### 2. 粒子群优化（PSO）+ 禁忌搜索（TS）组合模型案例题目

**题目：半导体车间多设备协同调度问题**

* **问题背景**：某半导体工厂的芯片封装车间有 10 台不同型号的封装设备，需处理 80 批不同规格的芯片（每批数量 500-2000 片）。芯片封装需经过 “固晶 - 焊线 - 塑封 - 切筋” 4 道工序，不同设备的工序加工效率差异显著，且设备切换规格时需停机调试（耗时 30-60 分钟）。当前调度方案导致设备闲置率达 25%，订单平均延期率 18%。
* **问题描述**：需优化各批次芯片在设备间的加工顺序和时间分配，目标包括：① 最小化总生产周期（≤72 小时）；② 最大化设备利用率；③ 最小化订单延期时间（重要订单延期罚款是普通订单的 3 倍）。约束条件：每道工序必须按顺序执行，某设备同一时间只能处理一批芯片，部分精密设备每日需预留 2 小时维护时间。
* **数据情况**：提供各批次芯片的工序需求（如 A 批次需在设备 1 完成固晶、设备 5 完成焊线）、各设备的工序加工速率（片 / 分钟）、设备切换规格的调试时间矩阵、各订单的交货截止时间及优先级标签、设备每日可用时间窗口。

### 2. 粒子群优化（PSO）+ 禁忌搜索（TS）求解半导体车间调度代码

|  |
| --- |
| import numpy as np  import random  import matplotlib.pyplot as plt  # 数据初始化  np.random.seed(42)  n\_jobs = 80 # 80批芯片  n\_machines = 10 # 10台设备  n\_stages = 4 # 4道工序  # 每批芯片的工序需求（每道工序对应一台设备）  processing\_machines = np.random.randint(0, n\_machines, size=(n\_jobs, n\_stages))  # 每台设备的工序加工速率（片/分钟）  process\_rate = np.random.uniform(50, 200, size=(n\_machines, n\_stages))  # 设备切换规格的调试时间（分钟）  setup\_time = np.random.randint(30, 61, size=(n\_machines, n\_jobs, n\_jobs))  # 订单截止时间（小时）  due\_dates = np.random.uniform(24, 72, size=n\_jobs)  # 订单优先级（1-3，3为最高）  priority = np.random.randint(1, 4, size=n\_jobs)  # 每批数量（片）  batch\_sizes = np.random.randint(500, 2001, size=n\_jobs)  # 计算加工时间（分钟）  processing\_time = np.zeros((n\_jobs, n\_stages))  for j in range(n\_jobs):  for s in range(n\_stages):  machine = processing\_machines[j, s]  processing\_time[j, s] = batch\_sizes[j] / process\_rate[machine, s]  # PSO参数  n\_particles = 30  max\_iter = 100  w = 0.8 # 惯性权重  c1, c2 = 0.5, 0.5 # 学习因子  # 禁忌搜索参数  tabu\_size = 20  tabu\_list = []  # 粒子表示：每个粒子是作业的加工顺序  def init\_particles(n\_particles, n\_jobs):  particles = []  for \_ in range(n\_particles):  particles.append(random.sample(range(n\_jobs), n\_jobs))  return np.array(particles)  # 计算makespan（总生产周期）  def calculate\_makespan(sequence):  # 设备上的完工时间  machine\_end = np.zeros((n\_machines, n\_stages))  # 作业的工序完工时间  job\_end = np.zeros((n\_jobs, n\_stages))    for job in sequence:  for stage in range(n\_stages):  machine = processing\_machines[job, stage]  # 前道工序完工时间  prev\_stage\_end = job\_end[job, stage-1] if stage > 0 else 0  # 设备空闲时间  machine\_idle = machine\_end[machine, stage]  start\_time = max(prev\_stage\_end, machine\_idle)    # 如果是同一设备上的前一个作业，需加切换时间  if stage == 0: # 第一道工序看设备上一个作业  prev\_job = None  for j in sequence:  if j == job:  break  if processing\_machines[j, 0] == machine:  prev\_job = j  if prev\_job is not None:  start\_time += setup\_time[machine, prev\_job, job]    job\_end[job, stage] = start\_time + processing\_time[job, stage]  machine\_end[machine, stage] = job\_end[job, stage]    return np.max(job\_end)  # 计算延期时间  def calculate\_tardiness(sequence):  makespan = calculate\_makespan(sequence)  tardiness = 0  for job in sequence:  t = max(0, makespan/60 - due\_dates[job]) # 转换为小时  tardiness += t \* priority[job] # 优先级加权  return tardiness  # 适应度函数（最小化makespan和延期时间）  def fitness(sequence):  makespan = calculate\_makespan(sequence)  tardiness = calculate\_tardiness(sequence)  # 设备利用率（1 - 空闲时间占比）  machine\_util = 1 - (np.sum(machine\_end) - np.sum(processing\_time)) / np.sum(machine\_end)  return 1 / (makespan + tardiness \* 60 - machine\_util \* 1000) # 60转换为分钟  # 禁忌搜索局部优化  def tabu\_search(best\_seq, fitness\_func):  current\_seq = best\_seq.copy()  current\_fit = fitness\_func(current\_seq)  best\_tabu\_seq = current\_seq  best\_tabu\_fit = current\_fit    for \_ in range(50): # 邻域搜索次数  # 生成邻域解（交换两个作业）  a, b = random.sample(range(len(current\_seq)), 2)  new\_seq = current\_seq.copy()  new\_seq[a], new\_seq[b] = new\_seq[b], new\_seq[a]    # 检查禁忌表  if tuple(new\_seq) in tabu\_list:  continue    new\_fit = fitness\_func(new\_seq)  if new\_fit > best\_tabu\_fit:  best\_tabu\_seq = new\_seq  best\_tabu\_fit = new\_fit  current\_seq = new\_seq  current\_fit = new\_fit  # 更新禁忌表  tabu\_list.append(tuple(new\_seq))  if len(tabu\_list) > tabu\_size:  tabu\_list.pop(0)  else:  # 藐视准则：如果解更优则接受  if new\_fit > current\_fit:  current\_seq = new\_seq  current\_fit = new\_fit    return best\_tabu\_seq, best\_tabu\_fit  # PSO主算法  def pso\_ts\_optimize():  particles = init\_particles(n\_particles, n\_jobs)  velocities = [random.sample(range(n\_jobs), n\_jobs) for \_ in range(n\_particles)] # 速度表示交换位置    pbest = particles.copy()  pbest\_fitness = [fitness(seq) for seq in particles]  gbest\_idx = np.argmax(pbest\_fitness)  gbest = pbest[gbest\_idx].copy()  gbest\_fitness = pbest\_fitness[gbest\_idx]    # 禁忌搜索优化全局最优  gbest, gbest\_fitness = tabu\_search(gbest, fitness)    for iter in range(max\_iter):  for i in range(n\_particles):  # 更新速度（粒子群优化的离散化处理）  # 1. 向个体最优学习  pbest\_diff = [idx for idx, (a, b) in enumerate(zip(particles[i], pbest[i])) if a != b]  # 2. 向全局最优学习  gbest\_diff = [idx for idx, (a, b) in enumerate(zip(particles[i], gbest)) if a != b]    # 生成新速度（合并差异并抽样）  new\_vel = pbest\_diff[:int(c1\*len(pbest\_diff))] + gbest\_diff[:int(c2\*len(gbest\_diff))]  new\_vel = list(set(new\_vel)) # 去重  if len(new\_vel) < 2:  new\_vel += random.sample(range(n\_jobs), 2 - len(new\_vel))  random.shuffle(new\_vel)    # 更新位置（根据速度进行交换）  new\_seq = particles[i].copy()  for k in range(0, len(new\_vel)-1, 2):  if k+1 < len(new\_vel):  a, b = new\_vel[k], new\_vel[k+1]  new\_seq[a], new\_seq[b] = new\_seq[b], new\_seq[a]  particles[i] = new\_seq    # 评估适应度  current\_fit = fitness(new\_seq)  # 更新个体最优  if current\_fit > pbest\_fitness[i]:  pbest[i] = new\_seq.copy()  pbest\_fitness[i] = current\_fit  # 更新全局最优  if current\_fit > gbest\_fitness:  gbest = new\_seq.copy()  gbest\_fitness = current\_fit  # 禁忌搜索优化新的全局最优  gbest, gbest\_fitness = tabu\_search(gbest, fitness)    if iter % 10 == 0:  print(f"迭代 {iter}, 最优适应度: {gbest\_fitness:.4f}")    return gbest, gbest\_fitness  # 运行算法  best\_sequence, best\_fit = pso\_ts\_optimize()  print("最优加工顺序（作业索引）：", best\_sequence[:10], "...（共80个）")  print("总生产周期（分钟）：", calculate\_makespan(best\_sequence))  print("总加权延期时间（小时）：", calculate\_tardiness(best\_sequence))  # 绘制甘特图（简化版）  plt.figure(figsize=(12, 8))  for i, job in enumerate(best\_sequence[:10]): # 只显示前10个作业  for stage in range(n\_stages):  machine = processing\_machines[job, stage]  start = job\_end[job, stage] - processing\_time[job, stage]  plt.barh(machine \* n\_stages + stage, processing\_time[job, stage],  left=start, height=0.8, label=f"作业{job}" if stage == 0 else "")  plt.xlabel("时间（分钟）")  plt.ylabel("设备-工序")  plt.title("半导体车间调度甘特图（前10个作业）")  plt.legend(bbox\_to\_anchor=(1.05, 1), loc='upper left')  plt.tight\_layout()  plt.show() |