} \operatorname{sgn}(\sum_{i \in N} y_{i,k}). \tag{5.2}$$

5.1.2 约束条件

根据 $y_{i,k}$ 的定义，有

$$y_{i,k} \in \{0,1\}, \forall i \in N, k = 1,2,\cdots,m+1. \tag{5.3}$$

由于一架飞机只能停在一个登机口或临时机位, 从而

$$\sum_{k=1}^{m+1} y_{i,k} = 1, \forall i \in N. \tag{5.4}$$

如果记 $Y = (y_{i,k})_{n \times (m+1)}$, 那么矩阵 Y 的每一行有且仅有一个 1. 而对于有些飞机，由于机型的限制，只有部分登机口才可以停靠. 如果我们把飞机 i 不能分配到的登机口所构成的集合记为 K_i , 从而有

$$y_{i,k} = 0, \forall i \in N, k \in K_i, \tag{5.5}$$

如果又记

$$z_{i,j,k} = \begin{cases} 1, \text{飞机} i \text{和} j \text{都被分配到登机口} k \text{且飞机} j \text{紧接在} i \text{之后} (i, j \in N, k = 1,2,\cdots,m) \\ 0, \text{其他} \end{cases},$$

由于每架飞机最多可以紧接着另一架飞机飞行一次，也最多可以在一个登机口起飞，故有

$$y_{i,k} = \sum_{j \in N} z_{i,j,k}, \quad \forall i \in N, \quad k \in K; \tag{5.6}$$

$$y_{j,k} = \sum_{i \in N} z_{i,j,k}, \quad \forall j \in N, \quad k \in K. \tag{5.7}$$

如果记 T_i 表示飞机 i 的到达时间， T_i' 表示飞机 i 的离开时间， β 表示在同一个登机口的飞机的出发时间与下一架飞机的到达时间之间的缓冲时间，这里为

45 分钟，M 表示一个足够大的数，这里取 1000.由于同一个登机口每个时刻最多只能服务一架飞机，并且两架飞机被同一个登机口服务的时间间隔至少为 45 分钟，那么有

$$T_j + M(1 - z_{i,j,k}) \geq T_i' + \beta, \forall i, j \in N, k \in K. \tag{5.8}$$

根据 $z_{i,j,k}$ 的定义，又有

$$z_{i,j,k} \in \{0,1\}, \forall i, j \in N, k \in K. \tag{5.9}$$

5.1.3 模型建立

综上所述，那么问题一模型的一级目标函数为公式（5.1），二级目标函数为公式（5.2），约束条件为公式（5.3）-（5.9）.那么，我们可以建立如下问题一模型

$$\min \sum_{i \in N} y_{i,m+1}$$

$$\min \sum_{k \in K} (\operatorname{sgn} \sum_{i \in N} y_{i,k})$$

$$s.t. \begin{cases} y_{i,k} \in \{0,1\}, \forall i \in N, k = 1,2,\cdots,m+1 \\ \sum_{k=1}^{m+1} y_{i,k} = 1, \forall i \in N \\ y_{i,k} = 0, \forall i \in N, k \in K_i \\ y_{i,k} = \sum_{j \in N} z_{i,j,k}, \quad \forall i \in N, k \in K \\ y_{j,k} = \sum_{i \in N} z_{i,j,k}, \quad \forall j \in N, k \in K \\ T_j + M(1 - z_{i,j,k}) \geq T_i' + \beta, \forall i, j \in N, k \in K \\ z_{i,j,k} \in \{0,1\}, \forall i, j \in N, k \in K \end{cases}.$$

5.2 问题二模型的建立

5.2.1 目标函数

由于本问题是在问题一的基础上再加上旅客换乘因素，然后要计算中转旅客的最短流程时间，那么该模型的一级目标函数仍然是公式（5.1），如果记

$$D_i = \begin{cases} 1, \text{飞机} i \text{到达类型为} D; \\ 0, \text{飞机} i \text{到达类型为} I; \end{cases}$$

其中 $i \in N$，又 σ_i 表示飞机 i 的到达类型，是一个 4 维向量（DT, DS, IT, IS）.而机场的航站楼 T 有 28 个登机口，卫星厅 S 有 41 个登机口，共 69 个登机口，那么

$$DT = D_i \sum_{k=1}^{28} y_{i,k}, DS = D_i \sum_{k=29}^{69} y_{i,k}, IT = (1 - D_i) \sum_{k=1}^{28} y_{i,k}, IS = (1 - D_i) \sum_{k=29}^{69} y_{i,k}, \quad (i \in N)$$

即

$$\sigma_i = (D_i \sum_{k=1}^{28} y_{i,k}, D_i \sum_{k=29}^{69} y_{i,k}, (1 - D_i) \sum_{k=1}^{28} y_{i,k}, (1 - D_i) \sum_{k=29}^{69} y_{i,k}), (i \in N). \tag{5.10}$$

又记

$$D_i' = \begin{cases} 1, \text{飞机} i \text{出发类型为} D; \\ 0, \text{飞机} i \text{出发类型为} I; \end{cases}$$

其中 $i \in N$，又 σ_i' 表示飞机 i 的出发类型，也是一个 4 维向量（DT, DS, IT, IS）. 而机场的航站楼 T 有 28 个登机口，卫星厅 S 有 41 个登机口，共 69 个登机口，那么

$$DT = D_i' \sum_{k=1}^{28} y_{i,k}, DS = D_i' \sum_{k=29}^{69} y_{i,k}, IT = (1 - D_i') \sum_{k=1}^{28} y_{i,k}, IS = (1 - D_i') \sum_{k=29}^{69} y_{i,k}, \ i \in N$$

即

$$\sigma_i' = (D_i' \sum_{k=1}^{28} y_{i,k}, D_i' \sum_{k=29}^{69} y_{i,k}, (1 - D_i') \sum_{k=1}^{28} y_{i,k}, (1 - D_i') \sum_{k=29}^{69} y_{i,k}), (i \in N). \quad (5.11)$$

又假设 t_p 表示第 p 组旅客的流程时间，$\sigma_{a_p}(i)$ 表示向量 σ_{a_p} 的第 i 个分量，F 表示最短流程矩阵，为 4 阶方阵，$F_{i,j}$ 表示矩阵 F 的第 i 行第 j 列所对应的元素，而本问题不考虑旅客乘坐捷运时间和步行时间，因此

$$t_p = \sum_{i,j=1}^{4} \sigma_{a_p}(i) \sigma_{d_p}'(j) F_{i,j}, p \in P, \quad (5.12)$$

又记

$$S_p = \begin{cases} 1, \ T_{a_p} + t_p \le T_{d_p}' (\text{即第} p \text{组旅客换乘成功}, p \in P); \\ 0, \ \text{其他}; \end{cases} \quad (5.13)$$

而又假设换乘失败的旅客的换乘时间相当于 6 小时，那么所有旅客的流程时间之和为

$$\sum_{p \in P} [t_p S_p + 360(1 - S_p)] n_p,$$

从而二级目标函数为

$$\min \sum_{p \in P} [t_p S_p + 360(1 - S_p)] n_p. \quad (5.14)$$

在此基础上，要使被使用登机口的数量最少，那么三级目标函数为公式（5.2）.

5.2.2 约束条件

由于本模型是在问题一模型的基础上建立的，并没有增加或减少其他的约束条件，从而约束条件还是（5.3）-（5.9）式.

5.2.3 模型建立

经过上述对目标函数和约束条件的讨论，我们可以建立如下问题二模型

$$\min \sum_{i\in N} y_{i,m+1}$$

$$\min \sum_{p\in P} \left[t_p S_p + 360(1-S_p)\right] n_p$$

$$\min \sum_{k\in K} \operatorname{sgn}\left(\sum_{i\in N} y_{i,k}\right)$$

$$s.t.\begin{cases} y_{i,k} \in \{0,1\}, \forall i \in N, k = 1,2,\cdots,m+1 \\ \sum_{k=1}^{m+1} y_{i,k} = 1, \forall i \in N \\ y_{i,k} = 0, \forall i \in N, k \in K_i \\ y_{i,k} = \sum_{j\in N} z_{i,j,k}, \quad \forall i \in N, k \in K \\ y_{j,k} = \sum_{i\in N} z_{i,j,k}, \quad \forall j \in N, k \in K \\ T_j + M(1-z_{i,j,k}) \geq T_i' + \beta, \forall i, j \in N, k \in K \\ z_{i,j,k} \in \{0,1\}, \forall i, j \in N, k \in K \end{cases}.$$

5.3 问题三模型的建立

5.3.1 目标函数

由于本模型是考虑了中转旅客的换乘时间（包括流程时间、行走时间和捷运时间），并且考虑换乘旅客的总体紧张度最小化问题. 如果我们记 C_p 表示第 p 组旅客的换乘紧张度，r_p 表示第 p 组旅客的换乘时间，t_p' 表示第 p 组旅客的最短流程时间，w_p 表示第 p 组旅客的捷运时间，而旅客换乘时间=最短流程时间+捷运时间+行走时间，即

$$r_p = t_p + t_p' + w_p (p \in P). \tag{5.15}$$

记飞机 i 所在的登机口区域为 γ_i，为一个 7 维 0-1 向量（$TN, TC, TS, SN, SC, SS, SE$），根据所给的 Gates 表可知：数据的第 1-9 行的登机口区域类型为 $T-North$，10-19 行的登机口区域类型为 $T\text{-}Center$，第 20-28 行的登机口区域类型为 $T\text{-}South$，第 29-38 行的登机口区域类型为 $S\text{-}North$，第 39-48 行的登机口区域类型为 $S\text{-}Center$，第 49-58 行的登机口区域类型为 $S\text{-}South$，第 59-69 行的登机口区域类型为 $S\text{-}East$，即

$$TN = \sum_{k=1}^{9} y_{i,k}, TC = \sum_{k=10}^{19} y_{i,k}, TS = \sum_{k=20}^{28} y_{i,k}, SN = \sum_{k=29}^{38} y_{i,k},$$

$$SC = \sum_{k=39}^{48} y_{i,k}, SS = \sum_{k=49}^{58} y_{i,k}, SE = \sum_{k=59}^{69} y_{i,k}, (i \in N),$$

从而

$$\gamma_i=(\sum_{k=1}^{9}y_{i,k},\sum_{k=10}^{19}y_{i,k},\sum_{k=20}^{28}y_{i,k},\sum_{k=29}^{38}y_{i,k},\sum_{k=39}^{48}y_{i,k},\sum_{k=49}^{58}y_{i,k},\sum_{k=59}^{69}y_{i,k}),(i\in N) \quad (5.16)$$

记第 p 组旅客的步行时间为 w_p，M 为登机口区域之间的行走时间矩阵，为 7 阶方阵，$M_{i,j}$ 为矩阵 M 的第 i 行第 j 行对应的元素，$\gamma_{a_p}(i)$ 表示向量 γ_{a_p} 的第 i 个分量，那么第 p 组旅客的行走时间

$$w_p=\sum_{i,j=1}^{7}\gamma_{a_p}(i)\gamma_{d_p}(j)M_{i,j},(p\in P). \quad (5.17)$$

记 F' 为登机口区域之间的捷运次数矩阵，为 4 阶方阵，$F'_{i,j}$ 为矩阵 F 的第 i 行第 j 行对应的元素，而单程捷运时间为 8 分钟，从而第 p 组旅客的捷运时间

$$t'_p=\sum_{i,j=1}^{4}8\sigma_{a_p}(i)\sigma'_{d_p}(j)F'_{i,j},(p\in P), \quad (5.18)$$

由公式（5.12），（5.15），（5.17），（5.18）可知，

$$r_p=\sum_{i,j=1}^{4}\sigma_{a_p}(i)\sigma'_{d_p}(j)F_{i,j}+\sum_{i,j=1}^{4}8\sigma_{a_p}(i)\sigma'_{d_p}(j)F'_{i,j}+\sum_{i,j=1}^{7}\gamma_{a_p}(i)\gamma_{d_p}(j)M_{i,j},(p\in P). \quad (5.19)$$

如果记第 p 组旅客乘坐的后一航班出发时间为 T'_{d_p}，前一航班到达时间为 T_{a_p}，那么由于航班连接时间=后一航班出发时间-前一航班到达时间，即第 p 组旅客的航班连接时间=$T'_{d_p}-T_{a_p},(p\in P)$. 由于旅客换乘成功的标准是前一航班到达时间加上换乘时间（包括最短流程时间、捷运时间和行走时间）小于下一航班到达时间，那么

$$S'_p=\begin{cases}1,\ T_{a_p}+r_p\le T'_{d_p}(\text{第}p\text{组旅客换乘成功},\ p\in P)\\ 0,\ \text{其他}\end{cases}. \quad (5.20)$$

又由于

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

而换乘失败的旅客的换乘时间相当于 6 小时，从而第 p 组旅客换乘紧张度

$$C_p=\frac{r_pS'_p+360(1-S'_p)}{T'_{d_p}-T_{a_p}},(p\in P), \quad (5.21)$$

如果第 p 组旅客的人数记为 n_p，因此所有旅客的换乘紧张度之和为

$$\sum_{p\in P} C_p n_p.$$

由于本模型是在问题二的基础上细化，并且考虑坐捷运时间和步行时间，引入旅客换乘连接变量，从而本模型的一级目标函数还是公式（5.1），二级目标是所有旅客换乘紧张度之和最小，故二级目标函数为

$$\min \sum_{p\in P} C_p n_p, \tag{5.22}$$

三级目标函数还是公式（5.2）.

5.3.2 约束条件

由于本模型是在问题一的基础上建立、问题二模型的基础上细化，并没有增加其他的约束条件，从而约束条件还是（5.3）-（5.9）式.

5.3.3 模型建立

经过上述对目标函数和约束条件的讨论，我们可以建立如下最优化中转旅客总体紧张度模型

$$\min \sum_{i\in N} y_{i,m+1}$$

$$\min \sum_{p\in P} C_p n_p$$

$$\min \sum_{k\in K} \operatorname{sgn}\left(\sum_{i\in N} y_{i,k}\right)$$

$$s.t.\begin{cases} y_{i,k} \in \{0,1\}, \forall i \in N, k = 1,2,\cdots,m+1 \\ \sum_{k=1}^{m+1} y_{i,k} = 1, \forall i \in N \\ y_{i,k} = 0, \forall i \in N, k \in K_i \\ y_{i,k} = \sum_{j\in N} z_{i,j,k}, \quad \forall i \in N, k \in K \\ y_{j,k} = \sum_{i\in N} z_{i,j,k}, \quad \forall j \in N, k \in K \\ T_j + M(1 - z_{i,j,k}) \geq T_i' + \beta, \forall i, j \in N, k \in K \\ z_{i,j,k} \in \{0,1\}, \forall i, j \in N, k \in K \end{cases}$$

六、模型的求解与结果的分析

6.1 模型特点的分析

由于本模型建立的相对简单，前期并没有对数据进行处理，从而需要把数据导入程序里，使得程序的编写能力明显更强.

6.2 模拟退火算法

6.2.1 算法的简介[2]

模拟退火其实也是一种贪心算法，但是它的搜索过程引入了随机因素.在迭代更新可行解时，以一定的概率来接受一个比当前解要差的解，因此有可能会跳

出这个局部的最优解，达到全局的最优解. 以下图 6.1 为例，假定初始解为左边蓝色点 A，模拟退火算法会快速搜索到局部最优解 B，但在搜索到局部最优解后，不是就此结束，而是会以一定的概率接受到左边的移动. 也许经过几次这样的不是局部最优的移动后会到达全局最优点 D，于是就跳出了局部最小值.

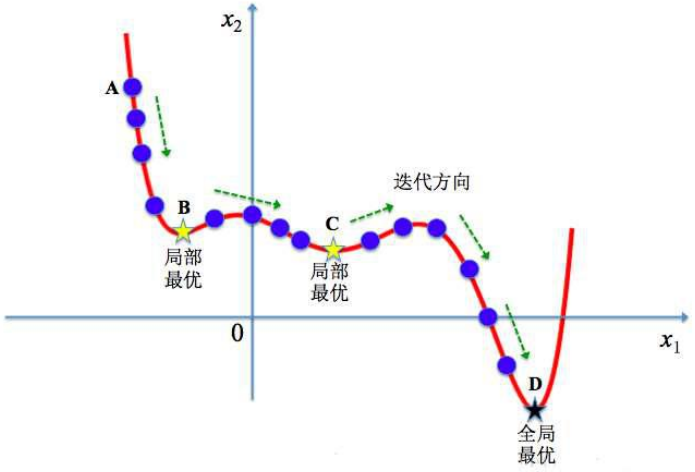


图 6.1 模拟退火示意图

6.2.2 算法的流程

模拟退火算法的基本步骤为：

1、初始温度 $T_0 = 2nK$，温度下限 $T_{\min} = 0.01K$，初始解矩阵 $Y_0 = (y_{i,k})$，并且取

$$y_{i,k} = \begin{cases} 1, k = m+1, \forall i \in N; \\ 0, k \neq m+1, \forall i \in N. \end{cases}$$

即把所有飞机都排列在临时机位，这个解是最差的可行解，我们以这个 Y_0 作为迭代的起点.

2、产生新解 Y_{i+1} 的机制：在 Y_i 中随机取一行，将这一行中的 1（每行有且仅有一个 1）在该行中重新指定一个随机位置，就形成了 Y_{i+1}.

3、验证 Y_{i+1} 是否满足约束条件，若满足，进行下一步；若不满足，返回第 2 步，重新产生新的解.

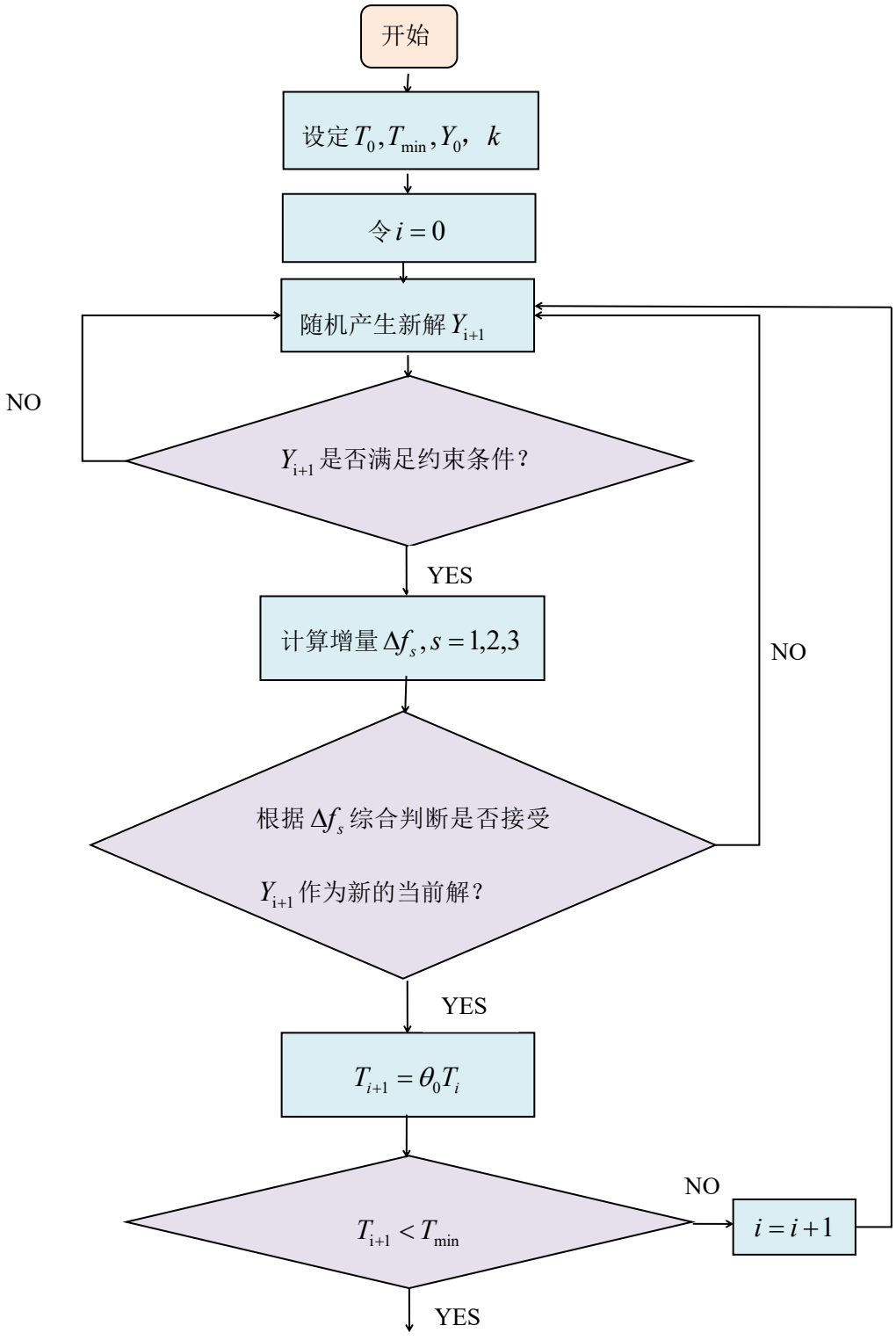
4、计算增量 $\Delta f_s = f_s(Y_{i+1}) - f_s(Y_i), s = 1,2,3$（其中 f_s 表示第 s 级目标函数），并进行如下分支判断：

（1）若 $\Delta f_1 < 0$，接受 Y_{i+1} 作为新的当前解；

（2）若 $\Delta f_1 > 0$，则以概率 $e^{-\frac{\Delta f_i}{kTf_1(Y_i)}}$ 接受 Y_{i+1} 作为新的当前解；

（3）若 $\Delta f_1 = 0$，且 $\begin{cases} \Delta f_3 < 0, \text{接受} Y_{i+1} \text{作为为新的当前} \\ \Delta f_3 < 0, \text{以概率} e^{-\frac{\Delta f_3}{kTf_3(Y_i)}} \text{接受} Y_{i+1} \text{作为为新的当前.} \\ \Delta f_3 = 0, \text{返回步骤2} \end{cases}$

5、令 $T_{i+1} = \theta_0 T_i$(其中 θ_0 为降温系数，这里取0.999).若 $T_{i+1} < T_{\min}$，则将当前解作为最优解，结束程序；若 $T_{i+1} \geq T_{\min}$，则令 $i = i+1$，返回步骤 2.



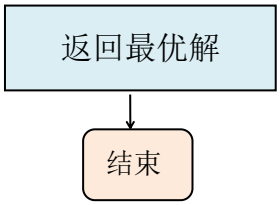


图 6.2 模拟退火算法的步骤

6.3 问题一模型的求解

利用 C++对 5.1 所建的模型根据模拟退火算法计算出一级目标函数值为 84，二级目标函数值为 67.(程序见附录，具体结论见附件 1）

如果我们又记问题一模型的最优解为$Y^{(1)}=(y_{i,k}^{(1)})_{n\times(m+1)}$，其中

$$y_{i,k}^{(1)}=\begin{cases}1,\text{飞机}i\text{停靠在的登机口}k\\0,\text{其他}\end{cases},$$

6.3.1 宽体机和窄体机中分别成功分配到登机口的航班数量和比例的计算

该模型的所有分配到登机口的航班数记为n_1，登机口使用比例记为e_1，那么

$$n_1=2\sum_{i\in N}(1-y_{i,70}^{(1)}),e_1=\frac{n_1}{\sum_{i\in N}2}. \tag{6.1}$$

如果记N_1表示N中窄体机的全体，W_1表示N中窄体机的全体，那么该模型窄体机中被成功分配到登机口的航班数量

$$h_1=2\sum_{i\in N_1}(1-y_{i,70}^{(1)}), \tag{6.2}$$

窄体机中被成功分配到登机口的航班比例

$$g_1=\frac{h_1}{\sum_{i\in N_1}2}; \tag{6.3}$$

宽体机中被成功分配到登机口的航班数量

$$h_1'=2\sum_{i\in W_1}(1-y_{i,70}^{(1)}), \tag{6.4}$$

宽体机中被成功分配到登机口的航班比例

$$g_1'=\frac{h_1'}{\sum_{i\in W_1}2}. \tag{6.5}$$

然后我们利用 C++解出h_1=366, g_1=0.717647, h_1'=76, g_1'=0.76（程序见附录）.

我们又利用 matlab 画出按宽、窄体机分别成功分配到登机口的航班数量和比例对比柱形图，如图 6.3 所示。

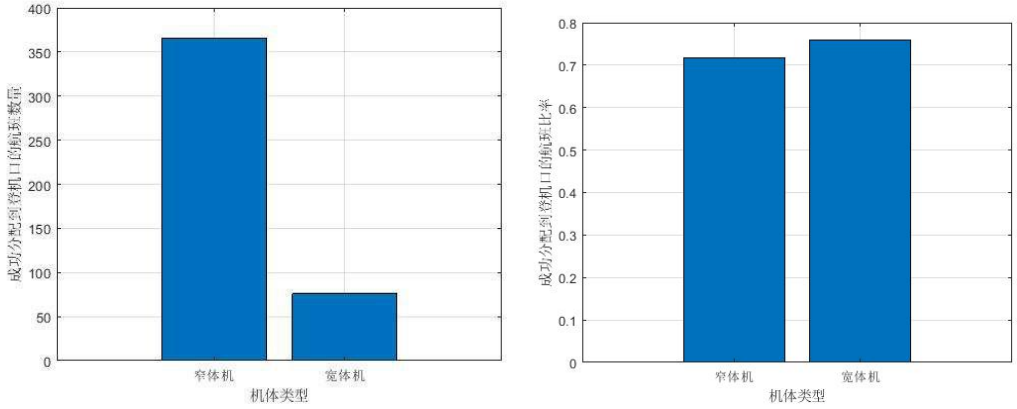


图 6.3 问题一宽、窄体机分别成功分配到登机口的航班数量和比例对比图

6.3.2 T 和 S 登机口的使用数目和被使用登机口的的平均使用率的计算

由于 1-28 号是航站楼 T，29-69 是卫星厅 S，故问题一中 T 登机口的使用数目

$$l_1 = \sum_{k=1}^{28} \operatorname{sgn}\left(\sum_{i \in N} y_{i,k}^{(1)}\right), \tag{6.6}$$

S 登机口的使用数目

$$l_1' = \sum_{k=29}^{69} \operatorname{sgn}\left(\sum_{i \in N} y_{i,k}^{(1)}\right). \tag{6.7}$$

T 登机口的平均使用率

$$q_1 = \frac{\sum_{k=1}^{28} \sum_{i \in A_k^{(1)}} (\min\{T_i', 20\text{日}24\text{时}\} - \max\{T_i, 20\text{日}0\text{时}\})}{24 \times 60 \times l_1}, \tag{6.8}$$

S 登机口的平均使用率

$$q_1' = \frac{\sum_{k=29}^{69} \sum_{i \in A_k^{(1)}} (\min\{T_i', 20\text{日}24\text{时}\} - \max\{T_i, 20\text{日}0\text{时}\})}{24 \times 60 \times l_1'}, \tag{6.9}$$

其中 $A_k^{(1)}$ 表示矩阵 $Y^{(1)}$ 中第 k 列中取 1 的所有元的行标的集合（$k \in K$）.

然后我们利用 C++解出 l_1=28, l_1'=39, q_1=0.15062, q_1'=0.16699, 程序见附录. 然后利用 matlab 分别画出 T 和 S 登机口的使用数目和被使用登机口的平均使用率对比柱形图，如图 6.4 所示。

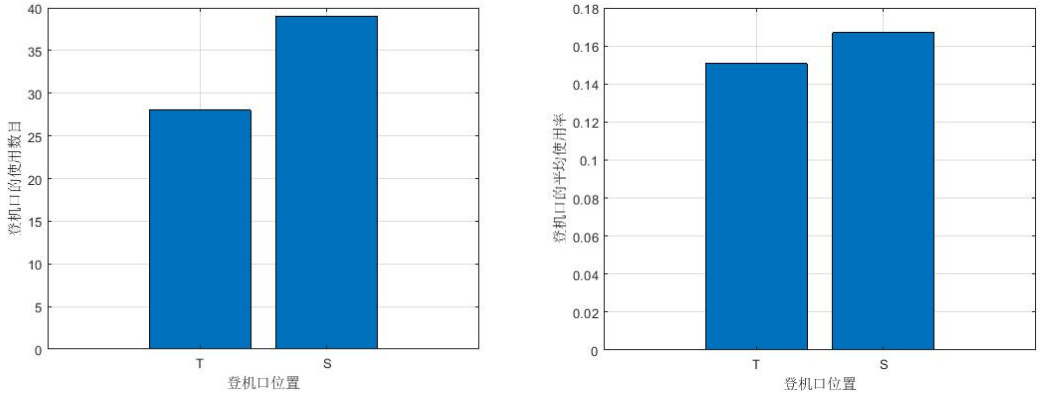


表 6.4 T 和 S 登机口的使用数目和被使用登机口的平均使用率对比柱形图

6.4 问题二模型的求解

利用 C++对 5.2 所建的模型根据模拟退火算法计算出一级目标函数值为 87，二级目标函数值为 45850，三级目标函数值为 67(程序见附录，具体结论见附件 2）.

如果我们记问题二模型的最优解为 $Y^{(2)}=(y_{i,k}^{(2)})_{n\times(\mathrm{m}+1)}$，其中

$$y_{i,k}^{(2)}=\begin{cases}1,\text{飞机}i\text{停靠在的登机口}k\\0,\text{其他}\end{cases},$$

6.4.1 宽体机和窄体机中分别成功分配到登机口的航班数量和比例的计算

该模型的所有分配到登机口的航班数记为 n_2，登机口使用比率记为 e_2，那么

$$n_2=2\sum_{i\in N'}(1-y_{i,70}^{(2)}),e_2=\frac{n_2}{\sum_{i\in N'}2}. \tag{6.10}$$

如果 N_2 表示 N' 中窄体机的全体，W_2 表示 N' 中窄体机的全体，那么该模型窄体机中被成功分配到登机口的航班数量

$$h_2=2\sum_{i\in N_2}(1-y_{i,70}^{(2)}), \tag{6.11}$$

窄体机中被成功分配到登机口的航班比例

$$g_2=\frac{h_2}{\sum_{i\in N_2}2}, \tag{6.12}$$

宽体机中被成功分配到登机口的航班数量

$$h_2'=2\sum_{i\in W_2}(1-y_{i,70}^{(2)}), \tag{6.13}$$

宽体机中被成功分配到登机口的航班比例

$$g_2'=\frac{h_2'}{\sum_{i\in W_2}2}. \tag{6.14}$$

然后我们利用 C++解出 h_2=388, g_2=0.723881, h_2'= 80, g_2'=0.754717(程序见附录).我们又利用 matlab 画出按宽、窄体机分别成功分配到登机口的航班数量和比例对比柱形图，如图 6.5 所示。

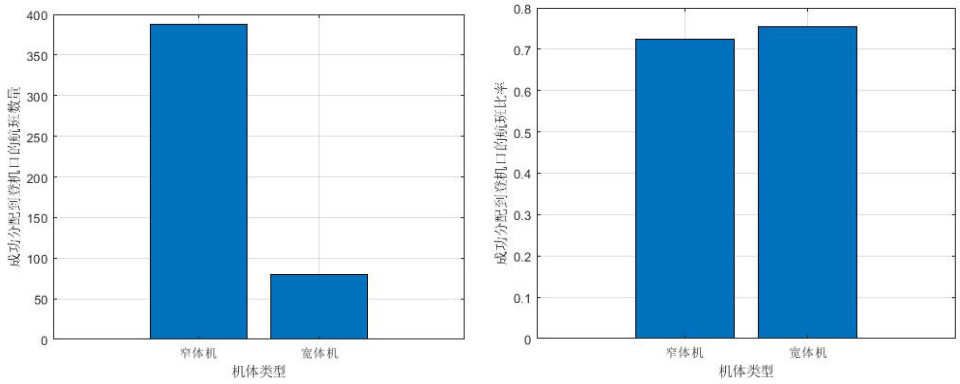


图 6.5 问题二宽、窄体机分别成功分配到登机口的航班数量和比例对比图

6.4.2 T 和 S 登机口的使用数目和被使用登机口的的平均使用率的计算

由于 1-28 号是航站楼 T，29-69 是卫星厅 S，故问题二中 T 登机口的使用数目

$$l_2=\sum_{k=1}^{28}\operatorname{sgn}(\sum_{i\in N'}y_{i,k}^{(2)}),\tag{6.15}$$

S 登机口的使用数目

$$l_2'=\sum_{k=29}^{69}\operatorname{sgn}(\sum_{i\in N'}y_{i,k}^{(2)}).\tag{6.16}$$

T 登机口的平均使用率

$$q_2=\frac{\sum_{k=1}^{28}\sum_{i\in A_k^{(2)}}(\min\{T_i',20日24时\}-\max\{T_i,20日0时\})}{24\times60\times l_2},\tag{6.17}$$

S 登机口的平均使用率

$$q_2'=\frac{\sum_{k=29}^{69}\sum_{i\in A_k^{(2)}}(\min\{T_i',20日24时\}-\max\{T_i,20日0时\})}{24\times60\times l_2'},\tag{6.18}$$

其中 $A_k^{(2)}$ 表示矩阵 $Y^{(2)}$ 中第 k 列中取 1 的所有元的行标的集合（$k\in K$）.

然后我们利用 C++解出 l_2=28, l_2'=39, q_2= 0.136105, q_2'=0.153577,(程序见附录).然后利用 matlab 分别画出 T 和 S 登机口的使用数目和被使用登机口的平均使用率对比柱形图，如图 6.6 所示。

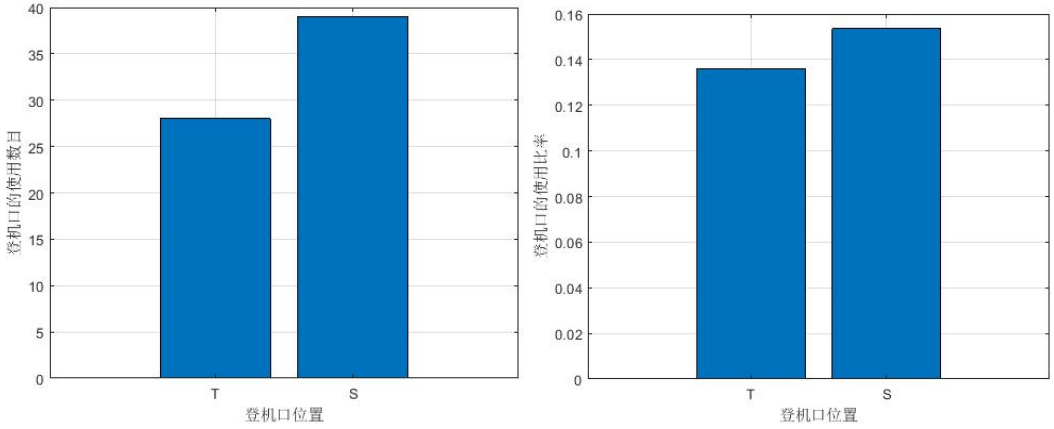


表 6.6 T 和 S 登机口的使用数目和被使用登机口的平均使用率对比柱形图

6.4.3 换乘失败旅客数量和比例的计算

问题二的换乘失败旅客数量

$$o=\sum_{p\in P}(1-S_p)n_p\ , \tag{6.19}$$

换乘失败的比例

$$b=\frac{o}{\sum_{p\in P}n_p}. \tag{6.20}$$

然后我们利用 C++解出 o =0, b =0，程序见附录.

6.5 问题三模型的求解

利用 C++对 5.3 所建的模型根据模拟退火算法计算出一级目标函数值为 91，二级目标函数值为 814.799，三级目标函数值为 67(程序见附录，具体结论见附件 3）.

如果我们记问题三模型的最优解为 $Y^{(3)}=(y_{i,k}^{(3)})_{n\times(\mathrm{m}+1)}$，其中

$$y_{i,k}^{(3)}=\begin{cases}1,\text{飞机}i\text{停靠在的登机口}k\\0,\text{其他}\end{cases},$$

6.5.1 宽体机和窄体机中分别成功分配到登机口的航班数量和比例的计算

该模型的所有分配到登机口的航班数记为 n_3，登机口使用比率记为 e_3，那么

$$n_3=2\sum_{i\in N'}(1-y_{i,70}^{(3)}),e_3=\frac{n_3}{\sum_{i\in N'}2}. \tag{6.21}$$

如果 N_2 表示 N' 中窄体机的全体， W_2 表示 N' 中窄体机的全体，那么该模型窄体机中被成功分配到登机口的航班数量

$$h_3=2\sum_{i\in N_2}(1-y_{i,70}^{(3)}), \tag{6.22}$$

窄体机中被分配到登机口的航班的比例

$$g_3 = \frac{h_3}{\sum_{i \in N_2} 2}, \tag{6.23}$$

宽体机中被成功分配到登机口的航班数量

$$h_3' = 2\sum_{i \in W_2} (1 - y_{i,70}^{(3)}), \tag{6.24}$$

宽体机中被成功分配到登机口的航班比例

$$g_3' = \frac{h_3'}{\sum_{i \in W_2} 2}. \tag{6.25}$$

然后我们利用 C++解出 h_3=380, g_3=0.708955, h_3'=80, g_3'=0.754717(程序见附录).我们又利用 matlab 画出按宽、窄体机分别成功分配到登机口的航班数量和比例对比柱形图，如图 6.7 所示。

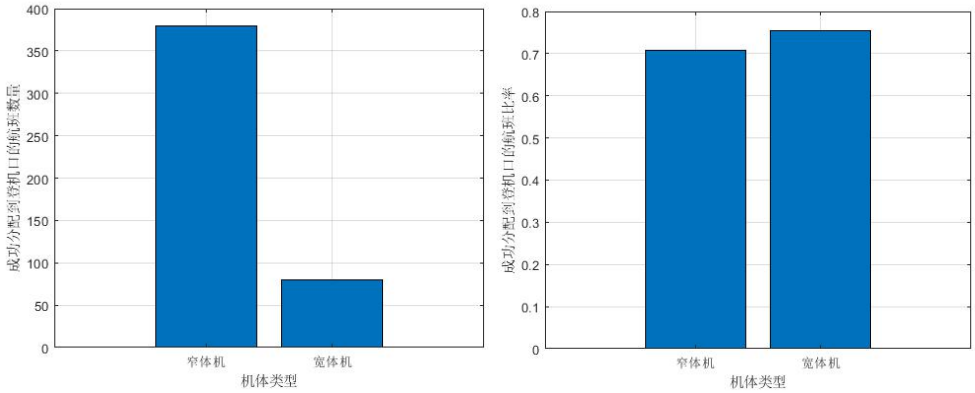


图 6.7 问题三宽、窄体机分别成功分配到登机口的航班数量和比例对比图

6.5.2 T 和 S 登机口的使用数目和被使用登机口的的平均使用率的计算

由于 1-28 号是航站楼 T，29-69 是卫星厅 S，故问题三中 T 登机口的使用数目

$$l_3 = \sum_{k=1}^{28} \operatorname{sgn}(\sum_{i \in N'} y_{i,k}^{(3)}), \tag{6.26}$$

S 登机口的使用数目

$$l_3' = \sum_{k=29}^{69} \operatorname{sgn}(\sum_{i \in N'} y_{i,k}^{(3)}). \tag{6.27}$$

T 登机口的平均使用率

$$q_3 = \frac{\sum_{k=1}^{28} \sum_{i \in A_k^{(3)}} (\min\{T_i', 20日24时\} - \max\{T_i, 20日0时\})}{24 \times 60 \times l_3}, \tag{6.28}$$

S 登机口的平均使用率

$$q_3' = \frac{\sum_{k=29}^{69}\sum_{i\in A_k^{(3)}}(\min\{T_i', 20日24时\} - \max\{T_i, 20日0时\})}{24\times 60\times l_3'}, \tag{6.29}$$

其中 $A_k^{(3)}$ 表示矩阵 $Y^{(3)}$ 中第 k 列中取 1 的所有元的行标的集合（$k \in K$）.

然后我们利用 C++解出 l_3=28, l_3'=39, q_3=0.125203, q_3'=0.158712(程序见附录).然后利用 matlab 分别画出 T 和 S 登机口的使用数目和被使用登机口的平均使用率对比柱形图，如图 6.8 所示。

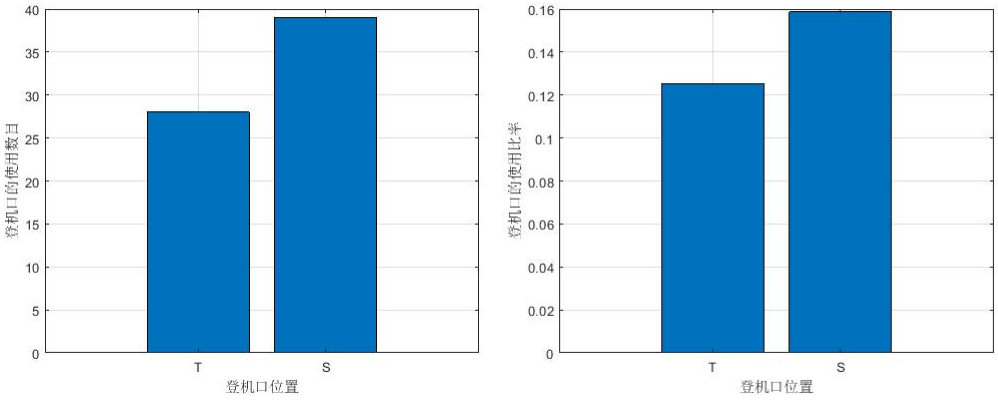


表 6.8 T 和 S 登机口的使用数目和被使用登机口的平均使用率对比柱形图

6.5.3 换乘失败旅客数量和比例的计算

问题三的换乘失败旅客数量

$$o' = \sum_{p\in P}(1 - S_p')n_p, \tag{6.30}$$

换乘失败的比例

$$b' = \frac{o'}{\sum_{p\in P} n_p}. \tag{6.31}$$

然后我们利用 C++解出 o'=0, b'=0，程序见附录.

6.5.4 总体旅客换乘时间分布图和总体紧张度分布图

根据前面的计算，我们利用 matlab 画出总体旅客换乘时间分布图和总体紧张度分布图如图 6.9 所示.

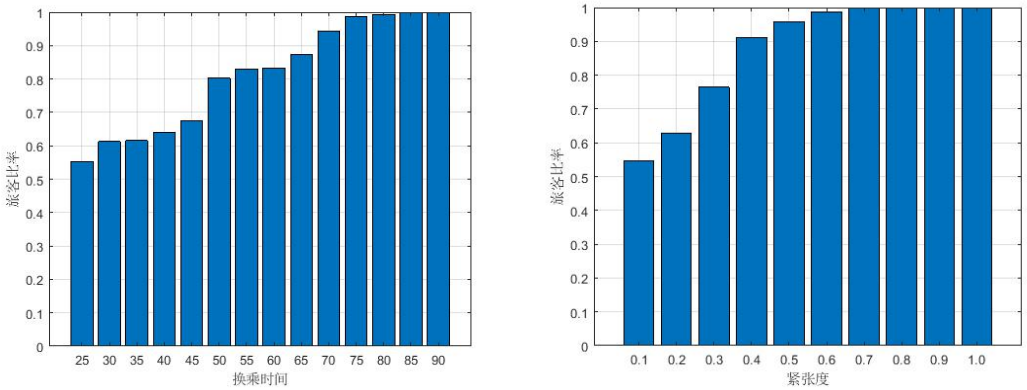


图 6.9 总体旅客换乘时间分布图和总体紧张度分布图

七、模型的评价与推广

7.1 模型的评价

7.1.1 模型的优点

1、由于本文在模型建立的过程中，没有对数据作复杂的处理，从而减少了很多工作量，但是对于程序的书写上增加了很大的难度；

2、对于本文所建立的三个模型，实用性很高，可以应用在相关的问题上；

3、本文在模型建立的过程中，并没有作很多理想性假设，从而结果相对更准确；

7.1.2 模型的缺点

由于模拟退火算法的特点，得出的结果只是有可能跳出局部最优解，从而有可能得到的结果并不是全局最优解.

7.2 模型的推广

由于本文中所建的三个模型是一个实用性非常强的模型，也可以应用在机场的相关问题的优化上，从而对类似的问题，可以直接把本文中建立的模型的参数作一些适当的改变，得到相应的优化后的方案.

八、参考文献

[1] Airline Innovation Centre of Excellence Team & Wipro Technologies, Gate Assignment Solution (GAM) Using Hybrid Heuristics Algorithm, Draft Technical White Paper On Gate Assignment Solution, 1-9,2009

[2] AI_BigData_WH，模拟退火算法，https://blog.csdn.net/AI_BigData_wh/article/details/77943787?locationNum=2&fps=1，2018.9.18.

附录

解决该问题的代码分成三个文件，第一个是DataStruture.h 文件；第二个文件AirPort.hpp是对类文件,它封装装了解决该问题的算法;第三个文件Run.cpp是主程序入口。代码中主要用了三个结构体Pucks, Tickets 和 Gates 用于读取文件中的相应的信息和一个类AirPort，其主要是封装对用模拟退火解决航班登记口登记问题的实现，对产生数据的统计。最后一个文件run.cpp有一个主程序直接调用AirPort的接口，解决三个问题。

文件 “DataStructure.h” 定义了一些数据结构

```
#pragma once
#include <iostream>
#include <fstream>
#include <set>
#include <vector>
#include <sstream>
#include <string>
#include <map>
#include <algorithm>
#include <cmath>
#include <forward_list>
#include <cassert>
using namespace std;
/*********************************************************************************
************/
/*****************************一些重要的数据结构
*********************************************/
/*********************************************************************************
***********/
enum FLIGHT_SHAPE {
    NARROW,
    WIDE,
};
enum A_D_TYPE {
    DEMOSTIC = 5,
    INTER,
    D_OR_I,
};
enum TERMINAL {
    T = 10,
    S,
};
enum AREA {
    NORTH = 20,
    CENTER,
```

```
    SOUTH,
    EAST,
    WEST,
};
/* D_T, D_S, I_T, I_S <-> 0, 1, 2, 3*/
inline int map_DIST(TERMINAL TS, A_D_TYPE DI) {
    return TS - T + (DI - DEMOSTIC) * 2;
}
/* {0, ..., 6} <-> {T-N, T-C, T-S, S-N, S-C, S-S, S-E} */
inline int map_TSNCE(TERMINAL TS, AREA NCE) {
    return (TS - T) * 3 + NCE - NORTH;
}
/* 定义存储tickets 的数据结构 */
typedef struct Tickets {
    int parssengerId;
    int num_parssengers;
    string arrival_flight;
    string departure_flight;
    int arrival_date;
    int departure_date;
} Tickets;
/* 定义存储Gates 的数据结构 */
typedef struct Gates {
    string name;
    int gates_id;
    TERMINAL gates_type;
    int area;
    A_D_TYPE arrival_resive_type; // D, I, D_OR_I
    A_D_TYPE departure_release_type;
    FLIGHT_SHAPE flight_resive_shape; // W, N
} Gates;
/* 定义存储Pucks 的数据结构 */
typedef struct Pucks {
    int flight_id;
    int arrival_date;   // range at {20}
    int departure_date; // range at {20}
    int arrival_time; // range at [0, 288], set every 5 minitues a unit.
    int departure_time;
    string arrival_flight;
    string departure_flight;
    string plane_num;
    A_D_TYPE arrival_type; // D, I
    A_D_TYPE departure_type;
    FLIGHT_SHAPE flight_shape; // W, N
```

```
} Pucks;
inline std::ostream & operator << (std::ostream &out, std::pair<int, int> x) {
    std::cout << "(" << x.first << "," << x.second << ")";
    return out;
}
template <class T>
inline std::ostream & operator << (std::ostream &out, std::vector<T> &vec) {
    if (vec.empty()) {
        out << "" << std::endl;
        return out;
    }
    int size_ = vec.size();
    for (int i = 0; i < size_; i++) {
        out << vec[i];
        if (i < size_ - 1)
            out << ",\t";
    }
    return out;
}
```

文件 “AirPort.hpp”，将算法封装到一个类里面

```
#pragma once
#include "DataStructure.h"
#include <algorithm>
#include <cstring>
#include <cstdlib>
#include <random>
#include <functional>
#include <cassert>
#include <ctime>
using namespace std;
/*************************************************************************************
***********/
/*****************************  定  义  一  些  常  用  的  函  数
*****************************************/
/*************************************************************************************
**********/
inline float MAX(float x, float y) { return ((x) > (y) ? (x) : (y)); }
inline float MIN(float x, float y) { return ((x) < (y) ? (x) : (y)); }
static int  string2int(const string &str) {
    int res = 0;
    char c = '0';
    for (int i = 0; i < str.size(); i++)
        res = res * 10 + str[i] - c;
    return res;
```

```
}
static void splitBySpace(vector<string> &result, const string &line) {
    result.clear();
    string str = line + " ";
    int start = 0;
    for (int i = 0; i < str.size(); i++) {
        if (str[i] == ' ' || str[i] == '\t') {
            if (i != start) {
                string token = str.substr(start, i - start);
                result.push_back(token);
                start = i;
            }
            start++;
        }
    }
}
static void splitByColon(vector<string> &result, const string &line) {
    result.clear();
    string str = line + ":";
    int start = 0;
    for (int i = 0; i < str.size(); i++) {
        if (str[i] == ':') {
            if (i != start) {
                string token = str.substr(start, i - start);
                result.push_back(token);
                start = i;
            }
            start++;
        }
    }
}
/*************************************************************************
***********/
/************************ 将 解 决 本 题 的 方 法 封 装 到 一 个 类 AirPort 里
*********************************/
/*************************************************************************
************/
class AirPort{
public:
    /* 统计三个问题的数据结构 */
    typedef struct Stat {
        float n_i;
        float e_i;
        float h_i;
```

```
        float h_i_p;
        float g_i;
        float g_i_p;
        float l_i;
        float l_i_p;
        float q_i;
        float q_i_p;
    } Stat;
    /* 统计问题 2，3 要的数据结构 */
    typedef struct stat_2 {
        float O;
        float B;
    } Stat2;
    /* 统计问题三要的数据结构 */
    typedef struct stat_3 {
        float O;
        float B;
        vector<float> v_t;
        vector<float> v_t_p;
    } Stat3;
    /* 生成结果的要放到的地址前缀 */
    string prefix = "E:/VS/Airport";
    typedef float (AirPort::*Objective) ();
    AirPort();
    ~AirPort();
    /* 读取 pucks 信息 */
    void readPucks();
    /* 读取 tickets 信息 */
    void readTickets();
    /* 读取 gates 信息*/
    void readGates();
    int findf_i(const Tickets &t);
    /* 解决第一题的函数 */
    void solution1();
    /* 解决第二题的函数 */
    void solution2();
    /* 解决第三题的函数 */
    void solution3();
    /* 主要的算法实现 */
    void masterAlgorithm(int n, int K, vector<Objective> secondary_objs);
    /* 输出运行结果 */
    void outputResolut(string dir = "", int solution = 1);
private:
    vector<Pucks> pucks;
```

```
    vector<Tickets> tickets;
    vector<Gates> gates;
    map<string, FLIGHT_SHAPE> map_flight_shape;
    /* 定义集合  */
    vector<int> N; // 问题在所涉及的航班集合
    vector<int> P; // 涉及的旅客集合
    int K; // 大小 0 to 68;
    map<int, vector<int> > K_i; // 一个集合存储着不能接受航班 i 的登机口好
    map<int, vector<int> > NK_i; // K_i 的补集
    map<int, int> a_p; // 映射旅客好到航班号
    map<int, int> d_p;
    int m, n;
    /* 定义参数 */
    vector<int> a_i; // 航班到达时间表
    vector<int> d_i; // 航班离开时间表
    int beta; // 从一个航班离开到下个航班到达之间的缓冲时间
    int M; // 一个够大的数
    int W;
    int sum_people;
    /* 定义变量 */
    vector<int>    y_i;
    vector<int>    z_ij;
    /* 模型中定义的目标函数和限制 */
    float primary_objective();
    float secondary_objective();
    float solution2_objective();
    float solution3_objective();
    // a_j + (1-Z_ijk) * M >= d_i; for all i,j, all k.
    bool constrain_4();
    bool constrain_4(const set<int>& changedK);
    void calculateZij();
    void calculateZij(const set<int>& changedK);
    /* 在模拟退火算法中用 randomMove 生成新解 */
    void randomMove();
    void randomMove(set<int>& changedK);
    /* 统计生成的结果 */
    void statistic(Stat &stat);
    void statistic(Stat2 &stat);
    void statistic(Stat3 &stat);
    float diffCost();
    void printY_i(string file);
protected:
    vector<Objective> secondary_objs;
    vector<float> f_current;
```

```
    vector<float> f_next;
private:
    /* D_T, D_S, I_T, I_S <-> 0, 1, 2, 3*/
    int tram_time_table[4][4];
    int procedure_time_table[4][4];
    /* {0, ..., 6} <-> {T-N, T-C, T-S, S-N, S-C, S-S, S-E} */
    int walk_time_table[7][7];
};
/*****************************************************************************
****************/
/********************** 以 下 时 对 类 中 每 个 函 数 的 实 现
*************************************************/
/*****************************************************************************
****************/
AirPort::AirPort() : W(500), beta(9), M(5000), sum_people(0) {
    /* init map_flight_shape */
    {
        map_flight_shape["332"] = WIDE;
        map_flight_shape["333"] = WIDE;
        map_flight_shape["33E"] = WIDE;
        map_flight_shape["33H"] = WIDE;
        map_flight_shape["33L"] = WIDE;
        map_flight_shape["773"] = WIDE;
        map_flight_shape["319"] = NARROW;
        map_flight_shape["320"] = NARROW;
        map_flight_shape["321"] = NARROW;
        map_flight_shape["323"] = NARROW;
        map_flight_shape["325"] = NARROW;
        map_flight_shape["738"] = NARROW;
        map_flight_shape["73A"] = NARROW;
        map_flight_shape["73E"] = NARROW;
        map_flight_shape["73H"] = NARROW;
        map_flight_shape["73L"] = NARROW;
        /* D_T, D_S, I_T, I_S <-> 0, 1, 2, 3*/
        int tmp1[4][4] = { {0,1,0,1},{1,0,1,0},{0,1,0,1},{1,2,1,0} };
        memcpy(tram_time_table, tmp1, 4 * 4 * sizeof(int));
        int tmp2[4][4] = { {3,4,7,8}, {4,3,8,7}, {7,8,4,6}, {8,9,6,4} };
        memcpy(procedure_time_table, tmp2, 4 * 4 * sizeof(int));
        /* {0, ..., 6} <-> {T-N, T-C, T-S, S-N, S-C, S-S, S-E} */
        int tmp3[7][7] = { {2, 3, 4, 5, 4, 5, 5},
                           {3, 2, 3, 4, 3, 4, 4},
                           {4, 3, 2, 5, 4, 5, 5},
                           {5, 4, 5, 2, 3, 4, 4},
                           {4, 3, 4, 3, 2, 3, 3},
```

```
                                {5, 4, 5, 4, 3, 2, 4},
                                {5, 4, 5, 4, 3, 4, 2} };
        memcpy(walk_time_table, tmp3, 7 * 7 * sizeof(int));

    }
    readPucks();
    readTickets();
    readGates();
    srand((int)time(NULL));
}
AirPort::~AirPort()
{
}
void AirPort::readPucks()
{
    string file = prefix + "/" + "data1.txt";
    fstream f;
    f.open(file, ifstream::in);
    assert(f.is_open());
    while (!f.eof()) {
        Pucks p;
        string line;
        getline(f, line);
        vector<string> words;
        splitBySpace(words, line);
        p.plane_num = words[0];
        // arrival_date-departure_date
        p.arrival_date = string2int(words[1].substr(0, 2));
        p.departure_date = string2int(words[6].substr(0, 2));
        // arrival/departure-time
        vector<string> time;
        splitByColon(time, words[2]);
        int hour, mins;
        hour = string2int(time[0]);
        mins = string2int(time[1]);
        p.arrival_time = hour * 12 + mins / 5;
        splitByColon(time, words[7]);
        hour = string2int(time[0]);
        mins = string2int(time[1]);
        p.departure_time = hour * 12 + mins / 5;
        p.arrival_flight = words[3];
        p.departure_flight = words[8];
        p.arrival_type = ( words[4] == "D" ? DEMOSTIC : (words[4] == "I" ? INTER :
D_OR_I) );
```

```
        p.departure_type = ( words[9] == "D" ? DEMOSTIC : (words[9] == "I" ? INTER :
D_OR_I) );
        p.flight_shape = map_flight_shape[words[5]];
        pucks.push_back(p);
    }
    for (int i = 0; i < pucks.size(); i++) {
        Pucks &p = pucks[i];
        p.flight_id = i; // flightId
        int exact_arrival_time = (p.arrival_date - 18) * 288 + p.arrival_time;
        int  exact_departure_time  =  (p.departure_date  -  18)  *  288  +
p.departure_time;
        a_i.push_back(exact_arrival_time);
        d_i.push_back(exact_departure_time);
    }
}
void AirPort::readTickets()
{
    string file = prefix + "/" + "data2.txt";
    fstream f;
    f.open(file, ifstream::in);
    assert(f.is_open());
    int i = 0;
    while (!f.eof()) {
        Tickets t;
        string line;
        getline(f, line);
        vector<string> words;
        splitBySpace(words, line);
        t.parssengerId = i++;
        t.num_parssengers = string2int(words[1]);
        t.arrival_flight = words[2];
        t.departure_flight = words[4];
        t.arrival_date = string2int(words[3].substr(0, 2));
        t.departure_date = string2int(words[5].substr(0, 2));
        tickets.push_back(t);
        sum_people += t.num_parssengers;
    }
}
void AirPort::readGates()
{
    string file = prefix + "/" + "data3.txt";
    fstream f;
    f.open(file, ifstream::in);
    assert(f.is_open());
```

```
    while (!f.eof()) {
        Gates g;
        string line;
        getline(f, line);
        vector<string> words;
        splitBySpace(words, line);
        string num = words[0].substr(1, words[0].size() - 2);
        g.name = words[0];
        g.gates_id = string2int(num);
        g.gates_id += (words[0][0] == 'T' ? 21 : 0);
        g.gates_type = (words[1] == "T" ? T : S);
        g.area = (words[2] == "North" ? NORTH :
                    (words[2] == "Center" ? CENTER :
                        (words[2] == "South" ? SOUTH : EAST)));
        g.arrival_resive_type = (words[3] == "D" ? DEMOSTIC :
            (words[3] == "I" ? INTER : D_OR_I));
        g.departure_release_type = (words[4] == "D" ? DEMOSTIC :
            (words[4] == "I" ? INTER : D_OR_I));
        g.flight_resive_shape = (words[5] == "W" ? WIDE : NARROW);
        gates.push_back(g);
    }
    K = gates.size();
}
void AirPort::solution1() {
    /* construct Sets */
    int totalFlights = pucks.size();
    for (int i = 0; i < totalFlights; i++) {
        if (pucks[i].departure_date >= 20 && pucks[i].arrival_date <= 20) {
            N.push_back(i);
        }
    }
    m = K + 1;
    n = N.size();
    for (vector<int>::const_iterator it = N.begin(); it != N.end(); it++) {
        vector<int> k_i;
        vector<int> nk_i;
        for (int j = 0; j < K; j++) {
            bool condition_1 = ( gates[j].arrival_resive_type == D_OR_I ||
                (pucks[*it].arrival_type == gates[j].arrival_resive_type) );

            bool condition_2 = ( gates[j].departure_release_type == D_OR_I ||
                (pucks[*it].departure_type                                    ==
gates[j].departure_release_type) );
```

```
            bool     condition_3     =     (     pucks[*it].flight_shape     ==
gates[j].flight_resive_shape );
            if (condition_1 && condition_2 && condition_3)
                nk_i.push_back(j);
            else
                k_i.push_back(j);
        }
        nk_i.push_back(m-1);

        K_i[*it] = k_i;
        NK_i[*it] = nk_i;
    }
    y_i.resize(n);
    z_ij.resize(n*n);
    fill(y_i.begin(), y_i.end(), 0);
    fill(z_ij.begin(), z_ij.end(), -1);
    secondary_objs.push_back(&AirPort::secondary_objective);
    masterAlgorithm(n, K, secondary_objs);
    outputResolut("res1", 1);
    y_i.clear();
    z_ij.clear();
    secondary_objs.clear();
    f_current.clear();
    f_next.clear();
    N.clear();
    K_i.clear();
    NK_i.clear();
}
int AirPort::findf_i(const Tickets &t) {
    int totalFlights = pucks.size();
    int f_1 = -1;
    int f_2 = -1;
    if (t.departure_date >= 20 && t.arrival_date <= 20) {
        for (int i = 0; i < totalFlights; i++) {
            if (t.arrival_date == pucks[i].arrival_date && t.arrival_flight ==
pucks[i].arrival_flight) {
                f_1 = i;
            }
            if (t.departure_date == pucks[i].departure_date && t.departure_flight
== pucks[i].departure_flight) {
                f_2 = i;
            }
        }
```

```
            if (f_1 > 0 && f_2 > 0) {
                a_p[t.parssengerId] = f_1;
                d_p[t.parssengerId] = f_2;
                return 1;
            }
        }
        return 0;
}
void AirPort::solution2()
{
        /* construct Sets */
        int totalFlights = pucks.size();
        int totalPassengers = tickets.size();
        set<int> setN;
        for (int i = 0; i < totalPassengers; i++) {
                int f_i = findf_i(tickets[i]);
                if (f_i > 0) {
                        P.push_back(i);
                        setN.insert(a_p[i]);
                        setN.insert(d_p[i]);
                }
        }
        for (int i = 0; i < totalFlights; i++) {
                if (pucks[i].departure_date >= 20 && pucks[i].arrival_date <= 20) {
                        setN.insert(i);
                }
        }
        for (set<int>::const_iterator it = setN.begin(); it != setN.end(); it++) {
                N.push_back(*it);
        }
        m = K + 1;
        n = N.size();

        for (vector<int>::const_iterator it = N.begin(); it != N.end(); it++) {
                vector<int> k_i;
                vector<int> nk_i;
                for (int j = 0; j < K; j++) {
                        bool condition_1 = (gates[j].arrival_resive_type == D_OR_I ||
                                (pucks[*it].arrival_type == gates[j].arrival_resive_type));

                        bool condition_2 = (gates[j].departure_release_type == D_OR_I ||
                                (pucks[*it].departure_type == gates[j].departure_release_type));

                        bool condition_3 = (pucks[*it].flight_shape == gates[j].flight_resive_shape);
```

```
                if (condition_1 && condition_2 && condition_3)
                    nk_i.push_back(j);
                else
                    k_i.push_back(j);
            }
            nk_i.push_back(m - 1);
            K_i[*it] = k_i;
            NK_i[*it] = nk_i;
        }
        y_i.resize(n);
        z_ij.resize(n*n);
        fill(y_i.begin(), y_i.end(), 0);
        fill(z_ij.begin(), z_ij.end(), -1);
        secondary_objs.push_back(&AirPort::solution2_objective);
        secondary_objs.push_back(&AirPort::secondary_objective);
        masterAlgorithm(n, K, secondary_objs);
        outputResolut("res2", 2);
        y_i.clear();
        z_ij.clear();
        secondary_objs.clear();
        f_current.clear();
        f_next.clear();
        a_p.clear();
        d_p.clear();
        N.clear();
        K_i.clear();
        NK_i.clear();
}
void AirPort::solution3()
{
        /* construct Sets */
        int totalFlights = pucks.size();
        int totalPassengers = tickets.size();
        set<int> setN;
        for (int i = 0; i < totalPassengers; i++) {
            int f_i = findf_i(tickets[i]);
            if (f_i > 0) {
                P.push_back(i);
                setN.insert(a_p[i]);
                setN.insert(d_p[i]);
            }
        }
        for (int i = 0; i < totalFlights; i++) {
```

```
        if (pucks[i].departure_date >= 20 && pucks[i].arrival_date <= 20) {
            setN.insert(i);
        }
    }
    for (set<int>::const_iterator it = setN.begin(); it != setN.end(); it++) {
        N.push_back(*it);
    }
    m = K + 1;
    n = N.size();
    for (vector<int>::const_iterator it = N.begin(); it != N.end(); it++) {
        vector<int> k_i;
        vector<int> nk_i;
        for (int j = 0; j < K; j++) {
            bool condition_1 = (gates[j].arrival_resive_type == D_OR_I ||
                (pucks[*it].arrival_type == gates[j].arrival_resive_type));
            bool condition_2 = (gates[j].departure_release_type == D_OR_I ||
                (pucks[*it].departure_type == gates[j].departure_release_type));
            bool condition_3 = (pucks[*it].flight_shape == gates[j].flight_resive_shape);

            if (condition_1 && condition_2 && condition_3)
                nk_i.push_back(j);
            else
                k_i.push_back(j);
        }
        nk_i.push_back(m - 1);
        K_i[*it] = k_i;
        NK_i[*it] = nk_i;
    }
    y_i.resize(n);
    z_ij.resize(n*n);
    fill(y_i.begin(), y_i.end(), 0);
    fill(z_ij.begin(), z_ij.end(), -1);
    secondary_objs.push_back(&AirPort::solution3_objective);
    secondary_objs.push_back(&AirPort::secondary_objective);
    masterAlgorithm(n, K, secondary_objs);
    outputResolut("res3", 3);
    y_i.clear();
    z_ij.clear();
    secondary_objs.clear();
    f_current.clear();
    f_next.clear();
    a_p.clear();
    d_p.clear();
    N.clear();
```

```
    K_i.clear();
    NK_i.clear();
}
float AirPort::primary_objective()
{
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += (y_i[i] == K);
    }
    return (float)sum;
}
float AirPort::secondary_objective()
{
    int sum = 0;
    for (int k = 0; k < m-1; k++) {
        int sgn = 0;
        for (int i = 0; i < n; i++) {
            sgn |= (y_i[i] == k);
        }
        sum += sgn;
    }
    return (float)sum;
}
float AirPort::solution2_objective() {
    map<int, float> a_sigma_1, a_sigma_2, a_sigma_3, a_sigma_4;
    map<int, float> d_sigma_1, d_sigma_2, d_sigma_3, d_sigma_4;
    for (int i = 0; i < n; i++) {
        int n_i = N[i];
        a_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == DEMOSTIC);
        a_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type ==
DEMOSTIC);
        a_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == INTER);
        a_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type == INTER);

        d_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == DEMOSTIC);
        d_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
DEMOSTIC);
        d_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == INTER);
        d_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
INTER);
    }
    vector<map<int, float> > a_sigmas;
    vector<map<int, float> > d_sigmas;
    a_sigmas.push_back(a_sigma_1);
```

```
    a_sigmas.push_back(a_sigma_2);
    a_sigmas.push_back(a_sigma_3);
    a_sigmas.push_back(a_sigma_4);
    d_sigmas.push_back(d_sigma_1);
    d_sigmas.push_back(d_sigma_2);
    d_sigmas.push_back(d_sigma_3);
    d_sigmas.push_back(d_sigma_4);
    map<int, float> t_p;
    for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
        int I = a_p[*p];
        int J = d_p[*p];
        float tmp = 0.0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                tmp += a_sigmas[i][I] * d_sigmas[j][J] * procedure_time_table[i][j];
            }
        }
        t_p[*p] = tmp;
    }
    float sum = 0;
    for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
        int I = a_p[*p];
        int J = d_p[*p];
        int S_p = (a_i[I] + t_p[*p] <= d_i[J]);
        sum += (t_p[*p] * S_p + 72 * (1-S_p)) * tickets[*p].num_parssengers;
    }
    return sum;
}
float AirPort::solution3_objective() {
    map<int, float> gamma_1, gamma_2, gamma_3, gamma_4, gamma_5, gamma_6, gamma_7;
    vector<map<int, float> > gammas;
    for (int i = 0; i < n; i++) {
        int n_i = N[i];
        gamma_1[n_i] = (y_i[i] >= 0 && y_i[i] < 9);
        gamma_2[n_i] = (y_i[i] >= 9 && y_i[i] < 19);
        gamma_3[n_i] = (y_i[i] >= 19 && y_i[i] < 28);
        gamma_4[n_i] = (y_i[i] >= 28 && y_i[i] < 38);
        gamma_5[n_i] = (y_i[i] >= 38 && y_i[i] < 48);
        gamma_6[n_i] = (y_i[i] >= 48 && y_i[i] < 58);
        gamma_7[n_i] = (y_i[i] >= 58 && y_i[i] < 69);
    }
    gammas.push_back(gamma_1);
    gammas.push_back(gamma_2);
    gammas.push_back(gamma_3);
```

```
    gammas.push_back(gamma_4);
    gammas.push_back(gamma_5);
    gammas.push_back(gamma_6);
    gammas.push_back(gamma_7);
    map<int, float> w_p;
    for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
        float tmp = 0.;
        int I = a_p[*p];
        int J = d_p[*p];
        for (int i = 0; i < 7; i++) {
            for (int j = 0; j < 7; j++) {
                tmp += gammas[i][I] * gammas[j][J] * walk_time_table[i][j];
            }
        }
        w_p[*p] = tmp;
    }
    map<int, float> a_sigma_1, a_sigma_2, a_sigma_3, a_sigma_4;
    map<int, float> d_sigma_1, d_sigma_2, d_sigma_3, d_sigma_4;
    for (int i = 0; i < n; i++) {
        int n_i = N[i];
        a_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == DEMOSTIC);
        a_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type ==
DEMOSTIC);
        a_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == INTER);
        a_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type == INTER);
        d_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == DEMOSTIC);
        d_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
DEMOSTIC);
        d_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == INTER);
        d_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
INTER);
    }
    vector<map<int, float> > a_sigmas;
    vector<map<int, float> > d_sigmas;
    a_sigmas.push_back(a_sigma_1);
    a_sigmas.push_back(a_sigma_2);
    a_sigmas.push_back(a_sigma_3);
    a_sigmas.push_back(a_sigma_4);
    d_sigmas.push_back(d_sigma_1);
    d_sigmas.push_back(d_sigma_2);
    d_sigmas.push_back(d_sigma_3);
    d_sigmas.push_back(d_sigma_4);
    map<int, float> t_pp;
    for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
```

```
            int I = a_p[*p];
            int J = d_p[*p];
            float tmp = 0.0;
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    tmp += a_sigmas[i][I] * d_sigmas[j][J] * tram_time_table[i][j] * 1.6;
                }
            }
            t_pp[*p] = tmp;
        }
        map<int, float> t_p;
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            int I = a_p[*p];
            int J = d_p[*p];
            float tmp = 0.0;
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    tmp += a_sigmas[i][I] * d_sigmas[j][J] * procedure_time_table[i][j];
                }
            }
            t_p[*p] = tmp;
        }
        float sum = 0.;
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            int I = a_p[*p];
            int J = d_p[*p];
            float r_p = (t_p[*p] + t_pp[*p] + w_p[*p]);
            int S_pp = (a_i[I] + r_p <= d_i[J]);
            sum += (float)(r_p * S_pp + 72 * (1 - S_pp)) / (d_i[J] - a_i[I]) *
tickets[*p].num_parssengers;
        }
        return sum;
}
bool AirPort::constrain_4()
{
        bool res = true;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < K; k++)
                    res = (res &&
                            (a_i[N[j]] + (z_ij[i*n + j] != k) * M >= d_i[N[i]] + beta));
                if (!res)
                    return false;
            }
```

```
        }
        return res;
}
/*  对限制函数 constrain_4 的一个重载  */
bool AirPort::constrain_4(const set<int> &changedK)
{
        bool res = true;
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        for (set<int>::const_iterator it = changedK.begin(); it != changedK.end(); it++)
                                res = ( res && (a_i[N[j]] + (z_ij[i*n + j] != *it) * M >= d_i[N[i]] + beta) );
                        if (!res)
                                return false;
                }
        }
        return res;
}
void AirPort::masterAlgorithm(int n, int K, vector<Objective> secondary_objs)
{
        float tConst = 2.0;
        float factor_1 = 2;
        float factor_2 = 2.25;
        float coolRate = 0.9999;
        float reheatFactor = 1.05;
        float terminateT = .01;
        int maxUnimproved = 600;
        int maxUnaccepted = 600;
        int unimproved = 0;
        int unaccepted = 0;
        float annealingTemp = tConst * n;
        /* Heuristics Algorithm */
        for (int i = 0; i < n; i++)
                y_i[i] = m - 1;
        calculateZij();
        /* best record */
        vector<float> bestF(1 + secondary_objs.size());
        vector<int> gateFlightMap(n);
        std::copy(y_i.begin(), y_i.end(), gateFlightMap.begin());
        vector<int> current(n);
        vector<int> current_zij(n*n);
        f_current.resize(1 + secondary_objs.size());
        f_next.resize(1 + secondary_objs.size());
        f_current[0] = primary_objective();
        for (int i = 0; i < secondary_objs.size(); i++)
```

```
        f_current[i+1] = (this->*secondary_objs[i])();
/* Simulated Annealing */
do {
        copy(y_i.begin(), y_i.end(), current.begin());
        copy(z_ij.begin(), z_ij.end(), current_zij.begin());
        set<int> changedK;
        randomMove(changedK);
        calculateZij(changedK);
        bool condition = constrain_4(changedK);
        if (condition) {
                float diff = diffCost();
                if (diff < 0) {
                        copy(y_i.begin(), y_i.end(), gateFlightMap.begin());
                        copy(f_next.begin(), f_next.end(), f_current.begin());
                        copy(f_next.begin(), f_next.end(), bestF.begin());
                }
                else {
                        //float r = distribution(engine);
                        float r = (float)(rand() % 100000001) / 100000001.0;
                        float probs = factor_1 * exp(-diff / (factor_2 * annealingTemp));
                        if (r < probs) {
                                unimproved += 1;
                                copy(f_next.begin(), f_next.end(), f_current.begin());
                        }
                        else {
                                unaccepted += 1;
                                copy(current.begin(), current.end(), y_i.begin());
                                copy(current_zij.begin(), current_zij.end(), z_ij.begin());
                        }
                }
        }
        else {
                unaccepted += 1;
                cout << "don't satisfy conditions " << endl;
                copy(current.begin(), current.end(), y_i.begin());
                copy(current_zij.begin(), current_zij.end(), z_ij.begin());
        }
        if (unimproved > maxUnimproved || unaccepted > maxUnaccepted) {
                annealingTemp *= reheatFactor;
                unimproved = 0;
                unaccepted = 0;
        }
        annealingTemp *= coolRate;
        cout << "tempature: " << annealingTemp << "\t";
```

```
            for (int i = 0; i < f_current.size(); i++)
                cout << "cost: " << f_current[i] << "\t";
            cout << endl;
        } while (annealingTemp > terminateT);
        copy(gateFlightMap.begin(), gateFlightMap.end(), y_i.begin());
        copy(bestF.begin(), bestF.end(), f_current.begin());
}
void AirPort::outputResolut(string dir, int solution) {
        printY_i(dir + "flightmap.txt");
        ofstream f;
        f.open(prefix + "/" + dir + "res.txt", ofstream::out);
        assert(f.is_open());
        for (int i = 0; i < f_current.size(); i++)
            f << f_current[i] << endl;
        f.close();
        Stat stat;
        Stat2 stat2;
        Stat3 stat3;
        switch (solution) {
        case 1:
            f.open(prefix + "/" + dir + "stat.txt", ofstream::out);
            statistic(stat);
            f << "ni\t" << stat.n_i << endl;
            f << "ei\t" << stat.e_i << endl;
            f << "hi\t" << stat.h_i << endl;
            f << "hip\t" << stat.h_i_p << endl;
            f << "gi\t" << stat.g_i << endl;
            f << "gip\t" << stat.g_i_p << endl;
            f << "li\t" << stat.l_i << endl;
            f << "lip\t" << stat.l_i_p << endl;
            f << "qi\t" << stat.q_i << endl;
            f << "qip\t" << stat.q_i_p << endl;
            f.close();
            break;
        case 2:
            f.open(prefix + "/" + dir + "stat1.txt", ofstream::out);
            statistic(stat);
            f << "ni\t" << stat.n_i << endl;
            f << "ei\t" << stat.e_i << endl;
            f << "hi\t" << stat.h_i << endl;
            f << "hip\t" << stat.h_i_p << endl;
            f << "gi\t" << stat.g_i << endl;
            f << "gip\t" << stat.g_i_p << endl;
            f << "li\t" << stat.l_i << endl;
```

```
            f << "lip\t" << stat.l_i_p << endl;
            f << "qi\t" << stat.q_i << endl;
            f << "qip\t" << stat.q_i_p << endl;
            f.close();
            f.open(prefix + "/" + dir + "stat2.txt", ofstream::out);
            statistic(stat2);
            f << "B\t" << stat2.B << endl;
            f << "O\t" << stat2.O << endl;
            f.close();
            break;
        case 3:
            f.open(prefix + "/" + dir + "stat1.txt", ofstream::out);
            statistic(stat);
            f << "ni\t" << stat.n_i << endl;
            f << "ei\t" << stat.e_i << endl;
            f << "hi\t" << stat.h_i << endl;
            f << "hip\t" << stat.h_i_p << endl;
            f << "gi\t" << stat.g_i << endl;
            f << "gip\t" << stat.g_i_p << endl;
            f << "li\t" << stat.l_i << endl;
            f << "lip\t" << stat.l_i_p << endl;
            f << "qi\t" << stat.q_i << endl;
            f << "qip\t" << stat.q_i_p << endl;
            f.close();
            f.open(prefix + "/" + dir + "stat2.txt", ofstream::out);
            statistic(stat3);
            f << "B\t" << stat3.B << endl;
            f << "O\t" << stat3.O << endl;
            f << "vt\t" << stat3.v_t << endl;
            f << "vtp\t" << stat3.v_t_p << endl;
            f.close();
            break;
        default:
            break;
        };
}
void AirPort::calculateZij() {

    for (int k = 0; k < K; k++) {
        for (int i = 0; i < n; i++) {
            if (y_i[i] != k)
                continue;
            for (int j = i + 1; j < n; j++) {
                if (y_i[j] != k)
```

```
                    continue;
                else {
                    z_ij[i*n + j] = k;

                    i = j - 1;
                    break;
                }

            }
        }
    }
}

/* 对函数 calculateZij 的一个重载 */
void AirPort::calculateZij(const set<int> &changedK) {

    for (set<int>::const_iterator it = changedK.begin(); it != changedK.end(); it++) {
        //int k = *it;
        for (int i = 0; i < n; i++) {
            if (y_i[i] != *it)
                continue;
            for (int j = i + 1; j < n; j++) {
                if (y_i[j] != *it)
                    continue;
                else {
                    z_ij[i*n + j] = *it;

                    i = j - 1;
                    break;
                }

            }
        }
    }
}

void AirPort::randomMove() {

    int i_1 = rand() % n;

    int k_1 = rand() % (NK_i[N[i_1]].size());

    y_i[i_1] = NK_i[N[i_1]][k_1];
}
```

```
/*  对函数 randomMove 的一个重载  */
void AirPort::randomMove(set<int> &changedK) {
	int i_1 = rand() % n;
	int k_1 = rand() % (NK_i[N[i_1]].size());

	if (y_i[i_1] < K)
		changedK.insert(y_i[i_1]);

	y_i[i_1] = NK_i[N[i_1]][k_1];

	if (y_i[i_1] < K)
		changedK.insert(y_i[i_1]);
}

/*   计算目标值增量  */
float AirPort::diffCost() {
	f_next[0] = primary_objective();
	for (int i = 0; i < secondary_objs.size(); i++)
		f_next[i+1] = (this->*secondary_objs[i])();

	vector<float> df;
	for (int i = 0; i < f_current.size(); i++) {
		df.push_back(f_next[i] - f_current[i]);
	}
	float factor = 300;

	for (int i = 0; i < df.size(); i++) {
		if (df[i] != 0.)
			return df[i] / f_current[i] *    factor;
	}
	return 0;
}

void AirPort::printY_i(string file)
{
	ofstream f;
	f.open(prefix + "/" + file, ofstream::out);
	assert(f.is_open());

	for (int i = 0; i < n; i++) {
		int k = y_i[i];
		if (k < K)
			f << gates[k].name << endl;
```

```
            else
                f << "temp Gate" << endl;
        }
        f.close();
}


void AirPort::statistic(Stat & stat)
{
        set<int> N_i;
        set<int> W_i;

        memset(&stat, 0, sizeof(stat));
        for (int i = 0; i < N.size(); i++) {
            stat.n_i += 2 * (y_i[i] != m-1);
            if (pucks[N[i]].flight_shape == NARROW) {
                N_i.insert(i);
            }
            else {
                W_i.insert(i);
            }

        }
        stat.e_i = (float)stat.n_i / (2 * N.size());

        for (set<int>::const_iterator it = N_i.begin(); it != N_i.end(); it++) {
            stat.h_i += 2 * (y_i[*it] != m - 1);
        }
        for (set<int>::const_iterator it = W_i.begin(); it != W_i.end(); it++) {
            stat.h_i_p += 2 * (y_i[*it] != m - 1);
        }
        stat.g_i = stat.h_i / (2 * N_i.size());
        stat.g_i_p = stat.h_i_p / (2 * W_i.size());

        stat.l_i = 0;
        for (int k = 0; k < 28; k++) {
            int sgn = 0;
            for (int i = 0; i < n; i++) {
                sgn += (y_i[i] == k);
            }
            stat.l_i += !!sgn;
        }
        stat.l_i_p = 0;
        for (int k = 28; k < 69; k++) {
```

```
        int sgn = 0;
        for (int i = 0; i < n; i++) {
            sgn += (y_i[i] == k);
        }
        stat.l_i_p += !!sgn;
}

vector<set<int> > A_k(K);
for (int k = 0; k < K; k++) {
        for (int i = 0; i < N.size(); i++) {
            if (y_i[i] == k)
                A_k[k].insert(i);
        }
}

stat.q_i = 0;
stat.q_i_p = 0;
float sum_1 = 0., sum_2 = 0.;

        float tmp = 0.0;
        for (int k = 0; k < 28; k++) {
            if (A_k[k].empty())
                continue;
            for (set<int>::const_iterator it = A_k[k].begin(); it != A_k[k].end(); it++) {
                int I = N[*it];
                int exact_time1 = 3 * 288;
                int exact_time2 = 2 * 288;
                tmp += MIN(d_i[I], exact_time1) - MAX(a_i[I], exact_time2);
                sum_1 += 1;
            }
        }
        stat.q_i += tmp / 288.;

        float tmp_1 = 0.;
        for (int k = 28; k < K; k++) {
            if (A_k[k].empty())
                continue;
            for (set<int>::const_iterator it = A_k[k].begin(); it != A_k[k].end(); it++) {
                    int I = N[*it];
                    int exact_time1 = 3 * 288;
                    int exact_time2 = 2 * 288;
                    tmp_1 += MIN(d_i[I], exact_time1) - MAX(a_i[I], exact_time2);
                    sum_2 += 1.;
            }
```

```
            }
            stat.q_i_p += tmp_1 / 288;


        stat.q_i /= sum_1;
        stat.q_i_p /= sum_2;
}

void AirPort::statistic(Stat2 & stat)
{
        map<int, float> a_sigma_1, a_sigma_2, a_sigma_3, a_sigma_4;
        map<int, float> d_sigma_1, d_sigma_2, d_sigma_3, d_sigma_4;
        memset(&stat, 0, sizeof(stat));
        for (int i = 0; i < n; i++) {
                int n_i = N[i];
                a_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == DEMOSTIC);
                a_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type ==
DEMOSTIC);
                a_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == INTER);
                a_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type == INTER);

                d_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == DEMOSTIC);
                d_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
DEMOSTIC);
                d_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == INTER);
                d_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
INTER);
        }
        vector<map<int, float> > a_sigmas;
        vector<map<int, float> > d_sigmas;
        a_sigmas.push_back(a_sigma_1);
        a_sigmas.push_back(a_sigma_2);
        a_sigmas.push_back(a_sigma_3);
        a_sigmas.push_back(a_sigma_4);
        d_sigmas.push_back(d_sigma_1);
        d_sigmas.push_back(d_sigma_2);
        d_sigmas.push_back(d_sigma_3);
        d_sigmas.push_back(d_sigma_4);

        map<int, float> t_p;
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
                int I = a_p[*p];
                int J = d_p[*p];
                float tmp = 0.0;
```

```
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    tmp += a_sigmas[i][I] * d_sigmas[j][J] * procedure_time_table[i][j];
                }
            }
            t_p[*p] = tmp;
        }

        stat.O = 0;
        float num_p = 0;
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            int I = a_p[*p];
            int J = d_p[*p];
            int S_p = (a_i[I] + t_p[*p] <= d_i[J]);
            stat.O += (1 - S_p) * tickets[*p].num_parssengers;
            num_p += tickets[*p].num_parssengers;
        }
        stat.B = stat.O / num_p;
}

void AirPort::statistic(Stat3 & stat)
{
        map<int, float> gamma_1, gamma_2, gamma_3, gamma_4, gamma_5, gamma_6, gamma_7;
        vector<map<int, float> > gammas;
        for (int i = 0; i < n; i++) {
            int n_i = N[i];
            gamma_1[n_i] = (y_i[i] >= 0 && y_i[i] < 9);
            gamma_2[n_i] = (y_i[i] >= 9 && y_i[i] < 19);
            gamma_3[n_i] = (y_i[i] >= 19 && y_i[i] < 28);
            gamma_4[n_i] = (y_i[i] >= 28 && y_i[i] < 38);
            gamma_5[n_i] = (y_i[i] >= 38 && y_i[i] < 48);
            gamma_6[n_i] = (y_i[i] >= 48 && y_i[i] < 58);
            gamma_7[n_i] = (y_i[i] >= 58 && y_i[i] < 69);
        }
        gammas.push_back(gamma_1);
        gammas.push_back(gamma_2);
        gammas.push_back(gamma_3);
        gammas.push_back(gamma_4);
        gammas.push_back(gamma_5);
        gammas.push_back(gamma_6);
        gammas.push_back(gamma_7);
        map<int, float> w_p;
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            float tmp = 0.;
```

```
            int I = a_p[*p];
            int J = d_p[*p];
            for (int i = 0; i < 7; i++) {
                for (int j = 0; j < 7; j++) {
                    tmp += gammas[i][I] * gammas[j][J] * walk_time_table[i][j];
                }
            }
            w_p[*p] = tmp;
        }
        map<int, float> a_sigma_1, a_sigma_2, a_sigma_3, a_sigma_4;
        map<int, float> d_sigma_1, d_sigma_2, d_sigma_3, d_sigma_4;
        for (int i = 0; i < n; i++) {
            int n_i = N[i];
            a_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == DEMOSTIC);
            a_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type ==
DEMOSTIC);
            a_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].arrival_type == INTER);
            a_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].arrival_type == INTER);

            d_sigma_1[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == DEMOSTIC);
            d_sigma_2[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
DEMOSTIC);
            d_sigma_3[n_i] = (y_i[i] < 28) * (pucks[N[i]].departure_type == INTER);
            d_sigma_4[n_i] = (y_i[i] >= 28 && y_i[i] < K) * (pucks[N[i]].departure_type ==
INTER);
        }
        vector<map<int, float> > a_sigmas;
        vector<map<int, float> > d_sigmas;
        a_sigmas.push_back(a_sigma_1);
        a_sigmas.push_back(a_sigma_2);
        a_sigmas.push_back(a_sigma_3);
        a_sigmas.push_back(a_sigma_4);
        d_sigmas.push_back(d_sigma_1);
        d_sigmas.push_back(d_sigma_2);
        d_sigmas.push_back(d_sigma_3);
        d_sigmas.push_back(d_sigma_4);
        map<int, float> t_pp;
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            int I = a_p[*p];
            int J = d_p[*p];
            float tmp = 0.0;
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    tmp += a_sigmas[i][I] * d_sigmas[j][J] * tram_time_table[i][j] * 1.6;
```

```
            }
        }
        t_pp[*p] = tmp;
}
map<int, float> t_p;
for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
        int I = a_p[*p];
        int J = d_p[*p];
        float tmp = 0.0;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                tmp += a_sigmas[i][I] * d_sigmas[j][J] * procedure_time_table[i][j];
            }
        }
        t_p[*p] = tmp;
}
float sum_p = 0.;
stat.O = 0;
for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
        int I = a_p[*p];
        int J = d_p[*p];
        float r_p = (t_p[*p] + t_pp[*p] + w_p[*p]);
        int S_pp = (a_i[I] + r_p <= d_i[J]);
        sum_p += tickets[*p].num_parssengers;
        stat.O += (1 - S_pp) * tickets[*p].num_parssengers;
}
stat.B = stat.O / sum_p;
float alpha[14] = { 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 };
vector<map<int, int> > mu_pt(14);
for (int i = 0; i < mu_pt.size(); i++) {
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            float r_p = (t_p[*p] + t_pp[*p] + w_p[*p]);
            mu_pt[i][*p] = (r_p <= alpha[i]);
        }
}
stat.v_t.resize(mu_pt.size());
fill(stat.v_t.begin(), stat.v_t.end(), 0);
for (int i = 0; i < mu_pt.size(); i++) {
        for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
            stat.v_t[i] += mu_pt[i][*p] * tickets[*p].num_parssengers;
        }
        stat.v_t[i] /= sum_p;
}
float mu_t[10] = { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 };
```

```
        vector<map<int, int> > mu_ptp(10);
        for (int i = 0; i < mu_ptp.size(); i++) {
            for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
                int I = a_p[*p];
                int J = d_p[*p];
                float r_p = (t_p[*p] + t_pp[*p] + w_p[*p]);
                int s_pp = (a_i[I] + r_p <= d_i[J]);
                float c_p = (r_p * s_pp + 72 * (1 - s_pp)) / (d_i[J] - a_i[I]);
                mu_ptp[i][*p] = (c_p <= mu_t[i]);
            }
        }
        stat.v_t_p.resize(mu_ptp.size());
        fill(stat.v_t_p.begin(), stat.v_t_p.end(), 0);
        for (int i = 0; i < mu_ptp.size(); i++) {
            for (vector<int>::const_iterator p = P.begin(); p != P.end(); p++) {
                stat.v_t_p[i] += mu_ptp[i][*p] * tickets[*p].num_parssengers;
            }
            stat.v_t_p[i] /= sum_p;
        }
}
```

主程序文件 “run.cpp”

```
#include <iostream>
#include "AirPort.hpp
using namespace std;
/*****************************************************************************/
/********************** 主程序入口 ***************************************/
/*****************************************************************************/

int main(int argc, char **argv) {
    AirPort A;
    A.solution1();
    A.solution2();
    A.solution3();
    return 0;
}
```