### 10. 禁忌搜索（TS）+ 0-1 整数规划（IP）组合模型案例题目

**题目：农村 5G 基站选址与覆盖优化问题**

* **问题背景**：某通信公司计划在某县的 30 个行政村部署 5G 基站，每个基站覆盖半径 1.5km，建设成本 50 万元 / 个。部分村庄人口少（≤500 人）但位于交通要道，需优先覆盖；部分村庄因地形（如山区）需多基站重叠覆盖。
* **问题描述**：需选择基站建设地点（从 50 个候选点中选取），目标包括：① 覆盖所有行政村（覆盖率 100%）；② 最小化建设总成本；③ 最大化人口覆盖率（≥95%）。变量为 0-1 型（1 表示建设，0 表示不建设），需满足山区村庄至少 2 个基站覆盖的约束。
* **数据情况**：提供各村的位置坐标、人口数量、地形类型（平原 / 山区），候选基站的位置坐标、建设成本、预计覆盖范围（受地形影响的修正系数），以及村庄之间的道路连接情况（影响信号传播）。

### 10. 禁忌搜索（TS）+ 0-1 整数规划（IP）求解农村 5G 基站选址与覆盖优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  import random  import pulp  from matplotlib.patches import Circle, Polygon  from matplotlib.colors import LinearSegmentedColormap  import copy  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  # 问题数据初始化  def initialize\_rural\_5g\_data(n\_villages=30, n\_candidates=15, coverage\_radius=3.0):  """  初始化农村5G基站选址问题数据  n\_villages: 村庄数量  n\_candidates: 候选基站位置数量  coverage\_radius: 基站覆盖半径（公里）  """  # 生成村庄坐标（模拟农村区域分布）  # 假设区域范围为10x10平方公里  villages = np.random.rand(n\_villages, 2) \* 10    # 村庄人口（影响优先级）  population = np.random.randint(100, 1501, size=n\_villages) # 100-1500人    # 村庄经济水平（1-5，影响权重）  economic\_level = np.random.randint(1, 6, size=n\_villages)    # 地形复杂度（影响建设成本和信号衰减）  # 0-平坦，1-丘陵，2-山区  terrain = np.random.choice([0, 1, 2], size=n\_villages, p=[0.5, 0.3, 0.2])    # 候选基站位置（尽量选择地势较高处）  candidates = np.random.rand(n\_candidates, 2) \* 10    # 基站建设成本（受地形影响）  base\_cost = 150 # 基础建设成本（万元）  build\_cost = []  for i in range(n\_candidates):  # 计算候选点周围地形影响  nearby\_terrain = []  for j in range(n\_villages):  dist = np.sqrt(np.sum((candidates[i] - villages[j])\*\*2))  if dist < coverage\_radius:  nearby\_terrain.append(terrain[j])    if not nearby\_terrain:  terrain\_factor = 1.0  else:  # 地形越复杂，建设成本越高  terrain\_factor = 1.0 + 0.3 \* np.mean(nearby\_terrain)    # 最终建设成本  cost = base\_cost \* terrain\_factor + np.random.normal(0, 10)  build\_cost.append(max(100, cost)) # 最低100万元    # 计算每个候选基站对村庄的覆盖情况（考虑地形衰减）  coverage = np.zeros((n\_candidates, n\_villages)) # 1表示可覆盖，0表示不可  signal\_strength = np.zeros((n\_candidates, n\_villages)) # 信号强度（0-1）    for i in range(n\_candidates):  for j in range(n\_villages):  # 计算距离  dist = np.sqrt(np.sum((candidates[i] - villages[j])\*\*2))    # 距离衰减  if dist <= coverage\_radius:  # 距离越近信号越强  dist\_factor = 1 - (dist / coverage\_radius)    # 地形衰减  terrain\_loss = {0: 0.1, 1: 0.3, 2: 0.5}[terrain[j]]    # 最终信号强度  strength = dist\_factor \* (1 - terrain\_loss)  signal\_strength[i, j] = strength    # 信号强度大于0.3视为有效覆盖  if strength > 0.3:  coverage[i, j] = 1    # 基站间干扰系数（距离越近干扰越大）  interference = np.zeros((n\_candidates, n\_candidates))  for i in range(n\_candidates):  for j in range(n\_candidates):  if i != j:  dist = np.sqrt(np.sum((candidates[i] - candidates[j])\*\*2))  # 距离越近，干扰系数越大（0-1）  interference[i, j] = max(0, 1 - dist / (2 \* coverage\_radius))    return {  'villages': villages,  'population': population,  'economic\_level': economic\_level,  'terrain': terrain,  'candidates': candidates,  'build\_cost': np.array(build\_cost),  'coverage': coverage,  'signal\_strength': signal\_strength,  'interference': interference,  'coverage\_radius': coverage\_radius,  'n\_villages': n\_villages,  'n\_candidates': n\_candidates,  'annual\_operation\_cost': 15 # 年运维成本（万元/基站）  }  # 0-1整数规划模型（固定基站集合下的优化）  def ip\_coverage\_optimization(data, selected\_candidates):  """  在选定的候选基站集合中，使用0-1整数规划优化最终选址  目标：最小化总成本，最大化覆盖质量  """  n\_selected = len(selected\_candidates)  n\_villages = data['n\_villages']    if n\_selected == 0:  return [], float('inf')    # 创建问题实例（最小化成本）  prob = pulp.LpProblem("5G\_Base\_Station\_Optimization", pulp.LpMinimize)    # 决策变量：是否建设基站（1-建设，0-不建设）  x = pulp.LpVariable.dicts(  "Build", range(n\_selected), lowBound=0, upBound=1, cat=pulp.LpInteger  )    # 辅助变量：村庄是否被覆盖  y = pulp.LpVariable.dicts(  "Covered", range(n\_villages), lowBound=0, upBound=1, cat=pulp.LpInteger  )    # 1. 目标函数：最小化建设成本 + 未覆盖惩罚  total\_cost = 0    # 建设成本  for i in range(n\_selected):  idx = selected\_candidates[i]  total\_cost += x[i] \* data['build\_cost'][idx]    # 未覆盖惩罚（基于村庄重要性）  # 重要性 = 人口 \* 经济水平  importance = data['population'] \* data['economic\_level']  max\_importance = np.max(importance)    for j in range(n\_villages):  # 惩罚系数：相当于覆盖100人的成本  penalty = max\_importance \* 0.1  total\_cost += (1 - y[j]) \* penalty    prob += total\_cost, "Total\_Cost"    # 2. 约束条件  # 2.1 覆盖约束：至少一个基站覆盖村庄  for j in range(n\_villages):  coverage\_expr = 0  for i in range(n\_selected):  candidate\_idx = selected\_candidates[i]  coverage\_expr += x[i] \* data['coverage'][candidate\_idx, j]    # 如果有基站覆盖，则y[j]必须为1  prob += coverage\_expr >= y[j], f"Coverage\_Constraint\_{j}"    # 2.2 干扰约束：避免过近的基站同时建设  for i in range(n\_selected):  for k in range(i+1, n\_selected):  i\_idx = selected\_candidates[i]  k\_idx = selected\_candidates[k]    # 干扰系数大于0.5的基站不能同时建设  if data['interference'][i\_idx, k\_idx] > 0.5:  prob += x[i] + x[k] <= 1, f"Interference\_Constraint\_{i}\_{k}"    # 2.3 预算约束（假设总预算）  total\_budget = 1200 # 总预算（万元）  prob += pulp.lpSum([x[i] \* data['build\_cost'][selected\_candidates[i]] for i in range(n\_selected)]) <= total\_budget, "Budget\_Constraint"    # 求解  prob.solve(pulp.PULP\_CBC\_CMD(msg=0)) # 静音模式    # 提取结果  selected = [selected\_candidates[i] for i in range(n\_selected) if x[i].varValue > 0.9]  total\_cost = pulp.value(prob.objective)    return selected, total\_cost  # 禁忌搜索算法  class TabuSearch:  def \_\_init\_\_(self, data, tabu\_size=10, max\_iter=50, neighbor\_size=5):  """  禁忌搜索算法初始化  tabu\_size: 禁忌表大小  max\_iter: 最大迭代次数  neighbor\_size: 每次生成的邻域解数量  """  self.data = data  self.tabu\_size = tabu\_size  self.max\_iter = max\_iter  self.neighbor\_size = neighbor\_size    self.n\_candidates = data['n\_candidates']  self.tabu\_list = [] # 禁忌表，存储被禁忌的操作  self.best\_solution = []  self.best\_cost = float('inf')  self.history = [] # 记录优化历史    def initialize\_solution(self):  """生成初始解：随机选择部分候选基站"""  # 初始选择30%-50%的候选点  n\_selected = int(self.n\_candidates \* (0.3 + 0.2 \* random.random()))  return random.sample(range(self.n\_candidates), n\_selected)    def evaluate\_solution(self, solution):  """评估解的质量：调用整数规划模型"""  if not solution:  return float('inf')    # 调用IP模型进行优化  optimized\_sol, cost = ip\_coverage\_optimization(self.data, solution)  return optimized\_sol, cost    def generate\_neighbors(self, current\_solution):  """生成邻域解"""  neighbors = []    # 当前选中的基站集合  current\_set = set(current\_solution)  # 未选中的基站集合  unselected = set(range(self.n\_candidates)) - current\_set    # 生成多种邻域解  for \_ in range(self.neighbor\_size):  # 随机选择操作类型：添加、删除或替换  operation = random.choice(['add', 'remove', 'swap'])    if operation == 'add' and unselected:  # 添加一个基站  to\_add = random.choice(list(unselected))  new\_sol = current\_solution + [to\_add]    elif operation == 'remove' and len(current\_solution) > 1:  # 删除一个基站  to\_remove = random.choice(current\_solution)  new\_sol = [s for s in current\_solution if s != to\_remove]    elif operation == 'swap' and unselected and current\_solution:  # 替换一个基站  to\_remove = random.choice(current\_solution)  to\_add = random.choice(list(unselected))  new\_sol = [s for s in current\_solution if s != to\_remove] + [to\_add]    else:  # 无法执行所选操作，默认选择swap  if unselected and current\_solution:  to\_remove = random.choice(current\_solution)  to\_add = random.choice(list(unselected))  new\_sol = [s for s in current\_solution if s != to\_remove] + [to\_add]  elif unselected:  to\_add = random.choice(list(unselected))  new\_sol = current\_solution + [to\_add]  else:  to\_remove = random.choice(current\_solution)  new\_sol = [s for s in current\_solution if s != to\_remove]    # 去重并排序（便于比较）  new\_sol = sorted(list(set(new\_sol)))  neighbors.append(new\_sol)    return neighbors    def is\_tabu(self, solution, current\_solution):  """判断解是否为禁忌"""  # 计算与当前解的差异（操作）  current\_set = set(current\_solution)  new\_set = set(solution)    added = new\_set - current\_set  removed = current\_set - new\_set    # 操作表示为(添加的基站, 删除的基站)  operation = (tuple(sorted(added)), tuple(sorted(removed)))    return operation in self.tabu\_list    def update\_tabu\_list(self, solution, current\_solution):  """更新禁忌表"""  current\_set = set(current\_solution)  new\_set = set(solution)    added = new\_set - current\_set  removed = current\_set - new\_set    operation = (tuple(sorted(added)), tuple(sorted(removed)))    # 添加新操作到禁忌表  self.tabu\_list.append(operation)    # 如果超出禁忌表大小，移除最早的  if len(self.tabu\_list) > self.tabu\_size:  self.tabu\_list.pop(0)    def optimize(self):  """执行禁忌搜索优化"""  # 初始解  current\_solution = self.initialize\_solution()  current\_solution, current\_cost = self.evaluate\_solution(current\_solution)    # 更新最优解  self.best\_solution = current\_solution  self.best\_cost = current\_cost  self.history.append(self.best\_cost)    print(f"初始解: 选择{len(current\_solution)}个基站, 成本{current\_cost:.2f}万元")    # 迭代优化  for iter in range(self.max\_iter):  # 生成邻域解  neighbors = self.generate\_neighbors(current\_solution)    # 评估所有邻域解  neighbor\_results = []  for neighbor in neighbors:  # 避免重复解  if neighbor == current\_solution:  continue    # 评估邻域解  opt\_neighbor, cost = self.evaluate\_solution(neighbor)  neighbor\_results.append((neighbor, opt\_neighbor, cost))    # 按成本排序  neighbor\_results.sort(key=lambda x: x[2])    # 选择下一个解  next\_solution = None  next\_opt\_solution = None  next\_cost = float('inf')    for nr in neighbor\_results:  neighbor, opt\_neighbor, cost = nr    # 检查是否为禁忌解  if not self.is\_tabu(neighbor, current\_solution):  next\_solution = neighbor  next\_opt\_solution = opt\_neighbor  next\_cost = cost  break  else:  # 特赦准则：如果禁忌解明显优于当前最优解，则接受  if cost < self.best\_cost \* 0.9:  next\_solution = neighbor  next\_opt\_solution = opt\_neighbor  next\_cost = cost  print(f"触发特赦准则: 成本{cost:.2f}优于当前最优{self.best\_cost:.2f}")  break    # 如果没有找到合适的解，随机选择一个  if next\_solution is None and neighbor\_results:  next\_solution, next\_opt\_solution, next\_cost = neighbor\_results[0]    # 更新当前解  current\_solution = next\_solution  current\_cost = next\_cost    # 更新最优解  if current\_cost < self.best\_cost:  self.best\_solution = next\_opt\_solution  self.best\_cost = current\_cost  print(f"迭代{iter+1}: 找到更优解, 选择{len(self.best\_solution)}个基站, 成本{self.best\_cost:.2f}万元")    # 更新禁忌表  self.update\_tabu\_list(current\_solution, [s for s in current\_solution if s not in next\_solution or next\_solution])    # 记录历史  self.history.append(self.best\_cost)    print(f"优化完成: 最优解选择{len(self.best\_solution)}个基站, 总成本{self.best\_cost:.2f}万元")  return self.best\_solution, self.best\_cost  # 结果分析与可视化  def analyze\_and\_visualize(data, best\_solution):  """分析并可视化基站选址结果"""  # 1. 计算覆盖情况  covered\_villages = set()  coverage\_detail = []    for station in best\_solution:  for village in range(data['n\_villages']):  if data['coverage'][station, village] == 1:  covered\_villages.add(village)  coverage\_detail.append({  'village': village,  'station': station,  'signal': data['signal\_strength'][station, village]  })    coverage\_rate = len(covered\_villages) / data['n\_villages'] \* 100  print(f"村庄覆盖率: {coverage\_rate:.2f}%")    # 2. 计算建设总成本  total\_build\_cost = sum(data['build\_cost'][s] for s in best\_solution)  print(f"总建设成本: {total\_build\_cost:.2f}万元")  print(f"年均</doubaocanvas> |