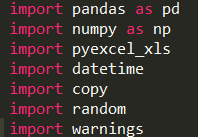
**智慧航空AI大赛总结报告**

# 开发工具，包括系统配置、开发语言、优化工具

## 开发工具

Sublime + ipython

所使用的第三方库：



## 系统配置



## 开发语言

Python

## 优化工具

无

# 结果迭代说明

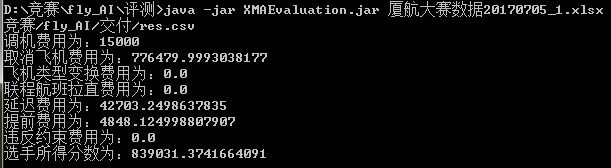
算法总共有5次迭代过程，第一次迭代是先收敛到局部最优解，第二次迭代是在局部最优解附近迭代，减少后面的计算量。第三次迭代向后是自己优化的迭代算法。

因为迭代设计了两种模式，一种是随机收敛模式，一种是选择周围所有邻域解中费用下降最快方向的解来收敛的模式。其中随机收敛模式是用来考证代码的正确性的，速度较快，而选择费用下降最快方向来收敛到局部最优解所能达到的局部最优解的效果离全局最优解更近，解的质量更好。

因为选择费用下降最快方向来收敛需要对当前解的周围所有解都遍历一遍，而且所用的语言是python，因此所花费的时间较长，对于10分钟，20分钟，讨论随机收敛的方法。对于1小时以上，讨论费用下降最快方向收敛的方法。

## 10分钟内的最好结果

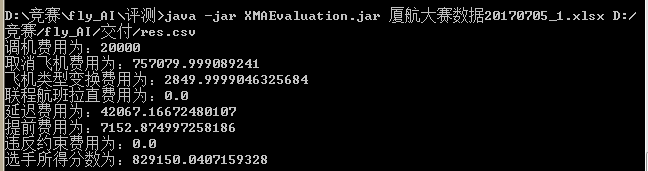
其中，第一次迭代次数为500次，第二次迭代次数为160次，第三次迭代为100次，第四次迭代为20次，第5次迭代为0次。



## 20分钟内的最好结果

因为第一次迭代的结果并不好，而且后面的迭代在这个基础上的优化只有前几十次的优化结果最好，所以20分钟对比10分钟的结果好的程度并不多。

第一次迭代次数为500次，第二次迭代次数为200次，第三次迭代为150次，第四次迭代为70次，第5次迭代为10次。

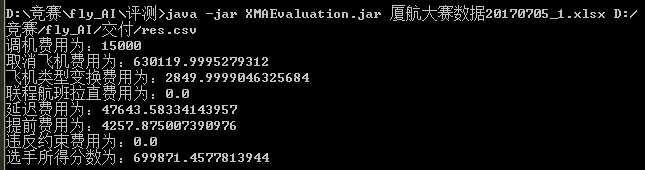


## 60分钟内的最好结果

第一次迭代采用选择费用下降最快方向来收敛。所需要的时间在50分钟左右，所以60分钟内的最好结果不太容易找到。取80分钟左右的最好结果。

第一次迭代次数为500次，第二次迭代次数为1000次，第三次迭代为500次，第四次迭代为500次，第5次迭代为0次。

总花费时间约为85分钟。



## 大于60分钟的最好结果

第一次迭代采用选择费用下降最快方向来收敛。

第一次迭代次数为500次，第二次迭代次数为2000次，第三次迭代为2000次，第四次迭代为2000次，第5次迭代为2000次。

结果如下：

第一次迭代花费时间：

第二次迭代花费的时间：

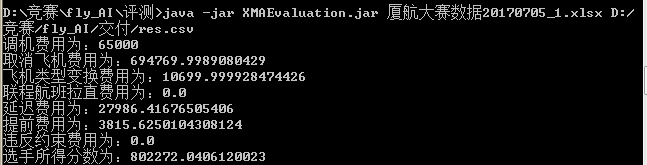
第三次迭代花费的时间：

第四次迭代花费的时间：

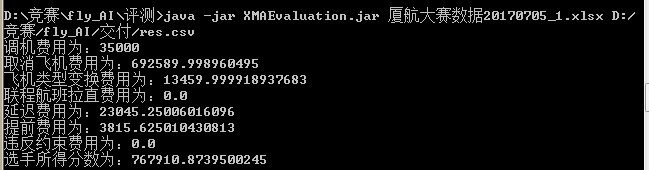
第五次迭代花费的时间：

总共迭代花费时间为：

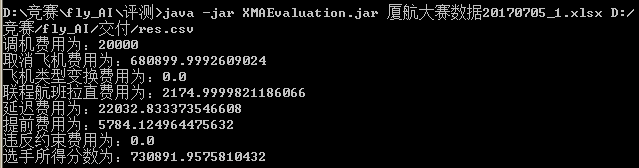
第一次迭代的结果：



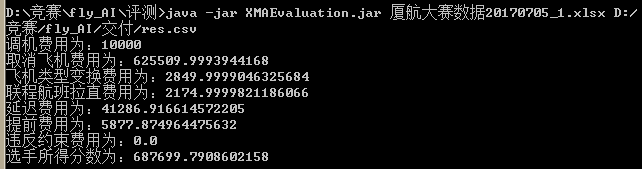
第二次迭代的结果：



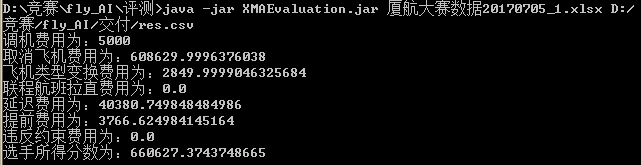
第三次迭代的结果：



第四次迭代的结果：



第五次迭代结果：



因为所使用的语言是python，所以实时性方面不会太如人意。而且迭代过程中邻域解的选择具有随机性，迭代次数也不一样，所以结果与最终提交的结果稍有差别。

# 原始算法原理与改进

本次竞赛所使用的算法是基于唐小卫的博士论文《协同决策机制下航空运输系统不正常航班问题研究》，并在该论文所描述的基础上进行了算法的优化，并进行了改进，使之能适应当前题目的情景。为了更好的介绍所使用的算法原理，将分为三部分来介绍，第一部分是论文所讲述的算法，第二部分是自己所优化的部分，第三部分是总的来介绍一下经过优化后的算法实现的过程。

所使用的算法为GRASP算法（贪婪随机自适应算法）。

## 算法基本原理描述

GRASP算法是一种启发式算法，唐小卫在传统GRASP的基础上，加入了模拟退火算法的思想，使解的质量有一定提升。这里先简单介绍一下加入了模拟退火算法之后的GRASP算法。

先介绍以下几个定义：

航班串：在空间以及时间上连续的一组航班。

航班环：第一个航班的起始机场与最后一个航班终止机场一致的航班串。

尾航班串：一个飞机路线的最后N个连续航班。

子航班环：一个航班串中可以构成航班环的一个子航班串。

该算法是以飞机为单位的，无论是航班串还是航班环，都是对于一个飞机的航班路线来说的。

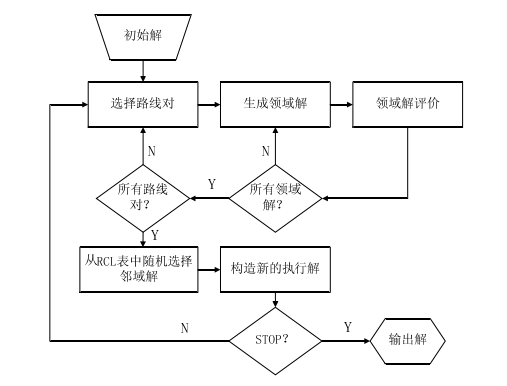
唐小卫的GRASP算法步骤如下：

* 构造初始解，初始解的构造方法是，取消所有受影响的航班。比如一个飞机，有10个航班，其中5,6航班受影响，那么就会生成两个航班串，一个是1到4号航班，一个是7到10号航班，对所有飞机的原航班都做这个处理，可以得到不受影响的m个航班串以及受影响的n个航班串，前m个航班有具体飞机可以执行，后n个代表没有飞机来执行的航班串。
* 对初始解进行领域搜索，将成本下降的解放在一个池中，称为限制选择列表。然后从限制选择列表中随机选择一个解，作为下一次迭代的解。将成本上升的解放在一个池中，当没有成本下降的解时，从成本上升的解中随机选取一个解来作为下一次迭代的解。这样迭代下去，最终达到运行时间或者特定条件。

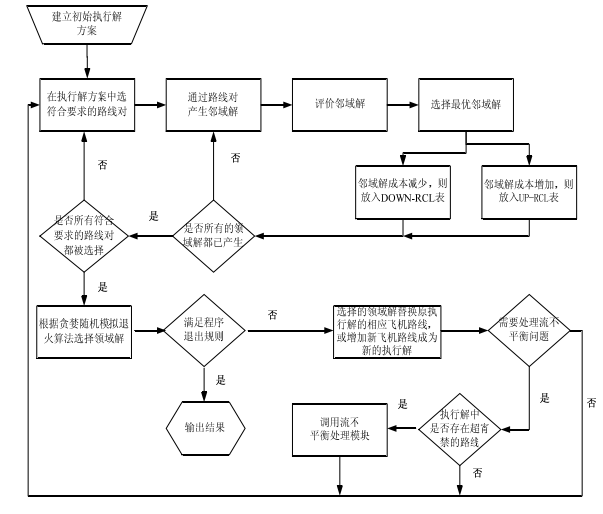
领域搜索的方法有如下几个：

1. 将一个飞机路线的子航班环插入到另一个飞机路线的合适的位置，包括插入到头部，中部，尾部。
2. 交换两个飞机路线起始机场与终止机场都相同的航班串（航班环）。
3. 交换两个飞机的尾航班串。
4. 将一个飞机路线的尾航班串插入到另一个飞机的尾部。
5. 取消子航班环。

算法简化框图如下：



带有模拟退火算法的GRASP算法框图如下：



## 算法的改进

### 搜索方向改变

通过测试发现，下降方向取随机所得到的解的质量并不好，因此测试了几个候选方向，第一个是成本下降最大的方向，第二个是操作过程中时间最紧凑的方向（思路是可以使后面的迭代次数更多，最终解的质量更好），第三个方向是成本+紧凑时间双决定的方向，设置系数来确定哪个做主要方向。

最后测试发现，下降方向取成本下降最快方向的解的质量最好，将成本从100万降到了80万。但是缺点也是有的，那就是需要遍历当前解的所有领域解，而这个运算量随着延时参数的增加以及联程航班是否可拆而越来越大。在最终提交的参数情况下，第一个解的领域成本下降解的个数达到了5000个之多。

### 取多次成本不变的解

在迭代过程中，花费时间最多的就是遍历当前解的所有领域解的过程。在模拟退火的过程中，若每次迭代到成本不变的解以后就去寻找成本下降的解，那么耗时将过长。

为了减少程序运行时间，若当前没有寻找到成本下降的解后，下次取成本不变的解以后，不立刻去寻找成本下降的解，而是继续多次取成本不变的解，使解的结构发生较大变化后，再去寻找成本下降的解。这样可以显著减少程序运行时间。

### 全局基地平衡

在原始的算法中，并没有这个约束，在算法的迭代过程中，并没有对结束航班的机场做任何要求，而要满足这个约束，情况较为复杂。

这里采用了标记+调机的方法来使全局基地平衡，调机的算法写了三种，将在下面详细介绍。

其中标记是指，在n + m个航班路线上，在赛题原给的n个飞机路线所对应的最后一个航班所在的串做上标记，表示这个航班串会飞到最终的机场。所以共有n个标记，迭代过程中会保持n个标记总数不变。比如一个飞机原来执行1到10号航班，5,6受影响取消，那么标记会在7到10号航班的航班串上。

在迭代过程中，尾串插入操作，需要不能插入到已经有标记的航班串的后面。这样可以保证标记次数不变。

在迭代结束以后，寻找n个已有飞机执行的飞机路线中没有标记的k个飞机路线，再在没有飞机执行的航班串中寻找有标记的k个飞机路线。然后将这个k对航班串进行调机操作，将之连起来。形成最终结果。

调机算法介绍如下：

前两种算法的步骤相同，区别是调机算法，其步骤如下：

* 创建调机矩阵(k\*k, 1代表可调机，0代表不可以)。
* 统计k对航班串，将其放在两个列表中。
* 统计第一个已有飞机执行的飞机路线（称为ef），与第一个没有飞机执行的航班串（称为df）之间是否可以调机。过程如下：查看ef的最后一个航班是否可以与df的第一个航班是否可以调机，包括时间是否可以满足，航线限制，宵禁限制等等。然后查看是否可以与第二个航班是否可以调机。。。若可以调机，记录两个飞机路线中对应航班的位置。并将调机矩阵对应位置标记为1。若与所有没有飞机执行的航班串之间都无法调机，那么将调机矩阵对应位置标记为0。
* 循环第二步，创建调机矩阵。
* 通过调机矩阵使用算法来获得调机方案。

第一个算法，首先选择矩阵元素为1，计算min(行中1个的个数，列中1的个数)，选择最小的对应的调机矩阵的位置，将该位置对应的一对飞机路线进行调机操作，并将该元素所在的行与所在的列所有的元素置为0，表示当前飞机路线已经使用过了，重复当前过程，直到所有的飞机路线都调机结束。这里计算min(行中1个的个数，列中1的个数)是因为先将不容易调机的飞机先进行调机，以免后面无法调机。

第二个算法，是DFS算法来获得最优的调机方案，相比于第一个算法，成本更低，但是耗时太长，不是多项式时间的时间复杂度。具体做法是，在构造调机矩阵的同时，记录下当前调机方案的花费矩阵。通过DFS进行遍历，选择所有结果中成本最小的作为最终调机结果。

第三个方案是基于机场-机场字典的（后面将详细介绍），思路是先统计k个最终要达到的机场。对于k个最终机场中的第一个最终机场ep，然后对于第一个有飞机执行的航班串，在机场-机场字典中搜索1到ep的航班方案，若有，则记录调机费用，然后搜索2到ep的航班方案，记录调机费用……直到搜索完毕79（塞题共有79个机场）到ep的航班方案，记录费用。然后选择费用最小的方案，将第一个有飞机执行的航班串先使它能够到达最终机场。对剩下的k-1个有飞机执行的航班串都做这个操作，获得最终的调机方案。

前两个调机方案是刚开始的时候没有引入机场-机场字典的时候写出来的，但是调机成功的可能性更大，但是第二个DFS方案因为时间不是多项式复杂度，所以当k<=7时，可以较快的得出结果，对于大于7的规模，耗时太长。第一个方案在各种情况下都很快的得到结果，但是结果的耗费可能会较高。

方案成本是第三个方案<第二个方案<第一个方案。

因此最终的选择方案是，首先尝试方案三，若方案三不可行，根据规模来选择使用方案1或者方案2。

### 机场-机场字典

对于传统的GRASP算法，没有执行的航班是成以航班串的形式存在的，但是这样向已有飞机执行的航班串中插入的时候，就可能效果不如人意，因此引入了机场-机场字典，以期进行更灵活的操作。

机场-机场字典的构造过程如下：

* 将没有飞机执行的航班串首先先拆成单个航班，或者一对联程航班，形成q个航班串。
* 对于q个航班串中第一个航班串，记录其开始机场与终止机场。在机场-机场字典中，key设为(开始机场，终止机场，飞机类型)，把航班串加入到value中。
* 对于第k个航班串，比如25机场-37机场的航班串，除了将(25，37，飞机类型)直接加入到字典中，还有以下两种可能：

第一种是m机场-25机场，与25机场-37机场构成m机场-37机场的航班串。

第二种是37机场-m机场，与25机场-37机场构成25机场-m机场的航班串。

对于这两种操作，通过遍历m的值，在字典中寻找对应的所有航班串，将可能的组成结果加入到字典中。

* 遍历完所有的航班串以后，形成最终的航班字典。

构建完机场-机场字典以后，对于航班环的插入，互换尾航班串，互换首尾相同的航班串，航班串插入尾部，都可以通过查找机场-机场字典来获得这些航班串。

机场-机场字典的本质就是讲航班打散，然后构建出所有的可能组成方式。这样可以得到更多的领域解，从而得到更好的结果。

通过机场-机场字典来进行GRASP算法的缺点就是，如果剩余未执行航班过多，那么机场-机场字典就会过于庞大，导致算法迭代一次的时间过长。因此，最好先通过传统GRASP将未执行航班规模减小，再进行构建机场-机场字典操作。

### 加入可通过延迟不取消的航班

本着将所有能够使用的航班都使用的原则，将可以通过延迟10小时内不取消的航班延迟一定时间放入到没有飞机执行的航班串中。

事实上，在构建初始解的时候，有一个延迟参数，但是这个延迟是以飞机为单位的，会使需要延迟的航班后面的所有航班都延迟，因此若延迟参数太大，会导致延迟成本过大。

这里的加入航班，是以航班为单位的，只要满足当前航班不处于台风场景中即可。

后面将做进一步解释。

### 引入延时操作

延时字典：记录每个航班的延时时间的字典。

延迟操作的做法就是改变延时字典中的航班的延时时间的操作。

具体做法如下：

在交换尾串、插入子航班环、插入尾串、交换首尾相同的串的时候，在前后航班时间约束的判断过程中，不进行严格的判断，而进行允许将后续航班延迟的操作。只要成本可以下降，就可以放入成本下降解池中，作为备选解。

对于交换尾串：允许未执行的航班串进行延迟操作。

对于插入子航班环，允许未执行的航班串与已有飞机执行的航班串同时进行延迟的操作。

对于插入尾串，允许未执行的航班串进行延迟操作。

对于交换首尾相同的串的操作，允许未执行的航班串与已有飞机执行的航班串同时进行延迟的操作。

### 加入新操作

加入减少延时时间的新操作，思路是可以更紧凑的使航班排在一起，从而能够插入更多的未执行航班。这个操作，最终使成本从71万降到了66万，所以是一个非常有效的操作。

具体做法如下：

对于一个已经有飞机执行的航班，从第二个航班起：

对于当前航班，若可以减小当前航班的延时时间，则延迟，同时把新的延迟时间写入延迟字典中。下一个航班基于新生成的延迟字典来操作。

### 不同飞机类型组成航班串

在生成机场-机场字典时候，不进行飞机类型的判断就可以了。在计算成本时，判断若飞机类型不同，则不降低成本（因为取消与换机型惩罚值一样）。

### 总结

这次竞赛，最终选择了GRASP这种启发式算法，初始效果不如人意，后面做了很多尝试，最后发现，只有上面几个措施较为有效。所以最终采取了上面几个优化的方法。最终的结果虽然还算可以，但是GRASP这种算法由于其是启发式算法，虽然较容易理解，但是精确性并不好。后面虽然采取了很多措施来从航班与时间两个方面增加算法的灵活性，但是时间上的灵活性由于太过于复杂，该算法并不能很好的来处理，所以最终的结果相比于前三名还是有很大的差别。

# 算法实现详细过程

## 算法中基本概念

### 解的构成

解主要由三个部分构成：

第一个部分是有飞机执行的航班串。在本次竞赛中共有142个飞机，所以有飞机执行的航班串共有142个，其中包括空串也无所谓。另外每个航班串会有以下几个属性：飞机号、飞机类型、飞机最开始的机场(防止空串时不从初始机场出发)、是否包含最终机场(基地全局平衡使用)。

第二个部分是没有飞机执行的航班串。这些航班串的个数是不固定的，而且也没有顺序，若包含空串，则空串的包含最终机场属性应该为1，即那些无用的空串可以扔掉。

每个航班串应该要有的属性有以下几个：原飞机类型、航班的最终机场(防止空串时找不到最终机场)、是否包含最终机场。

第三个部分是延迟时间字典。每个航班都会有自己的一个延迟时间，记录在延迟时间字典中。

通过这种形式的解，通过调机操作，然后就可以通过解的第一个部分与第三部分延时时间就可以得到最终的解。

### 一些使用的字典或集合

国内机场集合：将所有国内航班的开始机场与结束机场都放入国内机场的集合。

飞行时间字典：通过开始机场、终止机场，飞机类型可以查找飞行时间。

航线限制集合：将起始机场、终止机场、飞机号有航线限制的组合放入。

机场关闭字典：记录各个机场宵禁时间的字典。

航班字典：记录航班的一些信息，有是否联程航班，起始时间，终止时间，起始机场，终止机场，重要系数，航班的原飞机型号，是否取消，互相联程的航班ID，是否拉直。

参考延迟时间字典：记录每个航班最小的延迟时间。用作减少EF中串的延迟时间使用。

### 辅助函数

#### 判断函数

* 判断宵禁函数：

参数：机场关闭字典，机场，时间

功能：判断机场与时间是否处于宵禁中。

* 判断是否航线限制函数：

参数：飞机号，航班列表，航班字典，航线限制字典

功能：判断一个飞机是否对一个航班串是否有航线限制。

* 判断是否时间一个区间函数：

参数：时间1，时间2，台风场景(dataframe)，机场

功能：判断两个时间是否在一个台风场景的开始之前，或者结束之后。这样做是为了防止导致台风期间停机。若一个开始时间在台风之后的航班串与一个开始时间在台风之前的航班串相交换，那新生成的航班串就会在台风期间停机。

#### 生成函数

* 获得航班环函数

参数：航班字典，航班列表

功能：寻找一个航班串中的所有航班环

* 获得相同起始机场与结束机场的航班串

参数：航班字典，两个航班列表

功能：寻找两个航班列表中所有起始机场与结束机场相同的航班子串

* 获得相同起始机场的航班尾串

参数：航班字典，两个航班列表

功能：寻找两个航班列表中所有起始机场相同的航班尾串

* 分离航班串函数

参数：航班字典，航班列表

功能：将航班列表分成单个航班或者一对联程航班。

* 产生新延迟字典函数

参数：延迟时间，航班列表，原延迟字典，航班字典，机场关闭字典，台风场景(dataframe)

功能：对航班列表中的航班串进行延迟，长度为延迟时间。返回新的延迟字典。

## 迭代算法的一次操作函数

有产生成本下降解的操作，也有产生成本不变的用来模拟退火跳出局部最优解的解得操作。

**成本下降的解的产生：**

选择EF中的一个串，减小航班的延迟时间。将解放入下降池中。

选择EF中的一个串与DF中的一个串，进行尾串插入，子航班环插入，尾串交换，首尾相同串交换，将产生的成本下降的解放入下降池中。

随机模式就是随机选择EF的一个串，DF中的一个串，直到产生下降的解以后就停止。

寻找全部解就是遍历EF的所有串，遍历DF的所有串，将所有下降的解放入下降池中。

**成本不变的解的产生：**

当没有下降的解的时候，随机选择EF中的两个串：

进行首尾相同的串互相交换、进行尾串的互换、将第二个串的子航班环插入到第一个串中。把产生的解都放入到成本不变池中。

**对于硬约束的条件，主要有以下几个方面：**

机场约束：必须保证前一个航班的到达机场与后一个航班的起始机场一致。

时间约束：必须保证中间间隔时间大于最小过站时间或原来时间。

联程航班约束：为了简单考虑，不允许联程航班拆开。

飞机-航线限制：保证新加入的航班不会违反飞机-航班限制。

台风场景不能停机：保证台风场景不能停机，方法是调用辅助函数，保证两个时间在同一范围内。

**详细过程举例：**

下面对插入航班环操作进行详细的说明，

对于EF中的一个航班串ef，对于DF中的一个航班串df：

1. 将df的子航班环插入到ef的合适的位置，可以插入到头部，中部或者尾部。

对于硬性约束，按如下步骤来判断：

* for 航班 in ef 的所有航班
* for 航班环 in df 的所有子航班环
* 判断df的子航班环拿掉是否会造成联程航班的拆开。若是则continue。
* 判断ef的当前航班的到达机场是否与航班环的起始机场相同，若不是则continue。
* 判断ef的当前航班的到达时间是否大于航班环的起始时间一定值，若是continue。若小于负的最小过站时间，则不用重新构造延迟字典，否则重新构造延迟字典，若无法构造，则continue。
* 判断ef的当前航班的下一个航班的起始时间是否小于航班环结束时间一定值，若是则continue。若小于负的最小过站时间，则不用重新构造延迟字典，否则重新构造延迟字典，若无法构造，则continue。
* 判断当前航班与下一个航班是否为联程航班，若是，则continue。
* 判断当前飞机是否对子航班环是否有飞机-航班限制。若有则continue。
* 判断当前航班的到达时间是否与子航班环的起始时间在一个时间范围内，判断下一个航班的起始时间是否与子航班环的到达时间在一个范围内。若不在则continue。（防止台风期间停机）
* 计算当前新解的成本，若成本下降，则改变ef与df的航班串的内容。放入成本下降解池中。

## 参数接口

是否选用随机模式(获得下降解就停止 或者 遍历当前解的所有领域解获得所有下降解)

初始解延迟时间

迭代过程中允许延迟时间

迭代次数

迭代等待次数(成本不再下降后，等待多少次再检测成本下降解)

## 详细步骤

### 形成初始解

为了便于编写与阅读，将已有飞机执行的航班串的列表记为EF(Execute Flight)，将未执行的航班串的列表记为DF(Dismiss Flight)。形成初始解主要有以下几个过程：

1. 筛选出受影响需要取消的航班，起飞受限标记为1，降落受限标记为2.
2. 寻找取消标记为1(起飞受限)的可以通过提前来不取消的航班。若可以提前，分两种情况，第一种情况是提前后仍然满足过站时间，这种可以直接提前，将取消标记改为0，等到后面直接放入EF中。另一种是提前后不满足中间过站时间，在判断不会造成联程航班断开的情况下，将提前航班直接放入到DF中。
3. 将台风场景结束后的50分钟内起飞机场为台风机场的起飞的航班都延迟到台风场景结束时间+50分钟时起飞。在本次题目中，就是将所有起飞时间是5月7日17点到17点50之间的航班都推迟到17点50分起飞。
4. 设置一个延迟参数x(小时)，对于每一个飞机的原始航班，寻找到其第一个可以通过延迟x小时之内不取消的航班，首先判断后面的航班若延迟是否会进入机场关闭的时间内，若不会，则将后面的航班都顺延推迟满足最小过站时间的最小延迟时间，将延迟时间写入到延迟字典中，且将取消标记改为0。若会，则都不推迟。
5. 构建航班字典。
6. 对于一个飞机路线，将不受影响的部分放入到EF中，将受影响的部分放入到DF中。对所有飞机都做这个操作，生成EF与DF。

### 迭代过程

总共进行了5次迭代过程。

**第一次迭代：**

第一次迭代时获取局部最优解的过程，将EF中的航班数量大大增加。过程就是对初始解进行不断的迭代，直到不能下降的时候停止。

**第二次迭代：**

第二次迭代就是当没有成本下降解以后，取成本不变的解，经过一定次数以后，再检测是否有成本下降解。

到达迭代次数之后停止。

**第三次迭代：**

将两个联程航班都没有使用过的可拉直的联程航班拉直。

将DF中的航班串使用分离函数进行拆分为单个航班以及一对联程航班。

对DF构建机场-机场字典。

通过机场-机场字典进行尾串互换、首尾相同串互换、插入子航班环、插入尾串操作。

达到迭代次数之后停止。

**第四次迭代：**

将可以通过延迟10小时内而不取消的航班进行延迟，将这些航班加入到DF中。

将DF中的航班串使用分离函数进行拆分为单个航班以及一对联程航班。

对DF构建机场-机场字典。

通过机场-机场字典进行尾串互换、首尾相同串互换、插入子航班环、插入尾串操作。

达到迭代次数之后停止。

**第五次迭代：**

将DF中的航班串使用分离函数进行拆分为单个航班以及一对联程航班。

对DF构建机场-机场字典（允许不同机型组合）。

通过机场-机场字典进行尾串互换、首尾相同串互换、插入子航班环、插入尾串操作。

达到迭代次数之后停止。

### 调机及解的生成

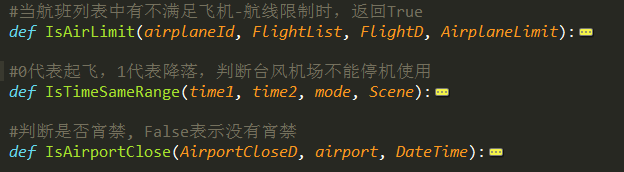
调用调机函数对最终的解进行调机操作，通过EF与延迟字典生成最终结果。

## 所有函数列表

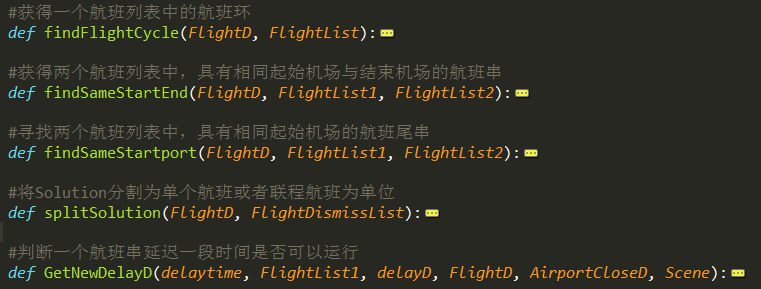
获取初始数据：



辅助判断函数



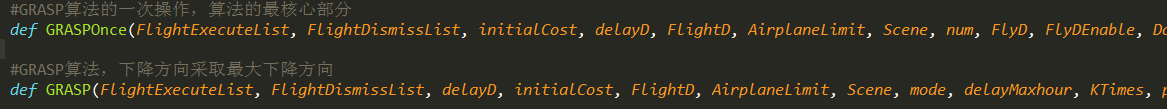
辅助生成函数



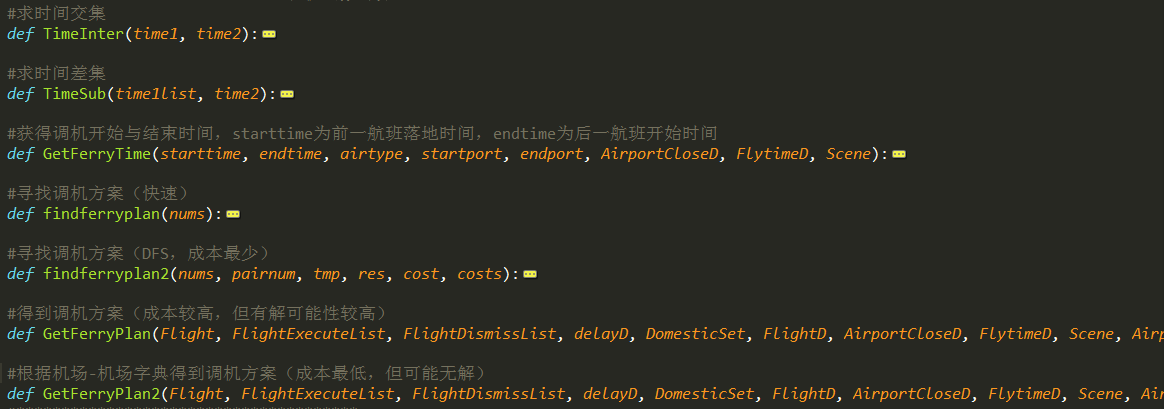
获取初始解



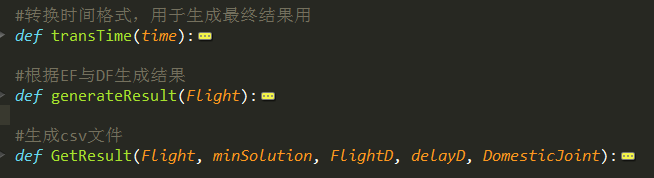
迭代函数



调机函数



生成结果



# 感悟

对于这次比赛，经过20天的努力，感悟还是挺多的……

这其实是我第一次打比赛，而且第一次一个人打比赛，其实是为了检验一下自己的解决问题能力才来参加的，从7月10号才开始看到题目，感觉题目真是难，瞬间就被打击了……本来想着能做个可行解出来就算了，但是作为一个比较自信的人，想想要是不拿个奖就白参加了，哈哈。

后来就参照做项目的经验，从10号开始给自己划了时间节点，2天时间理解题意，4天时间取查找资料，确定模型以及方法，5天时间来实现模型，得到最初的可行解。剩下的10天用来优化模型。题意的理解倒是用了两天不到的时间就完成了，但是在查找资料与模型的选择上面出现了很大的纠结。国内以及国外的论文在航空路线恢复算法上面主要使用的只有GRASP算法以及时空网络模型两种算法，只是求解的方式各有不同或者对算法进行了一些优化而已。

刚开始我是偏向于时空网络模型的，因为在一个论文中写到，它可以解决调机、延迟、多机型、取消等多种操作并存的场景，而且是以机场为单位的，在满足全局基地平衡这个麻烦的约束上面有着很大的优势。但是在读了国内的好多论文以及国外的一篇论文以后，发现对于我一个人来说，还是过于复杂了….而且有些论文对时空网络模型的时间节点的理解还不同。最后受时间的影响，最终下决定使用GRASP这种理解起来非常容易的启发式算法。

虽然不知道前几名用的什么算法，不过GRASP算法由于其精确性太差导致结果达到80W成本，离当时的第一名68W差了茫茫多。后面就开始了慢慢的优化过程，每天各种找思路，各种尝试新方法，成本慢慢的降低……

后期受黑马的影响较大，有几个很厉害的队伍居然最后两天才提交……导致在倒数第二天的时候降到了第10名，当时觉得肯定前十不保了，差点自暴自弃了。但是后来还是冷静的去分析已经得到的最优解，最后想出来减少延迟时间这个操作，将成本从71万直接降到了66W，保住了前十。

总的来说，第一次做比赛取到了这个成绩已经很满意了，无论在解决问题上还是心态上，觉得自己都有很大的进步。但是以后还是找个队友吧~