### 14. 遗传算法（GA）+ 约束规划（CP）组合模型案例题目

**题目：航空公司机组人员排班优化问题**

* **问题背景**：某航空公司有 100 条国内航线，需为每日 300 个航班分配机组人员（飞行员 + 乘务员）。机组人员需满足严格的约束：如单日飞行时间≤8 小时、连续休息≥10 小时、每月飞行时间≤100 小时，且特定机型需匹配持证飞行员。当前排班依赖人工，常出现合规性冲突（月冲突率 12%）。
* **问题描述**：需优化机组人员与航班的匹配方案，目标包括：① 满足所有法规约束（冲突率 = 0）；② 最小化机组人员出差住宿成本；③ 平衡各机组的月飞行时长（差异≤5 小时）。约束条件复杂且相互关联（如休息时间受前一日航班结束时间影响）。
* **数据情况**：提供航班时刻表（起飞 / 降落时间、机型、航线）、机组人员信息（资质、居住地、月累计飞行时间）、不同城市的住宿成本、法规约束细则（如飞行时间限制条款）。

### 14. 遗传算法（GA）+ 约束规划（CP）求解航空公司机组人员排班优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  import random  import copy  from datetime import datetime, timedelta  import itertools  from collections import defaultdict  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  # 1. 数据生成与初始化  def generate\_airline\_data(n\_flights=300, n\_crews=150, n\_days=30):  """  生成航空公司机组人员排班问题的数据  n\_flights: 每日航班数量  n\_crews: 机组人员总数  n\_days: 优化周期（天数）  """  # 1.1 机场与城市信息  cities = [  "北京", "上海", "广州", "深圳", "成都",  "杭州", "重庆", "西安", "武汉", "南京",  "沈阳", "大连", "青岛", "厦门", "昆明"  ]  n\_cities = len(cities)    # 城市间距离（km）和住宿成本（元/晚）  city\_data = pd.DataFrame({  "城市": cities,  "住宿成本": np.random.randint(300, 801, size=n\_cities), # 300-800元/晚  "机场代码": [f"CK{i+1:02d}" for i in range(n\_cities)]  })    # 1.2 机型信息  aircraft\_types = pd.DataFrame({  "机型": ["A320", "B737", "A330", "B777"],  "所需飞行员": [2, 2, 3, 4],  "所需乘务员": [4, 4, 8, 10],  "资质要求": ["C1", "C2", "C3", "C4"]  })    # 1.3 航班数据  flights = []  flight\_ids = [f"FL{i+1:03d}" for i in range(n\_flights)]    # 生成每日航班计划（简化为一天的航班，可扩展到n\_days天）  for flight\_id in flight\_ids:  # 随机选择起飞机场和目的机场  dep\_city\_idx, arr\_city\_idx = random.sample(range(n\_cities), 2)  dep\_city = cities[dep\_city\_idx]  arr\_city = cities[arr\_city\_idx]    # 飞行时间（小时）  distance = np.random.randint(500, 2501) # 500-2500km  flight\_time = distance / 800 # 平均时速800km/h    # 起飞时间（一天中的随机时间）  dep\_hour = random.randint(6, 20) # 6:00-20:00之间起飞  dep\_minute = random.choice([0, 15, 30, 45])  dep\_time = datetime(2023, 1, 1, dep\_hour, dep\_minute)    # 到达时间  arr\_time = dep\_time + timedelta(hours=flight\_time)    # 选择机型  aircraft = random.choice(aircraft\_types["机型"].values)  ac\_info = aircraft\_types[aircraft\_types["机型"] == aircraft].iloc[0]    flights.append({  "航班ID": flight\_id,  "起飞城市": dep\_city,  "到达城市": arr\_city,  "起飞时间": dep\_time,  "到达时间": arr\_time,  "飞行时长": round(flight\_time, 2),  "机型": aircraft,  "所需飞行员": ac\_info["所需飞行员"],  "所需乘务员": ac\_info["所需乘务员"],  "资质要求": ac\_info["资质要求"],  "住宿成本": city\_data[city\_data["城市"] == arr\_city]["住宿成本"].values[0]  })    flights\_df = pd.DataFrame(flights)    # 1.4 机组人员数据  crew\_types = ["飞行员", "乘务员"]  crews = []    for crew\_id in range(n\_crews):  # 随机选择人员类型  crew\_type = random.choice(crew\_types)    # 飞行员有资质要求，乘务员无  if crew\_type == "飞行员":  # 随机拥有1-2种资质  qualifications = random.sample(aircraft\_types["资质要求"].unique().tolist(),  random.randint(1, 2))  else:  qualifications = []    # 居住地  base\_city = random.choice(cities)    # 月累计飞行时间（优化开始前）  monthly\_hours = random.uniform(40, 80) # 40-80小时    # 日累计飞行时间（初始为0）  daily\_hours = 0.0    crews.append({  "人员ID": f"CR{crew\_id+1:03d}",  "类型": crew\_type,  "资质": qualifications,  "居住地": base\_city,  "月累计飞行时间": round(monthly\_hours, 2),  "日累计飞行时间": daily\_hours,  "最后休息开始时间": datetime(2023, 1, 1, 0, 0), # 初始休息时间  "连续休息时长": 12.0 # 初始有12小时休息  })    crews\_df = pd.DataFrame(crews)    # 1.5 法规约束参数  regulations = {  "单日最大飞行时间": 8.0, # 小时  "连续最小休息时间": 10.0, # 小时  "每月最大飞行时间": 100.0, # 小时  "最大连续工作天数": 6, # 天  "最小连续休息天数": 1, # 天  "同机组最大连续合作天数": 5 # 天  }    return {  "航班数据": flights\_df,  "机组人员数据": crews\_df,  "城市数据": city\_data,  "机型数据": aircraft\_types,  "法规约束": regulations,  "优化周期": n\_days,  "城市列表": cities  }  # 2. 约束规划（CP）检查与修复  class CrewSchedulingCP:  def \_\_init\_\_(self, data):  """  初始化约束规划模块  data: 包含航班、机组人员和约束规则的数据  """  self.data = data  self.regulations = data["法规约束"]  self.flights = data["航班数据"]  self.crews = data["机组人员数据"].copy()    # 记录机组人员的排班历史  self.crew\_history = defaultdict(list) # crew\_id -> 排班记录  self.conflict\_stats = defaultdict(int) # 冲突类型统计    def check\_constraints(self, assignment, day):  """  检查单个航班的机组分配是否满足所有约束  assignment: {航班ID: {飞行员列表, 乘务员列表}}  day: 当前日期（用于计算月度累计）  返回值: 是否满足所有约束，冲突信息  """  conflicts = []  all\_valid = True    # 1. 检查每个航班的人员配置是否满足机型要求  for flight\_id, crew\_assign in assignment.items():  flight\_info = self.flights[self.flights["航班ID"] == flight\_id].iloc[0]  required\_pilots = flight\_info["所需飞行员"]  required\_cabin = flight\_info["所需乘务员"]  required\_qual = flight\_info["资质要求"]    # 人数检查  if len(crew\_assign["飞行员"]) != required\_pilots:  conflicts.append({  "类型": "人员数量不足",  "航班ID": flight\_id,  "详情": f"需要{required\_pilots}名飞行员，实际分配{len(crew\_assign['飞行员'])}名"  })  all\_valid = False    if len(crew\_assign["乘务员"]) != required\_cabin:  conflicts.append({  "类型": "人员数量不足",  "航班ID": flight\_id,  "详情": f"需要{required\_cabin}名乘务员，实际分配{len(crew\_assign['乘务员'])}名"  })  all\_valid = False    # 资质检查（飞行员）  for pilot\_id in crew\_assign["飞行员"]:  pilot\_info = self.crews[self.crews["人员ID"] == pilot\_id].iloc[0]  if required\_qual not in pilot\_info["资质"]:  conflicts.append({  "类型": "资质不匹配",  "航班ID": flight\_id,  "人员ID": pilot\_id,  "详情": f"飞行员无{required\_qual}资质，无法执行该机型航班"  })  all\_valid = False    # 2. 检查每个机组人员的约束  all\_assigned\_crews = []  for crew\_assign in assignment.values():  all\_assigned\_crews.extend(crew\_assign["飞行员"])  all\_assigned\_crews.extend(crew\_assign["乘务员"])    # 去重  all\_assigned\_crews = list(set(all\_assigned\_crews))    for crew\_id in all\_assigned\_crews:  crew\_info = self.crews[self.crews["人员ID"] == crew\_id].iloc[0]  crew\_history = self.crew\_history[crew\_id]    # 找到该人员当天分配的所有航班  assigned\_flights = []  for flight\_id, crew\_assign in assignment.items():  if crew\_id in crew\_assign["飞行员"] or crew\_id in crew\_assign["乘务员"]:  assigned\_flights.append(flight\_id)    # 计算当天总飞行时间  total\_flight\_time = 0.0  flight\_times = []  for flight\_id in assigned\_flights:  flight\_info = self.flights[self.flights["航班ID"] == flight\_id].iloc[0]  total\_flight\_time += flight\_info["飞行时长"]  flight\_times.append((flight\_info["起飞时间"], flight\_info["到达时间"]))    # 2.1 单日飞行时间约束  if total\_flight\_time > self.regulations["单日最大飞行时间"]:  conflicts.append({  "类型": "单日飞行超时",  "人员ID": crew\_id,  "详情": f"当日飞行{total\_flight\_time:.2f}小时，超过最大限制{self.regulations['单日最大飞行时间']}小时"  })  all\_valid = False  self.conflict\_stats["单日飞行超时"] += 1    # 2.2 休息时间约束（与前一天最后航班比较）  if crew\_history:  last\_day = max([h["日期"] for h in crew\_history])  last\_flights = [h for h in crew\_history if h["日期"] == last\_day]    if last\_flights:  # 前一天最后到达时间  last\_arrival = max([f["到达时间"] for h in last\_flights for f in h["航班"]])  # 今天最早起飞时间  first\_departure = min([ft[0] for ft in flight\_times])    # 计算休息时长  rest\_duration = (first\_departure - last\_arrival).total\_seconds() / 3600    if rest\_duration < self.regulations["连续最小休息时间"]:  conflicts.append({  "类型": "休息时间不足",  "人员ID": crew\_id,  "详情": f"仅休息{rest\_duration:.2f}小时，低于最小要求{self.regulations['连续最小休息时间']}小时"  })  all\_valid = False  self.conflict\_stats["休息时间不足"] += 1    # 2.3 月度飞行时间约束  current\_monthly = crew\_info["月累计飞行时间"] + total\_flight\_time  if current\_monthly > self.regulations["每月最大飞行时间"]:  conflicts.append({  "类型": "月度飞行超时",  "人员ID": crew\_id,  "详情": f"月度累计将达{current\_monthly:.2f}小时，超过最大限制{self.regulations['每月最大飞行时间']}小时"  })  all\_valid = False  self.conflict\_stats["月度飞行超时"] += 1    return all\_valid, conflicts    def repair\_solution(self, assignment, day):  """  修复不满足约束的排班方案  assignment: 初始排班方案  day: 当前日期  返回值: 修复后的排班方案  """  # 检查当前方案  valid, conflicts = self.check\_constraints(assignment, day)  if valid:  return assignment    # 统计冲突人员  conflict\_crews = set()  for c in conflicts:  if "人员ID" in c:  conflict\_crews.add(c["人员ID"])    # 收集可用的替代人员  all\_crews = self.crews["人员ID"].tolist()  available\_crews = [c for c in all\_crews if c not in conflict\_crews]    # 按冲突类型进行修复  for conflict in conflicts:  if conflict["类型"] == "资质不匹配":  # 替换为有相应资质的飞行员  flight\_id = conflict["航班ID"]  crew\_id = conflict["人员ID"]  flight\_info = self.flights[self.flights["航班ID"] == flight\_id].iloc[0]  required\_qual = flight\_info["资质要求"]    # 找到该航班的分配  for fa in assignment.values():  if flight\_id in fa:  # 找到该飞行员在列表中的位置  if crew\_id in fa["飞行员"]:  idx = fa["飞行员"].index(crew\_id)    # 寻找替代飞行员  eligible\_pilots = []  for c in available\_crews:  c\_info = self.crews[self.crews["人员ID"] == c].iloc[0]  if c\_info["类型"] == "飞行员" and required\_qual in c\_info["资质"]:  eligible\_pilots.append(c)    if eligible\_pilots:  # 选择一个替代  replacement = random.choice(eligible\_pilots)  fa["飞行员"][idx] = replacement  available\_crews.remove(replacement)  conflict\_crews.add(crew\_id)  available\_crews.append(crew\_id)    # 再次检查修复后的方案  valid, \_ = self.check\_constraints(assignment, day)  if valid:  return assignment    # 如果仍有冲突，进行更深度的修复（随机替换冲突人员）  for crew\_id in conflict\_crews:  # 找到该人员分配的所有航班  flights\_to\_reassign = []  for flight\_id, fa in assignment.items():  if crew\_id in fa["飞行员"]:  flights\_to\_reassign.append((flight\_id, "飞行员", fa["飞行员"].index(crew\_id)))  elif crew\_id in fa["乘务员"]:  flights\_to\_reassign.append((flight\_id, "乘务员", fa["乘务员"].index(crew\_id)))    # 为每个航班重新分配人员  for flight\_id, crew\_type, idx in flights\_to\_reassign:  flight\_info = self.flights[self.flights["航班ID"] == flight\_id].iloc[0]    # 寻找合适的替代人员  eligible = []  for c in available\_crews:  c\_info = self.crews[self.crews["人员ID"] == c].iloc[0]  if c\_info["类型"] == crew\_type:  # 飞行员需要检查资质  if crew\_type == "飞行员":  if flight\_info["资质要求"] in c\_info["资质"]:  eligible.append(c)  else:  eligible.append(c)    if eligible:  replacement = random.choice(eligible)  # 更新分配  assignment[flight\_id][crew\_type][idx] = replacement  available\_crews.remove(replacement)  available\_crews.append(crew\_id)    return assignment    def update\_crew\_history(self, assignment, day):  """更新机组人员的排班历史"""  for flight\_id, crew\_assign in assignment.items():  flight\_info = self.flights[self.flights["航班ID"] == flight\_id].iloc[0]  flight\_data = {  "航班ID": flight\_id,  "起飞时间": flight\_info["起飞时间"],  "到达时间": flight\_info["到达时间"],  "飞行时长": flight\_info["飞行时长"],  "到达城市": flight\_info["到达城市"]  }    # 更新飞行员历史  for pilot\_id in crew\_assign["飞行员"]:  self.crew\_history[pilot\_id].append({  "日期": day,  "航班": [flight\_data]  })  # 更新累计飞行时间  self.crews.loc[self.crews["人员ID"] ==</doubaocanvas> |