**中山大学计算机学院**

**人工智能**

**本科生实验报告**

**（2022学年春季学期）**

课程名称：Artificial Intelligence

|  |  |  |  |
| --- | --- | --- | --- |
| 教学班级 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 实验题目

在TSPLIB(http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/，多个地址有备份；其他网站还可以找到有趣的art TSP和national TSP)中选一个大于100个城市数的TSP问题，使用模拟退火和遗传算法求解。

**模拟退火：**

采用多种邻域操作的局部搜索local search策略求解；

在局部搜索策略的基础上加入模拟退火simulated annealing策略，并比较两者的效果；

要求求得的解不超过最优值的10％，并提供可视化，观察路径的变化和交叉程度。

最优值查询： <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html>

**遗传算法：**

设计较好的交叉操作，并且引入多种局部搜索操作（可替换通常遗传算法的变异操作）

和之前的模拟退火算法（采用相同的局部搜索操作）进行比较

得出设计高效遗传算法的一些经验，并比较单点搜索和多点搜索的优缺点。

# 实验内容

1. 算法原理

**局部搜索：**

局部搜索算法是一种目标路径与问题解不相关，将考虑各种不关心路径（耗散）的算法。这一算法从单独的一个当前状态出发，通常只移动到与之相邻的状态，直到在邻域解中找到答案。

而在本次实验中，采用的局部搜索算法是爬山算法。爬山搜索算法是一种局部搜索算法，可在海拔/值不断增加的方向上连续移动以找到山峰或对该问题的最佳解决方案。当它达到一个峰值，没有邻居有更高的值时，它将终止。

**模拟退火：**

模拟退火算法采用类似于物理退火的过程，先在一个高温状态下（相当 于算法随机搜索,大概率接受劣解），然后逐渐退火，徐徐冷却（接受劣 解概率变小直至为零，相当于算法局部搜索），最终达到物理基态（相 当于算法找到最优解）。算法的本质是通过温度来控制算法接受劣解的 概率（劣向转移是脱出局部极小的核心机制）。算法模型如下所示：

*模拟退火算法可以分解为解空间、目标函数和初始解三部分。*

*(1) 初始化：初温T(充分大)，初始解状态S(是算法迭代的起点)，每个T值的迭代次数L*

*(2) 对k=1, …, L做第(3)至第6步：*

*(3) 产生新解S′*

*(4) 计算增量ΔT=C(S′)-C(S)，其中C(S)为评价函数*

*(5) 若ΔT<0则接受S′为新的当前解，否则以概率exp(-ΔT/T)接受S′作为新的当前解.*

*(6) 如果满足终止条件则输出当前解作为最优解，结束程序。*

*终止条件通常取为连续若干个新解都没有被接受时终止算法。*

*(7) T逐渐减少，且T->0，然后转第2步。*

**遗传算法：**

遗传算法（genetic algorithms，GA）：一类借鉴生物界自然选择和自然遗传机制的求解目标函数最优解的一种随机搜索算法, 非常适用于处理传统搜索方法难以解决的复杂和非线性优化问题。算法的关键步骤如下所示：

*1.选取初始种群。对种群中每个个体选用合适的编码。*

*2.解码。将每个个体的二进制的编码解码，成为目标函数定义域内对应的浮点数。*

*3.适应度计算。计算出各个个体对环境的适应度，即浮点数对应的函数值。同时，为了保证适应度大于零，将所有计算出来的函数值减去最小的函数值再加上一个很小的正数作为该个体的适应度。*

*4.物竞天择。用轮盘赌选择法筛选一部分个体出来，基因越优良的个体被选中的概率越大。*

*5.繁衍后代。在生物学上，产生后代时染色体上会发生交叉互换和基因突变。根据这两个特性我们可以如下操作：将上一步骤筛选出来的个体每一个都作一次父本，同时随机选择该筛选后群体中的某个个体作为母本。而亲本基因的组合方式又有多种。只要是在同一个规律下进行随机组合即可。除此之外，还可以设计一个求邻域解的函数来作为其变异函数。*

*6.重复进化。对产生子代后的种群重复2~5步骤，迭代到最大次数后，选取所有个体中函数值最优的作为最终结果。*

1. 关键代码展示（带注释）

评估函数：由于本问题为旅行商问题，因此估计算法好坏程度可以靠旅行的路径总长度来衡量。因此评估函数直接计算在当前状态下所有点的路径总长度即可。

def get\_dist(way): #计算距离总和 算评估函数

    ans=0

    for i in range(1,len(way)+1):

        if i==len(way):

            way1=way[0]-1

            way2=way[len(way)-1]-1

        else:

            way1=way[i-1]-1

            way2=way[i]-1

        x1=data[way1][1]

        y1=data[way1][2]

        x2=data[way2][1]

        y2=data[way2][2]

        ans+=math.sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2))

    return ans

邻域解产生函数：在参考了网上的一些博客后，发现在产生邻域解时采用较多交叉方式有利于最优解的搜索。因此在多次尝试后决定采用交换段（首尾换/中间换）、交换随机两点、反转段的方式。详细代码如下所示。

def get\_neighbur(way): #邻域解

    p=random.random()

    if p<0.5:#分成四段

        a,b,c=random.sample(range(1, len(way)-1), 3)

        point=sorted([a,b,c]) #升序

        a,b,c=point[0],point[1],point[2]

        if p<0.25: #交换中间两段

            ans=way[0:a]+way[b:c]+way[a:b]+way[c:len(way)]

        else: #交换首尾两段

            ans=way[c:len(way)]+way[a:b]+way[b:c]+way[0:a]

    elif p<0.75: #选择其中两个点

        i=random.randint(0,len(way)-2)

        j=random.randint(1,len(way)-1)

        while True:

            if i!=j: #如果不相同 则交换两个城市

                way[i],way[j]=way[j],way[i]

                ans=way[:]

                way[i],way[j]=way[j],way[i]

                break

            else:

                i=random.randint(0,len(way)-2)

                j=random.randint(1,len(way)-1)

    else:#分成三段

        a,b=random.sample(range(1, len(way)-1), 2)

        if a>b: a,b=b,a

        ans=way[0:a]+way[a:b][::-1]+way[b:len(way)] #中间反转

return ans

gif的实质是先画出每个线条的完整图片，再将这些图片保存成一个数组，将这个数组的图片依次保存到gif的文件中，并在每张图片之间加入延时。

过程图生成部分：

    fig1=plt.figure(1)

gif\_pic=[] #保存动态图

while sum<250000:

……

if sum%100==0:

            x=[]

            y=[]

            for i in way:

                x.append(data[i-1][1])

                y.append(data[i-1][2])

            x.append(data[0][1])

            y.append(data[0][2])

            plt.title('process')

            pic=plt.plot(x, y, marker = '.', color = 'blue',linewidth=1)

            gif\_pic.append(pic)

    #GIF保存

    ani=animation.ArtistAnimation(fig1,gif\_pic,interval=200, repeat\_delay=1000)

    ani.save("GA\_TSP.gif",writer='pillow')

    fig2=plt.figure(2) #展现最优距离的变化

    plt.title('cost change')

    x\_=[i for i in range(len(record\_weight))] #横轴

    plt.plot(x\_,record\_weight) #横轴纵轴

    plt.show()

以上三个函数是三个算法里共用的部分，下面展示遗传算法的关键部分。

遗传算法的特点之一就是种群的概念以及子代的产生方法。在初始种群的产生中，我先随机生成五十个个体，再选择其中最优的四个，从而提高算法的效率。

def first\_group(way): #生成若干个初始样例 选择其中最好的一部分并作为初始种群返回

    population=[]

    for i in range(0,50):

        temp\_way=way[:]

        random.shuffle(temp\_way)

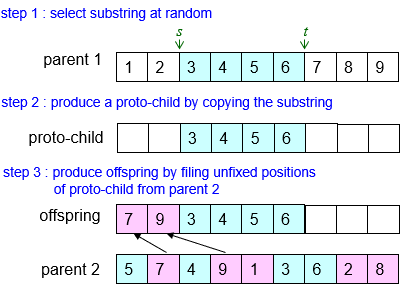
        if temp\_way not in population:

            population.append(temp\_way)

    ans=sorted(population,key=lambda x:get\_dist(x))

    return ans[:4] #返回前4个距离最小的

产生子代的方法有PMX\OX\PBX\OBX\CX等。而我在这里选择了OX，主要是因为容易实现且效果不错。算法思路如下：



具体代码如下：

def get\_kid(dad,mom):

    ans=[0 for i in range(len(dad))]

    a,b=random.sample(range(1, len(dad)-1), 2)

    if a>b: a,b=b,a

    for i in range(a,b): ans[i]=dad[i]

    i=j=0

    for i in range(0,len(mom)):

        if mom[i] not in dad[a:b]:

            ans[j]=mom[i]

            j+=1

        if j==a: j=b

    return ans

亲本的选择则是根据当前种群所有个体计算其被选中的概率（根据评估函数），并根据这个概率采用轮盘赌的方式选择。

    all\_weight,all\_p=get\_all\_p(population,len(population)) #群体中每个个体被选择的概率

    dad=random.choices(population,all\_p,k=1) #轮盘赌

    mom=random.choices(population,all\_p,k=1)

1. 创新点&优化（如果有）

1、领域解函数：比起单一的选择一种方式产生邻域解，在一个函数内写多种生成方法并根据概率值产生不同变异，能加速最优路径的搜索。

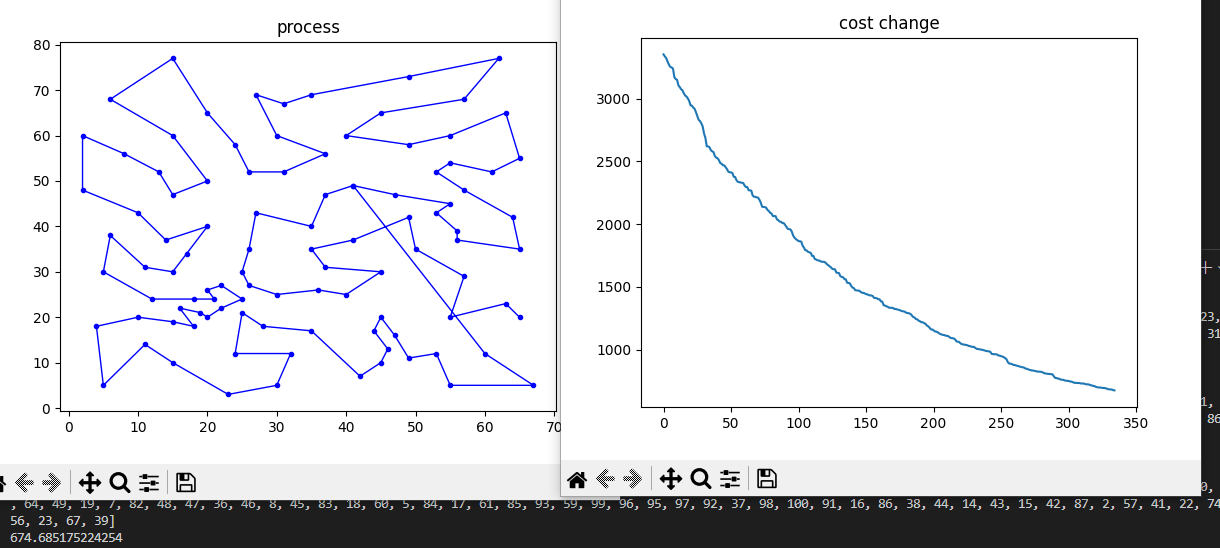
2、在生成路径变化过程图的同时生成一个每次被选择的路径长度的变化值，从而长度方面而非地图中线的密集程度展现算法的效果，比较直观。

# 实验结果及分析

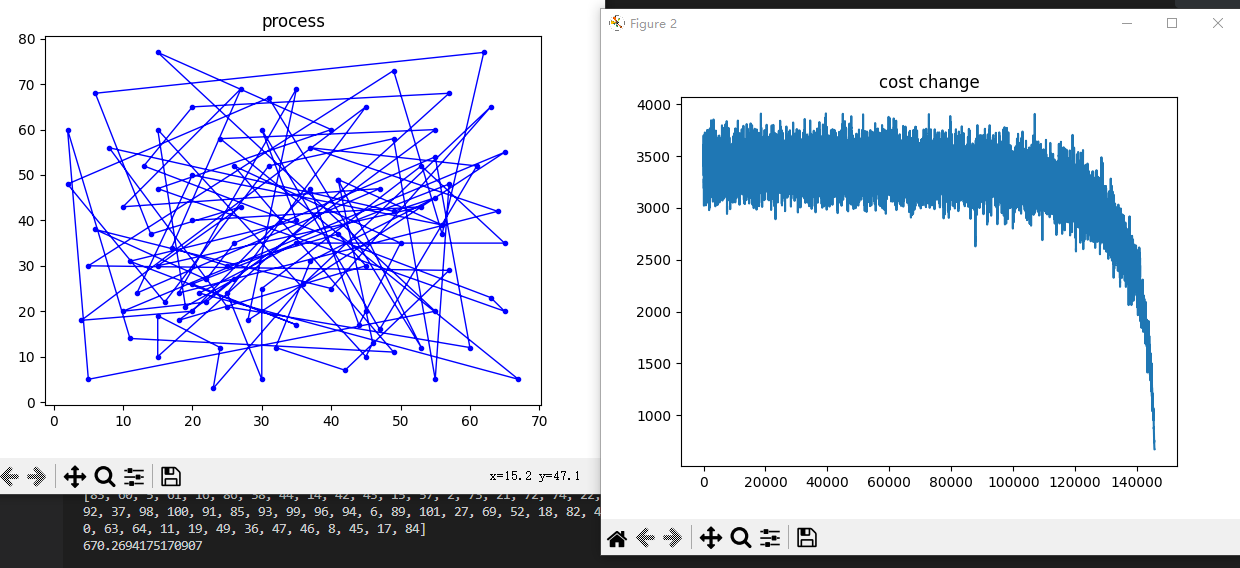
1. 实验结果展示示例（可图可表可文字，尽量可视化）

本次实验我选择了eil101的测试用例，其最优值是629。根据算法所求不能超过最优值的10%，这个临界值是691.9。下面是算法跑出来的最终结果

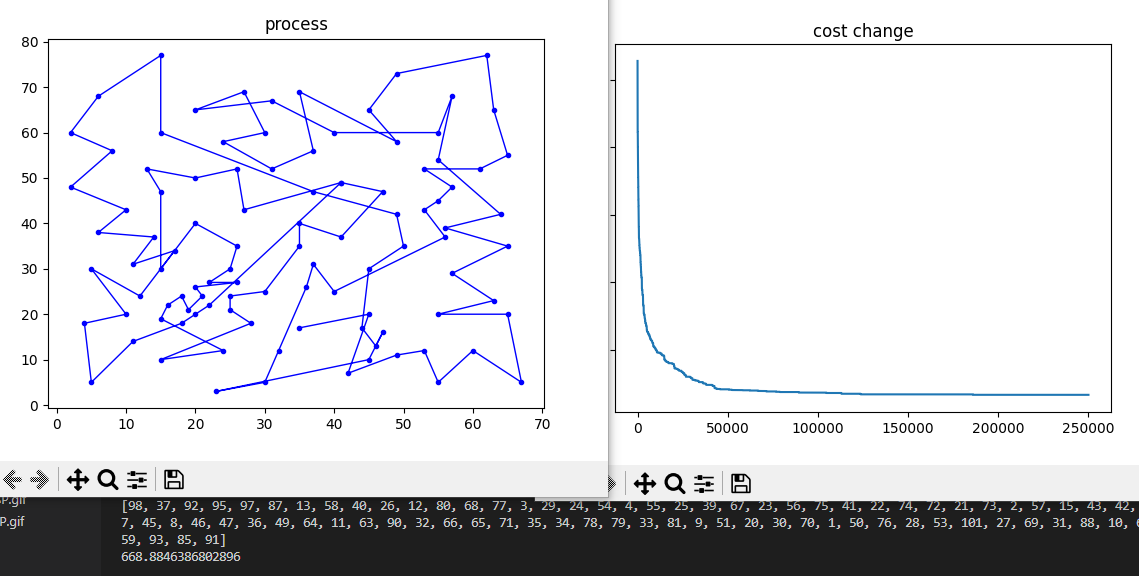
爬山算法：674.68517



模拟退火算法：670.269418



遗传算法：668.8846



**SA算法中采用多种邻域操作的局部搜索算法结果对比：**

|  |  |
| --- | --- |
| 邻域操作 | 路径长 |
| N1（交换首尾/中间两段） | 744.133839 |
| N2（交换点） | 782.110312 |
| N3（中段反转） | 738.137622 |
| get\_neighbur（综合以上三种方法） | 666.880993 |

可见第四种方法高贵的踏入了700内的行列，效果最好。

**模拟退火算法与局部搜索算法对比：**

由于我采用的是样例较少的样本，所以两个算法在运行时间上类似。然而平均路径长度总是模拟退火算法的长度小于局部搜索算法的长度。这是因为局部搜索算法容易陷入局部最优解，如本样例的爬山算法就经常卡在670~690之间，难以突破，模拟退火算法则可达到666甚至更小。

**模拟退火算法与遗传算法对比：**

在我看来，模拟退火算法与遗传算法的对比也可以视作是单点搜索和多点搜索的对比，但又有所不同。模拟退火作为单点搜索，具有陷入局部最优解的可能，但是这一算法具有依概率选择的内容，所以是一个在时间空间以及搜索结果上都比较优秀的单点搜索算法。而遗传算法虽然在搜索结果上可以与模拟退火算法相近，然而这一算法需要的搜索时间较长，且它保存种群的个体所需要的空间也比模拟退火算法要多。

**设计高效遗传算法的经验总结：**

1. 采用适当的遗传编码和适应值函数，在TSP问题中可以直接采用途经点为编码；
2. 设计较好的交叉操作、变异操作。交叉思路如PMX\OX\PBX\CX\SEX等，变异操作主要要体现一个随机性，变异的结果可以参考邻域解的产生；
3. 选取恰当的控制参数（位串长度、群体规模、交叉概率、变异概率），这些概率需要在实际跑代码的时候慢慢调整，最终找到一组使算法效率最高的参数。

**比较单点搜索和多点搜索的优缺点：**

在参考了模拟退火算法和遗传算法后，可以总结出二者的优缺点为：

单点搜索：优点：收敛速度快 缺点：容易陷入局部最优解。

多点搜索：优点：利于全局择优 缺点：收敛速度慢。

# 参考资料

**超算习堂-课时17-第九讲实验课**

[遗传算法（Python）\_python 遗传算法](https://blog.csdn.net/weixin_58427214/article/details/125860212)

[Python绘制TSP、VRP问题求解结果](https://blog.csdn.net/hfy1237/article/details/129368484)