运行环境

Python 3.6+

Gurobi 9.0.1+

代码结构

`Options.py` 是论文中提到的各种常量参数配置；
`Utils.py` 是常量参数之后的各种派生内容，例如论文里面提到的 A、U；
`Models.py` 是封装的三个模型，其中 P3 模型 MedianPBasedOnSystemOptimality 需要使用 Gurobi 求解器。它是论文中 Lingo 求解器的替代品，懒得配置的话，可以注释掉此部分代码；
`Optimizes.py` 是智能算法的求解器，Python 代码很好读，可以当伪代码用；
`Main.py` 是论文中提到的数据的运行案例，P1 和 P2 的最优解与论文一致。P3 的最优解有多个，由于求解器的搜索方式不同，它的最优解略有不同，但是最优的目标函数值是一致的。

模拟退火算法

可以叫 Fixed 0-1 变量退火算法，模型的输入参数为优化目标函数 function，解变量验证函数 testFunction，固定的真值个数 (即 1 的个数) trueValueNum，固定的假值个数 (即 0 的个数) falseValueNum，初始温度 T，冷却率 cool。

算法的步骤 (对照代码讲解)

Step1 生成初始解 (Line 15)，由预先指定的 trueValueNum 和 falseValueNum 决定它包含的内容。生成的解是有序的，需要先使用 shuffle 打乱 (Line 16)，以模拟随机生成的初始解。由于该解可能不满足题设要求 (穷)，需要使用 testFunction 进行解变量验证 (Line 17~19)。

Step2 当系统温度 T 大于设定的最小温度 10^{-8} 时，执行搜索。它先在 falseValueNum + trueValueNum 个维度中抽取出 2 个维度 (不重复) minIndex 和 maxIndex (Line 25~27)，然后将这两个维度中间的数据进行反转 (Line 30~31)。

Step3 Step 2中得到的新的解变量同样有可能不满足题设要求，继续使用 testFunction 进行解变量验证。不满足条件时，循环生成，直到满足为止 (Line 34~39)。

Step4 计算成本值 (目标函数值, Line 42~43)，并判断是否接受该值 (Line 46~47)。判断逻辑为：如果新的解变量目标函数值 E_{new} 比原先的解变量 E_{old} 更优，则接受该值；当新的解变量比原先的解变量更差时，它以：

$$e^{-(\frac{E_{new}-E_{old}}{T})}$$

的概率接受该值。

Step5 完成一次搜索后，系统的温度下降至原先的 cool 倍 (Line 50)，返回 Step 2.

```
1  import math
2  import random
3
4  class SimulatedAnnealing(object):
5      def __init__(self, function, testFunction, trueValueNum=5,
   falseValueNum=5, T=1e8, cool=0.9):
6          self.function = function
7          self.testFunction = testFunction
8          self.trueValueNum = trueValueNum
9          self.falseValueNum = falseValueNum
10         self.T = T
11         self.cool = cool
12
13     def run(self):
14         # 随机的初始解
15         vec = [True] * self.trueValueNum + [False] *
   self.falseValueNum
16         random.shuffle(vec)
17         while (self.testFunction(vec)):
18             vec = [True] * self.trueValueNum + [False] *
   self.falseValueNum
19             random.shuffle(vec)
20         count = 0
21
22         while self.T > 1e-8:
23             count += 1
24             # 随机选择两个维度
25             indexTuple = random.sample(range(self.falseValueNum +
   self.trueValueNum), 2)
26             minIndex = min(indexTuple)
27             maxIndex = max(indexTuple)
28
29             # 创建一个代表解的新列表，逆序
30             vec_copy = vec.copy()
31             vec_copy[minIndex:maxIndex + 1] =
   reversed(vec_copy[minIndex:maxIndex + 1])
32
33             # 判断这种改变是否合法
34             while (self.testFunction(vec_copy)):
35                 indexTuple = random.sample(range(self.falseValueNum
   + self.trueValueNum), 2)
36                 minIndex = min(indexTuple)
37                 maxIndex = max(indexTuple)
38                 vec_copy = vec.copy()
```

```
                vec_copy[minIndex:maxIndex + 1] = reversed(vec_copy[minIndex:maxIndex + 1])

            # 计算成本
            e_origin = self.function(vec)
            e_copy = self.function(vec_copy)

            # 判断是否为更优解
            if e_copy < e_origin or random.random() < pow(math.e, -(e_copy - e_origin) / self.T):
                vec = vec_copy.copy()

            # 温度下降
            self.T = self.T * self.cool
            # print(f"This is NO.{count} iteration, the optimal solution for {vec}, f(x) = {self.function(vec)}")
        return vec

```