面向科学问题求解的编程实践

课程实验　TSP 问题的近似算法的实现与比较

PB20151796 王 世 炜
大数据学院
CS1501.01 面向科学问题求解的编程实践
(春季, 2023)

中国科学技术大学

2023 年 6 月 20 日

目 录

1 实验概述

本次课程实验完成了对于 TSP 问题的一些近似算法的实现，包括爬山算法、模拟退火算法，遗传算法和蚁群算法，并对于自己生成的一组数据以及VLSI datasets中的 **XQF131-131 points**[2] 进行了测试，得到结果并对运行时间以及结果好坏等进行了对比分析，以判断算法的优劣。

2 实验环境

工具	版本/名称
Windows	Windows10 64 位
python	3.9.5
jupyter	1.0.0
matplotlib	3.5.1
numpy	1.23.0
pandas	1.4.2

3 前言

3.1 背景

旅行商问题（Travelling salesman problem, TSP）[3] 是组合优化中的一个 NP 困难问题，在运筹学和理论计算机科学中非常重要。问题内容为“给定一系列城市和每对城市之间的距离，求解访问每座城市一次并回到起始城市的最短回路。”

作为计算复杂性理论中一个典型的判定性问题，TSP 的一个版本是给定一个图和长度 L，要求回答图中是否存在比 L 短的回路（英语：circuit 或 tour）。该问题被划分为 NP 完全问题。已知 TSP 算法最坏情况下的时间复杂度随着城市数量的增多而成超多项式（可能是指数）级别增长。

问题在 1930 年首次被形式化，为优化中被研究得最深入的问题之一，许多优化方法都奉其为基准。尽管问题在计算上很困难，但已经有了大量的启发式和精确方法，因此可以完全求解城市数量上万的实例，并且甚至能在误差 1% 范围内估计上百万个城市的问题。

甚至纯粹形式的 TSP 都有若干应用，如企划、物流、芯片制造。稍作修改，就是 DNA 测序等许多领域的一个子问题。在这些应用中，“城市”的概念用来表示客户、焊接点或 DNA 片段，而“距离”的概念表示旅行时间或成本或 DNA 片段之间的相似性度量。TSP 还被应用在天文学中，减少在不同光源之间转动望远镜的时间。在许多应用场景中（如资源或时间窗口有限等等），可能会需要加入额外的约束条件。

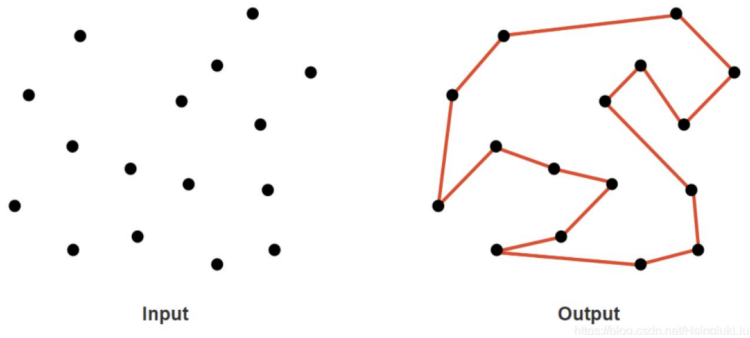


图 1: TSP 问题

4 问题分析

4.1 一些定义

定义 1 哈密顿图有关名词

哈密顿路径: 图的一条路，经过每个顶点恰好一次。

哈密顿圈: 在一条哈密顿路的基础上，再有一条边将其首尾连接，所构成的圈。注意，若有一个哈密顿圈，则移除其任一条边，皆可得到一条哈密顿路，但反之则不然，即给定一条哈密顿路，不一定能延伸成哈密顿圈，因为该路径的首尾两顶点之间，不一定有边相连。

哈密顿图: 有哈密顿圈的图。

定理 4.1 超过 2 个顶点的完全图是哈密顿图。n 阶无向完全图 K_n 上不同的（无向）哈密顿圈有 $\frac{(n-1)!}{2}$ 个；而若考虑方向，则有 $(n-1)!$ 个有向哈密顿圈。

4.2 将 TSP 问题转化为搜索问题

TSP 的任务是在一个图中找到一个哈密顿圈，如果是非完全图，我们可以先通过 Floyd 算法求出每两个点之间的最短路径，以最短路径作为新的边权进行重新建图，转化为完全图，那么由上面给出的定理，完全图一定可以找到哈密顿圈（本次实验仅考虑完全图）。

TSP 问题的解实际上是一个序列，序列中所有数字出现且仅出现一次。那么显然 TSP 问题的解的个数是 $n!$ 个，其中 n 是城市的个数，对于如此庞大的解空间，如果使用传统的搜索方法（深度优先、广度优先等），当 n 比较大时（例如 30! 的数量级是 10^{32}），需要的搜索时间是无法接受的，所以我们一般采用**启发式算法**来解决 TSP 问题。

5 算法原理及步骤

以下算法的伪代码参考 [5]

5.1 爬山算法（Hill-climbing Algorithm）

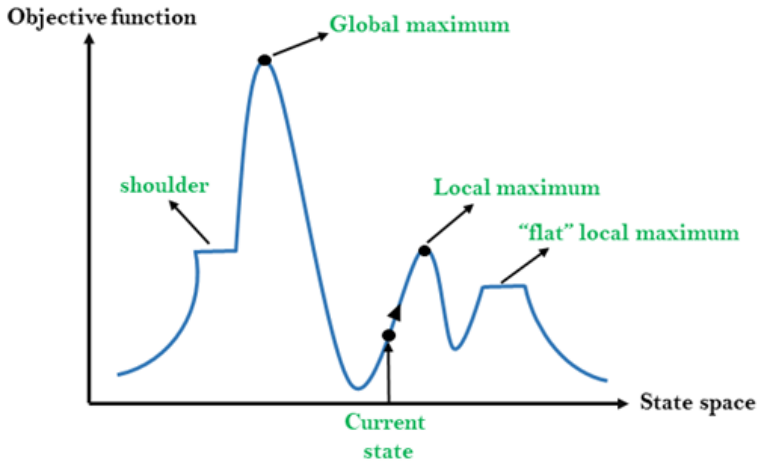


图 2: 爬山算法

爬山算法是一种优化算法，用于解决优化问题。它通过不断地迭代来找到最优解。我们首先随机初始化一个解，它是一个包含所有城市（以及添加的 m-1 个虚拟结点）序列。在每次迭代中，我们都会交换任意两个节点的顺序, 如果比当前的解优秀则采纳，重复 n 次（n 由用户设置）以找到局部最优解。

Algorithm 1 HILL-CLIMBING Algorithm

Input: problem
Output: a state that is a local maximum

```
current← MAKE-NODE(problem.INITIAL-STATE)
while True do
    neighbor← a highest-valued successor of current
    if neighbor.VALUE≤ current.VALUE then
        return currentSTATE
    end if
    current← neighbor
end while
```

5.2 模拟退火算法（Simulated-annealing Algorithm）

模拟退火算法（Simulated Annealing Algorithm）是一种全局优化算法，用于解决复杂的优化问题。它是一种对爬山算法的改进。它的灵感来源于金属退火的过程，其中金属被加热然后逐渐冷却，以减少其内部能量并增加其稳定性。

在模拟退火算法中，我们试图找到一个问题的全局最优解，而不仅仅是局部最优解。算法的基本思想是通过模拟金属退火的过程来跳出局部最优解，并朝着全局最优解的方向移动。

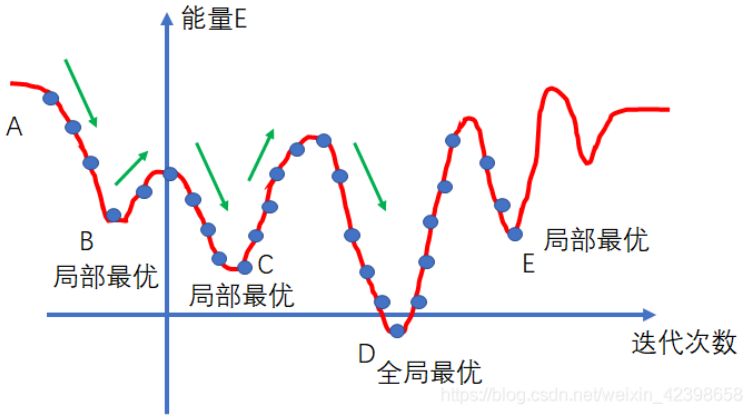


图 3: 模拟退火算法

模拟退火算法具有全局搜索能力和跳出局部最优解的能力。通过在搜索过程中接受一定程度的较差解，它有可能跳出局部最优解并找到全局最优解。然而，算法的性能高度依赖于参数的选择和停止条件的定义，需要根据具体问题进行调整。

下面给出算法的伪代码：

Algorithm 2 SIMULATED-ANNEALING Algorithm

Input: problem,a problem
schedule, a mapping from time to "temperature"
Output: a solution state

current← MAKE-NODE(problem.INITIAL-STATE)
for $t = 1$ to ∞ **do**
  $T \leftarrow$ schedule(t)
  if T = 0 **then**
    return current
  end if
  next←a randomly selected successor of current
  ΔE ←next.VALUE - current.VALUE
  if ΔE >0 **then**
    current←next
  else
    current←next only with probability $\exp^{\Delta E/T}$
  end if
end for

模拟退火算法存在的问题及我们的改进

- 虽然模拟退火算法有一定的跳出局部最优解的能力，但在面对本问题如此庞大的搜索空间时，也难以接近全局最优解，因此我们采用了随机重启退火算法，即多次用随机的初始值开始退火算法搜索，记录下最佳的一次的搜索结果。这样有效缓解了模拟退火算法陷入局部最优解的问题。

5.3 遗传算法（Genetic Algorithm）

遗传算法（GA）是一种基于生物进化原理的优化算法，它通过模拟自然界中生物的遗传、变异和选择过程，来寻找最优解。GA 的基本思想是将问题的解表示为一个个个体，每个个体都是一个 candidate solution，每个个体都有一定的基因组，基因组中包含了个体的特征和属性。

GA 的迭代过程可以分为三个步骤：选择、交叉和变异。在选择阶段，GA 会根据个体的适应度来选择一些个体，这些个体将作为下一代的父代。在交叉阶段，GA 会将一些父代个体进行交叉，生成新的子代个体。在变异阶段，GA 会对一些子代个体进行变异，生成新的子代个体。

GA 的优点是可以处理复杂的优化问题，可以处理多维度的决策变量，可以处理非线性的决策空间，可以处理离散的决策变量。GA 的缺点是计算量较大，需要大量的计算资源和时间。

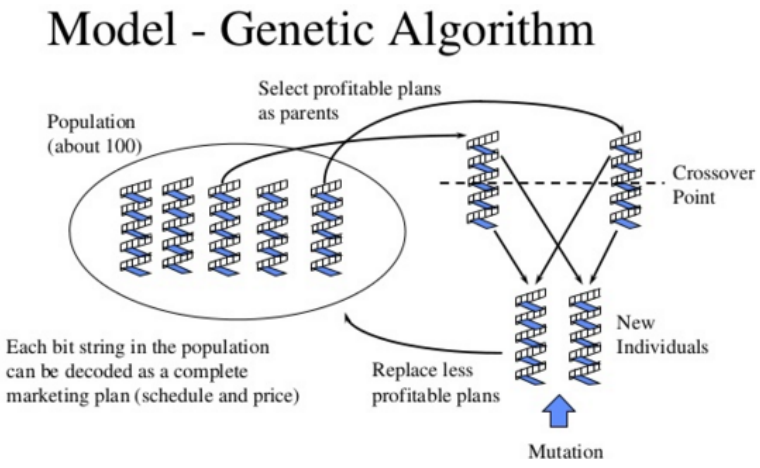


图 4: 遗传算法

下面详细讲解一下遗传算法的数学模型：

- GA 的基因组数学模型：我们采用了直观常见的编码方式：以遍历城市的次序排列进行编码。如码串 123456789 表示自城市 1 开始依次经过城市 2，3，4，5，6，7，8，9 最后返回到城市 1。这种编码方式保证了每个城市经过且只经过一次，并且保证在任何一个城市子集中不会形成完整的 TSP 回路。

- GA 的交叉算子模型：基于 TSP 问题的顺序编码，若采取简单的一点交叉或多点交叉策略，必然以极大的概率导致未能完全遍历所有城市的非法路径。解决这一约束的方法是对交叉操作做适当的修正，使其满足 TSP 的约束条件。我们使用了顺序交叉（OX）法。OX 是 Davis 于 1985 年进提出的 [4]。OX 操作能保留排列并融合不同排列的有序结构单元。在两个父代染色体中随机选择起始和结束位置，将父代染色体 1 该区域内的基因复制到子代 1 相同位置上，再在父代染色体 2 上将子代 1 中缺少的基因按照某种顺序填入。

- GA 的变异算子模型：针对 TSP 问题，常见的有 4 种变异技术。(1) 点位变异，变异仅以一定的概率（通常很小）对串的某些位做值的变异；(2) 逆转变异，在串中随机选择两点，再将这两点内的子串按反序插入到原来的位置中；(3) 对换变异，随机选择串中的两点，交换其值（码）；(4) 插入变异，从串中随机选择 1 个码，将此码插入随机选择的插入点中间。在本次实验中我们选择了 (2) 逆转变异，这样能保证基因有足够大的变异，能增大搜索范围。

- GA 选择算子模型：经过比较，我们最终选择了使用锦标赛选择策略。锦标赛选择（tournament selection），其基本思想是从当前群体中随机选择一定数量的个体，将其中适应度最大的个体保存到下一代。反复执行该过程，直到下一代个体数量达到预定的群体规模。这样的选择算子保证了优秀个体能不被丢失，同时效果较差的带有不同基因的个体也有几率保留，保证了算法不会过快收敛到局部最优解上。

下面给出算法的伪代码：

Algorithm 3 GENETIC Algorithm

```
Input: population,a set of individuals
  FITNESS-FN,a function that measures the fitness of an individual
Output: an individual
  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x←RANDOM-SELECTION(population,FITNESS-FN)
      y←RANDOM-SELECTION(population,FITNESS-FN)
      child←REPORDUCE(x,y)
      if small random probability then
        child←MUTATE(child)
      end if
      add child to new_population
    end for
  until some individual is fit enough,or enough time has elapsed
  return  the best individual in population,according to FITNESS-FN
```

REPRODUCE $(x, y, individuals)$ **returns an indivial**

```
n←LENGTH(x)
c←random number from 1 to n
APPEND(SUBSTRING(X,1,c),SUBSTRING(y,c+1,n))
```

5.4 蚁群算法（Ant Colony Algorithm）

蚁群算法（Ant Colony Optimization，简称 ACO）是一种基于群体智能的优化算法，灵感来源于蚂蚁在寻找食物过程中的行为。蚁群算法通过模拟蚂蚁在寻找最短路径时释放信息素的行为，来解决各种优化问题，特别是旅行商问题（Traveling Salesman Problem，TSP）和资源分配问题。

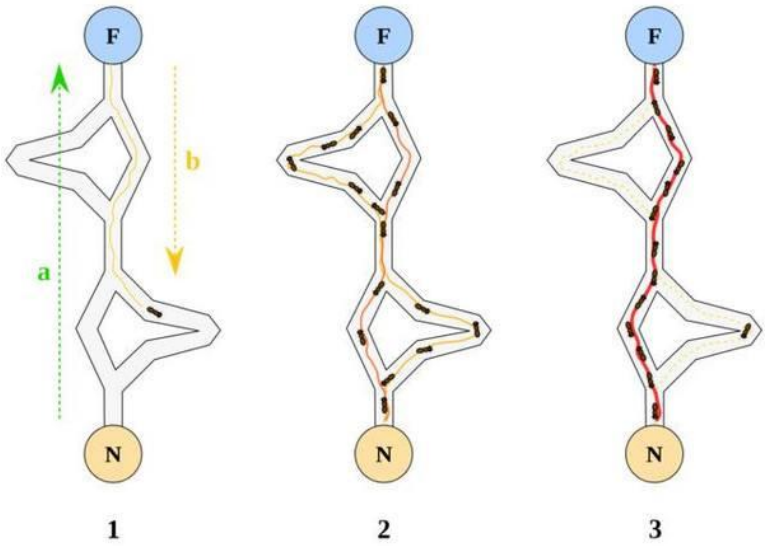


图 5: 蚁群算法

蚁群算法的基本原理是建立一个模拟的蚁群系统，其中包含多个虚拟蚂蚁。这些蚂蚁通过**释放和感知信息素**来进行通信。在蚁群算法中，信息素代表了路径上的某种度量，比如路径长度或者路径质量。蚂蚁根据信息素浓度选择路径，并在路径上释放信息素，增加该路径的信息素浓度。信息素的**挥发和更新过程**模拟了现实中信息素的衰减和更新。

蚂蚁系统（Ant System，AS）是最基本的 ACO 算法，是以 TSP 作为应用实例提出的。

AS 对于 TSP 的求解流程大致可分为两步：路径构建和信息素更新。

1. 路径构建

   伪随机比例选择规则（random proportional）。

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)]^\alpha [\eta(i,j)]^\beta}{\sum_{u \in J_k(i)} [\tau(i,j)]^\alpha [\eta(i,j)]^\beta} & if \quad j \in J_k(i) \\ 0 & otherwise \end{cases} \tag{1}$$

   对于每只蚂蚁 k，路径记忆向量 R^k 按照访问顺序记录了所有 k 已经经过的城市序号。设蚂蚁 k 当前所在城市为 i，则其选择城市 j 作为下一个访问对象的概率如上式。$J_k(i)$ 表示从城市 i 可以直接到达的、且又不在蚂蚁访问过的城市序列中的城市集合。$\eta(i,j)$ 是一个启发式信息，通常由 $\eta(i,j) = \frac{1}{d_{ij}}$ 直接计算。(i,j) 表示边 (i,j) 上的信息素量。

   长度越短、信息素浓度越大的路径被蚂蚁选择的概率越大。α 和 β 是两个预先设置的参数，用来控制启发式信息与信息素浓度作用的权重关系。当 $\alpha = 0$ 时，算法演变成传统的随机贪心算法，最邻近城市被选中的概率最大。当 $\beta = 0$ 时，蚂蚁完全只根据信息素浓度确定路径，算法将快速收敛，这样构建出的最优路径往往与实际目标有着较大的差异，算法的性能比较糟糕。

2. 信息素更新

   (a) 在算法初始化时，问题空间中所有的边上的信息素都被初始化为 t_0。

   (b) 算法迭代每一轮，问题空间中的所有路径上的信息素都会发生蒸发，我们为所有边上的信息素乘上一个小于 1 的常数。信息素蒸发是自然界本身固有的特征，在算法中能够帮助避免信息素的无限积累，使得算法可以快速丢弃之前构建过的较差的路径。

   (c) 蚂蚁根据自己构建的路径长度在它们本轮经过的边上释放信息素。蚂蚁构建的路径越短、释放的信息素就越多。一条边被蚂蚁爬过的次数越多、它所获得的信息素也越多。

   (d) 迭代（b），直至算法终止

下面给出算法的伪代码：

Algorithm 4 Ant Colony Algorithm

Input: a set of ants
Output: a solution

```
for each edge do
    set initial pheromone value τ0.
end for
while not stop do
    for each ant k do
        randomly choose an initial city.
        for i=1 to n do
            choose next city j with prob. given by equation1
        end for
    end for
    compute the length Ck of the tour constructed by the kth ant.
    for each edge do
        update the pheromone value
    end for
end while
return best solution
```

6 结果

6.1 20 个随机生成城市

这一部分，我们使用随机生成的 20 个城市（坐标范围 [0,100)，搜索空间大小为 $20! \approx 2 \times 10^{18}$，为了保证可复现性，之后所有需要随机的地方都设置了随机种子）测试算法：

首先画出随机生成的城市的坐标图：

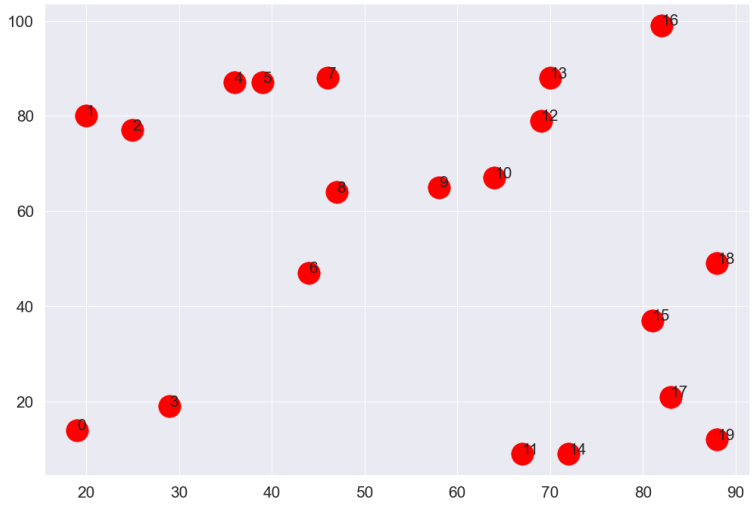
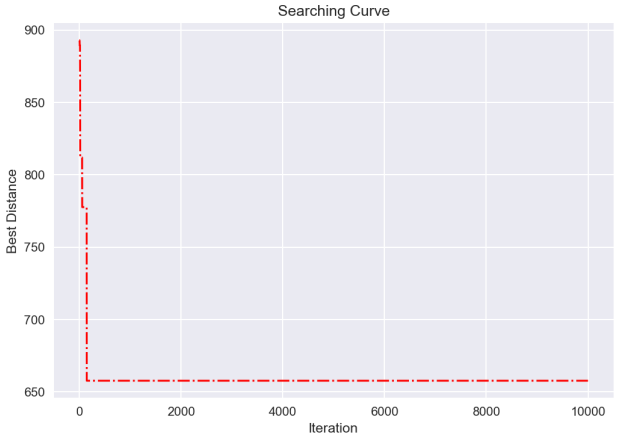


图 6: 随机生成的 20 个城市

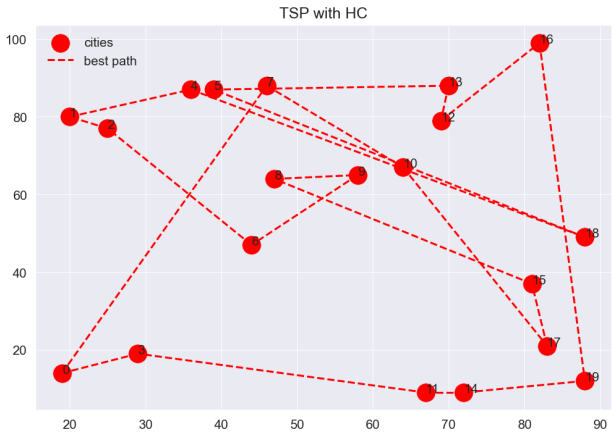
6.1.1 爬山算法

由于爬山算法的能力不足，所以我们采用的是随机重启爬山算法，即运行 100 次爬山算法，每次运行给不同的随机数种子（即不同的初始状态），保留 100 次爬山算法中的最佳路径和最佳距离，得到了如下

结果：



(a) 随机重启爬山算法距离下降曲线



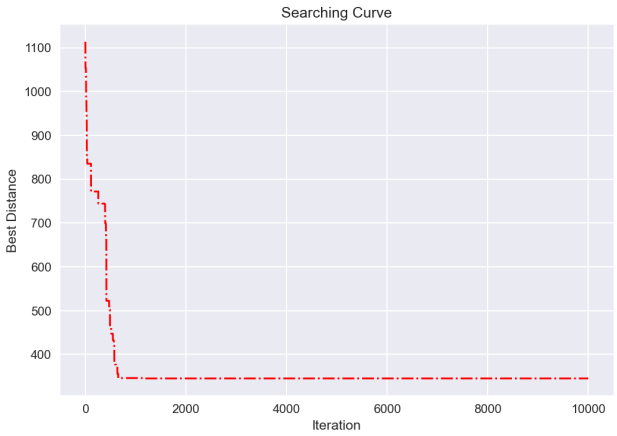
(b) 随机重启爬山算法最佳路径

随机重启爬山算法耗费 9.3s 给出的最佳距离是 657.6935，这并不是实际的最优解，从距离曲线图我们可以看出，爬山算法很容易陷入局部最优，对于较大的 TSP 问题来说，爬山算法能力不足。

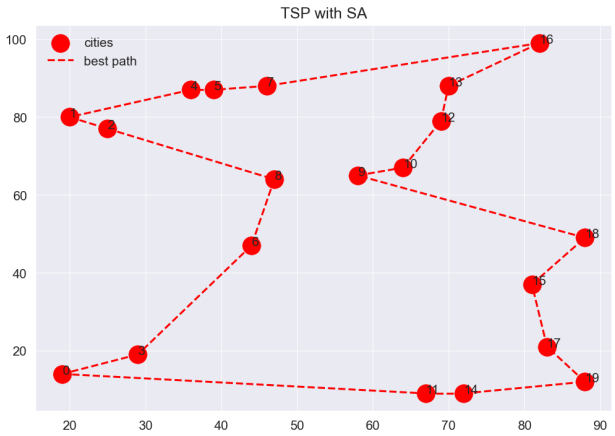
误差率：$\alpha = \frac{657.6935-345.2809}{345.2809} = 0.9048$, 这是一个较差的结果。

6.1.2 模拟退火算法

同样的，我们采用随机重启模拟退火算法，得到如下结果：



(a) 随机重启模拟退火算法距离下降曲线

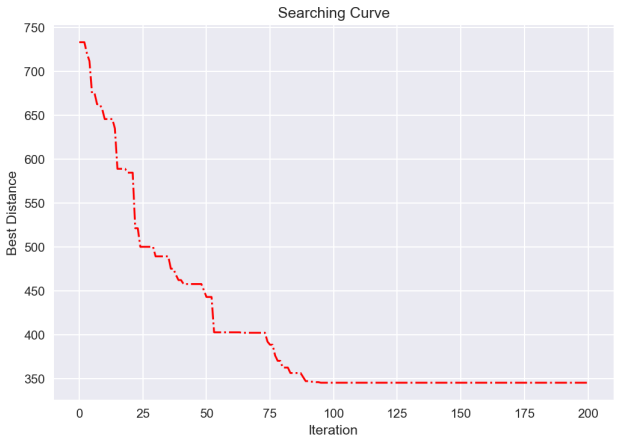


(b) 随机重启模拟退火算法最佳路径

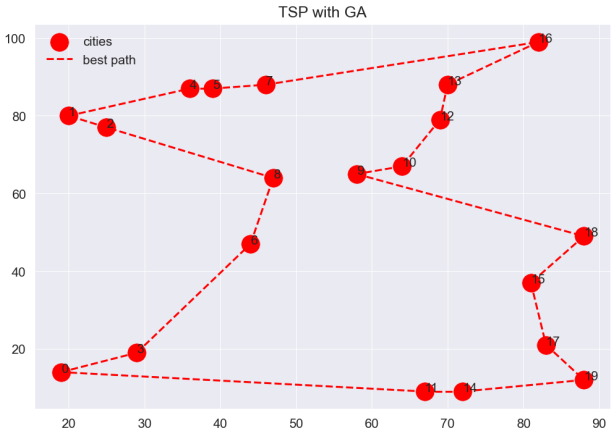
随机重启模拟退火算法耗费 21.3s 给出的最佳距离是 345.2809，这就是该问题的最优解，可见随机重启模拟退火算法是可以处理这个数量级的搜索问题的。

6.1.3 遗传算法

在适当的参数下我们给出遗传算法的结果：



(a) 遗传算法距离下降曲线

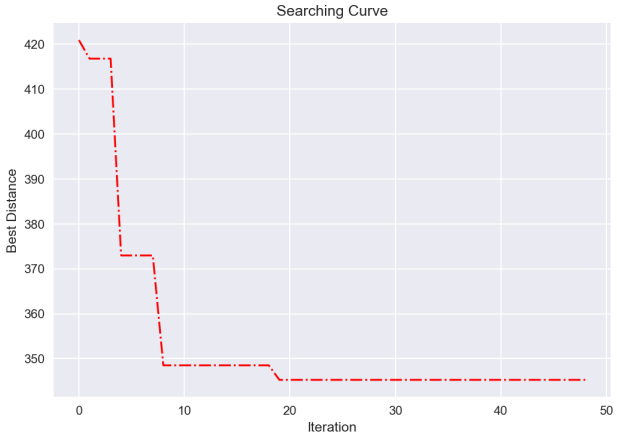


(b) 遗传算法最佳路径

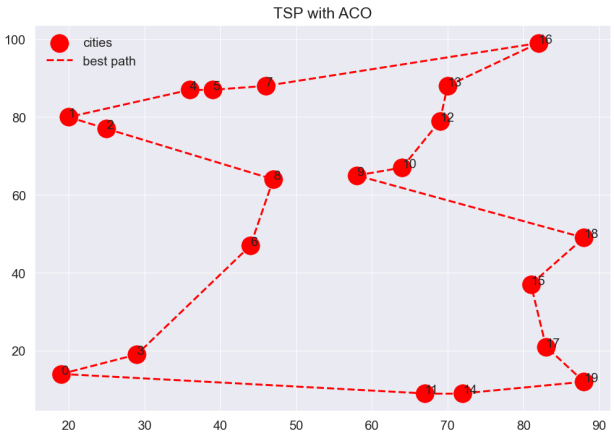
遗传算法耗费 1.9s 给出的最佳距离是 345.2809，这就是该问题的最优解，可见遗传算法处理这个数量级的搜索问题很轻松，并且相对于前两个算法而言，遗传算法只用了 100 轮左右便得到了最优解，耗时大大减少。

6.1.4 蚁群算法

在适当的参数下我们给出蚁群算法的结果：



(a) 蚁群算法距离下降曲线



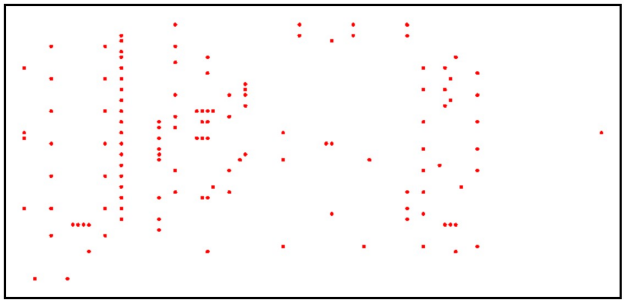
(b) 蚁群算法最佳路径

蚁群算法耗费 1.0s 给出的最佳距离是 345.2809，这就是该问题的最优解，可见蚁群算法处理这个数量级的搜索问题很轻松，蚁群算法只用了 20 轮左右便得到了最优解，比遗传算法更加有效。

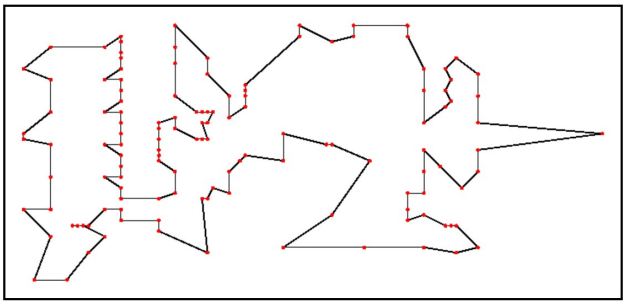
6.2 XQF131-131 points

这一部分，我们使用 XQF131-131 points（131 个城市，搜索空间大小为 $131! \approx 8 \times 10^{221}$），这是一个搜索空间极为庞大的问题，我们用此问题来测试算法的性能：

首先给出 XQF131-131 points 的坐标图以及其真正的最佳路径：



(a)XQF131-131 points

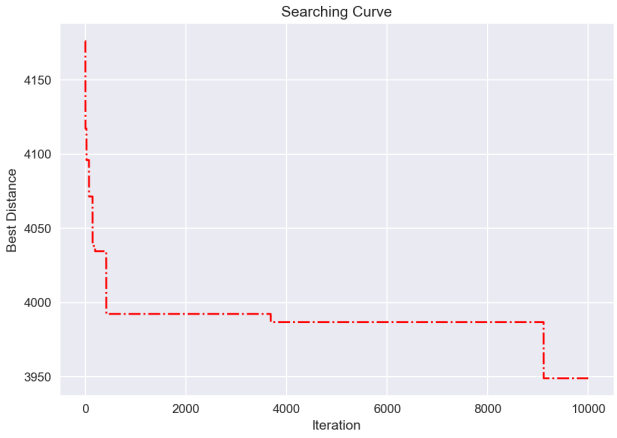


(b) 最佳路径

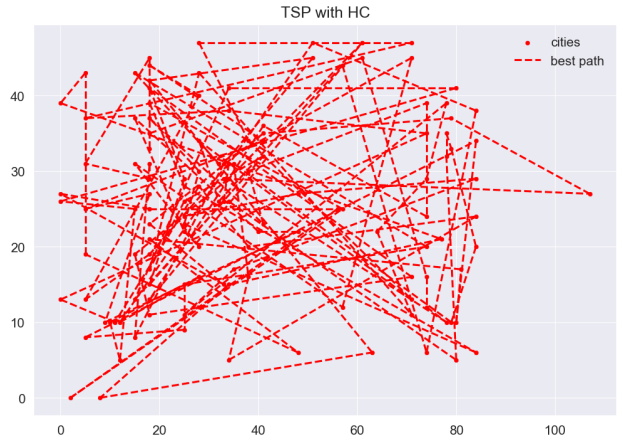
最佳路径的总距离为 564。

6.2.1 爬山算法

随机重启爬山算法得到了如下结果：



(a) 随机重启爬山算法距离下降曲线



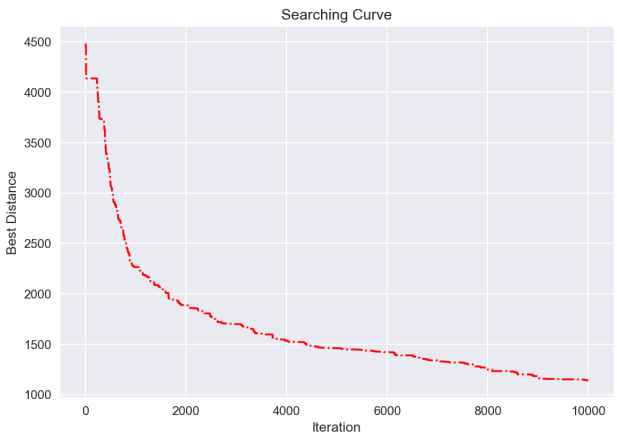
(b) 随机重启爬山算法最佳路径

随机重启爬山算法给出的最佳距离是 3948.9585，这显然不是最优解，并且可以说得到了非常差的结果，几乎没有起到搜索作用。

误差率：$\alpha = \frac{3948.9585-564}{564} = 6.0017$, 这是一个非常差的结果。

6.2.2 模拟退火算法

同样的，我们采用随机重启模拟退火算法，得到如下结果：



(a) 随机重启模拟退火算法距离下降曲线

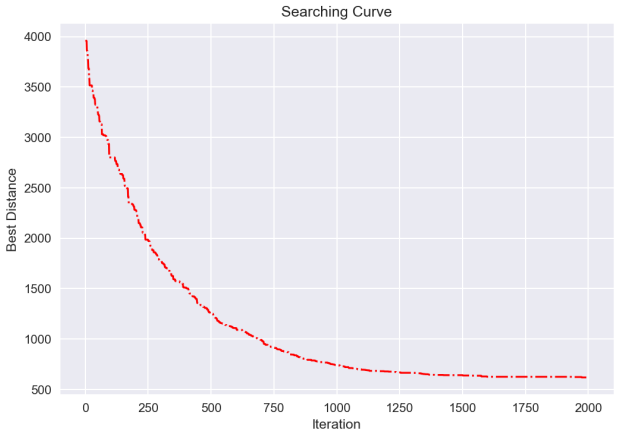
(b) 随机重启模拟退火算法最佳路径

随机重启模拟退火算法给出的最佳距离是 1140.6368，这个结果明显优于随机重启爬山算法，给出的最佳路径也逐渐向着最优解靠近，但是离最优解还有不少距离，从下降曲线中我们可以发现距离仍在下降，说明继续加大迭代次数，有可能找到更好的解，但是由于运行时间已经明显超出承受范围，我们认定该数量级问题已经超出该算法的能力。

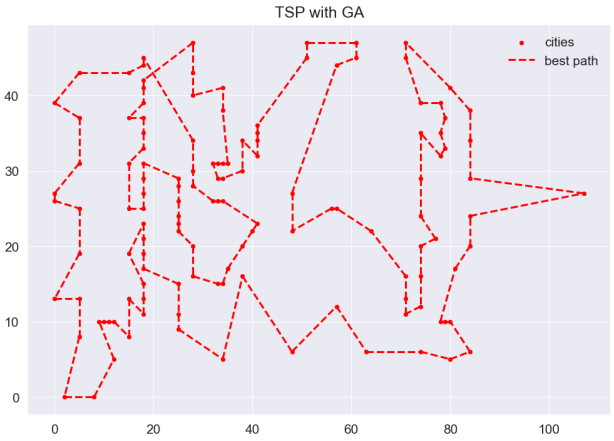
误差率：$\alpha = \frac{1140.6368-564}{564} = 1.0224$, 这是一个较差的结果。

6.2.3 遗传算法

在适当的参数下我们给出遗传算法的结果：



(a) 遗传算法距离下降曲线



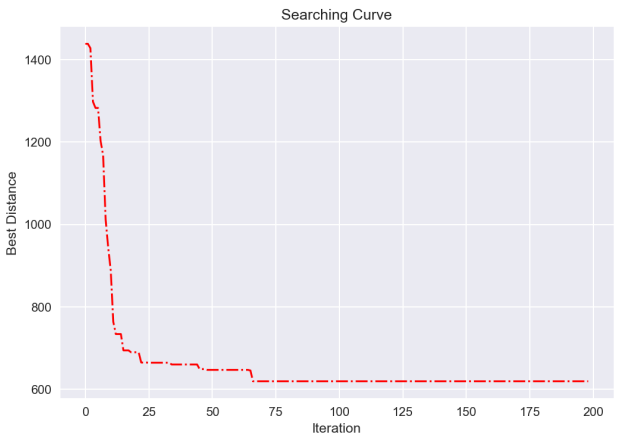
(b) 遗传算法最佳路径

遗传算法给出的最佳距离是 666.3494，遗传算法给出的解已经很接近最优解了，在几分钟的运行时间内给出了较为不错的解。从图中我们可以看出遗传算法已经收敛，即陷入局部最优，事实上如果我们对其中的某些参数进行调整，或者采用多种群遗传算法来扩大搜索范围，可以找到更好的解，但是也会带来更长的运行时间和资源消耗。

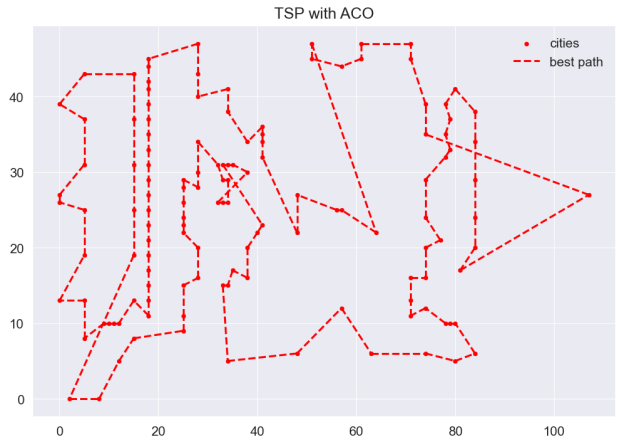
误差率：$\alpha = \frac{666.3494-564}{564} = 0.1815$, 这是一个比较好的结果。

6.2.4 蚁群算法

在适当的参数下我们给出蚁群算法的结果：



(a) 蚁群算法距离下降曲线



(b) 蚁群算法最佳路径

蚁群算法给出的最佳距离是 619.6321，蚁群算法给出的解相当接近最优解，在两分钟内的运行时间内给出了非常不错的解。从图中我们可以看出蚁群算法已经收敛，即陷入局部最优，同样的，如果我们对其中的某些参数进行调整，或者采用多种群遗传算法来扩大搜索范围，可以找到更好的解，但是也会带来更长的运行时间和资源消耗。

误差率：$\alpha = \frac{619.6321-564}{564} = 0.0986$, 这是一个相当好的结果。

7 结论

通过上面的结果我们可以得出以下结论：

- **爬山算法**：爬山算法在两个数据集上的表现都不尽如人意，这是因为该算法的能力较弱。它是一种局部搜索算法，通过在解空间中进行局部移动来寻找更好的解。然而，这种算法容易陷入局部最优解而无法找到全局最优解。在 TSP 问题中，爬山算法可能陷入一个局部最优解，导致不能找到整个问题的最优解。

- **模拟退火算法**：模拟退火算法在第一个数据集上表现较好，但时间较长，在第二个数据集上效果优于爬山算法，但仍是较差的结果。模拟退火算法是一种元启发式算法，通过模拟固体退火过程来搜索解空间。它可以通过接受差解的概率来跳出局部最优解，从而更有可能找到全局最优解。然而，模拟退火算法的性能高度依赖于参数的选择和退火策略的设计。可能在第二个数据集上，参数选择不当或者退火策略不够有效，导致其效果不如预期，还有可能这样的策略在大搜索空间内并不奏效。

- **遗传算法**：遗传算法在两个数据集上的结果都不错，但时间上不如蚁群算法，并且在第二个数据集上效果不如蚁群算法。遗传算法是一种通过模拟生物进化过程进行搜索的算法。它使用基因编码和选择、交叉和变异等操作来生成新的解，并逐步优化解的质量。遗传算法在 TSP 问题中通常具有较好的性能，因为它能够在解空间中进行广泛的搜索并快速收敛到较好的解。然而，由于遗传算法的操作涉及到种群的维护和进化过程，因此在大规模问题上可能需要更多的时间和计算资源。第二个数据集可能更加复杂，导致遗传算法的性能稍逊于蚁群算法。

- **蚁群算法**：蚁群算法在两个数据集上的效果最好。蚁群算法是通过模拟蚂蚁在寻找食物过程中释放信息素的行为来进行搜索的算法。蚂蚁在搜索路径时会根据信息素浓度的程度选择路径，从而逐渐形成优化的解。蚁群算法在 TSP 问题中通常能够找到较好的解，特别是在处理大规模问题时具有优势。它通过并行搜索和信息素的更新策略，能够有效地探索解空间并跳出局部最优解。

8 问题

事实上，解决 TSP 问题还有很多其他的算法，如分支定界法、基于 hopfield 神经网络的方法等，此外，我们还可以将上述算法进行融合，以得到更加高效，解的质量更好的一些算法。

参考文献

[1] 数学模型（第三版）/谭永基，蔡志杰编著，上海：复旦大学出版社，2019.8（复旦博学. 数学系列）

[2] XQF131-131 point

[3] 旅行推销员问题

[4] Davis, L. (1985, August). Applying adaptive algorithms to epistatic domains. In IJCAI (Vol. 85, pp. 162-164).

[5] 罗素, 诺维格, 殷建平. 人工智能: 一种现代的方法 [M]. 清华大学出版社, 2013.

附录 A 随机生成的 20 个城市下四种算法得到的路线及距离

算法	路线	路线距离
爬山算法	[15, 8, 9, 6, 2, 1, 4, 18, 5, 13, 12, 16, 19, 14, 11, 3, 0, 7, 10, 17]	657.6935
模拟退火算法	[18, 9, 10, 12, 13, 16, 7, 5, 4, 1, 2, 8, 6, 3, 0, 11, 14, 19, 17, 15]	345.2809
遗传算法	[9, 10, 12, 13, 16, 7, 5, 4, 1, 2, 8, 6, 3, 0, 11, 14, 19, 17, 15, 18]	345.2809
蚁群算法	[9, 18, 15, 17, 19, 14, 11, 0, 3, 6, 8, 2, 1, 4, 5, 7, 16, 13, 12, 10]	345.2809

表 1: 四种算法得到的路线及距离