# 二、15种组合优化模型的数学建模案例题目

### 1. 遗传算法（GA）+ 模拟退火（SA）组合模型案例题目

**题目：城市生鲜电商配送路径优化问题**

* **问题背景**：某生鲜电商企业在一线城市运营，需每日从 3 个仓库向 50 个社区站点配送生鲜产品。生鲜产品保质期短（常温下仅 12 小时），且运输过程中需维持 0-4℃冷链，配送成本高（含冷链能耗、超时损耗）。当前配送路线常因交通拥堵、重复绕路导致成本超支，客户投诉率达 15%。
* **问题描述**：需优化从各仓库到社区站点的配送路线，目标包括：① 最小化总配送时间（≤8 小时）；② 最小化总运输成本（含油费、冷链能耗）；③ 确保每个站点的生鲜损耗率≤5%。需考虑的约束：每个配送员日均工作时间≤8 小时，每辆车最大载重 500kg，早高峰（7:00-9:00）部分路段限行。
* **数据情况**：提供 3 个月的历史数据，包括：各仓库与站点的距离矩阵（单位：km）、各时段路段平均通行时间（含拥堵系数）、每辆车的冷链能耗率（元 /km）、不同超时时长对应的生鲜损耗率（如超时 1 小时损耗率增加 2%）、各站点的日订单量（10-80kg）。

### 1. 遗传算法（GA）+ 模拟退火（SA）求解城市生鲜电商配送路径优化代码

|  |
| --- |
| import numpy as np  import random  import matplotlib.pyplot as plt  from math import sqrt  # 数据初始化  np.random.seed(42)  warehouses = np.array([[0, 0], [10, 10], [20, 5]]) # 3个仓库坐标  stations = np.random.rand(50, 2) \* 30 # 50个社区站点坐标  all\_points = np.vstack((warehouses, stations)) # 合并所有点  n\_stations = len(stations)  demand = np.random.randint(10, 81, size=n\_stations) # 各站点需求量(kg)  max\_load = 500 # 车辆最大载重(kg)  # 计算距离矩阵  def distance\_matrix(points):  n = len(points)  dist = np.zeros((n, n))  for i in range(n):  for j in range(n):  dist[i][j] = sqrt(((points[i] - points[j])\*\*2).sum())  return dist  dist\_matrix = distance\_matrix(all\_points)  # 遗传算法参数  pop\_size = 50  generations = 100  mutation\_rate = 0.1  sa\_temp = 100.0  sa\_cool = 0.95  # 初始化种群（每个个体为从仓库出发的站点访问顺序）  def init\_population(warehouse\_idx, n\_stations, pop\_size):  pop = []  for \_ in range(pop\_size):  route = [warehouse\_idx] + random.sample(range(3, 3 + n\_stations), n\_stations)  pop.append(route)  return pop  # 适应度函数（综合成本、时间、损耗）  def fitness(route):  total\_dist = 0  total\_demand = 0  for i in range(len(route)-1):  total\_dist += dist\_matrix[route[i]][route[i+1]]  if i > 0: # 站点需求（跳过仓库）  total\_demand += demand[route[i]-3]    # 超载惩罚  overload\_penalty = max(0, total\_demand - max\_load) \* 100  # 时间惩罚（假设1km/分钟，总时间应≤480分钟）  time\_penalty = max(0, total\_dist - 480) \* 50  # 损耗率（超时1小时损耗率增加2%）  loss\_rate = max(0, (total\_dist - 480)/60) \* 0.02  loss\_penalty = loss\_rate \* 10000    return 1 / (total\_dist + overload\_penalty + time\_penalty + loss\_penalty)  # 选择操作（轮盘赌）  def select(pop, fitnesses):  total\_fit = sum(fitnesses)  probs = [f / total\_fit for f in fitnesses]  return random.choices(pop, probs, k=2)  # 交叉操作（部分映射交叉）  def crossover(p1, p2):  size = len(p1)  a, b = random.sample(range(1, size-1), 2)  if a > b:  a, b = b, a  child = [None] \* size  child[a:b+1] = p1[a:b+1]  ptr = b + 1  for gene in p2[b+1:] + p2[1:b+1]:  if gene not in child:  if ptr >= size:  ptr = 1  child[ptr] = gene  ptr += 1  child[0] = p1[0] # 保持仓库起点不变  return child  # 变异操作（交换变异）  def mutate(route, rate):  if random.random() < rate:  a, b = random.sample(range(1, len(route)), 2)  route[a], route[b] = route[b], route[a]  return route  # 模拟退火接受准则  def sa\_accept(old\_fit, new\_fit, temp):  if new\_fit > old\_fit:  return True  else:  prob = np.exp((new\_fit - old\_fit) \* temp)  return random.random() < prob  # 主算法  def ga\_sa\_optimize():  # 分别从3个仓库出发的最优解  best\_routes = []  for warehouse in range(3):  pop = init\_population(warehouse, n\_stations, pop\_size)  best\_route = None  best\_fit = 0  temp = sa\_temp    for gen in range(generations):  fitnesses = [fitness(route) for route in pop]  current\_best\_idx = np.argmax(fitnesses)  current\_best\_fit = fitnesses[current\_best\_idx]  current\_best\_route = pop[current\_best\_idx]    if current\_best\_fit > best\_fit:  best\_fit = current\_best\_fit  best\_route = current\_best\_route    new\_pop = [current\_best\_route] # 精英保留    while len(new\_pop) < pop\_size:  p1, p2 = select(pop, fitnesses)  child = crossover(p1, p2)  child = mutate(child, mutation\_rate)  # 模拟退火选择  old\_fit = fitness(child)  mutated\_child = mutate(child.copy(), mutation\_rate)  new\_fit = fitness(mutated\_child)  if sa\_accept(old\_fit, new\_fit, temp):  child = mutated\_child  new\_pop.append(child)    pop = new\_pop  temp \*= sa\_cool    best\_routes.append((best\_route, best\_fit))    # 选择最优仓库出发的路线  best\_idx = np.argmax([f for (r, f) in best\_routes])  return best\_routes[best\_idx]  # 运行并可视化  best\_route, best\_fit = ga\_sa\_optimize()  print("最优配送路线（仓库索引+站点索引）：", best\_route)  print("最优适应度值：", best\_fit)  # 绘制路线图  plt.figure(figsize=(10, 8))  plt.scatter(warehouses[:,0], warehouses[:,1], c='red', s=100, label='仓库')  plt.scatter(stations[:,0], stations[:,1], c='blue', s=50, label='社区站点')  route\_points = all\_points[best\_route]  plt.plot(route\_points[:,0], route\_points[:,1], 'g-', linewidth=2)  plt.title('生鲜电商最优配送路线')  plt.legend()  plt.show() |