工作指派问题可以描述如下： n个工作将要指派给n个工人分别完成， 工人 i 完成工作 j 的时间为 dij，问如何安排可使总的工作时间达到最小？请设计一种 SA 算法来解决上述指派问题。

提示：设 n=100，要求随机产生 dij 。讨论 SA 算法相关参数（例如初始温度、内循环次数、降温函数 等）的灵敏性。以小组的形式提交实验报告以及源代码。

实验代码（python）如下：

# -\*- coding: utf-8 -\*-  
# 用模拟退火算法（SA）解决指派问题  
**import** numpy **as** np  
**import** pandas **as** pd  
  
n = 100 # 100个工人做100个工作

**# 生成100x100的excel矩阵  
writer = pd.ExcelWriter('output.xlsx')  
df\_random = pd.DataFrame(data=np.random.randint(1, 100, size=(n, n))) # 生成1-100之间的随机数  
df\_random.to\_excel(writer, 'Sheet1')  
writer.save()**# 读取随机数矩阵  
excel\_path = **'output.xlsx'**d = pd.read\_excel(excel\_path)  
d = np.array(d) # 转换为数组  
d\_mean = np.mean(d) # 生成的随机数的均值  
  
  
# 时间函数  
**def** total\_time(arrange):  
 total\_time = 0 # 初始时间设为0  
 **for** i **in** range(n):  
 # 任务安排的形式如[2,1,4,3]表示第一个人做第二项工作，第二个人做第一项工作，第三个人做第四项工作，etc  
 total\_time = total\_time + d[i, arrange[i]]  
 **return** total\_time  
  
  
T = 200 # 设置初始温度，对应加温过程  
t = 0.8 # 终止温度-> 0  
alpha = 0.94 # 温度下降率，使得降温过程足够慢  
  
now\_arrange = np.random.permutation(n) # 随机生成一个初始化安排方案  
print(now\_arrange)  
  
now\_time = total\_time(now\_arrange) # 当前方案的消耗时间  
  
new\_arrange = now\_arrange.copy() # 新方案  
  
best\_arrange = now\_arrange.copy() # 保留历史最优安排  
best\_time = now\_time # 保留历史最优时间  
  
x = 0 # 定义x计算总循环次数  
y = 0 # 定义y计算外循环次数  
# 外循环(温度)  
**while** t < T:  
 y = y + 1 # 外循环+1  
 markov\_len = int(6000 / T) # 根据T动态设置内循环次数，随着T的减小，内循环次数增加;使用地板除取整  
 **for** i **in** range(markov\_len): # (内循环，尽量保证达到热平衡)  
 x = x + 1 # 内循环+1  
 **while True**:  
 k = np.random.randint(0, n, 2) # 随机产生新解  
 **if** k[0] != k[1]: # 保证是两个不同的位置  
 **break** new\_arrange[k[0]], new\_arrange[k[1]] = new\_arrange[k[1]], new\_arrange[k[0]] # 交换两个位置得到新的路径  
 new\_time = total\_time(new\_arrange) # 计算新的安排所消耗的时间  
 **if** new\_time < now\_time: # 无条件接受新解.  
 now\_time = new\_time # 更新时间  
 now\_arrange = new\_arrange.copy() # 更新安排  
  
 **if** new\_time < best\_time: # 更新历史最优解  
 best\_time = new\_time  
 best\_arrange = new\_arrange.copy()  
 print(best\_arrange, best\_time)  
  
 **else**: # Metropolis准则：以概率接受新状态  
 delta = new\_time - now\_time  
 **if** np.random.rand() < np.exp(-delta / T): # 如果Metropolis概率大于随机生成的服从0-1均匀分布的随机数则接受，  
 # 但是不更新历史最优解.  
 now\_time = new\_time  
 now\_arrange = new\_arrange.copy()  
 print(now\_arrange, now\_time)  
 **else**:  
 new\_arrange = now\_arrange.copy() # 否则拒绝新解,维持原解.  
  
 T = T \* alpha # 降温函数(按比例下降)  
  
# 输出最终结果  
  
print(**'外循环次数:'**, y) # 输出外循环次数  
print(**'总循环次数:'**, x) # 输出总循环次数  
print(**'dij的均值:'**, d\_mean) # 输出dij的均值  
print(**'最优时间:'**, best\_time) # 输出最优时间  
print(**'最优安排dij均值：'**, best\_time / n) # 最优安排dij均值  
print(**'最优安排：'**)  
**for** i **in** range(n): # 输出最佳安排  
 print(i + 1, **'->'**, best\_arrange[i] + 1, **','**, end=**""**)

外循环次数: 90

总循环次数: 122680

dij的均值: 50.1842（其中dij是1-100之间按均匀分布生成的随机数）

最优时间: 306

最优安排dij均值： 3.06

最优安排：

1 -> 56 ,2 -> 62 ,3 -> 24 ,4 -> 29 ,5 -> 19 ,6 -> 34 ,7 -> 9 ,8 -> 68 ,9 -> 39 ,10 -> 26 ,11 -> 63 ,12 -> 42 ,13 -> 4 ,14 -> 33 ,15 -> 66 ,16 -> 46 ,17 -> 99 ,18 -> 11 ,19 -> 80 ,20 -> 76 ,21 -> 83 ,22 -> 94 ,23 -> 38 ,24 -> 73 ,25 -> 87 ,26 -> 70 ,27 -> 54 ,28 -> 78 ,29 -> 2 ,30 -> 59 ,31 -> 8 ,32 -> 37 ,33 -> 55 ,34 -> 90 ,35 -> 35 ,36 -> 15 ,37 -> 14 ,38 -> 82 ,39 -> 96 ,40 -> 53 ,41 -> 49 ,42 -> 100 ,43 -> 52 ,44 -> 57 ,45 -> 48 ,46 -> 6 ,47 -> 44 ,48 -> 81 ,49 -> 71 ,50 -> 91 ,51 -> 72 ,52 -> 40 ,53 -> 47 ,54 -> 61 ,55 -> 28 ,56 -> 93 ,57 -> 13 ,58 -> 92 ,59 -> 45 ,60 -> 97 ,61 -> 12 ,62 -> 1 ,63 -> 5 ,64 -> 65 ,65 -> 32 ,66 -> 20 ,67 -> 86 ,68 -> 69 ,69 -> 89 ,70 -> 58 ,71 -> 16 ,72 -> 75 ,73 -> 60 ,74 -> 84 ,75 -> 88 ,76 -> 43 ,77 -> 77 ,78 -> 7 ,79 -> 51 ,80 -> 31 ,81 -> 30 ,82 -> 67 ,83 -> 10 ,84 -> 18 ,85 -> 98 ,86 -> 41 ,87 -> 64 ,88 -> 79 ,89 -> 85 ,90 -> 21 ,91 -> 50 ,92 -> 74 ,93 -> 95 ,94 -> 17 ,95 -> 27 ,96 -> 36 ,97 -> 3 ,98 -> 25 ,99 -> 23 ,100 -> 22 ,

以上结果表示：1号工人做第56项工作，2号工人做第62项工作，以此类推直到100号工人做第22项工作。

结果分析：

初始温度T = 200 ，对应加温过程

终止温度t = 0.8->0

内循环次数markov\_len = int(6000 / T) 根据T动态设置内循环次数，随着T的减小，内循环次数增加

降温函数T = T \* alpha 按0.94的比例下降

当dij是1-100之间按均匀分布生成的随机数时，最终优化结果保持在360左右，好的时候能达到306，坏的时候为380.

当初始温度或者内循环次数过低时，由于搜索次数很少，结果很差.

当初始温度或者内循环次数过高时，搜索次数显著增加，使得搜索时间过长，而且结果并没有明显变好.

在总循环次数相同的情况下，在低温处增加内循环次数要比在高温处增加内循环次数效果更好，而内循环次数与温度成反比，可以有效地在减少高温处的低效率搜索，从而提高整体搜索效果。

下降温度函数对搜索方式也有着较大的影响。使用成比例下降会在低温处搜索得比高温处更细致一些，因此使用成比例的方式要比均匀下降固定温度的效果更好。

当终止温度过低时也会导致搜索次数显著增加，搜索时间过长，结果并没有明显变好的情况.

降温函数决定SA何时从广域搜索转变为局域搜索，若温度下降过快则SA将很快从广域搜索转变为局域搜索，可能使得SA过早陷入局优。若下降过慢，则会影响算法进程，增加搜索时间.

我们选择简单易行的比例下降法，比例调试到0.94较为合适.

组员得分：

本组组员通力合作，一起完成此次作业，各人得分各占25%

操晶10

高婉卿10

张云佳 10

胡腾辉10