

TSP 问题及其求解

现代优化计算方法期末大作业

卢韬

自动化 1903 班

20194127

更新: *May 31, 2022*

摘 要

TSP 问题已经被研究多年 (Reinelt (2003)), 已有诸多工作从精确算法 (动态规划)、启发式算法 (随机贪婪搜索)、亚启发式算法 (禁忌搜索 (Dewantoro et al. (2019))), 元启发式算法 (遗传算法、粒子群优化 (Elloumi et al. (2014))) 等方向入手, 在获取精确解的能力上取得了较好的效果。然而, 由于 TSP 问题 NP 难的特点, 没有单一的方法能快速、精确地求解各种类型的 TSP 样例。而这些算法评价指标之间普遍存在着诸多 trade-off, 单一的求解最优解的能力不足以衡量算法好坏。因此面对不同特点的算例样本, 需要一种科学的评价指标衡量选取哪种算法更好。本项目将讨论多种用于求解 TSP 问题算法的原理, 并将使用 python 复现、对比各类算法。最终, 结合实验过程和关键评价指标 (精度、复杂度、速度) 给出一种统一的算法评价指标计算方法, 并得到不同特点的算例下最适合的算法。所有的代码和笔记将开源在: [github 开源地址](#)。

关键词: TSP 问题, 遗传算法, 禁忌搜索, 贪婪算法, 动态规划算法, 粒子群优化算法

目录

1 项目进度	4
2 TSP 问题介绍	4
2.1 TSP 问题的数学模型	4
2.2 TSP 问题数据集	5
2.3 ATSP 问题转化为 TSP 问题	6
2.3.1 改写距离获取函数	6
2.3.2 构造新的邻接矩阵	7
3 求解 TSP 问题的算法	7
3.1 精确算法	8
3.1.1 分支定界法	8
3.1.2 线性规划算法	9
3.1.3 动态规划算法	9
3.2 启发式算法	9
3.3 深度学习算法	9
4 禁忌搜索算法	9
4.1 禁忌搜索算法求解 TSP 问题	9
4.1.1 算法关键要素	10
4.1.2 算法实现	11
4.1.3 最近邻算法	11
4.1.4 k-opt 算法	11
4.1.5 算法优化	12
4.1.6 训练过程参数控制	12
5 粒子群优化算法	13
5.1 粒子群算法介绍	13
5.2 超参数（粒子个数、迭代次数等）对搜索性能的探究	13
5.2.1 使用对数坐标系进行可视化	14
5.3 算法实现	14
5.4 对照试验	15
5.4.1 实验设计	15
5.4.2 实验分析	15
5.5 粒子群算法求解 TSP	17

6 蚁群算法	17
7 评估模型	18
7.1 模型假设	18
7.2 数据规约	19
7.2.1 输入	19
7.2.2 建模	19
7.2.3 输出	19
7.3 建模	19
7.4 模型解释	19
8 总体算法实现、代码亮点	19
8.1 可复现性	19
8.2 可视化	19
8.3 TSP 问题路径可视化	20
8.4 调试工具	20
8.4.1 算法效率	20
8.4.2 算法可靠性	20
A 附录	22
A.1 程序代码	22
A.2 图片	28

1 项目进度

目前已完成 TS 和 PSO 算法的部分代码，同时对数据加载模块完成了更详细的书写，并封装为 TSP 类。保证其能对多种类型测试样例均可用。由于时间原因，本文档尚未完善，代码也没有完全实现手写。最终预测模型也没有搭建（但只要实验做完了，用 XGBOOST 等方法很快就能构建模型）。

目前完成的代码在 Colab 的线上平台可直接运行：[预测模型](#)、[TS 算法求解](#)、[PSO 算法](#)、[网络可视化](#)。

2 TSP 问题介绍

旅行推销员问题（Travelling salesman problem, TSP）描述如下：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。它是组合优化中的一个 NP 难问题，在运筹学和理论计算机科学中非常重要。TSP 问题在 1930 年代在维也纳和哈佛大学首次开始研究，在 1950 年代和 1960 年代在学术界开流行，当时的最优模型可以解决 48 城市的 TSP 问题。在 1970 年代，解决能力达到 120 维，不久又达到 318 维。如今，目前的最强模型能求解多达 24978 维的 TSP 问题。从图论的角度来看，该问题实质是在一个带权完全无向图中，找一个权值最小的 Hamilton 回路¹。由于该问题的可行解是所有顶点的全排列，随着顶点数的增加，会产生组合爆炸。1972 年，Richard Karp 发表了一篇 17k 引用的论文 [Karp \(1972\)](#)，证明了哈密顿循环问题是 NP 完全的。这意味着数学上来说，旅行商问题是 NP 难的。

TSP 问题具有实际应用背景。目前被广泛应用于公共服务可靠性、信号控制、路径规划（[Smith et al. \(2005\)](#)）、PCB 生产（[Reinelt \(2003\)](#)）、流水车间问题及其变体（[Bagchi et al. \(2006\)](#)）等诸多实际问题中。

2.1 TSP 问题的数学模型

对 TSP 问题有如下数学描述：

$$\min \sum_{i,j=1}^n x_{ij} d_{ij} \quad (1)$$

Subject to :

$$\sum_{i \in V} x_{ij} = 1, \forall i \in V_c \quad (2)$$

$$\sum_{j \in V} x_{ij} = 1, \forall j \in V_c \quad (3)$$

¹哈密顿图是一类无向图，从指定的起点前往指定的终点，途中经过所有其他节点且只经过一次。闭合的哈密顿路径称作哈密顿回路。

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \forall S \subset V_c, 2 \leq |S| \leq n \quad (4)$$

公式中未知数的含义如表1:

表 1: 未知数定义

未知数	含义	取值范围	注释
d_{ij}	i 到 j 的距离	$[0, -\infty)$	距离衡量可以用 p 范数
x_{ij}	在 i 后访问 j	0,1	使用 one-hot 编码
i, j	城市代号	$i \neq j$	不超过城市数量
N	节点集合	$\text{len}(N)=n$	所有城市的集合

2.2 TSP 问题数据集

TSP 问题可用的数据集非常之多, 比如Reinelt (1991) 提出的 TSPLIB—A。由于问题研究时间跨度长, 有非常多类型的数据格式、数据规约、路径计算方式都被提出。这极大影响了数据加载部分。主要表现为在数据集表头部分的可选参数处, 这使得数据无法按照统一的位置定位来一次性提取, 而应该读取对应位置数据规约, 来针对性计算参数。如, EDGEWEIGHTFORMAT 给出了边权重的给出方式, 其中有邻接矩阵、邻接表、优化为上三角矩阵的邻接矩阵, 优化为上三角矩阵但不带对角线元素的邻接矩阵, 优化为上三角矩阵但按列储存的邻接矩阵。因此, 我花费了极大精力, 完成了面向更普遍情况下的数据集加载器 (代码详见附录), 能读取对应参数进行针对性计算。

根据 Google 公司的一篇文档²解释了对一般 TSP 问题测试样例数据的规约, 可以得到读取数据集方式, 具体读取代码在附录中。但是仍有部分规约不符合此文档, 需要在读取过程中发现。

附录中给出了快速下载 TSP 测试样例³的下载 python 脚本。

附录中还给出了提取文件夹中所有.tsp 样例的 python 代码。

在数据规约中, 有 DISPLAYDATATYPE 参数可选, 但如果此参数显示 Undefined 或者 NoDisp, 则说明其中城市数据无法对应到真实的几何坐标系中。又或者, 数据以邻接表的方式或是邻接矩阵的方式给出, 只有相对距离而无绝对坐标, 这就使得数据无法可视化。因此, 在数据初步可视化阶段, 利用图论可视化工具 (Nikita Sharma (n.d.)), 可以将邻接矩阵读取并转化为邻接关系数列, 并进一步通过随机指定绝对坐标来可视化无向图, 如图1。进一步的, 还可以使用G.remove_edges_from(nx.selfloop_edges(G))来去除自环。

²数据详细解释: [下载地址](#)

³下载源来源: [数据集下载地址](#)

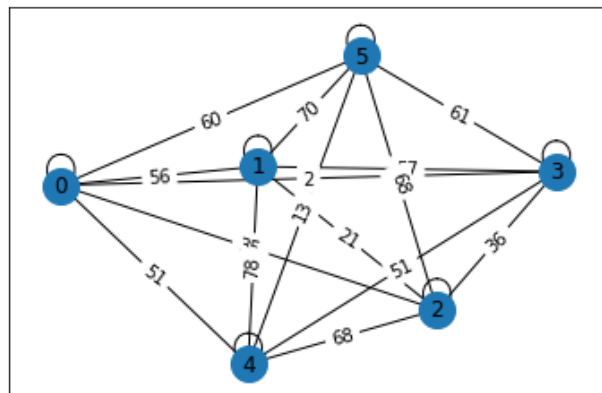


图 1: TSP 数据可视化 (蓝色边为可行路径。)

2.3 ATSP 问题转化为 TSP 问题

在上述的 TSP 问题中，算例给出的通常是城市的二维坐标。 i, j 两点之间的距离 $d_{i,j}$ 通过欧拉距离 (2-范数) 计算，因此两点间不同向的边距离是相同的。此类图组成的邻接矩阵有对称性，在图论中形成一个无向图，而这种天然的对称性将解的数量减少了一半。但实际生产过程中，部分问题只能转化为非对称旅行商问题 (Asymmetric Travelling Salesman Problem, ATSP)，其中 $d_{i,j} \neq d_{j,i}$ 。此类图组成的邻接矩阵无对称性，在图论上可转化为有向图。比如在《现代优化计算方法》课程的实验一中，需要解决的问题就是 ATSP。

对于 ATSP 数据集，其文件一般以 .atasp 结尾，同时表头信息中 TYPE 位置会有 ATSP 提示。数据加载器要有足够的能力来获取此信息，及时切换数据集加载器、数据处理方式和算法。

将未知类似问题转换为特殊情况下的已知问题是常用的手段，因此将 ATSP 转换为 TSP 问题求解将有助于扩大 TSP 求解能力。因此根据其特点，有如下两种方式将 ATSP 问题转换为 TSP 问题求解。

2.3.1 改写距离获取函数

由于 TSP 问题给出的邻接矩阵具有对称性，因此 TSP 算法在设计时就会利用对称性去重⁴。这样就导致以下三个问题：

- .atasp 和 .tsp 测试样例保存的格式不同。
- TSP 算法中对称邻接矩阵保存的数据结构会降重，导致缺失非对称距离储存能力。
- TSP 算法中对称邻接矩阵距离读取函数不具备非对称读取能力。

因此想要将 TSP 算法的求解程序应用到 ATSP 问题上，一种直接的思路是利用 python 的重写特性，以上三个位置的函数重写即可。对于启发式算法来说，这样的转化是非常完美的。

⁴参考对上三角、下三角、稀疏等特殊矩阵的优化储存。

2.3.2 构造新的邻接矩阵

经过文献检索，我找到了一种将非对称 TSP 问题转换为对称 TSP 问题的方法（Volgenant and Jonker (1983)）。通过这种方法，我们可以将非对称 TSP 问题转化为对称 TSP 问题，然后使用适合对称问题的算法求解该问题，而不是重新设计算法。

此算法将一个 A 称 TSP 问题的距离矩阵 C 转化为对称 TSP 问题的距离矩阵 \tilde{C} 。这样，只需要在读取数据是进行相应转换，就可以无须改动其他代码而直接使用已有 TSP 求解方法。算法步骤如下：

1. 令 $\bar{C} = C$ ，其中 $\bar{c}_{ij} = -M$ ($i \in N$)。其中 M 为相当大的数，表示无法通行。
2. 取 $U_{n \times n}$ 为 n 维方阵。即 $i, j \in N$ ，有 $u_{ij} = \infty$ 。
3. 有 ATSP 问题的转化对称距离矩阵

$$\tilde{C} = \begin{pmatrix} U & \bar{C}^T \\ \bar{C} & U \end{pmatrix} \quad (5)$$

4. 此时新的城市节点集合为：

$$\tilde{N} = \{1, 2, \dots, n, n+1, \dots, 2n\} \quad (6)$$

5. 此时需要转化后 TSP 问题得到的最优解有以下以下形式：

$$N(i_1) \rightarrow N(i_1 + n) \rightarrow N(i_2) \rightarrow N(i_2 + n) \rightarrow \dots \rightarrow N(i_n) \rightarrow N(i_n + n) \rightarrow N(i_1) \quad (7)$$

其对应的转化函数如下，拼接过程使用了 numpy 库。

```
# 为了使算法方便观察，没有使用节省内存和更快的写法，
# 但减少中间变量和新建数组，会显著加快程序运行
def ATSP2TSP_np(matrix):
    lu = np.ones_like(matrix)*(10000)
    ru = matrix.T
    rd = np.ones_like(matrix)*(10000)
    # 先竖着拼接
    left = np.concatenate((lu, matrix), axis=0)
    right = np.concatenate((ru, rd), axis=0)
    # 整体拼接
    return np.concatenate((left, right), axis=1)
```

[

3 求解 TSP 问题的算法

根据所学内容，TSP 问题有有如图2常用算法。总体分为精确算法、启发式算法和深度学习算法。

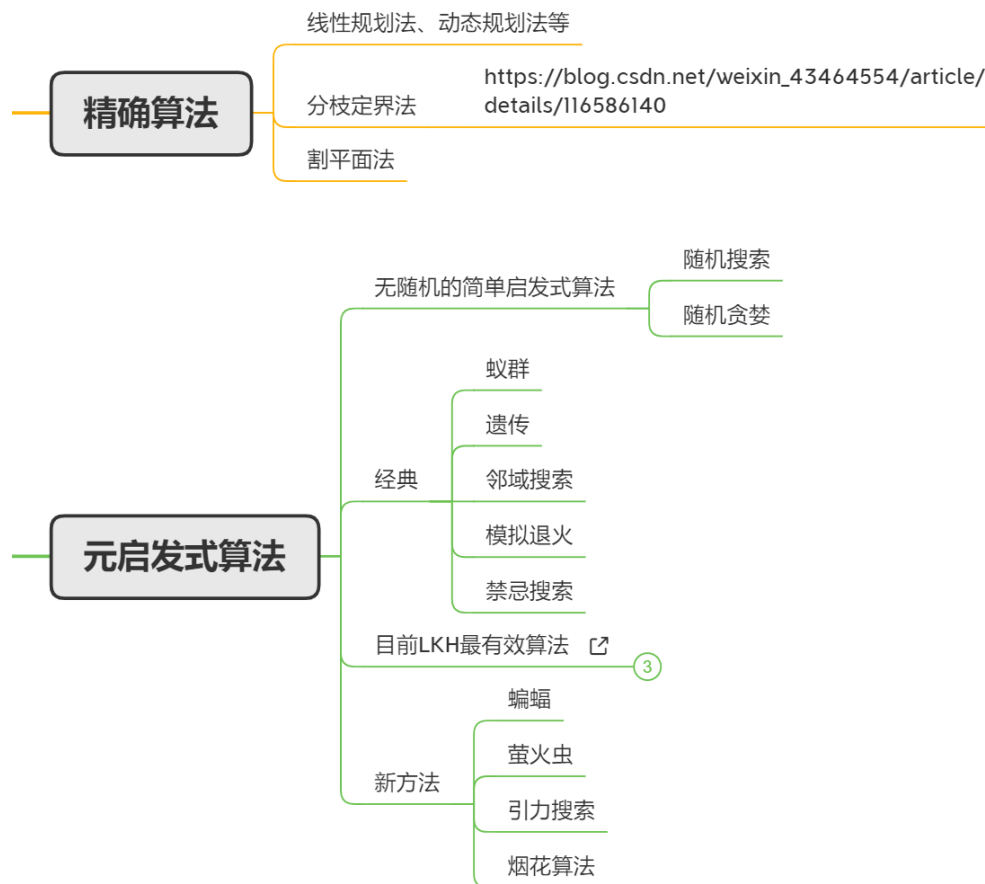


图 2: 算法汇总

3.1 精确算法

最直接的求解算法是对所有可能的路径进行完整枚举，以确定成本最低的路径。然而，对于 n 个城市求解的时间复杂度为 $O(n!)$ 时间，这种方法仅适用于较小的城市规模。

3.1.1 分支定界法

根据 2021 秋季学期课程《系统优化与人工智能》和《系统工程概论》，分支定界方法常用于求解各类优化问题。

ILP 首先被宽松并使用 Simplex 方法作为 LP 求解，然后通过枚举整数变量重新获得可行解。

3.1.2 线性规划算法

3.1.3 动态规划算法

3.2 启发式算法

NP-hard 问题难以在大尺度上以最优方式精确求最优解。启发式算法常用于给出一个好的近似解，但不一定是最优解。这些算法不保证最优解，但在合理的计算时间内给出接近最优的解。

3.3 深度学习算法

基于深度学习的算法准确率已经接近经典方法。但目前效率仍然较差²，在数百节点左右的规模上就很难快速收敛。因此无法将学习到的策略推广到实际规模的更大实例。Joshi et al. (2020) 的工作将目前主流的深度学习技巧重新整合，并提出了合适的网络结构和学习方式。同时其开源代码已有较多使用量，非常成熟。

表 2: 在不同的 TSP 大小和搜索设置中，SL 和 RL 训练的模型的近似训练时间（12.8M 样本）和推理时间（1280 样本）。GS: 贪婪策略，BS128: 宽度为 128 的束搜索，S128: 采样 128 个解决方案。RL 训练使用推出的基线，计时包括每 128,000 个样本后更新基线的时间。

图尺寸	训练时间 ⁵		推理时间		
	SL	RL	GS	BS128	S128
TSP20	4h 24m	8h 02m	2.62s	7.06s	63.37s
TSP20-50	9h 49m	15h 47m	-	-	-
TSP50	16h 11m	40h 29m	7.45s	29.09s	86.48s
TSP100	68h 34m	108h 30m	19.04s	98.26s	180.30s
TSP200	-	495h 55m	54.88s	372.09s	479.37s

4 禁忌搜索算法

4.1 禁忌搜索算法求解 TSP 问题

禁忌搜索 (Tabular Search, TS) 最初由Glover (1986) 提出。禁忌搜索是一种亚启发式随机搜索算法，它从一个初始可行解出发，通过随机指定的变化规则（比如 2-gram），选择实现让特定的目标函数值变化最多的移动。本质上是通过扩大邻域搜索结构来得到更好的解。为了避免陷入局部最优解，TS 搜索中采用了禁忌表作为记忆机制。对已经试探过得较优解，能在一定时间内避免尝试从而避免震荡求解的情况。

根据算法介绍，可以得到流程框图，如图 3。

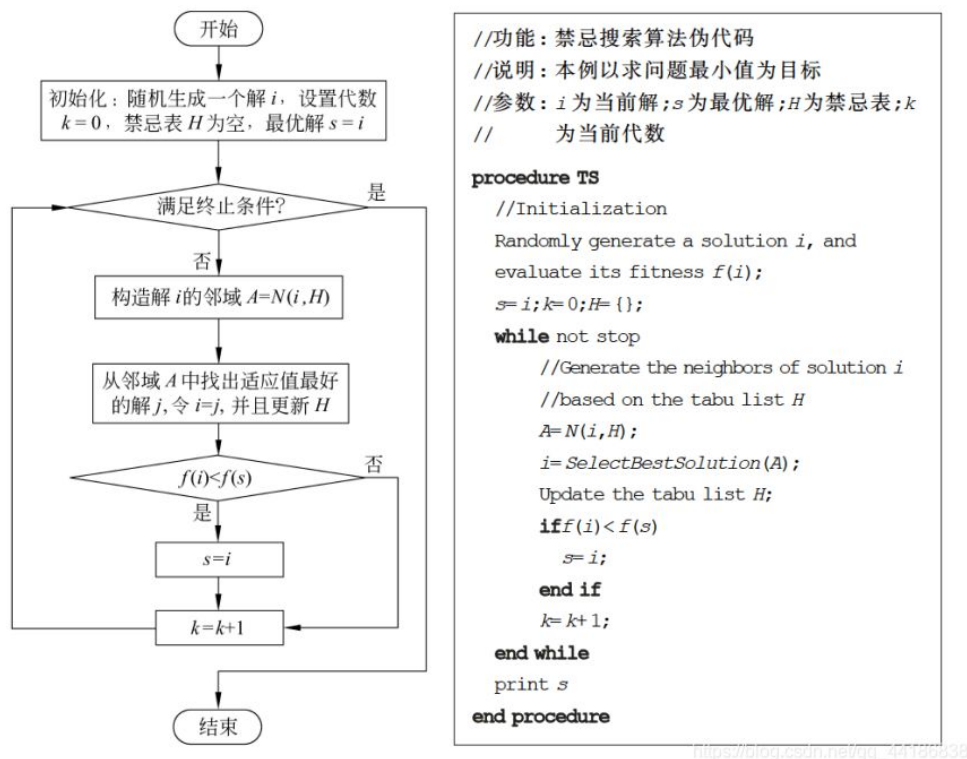


图 3: 禁忌搜索算法求解 TSP 问题思路

4.1.1 算法关键要素

1. 禁忌对象的选取 (禁忌表中被禁的那些变化元素)

- 解的简单变化: 将原来的解和变化后的解都记录下来。如: $ABCDE \rightarrow ACBDE$
- 向量的变化: 记录变化位置和变化量。如: $(1, 1, 0, 1, 0, 1) \rightarrow (1, 0, 0, 1, 0, 1)$
- 目标值的变化: 禁止相同目标值的解被接受。如: $f(x)$ 。

2. 邻域结构 (决定邻域按什么样的方式生成)

- 使用 k-opt, 既交换当前解级的 k 个城市。 $(123456) \rightarrow (126543)$
- 比如 da Costa et al. (2021) 中使用的 2-opt 方法, 每次交换两个城市, 如 $(123456) \rightarrow (123465)$ 。

3. 禁忌长度的确定 (被禁对象不允许选取的迭代代数, $\text{tabu}(x) = t$)

- t 为给定常数。
- t 为某一范围。
- t 动态变化, 比如我使用的余弦下降、幂指数下降等策略。

4. 特赦规则 (aspiration criteria)

- 基于评价值的规则: 遇到更好的解。
- 基于最小错误的规则: 当所有对象都被禁忌时, 选择最好。
- 基于影响力的规则: 对目标函数的影响大小。

5. 候选集合的确定

- 选择邻域中目标值最佳的若干个邻居。
- 随机选取。

6. 终止规则

- 确定步数终止：迭代代数超过给定值 K ，终止计算。
- 频率控制原则：某一解、目标值、元素序列的频率超过给定标准，终止计算。
- 目标控制原则：给定步数内目标值没有改进。
- 目标值偏离程度原则： $f(x) - Z_{LB} \leq \epsilon$ 。

4.1.2 算法实现

根据如图13算法探究，我完成了使用禁忌搜索的 TSP 求解。这里我选择了 ATSP 作为测试样例，但是本质上与 TSP 都是相通的。此类问题有两个思路，转换为 TSP 或者按照 ATSP 的非对称路径来读取。我将两种思路都实现了，最终对比发现后者在储存、速度上都更好。

4.1.3 最近邻算法

```
1  V = {1, ..., n-1}    // 除 0 以外的顶点
   U = {0}              // 顶点 0
3  while V not empty
   u = 最近添加到U的顶点
   找到V中离u最近的顶点v
   将顶点v加入U中，并从V中移除
7  endwhile
   根据加入U的顺序输出顶点
```

4.1.4 k-opt 算法

用于 TS 算法中生成解

```
T = 一些tour(初始解)
2 noChange = true
  repeat
4   for all T中可能的边-对
     T' = 交换边-对中的端点获得的tour
6     if T' < T
       T = T'
8     noChange = false
     break      // 一旦发现改进就退出循环
```

```

10         endif
        endfor
12    until noChange
    return T

```

4.1.5 算法优化

本模型首次完成时间较早，但是算法性能始终不好，因此反复尝试了一些技巧来提升性能。从最初的 2500+（最好）到现在的 1900+（最好）。本次实验的代码完全由本人完成，没有借鉴任何代码。

由于算法从零开始实现，因此数据结构的选取、迭代过程中中间数据的处理等细节有诸多实现方法。因此在完成实验内容后，我利用 python 中的修饰器技巧，通过函数包装来快速获取各个函数的运行时间，便于找到耗时的瓶颈位置。只需要在写好的函数上加入 @timmer 即可在调用过程中自动记录时间。此方法来源于我的个人博客，在初学 python 时候记录的笔记。

```

1 import time
def timmer(func):    #传入的参数是一个函数
3     def deco(*args, **kwargs): #本应传入运行函数的各种参数
        print('\n函数: {_funcname_} 开始运行: '.format(_funcname_=func.__name__))
        start_time = time.time() #调用代运行的函数，并将各种原本的参数传入
        res = func(*args, **kwargs)
        end_time = time.time()
        print('函数:{_funcname_} 运行了 {_time_} 秒'
9             .format(_funcname_=func.__name__, _time_=(end_time - start_time)))
        return res #返回值为函数
11    return deco

```

最终在算法运行速度优化上有总体 85% 时间优化。比如在读取距离方法、ATSP 问题处理等多个算法耗时的瓶颈处做了消融实验，选取了运行速度更快的解决方案。同时在算法书写时也使用 lamda 表达式排序等方法尽量减少代码复杂度。在执行时间判断函数过程中，使用了 jupyter notebook 独有的魔术指令 %timeit，在每个单元格中都运行多次同一份代码，求取平均值作为最终结果。下图记录的是数据结构的决定。

4.1.6 训练过程参数控制

为了保证算法的泛化性能，在代码实现过程中将所有可控变量解耦合。用较多的封装保证算法可以很容易的进行参数调节。在 run () 函数中，可以通过改变超参数传入，容易的完成各项功能的消融实验。下表所示的是算法的超参数单独单元定义。

```

12 # 算法需要的变量
    tabLength = 50                # 禁忌表长
14 banned_Table = []             # 禁忌表
    city_Num = tab_np.shape[0]    # 城市数
16 cityNum = tab_np.shape[0]
    randSeed = 42                 # 随机数种子
18 random.seed(randSeed)         # 给random库设计随机数生成器
    maxIter = 1000                # 终止准则
20 neighborNum = 20000           # 生成的邻居数量
    solution = list(range(city_Num)) # 解
22 solution_list = []            # 生成多个解构成的数组
    exchange_Num = 2              # 领域交换量 2-opt
24 # 当前路径长度（加一个变化的权重，路径长度和value不一样，非线性，比如平方）
    # 加大对离谱解的惩罚力度，但要小心震荡
26 thisLength = 1000000
    thisValue = 1000000
28 random_accept= 0.05           # 随机接受概率

```

[

同时我也结合自己在深度学习中学习的技巧，将 learning rate (tabu 长度) 下降、抖动（长时间梯度不下降就跳开）、允许随机犯错等手段加入了算法实现过程中，尽可能优化算法。

观察到表长度较小的时候，很容易出现算求解来回震荡的情况。因此算法需要在表长衰减时候做截断，来保证搜索能力。

5 粒子群优化算法

5.1 粒子群算法介绍

根据 PSO 课程上记录的笔记（图14），可以得到算法的流程图和重要参数。

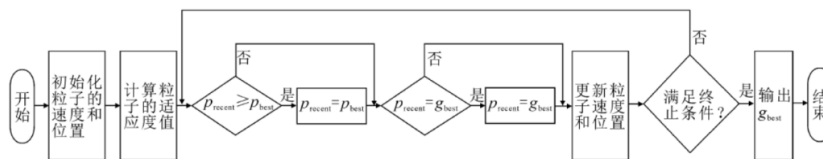


图 4: 粒子群算法标准流程

5.2 超参数（粒子个数、迭代次数等）对搜索性能的探究

由于 PSO 算法性能很大程度上取决于超参数选取，因此想要达到更好的性能，需要对超参数进行详细对照实验。但是使用整体算法求解 TSP 问题时速度较慢，很长时间才能得到最后的解，不

利于调参。所以这里先选用几个常用的测试函数对算法进行大致探索研究，再将其迁移到现有问题上。

这里选用 Griewark 函数、Rastrigrin 函数和 Rosenbrock 函数三种典型测试函数。根据实验结果和 TSP 问题特性来做具体参数选择。

观察 PSO 算法在粒子个数分别设置为 50、100 和 300，迭代次数分别设置为 100、500 和 1000 情况下的寻找最优解的过程。

5.2.1 使用对数坐标系进行可视化

当使用 30 维的多变数扩展测试函数时，算法搜索范围可以达到 $(1e0, 1e9)$ ，跨度很大。如下图所示 5 (a) 所示，可以观察到三种颜色表示的三种参数下的三条曲线紧贴在一起，无法直观观察的算法的搜索过程。

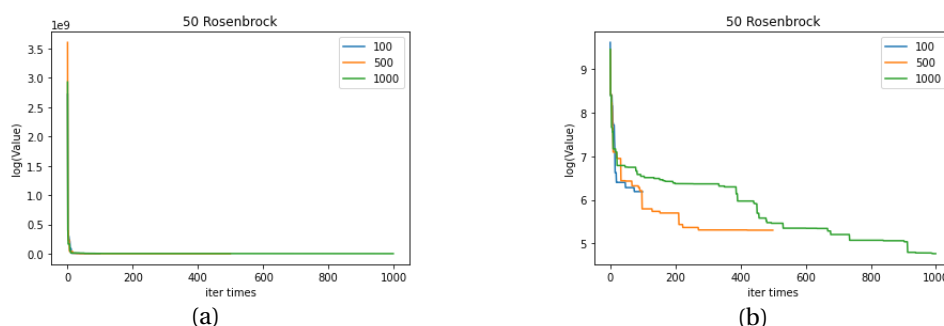


图 5: 坐标变换效果对比

因此，考虑到模电和自控原理中 bode 图的对数坐标系，我将数据进行了对数化，使用纵轴为对数坐标系的方法，使得不同参数情况下算法的可视化可以直观体现出来（如图 5 (b)）。具体实现时，可以采用 matplotlib 的对数系，也可以采用 numpy 将数据整体转换，还可以手工转换。这里为了更好体现算法细节，使用手写转换方式。

观察到三条曲线成功分离开了，便于算法性能的对比研究。但是也要注意，对尺度进行缩放也会导致差距较为接近的两条线水平方向重合，其斜率也被去了对数，变化趋势不敏感。因此对于算法执行末期，数量级差距不大情况下，还应该回到标准坐标系下探究。

5.3 算法实现

同时，吸取前两次实验中算法实现使用顺序结构的弊端。设置过多全局变量定义的超参数导致进行重复实验时出现变量泄露等问题，这些问题在 python 中（尤其是 jupyter notebook 中）debug 很困难。此次算法实现用类来封装，超参数使用类内变量，使得创建和销毁都更安全，做多组对照实验时出现问题较少。

同时，由于整体算法在求解 TSP 问题时速度较慢，不利于调参。这里先选用几个常用的测试函数对算法进行大致探索研究，再将其迁移到现有问题上。

这里选用 Griewark 函数、Rastrigrin 函数和 Rosenbrock 函数三种典型测试函数。

5.4 对照试验

5.4.1 实验设计

考虑到实验中涉及到三组变量：目标函数、迭代次数、种群量。为了直观展现这些变量之间的关系，我使用了 matplotlib 的子图绘制和隐函数的迭代器，可以方便的进行重复对照实验。同时，我也使用了变量控制，每个子图中的三条下降曲线都只有一个超参数变化。比如：当使用 50 种群搜索 rosenbrock 函数时，迭代次数分别设置为 100、500、1000 进行试验，其图像放置于大图6的 (1, 1) 位置。

图中：表头为次小图中实验的两个固定参数，用于固定此 2 参数，比对图例中的参数。大图中的 9 个小图表示不同参数情况下的运行情况，用于比对其余两参数的情况。

5.4.2 实验分析

从图6可以看到不论对于何种函数或采用何种规模种群，其中总存在非常大的“平台期”⁶，也存在有平台期对应的拐点。比如观察到对于 Rastrigrin 和 Griewark 函数，在 250iter 后搜索的边际效益逐渐降低，种群已经逐步落入局部最优的陷阱，很难再有进步。继续增加 iter，算法几乎没有进步。而对于 Rosenbrock 函数，由于其搜索难度较大，因此更多的迭代次数能显著增加结果的最优性。根据 100 次的重复实验，也可以判断对 Rosenbrock 函数，其搜索结果与迭代次数具有较强线性相关性。

通过衰减速率来看（需要注意的是，图示衰减速率取了对数，但不影响大小关系），大的种群能有更持久的搜索能力，图7（2，1）、图7（2，3）可以明显看出较小的种群容易陷入某个局部最优值，而种群数量大就有更多机会持续找到更优的局部最优值。既对于大多数情况，大种群的收敛位置要优于小种群。这样的想法也在更大规模、更细密的采样上得到了实验。这里推荐使用 google 的 colab pro 平台，其提供的服务器使用多个 Intel(R) Xeon(R) CPU @ 2.20GHz，运行速度要比本地快很多，支持了本次大规模测试。我也共享了我的代码，连接在最前方。

但是同时，我通过做额外的对照试验（图8）发现，过大的种群（5000）反而会导致收敛非常慢，计算时间也呈指数倍增长。而且根据问题不同，搜索算法也不是一定会明显更好。种群规模与算法效率、算法性能之间存在 trade-off，需要在具体实现测试样例中分析合适的参数取值。比如在 griewark 测试样例中，算法性能就没有明显提升。

这里运算计算时间利用 python 的修饰器完成，既将计算函数传入 @ 开头的另一个修饰器函数中。

⁶既多个迭代次数后算法达到的最优解没有更新。

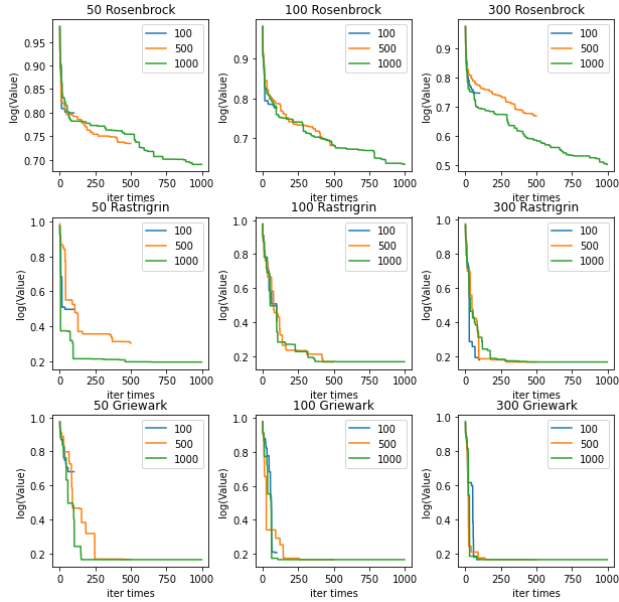


图 6: 对迭代次数的探究

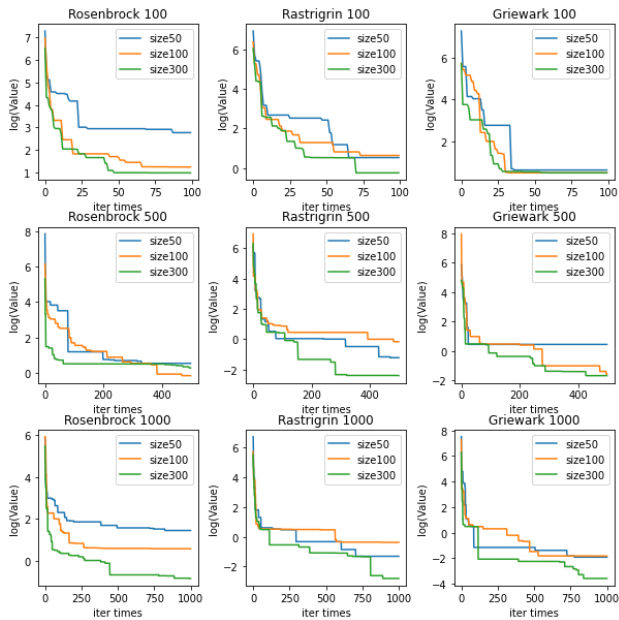


图 7: 对种群大小的探究

也可以使用 `tqdm` 的迭代器来完成 (如图9), 还可以用 `python` 自带的分析测试工具完成。观察到计算时间随着种群数量指数级上升, 图中也展示了多次运行求平均 (减少随机性干扰) 的思想。需要注意的是, `python` 中可以使用魔术指令 `%timeit` (或者 `cProfile` 分析) 来自动重复运行单元格, 获取平均算法耗时, 但是多次运行后, 部分变量会被系统优化进入 `L2chahe` 缓存、或者被其他解释器自带优化方法优化, 其统计的平均耗时将会小于真实情况。因此手写重复运算可能是更合适的方法, 这样内循环的变量会被 `python` 自动回收, 不会被机器优化。

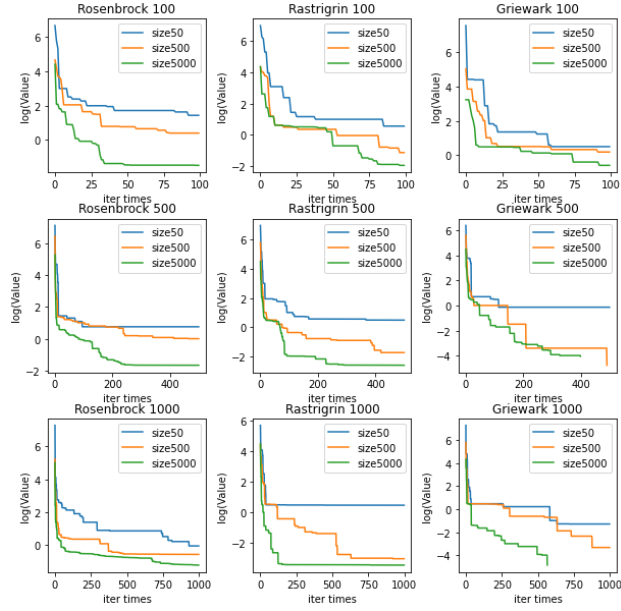


图 8: 种群量额外的对照实验 (种群量为 50、500、5000)

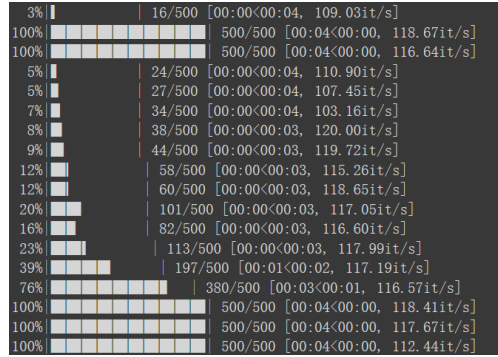


图 9: 不同种群量计算耗时分析 (单位: 每秒多少次迭代)

5.5 粒子群算法求解 TSP

使用简化的函数优化任务, 我们得到了粒子群算法的较优参数。下面使用这样的参数构建 TSP 问题求解器。

6 蚁群算法

蚁群算法 (Ant Colony Optimization, ACO), 又称蚂蚁算法, 是一种用来在图中寻找优化路径的几率型算法。它由 **Dorigo and Di Caro (1999)** 提出, 其灵感来源于蚂蚁在寻找食物过程中发现路径的行为。

蚂蚁在一开始时, 会以相同的概率选择路线, 在运动过程中, 能够在其经过的路径上留下信息素, 而且能感知这种物质的存在以及强度, 并以此指导自己的运动方向, 蚂蚁倾向于往信息素浓度

高的方向移动。在相等时间内，较短路径上的信息量就遗留的比较多，则选择较短路径的蚂蚁也随之增多。蚂蚁群体行为表现出了一种信息正反馈现象，即某一路径上走过的蚂蚁越多，则后来者选择该路径的概率就越大，蚂蚁个体之间就是通过这种信息交流机制来搜索食物，并最终沿着最短路径行进。

最大最小蚁群算法与普通蚁群算法区别不大，主要是每条路径上的信息素浓度都有个最大值最小值，每次更新都保证信息素浓度在 $[\min, \max]$ 区间范围内。最小信息素能增加对最优解探索的可能性 (exploration)，而最大信息素浓度保证经验对于蚂蚁的启发性 (exploitation)。每次都蒸发，但是每次蒸发后只有最优路径上的信息素会增加。这样的收敛速度会更快。

```

1  计算信息素轨迹的限制:  $\tau_{\min}$  and  $\tau_{\max}$ 
2  将信息素的轨迹值设置为  $\tau_{\max}$ 
3   $global\_best \leftarrow \emptyset$ 
4  for  $i \leftarrow 1$  to #iterations do
5      for  $j \leftarrow 0$  to #ants-1 do
6           $u \leftarrow \mathcal{U}\{0, n-1\}$  // 随机选择第一个节点
7           $route_{Ant(j)}[0] \leftarrow u$ 
8          把  $u$  加到  $tabu_{Ant(j)}$ 
9          for  $k \leftarrow 1$  to  $n-1$  do // 完成解决方案 (路线)
10              $u \leftarrow \text{select\_next\_node}(route_{Ant(j)}[k-1], tabu_{Ant(j)})$ 
11              $route_{Ant(j)}[k] \leftarrow u$ 
12             把  $u$  加到  $tabu_{Ant(j)}$ 
13      $iter\_best \leftarrow \text{select\_shortest}(route_{Ant(0)}, \dots, route_{Ant(\#ants-1)})$ 
14     if  $global\_best = \emptyset$  or  $iter\_best$  短于  $global\_best$  then
15          $global\_best \leftarrow iter\_best$ 
16         使用  $global\_best$  更新信息素轨迹限制  $\tau_{\min}$  和  $\tau_{\max}$ 
17     按照  $\rho$  参数衰减信息素
18     累积信息素基于  $iter\_best$ 

```

图 10: MAX-MIN 蚁群算法

7 评估模型

7.1 模型假设

假设问题的图都是完备的，可转化为邻接矩阵的形式。每两个不同节点都有带权边连接。

7.2 数据规约

由于实验需要进行多次，因此需要对每次运行的结果做限制，来保证模型不会确实数据。

7.2.1 输入

评估模型的输入分为两部分：算法运行参数和数据的情况。

算法运行参数：时间、精度（需要读取最优值）、数据名称、最优解（因为最优解是已知的）。

对应的测试样例：数据维度、长度分布不均衡性、等图的性质。

7.2.2 建模

7.2.3 输出

7.3 建模

7.4 模型解释

8 总体算法实现、代码亮点

8.1 可复现性

随机性使用随机数种子，确定了随机程度。

运行环境：线上平台 GoogLe Colab（保证不同机器算法运行环境相同）、requirement.txt（python 库环境）

8.2 可视化

如图11，本项目使用了 matplotlib 中的可视化方法，并加以封装。同时通过训练过程中数据结构保存中间结果，便于直观感受算法运行情况，给出调参数的方向。

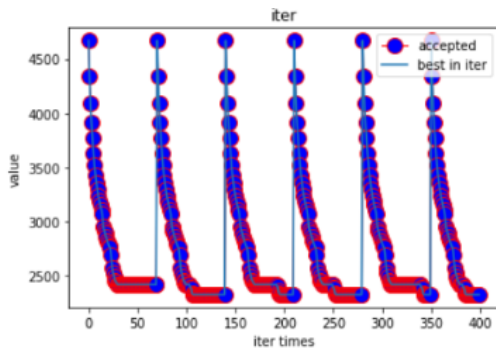


图 11: 同一个初始解进行多次搜索

使用了 tqdm 库，来可视化算法的训练进程。在大规模测试情况下得以判断运行时间。

8.3 TSP 问题路径可视化

使用复杂网络处理工具 `networkx`⁷ (Hagberg et al. (2008)) 有助于直观展示算法效果。比如环的散乱程度 (图12)。此方法将 `numpy.ndarray` 格式的临界矩阵数据经过方阵检查、编号、去重复等处理转化为邻接表, 进而可视化。

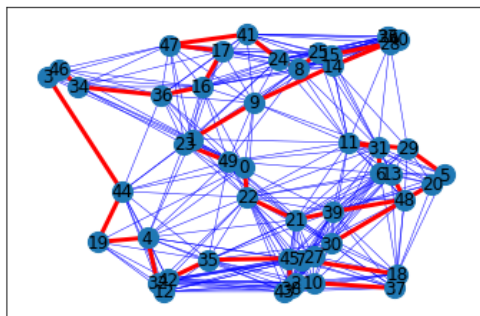


图 12: TSP 路径可视化 (其中红色边为最终结果, 蓝色边为可行驶路径。)

8.4 调试工具

8.4.1 算法效率

8.4.2 算法可靠性

使用 `warning`、`assert`、`raise Exception` 等错误处理机制, 便于切换测试样例带来问题的排查。

⁷复杂网络工具 `networkx`

参考文献

- Bagchi, Tapan P, Jatinder ND Gupta, and Chelliah Sriskandarajah**, “A review of TSP based approaches for flowshop scheduling,” *European Journal of Operational Research*, 2006, 169 (3), 816–854.
- da Costa, Paulo, Yingqian Zhang, Alp Eren Akçay, and Uzay Kaymak**, “Learning 2-opt Local Search from Heuristics as Expert Demonstrations,” *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.
- Dewantoro, Rico Wijaya, Poltak Sihombing et al.**, “The combination of ant colony optimization (ACO) and tabu search (TS) algorithm to solve the traveling salesman problem (TSP),” in “2019 3rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM)” IEEE 2019, pp. 160–164.
- Dorigo, Marco and Gianni Di Caro**, “Ant colony optimization: a new meta-heuristic,” in “Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406),” Vol. 2 IEEE 1999, pp. 1470–1477.
- Elloumi, Walid, Haikal El Abed, Ajith Abraham, and Adel M Alimi**, “A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP,” *Applied Soft Computing*, 2014, 25, 234–241.
- Glover, Fred**, “Future paths for integer programming and links to artificial intelligence,” *Computers & operations research*, 1986, 13 (5), 533–549.
- Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart**, “Exploring Network Structure, Dynamics, and Function using NetworkX,” in Gaël Varoquaux, Travis Vaught, and Jarrod Millman, eds., *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA 2008, pp. 11 – 15.
- Joshi, Chaitanya K., Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent**, “Learning the Travelling Salesperson Problem Requires Rethinking Generalization,” 2020.
- Karp, Richard M**, “Reducibility among combinatorial problems,” in “Complexity of computer computations,” Springer, 1972, pp. 85–103.
- Reinelt, Gerhard**, “TSPLIB—A traveling salesman problem library,” *ORSA journal on computing*, 1991, 3 (4), 376–384.
- , *The traveling salesman: computational solutions for TSP applications*, Vol. 840, Springer, 2003.
- Sharma, Ross Barnowski Nikita**, “NetworkX,” <https://networkx.org/documentation/stable/tutorial.html>.
- Smith, Harriet R, Brendon Hemily, and Miomir Ivanovic**, “Transit signal priority (TSP): A planning and implementation handbook,” 2005.
- Volgenant, Ton and Roy Jonker**, “The symmetric traveling salesman problem and edge exchanges in minimal 1-trees,” *European Journal of Operational Research*, 1983, 12 (4), 394–403.

A 附录

附录中存放过长的代码和尺寸过大的图片。

A.1 程序代码

```
# 获取数据集
30 import requests,os
    from lxml import etree
32 from urllib.parse import urlparse
    url = 'http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html'
34 resp = requests.get(url)
    text = resp.text
36 html = etree.HTML(text)
    parse = urlparse(url)
38 root = 'http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/'
    for a in html.xpath('//li/a'):
40         name = a.xpath('./@href')[0]
            path = os.path.join('res', name)
42         if os.path.exists(path):
                print('已存在%s' % name)
44             continue
            durl = root + name
46         dresp = requests.get(durl)
            with open(path, 'w') as fp:
48                 fp.write(dresp.text)
                print(name)
```

[

```
50 # 读取文件夹下所有.tsp文件
    import os
52 import pandas as pd
    path = "." #文件夹目录
54 files= os.listdir(path) #得到文件夹下的所有文件名称
    s = []
56 for file in files: #遍历文件夹
            if not os.path.isdir(file) and file[-4:-1]==".tsp" #判断是否是文件夹，不是文件夹
                才打开
58         df = pd.read_csv(file, sep=" ", header=None)
            s.append(df)
60 print(s) #打印结果
```

[

```

# TSP测试样例的读取
62 class TSP_DATA:
    '''用于读取数据、存放图的数据结构
64
    '''
66 # 初始化，读取数据，并获取表头的数据规约
def __init__(self, path):
68     """传入单个数据地址，读取并且加载数据的表头
    """
70     # 读取数据
    raw_data, name = self.read_any_file(path)
72     # # 无论如何名字都是第0行
    # self.name = raw_data.split(':')[1]
74     self.get_headline(raw_data)
    # 保证数据名称和文件名称相同
76     assert self.NAME == name
    # 保证是TSP问题的文件
78     assert 'TSP' in self.TYPE
    # 有数据区块
80     self.rawnum = self.spilt_raw_data(raw_data)

82 def split_raw_data_index(self, data_all):
    """ 返回数字数据，被关键字和EOF包裹。不做其他处理
84     EXPLICIT EDGE_WEIGHT_SECTION
    EUC_2D NODE_COORD_SECTION
86     """
    end_line = len(data_all)-2 # 去掉EOF
88     if self.EDGE_WEIGHT_TYPE == 'EXPLOFT':
        # 此时是n*1的
        beging_line = 0
        for i in range(10):
92             if 'EDGE_WEIGHT_SECTION' in data_all[0][i]:
                # 下一位开始是数据
94                 beging_line = i+1
                break
96             else:
                raise Exception("EDGE_WEIGHT_SECTION 关键字未找到")
98     string_data = data_all[0][beging_line]
    for h in range(beging_line+1,end_line+1):
100         string_data = string_data + data_all[0][h]
    string_data_split = string_data.strip().split(' ')
102     list_data = [int(i) for i in string_data_split]

```

```

104     else:
105         beging_line = 0
106         for i in range(10):
107             if 'NODE_COORD_SECTION' in data_all[0][i]:
108                 beging_line = i+1
109                 break
110         else:
111             raise Exception("NODE_COORD_SECTION 关键字未找到")
112         string_data = data_all[0][beging_line]
113         for h in range(beging_line+1,end_line+1):
114             string_data = string_data + data_all[0][h]
115             string_data_split = string_data.strip().split(' ')
116             list_data = [int(i) for i in string_data_split]
117         # 这里取了简便 #todo
118
119         return list_data
120
121 def get_headline(self, raw_data):
122     head_dict = {}
123     # 最多只有10个类型的表头
124     for i in range(10):
125         if ':' not in raw_data[0][i]:
126             break
127         head_dict[raw_data[0][i].split(':')[0]] = raw_data[0][i].split(':')[1]
128
129     # 从字典获取信息，去掉左右空格
130     self.NAME = head_dict['NAME'].strip()
131     self.TYPE = head_dict['TYPE'].strip()
132     self.DIMENSION = head_dict['DIMENSION'].strip()
133     self.EDGE_WEIGHT_TYPE = head_dict['EDGE_WEIGHT_TYPE'].strip()
134     if self.TYPE == "CVRP":
135         # 载货问题时，需要卡车容量
136         self.CAPACITY = head_dict['CAPACITY'].strip()
137     if self.EDGE_WEIGHT_TYPE == "EXPLICIT":
138         # 距离在raw code中指定的情况下，要给出邻接矩阵存放方式
139         # 比如 UPPER_DIAG_ROW 上三角矩阵
140         self.EDGE_WEIGHT_FORMAT = head_dict['EDGE_WEIGHT_FORMAT'].strip()
141
142     # 读取文件，不予处理
143
144 def read_any_file(self, thispath):
145     # 判断是否是数据集
146     if not os.path.isfile(thispath):
147         raise Exception('数据读取报错，地址非文件')
148     # 读取文件 raw数据

```



```

148 df = pd.read_csv(thispath, header=None)
# 文件名
150 if check_sys == 'Linux': # colab address
    name = thispath.split('/')[ -1].split('.')[0]
152 elif check_sys == 'Windows': # local address
    name = thispath.split('\\')[ -1].split('.')[0]
154 else:
    raise EnvironmentError
156 return df, name

# 根据 EDGE_WEIGHT_TYPE 距离计算方式, 返回距离
def cal_a_distance_by_adjacency_table(self, a, b):
'''
按照 城市编号 a → b , 给出两点之间的距离
可能的距离定义式: https://docs.rs/tspf/0.3.0/tspf/enum.WeightKind.html
Explicit 权重在数据文件中明确给出。直接读出来就是距离。
162 Euc2d 二维欧几里得距离。
164 Euc3d 三维欧几里得距离。
166 Max2d 二维马氏距离。???? 这是衡量矩阵的????
Max3d 三维马氏距离。
168 Man2d 二维曼哈顿距离。
Man3d 三维曼哈顿距离。
170 Ceil2d 向上取整的二维欧几里得距离。
Geo 地理距离。既欧氏距离
172 Att 问题的特殊距离函数att48和att532。
Xray1 版本 1 晶体学问题的特殊距离函数。
174 Xray2 版本 2 晶体学问题的特殊距离函数。
Custom 用户定义的距离函数。
176 Undefined 没有给出距离函数。
'''

178 if self.EDGE_WEIGHT_TYPE == "EUC_2D" or self.EDGE_WEIGHT_TYPE == "GEO":
    distance = (self.adjacency_table(a)[0] - self.adjacency_table(b)[0])**2 + \
180 (self.adjacency_table(a)[1] - self.adjacency_table(b)[1])**2
elif self.EDGE_WEIGHT_TYPE == "EUC_3D":
182 distance = abs((self.adjacency_table(a)[0] - self.adjacency_table(b)[0])**3 + \
    (self.adjacency_table(a)[1] - self.adjacency_table(b)[1])**3)
184 elif self.EDGE_WEIGHT_TYPE == "EXPLICIT":
    raise Exception("EXPLICIT距离必须由raw data的邻接矩阵给出")
186 elif self.EDGE_WEIGHT_TYPE == "MAN_2D":
    distance = abs((self.adjacency_table(a)[0] - self.adjacency_table(a)[1])**2 - \
188 (self.adjacency_table(b)[0] - self.adjacency_table(b)[1])**2)
elif self.EDGE_WEIGHT_TYPE == "MAN_3D":
190 distance = abs((self.adjacency_table(a)[0] - self.adjacency_table(a)[1])**3 - \
    (self.adjacency_table(b)[0] - self.adjacency_table(b)[1])**3)

```

```

192 elif self.EDGE_WEIGHT_TYPE == "CEIL_2D":
    distance = int((self.adjacency_table(a)[0]-self.adjacency_table(b)[0])**2+\
194 (self.adjacency_table(a)[1]-self.adjacency_table(b)[1])**2) + 1
    else: # 都不符合
196         raise Exception("此类型距离未定义")
    return distance

198
# 根据 TYPE 进行判断, 返回邻接矩阵
200 def get_adjacency_matrix(self):
    '''当确认矩阵储存方式 EDGE_WEIGHT_TYPE 为 邻接矩阵 EXPLICIT 时
202     self.EDGE_WEIGHT_FORMAT 保存矩阵储存格式
    self.rawnum 保存raw data中的所有数字
204 矩阵存放形式
    Function      权重由WeightKind中所述的函数计算。
206    FullMatrix    权重以完整矩阵形式给出。对应FULL_MATRIX于 TSPLIB 中的值。
    UpperRow      权重在上三角矩阵中按行给出, 没有对角线条目。 对应UPPER_ROW于 TSPLIB
        中的值。
208    LowerRow      权重在下三角矩阵中给出, 按行排列, 没有对角线条目。 对应LOWE_ROW于
        TSPLIB 中的值。
    UpperDiagRow   权重在上三角矩阵中给出, 按行与对角线条目。 对应UPPER_DIAG_ROW于
        TSPLIB 中的值。
210    LowerDiagRow  权重在下三角矩阵中给出, 按行与对角线条目。 对应LOWER_DIAG_ROW于
        TSPLIB 中的值。
    UpperCol       权重在上三角矩阵中给出, 没有对角线条目。 对应UPPER_COL于 TSPLIB 中的
        值。
212    LowerCol      权重在下三角矩阵中给出, 没有对角线条目。 对应LOWER_COL于 TSPLIB 中的
        值。
    UpperDiagCol   权重在上三角矩阵中给出, 与对角线条目并列。 对应UPPER_DIAG_COL于
        TSPLIB 中的值。
214    LowerDiagCol  权重在下三角矩阵中给出, 与对角线条目并列。 对应LOWER_DIAG_COL于
        TSPLIB 中的值。
    Undefined      没有说明如何存储权重。
216    '''
    matrix = []
218    # 上三角, 包含对角全0
    if self.EDGE_WEIGHT_FORMAT == "UPPER_DIAG_ROW":
220        # 数据长度应该是: d + d-1 + d-2 +...+ 1 = (1+d)*d/2
        d = self.DIMENSION
222        if not len(self.rawnum) == int((1+d)*d/2):
            raise Exception("获取的raw num数据量不对")
224        for i in range(d):
            # 由于自己有0, 第一行不填0, 第二行补一个, 第n行补n-1个
226            row = [0 for j in range(i)]
            # left = sum 0 d d-1 \

```

```

228     left = int( i*d-(0+i-1)*i/2 )
        right = int( (i+1)*d-(0+i)*(i+1)/2 )
230     # right = sum d d-1 d-2 ... = i*d-(0+i-1)*i/2
        row = row + self.rawnum[ left : right]
232     matrix.append(row)

234 # 上三角，不包含对角全0
elif self.EDGE_WEIGHT_FORMAT == "UPPER_ROW":
236     # 数据长度应该是: d + d-1 + d-2 +...+ 1 = (1+d)*d/2
        d = self.DIMENSION
238     if not len(self.rawnum) == int((1+d)*d/2-d):
        raise Exception("获取的raw num数据量不对")
240     for i in range(d):
        # 由于自己有0，第一行填1个0，第二行补2个，第n行补n个
242         row = [0 for j in range(i+1)]
        # left = sum 0 d-1 d-2
244         if i == 0:
            left = 0
246         else:
            left = int( d*(i-1) -(i-1)*i/2 )
248         # right = sum d-1 d-2 d-3 ... = int(d*i -(1+i)i/2)
            right = int( d*i -(1+i)*i/2 )
250         row = row + self.rawnum[ left : right]
            matrix.append(row)

252 elif self.EDGE_WEIGHT_FORMAT == "FULL_MATRIX":
254     # 数据长度应该是: d**2
        d = self.DIMENSION
256     if not len(self.rawnum) == int(d**2):
        raise Exception("获取的raw num数据量不对")
258     for i in range(d):
        row = self.rawnum[i:i+d]
260         matrix.append(row)

262 return matrix

```

[

A.2 图片

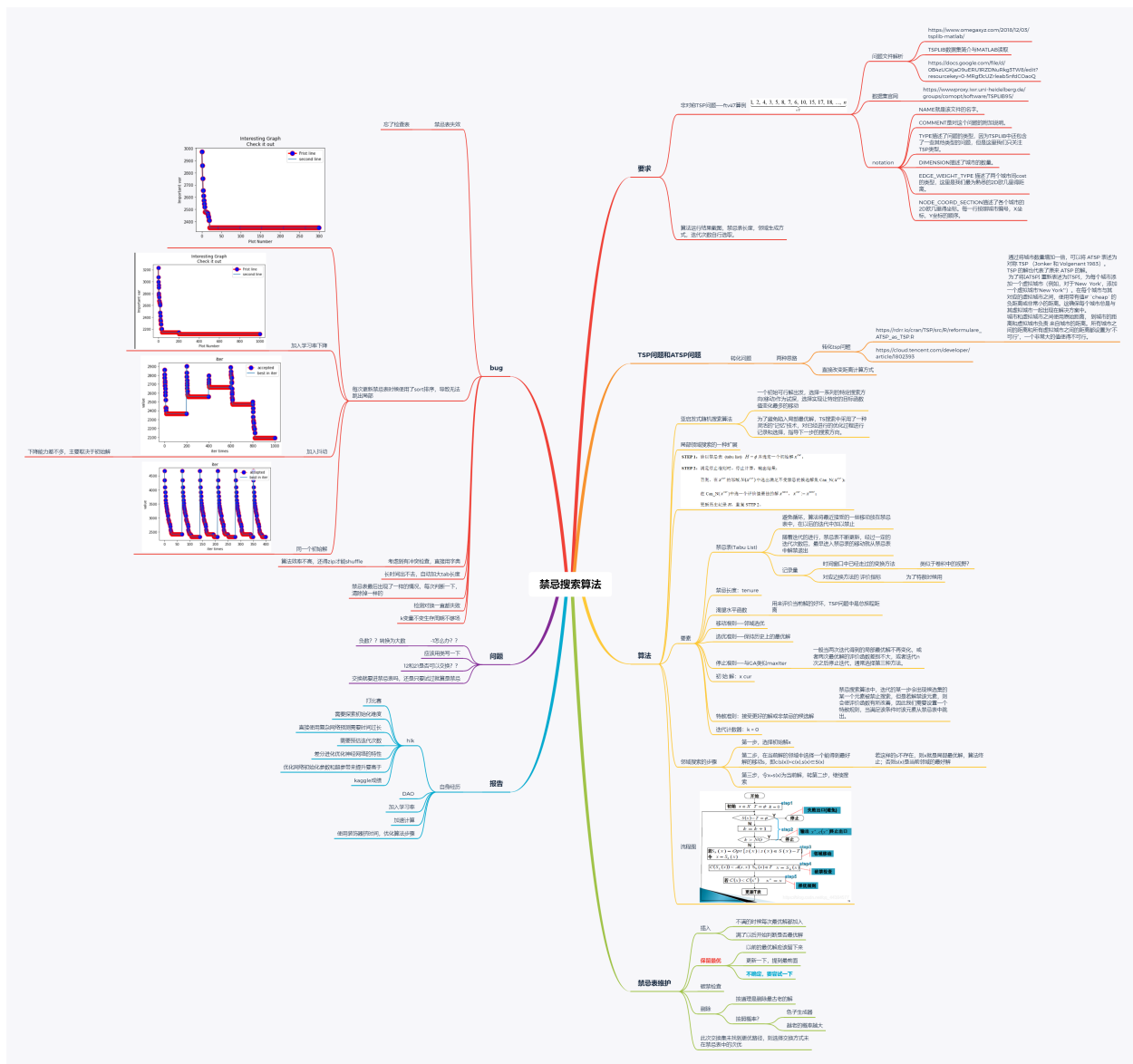


图 13: 禁忌搜索算法求解 TSP 问题思路

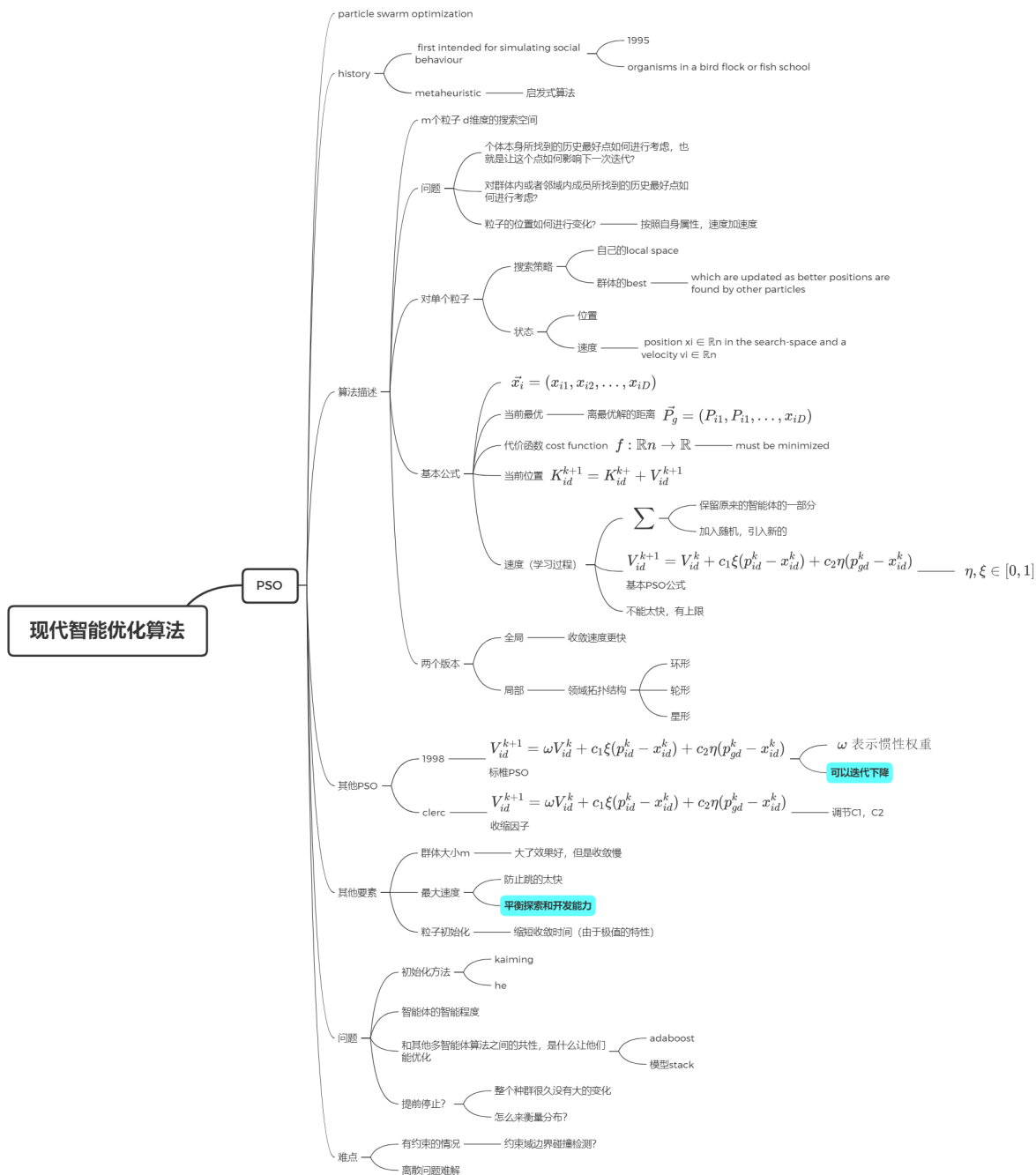


图 14: PSO 求解 TSP 问题思路