对旅行商问题多重解答的探讨

摘要**:**

本文主要对典型的旅行商 (Traveling salesman problem) 问题进行了分析，并尝试 使用蚁群算法、模拟退火算法以及遗传算法多种算法用 matlab 对其实现了解答。

#### 由于 TSP 是一个典型的组合优化问题，而 lingo 软件的长处就在于求解优化类 问题，并且具有输入模型简练直观、运行速度快、引入了集合的概念等特点，因此 我们尝试使用 lingo 软件描述出了 tsp 问题，并对其进行了求解。

为了将 4 种方法进行比对分析，我们对每种算法采用了相同的城市坐标数据。

经过分析，我们得到以下结论：

1.lingo 是 4 种方法里面对 TSP 问题适应性最好的。它仅需输入数据、约束条件 以及目标函数，即可自动选择合适的求解器。对于 TSP 问题，它采用了 B—and—B 求解器，即分枝定界算法。lingo 求得的目标函数值最为精确。而 lingo 相比 matlab 的缺陷在于，它的运行时间较长，显著多于 matlab 的算法，在运行效率上有着较大 的劣势。而且它不能实现最短路径的可视化，在描述运算结果方面不如 matlab 直 观。

#### 2.lingo 的求解稳定性是 4 者当中最优的。多次重复运行同样的代码，所得的计 算结果以及运行所用时间均无明显变化。

3. 不论采取何种算法，matlab 相比 lingo，都有着可以实现城市位置以及具体线 路的可视化的优势，这使得计算结果更加直观。4. 模拟退火算法是 matlab3 种算法 计算效率最高的，计算结果也略优于其他几个算法，这证明了模拟退火算法的可靠 性与有效性。

5. 蚁群算法的运行速度仅次于模拟退火算法，但是它的解的质量是最低的。

关键词：旅行商问题 模拟退火算法 蚁群算法 遗传算法 **lingo** 编程

一、问题重述

题目要求：已知一些点的位置以及它们之间相互的距离，求遍历所有点、且不 重复经过任何一个点的距离最短的路线。

这本质上是一个旅行商问题 (Traveling salesman problem)。TSP 问题如下：一名 商品推销员要去若干个城市推销商品，该推销员从一个城市出发，需要经过所有城 市后，回到出发地。问应当如何选择行进路线，以使总的行程最短。

#### 可见，两者的数学模型是一模一样的。我们只需对其中任意一个进行求解，另 外一个也会得到解答。故下面我们直接对 TSP 问题进行分析与解答。

二**.** 对问题的初步分析

旅行商问题是组合优化中的一个非线性规划 (NP) 困难问题，该问题的求解与 计算均有不低的难度。

若题中的城市数量较少，我们可以采用枚举的方式，穷举出所有可能的路径并 一一计算相应的距离，之后即可进行比对选取总距离最短的路径。

然而，随着城市数目的增加，会产生计算量爆炸的情形，导致计算机难以对其 进行求解。因此我们需要寻找更加高效、有针对性的解题方式。

查询资料可知：常用的算法主要为近似算法或启发式算法，主要有遗传算法、

模拟退火法、蚁群算法、禁忌搜索算法、贪婪算法和神经网络等。本文我们主要尝 试使用 matlab 以蚁群算法、模拟退火算法、遗传算法以及 lingo 对 TSP 问题进行求 解。最后运行的结果表明，这四种方式确实均是可以实现的。四种求解方法的优劣 势均在下文中得到了具体的探讨。

三、**MATLAB** 使用蚁群算法解 **TSP** 问题

**3.1** 蚁群算法的原理

#### 蚁群算法是一种基于信息正反馈原理的算法。它采用了仿生的形式，通过对蚂 蚁群体选择觅食路径的方法的学习，实现了与现实问题的结合，并在许多问题的求 解上取得了显著的效果。

自然界中的蚂蚁视觉十分不灵敏，但是它却总是能在缺乏提示的条件下自动选 择到从食物源到食物储存地的最短路径。这是因为每一只蚂蚁在寻找食物的路途 中，总是会在路上释放信息素，使得相邻的一定范围内的其他蚂蚁可以较为容易地 察觉到这条路径的存在。当经过某一条路径的蚂蚁数量增多时，该路径上的信息素 浓度会相应地上升，从而吸引更多蚂蚁来走这条路，进而促使该路径信息素浓度不 断上升，形成一个正反馈的机制。

在现实中，信息素浓度在没有其他蚂蚁持续供应的情况下，会由于各自外界原 因 (如雨水的冲洗) 而减弱。在将蚁群的行为进行仿生的过程中，也应该考虑到这 个因素。

**3.2** 蚁群算法 **TSP** 模型的建立

#### 假设蚂蚁会且仅会在每个城市之间的连线的线路上经过，蚂蚁总数量为 m，城 市数量为 n，城市 i 与城市 j 之间的距离为 *dij* ,t 时刻城市 i 与城市 j 连接路径上信 息素浓度为 *τij* (*t*). 初始时刻，各蚂蚁均匀分布在不同的城市，且各城市路径上的信

息素浓度相同，均为 *τij* 0 = *τ*0。之后蚂蚁会按照一定的概率选择线路，设 *pk* 为 t

*ij*

#### 时刻蚂蚁 k 从城市 i 移动到城市 j 的概率。而蚂蚁选择线路主要受到以下 2 个方面

的影响，一是访问某城市的期望值，二是线路上的信息素浓度。我们令：



 ∑

[*τij* (*t*)]*α ∗* [*ηij* (*t*)]*β*

*α*

*k*

*β , j ∈ allowk*

*pij* =

*k*

*s∈allow*

[*τis*(*t*)]

* [*ηis*(*t*)]

 0*, j ∈*/ *allowk*

其中，*ηij* (*t*) 为启发函数，表示蚂蚁从城市 i 转移到城市 j 的期望程度；*allowk* 为蚂 蚁 k 待访问的城市集合。起初，*allowk* 中有 n-1 个元素，随着时间推移，它会逐渐 减小，直至减为 0；*α* 为信息素因子，它的值的大小与信息素强度影响正相关；*β* 为启发函数重要程度因子，它的值的大小与信息素强度影响正相关。

#### 由 3.1 的原理知，信息素浓度会一定程度上自发降低。令 *ρ*(0 *< ρ <* 1) 表示信 息素的挥发程度。则当蚂蚁将所有的城市走完，信息素浓度为

 *τij* (*t* + 1) = (1 *− ρ*) *∗ τij* (*t*) + ∆*τij*



#### 

∆*τij* =

#### 

*m*

∑ *k*

∆*τ*

*ij*

*k*=*q*

其中，∆*τk* 为第 k 只蚂蚁在城市 i 与 j 连接路径上面释放信息素而使得其增加的浓

*ij*

#### 度大小；∆*τij* 为所有蚂蚁在城市 i 与 j 连接路径上面释放信息素而使得其增加的浓 度大小

而

 *Q*

∆*τk* =  *k*

*ij*

####  0

Q 为信息素常数，表示蚂蚁经过一次所释放的信息素总量；*k* 为第 k 只蚂蚁经过路 径的总长度。

**3.3matlab** 实现蚁群算法 **-TSP** 模型的具体操作

matlab 对蚁群算法的具体实现方式可以分为以下五步：

1. 准备数据：

首先清空环境变量，并开始计时。其次，使用 xlsread 函数读入所有城市的坐 标值并将其保存在 citys 矩阵中。

2. 计算城市之间的距离：

#### 使用 for 函数求出每 2 个城市之间的距离，并记录在矩阵 D 中。为了保证启发 函数的分母不为 0，我们将 D 的对角线上的所有元素由原先的 0 修改为足够小的正 数：1e-4。

3. 初始化参数： 首先，我们将蚂蚁数量、信息素重要程度因子、启发函数重要程度因子与信息

素挥发因子等参数赋予一个合适的值。然后，我们将信息素矩阵、路径记录表、最 大迭代次数等参数初始化为 0 或 1 或其他便于存储的数据。

4. 迭代寻找最佳路径。这是整个算法中的最核心的一部分，是决定整个程序优 劣的关键因素。

设置一个 while 循环，当迭代次数等于最大次数时，则跳出循环，否则，依次 进行以下几步。首先，用 randperm 函数随机产生各个蚂蚁的起点城市，构造 m\*1 的列向量 start 记录各个蚂蚁的起点位置，用 *cityindex* 表示城市列表。然后，依次 按照每只蚂蚁和每个城市进行循环，用 tabu 矩阵表示禁忌表，即每一次循环时都 将前一次经过的城市编号记录上去。而剩下的城市序号则记录在 allow 矩阵中。可 知，随着迭代次数的增加，tabu 矩阵中实际表示的城市会越来越多，allow 矩阵中 实际表示的城市会越来越少，直至循环终止 (即每一只蚂蚁都经过了所有城市)。根

#### 据 3.2 提到的函数求解蚂蚁访问下一个城市的概率，再使用轮盘赌法决定每一只蚂 蚁的下一个访问城市。

当该 2 个循环结束后，求得每一次 while 循环的所有路径长度的平均值。使用 Length 列向量记录每一只蚂蚁经过的路径之和，并比较 while 循环前一次所得到的 结果，得到当前最短路径的城市序号的顺序、当前最短路径的距离。

为了更进一步地贴合实际并且提高算法质量，我们需要更新信息素浓度。设置 两个 for 循环，依次对每只蚂蚁与对每个城市进行循环，每一次都使用 3.2 提到的 信息素浓度求解函数更新相应节点的信息素浓度。随后将路径记录表清空。

信息素更新完之后，使 while 迭代次数 iter 数量增加 1，并判断是否达到最大循 环次数。若已经达到，则跳出 while 循环，否则进行下一次循环。

#### 5. 结果显示： 首先，输出最短距离、最短路径的城市序号、收敛迭代次数以及程序执行时

间。随后用 plot 函数作图，表现出蚁群算法的最优化路径与算法收敛轨迹。 蚁群算法应用到 TSP 问题中的代码的思路的流程框图如下：

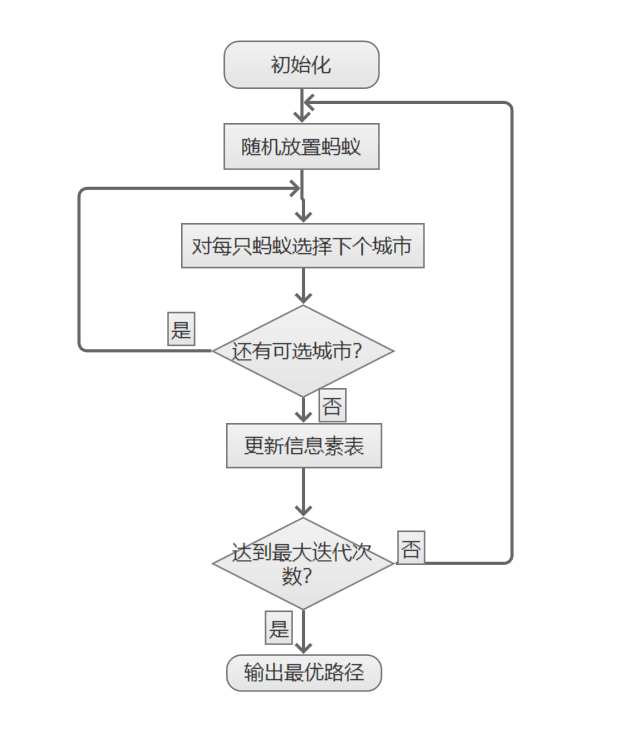


图 1: 蚁群算法结构程序框图

**3.4** 对蚁群算法求解结果的展示与分析

输出结果如下： 最短距离:7677.6608

最短路径:46 44 34 35 36 39 40 38 37 48 24 5 15 6 4 25 12 28 27

26 47 13 14 52 11 51 33 43 10 9 8 41 19 45 32 49 1 22 31 18 3 17

21 42 7 2 30 29 50 20 23 16 46

收敛迭代次数:73 程序执行时间:20.348 秒 蚁群算法最优化路径如下：

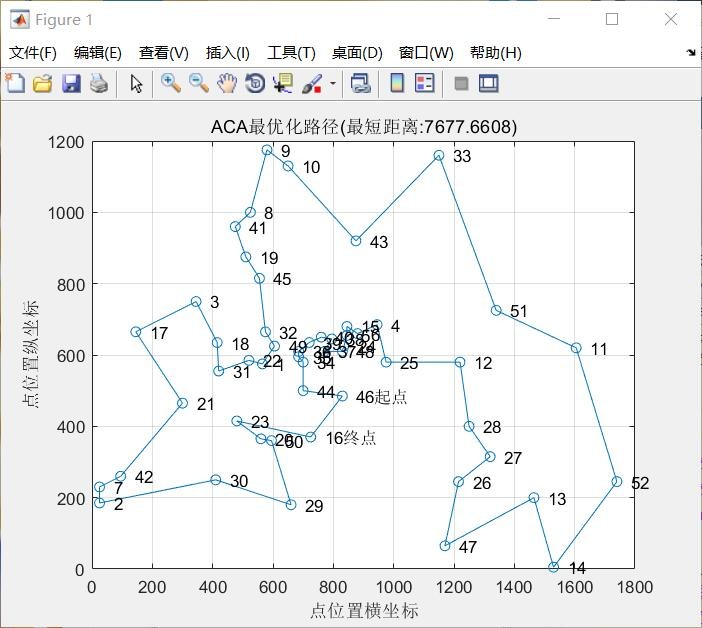


图 2: 蚁群算法最优化路径

#### 蚁群算法收敛轨迹如下：

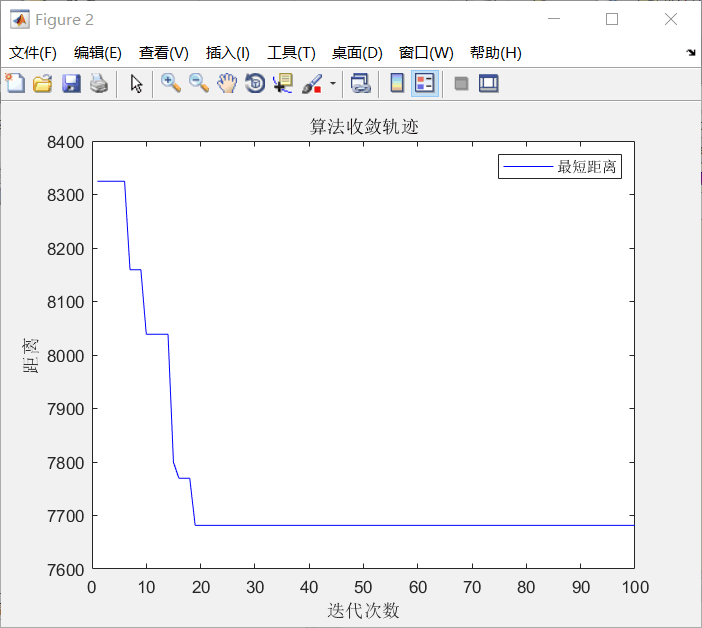


图 3: 蚁群算法收敛轨迹

#### 结果分析：程序执行时间为 20.348 秒，在可以接受的时间范围内即完成了如此 复杂的计算，证明了蚁群算法的有效性。

反复多次运行同样的代码，运行时间分别为 22.256 秒，21.183 秒，20.091 秒，

19.827 秒，最短路径长度分别为 7681.4537，7774.2519，7681.4537，7681.4537。

#### 蚁群算法表现出了一下特点：

1. 它并非强求全局最优解，而是会满足于一个质量相对较高的局部最优解。

2. 起初算法的收敛速度较快，但是随着迭代次数的增加，收敛速度显著降低， 甚至会在测试中出现停滞的情况。

3. 蚁群算法对 TSP 问题有较好的适应性，无论数据规模大小，均能在一定时 间内得到较高质量的解。

4. 稳定性较差，即便不改变参数，也可能前后运行的结果迥异。为了提高解的

质量，不妨采用同样的参数运行 5 遍以上，选取其中最优解。

5. 算法中参数较多，参数的微小调动可能对运行结果的质量、运行时间等产生 意外的影响。例如，若蚂蚁数量 m 值过大，则会导致搜索路径上的信息素量变化 趋于平均，正反馈作用被削弱，导致收敛速度减慢。若 m 过小，则在处理城市数 量较多的情况时，易致使未经过的路径信息素恒为 0，使程序容易过早停滞。

四、**MATLAB** 使用模拟退火算法解 **TSP** 问题

**4.1** 模拟退火算法的原理

#### 模拟退火算法是一种通用大概率算法，可以在一个较大的搜索空间内寻求问题 的最优解。它的优点在于能有效解决 NP 难题、避免陷入局部最优解、对初值的依 赖关系相对较弱。

它的思想源于固体退火过程：将固体加热至高温状态，再以足够缓慢的速度冷 却。升温时，固体内部粒子随着温度上升而呈现无序状，内能增大，而在缓慢冷却 过程中，粒子又趋于有序状。理论上而言，若冷却过程足够缓慢，则冷却过程中任 意温度下，固体均可以达到热平衡，而冷却到低温时，将达到这一低温下的内能最 小状态。物理退火过程与模拟退火算法的类比关系图如下所示：

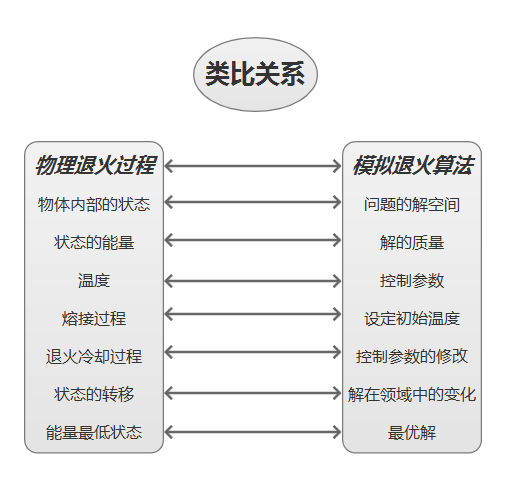


图 4: 模拟退火算法与固体退火过程类比图

**4.2** 模拟退火算法 **TSP** 模型的建立

#### 模型可以分为以下 5 步：

1. 构造 TSP 问题的解空间和初始解

TSP 问题的解空间 S 是一个遍历所有城市且每个城市仅经过一次的所有的回 路，是所有城市排列的集合。TSP 问题的解空间 S 可表示为 1,2,...,n 的全排列组合， 即

*S* = *{*(*c*1*, c*2*, ..., cn*)*|*(*c*1*, c*2*, ..., cn*)*}*

其中 *§i* 表示遍历 n 个城市的一个路径，*i* = *j* 表示第 i 次访问城市 j。由于该算法的 解的质量对初始状态依赖性较弱，故初始解为随机函数生成一个 {1,2,...,n} 的随机 排列作为 *§*0

2. 构建目标函数

#### TSP 问题的目标函数即为遍历所有城市的路径总长度，即

*n*+1

*C*(*c*1*, c*2*, ..., cn*) = ∑ *d ∗* (*ci, ci*+1) + *d ∗* (*c*1*, cn*)

*i*=1

#### 我们的目的为求解上式的最小值，则当上式最小时的城市排列即为所求的最短路 径。

3. 产生新解

新解的产生可以通过以下 2 种方法分别使用或者交替使用产生： 二变换法：任选序号 u、v(设 u<v<n)，交换 u 与 v 之间的访问顺序。 三变换法：任选序号 u、v(设 u<v<n)，u，v，w(设 u<v<w) 将 u 与 v 的路径插

到 w 之后访问。

#### 4. 目标函数差 计算变换前的解和变换后的目标函数的差值：

∆*C*´ = *C*(*s*´*i*) *− C*(*si*)

5.Metropolis 接受准则 以新解与当前解的目标函数之差定义接受概率，即

{ 1*, C*´ *<* 0

*P* =

exp(*−*∆*C*´/*T* )*, C*´ *>* 0

**4.3matlab** 实现模拟退火—**TSP** 模型的具体操作

首先，初始化温度衰减函数的参数，设定 markov 链程度为 10000(即最大迭代次 数为 10000)，并且读取城市坐标，记录到矩阵 coordinates 中。amount 表示城市的数 量。将初始解设置为 1 到 amount 的排列，记为 *solnew*。

#### 其次，计算每 2 个城市之间相邻的距离，并且将其保存到矩阵 *distmatrix* 中。 随后，进行随机扰动。采用 while 循环，当温度高于终止温度，循环进行一下 操作：采用 for 循环，当迭代次数小于 markov，则等可能地对当前解进行二交换或 三交换产生新解 *solnew*。并对比新解与旧解的内能 (目标函数值) 的大小。若新解 的内能更小，则将新解赋值给当前解，若当前解的内能更小，则仅以一定的概率接

受新解。

每当循环达到迭代次数，即上述 for 循环完成一次，则降低温度值 t，模拟退火 的降温过程。当 t 降至终止温度 tf 时，跳出 while 循环。

#### 最后，输出程序运行时间、最优解以及最短距离。

**4.4** 对模拟退火算法求解结果的展示与分析

matlab 命令窗口输出结果为： 程序执行时间:15.884 秒 最优解为：

1 至 15 列

18 3 17 21 42 7 2 30 23 20 50 29 16 46 44

|  |  |
| --- | --- |
| 16 至 30 列  34 35 36 39 40 37 38 48 24 | 5 15 6 4 25 12 |
| 31 至 45 列 |  |
| 28 27 26 47 13 14 52 11 51 33  46 至 52 列 | 43 10 9 8 41 |
| 19 45 32 49 1 22 31  最短距离： |  |
| 7.5444e+03  最短路径的图像如下所示： |  |

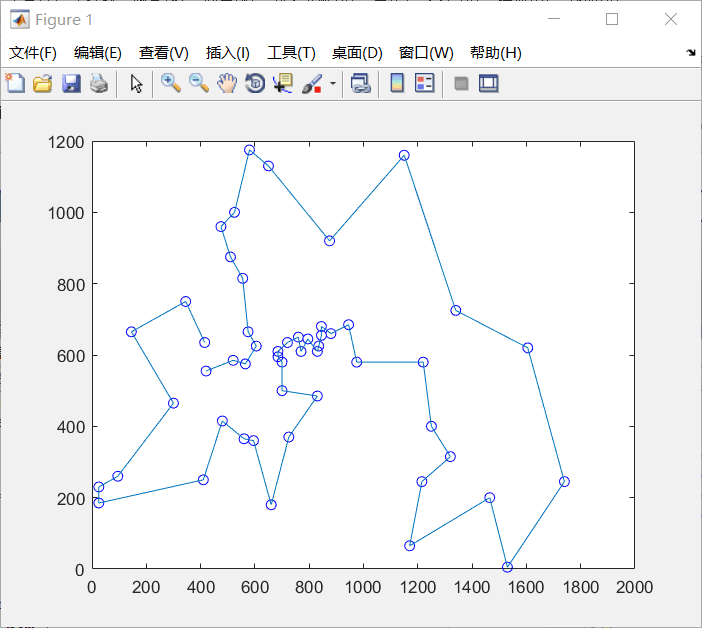


图 5: 模拟退火算法最优化路径

反复多次运行同样的代码，所得的最短路径长度分别为 7.5444e+03，7.6758e+03， 7.5444e+03，所耗费的运行时间分别为 19.031 秒，18.15 秒，18.769 秒

#### 分析可知：模拟退火算法的特点如下： 优点为：

1. 局部搜索能力强。

#### 2. 运行效率高，运行时间较短。 缺点为：

1. 全局搜索能力差。

2. 容易受参数的影响。

五、**MATLAB** 使用遗传算法解 **TSP** 问题

**5.1** 遗传算法的原理

#### 遗传算法基于生物学上的自然选择和基因遗传学原理，借鉴了优胜劣汰的自然 选择机理和生物界繁衍进化的基因重组、突变的遗传机制，是一种全局自适应概率 搜索算法。

遗传算法通常的实现方式为一种计算机模拟。对于最优化问题，一定数量的候 选解 (即个体) 可抽象表示为染色体，使种群向更优的解进化。进化从完全随机个 体的种群开始，逐代繁衍下去。在每一代中评价整个种群的适应度，从当前种群中 随机地选择多个个体，基于它们的适应度大小，通过自然选择和突变的方式产生新 的生命种群，这些种群在算法的下一次迭代中成为当前种群。

* 1. **matlab** 实现遗传算法—**TSP** 模型的具体操作
     1. 构造初始化填充函数 *create*\_*permutations*, 创建交叉子集的函数 *crossover*\_*permutation*,

产生变异后代的函数 *mutate*\_*permutation*，以及用于绘制由算法计算的数据的函 数 *my*\_*plot*。

2. 加载问题的数据与作图：

读入文件 points 中的点的坐标，并将城市的横坐标 x 与纵坐标 y 绘制成二维直 角图。

3. 计算点之间的距离：

使用 2 个 for 循环，依次计算出每 2 个城市之间的距离大小，并记录在 pointdis

#### 矩阵中，便于后续的计算。

4. 定义目标函数 *points*\_*fitness*, 用于求解适应度。

#### 5. 设置优化属性并执行遗传算法求解

**5.3** 遗传算法求解结果的展示与分析

运行代码后，MATLAB 命令行输出以下内容：

Optimization terminated: maximum number of generations exceeded.

#### 程序执行时间:37.562 秒此时已达到最大的遗传代数，故程序终止。此时，图像 窗口的最终形态为下图：

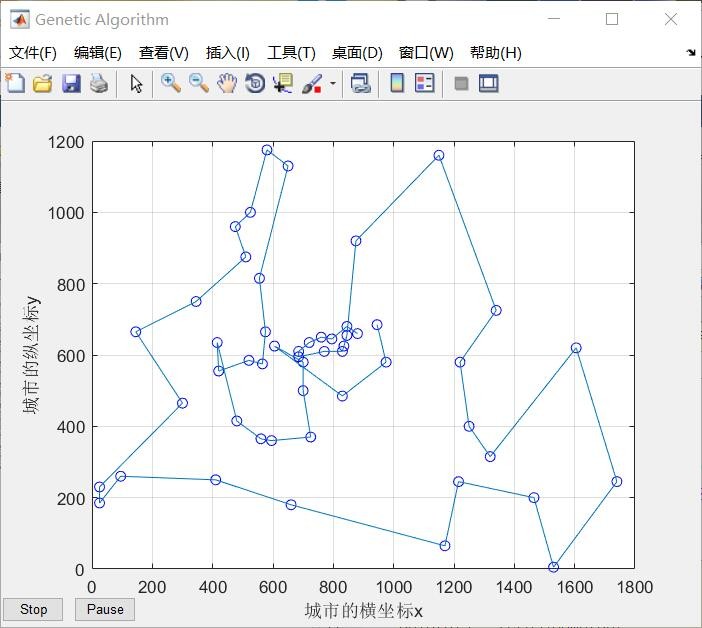


图 6:

反复运行该程序，得到的运行时间分别为 39.378 秒,36.357 秒，44.565 秒,68.874 秒，31.092 秒。说明它的求解稳定性一般，且运行时间多于其他 2 种 matlab 算法。 经过分析知：遗传算法每次终止条件都是达到最大代数，因此最大代数对运行时间 以及解的质量有着较大的影响

#### 遗传算法在该问题中表现了以下特点：

1. 同时处理群体中的多个个体，即对搜索空间中的多个解进行评估，减少了陷 入局部最优解的风险，同时算法本身易于实现并行化。

2. 具有自组织、自适应和自学习性。遗传算法利用进化过程获得的信息自行组 织搜索时，适应度大的个体具有较高的生存概率，并获得更适应环境的基因结构。 3. 从问题解的串集开始搜索，而不是从单个解开始。这是遗传算法与传统优化 算法的极大区别。传统优化算法是从单个初始值迭代求最优解的；容易误入局部最

优解。遗传算法从串集开始搜索，覆盖面大，利于全局择优。

六、**lingo** 解 **TSP** 问题

* 1. **lingo** 的特点与优势

#### (1) 既能求解线性规划问题，也有较强的求解非线性规划问题的能力；

(2) 输入模型简练直观；

(3) 运行速度快，计算能力强；

#### (4) 内置建模语言，提供几十个内部函数，从而能以较少语句，较直观的方式描述 较大规模的优化模型；

(5) 将集合的概念引入编程语言，很容易将实际问题转换为 LINGO 模型；

(6) 能方便地与 Excel、数据库等其他软件交换数据；

* 1. **lingo-TSP** 模型的建立

#### 由于 lingo 没有可视化的功能，我们不能直观地将路线图表示出来，因此不再 需要将每个点的坐标进行表示，而是直接应用每两个城市间的距离直接进行计算。 为了简化模型，我们使用 matlab 计算出所有的相邻 2 个城市之间的距离并输出一 个 Excel 表格，将表格中的数据复制到 lingo 的数据输入部分。

假使每次路线的起点与终点相同，则路线为一条闭合的回路。则无论起点为 52 个城市中的哪一个，所求得的路线均为相同的。我们不妨假设起点城市的序号即为 1。

* 1. **lingo** 编程与模型结合的思路

一般而言，lingo 编程解题可以分为一下四步：

1. 在集合段声明变量：

声明名为 city 的集合，集成员为 1 到 52 的整数。其中 num 为 city 的集属性， 用来表示 52 个不同的城市。

声明名为 link 的派生集，它由 2 个 city 派生而成，dist 与 x 为 link 的集属性。其 中 dist 为距离矩阵，dist(i,j) 表示城市 i 与城市 j 之间的距离。x 为路线矩阵，x(i,j)=0

表示在求解的路线中城市 i 与城市 j 之间不相连，x(i,j)=1 表示在求解的路线中城市

i 与城市 j 之间不相连。

#### 2. 在数据段输入解题所必需的数据：

3. 在初始段输入初始条件： 由于我们已经假设每次路线的起点与终点相同，则路线为一条闭合的回路。则

无论起点为 52 个城市中的哪一个，所求得的路线均为相同的。我们不妨假设起点 城市的序号即为 1。因此这个环节在我们对 TSP 问题的求解中不必要。

4. 在目标与约束段输入目标函数和约束条件： 由模型可知，我们的目标函数即为路线的总距离，它可以用距离矩阵和路线矩

阵相同位置的项的乘积来进行表示。我们可以使用 lingo 自带的 min 函数对其进行

#### 求解。

而约束条件有三个。首先由于每次选择的路线的唯一性，我们可以知道，每一 个城市 i 之前，有且仅有一个城市 j 与其相连。相应的，每一个城市 j 之后，有且 仅有一个城市 i 与其相连。我们可以用集循环函数对这两个约束条件进行描述。

第三个条件是最为关键，也最容易忽略的一个。若城市 i 与城市 j 相邻，则 x

（i,j)=1，且 num(i)-num(j)=-1，则恰好使得 num(i)-num(j)+52\*x(i,j)<=51 的等号成立， 若城市 i 与城市 j 不相邻，则 x（i,j)=0，且 num(i)-num(j) 的最大值为 51，同样满足 此式的等号成立。这样就有效的限制了路线是一个由 52 个城市组成的且不重复经 过任何一个城市的闭合回路。

**6.4** 对 **lingo** 解答的结果分析

lingo 自带的运行状态窗口如下图：

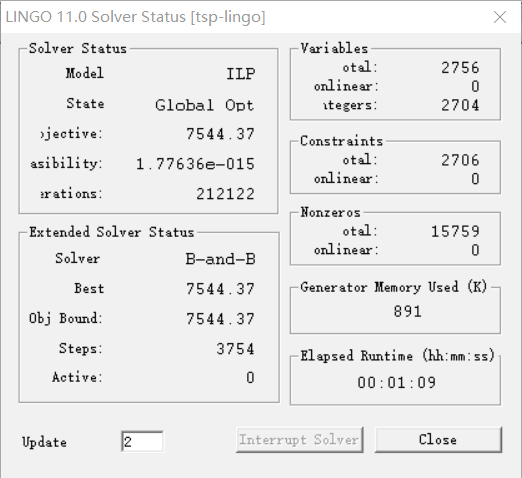


图 7: lingo 运行状态窗口

由上可知：TSP 为整数线性规划问题，该模型共 2756 个变量，其中 2704 个为

整数变量。约束条件有 2706 个，全为线性约束。非零的系数有 15759 个。目标函数 的最小值为 7544.37.lingo 采用的是 B-B 求解器 (分枝定界算法)。值得注意的是，其 运行时间较长，比 matlab3 种算法所耗费的时间均要长。这直观地证明了 matlab 对 求解 TSP 问题的效率之高。

#### 反复多次运行 lingo 程序，得到的目标函数值稳定不变，而运行时间分别为 1

分 09 秒，1 分 17 秒，1 分 10 秒。这证明了 lingo 求解的稳定性是比较高的。 将第三个之前提到的第三个约束条件删去，得到的目标函数的值为 6285.96，小

#### 于有约束条件时的目标函数值 7544.37，且运行时间仅需 1 秒。由此可知第三个约束 是紧约束，是不可缺少的关键约束条件。

对比上面四种解题方法，我们可以发现：使用 lingo 进行求解是在描述模型上

面最直观。使用 lingo 可以避免复杂繁琐的函数，而直接用内置的 min 函数即可自 动选取合适的求解器有针对性地实现解答。它的解的精确度也是最高的。但是它的 缺点在于求解效率显著低于 matlab 的算法，且不能将每个城市的坐标以及具体路 线进行可视化。

七、对比总结

由于我们对每种算法采用了相同的城市坐标数据，我们很容易能将 4 种方法进 行比对分析。经过分析，我们得到以下结论：

1.lingo 是 4 种方法里面对 TSP 问题适应性最好的。它仅需输入数据、约束条件 以及目标函数，即可自动选择合适的求解器。对于 TSP 问题，它采用了 B—and—B 求解器，即分枝定界算法。lingo 求得的目标函数值最为精确。而 lingo 相比 matlab 的缺陷在于，它的运行时间较长，显著多于 matlab 的算法，在运行效率上有着较大 的劣势。而且它不能实现最短路径的可视化，在描述运算结果方面不如 matlab 直 观。

#### 2.lingo 的求解稳定性是 4 者当中最优的。多次重复运行同样的代码，所得的计 算结果以及运行所用时间均无明显变化。

3. 不论采取何种算法，matlab 相比 lingo，都有着可以实现城市位置以及具体线 路的可视化的优势，这使得计算结果更加直观。4. 模拟退火算法是 matlab3 种算法 计算效率最高的，计算结果也略优于其他几个算法，这证明了模拟退火算法的可靠 性与有效性。

5. 蚁群算法的运行速度仅次于模拟退火算法，但是它的解的质量是最低的。

八、**4** 种方法可能改进的方向

**8.1** 蚁群算法的可能改进方向

#### 1. 蚂蚁数量 m 可以多次修改，并对比各种不同 m 下程序的运行时间与解的质量， 以求得更为合适的参数值。分析可知：当 m 过大，会导致路径上的信息素量变化 趋于平均，正反馈作用减弱，导致收敛速度减慢。当 m 过小，可能使程序过早停 滞，以致于解的全局优化性降低。

2. 若信息素因子的值过大，则蚂蚁选择已经走过的路径的概率会过更大，搜索的随 机性就会减弱；若过小，则等同于贪婪算法，使搜索过早陷入局部最优。

3. 若启发函数因子过大，会使搜索速度提高，但是搜索全局最优解的随机性会减 弱，易于陷入局部最优。若值过小，则群体会陷入纯粹的随机搜索，难以找到最优 解。

4. 信息素挥发因子的值过大，则可能会重复搜索，若过小，则会减慢收敛速度。

5. 信息素常数的值若过大收敛速度加快，但是易于陷入局优。若过小，则运行速率 会减慢。

6. 最大迭代次数过大，会浪费时间，可能在达到最大迭代次数前就已经收敛。若过 小，则可能来不及收敛

**8.2** 模拟退火算法的可能改进方向

#### 1. 初始温度应该足够高。若温度不够高，则可能退火过程终止过快。若温度过 高，则算法收敛速度会减慢。因此我们应反复调节初始温度，选择最合适的值。

2. 终止温度与初始温度同理。应当反复调参，确保参数的合理性。

3.markov 链长度应当合理。迭代次数不宜过多或过少。

4.metropolis 准则中的接受函数的参数 k 也应当取一个适应各个具体问题的值。

**8.3** 遗传算法的可能改进方向

#### 分析可知，遗传算法的函数 ga 是 matlab 内置的函数，而在实际代码中，几乎 没有输入可以影响算法的运行速度和解的质量的参数。因此遗传算法在 TSP 问题 的代码上优化空间可以忽略。

**8.4lingo** 代码的可能改进方向

#### 分析可知：整个 lingo 代码仅有输入数据、表示目标函数、表示约束条件三个 部分，而求解器是由 lingo 自动选择，其求解速度和解的质量由 lingo 本身所决定， 故 lingo 代码在提高运行速度和解的质量方面几乎不存在可以优化的空间。

九、参考文献

[1] 卓金武 王鸿钧.MATLAB 数学建模方法与实践 [M].3. 北京航天航空大学,

2018.

## 附录

1. **MATLAB** 使用蚁群算法解 **TSP** 问题的代码

### clear all

**clc**

t0 = **clock** ;

p o in ts=x l s re a d ( ’ points \_ data . x l s x ’ , ’ B2 : C53 ’ ) ;

n = **size** ( po ints , 1 ) ; D = **zeros** ( n , n ) ;

**for** i = 1 : n **for** j = 1 : n **i f** i ~= j

D( i , j ) = **sqrt** (**sum**( ( p o in ts ( i , : ) *−* p o in ts ( j , : ) ) . ^ 2 ) ) ;

### else

D( i , j ) = 1 e *−*4;

### end

**end end**

m = 7 5 ;

alpha = 1 ; **beta** = 5 ; vo l = 0 . 2 ;

Q = 1 0 ;

Heu\_F = 1 . /D; Tau = one s ( n , n ) ;

Table = **zeros** (m, n ) ; i t e r = 1 ;

iter\_max = 1 0 0 ;

Route\_best = **zeros** ( iter\_max , n ) ; Length\_best = **zeros** ( iter\_max , 1 ) ;

Length\_ave = **zeros** ( iter\_max , 1 ) ; Limit\_ ite r = 0 ;

**while** i t e r <= iter\_max s t a r t = **zeros** (m, 1 ) ;

**for** i = 1 :m

temp = **randperm**( n ) ; s t a r t ( i ) = temp ( 1 ) ; **end**

Table ( : , 1 ) = s t a r t ;

points \_ inde x = 1 : n ;

**for** i = 1 :m

**for** j = 2 : n

tabu = Table ( i , 1 : ( j *−* 1 ) ) ;

allow\_inde x = ~ ismember ( points \_inde x , tabu ) ; a l lo w = points \_ inde x ( allow\_inde x ) ;

P = a l lo w ;

**for** k = 1 : **length** ( a l lo w )

P( k ) = Tau( tabu (**end**) , a l lo w ( k ))^ alpha \* Heu\_F( tabu (**end**) , a l lo w ( k ))^ **beta** ;

### end

P = P/**sum**(P ) ;

Pc = **cumsum**(P ) ;

ta rg e t\_ inde x = **find** ( Pc >= **rand** ) ; ta rg e t = a l lo w ( ta rg e t\_ inde x ( 1 ) ) ; Table ( i , j ) = ta rg e t ;

### end end

Length = **zeros** (m, 1 ) ;

**for** i = 1 :m

Route = Table ( i , : ) ;

**for** j = 1 : ( n *−* 1 )

Length ( i ) = Length ( i ) + D( Route ( j ) , Route ( j + 1 ) ) ;

### end

Length ( i ) = Length ( i ) + D( Route ( n ) , Route ( 1 ) ) ;

### end

**i f** i t e r == 1

[ min\_Length , min\_index ] = **min**( Length ) ; Length\_best ( i t e r ) = min\_Length ; Length\_ave ( i t e r ) = **mean**( Length ) ; Route\_best ( i te r , : ) = Table ( min\_index , : ) ; Limit\_ ite r = 1 ;

### else

[ min\_Length , min\_index ] = **min**( Length ) ;

Length\_best ( i t e r ) = **min**( Length\_best ( i t e r *−* 1 ) , min\_Length ) ; Length\_ave ( i t e r ) = **mean**( Length ) ;

**i f** Length\_best ( i t e r ) == min\_Length Route\_best ( i te r , : ) = Table ( min\_index , : ) ; Limit\_ ite r = i t e r ;

### else

Route\_best ( i te r , : ) = Route\_best (( i t e r *−* 1 ) , : ) ;

### end

**end**

Delta\_Tau = **zeros** ( n , n ) ;

**for** i = 1 :m

**for** j = 1 : ( n *−* 1 )

Delta\_Tau ( Table ( i , j ) , Table ( i , j +1)) = Delta\_Tau ( Table ( i , j ) , Table ( i , j +1)) + Q

### end

Delta\_Tau ( Table ( i , n ) , Table ( i , 1 ) ) = Delta\_Tau ( Table ( i , n ) , Table ( i , 1 ) ) + Q/ Len

### end

Tau = (1*−* vo l ) \* Tau + Delta\_Tau ;

i t e r = i t e r + 1 ; Table = **zeros** (m, n ) ; **end**

[ Shortest\_Length , inde x ] = **min**( Length\_best ) ; Shortest\_Route = Route\_best ( index , : ) ; Time\_Cost=**etime**( **clock** , t0 ) ;

**disp** ( [ ’ 最短距离 : ’ **num2str**( Shortest\_Length ) ] ) ;

**disp** ( [ ’ 最短路径 : ’ **num2str** ( [ Shortest\_Route Shortest\_Route ( 1 ) ] ) ] ) ;

**disp** ( [ ’ 收敛迭代次数 : ’ **num2str**( Limit\_ ite r ) ] ) ;

**disp** ( [ ’ 程序执行时间 : ’ **num2str**( Time\_Cost) ’ 秒 ’ ] ) ;

点 ’ ) ;

**figure** ( 1 )

**plot** ( [ p o in ts ( Shortest\_Route , 1 ) ; p o in ts ( Shortest\_Route ( 1 ) , 1 ) ] , . . . [ p o in ts ( Shortest\_Route , 2 ) ; p o in ts ( Shortest\_Route ( 1 ) , 2 ) ] , ’ o*−*’ ) ; **grid** on

**for** i = 1 : **size** ( po ints , 1 )

**text** ( p o in ts ( i , 1 ) , p o in ts ( i , 2 ) , [ ’ ␣␣␣ ’ **num2str**( i ) ] ) ;

### end

**text** ( p o in ts ( Shortest\_Route ( 1 ) , 1 ) , p o in ts ( Shortest\_Route ( 1 ) , 2 ) , ’ ␣␣␣␣␣␣␣起

**text** ( p o in ts ( Shortest\_Route (**end**) , 1 ) , p o in ts ( Shortest\_Route (**end**) , 2 ) , ’ ␣␣␣␣␣␣␣终

点 ’ ) ;

**xlabel** ( ’ 点位置横坐标 ’ )

**ylabel** ( ’ 点位置纵坐标 ’ )

**title** ( [ ’ 最优化路径ACA最短距离 ( : ’ **num2str**( Shortest\_Length ) ’ ) ’ ] )

**figure** ( 2 )

**plot** ( 1 : iter\_max , Length\_best , ’ b ’ )

**legend** ( ’ 最短距离 ’ ) **xlabel** ( ’ 迭代次数 ’ ) **ylabel** ( ’ 距离 ’ )

**title** ( ’ 算法收敛轨迹 ’ )

1. **MATLAB** 使用模拟退火算法解 **TSP** 问题的代码

c l e a r a l l , c l c t t t=c lo c k ;

a = 0 . 9 9 ;

t0 = 9 7 ; t f = 3 ; t = t0 ; Markov\_length = 1 0 0 0 0 ;

p o in ts = x l s re a d ( ’ points \_ data . x l s x ’ , ’ B2 : C53 ’ ) ;

numofpoi = s i z e ( po ints , 1 ) ;

p o i n t s i d i s = z e ro s ( numofpoi , numofpoi ) ;

coor\_x\_tmp1 = p o in ts ( : , 1 ) \* one s ( 1 , numofpoi ) ; coor\_x\_tmp2 = coor\_x\_tmp1 ’ ;

␣␣␣␣␣␣␣␣ coor\_y\_tmp1 ␣=␣ p o in ts ( : , 2 ) ␣\*␣ one s ( 1 , numofpoi ) ;

␣␣␣␣␣␣␣␣ coor\_y\_tmp2 ␣=␣coor\_y\_tmp1 ’ ;

p o i n t s i d i s = s q rt ( ( coor\_x\_tmp1*−*coor\_x\_tmp2 ) . ^ 2 + . . . ( coor\_y\_tmp1*−*coor\_y\_tmp2 ) . ^ 2 ) ;

sol\_new = 1 : numofpoi ; E\_current = i n f ; E\_best = i n f ;

s o l\_ c u rre n t = sol\_new ; s o l\_ be s t = sol\_new ;

p = 1 ;

**while** t>=t f

**for** r =1: Markov\_length

**i f** ( rand < 0 . 5 )

ind 1 = 0 ; ind 2 = 0 ;

**while** ( ind 1 == ind 2 )

ind 1 = c e i l ( rand . \* numofpoi ) ; ind 2 = c e i l ( rand . \* numofpoi ) ; end

tmp1 = sol\_new ( ind 1 ) ;

sol\_new ( ind 1 ) = sol\_new ( ind 2 ) ; sol\_new ( ind 2 ) = tmp1 ;

### else

ind 1 = 0 ; ind 2 = 0 ; ind 3 = 0 ;

**while** ( ind 1 == ind 2 ) | | ( ind 1 == ind 3 ) . . .

| | ( ind 2 == ind 3 ) | | ( abs ( ind1*−*ind 2 ) == 1 ) ind 1 = c e i l ( rand . \* numofpoi ) ;

ind 2 = c e i l ( rand . \* numofpoi ) ; ind 3 = c e i l ( rand . \* numofpoi ) ; end

tmp1 = ind 1 ; tmp2 = ind 2 ; tmp3 = ind 3 ;

**i f** ( ind 1 < ind 2 ) && ( ind 2 < ind 3 )

;

e l s e i f ( ind 1 < ind 3 ) && ( ind 3 < ind 2 )

ind 2 = tmp3 ; ind 3 = tmp2 ;

e l s e i f ( ind 2 < ind 1 ) && ( ind 1 < ind 3 ) ind 1 = tmp2 ; ind 2 = tmp1 ;

e l s e i f ( ind 2 < ind 3 ) && ( ind 3 < ind 1 ) ind 1 = tmp2 ; ind 2 = tmp3 ; ind 3 = tmp1 ; e l s e i f ( ind 3 < ind 1 ) && ( ind 1 < ind 2 ) ind 1 = tmp3 ; ind 2 = tmp1 ; ind 3 = tmp2 ; e l s e i f ( ind 3 < ind 2 ) && ( ind 2 < ind 1 ) ind 1 = tmp3 ; ind 2 = tmp2 ; ind 3 = tmp1 ; end

tm p l i s t 1 = sol\_new ( ( ind 1 + 1 ):( ind2 *−*1 )); sol\_new ( ( ind 1 + 1 ):( ind 1+ind3*−*ind 2 +1)) = . . . sol\_new ( ( ind 2 ) : ( ind 3 ) ) ;

sol\_new ( ( ind 1+ind3*−*ind 2 +2 ): ind 3 ) = . . . tm p l i s t 1 ;

end

E\_new = 0 ;

**for** i = 1 : ( numofpoi *−*1) E\_new = E\_new + . . .

p o i n t s i d i s ( sol\_new ( i ) , sol\_new ( i + 1 )); end

E\_new = E\_new + . . .

p o i n t s i d i s ( sol\_new ( numofpoi ) , sol\_new ( 1 ) ) ;

**i f** E\_new < E\_current E\_current = E\_new ;

s o l\_ c u rre n t = sol\_new ;

**i f** E\_new < E\_best

E\_best = E\_new ; s o l\_ be s t = sol\_new ; end

### else

**i f** rand < exp(*−*(E\_new*−*E\_curre nt ) . / t )

E\_current = E\_new ;

s o l\_ c u rre n t = sol\_new ;

### else

sol\_new = s o l\_ c u rre n t ; end

end end

t=t . \* a ; end

Time\_Cost=e time ( c lo c k , t t t ) ;

dis p ( [ ’ 程序执行时间 : ’ num2str ( Time\_Cost) ’ 秒 ’ ] ) ; dis p ( ’ 最优解为： ’ )

dis p ( s o l\_ be s t ) dis p ( ’ 最短距离： ’ ) dis p ( E\_best )

p lo t ( p o in ts ( : , 1 ) , p o in ts ( : , 2 ) , ’ bo ’ ) a x i s ( [ 0 2000 0 1 2 0 0 ] ) ;

hold on

p lo t ( p o in ts ( s o l\_ be s t , 1 ) , p o in ts ( s o l\_ be s t , 2 ) )

* 1. **MATLAB** 使用遗传算法解 **TSP** 问题的主代码

t t t = **clock** ;

p o in ts =x l s re a d ( ’ points \_ data . x l s x ’ , ’ B2 : C53 ’ ) ; numofpo=**size** ( po ints , 1 ) ;

**plot** ( p o in ts ( : , 1 ) , p o in ts ( : , 2 ) , ’ bo ’ ) ;

**xlabel** ( ’ 点的横坐标x ’ ) ; **ylabel** ( ’ 点的纵坐标y ’ ) ;

**grid** on

p o in td i s = **zeros** ( numofpo ) ;

**for** count1 =1: numofpo

**for** count2 =1: count1

x1 = p o in ts ( count1 , 1 ) ; y1 = p o in ts ( count1 , 2 ) ; x2 = p o in ts ( count2 , 1 ) ;

y2 = p o in ts ( count2 , 2 ) ;

p o in td i s ( count1 , count2 )=**sqrt** ( ( x1*−*x2 )^2+( y1*−*y2 ) ^ 2 ) ; p o in td i s ( count2 , count1 )= p o in td i s ( count1 , count2 ) ;

### end end

Fitne s s Fc n = @( x ) p o in t s \_ f i tn e s s ( x , p o in td i s ) ;

my\_plot = @( o ptio ns , s ta te , **flag** ) po ints \_ plo t ( o ptio ns , . . . s ta te , **flag** , p o in ts ) ;

o p t io n s = o ptimo ptio ns (@ga , ’ Population Type ’ , ’ custom ’ , ’ In i t ia l P o p u la t io n R a [ 1 ; numofpo ] ) ;

o p t io n s = o ptimo ptio ns ( o ptio ns , ’ Creation Fcn ’ , @ create\_permutations , . . . ’ CrossoverFcn ’ , @ crossover\_permutation , . . .

’ MutationFcn ’ , @mutate\_permutation , . . . ’ PlotFcn ’ , my\_plot , . . .

’ MaxGenerations ’ , 5 0 0 , ’ Po p u la t io n S iz e ’ , 6 0 , . . .

’ Max S tall Ge ne rations ’ , 2 0 0 , ’ Us e Ve c to riz e d ’ , true ) ;

numberOfVariables = numofpo ; [ x , fva l , re as on , output ] = . . .

ga ( Fitne s s Fcn , numberOfVariables , [ ] , [ ] , [ ] , [ ] , [ ] , [ ] , [ ] , o p t io n s ) ;

Time\_Cost=**etime**( **clock** , t t t ) ;

**disp** ( [ ’ 程序执行时间 : ’ **num2str**( Time\_Cost) ’ 秒 ’ ] ) ;

# create\_permutations 函数的代码

**function** pop = c re a te \_ pe rmuta tio ns (NVARS, Fitne s s Fcn , o p t io n s ) to ta l P o p u la t io n S i z e = **sum**( o p t io n s . Po p u la t io n S iz e ) ;

n = NVARS;

pop = c e l l ( to ta l P o p u la t io n S i z e , 1 ) ;

**for** i = 1 : to ta l P o p u la t io n S i z e pop{ i } = **randperm**( n ) ;

### end

* 1. **crossover\_permutations** 函数的代码

**function** xove r Kids = c ro s s o ve r\_ pe rmuta tio n ( pare nts , o ptio ns ,NVARS, . . . Fitne s s Fcn , th i s S c o re , th i s P o p u la t io n )

nKids = **length** ( pa re nts ) / 2 ;

xove rKids = c e l l ( nKids , 1 ) ; *% Normally zeros (nKids ,NVARS) ;*

inde x = 1 ;

**for** i =1: nKids

pare nt = th i s P o p u la t io n { pa re nts ( inde x ) } ; inde x = inde x + 2 ;

p1 = **ceil** (( **length** ( pare nt ) *−*1) \* **rand** ) ;

p2 = p1 + **ceil** (( **length** ( pare nt ) *−* p1*−* 1 ) \* **rand** ) ; c h i ld = pare nt ;

c h i ld ( p1 : p2 ) = **f l iplr** ( c h i ld ( p1 : p2 ) ) ;

xove rKids { i } = c h i ld ; *% Normally , xoverKids ( i , : ) ;*

### end

* 1. **mutate\_permutations** 函数的代码

**function** mutation Childre n = mutate\_permutation ( pa re nts , o ptio ns ,NVARS, . . . Fitne s s Fcn , s ta te , th i s S c o re , this Po pula tio n , mutation Rate )

mutation Childre n = c e l l ( **length** ( pa re nts ) , 1 ) ;

**for** i =1:**length** ( pa re nts )

pare nt = th i s P o p u la t io n { pa re nts ( i ) } ;

p = **ceil** ( **length** ( pare nt ) \* **rand** ( 1 , 2 ) ) ; c h i ld = pare nt ;

c h i ld ( p ( 1 ) ) = pare nt ( p ( 2 ) ) ;

c h i ld ( p ( 2 ) ) = pare nt ( p ( 1 ) ) ; mutation Childre n { i } = c h i ld ; **end**

* 1. **points\_ﬁtness** 函数的代码
  2. **points\_plot** 函数的代码

**4.lingo** 解 **TSP** 问题的代码

**model** : s e t s :

c i t y / 1 . . 5 2 / : num;

l i n k ( c i ty , c i t y ) : d i s t , **x** ;

e n d s e ts data :

d i s t=

|  |  |  |  |
| --- | --- | --- | --- |
| 0 666 . 1080993 281 . 1138559 | 395 . 6008089 | 291 . 2043956 | 326 |
| 666 . 1080993 0 649 . 3265742 | 1047 . 091209 | 945 . 1454914 | 978 |
| 281 . 1138559 649 . 3265742 0 | 603 . 5105633 | 508 . 9449872 | 542 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 395 . 6008089 | 1047 . 091209 | 603 . 5105633 | 0 104 . 4030651 69 . | | | |
| 291 . 2043956 | 945 . 1454914 | 508 . 9449872 | 104 . 4030651 0 35 . | | | |
| 326 . 26676 2 | 978 . 0848634 | 542 . 5172808 | 69 . 64194139 35 . 35533906 | | | |
| 640 . 8002809 | 45 610 . 5735009 1026 . 364945 | | | 923 . 5935253 | | 957 |
| 426 . 8782028 | 956 . 1511387 308 . 058436 525 | | | 470 . 5581792 | | 491 |
| 600 . 1874707 | 1134 . 955946 | 485 . 6439025 | 611 . 0032733 | | 583 . 6308765 | |
| 561 . 4712815 | 1132 . 982789 | 487 . 2627628 | 533 . 9007398 | | 513 . 4685969 | |
| 1040 . 973102 | 1638 . 787662 | 1266 . 688596 | 663 . 1930337 | | 760 . 8054942 | |
| 655 . 0190837 | 1258 . 590481 | 891 . 3613184 | 294 . 3637206 | | 382 . 4264635 | |
| 975 1440 . 078123 1247 . 757989 711 . 0731327 769 . 0416114 744 | | | | | | |
| 1120 . 769825 | 1515 . 725899 | 1399 . 732117 | 897 . 0089186 | | 944 . 3119188 | |
| 299 . 0401311 | 957 . 8230526 | 504 . 8762225 | 100 . 124 922 | | 25 40 . | |
| 260 . 0480725 | 724 . 033839 | 537 . 4011537 | 384 . 2199891 | | 309 . 2329219 | |
| 429 . 5346319 | 494 . 7726751 | 217 . 3131381 | 800 . 2499609 | | 700 . 0714249 | |
| 161 . 5549442 | 595 . 4829972 | 134 . 6291202 | 532 . 3532662 | | 430 . 46486 5 | |
| 305 843 . 4008537 207 . 0024154 474 . 6841055 400 . 7804885 427 | | | | | | |
| 210 . 0595154 | 564 . 4687768 | 440 . 9648512 | 500 . 6246099 | | 406 . 6017708 | |
| 286 . 9233347 | 392 . 4601891 | 288 . 5307609 | 681 . 4 87344 | | 577 . 16981 9 | |
| 46 . 09772229 | 636 . 4157446 | 240 . 5202694 | 436 . 6062299 | | 332 . 4530042 | |
| 181 . 1767093 | 509 . 8284025 | 361 . 1786262 | 537 . 7034499 | | 436 . 8352092 | |
| 274 . 5906044 | 921 . 7917335 | 505 . 6925944 | 125 . 2996409 | | 31 . 62277 66 | |
| 410 . 0304867 | 1028 . 846441 | 652 . 533524 | 109 . 2016483 | | 150 . 0833102 | |

enddata

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 728 . 9718787 | 1191 . 511645 | 1005 . 94483 | 5 1 6 . 2 3 6 3 8 | 552 . 2680509 |
| 798 . 5142453 | 13 01 . 50874 | 1067 . 637579 | 526 . 8064161 | 584 . 1446739 |
| 707 . 0007072 | 1243 . 724246 | 970 . 3221115 | 417 . 4326293 | 478 . 591684 |
| 406 . 2634613 | 635 . 0196847 | 651 . 2488004 | 579 . 8706752 | 509 . 754843 |
| 360 . 0694377 | 390 . 4484601 | 504 . 2072986 | 689 . 5288246 | 594 . 3483827 |
| 146 . 3728117 | 541 . 2254613 | 208 . 9258242 | 540 . 8558033 | 436 . 6062299 |
| 90 . 55385138 730 245 . 2039967 370 . 5401463 270 . 1851217 305 | | | | |
| 827 . 3149 34 | 1488 . 707493 | 903 . 3963693 | 517 . 3490118 | 589 . 9576256 |
| 135 . 0925609 | 782 . 0805585 | 393 . 6051321 | 266 . 5520587 | 163 . 2482772 |
| 121 . 6552506 | 776 . 9813383 | 373 . 6642878 | 275 . 1363298 | 170 . 8800749 |
| 125 785 | 367 . 6955262 | 270 . 6011826 | 166 . 2077014 | 201 . 3082214 |
| 207 . 9663434 | 857 . 7004139 | 447 . 4650824 | 190 . 3943276 | 87 . 46427842 |
| 240 . 4163056 | 896 . 9392399 | 462 . 087654 | 155 . 241 747 | 50 . 99019514 |
| 166 . 2077014 | 827 . 9643712 | 392 . 2371731 | 230 . 4886114 | 126 . 589889 |
| 208 . 9258242 | 869 . 7413409 | 426 . 8782028 | 188 . 2817038 | 85 . 14693183 |
| 395 . 3795645 | 896 . 1724164 | 246 . 9817807 | 544 . 5410912 | 479 . 5049531 |
| 565 . 7958996 | 102 . 5914226 | 550 . 0909016 | 950 . 3288904 | 847 . 6585397 |
| 463 . 8156962 | 1123 . 710372 | 55 6 . 596802 | 245 . 2039967 | 266 . 6927071 |
| 154 . 4344521 | 744 . 8825411 | 434 . 1946568 | 307 . 0016287 | 212 . 2498528 |
| 240 . 20 8243 | 823 . 2860985 | 219 . 8294794 | 411 . 0960958 | 331 . 2099032 |
| 279 . 8660394 | 859 . 0838143 | 552 . 6753116 | 230 . 7054399 | 170 . 6604817 |
| 791 . 2806076 | 1151 . 271037 | 1072 . 310589 | 659 . 5642501 | 673 . 5911223 |
| 267 . 3013281 | 910 . 3021476 | 504 . 8019414 | 137 . 2953022 | 47 . 43416 49 |
| 64 . 03124237 | 728 . 0109889 | 288 . 4874347 | 34 5 . 2 5 3 5 3 | 241 . 8677324 |
| 217 . 0829335 | 596 . 2591718 | 463 . 2493929 | 477 . 6243294 | 386 . 6846 26 |
| 789 . 3826702 | 1421 . 557245 | 995 . 3140208 | 397 . 0201506 | 499 . 9249944 |
| 1220 . 460978 | 1716 . 049242 | 1483 . 59361 | 908 . 6390923 | 984 . 4414 66 |
| ; |  |  |  |  |

**min**=@sum( l i n k : d i s t \***x** ) ;

@ for ( c i t y ( j ) : @sum( c i t y ( i ) | j#ne#i : **x**( i , j ))= 1 ) ;

@ for ( c i t y ( i ) : @sum( c i t y ( j ) | i#ne#j : **x**( i , j ))= 1 ) ;

@ for ( l i n k ( i , j ) | i#ne#j#**and**#i#g t #1:num( i )*−*num( j )+52\***x**( i , j )<=51);

@ for ( l i n k : @bin (**x** ) ) ;

### end