### 7. 多目标遗传算法（MOGA）+ 层次分析法（AHP）组合模型案例题目

**题目：新能源汽车工厂选址与产能规划问题**

* **问题背景**：某车企计划在全国 5 个候选城市（如武汉、重庆、广州）中选择 2 个建设新能源汽车工厂，需考虑原材料运输成本、劳动力资源、政策补贴、环保要求等因素。不同城市的土地价格、供应链成熟度差异显著，且工厂投产后需确定各厂的年产能（5-20 万辆）。
* **问题描述**：需选择最优选址方案并分配产能，目标包括：① 最小化年均总成本（土地 + 运输 + 人力）；② 最大化政策补贴总额（部分城市对新能源项目补贴 20%-30%）；③ 最小化碳排放（≤50 万吨 / 年）。需结合专家意见（如供应链稳定性权重）进行多目标决策。
* **数据情况**：提供各候选城市的基础数据：土地价格（万元 / 亩）、劳动力成本（元 / 小时）、主要零部件供应商的距离及运输成本（元 / 辆）、当地环保排放标准、政策补贴细则，以及不同产能下的固定成本和变动成本核算表。

### 7. 多目标遗传算法（MOGA）+ 层次分析法（AHP）新能源汽车工厂选址与产能规划代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import random  from scipy.stats import rankdata  from matplotlib.patches import Patch  import seaborn as sns  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  # 候选城市数据初始化  def initialize\_candidate\_cities():  """初始化10个候选城市的各项指标数据"""  cities = {  0: {'name': '武汉', 'land\_price': 85, 'labor\_cost': 28, 'transport': 1.2,  'subsidy': 0.25, 'carbon\_limit': 45, 'supply\_chain': 0.85, 'market': 0.82},  1: {'name': '重庆', 'land\_price': 65, 'labor\_cost': 22, 'transport': 1.4,  'subsidy': 0.3, 'carbon\_limit': 50, 'supply\_chain': 0.78, 'market': 0.75},  2: {'name': '广州', 'land\_price': 110, 'labor\_cost': 32, 'transport': 1.0,  'subsidy': 0.22, 'carbon\_limit': 42, 'supply\_chain': 0.90, 'market': 0.90},  3: {'name': '西安', 'land\_price': 55, 'labor\_cost': 20, 'transport': 1.6,  'subsidy': 0.35, 'carbon\_limit': 55, 'supply\_chain': 0.72, 'market': 0.68},  4: {'name': '上海', 'land\_price': 130, 'labor\_cost': 38, 'transport': 0.9,  'subsidy': 0.18, 'carbon\_limit': 38, 'supply\_chain': 0.95, 'market': 0.95},  5: {'name': '成都', 'land\_price': 70, 'labor\_cost': 25, 'transport': 1.3,  'subsidy': 0.28, 'carbon\_limit': 48, 'supply\_chain': 0.80, 'market': 0.78},  6: {'name': '沈阳', 'land\_price': 60, 'labor\_cost': 23, 'transport': 1.5,  'subsidy': 0.32, 'carbon\_limit': 52, 'supply\_chain': 0.70, 'market': 0.65},  7: {'name': '杭州', 'land\_price': 100, 'labor\_cost': 30, 'transport': 1.1,  'subsidy': 0.20, 'carbon\_limit': 40, 'supply\_chain': 0.88, 'market': 0.85},  8: {'name': '深圳', 'land\_price': 120, 'labor\_cost': 35, 'transport': 0.95,  'subsidy': 0.20, 'carbon\_limit': 35, 'supply\_chain': 0.92, 'market': 0.92},  9: {'name': '合肥', 'land\_price': 75, 'labor\_cost': 26, 'transport': 1.35,  'subsidy': 0.30, 'carbon\_limit': 47, 'supply\_chain': 0.83, 'market': 0.70}  }  return cities  # 层次分析法（AHP）计算权重  def ahp\_weight\_calculation():  """  基于专家判断矩阵计算各目标权重  目标包括：成本、政策补贴、碳排放、供应链完整性、市场潜力  """  # 构建判断矩阵（5x5）  # 行/列：0-成本，1-补贴，2-碳排放，3-供应链，4-市场  judge\_matrix = np.array([  [1, 3, 5, 2, 2], # 成本  [1/3, 1, 3, 1/2, 1/2], # 补贴  [1/5, 1/3, 1, 1/3, 1/3], # 碳排放  [1/2, 2, 3, 1, 1], # 供应链  [1/2, 2, 3, 1, 1] # 市场  ])    # 计算权重（特征值法）  eigvals, eigvecs = np.linalg.eig(judge\_matrix)  max\_idx = np.argmax(eigvals)  weights = eigvecs[:, max\_idx].real  weights = np.abs(weights) # 确保非负  weights /= np.sum(weights) # 归一化    print("AHP计算的各目标权重:")  print(f"成本权重: {weights[0]:.4f}")  print(f"补贴权重: {weights[1]:.4f}")  print(f"碳排放权重: {weights[2]:.4f}")  print(f"供应链权重: {weights[3]:.4f}")  print(f"市场权重: {weights[4]:.4f}")    return weights  # 多目标遗传算法参数设置  def set\_moga\_parameters():  """设置多目标遗传算法参数"""  return {  'pop\_size': 80, # 种群规模  'generations': 150, # 迭代次数  'mutation\_rate': 0.25, # 变异率  'n\_objectives': 5, # 目标函数数量  'n\_cities': 10, # 候选城市数量  'n\_plants': 2, # 计划建设工厂数量  'min\_capacity': 5, # 最小产能（万辆/年）  'max\_capacity': 25, # 最大产能（万辆/年）  'total\_demand': 35 # 总需求（万辆/年）  }  # 初始化种群  def initialize\_population(params):  """初始化遗传算法种群"""  pop = []  for \_ in range(params['pop\_size']):  # 随机选择n\_plants个不同的城市  selected\_cities = random.sample(range(params['n\_cities']), params['n\_plants'])    # 分配产能（总和接近总需求）  capacities = []  remaining = params['total\_demand']  for i in range(params['n\_plants'] - 1):  cap = random.uniform(params['min\_capacity'], min(params['max\_capacity'], remaining - params['min\_capacity']))  capacities.append(cap)  remaining -= cap  capacities.append(remaining) # 最后一个工厂承担剩余产能    # 确保产能在有效范围内  capacities = [np.clip(c, params['min\_capacity'], params['max\_capacity']) for c in capacities]    # 构建个体：(城市1, 产能1, 城市2, 产能2, ...)  individual = []  for i in range(params['n\_plants']):  individual.extend([selected\_cities[i], capacities[i]])    pop.append(tuple(individual))    return pop  # 目标函数计算  def calculate\_objectives(individual, cities, params):  """计算个体的各项目标函数值"""  n\_plants = params['n\_plants']  total\_cost = 0  total\_subsidy = 0  total\_carbon = 0  avg\_supply\_chain = 0  avg\_market = 0    # 工厂固定成本参数（亿元）  fixed\_cost\_coef = 2.5 # 每万辆产能的固定成本系数  var\_cost\_coef = 0.8 # 每万辆产能的可变成本系数    for i in range(n\_plants):  city\_idx = individual[2\*i]  capacity = individual[2\*i + 1]  city = cities[city\_idx]    # 1. 成本目标（土地+固定+可变+运输）  land\_cost = city['land\_price'] \* capacity \* 0.01 # 土地成本（亿元）  fixed\_cost = fixed\_cost\_coef \* capacity # 固定成本（亿元）  var\_cost = var\_cost\_coef \* capacity # 可变成本（亿元）  transport\_cost = city['transport'] \* capacity \* 0.1 # 运输成本（亿元）  total\_cost += land\_cost + fixed\_cost + var\_cost + transport\_cost    # 2. 补贴目标（投资额×补贴率）  investment = land\_cost + fixed\_cost  total\_subsidy += investment \* city['subsidy']    # 3. 碳排放目标（万辆对应碳排放量）  carbon\_emission = np.minimum(capacity \* 0.8, city['carbon\_limit']) # 假设每万辆排放0.8万吨  total\_carbon += carbon\_emission    # 4. 供应链完整性（平均值）  avg\_supply\_chain += city['supply\_chain']    # 5. 市场潜力（平均值）  avg\_market += city['market']    # 平均供应链和市场潜力  avg\_supply\_chain /= n\_plants  avg\_market /= n\_plants    # 返回目标值（成本、-补贴、碳排放、-供应链、-市场，统一为最小化问题）  return (  total\_cost,  -total\_subsidy,  total\_carbon,  -avg\_supply\_chain,  -avg\_market  )  # 非支配排序  def non\_dominated\_sorting(pop, cities, params):  """对种群进行非支配排序"""  n = len(pop)  objectives = [calculate\_objectives(ind, cities, params) for ind in pop]    # 初始化支配关系  dominated = [[] for \_ in range(n)] # 被该个体支配的个体列表  counts = np.zeros(n) # 支配该个体的个体数量  ranks = np.zeros(n) # 个体的排序等级    # 计算支配关系  for i in range(n):  for j in range(n):  if i != j:  # 检查i是否支配j  if all(o\_i <= o\_j for o\_i, o\_j in zip(objectives[i], objectives[j])):  dominated[i].append(j)  counts[j] += 1    # 分配排序等级  rank = 0  while np.any(counts == 0):  for i in range(n):  if counts[i] == 0:  ranks[i] = rank  counts[i] = -1 # 标记为已处理  # 减少被其支配的个体的计数  for j in dominated[i]:  counts[j] -= 1  rank += 1    return ranks  # 拥挤度计算  def crowding\_distance(objectives):  """计算种群中个体的拥挤度"""  n = len(objectives)  n\_obj = len(objectives[0])  dist = np.zeros(n)    for m in range(n\_obj):  # 按第m个目标排序  sorted\_indices = np.argsort([obj[m] for obj in objectives])  sorted\_objs = [objectives[i][m] for i in sorted\_indices]    # 边界个体的拥挤度设为无穷大  dist[sorted\_indices[0]] = np.inf  dist[sorted\_indices[-1]] = np.inf    # 计算中间个体的拥挤度  if sorted\_objs[-1] != sorted\_objs[0]: # 避免除以零  for i in range(1, n-1):  dist[sorted\_indices[i]] += (sorted\_objs[i+1] - sorted\_objs[i-1]) / (sorted\_objs[-1] - sorted\_objs[0])    return dist  # 选择操作（锦标赛选择）  def selection(pop, ranks, distances, params):  """锦标赛选择操作"""  selected = []  while len(selected) < params['pop\_size']:  # 随机选择两个个体  a, b = random.sample(range(len(pop)), 2)    # 选择排序等级低的个体，等级相同则选择拥挤度高的  if ranks[a] < ranks[b] or (ranks[a] == ranks[b] and distances[a] > distances[b]):  selected.append(pop[a])  else:  selected.append(pop[b])    return selected  # 交叉操作  def crossover(p1, p2, params):  """交叉操作生成子代个体"""  n\_plants = params['n\_plants']  # 随机选择交叉点  cross\_point = random.randint(1, 2\*n\_plants - 1)    # 生成子代  child1 = p1[:cross\_point] + p2[cross\_point:]  child2 = p2[:cross\_point] + p1[cross\_point:]    # 确保城市不重复  for child in [child1, child2]:  cities\_in\_child = [child[2\*i] for i in range(n\_plants)]  # 检查重复城市  if len(set(cities\_in\_child)) < n\_plants:  # 替换重复城市  all\_cities = set(range(params['n\_cities']))  used\_cities = set(cities\_in\_child)  available\_cities = list(all\_cities - used\_cities)    # 找出重复的位置  for i in range(n\_plants):  if cities\_in\_child.count(cities\_in\_child[i]) > 1:  # 替换为可用城市  if available\_cities:  child[2\*i] = available\_cities.pop(0)    return tuple(child1), tuple(child2)  # 变异操作  def mutate(individual, params, mutation\_rate=None):  """变异操作"""  if mutation\_rate is None:  mutation\_rate = params['mutation\_rate']    n\_plants = params['n\_plants']  mutated = list(individual)    # 城市变异  if random.random() < mutation\_rate:  plant\_idx = random.randint(0, n\_plants - 1)  current\_city = mutated[2\*plant\_idx]  # 选择不同的城市  other\_cities = [c for c in range(params['n\_cities']) if c != current\_city]  # 确保不与其他工厂城市重复  existing\_cities = [mutated[2\*i] for i in range(n\_plants) if i != plant\_idx]  possible\_cities = [c for c in other\_cities if c not in existing\_cities]  if possible\_cities:  mutated[2\*plant\_idx] = random.choice(possible\_cities)    # 产能变异  if random.random() < mutation\_rate:  plant\_idx = random.randint(0, n\_plants - 1)  # 小幅调整产能  current\_cap = mutated[2\*plant\_idx + 1]  change = random.uniform(-2, 2) # 产能变化范围  new\_cap = current\_cap + change  # 确保在有效范围内  new\_cap = np.clip(new\_cap, params['min\_capacity'], params['max\_capacity'])  mutated[2\*plant\_idx + 1] = new\_cap    # 调整其他工厂产能以保持总产能稳定  if n\_plants > 1:  other\_idx = random.choice([i for i in range(n\_plants) if i != plant\_idx])  mutated[2\*other\_idx + 1] -= change  mutated[2\*other\_idx + 1] = np.clip(  mutated[2\*other\_idx + 1],  params['min\_capacity'],  params['max\_capacity']  )    return tuple(mutated)  # 多目标遗传算法主函数  def moga\_optimization(cities, params, weights):  """执行多目标遗传算法优化"""  # 初始化种群  pop = initialize\_population(params)  best\_solutions = []  best\_scores = []    for gen in range(params['generations']):  # 计算非支配排序和拥挤度  ranks = non\_dominated\_sorting(pop, cities, params)  objectives = [calculate\_objectives(ind, cities, params) for ind in pop]  distances = crowding\_distance(objectives)    # 选择操作  selected = selection(pop, ranks, distances, params)    # 交叉操作  offspring = []  for i in range(0, params['pop\_size'], 2):  p1 = selected[i]  p2 = selected[i+1] if i+1 < params['pop\_size'] else selected[0]  c1,</doubaocanvas> |