背景挖掘

在离散数学甚至整个理论计算机领域，一个重大的问题是P与NP的问题。NP问题，也就是并非知道是否存在一个多项式时间能够解决，但是可以在一个多项式时间内验证并得出这个问题的一个正确解。理解为通过猜的方式来得到一个解，然后再验证这个解是否符合答案。

旅行商问题是一个旅行商问题是一个典型的NP-C问题，我用模拟退火这一启发算法来猜出相对更优解。

问题描述和数学建模

自然语言描述

一个旅行商从中心城市出发，要遍历周围的n个城市，最后回到中心城市，尝试找出最短路径。

数学建模

已知条件

给城市编号，中心城市编号为1，其它城市编号为2~$n+1$，然后$n+2$也表示中心城市，那么所有解构成的集合就是$\{1,2,3,\ldots,n+2\}$首尾固定的排列。所有城市的坐标已知，为了计算方便，可以构建两两城市之间的距离矩阵d，其中d_{ij}表示从城市i到城市j的距离。

重要的是，在本次大作业中，我们认为来回的距离是相同的。

目标函数

$$d = f(x_1, x_2, x_3, \dots, x_i, \dots x_j, \dots, x_{n+2}) = \sum_{i=1}^{n+1} d_{x_i, x_{i+1}}$$

新解的产生

设某一温度下的第k个解是$< x_1, x_2, x_3, \dots, x_i, \dots x_j, \dots, x_{n+2} >$
我们通过交换x_i和x_j来产生新的路径$< x_1, x_2, x_3, \dots, x_j, \dots x_i, \dots, x_{n+2} >$
可以先方便地得到产生的路径差为

$$\delta_d = d_{x_{i-1}, x_j} + d_{x_j, x_{i+1}} + d_{x_{j-1}, x_i} + d_{x_j, x_{i+1}} - d_{x_{j-1}, x_j} - d_{x_j, x_{j+1}} - d_{x_{i-1}, x_i} - d_{x_i, x_{i+1}}$$

$$d_{k+1} = d_k + \delta_d$$

外加条件

为了实现最简单的核心功能并简单检验，我人工制造了一个14×14的距离矩阵

代码实现和算法流程图

具体实现SA_TSP.f90，由于基于课上的SA程序，这里只贴出更改的关键部分。

排序

```
DO i = 1, n
  index_org(i) = 1
  DO j = 1, n
    IF (theta(i) < theta(j)) THEN
      index_org(i) = index_org(i) + 1
    END IF
  END DO
END DO
DO i = 1,N
    index(i+1) = index_org(i)+1
    END DO
index(1)=1
index(n+2)=1
```

增加随机性

```
!  Generate XP, the trial value of X. Note use of VM to choose XP.
      DO i = 1, n
        IF (i == h) THEN
          xp(i) = x(i) + vm(i)*(2.0*ranmar() - 1.0) + 5.0*ranmar() !后面这项是为了加强扰动
        ELSE
          xp(i) = x(i)
        END IF
```

结果和讨论

结果

在尝试多个矩阵后，这个算法是可行的。
其中，对于

$$d_{ij} = i,\ where\ i > j$$

这样的距离矩阵来讲，通过算法得到的结果是

$$x =< 1, 13, 4, 12, 3, 11, 7, 8, 6, 9, 5, 10, 2, 14, 1 >$$

cost为

$$cost = 56$$

针对问题的讨论

稍加修改和限制，我们还可以得到更多有实际意义也更困难的问题。

- 可以实际化这个问题，比如我从上海想要经过江苏的十三个省再回上海，怎么最省时间。
- 利用实际数据并且利用python处理得到距离矩阵，传入SA_TSP.f90里面
- 对cost的定义不单单的时间层次的，而是路途时间、票价的二元函数，这要求对目标函数进行修改。
- 来回的cost并不同，原来的无向图变成有向图。
- 在路径当中加入某些硬性要求，比如在去B城市之前一定要历经A城市，去了B城市后在一定时间内必须到达C城市，这或许要在代入解计算目标函数之前进行一定的条件判断。

针对解决方案的讨论

这个程序基于SA算法，但注意到，本次大作业通过不断产生随机数组x，然后对数组排序得到随机的index，显然是复杂化了程序。从头设计，vm并非必须的，而且也不需要保持接受率在50%左右。

理想的是，接受变量就是INTERGER，而产生下一个解通过交换任意两个位置的数字来实现。

总结

时间紧凑，很多想法来不及实现，比如应用并行计算、利用fortran和python的接口实现更广泛的功能。

在半学期的fortran学习当中，在不那么现代的语法当中，我也学到了模拟退火和蚁群这两种启发式算法，后面的并行计算也算让我产生了不少的兴趣，总的来讲，虽然我个人的能力并没有得到大的提升（更多原因是自己比较懒），我通过这门课也看见了更多有趣和有意义的领域。