“华为杯”第十五届中国研究生
数学建模竞赛

学　　校		浙江大学
参赛队号		18103350032
队员姓名	1.	朱奕杰
	2.	张豪伟
	3.	张炜承

“华为杯”第十五届中国研究生

数学建模竞赛

题 目　　基于遗传算法的机场航班--登机口分配问题

摘　　　　要：

由于旅行业的快速发展，航空公司在机场的航站楼 T 的旅客流量已达饱和状态，为了应对未来的发展，需要增设卫星厅 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。因为不同登机口之间的距离不同，面对优化分配登机口的数量同时最小化旅客行走时间的问题，合理安排航班给对应的登机口显得尤为重要。本论文旨在建立飞机与登机口的配对关系来建立航班起降安排，利用合理的假设和条件对实际问题进行简化，基于遗传算法对问题代代寻优，最终的得到一个满意的航班-登机口分配方案。

对于问题一，不需要考虑旅客的因素，所以卫星厅和航站楼的登机口暂时可以不加区分，最小化登机口启用数量和航班停飞数量的目标函数。通过建立航班之间的时间约束和航班—登机口的属性约束，建立起整数非线性规划模型。根据元启发式算法——遗传算法，我们设计了求解该数学模型的具体策略。经过 250 次的子代生成，从末代中挑选出较优的可行解作为原问题的解。该解对应的成功分配到登机口的航班数量为 242，比例为 79.9%，T 里的登机口都开启，S 里的登机口有 2 个未开启。

对于问题二，加入了旅客的因素，所以需要区分卫星厅和航站楼的登机口。通过在问题一的目标函数里加入中转旅客最短流程时间指标，建立起类似的优化模型。用遗传算法求解该模型，得到一个较优的解，该解对应的成功分配到登机口的航班数量为 211，比例为 69.6%，T 里的登机口都开启，S 里的登机口有 2 个未开启，换乘失败的旅客的数量为 0。

对于问题三，考虑中转旅客中转过程中总共需要花费的时间与航班间隔之间的关系，通过将问题二的目标函数里的时间项替换为换乘紧张程度指标，建立第三个优化模型。经过 250 代后，算法求得最优解对应的成功分配到登机口的飞机

数为 235，比例为 77.6%，T 里的登机口都开启，S 里的登机口有 2 个未开启，换乘失败的旅客的数量为 0。

我们考虑到该问题的是航空公司关心的问题，因此本文从航空公司的角度出发，给出飞机分配的可行方案，并能提供多个备选的方案，以预防突发事件使得方案更贴近实际需求，同时考虑到机场内部的业务流程优化问题和旅客满意度问题。

关键词：遗传算法；整数规划；航班—登机口分配；旅客换乘时间

1. 问题重述

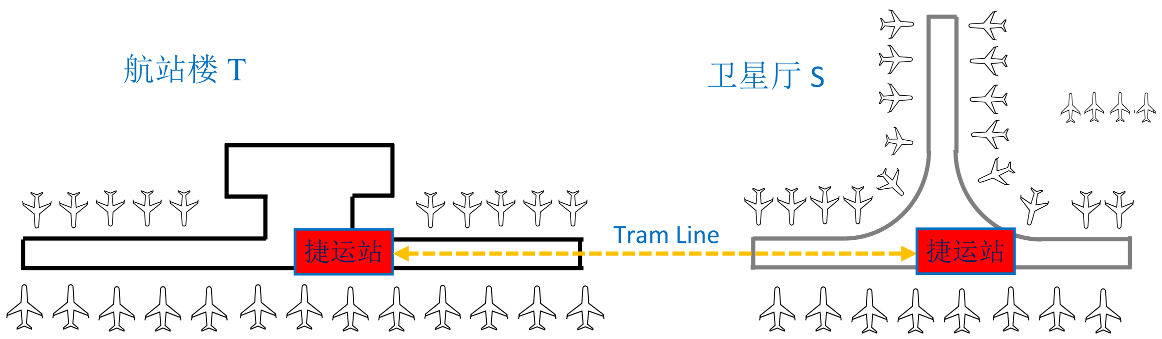
1.1 研究背景

由于旅行业的快速发展，某航空公司在某机场的现有航站楼 T 的旅客流量已达饱和状态，为了应对未来的发展，现正增设卫星厅 S。但引入卫星厅后，虽然可以缓解原有航站楼登机口不足的压力，对中转旅客的航班衔接显然具有一定的负面影响。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。航班—登机口分配问题可以表述成组合优化的数学模型，但是由于现实中需要考虑航班和登机口的多种属性和各种异常情况，该优化模型的约束条件往往具有多样性和模糊性。同时优化的目标函数也依据机场调度作业的需求而变化。

航班—登机口分配问题从上世纪七八十年代在优化决策领域已有研究，时至今日，机场的的规模越来越大，旅客的体量越来越大、航班的数量越来越多，因此航班—登机口的安排需要不断完善以满足日益增长的需求。伴随着优化模型的变量增多、约束更加复杂，虽然问题在数学上产生了本质没有变化，但是问题对求解的计算机算法提出越来越高的要求。在很多情况下，我们仍然需要在很短的时间内找到一个较优的可行解。

1.2 已知信息

本赛题取材于中国东方航空公司和上海浦东国际机场。飞机的一次停靠由一对航班标识，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。所谓合适的登机口就是登机口的大小与飞机的大小要匹配、登机口的国内/国际属性与航班的国内国际属性要匹配、同一个登机口的两架飞机的起降时间要有一定的间隔。所谓的中转旅客就是从到达航班换乘到由同一架或不同架飞机执行的出发航班的旅客。



卫星厅 S 相对于航站楼 T 示意图

航站楼 T 和卫星厅 S 的布局设计如上图所示。T 具有完整的国际机场航站功能，包括出发、到达、出入境和候机。新建的卫星厅 S 是航站楼 T 的延伸，可以候机，没有出入境功能。T 和 S 之间有捷运线相通，可以快速往来运送国内、国际旅客。假定旅客无需等待，随时可以发车，单程一次需要 8 分钟。本题只考虑中转的旅客，其他单程的旅客不予考虑。中转旅客从前一航班的到达至后一航班的出发之间的流程，按国内（D）和国际（I）、航站楼（T）和卫星厅（S）组合成 16 种不同的场景。本题中 T 中的登机口各分在三类区域之一（T-North, T-

Center, T-South）, S 中的登机口各分在四类区域之一（S-North, S-Center, S-South, S-East）。旅客流程和捷运时间时间只与(T/S)的(国内/国际)的(到达/出发)有关，旅客行走时间只与(T/S)的(登机口区域)有关，具体数据已有。另外，已有的数据还包含 19 号、20 号、21 号的航班信息和中转旅客的信息以及 T/S 登机口的固定信息。

1.3 问题提出

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

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

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

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

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

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

2. 模型假设

1. 仅对 20 号到达或出发的航班和旅客进行分析，19 号起飞 21 号降落的飞机不予考虑。
2. T 和 S 的所有登机口统筹规划分配，航班只能分配到与之属性相吻合的登机口。
3. 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。
4. 分配在同一登机口的两飞机之间的空挡间隔时间（后一架飞机的到达时间与前一架飞机的出发时间差）必须大于等于 45 分钟。
5. 机场另有简易机位，供分配不到固定登机口的飞机停靠，假定机位数量无限制。飞机到达临时机位后不再考虑它的出发航班。
6. 飞机总是准点出发和到达。
7. 假设机场可以全天 24 小时工作。
8. 假设无旅客、机场、天气和没有其他类型的突发情况或不可抗力发生。

3. 符号说明

问题参数：

N={1,2,…,|N|}：飞机的索引值集合；

M={1,2,…,|M|}：登机口的索引值集合，前 28 个为 T 的登机口，后 41 个为 S 的登机口，|M|=69；

A_i ：第 i 架飞机的到达时间；

D_i ：第 i 架飞机的出发时间；

u_i ：第 i 架飞机的宽窄类别，如果是宽的，$u_i = 1$，如果是窄的，$u_i = 0$；

v_k ：第 k 个登机口的宽窄类别，如果是宽的，$v_k = 1$，如果是窄的，$v_k = 0$；

$\text{A\_type}_{i,m}$ ：第 i 架飞机的到达属性（国内 D/国际 I）和出发属性（同上），$m \in \{1,2,3,4\}$ 分别对应 II,ID,DI,DD，并且 $\text{A\_type}_{i,m} \in \{0,1\}$，$\sum_{m\in\{1,2,3,4\}} \text{T\_type}_{i,m} = 1$。例如$\text{A_type}_{10,2} = 1$表示第 10 架飞机的到达出发属性是国际（I）到达和国内（D）出发。

$\text{T\_type}_{k,m}$ ：第 k 个登机口的到达属性（国内 D/国际 I）和出发属性（同上），$m \in \{1,2,3,4\}$ 分别对应 II,ID,DI,DD，并且 $\text{T\_type}_{i,m} \in \{0,1\}$，$1 \le \sum_{m\in\{1,2,3,4\}} \text{T\_type}_{i,m} \le 4$。例如$\text{T_type}_{10,2} = 1$表示第 10 个登机口允许国际 I 到达，国内 D 出发。

λ：目标函数的超参数，权衡登机口开启的数量和活动的航班数量之间的关系；

问题变量：

$x_{i,k}$ ：二元决策分配变量，如果第 i 架飞机被分配到第 k 个登机口，$x_{i,k} = 1$，否则$x_{i,k} = 0$。

$x_{i,|M+1|}$ ：如果第 i 架飞机被分配到临时机位，$x_{i,|M+1|} = 1$，否则$x_{i,|M+1|} = 0$。

4. 问题一模型建立与求解

4.1 问题分析及模型建立

问题一要求最大化航班的活动数量的同时最小化登机口的启用数量，在第二节的假设下，每架飞机对应两个航班，并且两个航班同时活动或同时关闭，所以问题近似为最大化飞机的活动数量的同时最小化登机口的启用数量，即最小化飞机的休眠数量（停在临时机位上的飞机数量）的同时最小化登机口的启用数量。由于仅考虑 20 号到达或离开的航班，在对原数据进行筛选后，只需要考虑 303 架飞机，即|N|=303。

（1） 目标函数的建立

根据第三节的变量说明，飞机的休眠数量为$x_{i,|M+1|}$对所有的 i 求和，登机口的启用数量为变量open_k对所有的 k 求和，其中open_k表示第 k 个登机口是否启用，定义如下

$$\text{open}_k \triangleq \begin{cases} 1, & if \;\; \sum_{i\in N} x_{i,k} \geq 1 \\ 0, & otherwise. \end{cases} \tag{4.1}$$

可建立如下的目标函数

$$\min_{\{x_{i,k}\}} \lambda \sum_{i\in N} x_{i,|M+1|} + (1-\lambda)\sum_{k\in M} open_k \tag{4.2}$$

在式（4.2）中，参数$\lambda \in (0,1]$可以人为调节，从而找到两项最优的平衡点，默认为 0.9。更进一步，由于第一项和第二项可能相差一个数量级，为了直观地体现λ的意义，我们将两项分别除以各自的总数，转换成比重的形式，即

$$\min_{\{x_{i,k}\}} \text{F} \triangleq \lambda \sum_{i\in N} x_{i,|M+1|}/|N| + (1-\lambda)\sum_{k\in M} open_k\,/|M| \tag{4.3}$$

由题设，我们将已知条件作为目标函数（4.3）的约束条件引入优化模型。

（2）约束条件

a. 飞机分配约束：对每一架飞机来说，它只能分配给唯一的一个登机口或者临时机位，故存在约束如下

$$\sum_{k\in M\cup\{|M|+1\}} x_{i,k} = 1, \qquad i \in N \tag{4.4}$$

b. 飞机特性约束：对每一架飞机，它都有宽窄之分，以及到达类型（国内/国际）和出发类型（国内/国际），由于飞机在同一个登机口到达和起飞，所以登机口对应的特性必须匹配飞机的这两个特性，故存在约束如下

$$(u_i - v_k)x_{i,k} = 0, \qquad i \in N, k \in M \tag{4.5}$$

$$\left(\sum_{m\in\{1,2,3,4\}} A\_type_{i,m} * T\_type_{i,m} - 1\right)x_{i,k} = 0,\; i \in N, k \in M \tag{4.6}$$

c. 飞机时间约束：每一个登机口在任何时刻都只能停一架飞机，从一架飞机降落后直到起飞都占据在该登机口，这意味着下一架飞机的到达时刻必须大于当前飞机的出发时刻，同时要求这个时间间隔大于等于 45 分钟，故存在约束如下

$$\left(D_j - A_i\right)(D_i - A_j)x_{i,k}x_{j,k} \leq 0, \qquad i,j \in N, k \in M \tag{4.7}$$

$$|D_i - A_j| \geq 45x_{i,k}x_{j,k}, \qquad i,j \in N, k \in M \tag{4.8}$$

综合上述公式，对问题一我们可以建立如下的优化模型：

$$\min_{\{x_{i,k}\}} \lambda \sum_{i\in N} x_{i,|M+1|} /|\mathrm{N}| + (1-\lambda) \sum_{k\in M} open_k /N$$

$$\text{s. t.} \begin{cases} \sum_{k\in M\cup\{|M|+1\}} x_{i,k} = 1 & i \in N \\ (u_i - v_k)x_{i,k} = 0 & i\in N, k\in M \\ \left(\sum_{m\in\{1,2,3,4\}} A\_type_{i,m} * T\_type_{i,m} - 1\right)x_{i,k} = 0 & i\in N, k\in M \\ (D_j - A_i)(D_i - A_j)x_{i,k}x_{j,k} \le 0 & i,j\in N, k\in M \\ |D_i - A_j| \ge 45x_{i,k}x_{j,k}, & i,j\in N, k\in M \\ x_{i,k} = 0,1 & i\in N, k\in M\cup\{|M|+1\} \\ \mathrm{open}_k = \begin{cases} 1, & if \ \ \sum_{i\in N} x_{i,k} \ge 1 \\ 0, & otherwise. \end{cases} \end{cases} \tag{4.9}$$

4.2 模型的求解

前面建立的航班-登机口分配优化问题是一个组合优化问题[1][2][3] 。且含有多个的约束条件，常用的精确算法（例如分支定界算法）很难求解出如此大规模的优化模型的最优解或近似最优解，因此我们提出基于遗传算法的元启发式求解方法来求解该问题。遗传算法通过模拟自然进化过程搜索最优解，其主要的思想是首先产生初代种群，按照适者生存和优胜劣汰的原理，逐代演化产生出越来越好的可行解，然后在每一代，根据问题域中个体的适应度大小(个体的适应度越大，个体对应的解越优)选择个体，并组合交叉和变异，产生出代表新的解集的种群。我们为这个优化模型定制了一套遗传算法，其中适应度根据目标函数 F 定义：

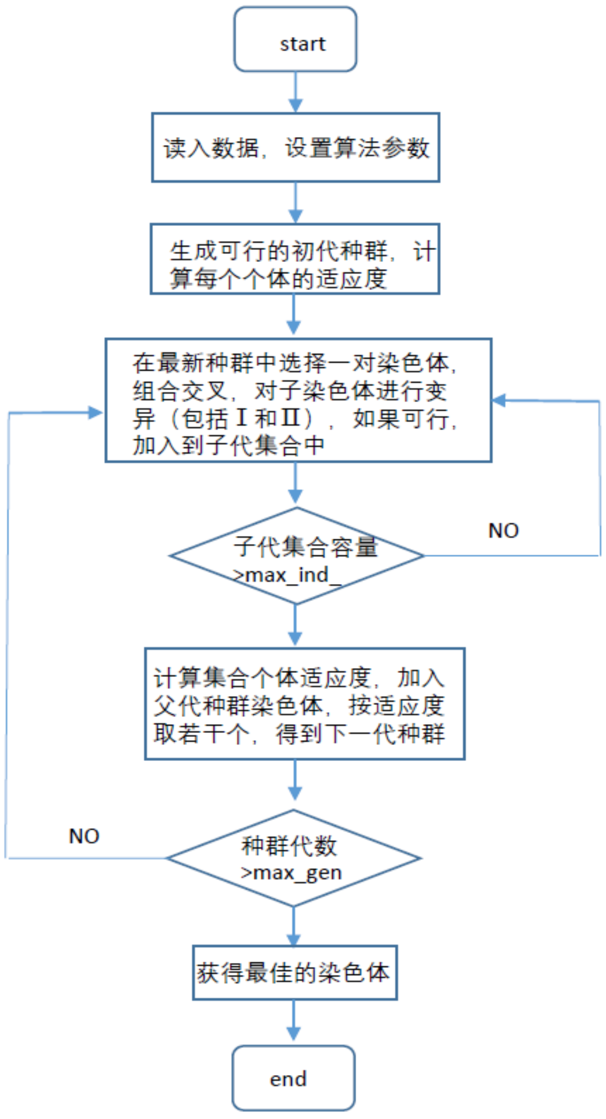
$$\mathrm{Fitness} \triangleq \lambda\left(1 - \sum_{i\in N}\frac{x_{i,|M+1|}}{|N|}\right) + (1-\lambda)(1 - \frac{\sum_{k\in M} open_k}{|M|}) \tag{4.10}$$

表示与生成：每一条染色体代表一个问题的一个解，与一个特定的二元决策分配变量相对应。具体来说，我们用一个 N 位的整数字符串表示一条染色体，字符串的第 i 位（从左至右数）对应的数字 k(i)表示第 i 架飞机被分配给第 k(i)登机口。在随机地初始化初代种群的时候，我们将宽窄和到达出发属性的约束条件考虑进去，这样能够大大降低随机生成的染色体的不可行程度。我们从左至右一次对每一个基因（即飞机）随机赋予一个可行的（即满足飞机特性约束条件）登机口的序号，使得这个基因不会与它之前的基因产生矛盾（即不满足时间约束条件）。

选择与交叉：the Roulette Wheel method 方法被用来从当前最新代的种群中选择父母，其主要的想法是适应度越高的染色体的被选中的概率越大。将父母染色体进行组合交叉，交叉的方法是 one-point crossover method。其主要的规则是产生一个 1 至 N-1 的随机数 i，将一条染色体的前 i 个基因与另一条染色体的后 N-i 个基因拼接起来，产生一个新的染色体。

变异：由于交叉得到的染色体在很大概率上不是可行解，如果随机选择一个基因变异，新得到的染色体很难满足所有的约束，所以，我们在变异操作上又多

加了三条规则。首先，对不可行的染色体进行变异；其次，像初始化初代种群那样，遍历基因，对第一个出现时间冲突的基因进行变异，接着继续向下遍历，最多变异 m 次，如果变异后的染色体还是不可行，就抛弃该染色体。m 的值不能过大，因为许多染色体距离最优染色体差异较大，修改的次数过多只会让染色体中取值为 70（对应临时机位）的基因增多，而我们的一个目标是使染色体中取值为 70 的基因尽可能少。因此，第三条规则要减少上述操作中得到的染色体中取值为 70 的基因的个数，即对于那些取值为 70 的基因我们再对其进行变异，与上一个变异操作不同的是这次变异不把 70 作为候选值。我们将这两类变异操作分别记为变异Ⅰ和变异Ⅱ，这个过程平衡了可行的染色体中取值为 70 的基因个数。总的来说，遗传算法的执行过程如下图所示：


start
读入数据，设置算法参数
生成可行的初代种群，计算每个个体的适应度
在最新种群中选择一对染色体，组合交叉，对子染色体进行变异（包括Ⅰ和Ⅱ），如果可行，加入到子代集合中
子代集合容量 >max_ind_
NO
计算集合个体适应度，加入父代种群染色体，按适应度取若干个，得到下一代种群
种群代数 >max_gen
NO
获得最佳的染色体
end

流程图：遗传算法流程示意（注：max_ind 为最大生成染色体数，max_gen 为最大代数）

用上述算法求解本体的过程中，设置$\lambda = 0.9$，每个种群里的个体数为 20 个，子代集合最大容量为 20 个，一共生成 250 代，对每条染色体的最大变异次数为 50 次，程序一次运行时间约 40min。 当$\lambda = 0.1$时，最优解下的登机口启用数为 63，被成功分配到登机口的飞机数量为 212；当$\lambda = 0.9$时，从最优解看，成功分

配到登机口的航班数量为 242，比例为 79.9%，T 里的登机口都开启，S 里的登机口有 2 个未开启（即 0.9 好于 0.1）。在$\lambda = 0.9$的情况下，图 1-1、1-2、1-3 依此表示窄体机线状图、宽体机线状图、登机口占用时间比率线状图。对结果数据进行分析，我们发现 S29、S30 两个登机口的属性是国内达到和出发并且为宽体，但数据中不存在与之属性匹配的航班，因此这两个登机口总是闲置的。另外，图 1-4 显示了对这个模型采用遗传算法的解的迭代收敛行为。

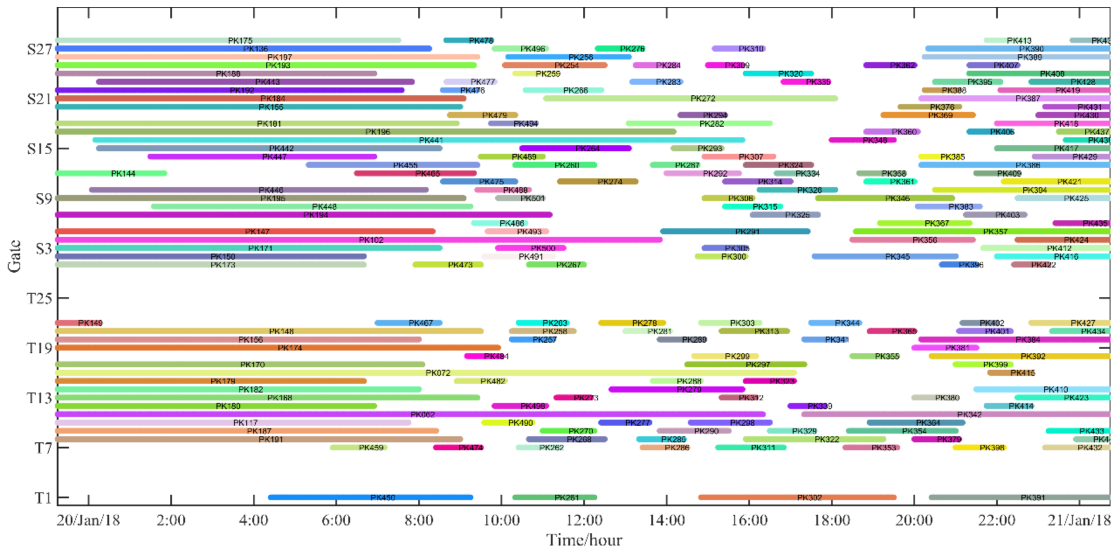


图 1-1：窄体登机口的分配情况

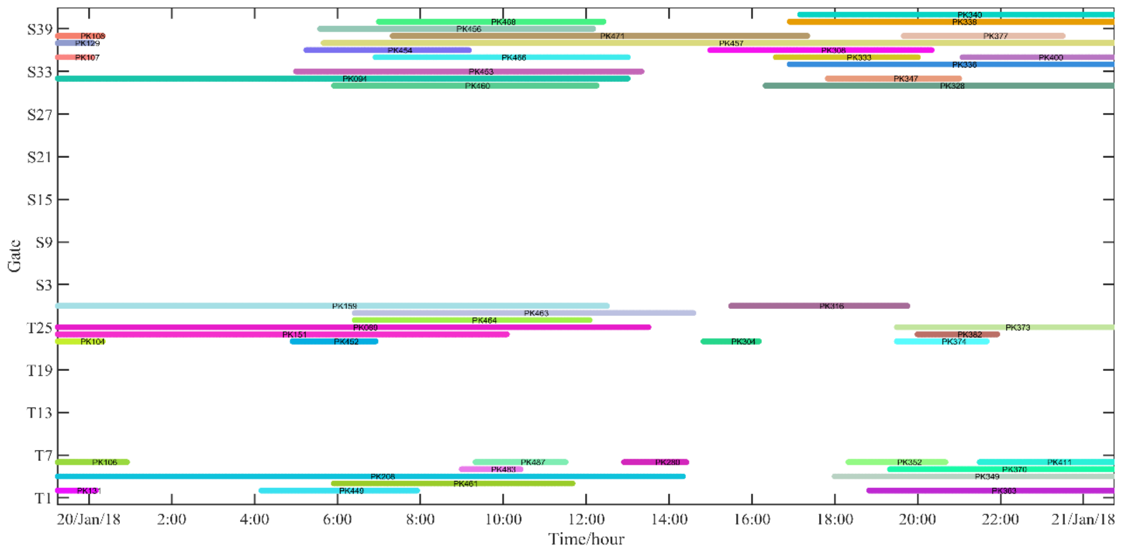


图 1-2：宽体登机口的分配情况

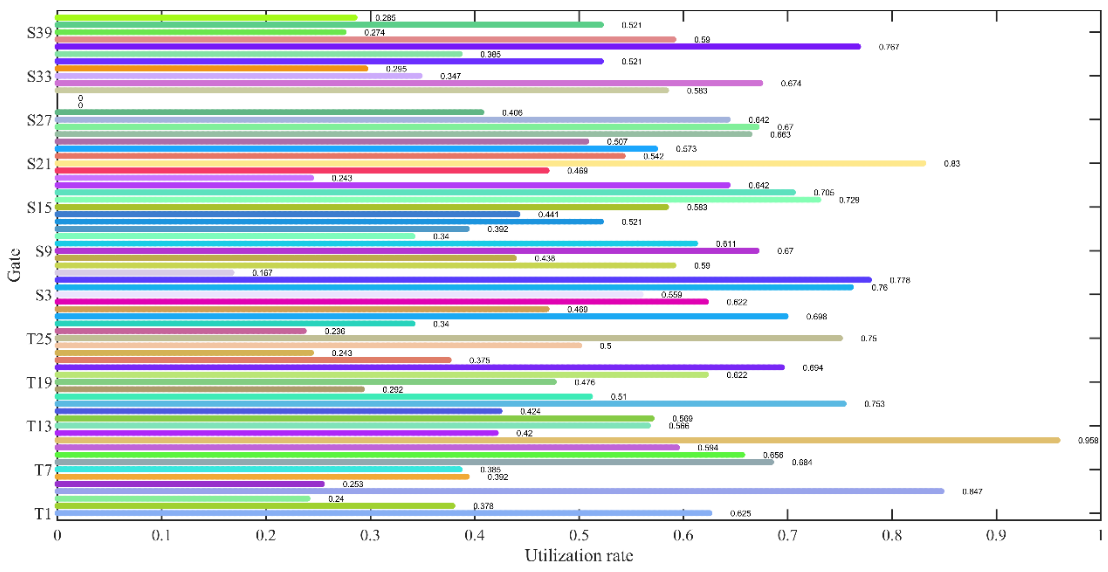


图 1-3：登机口占用时间比率

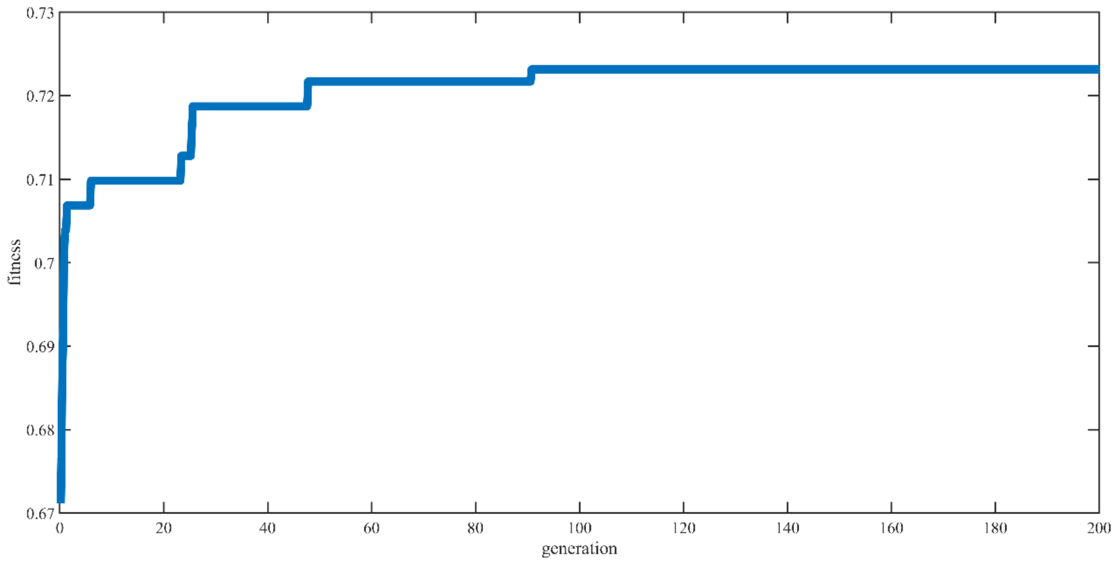


图 1-4：遗传算法过程（纵坐标为对应代数的种群中最好的适应度）

5. 问题二模型建立与求解

5.1 问题分析及模型建立

问题二要求在问题一的基础上考虑中转旅客的总体最短流程时间，因此优化的顺序应该是先优化航班休眠数，再优化旅客时间，最后优化登机口数量。为此，需要引入旅客的信息，这里我们仅考虑 20 号到达或 20 号离开的旅客。

（1） 目标函数的建立

在模型（4.9）的基础上，我们进行部分的修改。引入以下符号定义：P为本问题需考虑的旅客数量，$p_{i,j}$为从第 i 架飞机到第 j 架飞机的旅客总数，$w_{k,l}$为从第 k 个登机口到第 l 个登机口的最短流程时间。其中$\{w_{k,l}\}$是与$\{x_{i,j}\}$相关的变量集。那么中转旅客的总体最短流程时间为

$$t_1 \triangleq \sum_{i\in N}\sum_{j\in N}\sum_{k\in M}\sum_{l\in M} p_{i,j} w_{k,l} x_{i,k} x_{j,l} \tag{5.1}$$

另外，我们估计旅客的平均总体最短流程时间为 T_1 =35min 这样目标函数就可以定义为：

$$G \triangleq \left(1-\sum_{i\in N}\frac{x_{i,|M+1|}}{|N|}\right)+\lambda\frac{t_1}{T_1 P}+\lambda\gamma\left(1-\frac{\sum_{k\in M} open_k}{|M|}\right),\quad \lambda,\gamma\in(0,1] \tag{5.2}$$

这个目标函数的中的每一项依此表示了活动的飞机的比重、中转旅客总体最短流程时间与估计的总体旅客最短流程时间的比值、关闭的登机口数量的比重，常量系数的设置显示出第一项的优先度大于第二项的优先度大于第三项的优先度。

（2）约束条件

与问题一一致。

为了与 5.2 节中的求解算法中保持一致，我们将问题表述成最大化目标函数 G 的形式：

$$\max_{\{x_{i,k}\}}\left(1-\sum_{i\in N}\frac{x_{i,|M+1|}}{|N|}\right)+\lambda\frac{\sum_{i\in N}\sum_{j\in N}\sum_{k\in M}\sum_{l\in M} p_{i,j} w_{k,l} x_{i,k} x_{j,l}}{T_1 P}+\lambda\gamma\left(1-\frac{\sum_{k\in M} open_k}{|M|}\right)$$

$$\text{s.t.}\begin{cases}\sum_{k\in M\cup\{|M|+1\}} x_{i,k}=1 & i\in N\\ (u_i-v_k)x_{i,k}=0 & i\in N,k\in M\\ \left(\sum_{m\in\{1,2,3,4\}} A\_type_{i,m} * T\_type_{i,m}-1\right)x_{i,k}=0 & i\in N,k\in M\\ (D_j-A_i)(D_i-A_j)x_{i,k}x_{j,k}\le 0 & i,j\in N,k\in M\\ |D_i-A_j|\ge 45x_{i,k}x_{j,k}, & i,j\in N,k\in M\\ x_{i,k}=0,1 & i\in N,k\in M\cup\{|M|+1\}\\ open_k=\begin{cases}1, & \\ 0, & \end{cases} & \begin{matrix} if\ \ \sum_{i\in N} x_{i,k}\ge 1\\ otherwise.\end{matrix}\end{cases} \tag{5.3}$$

5.2 模型的求解

模型（5.3）与模型（4.9）的约束条件相同，仅仅是对目标函数做了一些改进，因此模型（4.9）的遗传算法的框架基本适用于问题二的求解。重新定义适应度：

$$Fitness \triangleq G \tag{5.4}$$

我们尝试流程 1 中的策略，可以得到一组可行解 C，在集合 C 里挑选出适应度最高的解作为问题二的近似最优解。

此外，我们还尝试了一种类似的策略，将问题一和问题二串联起来，按问题一提出的遗传算法（包括 Fitness）先求出子代种群，然后根据问题二适应度求出子代种群的子代种群。类似这样地轮换问题一二中的适应度的一个好处是使种群中的个体尽量多样化。

用前面流程图中策略求解本题的过程中，设置$\lambda = 1, \gamma = 0.5$，每个种群里的个体数为 20 个，子代集合最大容量为 20 个，一共生成 250 代，对每条染色体的最大变异次数为 50 次，程序一次运行时间约 45min。其他条件不变的情况下，改变目标函数后，从结果数据来看，被成功分配的飞机数会减少，航班间的时间间隔会增大，中转旅客平均换乘的流程时间会减少，这是因为在目标函数里加了时间的约束项，为了增多被成功分配的飞机数，我们可以在算法迭代过程中，在高适应度的前提下，挑选登机口启用数量较多的个体加入子代种群（例如要求最低被成功分配的飞机数为 200），从最优解看，成功分配到登机口的飞机数为 211，比例为 69.6%，T 里的登机口都开启，S 里的登机口有 2 个未开启，图 2-1、2-2、2-3 依此表示窄体机线状图、宽体机线状图、登机口占用时间比率线状图。换乘失败的旅客的数量为 0，比率为 0，图 2-4、2-5 依此表示总体旅客换乘时间分布图、总体乘客换乘紧张度分布图。

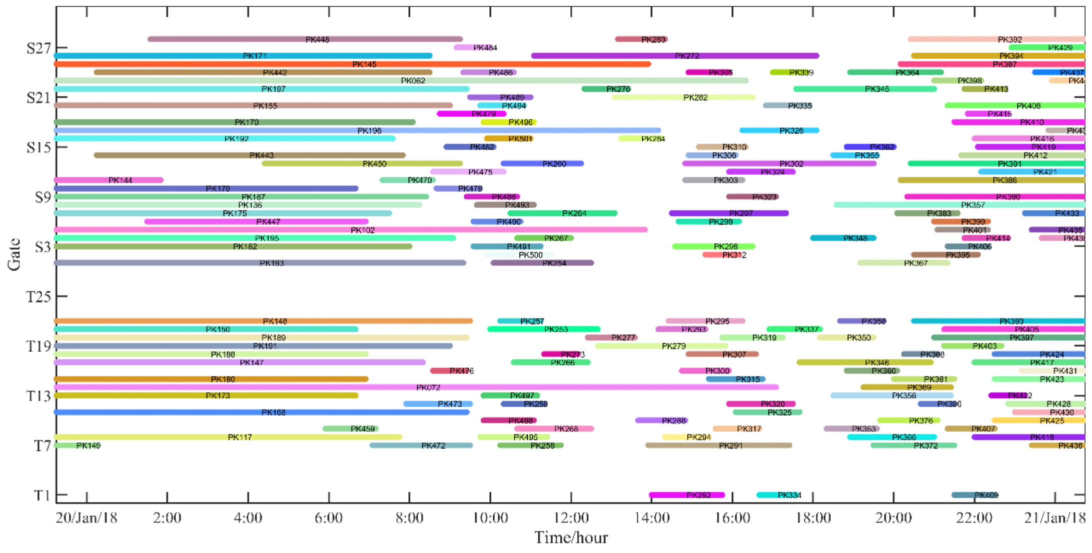


图 2-1：窄体登机口的分配情况

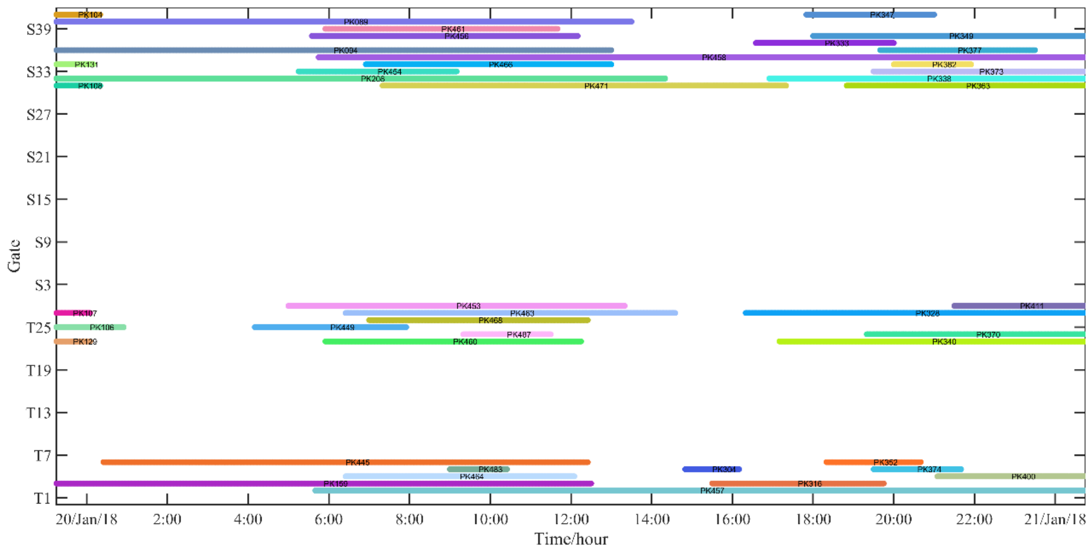


图 2-2：宽体登机口的分配情况

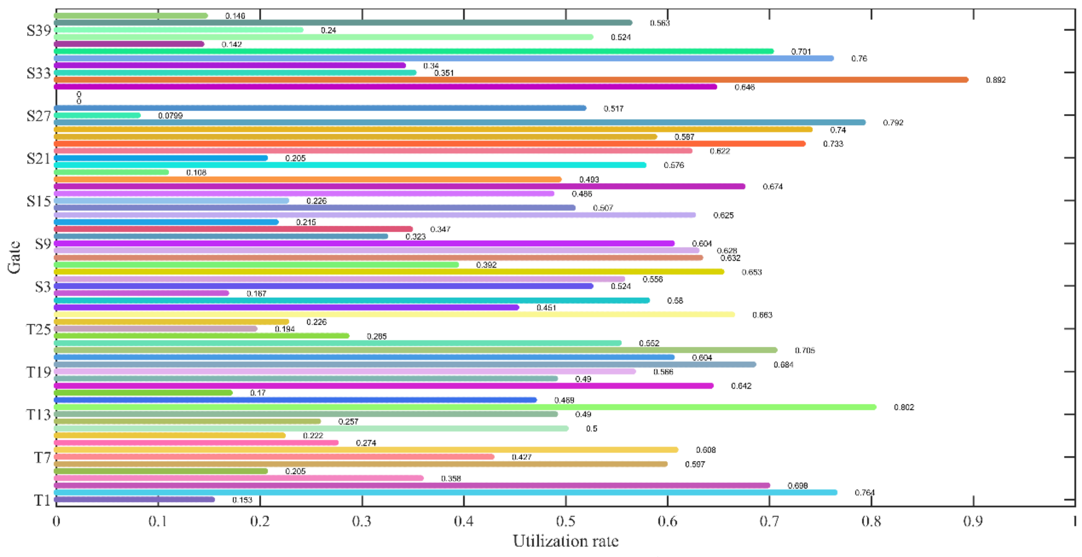


图 2-3：登机口占用时间比率

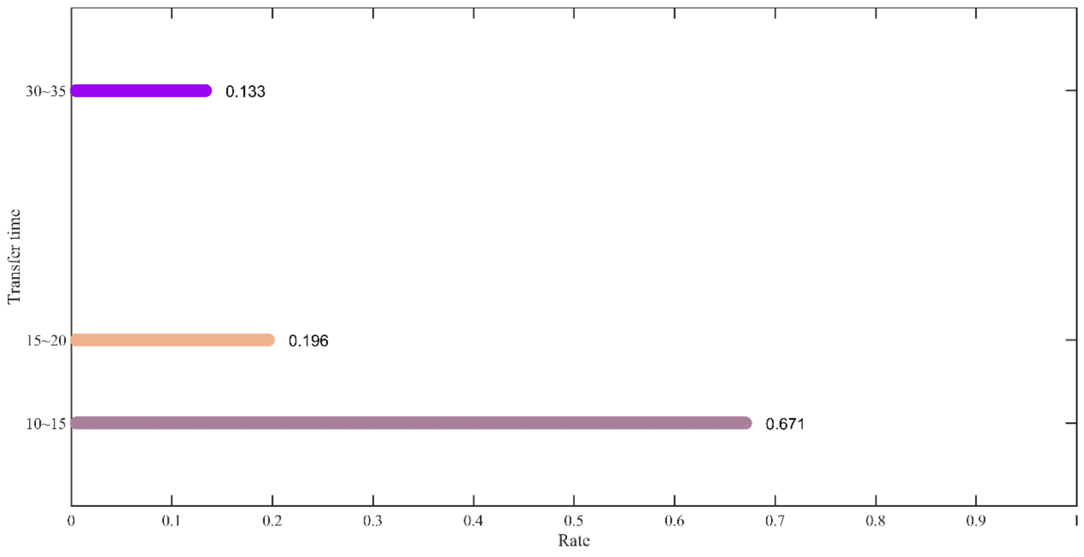


图 2-4：总体旅客换乘时间分布图

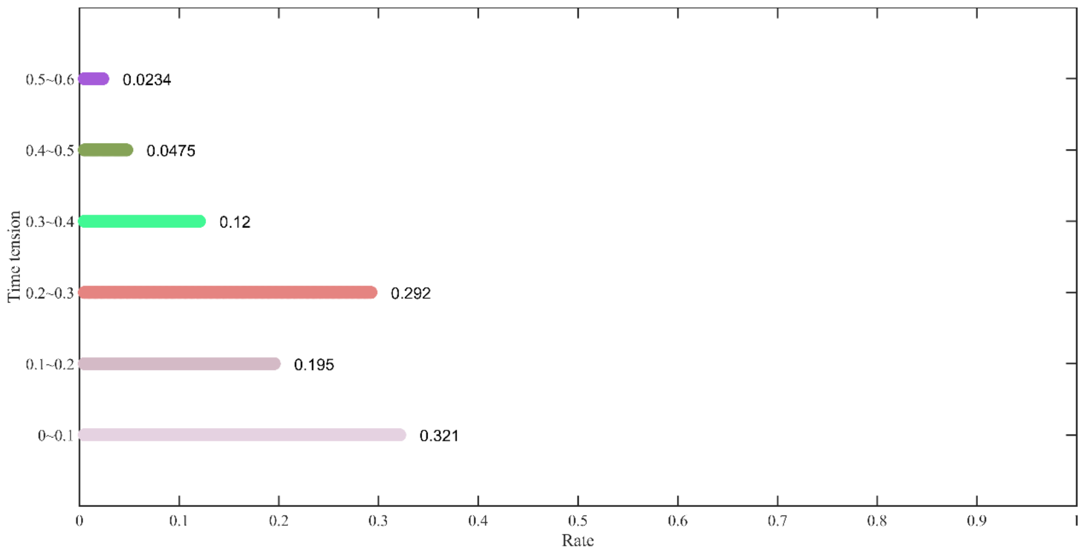


图 2-5：总体乘客换乘紧张度分布图

6. 问题三模型建立与求解

6.1 问题分析及模型建立

问题三要求考虑中转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。因此优化的顺序应该是先优化航班休眠数，再优化中转旅客总体紧张度，最后优化登机口数量。为此，需要引入旅客的信息，同问题一和问题二，我们仅考虑 20 号到达或 20 号离开的旅客。

（1）目标函数的建立

在模型（5.3）的基础上，我们进行部分的修改。引入以下符号定义：P为本问题需考虑的旅客数量，$\mathrm{p}_{i,j}$为从第 i 架飞机到第 j 架飞机的旅客总数，$\mathrm{w}^1{}_{k,l}$为从第 k 个登机口到第 l 个登机口的旅客换乘时间。其中$\{\mathrm{w}'_{k,l}\}$是与$\{x_{i,j}\}$相关的变量集。那么中转旅客的总体紧张度为

$$\mathrm{t}_2 \triangleq \sum_{i\in N}\sum_{j\in N}\sum_{k\in M}\sum_{l\in M}\mathrm{p}_{i,j}\frac{\mathrm{w}'_{k,l}}{\mathrm{D}_j-\mathrm{A}_i}x_{i,k}x_{j,l} \tag{6.1}$$

因为t_2的取值在 1 附近，所以t_2可以直接加到目标函数中。这样，目标函数就可以定义为：

$$\Gamma \triangleq \left(1-\sum_{i\in N}\frac{x_{i,|M+1|}}{|N|}\right)+\frac{\lambda}{\mathrm{t}_2}+\lambda\gamma\left(1-\frac{\sum_{k\in M}open_k}{|M|}\right),\quad \lambda,\gamma\in(0,1] \tag{6.2}$$

这个目标函数的中的每一项依此表示了活动的飞机的比重、中转旅客总体紧张度、关闭的登机口数量的比重，常量系数的设置显示出第一项的优先度大于第二项的优先度大于第三项的优先度。

（2）约束条件

与问题一一致。

为了与问题二中的求解算法中保持一致，我们将问题表述成最大化目标函数Γ的形式：

$$\max_{\{x_{i,k}\}}\left(1-\sum_{i\in N}\frac{x_{i,|M+1|}}{|N|}\right)+\frac{\lambda}{\sum_{i\in N}\sum_{j\in N}\sum_{k\in M}\sum_{l\in M}\mathrm{p}_{i,j}\frac{\mathrm{w}'_{k,l}}{\mathrm{D}_j-\mathrm{A}_i}x_{i,k}x_{j,l}}+\lambda\gamma\left(1-\frac{\sum_{k\in M}open_k}{|M|}\right)$$

$$\text{s. t.}\begin{cases}\sum_{k\in M\cup\{|M|+1\}}x_{i,k}=1 & i\in N\\ (u_i-v_k)x_{i,k}=0 & i\in N,k\in M\\ \left(\sum_{m\in\{1,2,3,4\}}A\_type_{i,m}*T\_type_{i,m}-1\right)x_{i,k}=0 & i\in N,k\in M\\ (D_j-A_i)(D_i-A_j)x_{i,k}x_{j,k}\le 0 & i,j\in N,k\in M\\ |D_i-A_j|\ge 45x_{i,k}x_{j,k}, & i,j\in N,k\in M\\ x_{i,k}=0,1 & i\in N,k\in M\cup\{|M|+1\}\\ \mathrm{open}_k=\begin{cases}1,\\0,\end{cases} & \begin{array}{r}if\ \ \sum_{i\in N}x_{i,k}\ge 1\\ otherwise.\end{array}\end{cases} \tag{6.3}$$

6.2 模型的求解

模型（6.3）与模型（5.3）的约束条件相同，仅仅是对目标函数做了一些改进，因此模型（5.3）的遗传算法的框架基本适用于问题三的求解。重新定义适应度：

$$\text{Fitness} \triangleq \Gamma \tag{6.4}$$

我们尝试流程 1 中的策略，可以得到一组可行解 C，在集合 C 里挑选出适应度最高的解作为问题三的近似最优解。

用流程 1 策略求解本题的过程中，设置$\lambda = 1, \gamma = 0.5$，每个种群里的个体数为 20 个，子代集合最大容量为 20 个，一共生成 250 代，对每条染色体的最大变异次数为 50 次，程序一次运行时间约 50min。从最优解看，被成功分配到登机口的飞机数为 235，比例为 77.6%，T 里的登机口都开启，S 里的登机口有 2 个未开启，图 3-1、3-2、3-3 依此表示窄体机线状图、宽体机线状图、登机口占用时间比率线状图。换乘失败的旅客的数量为 0，比率为 0 ，图 3-4、3-5 依此表示总体旅客换乘时间分布图、总体乘客换乘紧张度分布图。

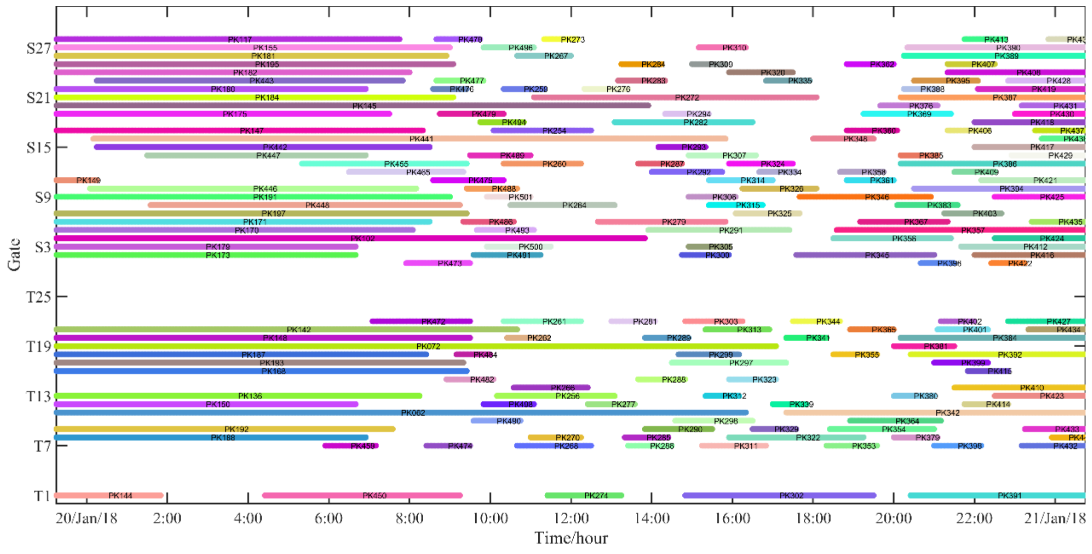


图 3-1：窄体机线状图

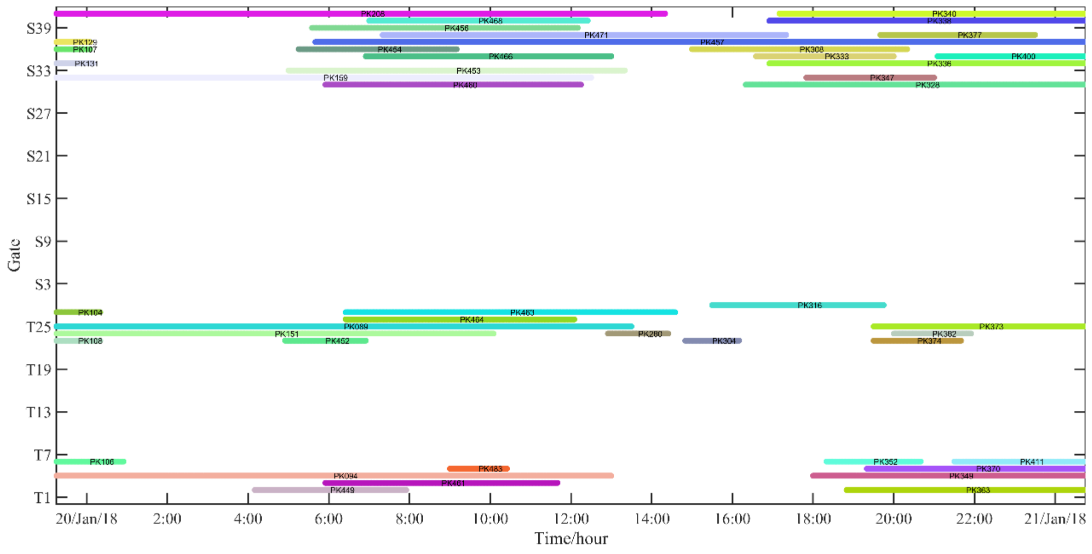


图 3-2：宽体机线状图

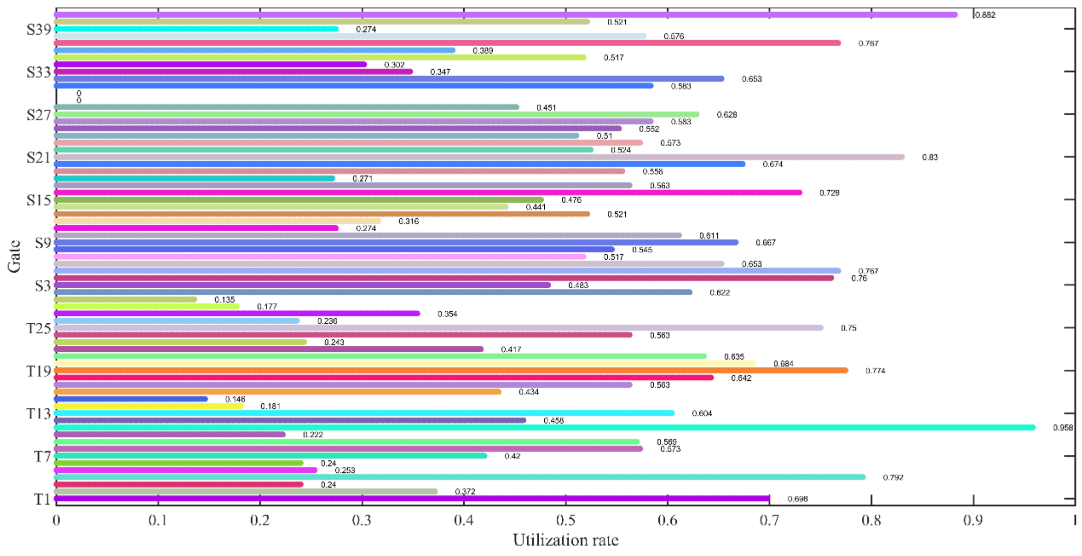


图 3-3：登机口占用时间比率线状图

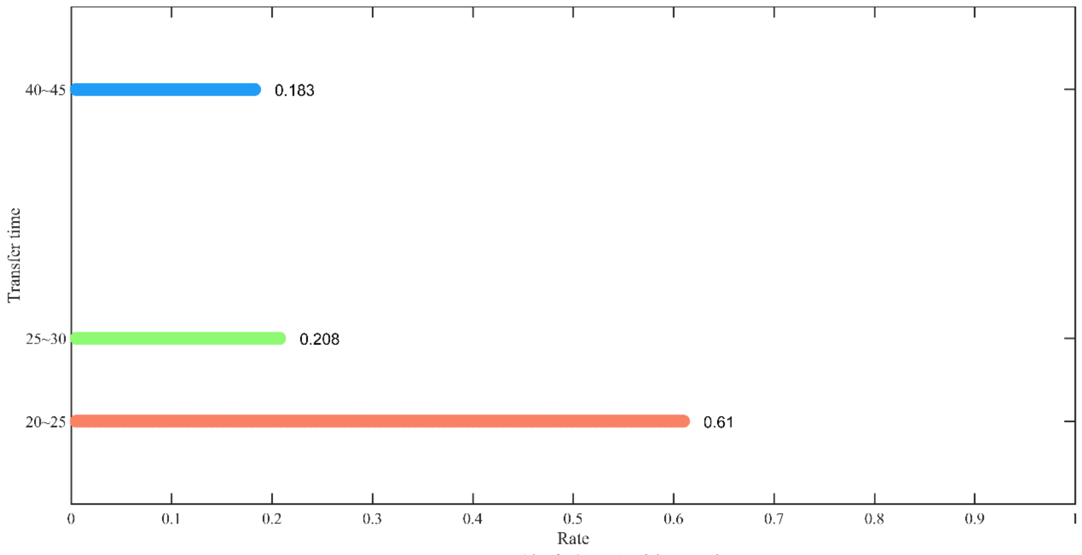


图 3-4：总体旅客换乘时间分布图

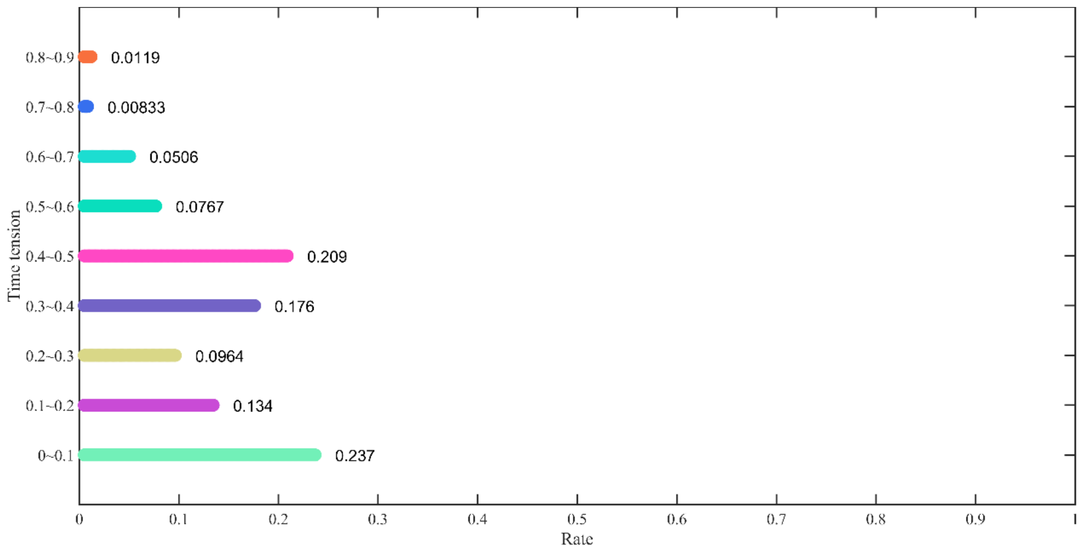


图 3-5：总体旅客换乘紧张度分布图

7. 模型的总结与评价

第一问我们建立了基于休眠航班数量最小和登机口开启数最小的整数规划模型，并将实际条件作为模型约束，客观合理抽象地描述了具体的航班—登机口分配问题。在求解方面，我们使用遗传算法，并结合领域知识，使问题得到简化，并大大缩小了可行解的搜索空间，极大地缩短了得到可行解的时间。

在第二问和第三问中，我们主要对目标函数进行修改，第二问加入了中转旅客最短流程时间一项指标，而第三问加入了换乘紧张度指标，两个模型的目标相似却各有侧重点。第二问是对实际情况下中转旅客办理业务手续的时间开销的估计，可以借此模型优化机场的业务流程，而第三问是把关注点更多地放在旅客身上，尽可能考虑旅客的实际情况。当紧张度大于 1 时，这样的调度安排显然是不合理的，因此第三问结果对实际情况下的航班调度和分配任务意义重大。

因为遗传算法本身的特性，针对一个优化模型，我们可以得到一系列的可行解，取出可行解集合中的最优个体，这个最优个体就是原问题的局部最优解。由于机场的航班调度情况复杂，一旦意外事件发生，可能会影响其他航班的正常起降，那么我们就可以从这个解的集合中再挑选一个满足当下情况的次优解，这就省却了重新计算的时间。尽管我们在算法中加入了许多随机性，但是由于初代种群的生成过程以及后面变异的过程遵循本文制定的规则，我们仍然很难从各个可行的方向搜寻可行解。另外由于种群里的个体数量并不多，算法迭代过程中很可能陷入一个局部的可行解空间，因此我们未来的研究可以往种群的多样性方面考虑。

8. 参考文献

[1] Bolat A. Models and a genetic algorithm for static aircraft-gate assignment problem[J]. Journal of the Operational Research Society, 2001, 52(10): 1107-1120.

[2] Cheng C H, Ho S C, Kwan C L. The use of meta-heuristics for airport gate assignment[J]. Expert systems with applications, 2012, 39(16): 12430-12437.

[3] Liu S, Chen W H, Liu J. Robust assignment of airport gates with operational safety constraints[J]. International Journal of Automation and Computing, 2016, 13(1): 31-41.

附录 Matlab 源程序

```
%%%%%%%%%%%%%%%%%%PROBLEM_SOLVER.m%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%PROBLEM_SOLVER.m%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%PROBLEM_SOLVER.m %%%%%%%%%%%%%%%%%%%%
% Annotation: the code generate a set of feasible solutions stored in
% a matrix x(num_of_samples,num_of_planes);
% num_of_samples is the number of solutions;
% num_of_planes is the number of planes;
% each element in x is the rank of the gate from 1~70,
% where 1~28 is the T gate, 29~69 is the S gate,
% and the 70th ghost gate is used to place the unscheduled planes.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%

close all;
clear all;
clc;
%% choose the problem you want to solve
% problem_choice=1, solve the problem 1;
% problem_choice=2, solve the problem 2;
% problem_choice=3, solve the problem 3;
problem_choice=1;
%% read data
% load data
[plane_struct gate_struct]=read_data_p1('InputData.txt', 'InputData.xlsx');
load gate_struct.mat;
load passen_struct.mat;
load plane_struct.mat;
% make the time in date=19 or 21 different with each other
[listdate19,~]=find(plane_struct.A<0);
plane_struct.A(listdate19)=0-listdate19*50;
[listdate21,~]=find(plane_struct.D>1440);
plane_struct.D(listdate21)=1440+listdate21*50;

% shift the time to be postive
shift_time=100-min(plane_struct.A);
plane_struct.A=plane_struct.A+shift_time;
plane_struct.D=plane_struct.D+shift_time;

%% input parameter
num_of_plane=size(plane_struct.A,1); % number of plane
```

```
num_of_gate=size(gate_struct.v,1); % number of gate
num_of_samples=20; % number of samples will be generated in each generation
lamda=0.8; % the weight parameter used in fitness(target) function
MAXGEN=2500; % the maximum generation that will be generated
NUM_PARENT=4; % the maximum number of parents will be chosed in each crossover
NUM_MUT=50; % the maximum attempt times in each mutation
MAX_ITER=floor(num_of_samples*10); % the maximum iteration times in each generation
myseed=sum(clock); % the seed for usage in random functions

%% program main
iter_times=0;
x=generate2(plane_struct,gate_struct,num_of_gate,num_of_plane,myseed,num_of_samples); %
generate the first x samples
for i_gen=1:MAXGEN
    save(['data', num2str(i_gen)], '*'); % save the data in each generation
    clear x_sons;
    x_sons=[]; % to store the new solutions
    iter_times=0;
    [x_sorted,
grade,~,~]=sort_by_target(x,plane_struct,passen_struct,gate_struct,num_of_gate,num_of_plane,la
mda,problem_choice); % sort the samples from good to bad
    x=unique(x,'rows'); % delete the same rows in x

    while(size(x_sons,1)<num_of_samples & iter_times<MAX_ITER) % iteration until
num_of_samples new solution are generated or the itertation times > MAX_ITER
        if(mod(iter_times,1)==0)
            disp([size(x_sons,1), iter_times, MAX_ITER, i_gen])
        end
        iter_times=iter_times+1;

        for i=1:NUM_PARENT
            % choose parent genes and generate new genes
            [id_male, id_female]=choose_parents(grade);

            % i==even, use method corssover to generate new solution
            % i==odd, use method crossover_replace to generate new solution
            if(mod(i,2)==0)
                [x1_son, x2_son]=crossover(x(id_male,:),
x(id_female,:),plane_struct,gate_struct,num_of_gate,num_of_plane);
            else
                [x1_son, x2_son]=crossover_replace(x(id_male,:),
x(id_female,:),plane_struct,gate_struct,num_of_gate,num_of_plane);
            end
            [~, okay]=check_1d(x1_son,plane_struct,gate_struct,num_of_gate,num_of_plane); %
check the x1_son is feasible or not
            if  (okay==1)   % remove duplicate solution and keep the new one
                x_tot=[x; x1_son];
                dimen1=size(x_tot,1);
                dimen2=size(unique(x_tot,'rows'),1);
                if dimen1==dimen2
                    x_sons=[x_sons; x1_son];
                end

            else
                [x_new,
success]=mutation(x1_son,NUM_MUT,plane_struct,gate_struct,num_of_gate,num_of_plane); %
x1_son is infeasible, x_new is feasible if success==1 by mutation
                if(success==1) % remove duplicate solution and keep the new one
                    x_tot=[x; x_new];
                    dimen1=size(x_tot,1);
                    dimen2=size(unique(x_tot,'rows'),1);
                    if dimen1==dimen2
                        x_sons=[x_sons; x_new];
                    end

                end
            end

            [~, okay]=check_1d(x2_son,plane_struct,gate_struct,num_of_gate,num_of_plane);
            if  (okay==1) % remove duplicate solution and keep the new one
                x_tot=[x; x2_son];
                dimen1=size(x_tot,1);
                dimen2=size(unique(x_tot,'rows'),1);
                if dimen1==dimen2
                    x_sons=[x_sons; x2_son];
                end

            else
                [x_new,
success]=mutation(x2_son,NUM_MUT,plane_struct,gate_struct,num_of_gate,num_of_plane); %
x2_son is infeasible, x_new is feasible if success==1 by mutation
                if(success==1) % remove duplicate solution and keep the new one
                    x_tot=[x; x_new];
                    dimen1=size(x_tot,1);
                    dimen2=size(unique(x_tot,'rows'),1);
                    if dimen1==dimen2
                        x_sons=[x_sons; x_new];
                    end

                end
            end
        end


        % mutation the each gene ranked by iter_times in x by reducing the number of ghost
gate directly
        if(iter_times<size(x,1))
            % mutation the iter_times ranked gene in x
            x_old=x(iter_times,:);
            if(randi([1,2])==1)
                [x_new,
success]=mutation_ghost(x_old,NUM_MUT,plane_struct,gate_struct,num_of_gate,num_of_plane);
            else
                [x_new,
success]=mutation_ghost_random(x_old,NUM_MUT,plane_struct,gate_struct,num_of_gate,num_
of_plane);
            end

            if(success==1) % remove duplicate solution and keep the new one
                x_tot=[x; x_new];
                dimen1=size(x_tot,1);
                dimen2=size(unique(x_tot,'rows'),1);
                if dimen1==dimen2
                    x_sons=[x_sons; x_new];
                end
            end
        end



        % generate random x by order of random list of plane

[x_new]=generate2_by_random(plane_struct,gate_struct,num_of_gate,num_of_plane,myseed,1);
        if(mod(iter_times,2)==0)
            % mutation the feasible sample.
            [x_new,
success]=mutation_good(x_new,NUM_MUT,plane_struct,gate_struct,num_of_gate,num_of_plane)
;
        end
        x_tot=[x; x_new]; % remove duplicate solution and keep the new one
        dimen1=size(x_tot,1);
```

```
            dimen2=size(unique(x_tot,'rows'),1);
            if dimen1==dimen2
                  x_sons=[x_sons; x_new];
            end
            x_sons=unique(x_sons,'row');
      end

      x_tot=[x; x_sons];
      [x_sorted,
grade,~,~]=sort_by_target(x_tot,plane_struct,passen_struct,gate_struct,num_of_gate,num_of_pla
ne,lamda,problem_choice); % sort the x from good to bad
      grade=(grade-min(grade))/(max(grade)-min(grade));

      % choose the new result by random delete the worse x
      while(size(x_sorted,1)>num_of_samples)
            grade_rand=rand();
            i_rand=randi([1,size(x_sorted,1)]);
            if(grade_rand>grade(i_rand))
                  x_sorted(i_rand,:)=[];
            end
      end
      % update the x
      x=x_sorted(1:num_of_samples,:);
end



%% choose parents samples
% INPUT: grade is the fitness of the solution set x
% OUTPUT: id_male and id_female are the ranks of parents
function [id_male, id_female]=choose_parents(grade)
n=length(grade);
prob=grade/sum(grade);
weight(1)=prob(1);
num_id=0;
id_male=0;
id_female=0;
for i=2:n
      weight(i)=weight(i-1)+prob(i);
end
value=rand();
for i=1:n
      value=value-weight(i);
      if(value<0)
```

```
            id_male=i;
            id_female=id_male;
            break;
        end

end
while(id_female==id_male)
        value=rand();
        for i=1:n
            value=value-weight(i);
            if(value<0)
                id_female=i;
                break;
            end
        end

end
end

%% generate random solution set x (may infeasible)
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_plane is number of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: myseed the seed for rand function
% INPUT: num_of_samples is the number of generated solutions
% OUTPUT: x is the generated solutions, x maybe not feasible
function
[x]=generate(plane_struct,gate_struct,num_of_gate,num_of_plane,myseed,num_of_samples)
x_temp=zeros(num_of_samples,1);
x=zeros(num_of_samples,num_of_plane);    %generate n*m samples
for i=1:num_of_plane
        A=find(gate_struct.v==plane_struct.u(i));    % find the list id of gate-plane W/N type are the
same
        B=find(gate_struct.T_type*plane_struct.A_type(i,:)'==1); % find the list id of gate-plane I/D
type are the same

        C=intersect(A,B); % choose the shared elements in A & B
        C=[C; num_of_gate+1];

        rand('seed',sum(100*clock)+myseed*i);
        x_temp=C(randi([1,length(C)],num_of_samples,1));

        x(:,i)=x_temp;
```

```
end

end

%% generate random solution set x (feasible)
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_plane is number of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: myseed the seed for rand function
% INPUT: num_of_samples is the number of generated solutions
% OUTPUT: x is the generated solutions, x are all feasible
function
[x]=generate2(plane_struct,gate_struct,num_of_gate,num_of_plane,myseed,num_of_samples)
x_temp=zeros(num_of_samples,1);
x=zeros(num_of_samples,num_of_plane);   %generate n*m samples
for i=1:num_of_plane
    i_in_generate2=i
    A=find(gate_struct.v==plane_struct.u(i));   % find the list id of gate-plane W/N type are the
same
    B=find(gate_struct.T_type*plane_struct.A_type(i,:)'==1); % find the list id of gate-plane I/D
type are the same

    C=intersect(A,B); % choose the shared elements in A & B
    C=[C; num_of_gate+1];
    C_2d=repmat(C,1,num_of_samples);
    for j=1:i-1
        is_j_in_c=ismember(x(:,j),C);                                    % air[j].gate.id is a member of
air[i].gate.id==C
        for k=1:length(is_j_in_c)

            if(is_j_in_c(k)==1 && x(k,j)~=num_of_gate+1)    % if for kth choice for j has the
same gate in C, delete this gate in C for this k column
                is_conflict=min([...
                    abs(plane_struct.D(i)-plane_struct.D(j)), ...
                    abs(plane_struct.D(i)-plane_struct.A(j)), ...
                    abs(plane_struct.A(i)-plane_struct.D(j)), ...
                    abs(plane_struct.A(i)-plane_struct.A(j)) ...
                    ])<45;    % less than 45 means conflict
                is_conflict=max(is_conflict, ...
                    (plane_struct.D(j)-plane_struct.A(i))* ...
                    (plane_struct.D(i)-plane_struct.A(j))>0);

                if(is_conflict==1)
```

```
                    C_2d(find(C_2d(:,k)==x(k,j)),k)=-1; % make the same gate as the ghost gate
id
                end

            end
        end

    end
    rand('seed',sum(100*clock)+myseed*i);
    for k=1:num_of_samples
        C=C_2d(:,k);
        C(find(C==-1))=[];
        x_temp(k)=C(randi([1,length(C)],1));
    end

    x(:,i)=x_temp;
end

end



%% generate random x by random list of plane
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_plane is number of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: myseed the seed for rand function
% INPUT: num_of_samples is the number of generated solutions
% OUTPUT: x is the generated solutions, x are all feasible
function
[x]=generate2_by_random(plane_struct,gate_struct,num_of_gate,num_of_plane,myseed,num_of_
samples)
x_temp=zeros(num_of_samples,1);
x=zeros(num_of_samples,num_of_plane);   %generate n*m samples

list_plane=1:num_of_plane;
list_plane=list_plane(randperm(numel(list_plane)));

for id1=1:num_of_plane
    i=list_plane(id1);   % choose the i by random list
    i_in_generate2=i;
    A=find(gate_struct.v==plane_struct.u(i));   % find the list id of gate-plane W/N type are the
same
    B=find(gate_struct.T_type*plane_struct.A_type(i,:)'==1); % find the list id of gate-plane I/D
```

```
type are the same

    C=intersect(A,B); % choose the shared elements in A & B
    C=[C; num_of_gate+1];
    C_2d=repmat(C,1,num_of_samples);
    for jd1=1:id1-1
        j=list_plane(jd1);   % choose the j by random list
        is_j_in_c=ismember(x(:,j),C);                          % air[j].gate.id is a member of
air[i].gate.id==C
        for k=1:length(is_j_in_c)

            if(is_j_in_c(k)==1 && x(k,j)~=num_of_gate+1)   % if for kth choice for j has the
same gate in C, delete this gate in C for this k column
                is_conflict=min([...
                    abs(plane_struct.D(i)-plane_struct.D(j)), ...
                    abs(plane_struct.D(i)-plane_struct.A(j)), ...
                    abs(plane_struct.A(i)-plane_struct.D(j)), ...
                    abs(plane_struct.A(i)-plane_struct.A(j)) ...
                    ])<45;   % less than 45 means conflict
                is_conflict=max(is_conflict, ...
                    (plane_struct.D(j)-plane_struct.A(i))* ...
                    (plane_struct.D(i)-plane_struct.A(j))>0);

                if(is_conflict==1)
                    C_2d(find(C_2d(:,k)==x(k,j)),k)=-1; % make the same gate as the ghost gate
id
                end

            end
        end

    end
    rand('seed',sum(100*clock)+myseed*i);
    for k=1:num_of_samples
        C=C_2d(:,k);
        C(find(C==-1))=[];
        x_temp(k)=C(randi([1,length(C)],1));
    end

    x(:,i)=x_temp;
end

end
```

```
%% generate feasible gates set for index'th element :compare 1:index-1 and index+1:end
sections in sub_x
% INPUT: sub_x is the sub solution that need to be improved
% INPUT: index is the id of bug gene in sub_x
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: C is the set of feasible gates
function
[C]=generate_feasible_gates(sub_x,index,plane_struct,gate_struct,num_of_gate,num_of_plane)
sub_x_length=length(sub_x);
A=find(gate_struct.v==plane_struct.u(index));    % find the list id of gate-plane W/N type are the
same
B=find(gate_struct.T_type*plane_struct.A_type(index,:)'==1); % find the list id of gate-plane I/D
type are the same

C=intersect(A,B); % choose the shared elements in A & B
for i=1:sub_x_length
    if(i==index)
        continue;
    end
    is_i_in_c=ismember(sub_x(i),C);
    if(is_i_in_c==1)

        is_conflict=min([...
            abs(plane_struct.D(i)-plane_struct.D(index)), ...
            abs(plane_struct.D(i)-plane_struct.A(index)), ...
            abs(plane_struct.A(i)-plane_struct.D(index)), ...
            abs(plane_struct.A(i)-plane_struct.A(index)) ...
            ])<45;    % less than 45 means conflict
        is_conflict=max(is_conflict, ...
            (plane_struct.D(index)-plane_struct.A(i))* ...
            (plane_struct.D(i)-plane_struct.A(index))>0);

        if(is_conflict==1)
            C(find(C==sub_x(i)))=[];
        end

    end
end
C=[C; num_of_gate+1];
end
```

```
%% check time information for the x solution, return the feasible set of solutions of x
% INPUT: x is the solutions set that need to be checked
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x_okay is the feasible solutions set
function [x_okay]=check(x,plane_struct,gate_struct,num_of_gate,num_of_plane)
[m,n]=size(x);
x_okay=[];

if isempty(find(x<1|x>num_of_gate+1))~=1
    pause;
end

for ii=1:m   % each sample
    x_temp=x(ii,:);
    judge=1;
    for i=1:n
        for j=i+1:n
            if(x_temp(i)==x_temp(j) && x_temp(i)<num_of_gate+1)
                judge=min([...
                    abs(plane_struct.D(i)-plane_struct.D(j)), ...
                    abs(plane_struct.D(i)-plane_struct.A(j)), ...
                    abs(plane_struct.A(i)-plane_struct.D(j)), ...
                    abs(plane_struct.A(i)-plane_struct.A(j)) ...
                    ])>=45;
                judge=min(judge, ...
                    (plane_struct.D(j)-plane_struct.A(i))* ...
                    (plane_struct.D(i)-plane_struct.A(j))<=0);
            end
            if(judge==0)
                break;
            end
        end
        if(judge==0)
            break;
            %                     disp("x warning, find x time constraint not satisfied !!!!!");
        end
    end
    if(judge==1)
        x_okay=[x_okay; x_temp];
    end
end
```

```
end

%% check time information for the x_1d solution
% INPUT: x_1d is the solution that need to be checked
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x1d_okay is the feasible solutions
% OUTPUT: okay is the check result
function [x1d_okay, okay]=check_1d(x_1d,plane_struct,gate_struct,num_of_gate,num_of_plane)
n=length(x_1d);
x1d_okay=[];
okay=0;

if isempty(find(x_1d<1|x_1d>num_of_gate+1))~=1
    pause;
end
x_temp=x_1d;
judge=1;
for i=1:n
    for j=i+1:n
        if(x_temp(i)==x_temp(j) && x_temp(i)<num_of_gate+1)
            judge=min([...
                abs(plane_struct.D(i)-plane_struct.D(j)), ...
                abs(plane_struct.D(i)-plane_struct.A(j)), ...
                abs(plane_struct.A(i)-plane_struct.D(j)), ...
                abs(plane_struct.A(i)-plane_struct.A(j)) ...
                ])>=45;
            judge=min(judge, ...
                (plane_struct.D(j)-plane_struct.A(i))* ...
                (plane_struct.D(i)-plane_struct.A(j))<=0);
        end
        if(judge==0)
            break;
        end
    end
    if(judge==0)
        break;
    end
end
if(judge==1)
    x1d_okay=x_temp;
    okay=1;
```

```
end
end

%% check time information for the x_1d solution
% INPUT: x_1d is the solution that need to be checked
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x1d_okay is the feasible solutions
% OUTPUT: okay is the check result
function [x1d_okay,
okay]=check_1d_last(x_1d,plane_struct,gate_struct,num_of_gate,num_of_plane)
n=length(x_1d);
x1d_okay=[];
okay=0;

if isempty(find(x_1d<1|x_1d>num_of_gate+1))~=1
    pause;
end
x_temp=x_1d;
judge=1;
j=n;
for i=1:n-1
    if(x_temp(i)==x_temp(j) && x_temp(i)<num_of_gate+1)
        judge=min([...
            abs(plane_struct.D(i)-plane_struct.D(j)), ...
            abs(plane_struct.D(i)-plane_struct.A(j)), ...
            abs(plane_struct.A(i)-plane_struct.D(j)), ...
            abs(plane_struct.A(i)-plane_struct.A(j)) ...
            ])>=45;
        judge=min(judge, ...
            (plane_struct.D(j)-plane_struct.A(i))* ...
            (plane_struct.D(i)-plane_struct.A(j))<=0);
    end
    if(judge==0)
        return;
    end

end
if(judge==1)
    x1d_okay=x_temp;
    okay=1;
end
```

```
end

%% check time information for the x_1d solution
% INPUT: x_1d is the solution that need to be checked
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x1d_okay is the feasible solutions
% OUTPUT: okay is the check result
function [x1d_okay,
okay]=check_1d_first(x_1d,plane_struct,gate_struct,num_of_gate,num_of_plane)
n=length(x_1d);
x1d_okay=[];
okay=0;

if isempty(find(x_1d<1|x_1d>num_of_gate+1))~=1
    disp("x_1d error, find x_1d<0 or x_1d>num_of_gate+1 !!!!!!");
    pause;
end
x_temp=x_1d;
judge=1;
j=1;
for i=2:n
    %     for j=i+1:n
    if(x_temp(i)==x_temp(j) && x_temp(i)<num_of_gate+1)
        judge=min([...
            abs(plane_struct.D(i)-plane_struct.D(j)), ...
            abs(plane_struct.D(i)-plane_struct.A(j)), ...
            abs(plane_struct.A(i)-plane_struct.D(j)), ...
            abs(plane_struct.A(i)-plane_struct.A(j)) ...
            ])>=45;
        judge=min(judge, ...
            (plane_struct.D(j)-plane_struct.A(i))* ...
            (plane_struct.D(i)-plane_struct.A(j))<=0);
    end
    if(judge==0)
        return;
    end
end
if(judge==1)
    x1d_okay=x_temp;
    okay=1;
end
```

```
end

%% sort solution x by target
% INPUT: x_1d is the solution that need to be checked
% INPUT: plane_struct stores the all information of planes
% INPUT: passen_struct stores the all information of passengers
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% INPUT: lamda is the weight parameter in fitness fuunctions
% INPUT: problem_choice is the choice of problem need to be solved
% OUTPUT: x_sorted the sorted solutions set
% OUTPUT: grade is the value of fitness, larger is better
% OUTPUT: num_used_planes is the number of used planes
function [x_sorted,
grade,num_open_gates,num_used_planes]=sort_by_target(x,plane_struct,passen_struct,gate_stru
ct,num_of_gate,num_of_plane,lamda,problem_choice)
[m,n]=size(x);   % m is the num of samples, n is the num of planes
grade=zeros(m,1); % to store the values of target functions

% calculate the grade for each sample
for i=1:m
    num_open_gates=numel(unique(x(i,:)));   % calculate the num of open gates
    if(find(x(i,:)==num_of_gate+1)) % it has the ghost gate used
        num_open_gates=num_open_gates-1;
    end

    % default solve the problem 1
    num_used_planes=num_of_plane-numel(find(x(i,:)==num_of_gate+1));   % calculate the
num of used planes
    grade(i,1)=lamda*(1-num_open_gates/num_of_gate)+(1-
lamda)*num_used_planes/num_of_plane; % the grade less is better

    %solve the problem 2
    if(problem_choice==2)
    [time tensity time_of_passen each_of_passen
time_gap_plane]=calculate_passenger_time(x(i,:),plane_struct,passen_struct,gate_struct,num_of_
gate,num_of_plane, problem_choice);
    grade(i,1)=grade(i,1)*0.2+0.8*(1-time/45);
    end

    %solve the problem 3
    if(problem_choice==3)
    [time tensity time_of_passen each_of_passen
time_gap_plane]=calculate_passenger_time(x(i,:),plane_struct,passen_struct,gate_struct,num_of_
gate,num_of_plane, problem_choice);
    grade(i,1)=grade(i,1)*0.2+0.8*(1-tensity);
    end

end


% sort the x from good to bad by grade, return the sorted x_sorted
[grade index]=sort(grade,'descend');
x_sorted=x(index,:);

% print the best result
num_open_gates=numel(unique(x_sorted(1,:)));   % calculate the num of open gates
if(find(x_sorted(1,:)==num_of_gate+1)) % it has the ghost gate used
    num_open_gates=num_open_gates-1;
end
num_used_planes=num_of_plane-numel(find(x_sorted(1,:)==num_of_gate+1));   % calculate the
num of used planes
disp("num_open_gates num_used_planes=");
disp([num_open_gates, num_used_planes]);
end

%% intersect two family samples x1_old+x2_old=x1_new+x2_new
% INPUT: x1_old & x2_old are parents
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTpUT: x1_new & x2_new are the sons

% old 1: 111111111111111111111111111111111111111111111111
% old 2: 222222222222222222222222222222222222222222222222
% cut_point:                                  ^
% new 1: 111111111111111111111111111111111122222222222222
% new 2: 222222222222222222222222222222222211111111111111
function [x1_new, x2_new]=crossover(x1_old,
x2_old,plane_struct,gate_struct,num_of_gate,num_of_plane)
m=length(x1_old);
cut_point=randi([1,m-1]);
x1_new=[x1_old(1:cut_point) x2_old(cut_point+1:end)];
x2_new=[x2_old(1:cut_point) x1_old(cut_point+1:end)];
end
```

```
%% replace one section between two family samples x1_old+x2_old=x1_new+x2_new
% INPUT: x1_old & x2_old are parents
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTpUT: x1_new & x2_new are the sons

% old 1: 11111111111111111111111111111111111111111111111111
% old 2: 22222222222222222222222222222222222222222222222222
% cut_point:             ^              ^
% new 1: 11111111111111111112222222222222222211111111111111
% new 2: 22222222222222222221111111111111111122222222222222
function [x1_new, x2_new]=crossover_replace(x1_old,
x2_old,plane_struct,gate_struct,num_of_gate,num_of_plane)
m=length(x1_old);
cut_point1=randi([2,m-2]);
cut_point2=randi([cut_point1+1, m-1]);
x1_new=[x1_old(1:cut_point1) x2_old(cut_point1+1:cut_point2) x1_old(cut_point2+1:end)];
x2_new=[x2_old(1:cut_point1) x1_old(cut_point1+1:cut_point2) x2_old(cut_point2+1:end)];
end

%% genovariation a infeasible gene into a feasible gene, try most k times
% INPUT: x_old is parent
% INPUT: num_of_mut is the max number of mutation times
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x_new is the son
% OUTPUT: success means the mutation success or not

% old: 1111111111111111111111E11111111111111111E11111111111
% bug:                     ^               ^
% new: 1111111111111111111111111111111111111111111111111111
% input x_old is 1D, num_of_var is the maximum times of the genovariation
function [x_new,
success]=mutation(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_plane) %
x_old is infeasible, x_new is feasible if success==1
irank=0;
success=0;
% [~, okay]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);
n=length(x_old);
for i=1:n
```

```
        [~, okay_last]=check_1d_last(x_old(1:i),plane_struct,gate_struct,num_of_gate,num_of_plane);
        [~,
okay_first]=check_1d_first(x_old(i:end),plane_struct,gate_struct,num_of_gate,num_of_plane);

        if((okay_last+okay_first)~=2 & irank<num_of_mut) % mean the ith plane is conflict with the
previous section

                C=generate_feasible_gates(x_old,i,plane_struct,gate_struct,num_of_gate,num_of_plane);
                C=[C; num_of_gate+1];


                rand('seed',sum(100*clock)*i);
                x_old(i)=C(randi([1,length(C)],1));
                irank=irank+1;
        end
end

if(randi([1,2])==1)
        [x_old,
~]=mutation_ghost(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_plane);
else
        [x_old,
~]=mutation_ghost_random(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_pl
ane);
end

[x_new, success]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);

num_70=numel(find(x_old==num_of_gate+1));
if(num_70>80)
        success=0;
end
end

%% genovariation a infeasible gene into a feasible gene, try most k times
% INPUT: x_old is parent
% INPUT: num_of_mut is the max number of mutation times
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x_new is the son
% OUTPUT: success means the mutation success or not
% old: 11111111111111111111E1111111111111111E11111111111
% bug:                                        ^                              ^
% new: 1111111111111111111111111111111111111111111111111111
% input x_old is 1D, num_of_var is the maximum times of the genovariation
function [x_new,
success]=mutation_good(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_plane
) % x_old is feasible, x_new is feasible if success==1
irank=0;
success=0;
% [~, okay]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);
n=length(x_old);
rand_num=randi([1,n],1);
rand_list=randi([1,n],1,rand_num);
rand_list=unique(rand_list);
for i=rand_list
    [~, okay_last]=check_1d_last(x_old(1:i),plane_struct,gate_struct,num_of_gate,num_of_plane);
    [~,
okay_first]=check_1d_first(x_old(i:end),plane_struct,gate_struct,num_of_gate,num_of_plane);

    if((okay_last+okay_first)==2 & irank<num_of_mut) % mean the ith plane is conflict with the
previous section

        C=generate_feasible_gates(x_old,i,plane_struct,gate_struct,num_of_gate,num_of_plane);
        C=[C; num_of_gate+1];

        rand('seed',sum(100*clock)*i);
        x_old(i)=C(randi([1,length(C)],1));
        irank=irank+1;
    end
end

if(randi([1,2])==1)
    [x_old,
~]=mutation_ghost(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_plane);
else
    [x_old,
~]=mutation_ghost_random(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_pl
ane);
end

[x_new, success]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);

num_70=numel(find(x_old==num_of_gate+1));
if(num_70>80)
```

```
    success=0;
end
end



%% genovariation a infeasible gene into a feasible gene, try most k times
% INPUT: x_old is parent
% INPUT: num_of_mut is the max number of mutation times
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x_new is the son
% OUTPUT: success means the mutation success or not

% old: 1111111111111111111170111111111111111170E1111111111
% bug:                      ^                ^
% new: 1111111111111111111111111111111111111111111111111
% input x_old is 1D, num_of_var is the maximum times of the genovariation
function [x_new,
success]=mutation_ghost(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num_of_plan
e) % mutation the plane for that in ghost gates
irank=0;
success=0;
% [~, okay]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);
n=length(x_old);

for i=1:n

    if(x_old(i)==(num_of_gate+1) & irank<num_of_mut) % mean the ith plane is conflict with
the previous section
        C=generate_feasible_gates(x_old,i,plane_struct,gate_struct,num_of_gate,num_of_plane);

        if(isempty(C)==1)
            irank=num_of_mut; % if C=[]; break out
            return;
        else
            rand('seed',sum(100*clock)*i);
            x_old(i)=C(randi([1,length(C)],1));
            irank=irank+1;
        end
    end
end
```

```
[x_new, success]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);
end



%% genovariation a infeasible gene into a feasible gene, try most k times
% INPUT: x_old is parent
% INPUT: num_of_mut is the max number of mutation times
% INPUT: plane_struct stores the all information of planes
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% OUTPUT: x_new is the son
% OUTPUT: success means the mutation success or not

% old: 11111111111111111111170111111111111111170E11111111111
% bug:                                  ^                            ^
% new: 11111111111111111111111111111111111111111111111111111
% input x_old is 1D, num_of_var is the maximum times of the genovariation
function [x_new,
success]=mutation_ghost_random(x_old,num_of_mut,plane_struct,gate_struct,num_of_gate,num
_of_plane) % mutation the plane for that in ghost gates by random
irank=0;
success=0;
n=length(x_old);

list=find(x_old==num_of_gate+1); % find the list of x(i)=70
% for i=1:n
for i=list

    if(x_old(i)==(num_of_gate+1) & irank<num_of_mut) % mean the ith plane is conflict with
the previous section
        C=generate_feasible_gates(x_old,i,plane_struct,gate_struct,num_of_gate,num_of_plane);

        if(isempty(C)==1)
            irank=num_of_mut; % if C=[]; break out
            return;
        else
            rand('seed',sum(100*clock)*i);
            x_old(i)=C(randi([1,length(C)],1));
            %                  [~,
okay]=check_1d(x_old(1:i),plane_struct,gate_struct,num_of_gate,num_of_plane);
            irank=irank+1;
            end
        end
    end

    [x_new, success]=check_1d(x_old,plane_struct,gate_struct,num_of_gate,num_of_plane);
end


%%%%%%%%%%%%%%%%%%%read_data_p1.m%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%read_data_p1.m%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%read_data_p1.m %%%%%%%%%%%%%%%%%%%%%%%
function [plane_struct gate_struct]=read_data_p1(filename_txt, filename_xlsx)
% INPUT: filename_txt is the name of txt file
% INPUT: filename_xlsx is the name of xlsx file
% OUTPUT: plane_struct stores all informations of planes
% OUTPUT: gate_struct stores all informations of gates
date1=datetime('18/1/19');
date2=datetime('18/1/20');
date3=datetime('18/1/21');
%% read the Pcuks information
% [pucks{1:12}]=textread('InputData.txt','%q %q %q %q %q %q %q %q %q %q %q %q');
[pucks{1:12}]=textread(filename_txt,'%q %q %q %q %q %q %q %q %q %q %q %q');
% load pucks.mat;
% [~,Pucks]=xlsread('InputData.xlsx','Pucks');
[~,Pucks]=xlsread(filename_xlsx,'Pucks');
head=Pucks(1,:);
arrive_date=datetime(Pucks(2:end,2));
depart_date=datetime(Pucks(2:end,7));
arrive_time=string(pucks{1,3});
depart_time=string(pucks{1,8});
num_of_planes=size(arrive_time,1);

puck_id=string(Pucks(2:end,1));
flight_id_arrive=string(Pucks(2:end,4));
flight_id_depart=string(Pucks(2:end,9));

for i=1:num_of_planes
    A=strsplit(arrive_time(i,1),':');
    B=strsplit(depart_time(i,1),':');
    hour1(i)=double(A(1,1));
    mint1(i)=double(A(1,2));
    hour2(i)=double(B(1,1));
    mint2(i)=double(B(1,2));
end
```

```
arrive_time=(hour1*60+mint1)';
depart_time=(hour2*60+mint2)';

[list1,~]=find((arrive_date==date1 & depart_date==date2));
[list2,~]=find((arrive_date==date2 & depart_date==date2));
[list3,~]=find((arrive_date==date2 & depart_date==date3));
% for list1 2 & 3
Atime1=ones(size(list1))*-45;
Dtime1=depart_time(list1);
Atime2=arrive_time(list2);
Dtime2=depart_time(list2);
Atime3=arrive_time(list3);
Dtime3=ones(size(list1))*(24*60+45);

Atime=[Atime1; Atime2; Atime3];
Dtime=[Dtime1; Dtime2; Dtime3];

% size of the plane narrow or wide, wide==1
size_of_plane=string(pucks{1,6});
wide_ref=string({"332" "333" "33E" "33H" "33L" "773"})';
narrow_ref=string({"319" "320" "321" "323" "325" "738" "73A" "73E" "73H" "73L"})';
size_of_plane=ismember(size_of_plane,wide_ref);
size_of_plane=[size_of_plane(list1); size_of_plane(list2); size_of_plane(list3)];

% arrive and depart type of the plane
arrive_type=string(pucks{1,5});
depart_type=string(pucks{1,10});
ad_type=strcat(arrive_type,depart_type); % II ID DI DD
AD_type=zeros(num_of_planes,4);
AD_type(:,1)=ad_type(:,1)=="II";
AD_type(:,2)=ad_type(:,1)=="ID";
AD_type(:,3)=ad_type(:,1)=="DI";
AD_type(:,4)=ad_type(:,1)=="DD";
AD_type=[AD_type(list1,:); AD_type(list2,:); AD_type(list3,:)];

% the id of the flight
puck_id=[puck_id(list1); puck_id(list2); puck_id(list3)];
flight_id_arrive=[flight_id_arrive(list1); flight_id_arrive(list2); flight_id_arrive(list3)];
flight_id_depart=[flight_id_depart(list1); flight_id_depart(list2); flight_id_depart(list3)];
% sort the data by arrive time
% [~,list]=sort(Atime);
% AD_type(:,:)=AD_type(list,:);
% size_of_plane=size_of_plane(list);
% Atime=Atime(list);
% Dtime=Dtime(list);

% create struct for plane
plane_struct=struct(...
      'A_type',double(AD_type), ...
      'u',double(size_of_plane), ...
      'A',double(Atime), ...
      'D',double(Dtime),...
      'flight_id_arrive',flight_id_arrive, ...
      'flight_id_depart',flight_id_depart, ...
      'puck_id', puck_id);

%% read the Passenger information
[~,Tickets]=xlsread(filename_xlsx,'Tickets');
head=Tickets(1,:);
num_of_tick=size(Tickets,1)-1;
num_of_pasen=load('num_of_pasen.txt');
pasen_id=string(Tickets(2:end,1));
f_id_arrive=string(Tickets(2:end,3));
f_id_depart=string(Tickets(2:end,5));

arrive_date=datetime(Tickets(2:end,4));
depart_date=datetime(Tickets(2:end,6));
% only consider the passengers arrive at date1
date1=datetime('18/1/20');
[list,~]=find((arrive_date==date1 & depart_date==date1));



passen_struct=struct(...
      'pasen_id', pasen_id(list), ...
      'pasen_num', num_of_pasen(list), ...
      'f_id_arrive', f_id_arrive(list), ...
      'f_id_depart', f_id_depart(list));

%% read the Gates information
% [~,Gates]=xlsread('InputData.xlsx','Gates');
[~,Gates]=xlsread(filename_xlsx,'Gates');
head=Gates(1,:);
num_of_gates=size(Gates,1)-1;
% size of the gate narrow or wide, wide==1
size_of_gate=string(Gates(2:end,6));
size_of_gate=ismember(size_of_gate,"W");
```

```
% arrive and depart type of the plane
arrive_type_gate=string(Gates(2:end,4));
depart_type_gate=string(Gates(2:end,5));
ad_type_gate=strcat(arrive_type_gate,depart_type_gate);
AD_type_gate=zeros(num_of_gates,4);
AD_type_gate(:,1)=ad_type_gate(:,1)=="II";
AD_type_gate(:,2)=ad_type_gate(:,1)=="ID";
AD_type_gate(:,3)=ad_type_gate(:,1)=="DI";
AD_type_gate(:,4)=ad_type_gate(:,1)=="DD";
for i=1:num_of_gates
    if(ad_type_gate(i)=="D, ID")
        AD_type_gate(i,:)=[0 1 0 1];
    else if(ad_type_gate(i)=="D, ID, I")
            AD_type_gate(i,:)=[1 1 1 1];
        else if(ad_type_gate(i)=="D, II")
                AD_type_gate(i,:)=[1 0 1 0];
            else if(ad_type_gate(i)=="ID, I")
                    AD_type_gate(i,:)=[1 1 0 0];
                else if(ad_type_gate(i)=="DD, I")
                        AD_type_gate(i,:)=[0 0 1 1];
                    end
                end
            end
        end
    end
end

% add gate type for TN TC TS SD SC SS SE
TS=string(Gates(2:end,2));
NS=string(Gates(2:end,3));
NS(NS=="North")="N";
NS(NS=="Center")="C";
NS(NS=="South")="S";
NS(NS=="East")="E";

TSNS=strcat(TS,NS);

% add the id of gate
gate_id=string(Gates(2:end,1));

% create struct for gates
gate_struct=struct(...
    'T_type',double(AD_type_gate), ...
    'v',double(size_of_gate), ...
    'TS', TS, ...
    'NS', NS, ...
    'TSNS', TSNS, ...
    'gate_id', gate_id);

%% save data
save('plane_struct.mat', 'plane_struct');
save('passen_struct.mat', 'passen_struct');
save('gate_struct.mat', 'gate_struct');
end




%%%%%%%%%%%%%%%%%calculate_passenger_time.m%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% %%%%%calculate_passenger_time.m%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%calculate_passenger_time.m %%%%%%%%%%%%%%%%%%%
function [time tensity time_of_passen each_of_passen
time_gap_plane]=calculate_passenger_time(x1d,plane_struct,passen_struct,gate_struct,num_of_g
ate,num_of_plane, num_of_p)
% INPUT: x_old is the solution
% INPUT: num_of_mut is the max number of mutation times
% INPUT: plane_struct stores the all information of planes
% INPUT: passen_struct stores the all information of passengers
% INPUT: gate_struct stores the all information of gates
% INPUT: num_of_gate is number of gates
% INPUT: num_of_plane is number of planes
% INPUT: num_of_p is the number of problem, == 2 or 3
% OUTPUT: time is the mean time for passenger transfering
% OUTPUT: tensity is the time tensity for passenger transfering
% OUTPUT: time_of_passen is the time of each passenger
% OUTPUT: each_of_passen is the number of passengers in each passenger rank
% OUTPUT: time_gap_plane is the gap time between two flights for each passenger rank

num_of_passen=size(passen_struct.pasen_id,1);
type=[...
    "DTDT", "DTDS", "DTIT", "DTIS", ...
    "DSDT", "DSDS", "DSIT", "DSIS", ...
    "ITDT", "ITDS", "ITIT", "ITIS", ...
    "ISDT", "ISDS", "ISIT", "ISIS"];
time1=[ ...
    15, 20, 35, 40, ...
    20, 15, 40, 35, ...
    35, 40, 20, 30, ...
    40, 45, 30, 20];
num=[ ...
    0, 1, 0, 1, ...
    1, 0, 1, 0, ...
    0, 1, 0, 1, ...
    1, 2, 1, 0];

% store the data in table 1 of word
p2_dict=struct(...
    'type', type, ...
    'time', time1, ...
    'num', num ...
    );

type=[...
    "TNTN", "TNTC", "TNTS", "TNSN", "TNSC", "TNSS", "TNSE", ...
    "TCTN", "TCTC", "TCTS", "TCSN", "TCSC", "TCSS", "TCSE", ...
    "TSTN", "TSTC", "TSTS", "TSSN", "TSSC", "TSSS", "TSSE", ...
    "SNTN", "SNTC", "SNTS", "SNSN", "SNSC", "SNSS", "SNSE", ...
    "SCTN", "SCTC", "SCTS", "SCSN", "SCSC", "SCSS", "SCSE", ...
    "SSTN", "SSTC", "SSTS", "SSSN", "SSSC", "SSSS", "SSSE", ...
    "SETN", "SETC", "SETS", "SESN", "SESC", "SESS", "SESE", ...
    ];
time1=[...
    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
    ];
% store the data in table 2 of word
p3_dict=struct(...
    'type', type, ...
    'time', time1 ...
    );

%% calculate the second problem
% 1st, rank_of_passen ==>> flight_id_arrive
% 2nd, rank_of_passen ==>> flight_id_depart
% 3rd, flight_id_arrive ==>> gate_type_arrive (gate_struct.TS)
% 4th, flight_id_depart ==>> gate_type_depart (gate_struct.TS)
% 5th, flight_id ==> flight_property_I_or_D (International or Domestic)
% 5th, gate_type_arrive (gate_struct.TS)   }
%                                                   } ==> the Flow Time (Á÷³ÌÊ±¼ä)
%         gate_type_depart (gate_struct.TS)   }
id1_date2=55;
id2_date2=249;

if(num_of_p==2)
    time_of_passen=[];
    each_of_passen=[];
    time_gap_plane=[];
    for i_pas=1:num_of_passen

flight_id_arrive=find(passen_struct.f_id_arrive(i_pas)==plane_struct.flight_id_arrive(id1_date2:id2
_date2));

flight_id_depart=find(passen_struct.f_id_arrive(i_pas)==plane_struct.flight_id_arrive(id1_date2:id
2_date2));
        if(x1d(flight_id_arrive)>0 & x1d(flight_id_arrive)<num_of_gate+1) & ...
                (x1d(flight_id_depart)>0 & x1d(flight_id_depart)<num_of_gate+1)
            % if true, the passenger will arrive the airport and leave the
            % airport successfully, calculate the time.
            type_DI=find(plane_struct.A_type(flight_id_arrive,:)==1);
            switch type_DI
                case 1
                    arrive_type_DI="I";
                    depart_type_DI="I";
                case 2
                    arrive_type_DI="I";
                    depart_type_DI="D";
                case 3
                    arrive_type_DI="D";
                    depart_type_DI="I";
                case 4
                    arrive_type_DI="D";
                    depart_type_DI="D";
            end

            arrive_gate_type_TS=gate_struct.TS(x1d(flight_id_arrive));
            depart_gate_type_TS=gate_struct.TS(x1d(flight_id_depart));


tot_type=strcat(arrive_type_DI,arrive_gate_type_TS,depart_type_DI,depart_gate_type_TS);

            time_of_passen=[time_of_passen p2_dict.time(find(p2_dict.type==tot_type))];
```

```
                each_of_passen=[each_of_passen passen_struct.pasen_num(i_pas)];           % store
each number of passengers for each guy in sheet2

                time_gap_plane=[time_gap_plane plane_struct.D(flight_id_depart)-
plane_struct.A(flight_id_arrive)];

            end

        end
        time=sum(time_of_passen.*each_of_passen)/sum(each_of_passen);
        tensity=mean(time_of_passen./time_gap_plane);
end




%% calculate the third problem
% 1st, rank_of_passen ==>> flight_id_arrive
% 2nd, rank_of_passen ==>> flight_id_depart
% 3rd, flight_id_arrive ==>> gate_location_arrive (gate_struct.TSNS)
% 4th, flight_id_depart ==>> gate_location_depart (gate_struct.TSNS)
% 5th, gate_location ==> gate_property_I_or_D (International or Domestic)
% 5th, gate_location_arrive (gate_struct.TSNS)   }
%                                                        } ==> the Flow Time (Á÷³ÌÊ±¼ä)
%         gate_location_depart (gate_struct.TSNS)   }
if(num_of_p==3) % only need to add the walk time
        time_of_passen=[];
        each_of_passen=[];
        time_gap_plane=[];
        for i_pas=1:num_of_passen

flight_id_arrive=find(passen_struct.f_id_arrive(i_pas)==plane_struct.flight_id_arrive(id1_date2:id2
_date2));

flight_id_depart=find(passen_struct.f_id_arrive(i_pas)==plane_struct.flight_id_arrive(id1_date2:id
2_date2));
            if(x1d(flight_id_arrive)>0 & x1d(flight_id_arrive)<num_of_gate+1) & ...
                    (x1d(flight_id_depart)>0 & x1d(flight_id_depart)<num_of_gate+1)
                % if true, the passenger will arrive the airport and leave the
                % airport successfully, calculate the time.
                type_DI=find(plane_struct.A_type(flight_id_arrive,:)==1);
                switch type_DI
                    case 1
                        arrive_type_DI="I";
                        depart_type_DI="I";
                    case 2
                        arrive_type_DI="I";
                        depart_type_DI="D";
                    case 3
                        arrive_type_DI="D";
                        depart_type_DI="I";
                    case 4
                        arrive_type_DI="D";
                        depart_type_DI="D";
                end

                arrive_gate_type_TS=gate_struct.TS(x1d(flight_id_arrive));
                depart_gate_type_TS=gate_struct.TS(x1d(flight_id_depart));

tot_type=strcat(arrive_type_DI,arrive_gate_type_TS,depart_type_DI,depart_gate_type_TS);

                % calculate the gate type of TN TS TC SN SC SS SE
                arrive_gate_type_NS=gate_struct.NS(x1d(flight_id_arrive));
                depart_gate_type_NS=gate_struct.NS(x1d(flight_id_depart));

tot_type_TNSC=strcat(arrive_gate_type_TS,arrive_gate_type_NS,depart_gate_type_TS,depart_gate
_type_NS);

                if p2_dict.num(find(p2_dict.type==tot_type))==2 % walk time need to multiple 2
                    walk_time=2*20;
                    if tot_type_TNSC=="TCSC"
                        walk_time=2*15;
                    end
                else
                    walk_time=p3_dict.time(find(p3_dict.type==tot_type_TNSC));
                end

                time_of_passen=[time_of_passen p2_dict.time(find(p2_dict.type==tot_type))+ ...
                    p2_dict.num(find(p2_dict.type==tot_type))*8+ ...   % added the shuttle time, 8
min for each time
                    walk_time]; % added the walk time
                each_of_passen=[each_of_passen passen_struct.pasen_num(i_pas)];           % store
each number of passengers for each guy in sheet2

                time_gap_plane=[time_gap_plane plane_struct.D(flight_id_depart)-
plane_struct.A(flight_id_arrive)];

            end
```

```
        end
        time=sum(time_of_passen.*each_of_passen)/sum(each_of_passen);
        tensity=mean(time_of_passen./time_gap_plane);

end
end



%%%%%%%%%%%%%%%%%%%%result_anaylze.m%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%result_anaylze.m%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%result_anaylze.m%%%%%%%%%%%%%%%%%%%%%%%%
filename_xlsx='InputData.xlsx';
filename_txt='InputData.txt';

load ../data_backup_set9_200_add_generate_random/each_20_max_plane_choose_by_random/b
est242/data250.mat;

[~, ~]=read_data_p1(filename_txt, filename_xlsx);
load plane_struct.mat;
load gate_struct.mat;
load passen_struct.mat;

x1d=x(1,:);
num_of_success_plane=numel(find(x1d<num_of_gate+1))

% input x1d
figure(1)
% wide plane
for i=1:1:size(x1d,2)
        if(x1d(i)<70 & plane_struct.u(i)==1)
        xstart=plane_struct.A(i);
        xend=plane_struct.D(i);

        xx=linspace(xstart,xend,floor(abs(xstart-xend)));
        yy=x1d(i)*ones(size(xx));
        mycolor=[rand rand rand];
        while(mean(mycolor)<0.5)
                mycolor=[rand rand rand];
        end

        plot(xx,yy,'.','MarkerSize', 25, 'Color', mycolor);
        drawnow; hold on;
        text((xstart+xend)/2,x1d(i),plane_struct.puck_id(i),'fontsize',10);
```

```
    end
end


set(gca,'linewidth',2,'FontSize',20,'FontName','Times New Roman');
set(gcf,'position',[60 0 1800 900]);
xlabel("Time/hour");
ylabel("Gate");
xlim([-45, 24*60+45]);
box on;

xtic=[0:60*2:60*24];
xlab=string(xtic/60);
xlab=strcat(xlab, ":00");
xlab(1)="20/Jan/18";
xlab(end)="21/Jan/18";

set(gca,'xtick',xtic);
set(gca,'xticklabel',xlab);     %style 3

ytic=[1:6:69];
ylab=gate_struct.gate_id(1:6:end)';
set(gca,'ytick',ytic);
set(gca,'yticklabel',ylab);     %style 3

figure(2);
% narrow plane
% input x1d
% wide plane
for i=1:1:size(x1d,2)
    if(x1d(i)<70 & plane_struct.u(i)==0)
    xstart=plane_struct.A(i);
    xend=plane_struct.D(i);

    xx=linspace(xstart,xend,floor(abs(xstart-xend)));
    yy=x1d(i)*ones(size(xx));
    mycolor=[rand rand rand];
    while(mean(mycolor)<0.5)
        mycolor=[rand rand rand];
    end

    plot(xx,yy,'.','MarkerSize', 25, 'Color', mycolor);
    drawnow; hold on;
    text((xstart+xend)/2,x1d(i),plane_struct.puck_id(i),'fontsize',10);
    end
end


set(gca,'linewidth',2,'FontSize',20,'FontName','Times New Roman');
set(gcf,'position',[60 0 1800 900]);
xlabel("Time/hour");
ylabel("Gate");
xlim([-45, 24*60+45]);
box on;

xtic=[0:60*2:60*24];
xlab=string(xtic/60);
xlab=strcat(xlab, ":00");
xlab(1)="20/Jan/18";
xlab(end)="21/Jan/18";

set(gca,'xtick',xtic);
set(gca,'xticklabel',xlab);     %style 3

ytic=[1:6:69];
ylab=gate_struct.gate_id(1:6:end)';
set(gca,'ytick',ytic);
set(gca,'yticklabel',ylab);     %style 3



% calculate the T S total number;
figure(2)
num_of_T=numel(unique(x1d(find(x1d<=28))));
num_of_S=numel(unique(x1d(find(x1d>=29&x1d<=69))));

time_used_of_gate=zeros(69,1);

for i=1:1:size(x1d,2)
    if(x1d(i)<70)
    xstart=plane_struct.A(i);
    xend=plane_struct.D(i);

    time_used_of_gate(x1d(i))=time_used_of_gate(x1d(i))+min(xend,24*60)-max(0,xstart);

%       xx=linspace(xstart,xend,floor(abs(xstart-xend)));
%       yy=x1d(i)*ones(size(xx));
%       hold on;
%       mycolor=[rand rand rand];
%       while(mean(mycolor)<0.5)
%             mycolor=[rand rand rand];
%       end
%
%       plot(xx,yy,'.','MarkerSize', 25, 'Color', mycolor);
%       drawnow;
%       text((xstart+xend)/2,x1d(i),plane_struct.puck_id(i));
      end
end
rate_used_of_gate=time_used_of_gate/(24*60);
for i=1:69
      mycolor=[rand rand rand];
      while(mean(mycolor)<0.5)
            mycolor=[rand rand rand];
      end

      x_rate=linspace(0,rate_used_of_gate(i),floor(1000*rate_used_of_gate(i)));
      y_rate=i*ones(size(x_rate));
      plot(x_rate,y_rate,'.','MarkerSize', 25, 'Color', mycolor);
      drawnow; hold on;
      text(rate_used_of_gate(i)+0.02,i,num2str(rate_used_of_gate(i),3),'fontsize',10);
end


set(gca,'linewidth',2,'FontSize',20,'FontName','Times New Roman');
set(gcf,'position',[60 0 1800 900]);
xlabel("Utilization rate");
ylabel("Gate");
xlim([0 1]);
box on;

ytic=[1:6:69];
ylab=gate_struct.gate_id(1:6:end)';
set(gca,'ytick',ytic);
set(gca,'yticklabel',ylab);     %sty
%
% text(1,69,[num2str(num_of_T), "gate T"])
% text(1,60,[num2str(num_of_S), "gate S"])

[time tensity time_of_passen each_of_passen
time_gap_plane]=calculate_passenger_time(x1d,plane_struct,passen_struct,gate_struct,num_of_g
ate,num_of_plane, 3);
fail_num=sum(each_of_passen(find((time_gap_plane-time_of_passen)<0)));
```

```
fail_rate=fail_num/sum(each_of_passen);
figure(3)
time_transf=[5:5:100];
rate_time_transf=[];
for i_time=time_transf
    rate_time_transf=[rate_time_transf sum(each_of_passen(find(time_of_passen<=i_time &
time_of_passen>i_time-5)))];
end
rate_time_transf=rate_time_transf/sum(rate_time_transf);
list=find(rate_time_transf~=0);
time_transf=time_transf(list);
rate_time_transf=rate_time_transf(list);



for i=1:size(time_transf,2)
    mycolor=[rand rand rand];
    while(mean(mycolor)<0.5)
        mycolor=[rand rand rand];
    end

    x_rate=linspace(0.005,rate_time_transf(i),floor(1000*rate_time_transf(i)));
    y_rate=time_transf(i)*ones(size(x_rate));
    plot(x_rate,y_rate,'.','MarkerSize', 55, 'Color', mycolor);
    drawnow; hold on;
    text(rate_time_transf(i)+0.02,time_transf(i),num2str(rate_time_transf(i),3),'fontsize',20);
end




set(gca,'linewidth',2,'FontSize',20,'FontName','Times New Roman');
set(gcf,'position',[60 0 1800 900]);
xlabel("Rate");
ylabel("Transfer time");
xlim([0 1]);
box on;

ytic=time_transf;
ylab=strcat(string(ytic-5), "~", string(ytic));
set(gca,'ytick',ytic);
set(gca,'yticklabel',ylab);     %sty

ylim([min(time_transf)-5, max(time_transf)+5]);
```

```
figure(4)
rate_of_time=time_of_passen./time_gap_plane;
time_transf=[0.1:0.1:1];
rate_time_transf=[];
for i_time=time_transf
    rate_time_transf=[rate_time_transf sum(each_of_passen(find(rate_of_time<=i_time &
rate_of_time>i_time-0.1)))];
end
rate_time_transf=rate_time_transf/sum(rate_time_transf);
list=find(rate_time_transf~=0);
time_transf=time_transf(list);
rate_time_transf=rate_time_transf(list);



for i=1:size(time_transf,2)
    mycolor=[rand rand rand];
    while(mean(mycolor)<0.5)
        mycolor=[rand rand rand];
    end

    x_rate=linspace(0.005,rate_time_transf(i),floor(1000*rate_time_transf(i)));
    y_rate=time_transf(i)*ones(size(x_rate));
    plot(x_rate,y_rate,'.','MarkerSize', 55, 'Color', mycolor);
    drawnow; hold on;
    text(rate_time_transf(i)+0.02,time_transf(i),num2str(rate_time_transf(i),3),'fontsize',20);
end




set(gca,'linewidth',2,'FontSize',20,'FontName','Times New Roman');
set(gcf,'position',[60 0 1800 900]);
xlabel("Rate");
ylabel("Time tension");
xlim([0 1]);
box on;

ytic=time_transf;
ylab=strcat(string(ytic-0.1), "~", string(ytic));
set(gca,'ytick',ytic);
set(gca,'yticklabel',ylab);     %sty

ylim([min(time_transf)-0.1, max(time_transf)+0.1]);
```

```
%% 5 generate a pair for Excel CSV
[~,Pucks]=xlsread(filename_xlsx,'Pucks');
New_Pucks=cell(size(Pucks,1),size(Pucks,2)+1);
New_Pucks(:,1:end-1)=Pucks;
New_Pucks(1,end)={'ÎÊÌâ¶þµÇ»ú¿Ú'};
puck_id=string(Pucks(2:end,1));
for i=1:size(x1d,2)
    if(x1d(i)<num_of_gate+1) % this is feasible
        gate_name=gate_struct.gate_id(x1d(i));
        plane_pk=plane_struct.puck_id(i);
        id_change=find(puck_id==plane_pk);
        New_Pucks(id_change+1,end)={gate_name};
    end
end
```