### 8. 模拟退火（SA）+ 线性规划（LP）组合模型案例题目

**题目：突发公共卫生事件医疗物资分配优化问题**

* **问题背景**：某省突发疫情，15 个地市出现病例，需紧急分配口罩、防护服、呼吸机等医疗物资。现有省级储备库 3 个，物资总量有限（如口罩 500 万只、防护服 10 万套），部分地市因交通管制导致运输难度增加。
* **问题描述**：需制定物资分配方案，目标包括：① 最大化物资覆盖率（高风险地区优先）；② 最小化运输时间（高风险地区≤24 小时）；③ 最小化运输成本（与距离和物资重量相关）。分配需满足线性约束（如某物资总分配量≤库存量），同时处理交通管制等非线性因素。
* **数据情况**：提供各地市的风险等级（高 / 中 / 低）、确诊病例数、床位数、各物资需求量，储备库的物资库存量，地市与储备库的距离及运输成本系数（元 / 吨・km），以及受交通管制影响的路段运输时间修正系数（如增加 50%）。

### 8. 模拟退火（SA）+ 线性规划（LP）求解突发公共卫生事件医疗物资分配优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import random  from math import exp  from scipy.optimize import linprog  import seaborn as sns  from matplotlib.colors import LinearSegmentedColormap  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  # 数据初始化  def initialize\_medical\_data():  """初始化医疗物资分配问题的基础数据"""  # 15个受灾地区，3个储备仓库，3类物资  n\_regions = 15  n\_warehouses = 3  n\_items = 3 # 0-防护装备，1-急救药品，2-医疗器械    # 各地区物资需求量（单位：防护装备-箱，药品-箱，器械-套）  demand = np.array([  [120, 80, 30], [90, 60, 20], [100, 70, 25],  [150, 90, 35], [80, 50, 15], [110, 65, 22],  [95, 68, 24], [130, 85, 32], [70, 45, 12],  [160, 95, 38], [92, 58, 21], [105, 72, 26],  [125, 82, 31], [85, 52, 16], [115, 75, 28]  ])    # 各仓库物资库存量  stock = np.array([  [600, 400, 150], # 仓库1  [500, 350, 130], # 仓库2  [450, 300, 120] # 仓库3  ])    # 地区风险等级（0-低，1-中，2-高）  risk\_level = np.array([2, 1, 1, 2, 0, 1, 1, 2, 0, 2, 1, 1, 2, 0, 1])    # 地区坐标（用于可视化）  region\_coords = np.random.rand(n\_regions, 2) \* 100  warehouse\_coords = np.array([[10, 10], [50, 80], [90, 30]]) # 仓库固定位置    # 运输成本系数（万元/单位物资·公里）  cost\_coef = np.array([0.08, 0.15, 0.25]) # 防护装备、急救药品、医疗器械    # 仓库到地区的距离（公里）  distance = np.zeros((n\_warehouses, n\_regions))  for i in range(n\_warehouses):  for j in range(n\_regions):  distance[i, j] = np.sqrt(np.sum((warehouse\_coords[i] - region\_coords[j])\*\*2))    # 通行时间修正系数（1为正常，>1表示通行困难）  time\_factor = np.ones((n\_warehouses, n\_regions))  # 随机设置20%的路段为困难路段  for i in range(n\_warehouses):  for j in range(n\_regions):  if random.random() < 0.2:  time\_factor[i, j] = 1.5 + random.random() \* 0.5 # 1.5-2.0倍时间    # 地区名称  region\_names = [f"地区{i+1}" for i in range(n\_regions)]  warehouse\_names = [f"仓库{i+1}" for i in range(n\_warehouses)]  item\_names = ["防护装备", "急救药品", "医疗器械"]    return {  "n\_regions": n\_regions,  "n\_warehouses": n\_warehouses,  "n\_items": n\_items,  "demand": demand,  "stock": stock,  "risk\_level": risk\_level,  "region\_coords": region\_coords,  "warehouse\_coords": warehouse\_coords,  "cost\_coef": cost\_coef,  "distance": distance,  "time\_factor": time\_factor,  "region\_names": region\_names,  "warehouse\_names": warehouse\_names,  "item\_names": item\_names  }  # 线性规划模型（固定分配方案下的物资优化）  def lp物资分配(data, warehouse, regions):  """  对指定仓库和负责地区进行物资分配优化  目标：最小化运输成本，满足需求和库存约束  """  n\_regions = len(regions)  n\_items = data["n\_items"]  n\_vars = n\_regions \* n\_items # 变量数量：地区×物资    # 目标函数系数（总运输成本）  c = []  for r\_idx, r in enumerate(regions):  for item in range(n\_items):  # 成本 = 距离×单位成本×时间系数  cost = data["distance"][warehouse, r] \* data["cost\_coef"][item] \* data["time\_factor"][warehouse, r]  c.append(cost)    # 约束条件：A\_ub @ x <= b\_ub  A\_ub = []  b\_ub = []    # 1. 仓库库存约束  for item in range(n\_items):  row = [0] \* n\_vars  for r\_idx in range(n\_regions):  row[r\_idx \* n\_items + item] = 1 # 对应物资的变量系数为1  A\_ub.append(row)  b\_ub.append(data["stock"][warehouse, item]) # 库存上限    # 2. 地区需求约束（按风险等级设置最低满足率）  for r\_idx, r in enumerate(regions):  risk = data["risk\_level"][r]  for item in range(n\_items):  row = [0] \* n\_vars  row[r\_idx \* n\_items + item] = -1 # 负号用于转换为 <= 约束  A\_ub.append(row)    # 高风险90%，中风险80%，低风险70%  min\_rate = 0.9 if risk == 2 else 0.8 if risk == 1 else 0.7  min\_demand = data["demand"][r, item] \* min\_rate  b\_ub.append(-min\_demand) # 负号对应不等式转换    # 求解线性规划  bounds = [(0, None) for \_ in range(n\_vars)] # 变量非负约束  result = linprog(c, A\_ub=A\_ub, b\_ub=b\_ub, bounds=bounds, method='highs')    if result.success:  # 重塑结果为（地区×物资）矩阵  allocation = result.x.reshape(n\_regions, n\_items)  return allocation, result.fun # 分配方案和总成本  else:  # 若无解，返回高成本方案（实际应用中需调整约束）  return np.zeros((n\_regions, n\_items)), 1e10  # 模拟退火算法（优化仓库-地区分配关系）  def sa\_optimization(data):  """  模拟退火算法优化仓库-地区分配方案  目标：找到全局最优的物资配送网络  """  n\_regions = data["n\_regions"]  n\_warehouses = data["n\_warehouses"]    # 初始解：随机分配地区给仓库  current\_assign = np.random.randint(0, n\_warehouses, size=n\_regions)  best\_assign = current\_assign.copy()    # 计算方案总成本  def total\_cost(assignment):  """计算当前分配方案的总成本"""  cost = 0  for w in range(n\_warehouses):  regions = np.where(assignment == w)[0]  if len(regions) == 0:  continue  \_, w\_cost = lp物资分配(data, w, regions)  cost += w\_cost  return cost    # 初始成本  current\_cost = total\_cost(current\_assign)  best\_cost = current\_cost  cost\_history = [best\_cost]    # 模拟退火参数  temp = 1000.0 # 初始温度  cooling\_rate = 0.95 # 冷却速率  iterations = 200 # 迭代次数  min\_temp = 1e-3 # 最低温度    print("模拟退火优化过程:")  print(f"初始成本: {current\_cost:.2f} 万元")    for i in range(iterations):  if temp < min\_temp:  break    # 生成邻域解（随机交换两个地区的分配）  a, b = random.sample(range(n\_regions), 2)  new\_assign = current\_assign.copy()  new\_assign[a], new\_assign[b] = new\_assign[b], new\_assign[a]  new\_cost = total\_cost(new\_assign)    # 接受准则  if new\_cost < current\_cost:  # 接受更优解  current\_assign = new\_assign  current\_cost = new\_cost  if new\_cost < best\_cost:  best\_cost = new\_cost  best\_assign = new\_assign  else:  # Metropolis准则接受较差解  acceptance\_prob = exp((current\_cost - new\_cost) / temp)  if random.random() < acceptance\_prob:  current\_assign = new\_assign  current\_cost = new\_cost    # 降温  temp \*= cooling\_rate  cost\_history.append(best\_cost)    # 定期输出进度  if (i + 1) % 20 == 0:  print(f"迭代 {i+1}, 温度: {temp:.4f}, 最优成本: {best\_cost:.2f} 万元")    print(f"优化完成, 最终最优成本: {best\_cost:.2f} 万元")  return best\_assign, best\_cost, cost\_history  # 生成详细分配结果  def generate\_allocation\_details(data, best\_assign):  """根据最优分配方案生成详细的物资分配表"""  n\_warehouses = data["n\_warehouses"]  allocation = np.zeros((n\_warehouses, data["n\_regions"], data["n\_items"]))  coverage = np.zeros((data["n\_regions"], data["n\_items"])) # 需求满足率    for w in range(n\_warehouses):  regions = np.where(best\_assign == w)[0]  if len(regions) == 0:  continue  alloc, \_ = lp物资分配(data, w, regions)  for r\_idx, r in enumerate(regions):  allocation[w, r] = alloc[r\_idx]  coverage[r] = alloc[r\_idx] / data["demand"][r] \* 100 # 转换为百分比    return allocation, coverage  # 结果可视化  def visualize\_results(data, best\_assign, allocation, coverage, cost\_history):  """可视化物资分配结果"""  plt.figure(figsize=(18, 14))    # 1. 成本变化曲线  plt.subplot(2, 2, 1)  plt.plot(cost\_history)  plt.title('模拟退火优化成本变化', fontsize=12)  plt.xlabel('迭代次数')  plt.ylabel('总成本（万元）')  plt.grid(True, linestyle='--', alpha=0.7)    # 2. 地区-仓库分配地图  plt.subplot(2, 2, 2)  cmap = LinearSegmentedColormap.from\_list('warehouse\_cmap', ['#FF9999', '#66B2FF', '#99FF99'])    # 绘制地区  scatter = plt.scatter(  data["region\_coords"][:, 0], data["region\_coords"][:, 1],  c=best\_assign, cmap=cmap, s=100, edgecolors='k', alpha=0.7  )  # 绘制仓库  plt.scatter(  data["warehouse\_coords"][:, 0], data["warehouse\_coords"][:, 1],  c=np.arange(data["n\_warehouses"]), cmap=cmap, s=300, marker='\*', edgecolors='k'  )  # 绘制分配关系  for r in range(data["n\_regions"]):  w = best\_assign[r]  plt.plot(  [data["region\_coords"][r, 0], data["warehouse\_coords"][w, 0]],  [data["region\_coords"][r, 1], data["warehouse\_coords"][w, 1]],  'gray', linestyle='--', alpha=0.3  )    plt.title('地区-仓库分配关系', fontsize=12)  plt.xlabel('X坐标')  plt.ylabel('Y坐标')  legend\_elements = [Patch(facecolor=cmap(i), edgecolor='k', label=f'仓库{i+1}')  for i in range(data["n\_warehouses"])]  plt.legend(handles=legend\_elements, loc='upper right')    # 3. 需求满足率热力图  plt.subplot(2, 1, 2)  region\_names = [f"地区{i+1}" for i in range(data["n\_regions"])]  sns.heatmap(coverage, annot=True, fmt='.1f', cmap='YlGn',  xticklabels=data["item\_names"], yticklabels=region\_names)  plt.title('各地区物资需求满足率（%）', fontsize=12)  plt.tight\_layout()    plt.show()    # 4. 各仓库物资分配量（条形图）  plt.figure(figsize=(15, 6))  bar\_width = 0.25  index = np.arange(data["n\_items"])    for w in range(data["n\_warehouses"]):  total\_alloc = np.sum(allocation[w], axis=0)  plt.bar(index + w \* bar\_width, total\_alloc, bar\_width,  label=data["warehouse\_names"][w])    plt.title('各仓库物资分配总量', fontsize=12)  plt.xlabel('物资类型')  plt.ylabel('分配数量')  plt.xticks(index + bar\_width, data["item\_names"])  plt.legend()  plt.grid(axis='y', linestyle='--', alpha=0.7)  plt.tight\_layout()  plt.show()  # 主函数  def main():  # 初始化数据  data = initialize\_medical\_data()    # 执行模拟退火优化  best\_assign, best\_cost, cost\_history = sa\_optimization(data)    # 生成详细分配结果  allocation, coverage = generate\_allocation\_details(data, best\_assign)    # 输出关键结果  print("\n优化结果摘要:")  for w in range(data["n\_warehouses"]):  regions = np.where(best\_assign == w)[0]  regions\_str = ", ".join([f"地区{r+1}" for r in regions])  print(f"仓库{w+1}负责的地区: {regions\_str}")    print(f"\n各风险等级平均满足率:")  for risk in [0, 1, 2]:  risk\_regions = np.where(data["risk\_level"] == risk)[0]  if len(risk\_regions) > 0:  avg\_cov = np.mean(coverage[risk\_regions])  print(f"{'低' if risk==0 else '中' if risk==1 else '高'}风险地区: {avg\_cov:.2f}%")    # 可视化结果  visualize\_results(data, best\_assign, allocation, coverage, cost\_history)  if \_\_name\_\_ == "\_\_main\_\_":  main() |