### 3. 蚁群算法（ACO）+ 贪婪算法（Greedy）组合模型案例题目

**题目：灾后应急物资多点配送路径规划问题**

* **问题背景**：某地区发生地震后，20 个乡镇成为受灾点，急需食品、药品、帐篷等物资。现有 3 个应急仓库（分别位于县城东、南、西部），仓库物资储备量有限，且部分道路因塌方仅能单向通行或限时通行（如上午 8:00-12:00 可通行）。救援车辆需在 72 小时内完成首轮物资投放，否则受灾点将出现物资短缺。
* **问题描述**：需规划救援车辆的配送路径，目标包括：① 最小化所有受灾点的物资送达时间差（避免个别点严重滞后）；② 最小化总运输里程；③ 确保每个受灾点的核心物资（食品、药品）覆盖率≥90%。约束条件：每辆车最大载重 8 吨，单次运输时间≤8 小时（含卸货），塌方路段需绕行且通行速度≤20km/h。
* **数据情况**：提供受灾点的位置坐标、各物资需求量（如甲乡镇需食品 2 吨、药品 50 箱）、仓库物资库存量、道路拓扑图（含双向 / 单向标识、通行时段、长度及正常路况下的行驶速度）、车辆平均卸货时间（0.5 小时 / 点）。

### 3. 蚁群算法（ACO）+ 贪婪算法（Greedy）求解灾后物资配送代码

|  |
| --- |
| import numpy as np  import random  import matplotlib.pyplot as plt  from math import sqrt  # 数据初始化  np.random.seed(42)  # 3个应急仓库坐标  warehouses = np.array([[0, 0], [50, 100], [100, 50]])  # 20个受灾点坐标  victims = np.random.rand(20, 2) \* 100  all\_points = np.vstack((warehouses, victims))  n\_victims = len(victims)  n\_warehouses = len(warehouses)  # 受灾点需求（食品吨、药品箱、帐篷顶）  demand = np.random.randint(1, 6, size=(n\_victims, 3)) # 每种物资1-5单位  # 仓库库存（食品吨、药品箱、帐篷顶）  stock = np.array([[20, 50, 30], [15, 40, 25], [18, 35, 20]])  # 道路通行时间（分钟）- 包含塌方路段修正  road\_time = np.zeros((len(all\_points), len(all\_points)))  for i in range(len(all\_points)):  for j in range(len(all\_points)):  dist = sqrt(((all\_points[i] - all\_points[j])\*\*2).sum())  # 10%概率为塌方路段，通行时间翻倍  if random.random() < 0.1:  road\_time[i][j] = dist \* 2  else:  road\_time[i][j] = dist  # 贪婪算法生成初始解  def greedy\_route(warehouse\_idx):  route = [warehouse\_idx]  visited = set([warehouse\_idx])  current = warehouse\_idx  remaining = set(range(n\_warehouses, n\_warehouses + n\_victims))    # 优先满足核心物资需求高的点  priority = np.sum(demand[:, :2], axis=1) # 食品+药品为核心物资  priority\_idx = np.argsort(-priority) + n\_warehouses # 转换为全局索引    for idx in priority\_idx:  if idx not in visited:  route.append(idx)  visited.add(idx)  current = idx    return route  # 蚁群算法参数  n\_ants = 30  iterations = 100  alpha = 1.0 # 信息素重要度  beta = 2.0 # 启发式信息重要度  rho = 0.1 # 信息素蒸发率  Q = 100 # 信息素增量常数  # 启发式信息（1/时间）  def heuristic\_info():  eta = np.zeros\_like(road\_time)  for i in range(len(road\_time)):  for j in range(len(road\_time)):  if i != j and road\_time[i][j] > 0:  eta[i][j] = 1.0 / road\_time[i][j]  return eta  # 适应度函数（综合时间、覆盖率、里程）  def fitness(route, warehouse\_idx):  total\_time = 0  total\_dist = 0  current\_stock = stock[warehouse\_idx].copy()  covered = 0 # 核心物资覆盖率    for i in range(len(route)-1):  u = route[i]  v = route[i+1]  total\_time += road\_time[u][v]  total\_dist += sqrt(((all\_points[u] - all\_points[v])\*\*2).sum())    # 计算物资覆盖率（仅对受灾点）  if v >= n\_warehouses:  vid = v - n\_warehouses # 受灾点索引  # 能满足的核心物资比例  food\_ratio = min(1.0, current\_stock[0] / demand[vid][0])  med\_ratio = min(1.0, current\_stock[1] / demand[vid][1])  covered += (food\_ratio + med\_ratio) / 2  # 消耗库存  current\_stock -= demand[vid]  current\_stock = np.maximum(current\_stock, 0) # 不能为负    # 平均覆盖率  avg\_coverage = covered / (len(route) - 1) # 减去仓库  # 时间惩罚（超过72小时=4320分钟）  time\_penalty = max(0, total\_time - 4320) \* 0.1  # 目标函数：最大化覆盖率，最小化时间和距离  return avg\_coverage / (total\_time + total\_dist + time\_penalty + 1e-6)  # 蚁群算法主函数  def aco\_greedy\_optimize():  best\_routes = []  eta = heuristic\_info() # 启发式信息    for warehouse in range(n\_warehouses):  # 初始化信息素  tau = np.ones\_like(road\_time) \* 0.1  # 贪婪算法生成初始最优解  best\_route = greedy\_route(warehouse)  best\_fitness = fitness(best\_route, warehouse)    for iter in range(iterations):  ant\_routes = []  ant\_fitnesses = []    for ant in range(n\_ants):  # 构建路径  route = [warehouse]  visited = set([warehouse])    while len(visited) < n\_victims + 1: # 所有点+仓库  current = route[-1]  # 可选节点  candidates = [v for v in range(len(all\_points))  if v not in visited and v != current]  if not candidates:  break    # 计算转移概率  probs = []  for v in candidates:  probs.append(tau[current][v]\*\* alpha \* eta[current][v] \*\* beta)  probs = np.array(probs) / np.sum(probs)    # 选择下一个节点  next\_v = np.random.choice(candidates, p=probs)  route.append(next\_v)  visited.add(next\_v)    ant\_routes.append(route)  ant\_fitnesses.append(fitness(route, warehouse))    # 更新最优解  current\_best\_idx = np.argmax(ant\_fitnesses)  if ant\_fitnesses[current\_best\_idx] > best\_fitness:  best\_route = ant\_routes[current\_best\_idx]  best\_fitness = ant\_fitnesses[current\_best\_idx]    # 信息素蒸发  tau = (1 - rho) \* tau    # 信息素更新  for i in range(n\_ants):  route = ant\_routes[i]  fit = ant\_fitnesses[i]  for j in range(len(route)-1):  u = route[j]  v = route[j+1]  tau[u][v] += Q \* fit # 根据适应度增加信息素    best\_routes.append((best\_route, best\_fitness, warehouse))    # 选择全局最优解  best\_idx = np.argmax([f for (r, f, w) in best\_routes])  return best\_routes[best\_idx]  # 运行并可视化  if \_\_name\_\_ == "\_\_main\_\_":  best\_route, best\_fit, warehouse = aco\_greedy\_optimize()  print(f"最优配送路线（从仓库{warehouse}出发）：", best\_route[:5], "...（省略部分点）")  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='应急仓库')  plt.scatter(victims[:,0], victims[:,1], c='blue', s=50, label='受灾点')  # 标记出发仓库  plt.scatter(all\_points[warehouse,0], all\_points[warehouse,1], c='green', s=150, label='出发仓库')  # 绘制路线  route\_points = all\_points[best\_route]  plt.plot(route\_points[:,0], route\_points[:,1], 'g-', linewidth=2)  plt.title('灾后物资最优配送路线')  plt.legend()  plt.show() |