基于局部搜索算法的柔性生物芯片检测调度优化问题

摘要

随着电子技术的发展, 越来越多的科技设备问世, 各种新型的科技仪器在解决免疫类疑难杂症问题的进程中变得越发突出. 然而在检查人体的免疫类疾病方面, 传统人工检测存在的效率低、结果精确度差、耗时费力等缺点. 对此, 生物芯片的发明有效地解决了这些缺陷. 化学发光免疫分析仪利用一系列指定的处理过程, 对于装载了人类体液的生物芯片进行高效处理, 即时检测(Point-of-Care Testing, POCT) 型检测系统, 可灵活地部署在各种需要快速诊断的场景. POCT 型化学发光免疫分析仪是当前最热门的免疫诊断研发方向之一, 此类仪器在医院的门、急诊, 卒中中心、胸痛中心和基层医院等基层医疗机构都能发挥巨大的作用. 芯片依次经过进仓读码、激光超声、过滤、定量、R1R2 加样、第一次温育、磁珠加样、第二次温育、清洗以及检测等步骤后, 便可以精准地得出监测样本是否患有某种疾病的信息.

如何最高效率地, 在有限的时间内检测出尽可能多的生物芯片样本, 是化学发光免疫分析仪系统极为看重的问题, 在本次数学模型的构架中, 我们利用了传统的动态规划模型, 结合深度学习算法, 在新的有重复的时间线阶段的背景中, 制定了单位时间内尽可能多地处理生物芯片的过程. 在处理生物芯片的数学模型上, 我们将这一流程抽象为了作业车间调度优化问题这一典型的 NP-hard 问题, 高效的调度算法对降低生产成本具有重要意义. 根据仪器和单人份多项目组合试剂芯片的特殊结构, 本文基于固定周期算法, 结合化学发光免疫的检测过程, 实现时序逻辑并应用于仪器多线程测试任务的调度使用.

在该算法的测试时序下本文针对智能加工系统多种情况下的调度问题, 基于决策论与动态规划的思想,以合理利用机床资源、提高成料产量为目标,建立了动态最短路径模型, 并结合改进后的单亲遗传算法对模型进行求解.

针对问题一至问题三: 对于芯片数目, 温育时间均为常量的情况进行分析, 在最短时间内处理尽可能多的芯片的关键在于在各阶段设备的容量允许的情况下, 尽可能使得仪器各阶段可容纳芯片数目达到峰值. 考虑到温育时存在凹槽数目影响,芯片在装满了 40 节凹槽的情况下, 需要检测下一组第一个(如第 41 个) 芯片是否满足进入条件, 由于 8 个清洗盘需要闲置才可以进入新芯片进行清洗, 因此同样需要检测这个分组的下一组第一个(如第 9 枚) 芯片是否满足进入条件, 因此本文建立温育抵达点时间差, 清洗抵达点时间差等指标, 将调度优化的问题转化为时间列表相对项目数值比较的问题, 随后基于概率搜索思想, 充分考虑了时间差与容纳量的影响, 通过短时大量产生随机芯片检测序列的方法, 寻找相关约束条件下的局部最优解, 同时运用了局部搜索算法有效地提高了搜索效率, 提高了算法的准确性.

针对问题四: 由于问题四中, 芯片的数量, 所需的温育时间皆为变化常量, 直接用输入相关数值计算具体结果显得不够现实, 因此基于柔性的流水线车间调度思想, 以最小化最大完工时间为目标, 将问题抽象为一个 NP-hard 问题, 在无法给出全局最优解的情况下, 通过对于芯片字符串转化为二进制串, 构建出了满足相关时间与容纳量约束的芯片基因染色体结构, 同时通过自定义迭代次数, 种群数目, 交叉变异阈值等参数, 对于该事件进行多次模拟. 遗传算法有效地提高了搜索局部最优解的效率, 同时提高了结果的准确性.

针对问题五：本题根据题目所给条件，使用柔性车间调度模型，并根据限制条件对A、B、C 进行排序调度，最终得出排序 20B+40A+10C+10A+20B+10C+10A+30B.

最后，我们对提出的模型进行全面的评价：本文的模型贴合实际，能合理解决提出的问题，具有实用性强，算法效率高等特点，该模型在卫星调度、车辆调度、课表排课等都是这类优化问题.

关键词：柔性车间调度；遗传算法；局部搜索；调度优化；NP 难问题

目录

1 问题综述

1.1 问题背景

化学发光标记免疫分析又称化学发光免疫分析(CLIA) ,是用化学发光剂直接标记抗原或抗体的免疫分析方法. 在传统的医院检验科室中, 存在样本周转时间慢的问题, 对于近年来越来越多的免疫类疾病的出现, 这些传统的检测方法已经很难满足医学单位对于救治时间窗口的要求, 另外医院在门诊、急症科室对于检测在时效性上也提出了更高的要求. 与传统的大型检验设备不同, 由于占用空间小, 而且能方便地维护, 越来越多的 POCT 型仪器被应用在基层医院, 五大中心和门诊、急诊科中.

目前, 已经有很多学者对 POCT 型仪器面临的提供样本检测效率的问题进行了研究, 也提出了很多算法来解决此类问题, 传统的优化调度算法, 动态规划算法、固定周期时间片算法、最短路径算法等, 近几年由于高性能计算设备的快速发展, 还出现了树搜索算法、神经网络算法等. 虽然这些算法可以快速得到调度方案, 但却是通用性框架, 只能保证大多数情况下的合理性, 个别情况可能出现时间空间冲突问题, 通路堵塞问题, 这也促使我们运用更好的方法建立更加合理的模型.

1.2 问题提出

按仪器结构对单个周期仪器各模块完成的任务进行详细分解, 仪器中进料模块、加样模块、搅拌模块、检测退料模块、磁分离模块与温育槽模块之间的工作方式、时间占用长度进行分析, 计算仪器按该方式运行时间片段所需的长度. 综合上面几个模块与温育槽交互的方式, 设计分析仪固定周期算法的单个周期任务:

(1)完成进料并对试剂芯片封膜穿刺.

(2) 完成至多 6 次的加样操作.

(3) 完成从温育槽拉料到搅拌槽里的操作.

(4) 完成从搅拌模块拉料推回温育槽的操作.

(5) 完成从温育槽拉料到磁分离模块的操作.

(6) 完成从磁分离模块拉料推回温育槽的操作.

(7) 完成拉料、检测、吸废液并退料的操作.

从时间, 效率和芯片自身特性等问题进行考虑, 思考如下问题:

(1) 问题 1:单位时间(1 小时) 内最多可以完成多少 A 芯片的检测(其第一次温育时间 10 分钟, 第二次温育时间 5 分钟)? 如果换成一组 B 类型芯片(第一次温育时间 40 分钟, 第二次温育时间 20 分钟)?

(2) 问题 2: 若有 A, B 两种类型的芯片各 80 片, 则它们全部检测完至少需要多少时间?

(3) 问题 3: 若还有 C 型芯片, 其第一次温育时间 25 分钟, 第二次温育时间 10 分钟, 而 A 型芯片 60 片、B 型芯片 50 片、C 型芯片 50 片, 则全部检测完至少需要多少时间?

(4) 问题 4: 将上述模型和算法, 分别推广到芯片数量为$m\ i$, 其第一次温育时间为 a i 分钟、第二次温育时间为 b i 分钟(i=1,2,3) 的一般情形. 进一步推广到 N 的情况.

(5) 问题 5: 若开始有 A 型芯片 60 片、B 型芯片 70 片, 运行 30 分钟后, 新到 C 型芯片 20 片, 全部检测完至少需要多少时间?

1.3 资料条件

如图 1.1, 为免疫分析仪器的系统组成, 其中对于芯片的测试功能为本文的主要研究对象. 在本文的系统过程中, 需要进行二次温育. 为了实现整个的化学发光免疫检测的流程, 需要多个模块之间的共同运作, 通过获取用户操作, 软件系统分析操作, 发送操作指令到硬件系统下位机中, 硬件系统控制各个模块工作完成整个检测流程.

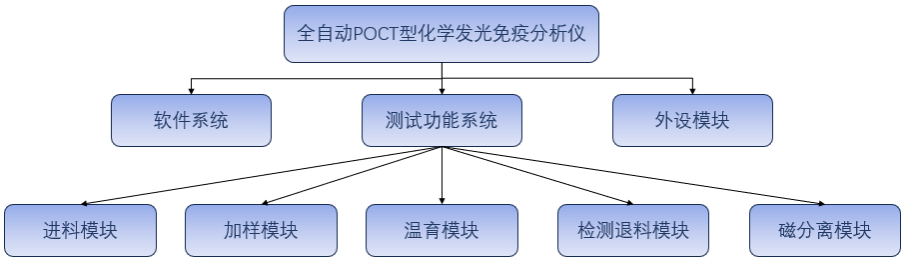


图 1.1 分析仪系统组成图

因此需要根据仪器实际情况, 设计一套可以合理调度仪器资源, 对多项目组合的试剂芯片进行测试的检测时序, 这是本软件系统设计实现上的难点之一. 不仅需要同一个试剂芯片当中, 各个项目的加样间隔、检测间隔都相同, 同时也要求在不同通道试剂芯片整体的检测流程能够相同, 这样才能够保持整个仪器系统检测的精确性. 在设计整个仪器时序的时候, 需要将这个问题考虑到其中.

通过多种模型对比分析, 最终认为该问题基本符合柔性的车间调度分配问题, 这一模型可描述为: 多个工件在多台机器上加工, 工件安排加工时严格按照工序的先后顺序, 至少有一道工序有多个可加工机器, 在某些优化目标下安排生产.

根据具体情况, 将该柔性的车间调度分配问题的约束改为:

(1) 同一工件的同一道工序在同一时刻被加工的机器数是一;

(2) 任意工序开始加工不能中断;

(3) 各个工件之间不存在的优先级的差别;

(4) 同一工件的工序之间存在先后约束, 不同工件的工序之间不存在先后约束;

(5) 所有工件在零时刻都可以被加工.

在对比多种算法的特点, 学习各种诊断仪器的检测时序之后, 决定采用遗传算法来进行问题计算与模型检验, 该算法特点

(1) 直接对结构对象进行操作, 不存在求导和函数连续性的限定;

(2) 具有内在的隐含并行性和更好的全局寻优能力;

(3) 采用概率化的寻优方法, 能自动获取和指导优化的搜索空间, 自适应地调整搜索方向, 不需要确定的规则.

对题目所给的相关附录中, 由于不同芯片的温育阶段所需时间不同, 单次可处理的温育芯片数收到凹槽数影响, 单次可清洗的芯片数收到清洗盒闲置数影响, 因此, 温育时间, 凹槽数与清洗盒数是本次建模中需要考虑的约束关键, 如何在满足相关的条件下, 使得芯片的检测效率达到最高, 是需要深入思考的相关问题.

2 模型假设与符号说明

2.1 模型基本假设

(1) 假定转盘或检测盘在每次的转动对位过程中, 无论如何转动、移动多少距离, 所需时长是一样的;
(2) 不考虑无容量标识中的设备阶段中的芯片容量;
(3) 假定每个转盘都具备同时处理进入芯片转位与转出芯片转位的功能;
(4) 假定所有芯片的清洗时间与磁珠工位均取最短时间;
(5) 假定每个阶段间的衔接处均可以堆积芯片;
(6) 工件的加工过程不允许中断.

2.2 符号说明

本文定义了如下个使用次数较多的符号, 其余符号在使用时注明.

表 2.1 符号说明

符号	含义	单位
$state_{ji}$	第 j 枚进入的芯片在第 i 阶段的状态	无
$time_{ji}$	第 j 枚进入的芯片从阶段 i 到 i+1 的实际时间	s
$moment_{ji}$	第 j 枚进入的芯片进入阶段 i 的时刻	s
num_{ni}	在第 n 时刻, 处于第 i 阶段的芯片总数	个
c_{ji}	第 j 枚进入的芯片从阶段 i 到 i+1 的理想时间	s
Makespan	所有芯片全部检测完毕所需时间	s
j	芯片个数	个
i	仪器处理阶段	无
n	整体处理过程进行到的时刻	s
N	芯片总数	个
a_j	第 j 个芯片第一次温育所需时间	s
b_j	第 j 个芯片第二次温育所需时间	s
x_{jk}	决策变量	

3 问题分析与模型建立

3.1 问题分析

题目以化学发光免疫分析仪器的生物芯片检测为背景，要求在满足特定工序与时间时长要求的情况下，求解芯片加工的最好效率顺序问题. 第一问要求在芯片种类单一的情况下，考虑仪器处理的最高效率，第二，三问通过增加芯片的种类，要求求出该情况下最高效的芯片处理顺序. 第四问要求将前几问的条件推广到普遍情景，建立出使用于一般情景的数学模型，最后一问则在考虑了芯片不同类且允许开始处理时间不同的情况下，对于排序选择的影响.

基于免疫仪器的工作流程中存在可并行处理工件的情况，我们考虑应用柔性的流水线车间调度优化模型对于问题进行一般性描述. 由于该模型问题属于 NP 难问题，当芯片的种类与数目足够多的情况下，得到全局最优解是不现实的. 因此，在对于问题进行求解时，我们使用了遗传算法进行了局部搜索，通过提高局部搜索效率来得到更接近全局最优解的局部最优解，将原问题的目标抽象为最小化最大完工时间问题，通过搜索最后完工时间的最小值来推断出芯片的处理顺序.

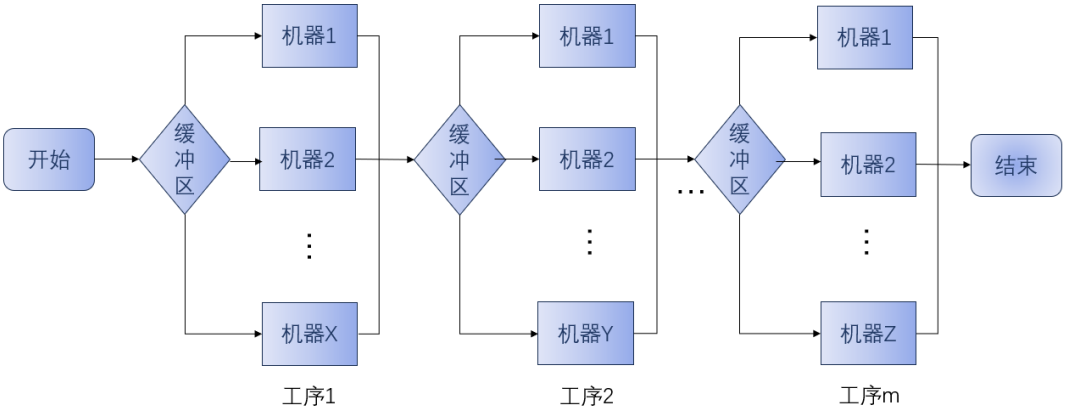


图 3.1 流水线加工程序图

从 90 年代柔性调度问题的提出至今，已有多项相关研究，如国外学者 Rajkumar 等提出用贪心随机自适应搜索过程来解决柔性作业车间有限资源约束问题. Zrbi 等采用分步法求解柔性作业车间调度问题(FlexibleJob-shopSchedulingProblem,FJSP)，先基于启发式与局部搜索方法解决机台选择问题，再对加工工序进行排序处理. Kacem 等以最小化最大完工时间、最大机器负荷和总机器负荷为目标，提出了一种基于模糊逻辑和进化算法的 Pareto 方法来解决 FJSP.

3.2 模型指标定义

x_{jk}作为决策变量，主要用来判断，是否第 j 个进入的芯片，处于队列中的 k 位置，以保证一个工件的不同顺序不可以同时进行. $state_{ji}$表示第 j 枚进入的芯片在第 i 阶段的状态；$time_{ji}$表示第 j 枚进入的芯片从阶段 i 到 i+1 的实际时间；$moment_{ji}$表示第 j 枚进入的芯片进入阶段 i 的时刻；num_{ni}表示在第 n 时刻，处于第 i 阶段的芯片总数；c_{ji}表示第 j 枚进入的芯片从阶段 i 到 i+1 的理想时间.

同时为了模型的简单化，我们自定义了一个判断函数 countif(x)，当括号内的条件被满足时，该函数输出值为 1，否则为 0. 通过这个函数，可以判断出在每一个检测时刻，每个加工处理阶段所具有的芯片数量.

3.3 调度优化模型建立

通过对题目的工序进行抽象简化后，得到如下的工序图，为了提高作业效率，将磁珠工位与清洗阶段分别取该阶段允许的最小值. 当第一次温育阶段占用了 x 个凹槽后，第二次温育需要从第 x+1 开始占用，直到 40 个凹槽占满.

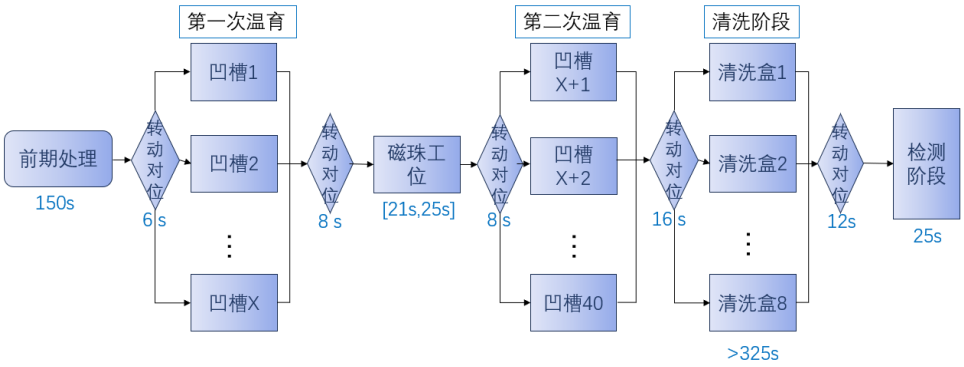


图 3.2 工件工序图

目标函数: Makespan=min(max $(moment_{j,6}+25)$)

s.t.

$$num_{n,2}+num_{n,4}<=40$$

$$num_{n,5}<=8$$

$$num_{n,i}=\sum_{j=1}^{N}\mathrm{COUNTIF}(moment_{j,i}\le n\le moment_{j,i}+1)$$

$$n<=\max(moment_{j,6}+25), j\in(1,m)\ 且 n\ge 1$$

$$moment_{1,1}=0$$

$$plan_{j,2}=a_j+8$$

$$plan_{j,4}=b_j+16$$

$$state_{j,i+1}-state_{j,i}=moment_{j,i+1}-moment_{j,i}=time_{j,i}\ge plan_{j,i}$$

$$state_{j+1,1}-state_{j,1}=moment_{j+1,1}-moment_{j,1}=30$$

$$state_{j,2}-state_{j,1}=moment_{j,1}\ge c_{j,1}=156$$

$$state_{j,3}-state_{j,2}=moment_{j,2}\ge c_{j,2}=a_j+8$$

$$state_{j,4}-state_{j,3}=moment_{j,3}\ge c_{j,3}=29$$

$$state_{j,5}-state_{j,4}=moment_{j,4}\ge c_{j,4}=b_j+16$$

$$state_{j,6}-state_{j,5}=moment_{j,5}\ge c_{j,5}=325+12$$

$$state_{j,7}-state_{j,6}=moment_{j,6}\ge c_{j,6}=25$$

$$x_{jk}=\begin{cases}1, & 芯片j是队列中第k个工件\\ 0, & 反之\end{cases}$$

$$\sum_{k=1}^{m}(x_{jk})=1, j \in \{1,2 \ldots m\}$$

其中, COUNTIF 为自定义的一个概念函数, 其具体作用为, 判断情况是否满足括号内条件, 若满足, 则输出值为1, 否则输出值为0, 利用该函数, 遍历所有芯片, 从而可以求出在 n 时刻, 处于指定阶段 i 的芯片数量.

当模型约束条件指定后, 可以列出目标函数为: F=min(max $(k_{j,6}+25)$),即求所有芯片全部结束检测的最短时间.

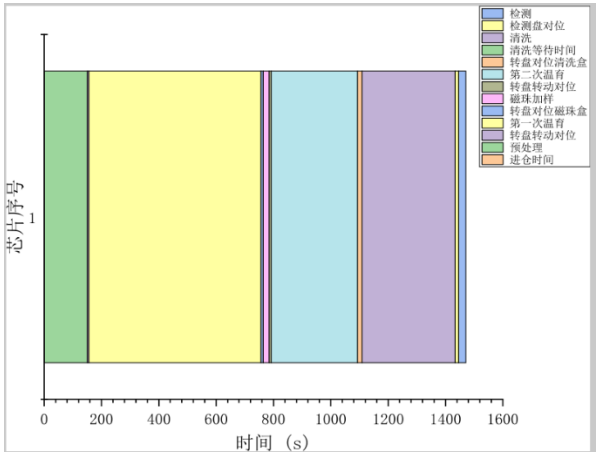


图 3.3 工序时长比例图

第五问思路, 本题主要采用柔性车间调度模型, 灵活安排 A、B、C 芯片进仓时间, 使得分析仪不间断运转, 得出满足题目的优解; A 芯片 60 片, B 芯片 70 片, 在前三十分钟内, 只有 A、B 芯片进仓, 在三十分钟后, C 芯片可以进仓; 根据不同芯片第一次温育时间的不同, 首先将第一次温育时间长的 B 芯片进仓, 再利用 B 芯片处理过程, 将 A、C 芯片进仓; 首先进仓 20 个 B 芯片, 再进仓 20 个 A 芯片, 此时卡槽放满, 由于 20 个卡槽能使得 A 芯片不间断运转, 接下来再将 20 个 A 芯片进仓, 在 60 个进仓间隔后, C 芯片可以进仓, 将 10 个 C 芯片进仓, 再进 10 个 A 芯片, 此时已经过了 80 个进仓间隔, 再将 20 个 B 芯片和 10 个 C 芯片进仓, 此时卡槽只剩下 B、C 芯片, 再将 10 个 A 芯片进仓, 此时需要等待 10 个进仓间隔进 10 个 B 芯片, 再等待十个进仓间隔, 将 10 个 B 芯片进仓, 最后等待二十个进仓间隔, 将剩下的 10 个 B 芯片进仓, 此任务完成.

3.4 遗传算法求解化学发光免疫分析仪运行的调度优化问题

3.4.1 遗传算法简介

本文研究的化学发光免疫分析仪运行的调度优化问题是典型的组合优化问题, 尤其是对于不同类型芯片的混合检测情况, 属于 NP-hard 问题. 此类问题有一系列较成熟的调度算法, 具体分类见下图, 目前更常用的是通过智能搜索算法求解. 常用到的智能优化搜索算法有遗传算法(Genetic Algorithm, GA)、粒子群算法(Particle Swarm Optimization, PS0)、蚁群算法(Ant Colony Optimization, AC0)和化学反应优化算法(Chemical Reaction Optimization Algorithm, CRO)等.

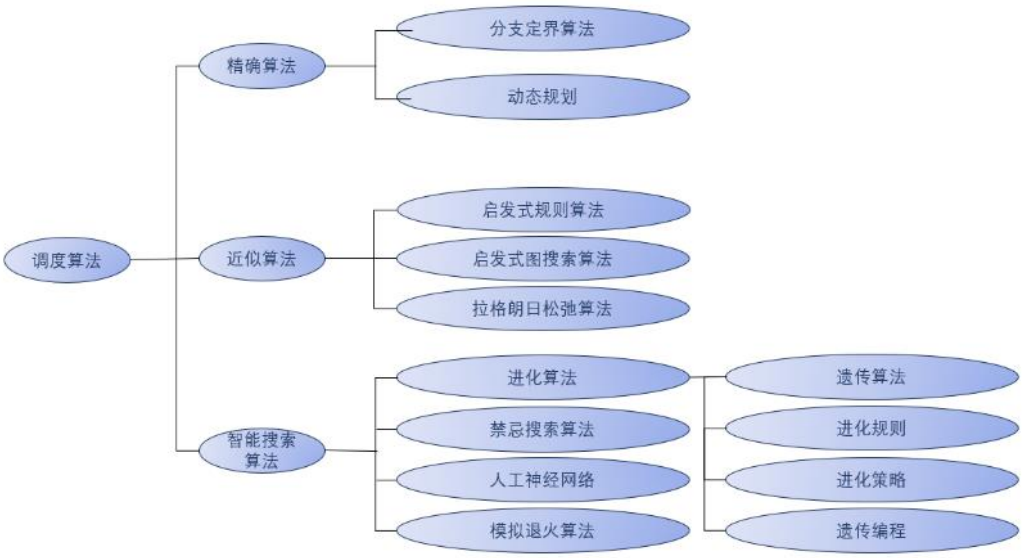


图 4.1 算法分类图

遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法. 其以直接对结构对象进行操作，不存在求导和函数连续性的限定、具有内在的隐并行性和更好的全局寻优能力、采用概率化的寻优方法以及不需要确定的规则就能自动获取和指导优化的搜索空间，自适应地调整搜索方向等优点，被广泛使用在组合优化问题当中，具体流程见图 4.2. 由于遗传算法具有比较完备的数学模型和理论，在解决很多 NP-hard 问题上具有良好的性能，针对化学发光免疫分析仪运行的调度优化问题，本文采用遗传算法进行求解. 本节对标准遗传算法进行介绍.

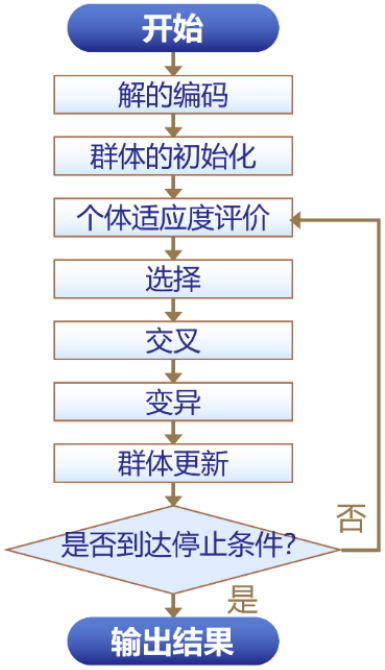


图 4.2 遗传算法流程图

编码是实现问题空间向遗传算法空间的转换，完成将优化问题可行域中的候选解设计成字符串(视为染色体) 的工作. 常用的编码方式包括二进制编码、实数编码、浮点数编码、字符编码等，要根据问题特点设计合适的编码方案.

群体初始化初始群体为遗传算法提供了进化的起点，且起点的高低(初始的种群的质量) 通常对算法的最终表现有重要的影响.

种群初始化的常用方法包括随机初始化和贪婪构造方式，前者随机产生 N (种群大小) 个互不相同的初始解；后者通过(随机) 贪婪构造算法产生初始解，如果产生的解不在初始群体中，则将它加进群体中，这个过程不断迭代，直到初始群体中个体个数达到 N .

适应度函数是遗传算法的寻优依据. 标准遗传算法在优化搜索过程中不依赖于外部信息，仅用适应度函数为寻优依据. 遗传算法对适应度函数的要求包括该函数值不能为负，适应度值进化过程不断增大.

选择是保留优秀的个体，淘汰不适应环境的个体的过程. 选择方法包括轮盘赌法、锦标赛法(tournament selection)、截断选取法(truncation selection)等. 其中轮盘赌法最为常用，根据概率的大小将圆盘分为 n 个扇形，选择时转动轮盘，假设指针最终落到扇形 i^*，则选择个体 i^*. 锦标赛法是从群体中随机挑选一定数量的个体构成一个子群体，这个子群体中最好的个体被选中进入下一代；上述过程反复迭代直到有 N(种群大小) 个个体被选中. 截断选取法中个体按照适应度值排序，淘汰 popsize*m (m 为一个给定的比例，例如, m=1/2,m=1/3 等) 个个体，用保留下来的个体产生出 popsize*m 个新的个体.

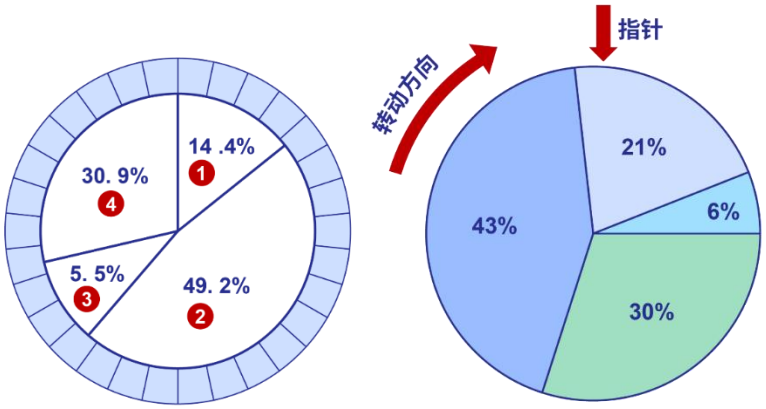


图 4. 3 轮盘赌法示意图

交叉是是父代向子代传递遗传信息的重要遗传操作. 遗传算法在当前群体中挑选出 m($m \geq 2$) 个父代解，并利用交叉算符产生出子代候选解的过程，交叉算符通常是把父代解的片段线性拼接在一起，对遗传算法的表现有重要的影响，需要根据问题做有针对性的设计. 常用的交叉方法有单点交叉、双点交叉和多点交叉. 其中单点交叉是在个体的编码序列中，随机设定一个交叉点，实行交叉时，两个个体在该点前(后) 的部分片段进行互换，并生成两个新个体.

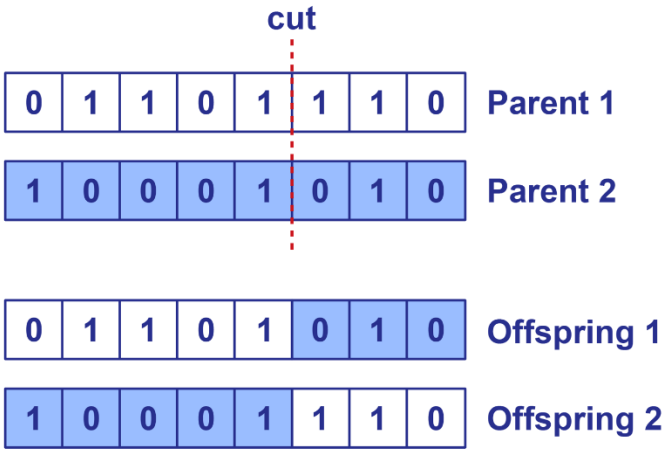


图 4.4 交叉示意图

变异的目的是引入新的基因，增加解的多样性. 常见的变异算子有两点交换、相邻交换、区间转换和单点移动等. 两点交换较为常见，首先选中需要进行变异操作的一个个体，产生给定区间内的随机整数来确定两个位置；例如，给定如下 TSP 问题的一个解，在[1,10]内产生 pos1=4, pos2=8，交换这两个位置上的基因.

图 4.5 变异示意图

更新得到的新群体包括选择后的群体、交叉产生的群体和变异产生的群体.

停止条件即为遗传算法迭代运行终止的准则，产检准则有给定最大进化步数 $C_{\max}$ 、判断最优值在连续若干代没有变化、给定最大的计算时间等.

3.4.2 遗传算法求解化学发光免疫分析仪运行的调度优化问题流程

Step 1 利用加码方式，将不同芯片种类字符串转化为二进制串，同时设置相应解码方式. 比如对于问题 2 中芯片 A 和芯片 B 各 80 个的情况，设定初始解为含 80 个 A 和 80 个 B 共 160 个字符的字符串，假设选择 A 编码为 1，选择 B 编码为 0，则随机生成初始编码为：“111000……00111”（含 80 个 1 和 80 个 0 的字符串）.

Step 2 设置染色体的基因数，种群数，迭代次数，交叉与变异的阈值.本问题的求解中，编码即为染色体，基因数和种群数主要依据为不同问题下所要求的不同类型芯片及数量.

Step 3 评估每条染色体所对应个体的适应度。适应度函数即为目标函数，在化学发光免疫分析仪运行调度问题中，所要求的是同类型芯片下的检测总时间越小越好，而考虑到适应度函数本身要求是越大越好，故可以采用负的检测总时间或检测总时间的倒数作为适应度函数.

Step 4 遵照适应度越高，选择概率越大的原则，从种群中随机选择两个个体作为父方和母方.

Step 5 通过阈值检验的情况下，抽取父母双方的染色体，进行交叉，产生子代. 交叉选择多种交叉方式随机选择的方式，增大种群内部生成的“邻域”可能情况.

Step 6 对子代的染色体按照给定的概率进行变异. 变异同样选择两点交换、相邻交换等多种交换方式随机选择使用，增大解的多样性.

Step 7 判断变异后的染色体基因种类数目是否满足题目条件, 满足直接加入新种群, 否则通过再次变异, 直至满足条件为止

Step 8 重复 2, 3, 4 步骤, 直到新种群的产生.

Step 9 对于新种群求适应度与目标函数值, 反复迭代至满足迭代次数.

Step 10 通过排序适应度列表数值, 找出符合条件的局部最优基因.

Step 11 通过调整参数, 得到较为稳定的结果.

4 问题求解与结果展示

4.1 问题 1

由于其芯片的种类单一且数目确定，因此不需要考虑芯片种类的投放顺序．为了提高检测效率，应当在尽可能避免排队等候的情况下，将每个加工的机器满载运作．因此，在第一步投放芯片时，每间隔 30s 便投放一次芯片．同时注意，由于温育的凹槽和清洗盒的最大容量分别为 40 和 8，因此对芯片分别每 8 个，每 40 个分成一组进行检查，后一组第一个进入温育与清洗时刻，至少要大于前一组中一枚芯片出这两道工序的时间．同时由于温育时间分别为 10 分钟和 5 分钟，经过测算不存在前一组芯片进入二次温育前，给后一组芯片抢先占领位置导致等待的情况．最终的计算结果表见附表 1，部分过程见图 4.1.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	芯片序号	进仓时间	出仓时间	第一次温育(40)起始时间	第一次温育(40)结束时间	磁珠(1)加样起始时间	磁珠(1)加样结束时间	第二次温育(40)起始时间	第二次温育(40)结束时间	清洗(8)起始时间	清洗(8)结束时间	检测(1)起始时间	检测结束时间	时间差		A	累计时间
2	1	0	150	156	756	764	785	793	1093	1109	1434	1446	1471		前处理	150	150
3	2	30	180	186	786	794	815	823	1123	1139	1464	1476	1501	30	转盘转动对位	6	156
4	3	60	210	216	816	824	845	853	1153	1169	1494	1506	1531	30	第一次温育	600	756
5	4	90	240	246	846	854	875	883	1183	1199	1524	1536	1561	30	转盘对位磁珠盒	8	764
6	5	120	270	276	876	884	905	913	1213	1229	1554	1566	1591	30	磁珠工位	21	785
7	6	150	300	306	906	914	935	943	1243	1259	1584	1596	1621	30	转盘转动对位	8	**793**
8	7	180	330	336	936	944	965	973	1273	1289	1614	1626	1651	30	第二次温育	300	1093
9	8	210	360	366	966	974	995	1003	1303	1319	1644	1656	1681	30	转盘对位清洗盒	16	1109
10	9	240	390	396	996	1004	1025	1033	1333	1434	1759	1771	1796	115	清洗工位	325	1434
11	10	270	420	426	1026	1034	1055	1063	1363	1464	1789	1801	1826	30	检测盘对位	12	1446
12	11	300	450	456	1056	1064	1085	1093	1393	1494	1819	1831	1856	30	检测工位	25	1471
13	12	330	480	486	1086	1094	1115	1123	1423	1524	1849	1861	1886	30		1471	
14	13	360	510	516	1116	1124	1145	1153	1453	1554	1879	1891	1916	30			
15	14	390	540	546	1146	1154	1175	1183	1483	1584	1909	1921	1946	30			
16	15	420	570	576	1176	1184	1205	1213	1513	1614	1939	1951	1976	30			
17	16	450	600	606	1206	1214	1235	1243	1543	1644	1969	1981	2006	30			
18	17	480	630	636	1236	1244	1265	1273	1573	1759	2084	2096	2121	115			
19	18	510	660	666	1266	1274	1295	1303	1603	1789	2114	2126	2151	30			
20	19	540	690	696	1296	1304	1325	1333	1633	1819	2144	2156	2181	30			
21	20	570	720	726	1326	1334	1355	1363	1663	1849	2174	2186	2211	30			
22	21	600	750	756	1356	1364	1385	1393	1693	1879	2204	2216	2241	30			
23	22	630	780	786	1386	1394	1415	1423	1723	1909	2234	2246	2271	30			
24	23	660	810	816	1416	1424	1445	1453	1753	1939	2264	2276	2301	30			
25	24	690	840	846	1446	1454	1475	1483	1783	1969	2294	2306	2331	30			
26	25	720	870	876	1476	1484	1505	1513	1813	2084	2409	2421	2446	115			
27	26	750	900	906	1506	1514	1535	1543	1843	2114	2439	2451	2476	30			

图 4.1 问题 1 中 A 型芯片整体调度结果

题目要求计算第一个小时内，可以处理的最多 A 芯片个数，查阅表格可知，第 55 枚芯片加工完毕时刻为 3601s，因此一个小时内最多可以处理完毕 54 枚 A 芯片.

表 4.1 A 型芯片第 55 个检测过程

芯片序号	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
55	1620	1776	2384	2413	3239	3576	3601

整个过程的甘特图见图 4.2，具体地，蓝色虚线表示清洗盒共有 8 个，当第 9 个芯片经过转盘对位清洗盒之后，要等待第 1 个芯片清洗完毕之后才能进入清洗盒，以此类推后续芯片同样要收到清洗盒数量的约束，所求结果满足题意.

同理，得到完整检测 1 个 B 型芯片所需时间最短为 4171s，超过 1h，故 1h 内可以调度的 B 型芯片数为 0 个.下表中展示的是最短的 4171s 的情况.

表 4.2 1 个 B 型芯片检测过程

芯片序号	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
0	0	(156)	(2564)	(2593)	(3809)	(4146)	(4171)

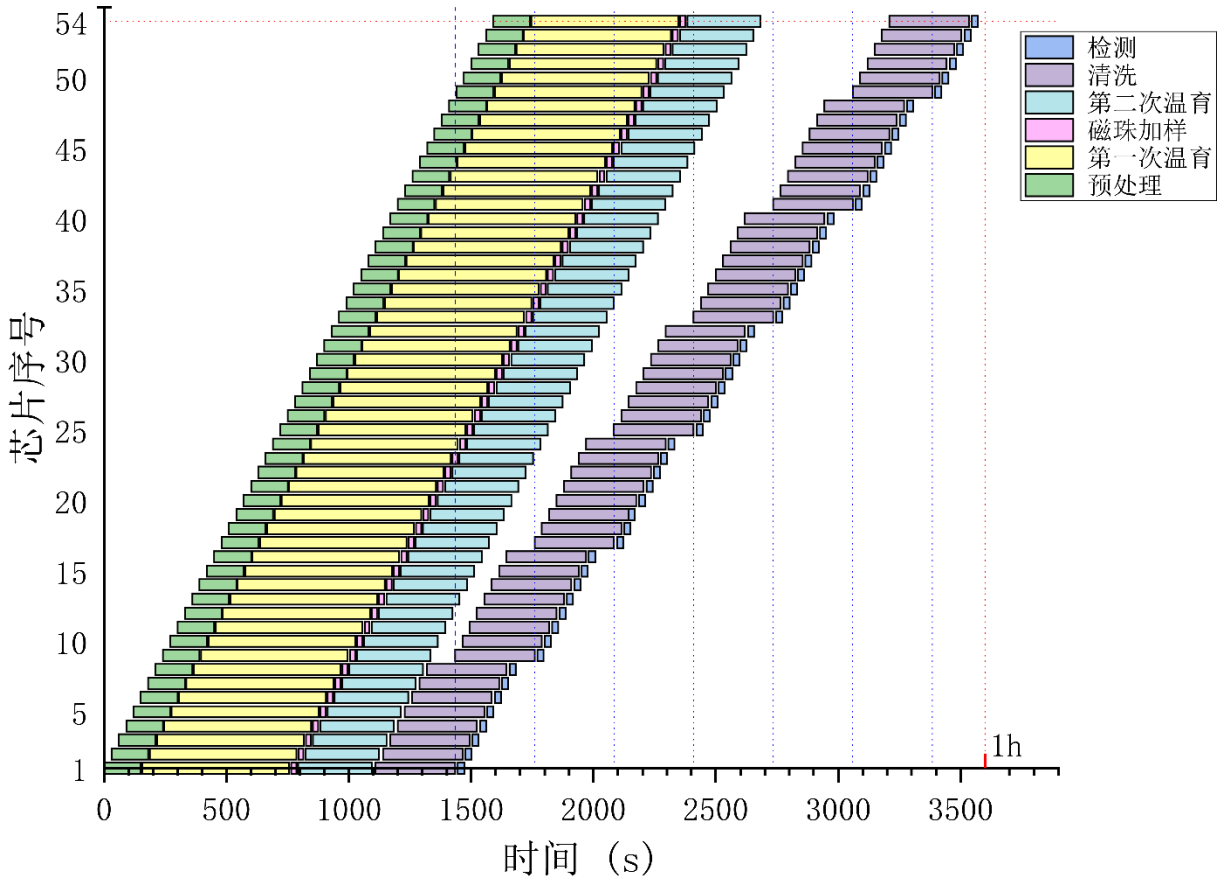


图 4.2 第一问结果 A 芯片甘特图

4.2 问题 2、3

对于第二， 三小问， 求解思路基本一致. 属于存在不同种类的芯片， 对于温育时间要求不一样. 因此在进行芯片检测时， 各类芯片的投放顺序会对检测结果产生较大的影响. 在对问题求解时， 通过对芯片种类进行分类讨论， 分别为每种芯片的每个工序工时进行赋值后， 设每次不同放入芯片的总顺序为一个不断进行排列组合的字符串，通过对数量与字母规定的该字符串调整字母存在顺序来进行编程计算.

由于该芯片数量比较理想， 因此我们在第二三小问选择了设定足够大的模拟次数进行局部搜索， 求出最小化最大加工时间的局部最优解， 当结果趋于稳定时， 所得到的字符串顺序即为最终的理想芯片投放顺序.

第二小问，只有 A，B 两芯片时，最终的稳定模拟结果见表 4.3

表 4.3 A、B 型芯片检测过程

芯片序号	芯片类型	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
1	A	0	156	764	793	1109	1446	1471
2	A	30	186	794	823	1139	1476	1501

3	A	60	216	824	853	1169	1506	1531
4	A	90	246	854	883	1199	1536	1561
5	B	120	276	2684	2713	3929	4266	4291
6	A	150	306	914	943	1259	1596	1621
7	B	180	336	2744	2773	3989	4326	4351
8	B	210	366	2774	2803	4019	4356	4381

芯片的投放顺序结果如下:

AAAABABBBAABBABBBBABAAABBABAAABBAAAABAABBBABBAAABABABABABBA
BBBAABAAABABBABBBBBABBABAAABBBBBBBABAABBABABBBBBAABABBBBABBBBA
ABBAABBBAAABBABAABBAABAAAAABAAAAAAAAAAA

全部完工的时间为: 8446s 后.

同理, 第三小问, 涉及 A,B,C 三种芯片进行运算时, 最终稳定的模拟结果见表 4.4.

表 4.4　A、B、C 芯片检测部分过程示意

芯片序号	芯片类型	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
1	B	0	156	2564	2593	3809	4146	4171
2	C	30	186	1694	1723	2339	2676	2701
3	B	60	216	2624	2653	3869	4206	4231
4	A	90	246	854	883	1199	1536	1561
5	B	120	276	2684	2713	3929	4266	4291
6	C	150	306	1814	1843	2459	2796	2821
7	A	180	336	944	973	1289	1626	1651
8	B	210	366	2774	2803	4019	4356	4381
9	B	240	396	2804	2833	4049	4386	4411
10	B	270	426	2834	2863	4079	4416	4441

芯片的投放顺序结果如下:

BCBABCABBBAAACABABABBBCBBBABABACBBACCCBBCABBBBACCABAABABCBAA
AAABCBACCACCCCBBBACCBCCACBBBBBBBBAAAAACAAACBABCCCAAABCCBACAACB
BBCAAAAABCCCCACACAACAACACACCCACACAAACC

全部完工的时间为: 8071s 后.

4.2 问题 4

第四问要求将算法模型推广至一般情况. 由于一般情况中, 对于芯片的种类与总数量有着更高的要求, 因此在进行模拟时, 需要引入更加高效的算法模型对其进行处理. 由于该问题满足柔性的流水线车间调度问题, 而此类问题已经被明确为是 NP 难问题, 不存在一个确定的算法可以求得全局最优解.

因此, 在处理问题四时, 我们通过改进的遗传算法, 对于可能的芯片处理顺序进行模拟, 使用高效的局部搜索策略得到局部最优解.

首先, 将芯片的不同种类数字符串编码转化为二进制编码字符串, 在符合芯片各种类所需数目要求的情况下, 通过随机生成的方式确定第一批种群数(解数) 已知的解集. 再通过设定阈值的方式, 选择父亲一代中, 可以交换染色体的解, 并且设定概率参数, 确定原种群中发生变异的染色体. 若子代染色体同样满足种类数目与题目相符这一条件, 则加入新种群, 反之则通过改变元素的种类个数方法使其满足条件, 对于新构建的种群, 计算适应度. 以此类推, 在迭代次数满足参数设置后, 所找到的即为最终的理想解.

所举例子为第二问的条件, A,B 芯片各 80 枚的情况, 基因数为 160, 种群数为 200, 交叉变异阈值分别为 0.8 和 0.005, 迭代次数为 500.

可以运行结果可以看出, 运用遗传算法, 所得的最小完全加工时间为 1645.001 秒, 优于原局部搜索算法, 芯片放入顺序为

['B', 'A', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B']

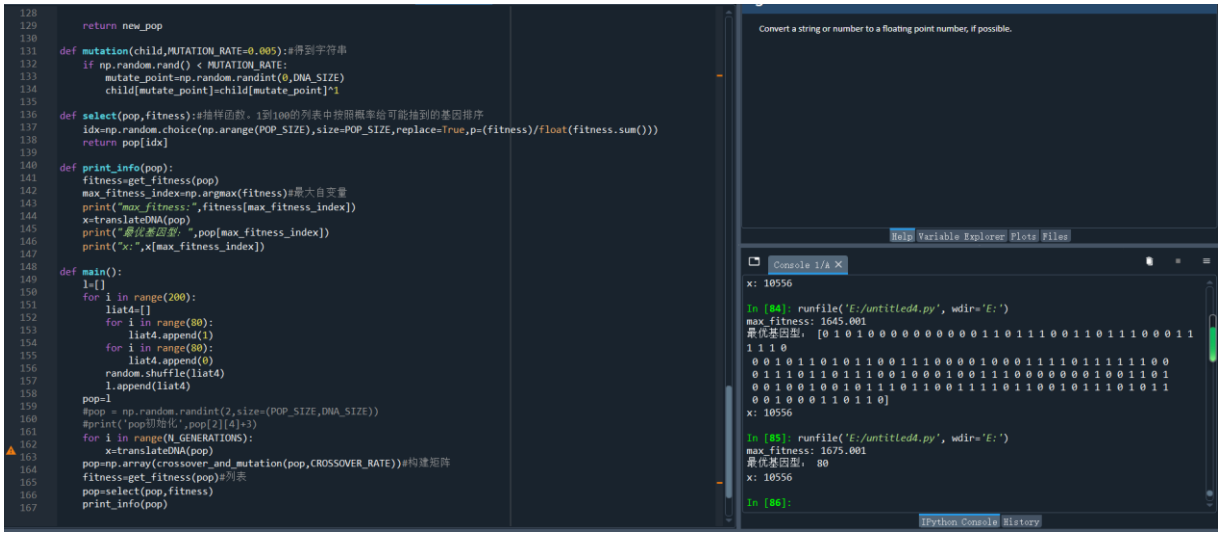
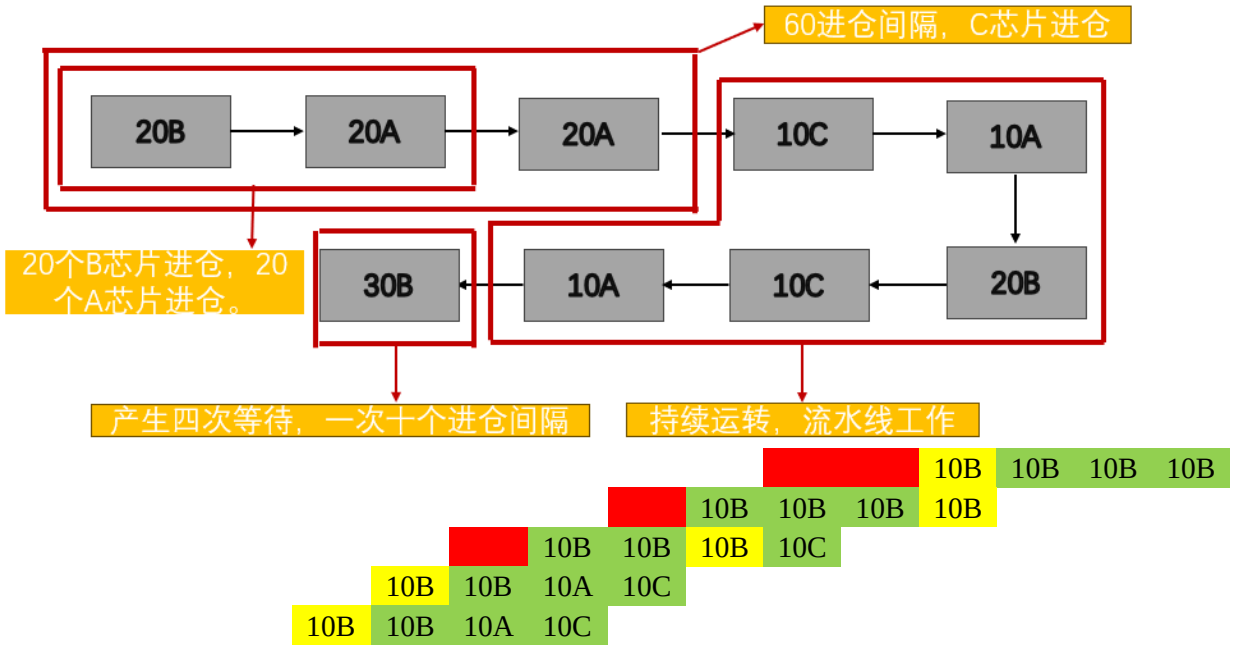


图 4.3 第四问实例结果图

在对于该小问进行实例验证时, 我们通过设置与第二问相同的参数, 使用改进后的遗传算法进行求解, 所得结果如下, 可以看出, 使用该模型算法后, 对于最小化最大时间的效果有了更明显的提升, 同时减少了最终解的不确定性.

4.4 问题 5



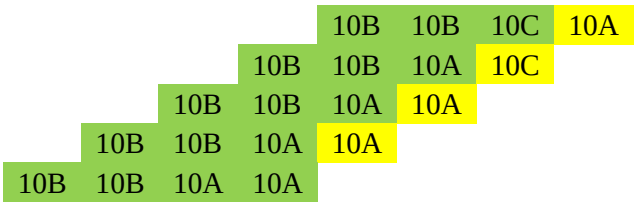


图 4.4 第五问进程图

流水线设计图: 根据第五题的柔性车间调度思想, 需要使得分析仪不断运转, 基于此, 给出的方案是 20B+40A+10C+10A+20B+10C+10A+30B, 绿色代表着上一步还在运行的芯片, 黄色代表新加入的芯片, 红色代表产生了时间间隔, 图中每格代表着 10 个进仓间隔, 由上图可知, 该方案仅产生四次等待, 能很好满足题目要求.

第五问求解, 刚开始有 A 芯片 60 片, B 芯片 70 片, 在运行 30 分钟后, 新到芯片 C 20 片, 题目需要我们解决检测完这些芯片至少需要多少时间?

问题分析: 由工序所需时长给出, 芯片进仓时间相差 30s, A 芯片的第一次温育时间为 10 分钟, 相当于 20 个进仓间隔, 这说明, 如果单独检测 A 芯片并且使其不间断运转, 我们至少需要 20 个卡槽; B 芯片的第一次温育时间 40 分钟(2400s) , 相当于 80 个进仓间隔, 这说明, 如果单独检测 B 芯片并且使其不间断运转, 至少需要 80 个卡槽; C 芯片的第一次温育时间 25 分钟(1500s) , 相当于 50 个进仓间隔, 这说明, 如果单独检测 C 芯片并且使其不间断运转, 至少需要 50 个卡槽; 根据问题分析, 为了使得其检测时间最短, 我们则需要使得分析仪不间断运转.

芯片序列计算: B 芯片第一次温育时间最久, C 芯片其次, A 芯片最短, 为了使其检测时间最短, 首先需要将 B 芯片进仓, 并利用 B 芯片处理间隔处理 A、C 芯片, 再根据 B 芯片处理间隔, 灵活安排 A、C 芯片, 根据 C 芯片, 灵活安排 A 芯片位置, 最终得出芯片处理序列 20B+40A+10C+10A+20B+10C+10A+30B.

5 模型评价

5.1 模型的优点

(1) 模型充分结合实际, 简化工件加工的转位过渡条件, 考虑了诸多重要因素得到合理的模型, 得到的模型贴合实际, 具有较高的应用价值, 可以推广到普遍情况;
(2) 模型运用柔性流水车间调度思想, 抓住影响时间问题的重要因素, 将复杂的加工顺序问题转化为简单的容量阈值问题, 合理设置参数, 模型的输出结果符合题目要求, 能解决实际问题;
(3) 本文使用的遗传算法具有效率高、输出稳定、成本低等特点, 对于求解 NP 难问题非常适用。

5.2 模型的不足

(4) 实际应用中,对于芯片种类转化为二进制字符串过程中, 可能由于字符串过长对于计算机的计算效率产生影响, 同时由于存在较多需要主观判断的参数, 对于参数的选择在一定程度上影响了模型的准确性;
(5) 本文提出的模型对于现有条件使用效果较好, 由于时间问题没有对其他情况进行检验. 对于其他情形(如芯片种类数巨量时), 可能无法达到较好的效果;

5.3 模型的推广

针对遗传算法容易陷入局部解这一问题, 对每个种群中评价函数值最高的染色体进行局部搜索. 对于调度问题的局部搜索问题, 现有研究通常采用有限次的交换、插入、逆序等邻域搜索策略. 后续研究中, 我们将采取循环交换、循环插入、循环逆序等三种邻域搜索算子, 以充分改善算法的局部搜索能力.

参考文献

[1]周鹏鹏, 翟志波, 戴玉森.基于改进遗传算法的柔性作业车间调度问题研究[J].组合机床与自动化加工技术,2023,(3): 183-186

[2]郝元峰.基于多目标优化算法的柔性流水车间生产调度问题研究[D].上海电机学院,2022

[3]苑明海, 张理志, 周开俊, 李亚东.基于混合遗传算法的柔性车间调度问题研究[J].工业工程与管理,2021,第 26 卷(6): 95-103

[4]李文韬.基于遗传算法的双目标流水车间调度问题研究[D].沈阳理工大学,2019

[5]杨帆, 方成刚, 吴伟伟.基于遗传-粒子群混合算法的柔性作业车间多资源调度问题[J].制造技术与机床,2020,(2): 138-142

.

附　录

附录 A：支撑材料列表

支撑材料列表

序号	文件名	材料说明
1	A 芯片调度结果.xlsx	问题 1 结果
2	A、B 型芯片调度结果.xlsx	问题 2 结果
3	A、B、C 型芯片调度结果.xlsx	问题 3 结果
4	问题 4 结果举例	问题 4 举例结果见图 4.3
5	第五问流水线工作图	问题 5 结果见图 4.4

附录 B：求解结果附表

附表 1：问题 1. A 型芯片的调度结果（单位：s）

芯片序号	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
1	0	156	764	793	1109	1446	1471
2	30	186	794	823	1139	1476	1501
3	60	216	824	853	1169	1506	1531
4	90	246	854	883	1199	1536	1561
5	120	276	884	913	1229	1566	1591
6	150	306	914	943	1259	1596	1621
7	180	336	944	973	1289	1626	1651
8	210	366	974	1003	1319	1656	1681
9	240	396	1004	1033	1434	1771	1796
10	270	426	1034	1063	1464	1801	1826
11	300	456	1064	1093	1494	1831	1856
12	330	486	1094	1123	1524	1861	1886
13	360	516	1124	1153	1554	1891	1916
14	390	546	1154	1183	1584	1921	1946
15	420	576	1184	1213	1614	1951	1976
16	450	606	1214	1243	1644	1981	2006
17	480	636	1244	1273	1759	2096	2121
18	510	666	1274	1303	1789	2126	2151
19	540	696	1304	1333	1819	2156	2181
20	570	726	1334	1363	1849	2186	2211
21	600	756	1364	1393	1879	2216	2241
22	630	786	1394	1423	1909	2246	2271
23	660	816	1424	1453	1939	2276	2301
24	690	846	1454	1483	1969	2306	2331
25	720	876	1484	1513	2084	2421	2446
26	750	906	1514	1543	2114	2451	2476

27	780	936	1544	1573	2144	2481	2506
28	810	966	1574	1603	2174	2511	2536
29	840	996	1604	1633	2204	2541	2566
30	870	1026	1634	1663	2234	2571	2596
31	900	1056	1664	1693	2264	2601	2626
32	930	1086	1694	1723	2294	2631	2656
33	960	1116	1724	1753	2409	2746	2771
34	990	1146	1754	1783	2439	2776	2801
35	1020	1176	1784	1813	2469	2806	2831
36	1050	1206	1814	1843	2499	2836	2861
37	1080	1236	1844	1873	2529	2866	2891
38	1110	1266	1874	1903	2559	2896	2921
39	1140	1296	1904	1933	2589	2926	2951
40	1170	1326	1934	1963	2619	2956	2981
41	1200	1356	1964	1993	2734	3071	3096
42	1230	1386	1994	2023	2764	3101	3126
43	1260	1416	2024	2053	2794	3131	3156
44	1290	1446	2054	2083	2824	3161	3186
45	1320	1476	2084	2113	2854	3191	3216
46	1350	1506	2114	2143	2884	3221	3246
47	1380	1536	2144	2173	2914	3251	3276
48	1410	1566	2174	2203	2944	3281	3306
49	1440	1596	2204	2233	3059	3396	3421
50	1470	1626	2234	2263	3089	3426	3451
51	1500	1656	2264	2293	3119	3456	3481
52	1530	1686	2294	2323	3149	3486	3511
53	1560	1716	2324	2353	3179	3516	3541
54	1590	1746	2354	2383	3209	3546	3571

附表 2：问题 1. B 型芯片的调度结果（单位：s）

完整检测 1 个 B 型芯片所需时间最短为 4171s，超过 1h，故 1h 内可以调度的 B 型芯片数为 0 个.下表中展示的是最短的 4171s 的情况，超过 1h.

芯片序号	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
0	0	(156)	(2564)	(2593)	(3809)	(4146)	(4171)

附表 3：问题 2. A、B 型芯片的调度结果（单位：s）

芯片序号	芯片类型	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
1	A	0	156	764	793	1109	1446	1471
2	A	30	186	794	823	1139	1476	1501
3	A	60	216	824	853	1169	1506	1531
4	A	90	246	854	883	1199	1536	1561
5	B	120	276	2684	2713	3929	4266	4291
6	A	150	306	914	943	1259	1596	1621

7	B	180	336	2744	2773	3989	4326	4351
8	B	210	366	2774	2803	4019	4356	4381
9	A	240	396	1004	1033	325	662	687
10	A	270	426	1034	1063	1379	1716	1741
11	A	300	456	1064	1093	1409	1746	1771
12	B	330	486	2894	2923	4139	4476	4501
13	A	360	516	1124	1153	1469	1806	1831
14	B	390	546	2954	2983	4199	4536	4561
15	B	420	576	2984	3013	4229	4566	4591
16	B	450	606	3014	3043	4259	4596	4621
17	B	480	636	3044	3073	4289	4626	4651
18	A	510	666	1274	1303	325	662	687
19	A	540	696	1304	1333	1649	1986	2011
20	B	570	726	3134	3163	4379	4716	4741
21	B	600	756	3164	3193	4409	4746	4771
22	B	630	786	3194	3223	4439	4776	4801
23	A	660	816	1424	1453	1769	2106	2131
24	A	690	846	1454	1483	1799	2136	2161
25	B	720	876	3284	3313	4529	4866	4891
26	B	750	906	3314	3343	4559	4896	4921
27	A	780	936	1544	1573	325	662	687
28	B	810	966	3374	3403	4619	4956	4981
29	A	840	996	1604	1633	1949	2286	2311
30	A	870	1026	1634	1663	1979	2316	2341
31	A	900	1056	1664	1693	2009	2346	2371
32	B	930	1086	3494	3523	4739	5076	5101
33	B	960	1116	3524	3553	4769	5106	5131
34	B	990	1146	3554	3583	4799	5136	5161
35	A	1020	1176	1784	1813	2129	2466	2491
36	A	1050	1206	1814	1843	325	662	687
37	A	1080	1236	1844	1873	2189	2526	2551
38	B	1110	1266	3674	3703	4919	5256	5281
39	A	1140	1296	1904	1933	2249	2586	2611
40	B	1170	1326	3734	3763	4979	5316	5341
41	B	1200	1356	3764	3793	5009	5346	5371
42	B	1230	1386	3794	3823	5039	5376	5401
43	B	1260	1416	3824	3853	5069	5406	5431
44	B	1290	1446	3854	3883	5099	5436	5461
45	B	1320	1476	3884	3913	5129	5466	5491
46	B	1350	1506	3914	3943	5159	5496	5521
47	B	1380	1536	3944	3973	5189	5526	5551
48	B	1410	1566	3974	4003	325	662	687
49	B	1440	1596	4004	4033	5249	5586	5611
50	A	1470	1626	2234	2263	2579	2916	2941
51	B	1500	1656	4064	4093	5309	5646	5671
52	B	1530	1686	4094	4123	5339	5676	5701

53	B	1560	1716	4124	4153	5369	5706	5731
54	B	1590	1746	4154	4183	5399	5736	5761
55	B	1620	1776	4184	4213	5429	5766	5791
56	A	1650	1806	2414	2443	2759	3096	3121
57	B	1680	1836	4244	4273	5489	5826	5851
58	A	1710	1866	2474	2503	325	662	687
59	A	1740	1896	2504	2533	2849	3186	3211
60	A	1770	1926	2534	2563	2879	3216	3241
61	A	1800	1956	2564	2593	2909	3246	3271
62	B	1830	1986	4394	4423	5639	5976	6001
63	B	1860	2016	4424	4453	5669	6006	6031
64	A	1890	2046	2654	2683	2999	3336	3361
65	A	1920	2076	2684	2713	3029	3366	3391
66	B	1950	2106	4514	4543	5759	6096	6121
67	B	1980	2136	4544	4573	5789	6126	6151
68	B	2010	2166	4574	4603	5819	6156	6181
69	B	2040	2196	4604	4633	5849	6186	6211
70	B	2070	2226	4634	4663	5879	6216	6241
71	A	2100	2256	2864	2893	325	662	687
72	A	2130	2286	2894	2923	3239	3576	3601
73	B	2160	2316	4724	4753	5969	6306	6331
74	B	2190	2346	4754	4783	5999	6336	6361
75	A	2220	2376	2984	3013	3329	3666	3691
76	B	2250	2406	4814	4843	6059	6396	6421
77	A	2280	2436	3044	3073	3389	3726	3751
78	B	2310	2466	4874	4903	6119	6456	6481
79	B	2340	2496	4904	4933	6149	6486	6511
80	B	2370	2526	4934	4963	6179	6516	6541
81	A	2400	2556	3164	3193	325	662	687
82	B	2430	2586	4994	5023	6239	6576	6601
83	A	2460	2616	3224	3253	3569	3906	3931
84	A	2490	2646	3254	3283	3599	3936	3961
85	A	2520	2676	3284	3313	3629	3966	3991
86	B	2550	2706	5114	5143	6359	6696	6721
87	B	2580	2736	5144	5173	6389	6726	6751
88	A	2610	2766	3374	3403	3719	4056	4081
89	A	2640	2796	3404	3433	3749	4086	4111
90	B	2670	2826	5234	5263	6479	6816	6841
91	B	2700	2856	5264	5293	6509	6846	6871
92	B	2730	2886	5294	5323	6539	6876	6901
93	B	2760	2916	5324	5353	6569	6906	6931
94	A	2790	2946	3554	3583	325	662	687
95	B	2820	2976	5384	5413	6629	6966	6991
96	B	2850	3006	5414	5443	6659	6996	7021
97	B	2880	3036	5444	5473	6689	7026	7051
98	A	2910	3066	3674	3703	4019	4356	4381

99	A	2940	3096	3704	3733	4049	4386	4411
100	A	2970	3126	3734	3763	4079	4416	4441
101	A	3000	3156	3764	3793	4109	4446	4471
102	A	3030	3186	3794	3823	4139	4476	4501
103	A	3060	3216	3824	3853	325	662	687
104	B	3090	3246	5654	5683	6899	7236	7261
105	A	3120	3276	3884	3913	4229	4566	4591
106	A	3150	3306	3914	3943	4259	4596	4621
107	B	3180	3336	5744	5773	6989	7326	7351
108	B	3210	3366	5774	5803	7019	7356	7381
109	B	3240	3396	5804	5833	7049	7386	7411
110	B	3270	3426	5834	5863	7079	7416	7441
111	B	3300	3456	5864	5893	7109	7446	7471
112	B	3330	3486	5894	5923	7139	7476	7501
113	B	3360	3516	5924	5953	7169	7506	7531
114	A	3390	3546	4154	4183	325	662	687
115	B	3420	3576	5984	6013	7229	7566	7591
116	A	3450	3606	4214	4243	4559	4896	4921
117	B	3480	3636	6044	6073	7289	7626	7651
118	A	3510	3666	4274	4303	4619	4956	4981
119	A	3540	3696	4304	4333	4649	4986	5011
120	A	3570	3726	4334	4363	4679	5016	5041
121	B	3600	3756	6164	6193	7409	7746	7771
122	A	3630	3786	4394	4423	4739	5076	5101
123	A	3660	3816	4424	4453	325	662	687
124	A	3690	3846	4454	4483	4799	5136	5161
125	B	3720	3876	6284	6313	7529	7866	7891
126	B	3750	3906	6314	6343	7559	7896	7921
127	A	3780	3936	4544	4573	4889	5226	5251
128	B	3810	3966	6374	6403	7619	7956	7981
129	A	3840	3996	4604	4633	4949	5286	5311
130	B	3870	4026	6434	6463	7679	8016	8041
131	A	3900	4056	4664	4693	5009	5346	5371
132	A	3930	4086	4694	4723	325	662	687
133	B	3960	4116	6524	6553	7769	8106	8131
134	B	3990	4146	6554	6583	7799	8136	8161
135	B	4020	4176	6584	6613	7829	8166	8191
136	A	4050	4206	4814	4843	5159	5496	5521
137	B	4080	4236	6644	6673	7889	8226	8251
138	A	4110	4266	4874	4903	5219	5556	5581
139	A	4140	4296	4904	4933	5249	5586	5611
140	B	4170	4326	6734	6763	7979	8316	8341
141	A	4200	4356	4964	4993	325	662	687
142	A	4230	4386	4994	5023	5339	5676	5701
143	B	4260	4416	6824	6853	8069	8406	8431
144	A	4290	4446	5054	5083	5399	5736	5761

145	B	4320	4476	6884	6913	8129	8466	8491
146	A	4350	4506	5114	5143	5459	5796	5821
147	A	4380	4536	5144	5173	5489	5826	5851
148	B	4410	4566	6974	7003	8219	8556	8581
149	A	4440	4596	5204	5233	5549	5886	5911
150	A	4470	4626	5234	5263	325	662	687
151	A	4500	4656	5264	5293	5609	5946	5971
152	A	4530	4686	5294	5323	5639	5976	6001
153	A	4560	4716	5324	5353	5669	6006	6031
154	A	4590	4746	5354	5383	5699	6036	6061
155	A	4620	4776	5384	5413	5729	6066	6091
156	A	4650	4806	5414	5443	5759	6096	6121
157	A	4680	4836	5444	5473	5789	6126	6151
158	A	4710	4866	5474	5503	5819	6156	6181
159	A	4740	4896	5504	5533	325	662	687
160	A	4770	4926	5534	5563	5879	6216	6241

附表 3：问题 2. A、B 型芯片的调度结果（单位：s）

芯片序号	芯片类型	进仓时间	第一次温育起始时间	磁珠加样起始时间	第二次温育起始时间	清洗起始时间	检测起始时间	检测结束时间
1	A	0	156	764	793	1109	1446	1471
1	B	0	156	2564	2593	3809	4146	4171
2	C	30	186	1694	1723	2339	2676	2701
3	B	60	216	2624	2653	3869	4206	4231
4	A	90	246	854	883	1199	1536	1561
5	B	120	276	2684	2713	3929	4266	4291
6	C	150	306	1814	1843	2459	2796	2821
7	A	180	336	944	973	1289	1626	1651
8	B	210	366	2774	2803	4019	4356	4381
9	B	240	396	2804	2833	4049	4386	4411
10	B	270	426	2834	2863	4079	4416	4441
11	A	300	456	1064	1093	325	662	687
12	A	330	486	1094	1123	1439	1776	1801
13	A	360	516	1124	1153	1469	1806	1831
14	C	390	546	2054	2083	2699	3036	3061
15	A	420	576	1184	1213	1529	1866	1891
16	B	450	606	3014	3043	4259	4596	4621
17	A	480	636	1244	1273	1589	1926	1951
18	B	510	666	3074	3103	4319	4656	4681
19	A	540	696	1304	1333	1649	1986	2011
20	B	570	726	3134	3163	4379	4716	4741
21	B	600	756	3164	3193	4409	4746	4771
22	B	630	786	3194	3223	4439	4776	4801
23	C	660	816	2324	2353	2969	3306	3331
24	B	690	846	3254	3283	4499	4836	4861

25	B	720	876	3284	3313	4529	4866	4891
26	B	750	906	3314	3343	4559	4896	4921
27	A	780	936	1544	1573	325	662	687
28	B	810	966	3374	3403	4619	4956	4981
29	A	840	996	1604	1633	1949	2286	2311
30	B	870	1026	3434	3463	4679	5016	5041
31	A	900	1056	1664	1693	2009	2346	2371
32	C	930	1086	2594	2623	3239	3576	3601
33	B	960	1116	3524	3553	4769	5106	5131
34	B	990	1146	3554	3583	4799	5136	5161
35	A	1020	1176	1784	1813	2129	2466	2491
36	C	1050	1206	2714	2743	3359	3696	3721
37	C	1080	1236	2744	2773	3389	3726	3751
38	C	1110	1266	2774	2803	3419	3756	3781
39	B	1140	1296	3704	3733	4949	5286	5311
40	B	1170	1326	3734	3763	4979	5316	5341
41	C	1200	1356	2864	2893	3509	3846	3871
42	A	1230	1386	1994	2023	325	662	687
43	B	1260	1416	3824	3853	5069	5406	5431
44	B	1290	1446	3854	3883	5099	5436	5461
45	B	1320	1476	3884	3913	5129	5466	5491
46	B	1350	1506	3914	3943	5159	5496	5521
47	A	1380	1536	2144	2173	2489	2826	2851
48	C	1410	1566	3074	3103	3719	4056	4081
49	C	1440	1596	3104	3133	3749	4086	4111
50	A	1470	1626	2234	2263	2579	2916	2941
51	B	1500	1656	4064	4093	5309	5646	5671
52	A	1530	1686	2294	2323	325	662	687
53	A	1560	1716	2324	2353	2669	3006	3031
54	B	1590	1746	4154	4183	5399	5736	5761
55	A	1620	1776	2384	2413	2729	3066	3091
56	B	1650	1806	4214	4243	5459	5796	5821
57	C	1680	1836	3344	3373	3989	4326	4351
58	B	1710	1866	4274	4303	5519	5856	5881
59	A	1740	1896	2504	2533	2849	3186	3211
60	A	1770	1926	2534	2563	2879	3216	3241
61	A	1800	1956	2564	2593	325	662	687
62	A	1830	1986	2594	2623	2939	3276	3301
63	A	1860	2016	2624	2653	2969	3306	3331
64	B	1890	2046	4454	4483	5699	6036	6061
65	C	1920	2076	3584	3613	4229	4566	4591
66	B	1950	2106	4514	4543	5759	6096	6121
67	A	1980	2136	2744	2773	3089	3426	3451
68	C	2010	2166	3674	3703	4319	4656	4681
69	C	2040	2196	3704	3733	4349	4686	4711
70	A	2070	2226	2834	2863	325	662	687

71	C	2100	2256	3764	3793	4409	4746	4771
72	C	2130	2286	3794	3823	4439	4776	4801
73	C	2160	2316	3824	3853	4469	4806	4831
74	C	2190	2346	3854	3883	4499	4836	4861
75	B	2220	2376	4784	4813	6029	6366	6391
76	B	2250	2406	4814	4843	6059	6396	6421
77	B	2280	2436	4844	4873	6089	6426	6451
78	A	2310	2466	3074	3103	3419	3756	3781
79	C	2340	2496	4004	4033	4649	4986	5011
80	C	2370	2526	4034	4063	4679	5016	5041
81	B	2400	2556	4964	4993	6209	6546	6571
82	C	2430	2586	4094	4123	4739	5076	5101
83	C	2460	2616	4124	4153	4769	5106	5131
84	A	2490	2646	3254	3283	325	662	687
85	C	2520	2676	4184	4213	4829	5166	5191
86	B	2550	2706	5114	5143	6359	6696	6721
87	B	2580	2736	5144	5173	6389	6726	6751
88	B	2610	2766	5174	5203	6419	6756	6781
89	B	2640	2796	5204	5233	6449	6786	6811
90	B	2670	2826	5234	5263	6479	6816	6841
91	B	2700	2856	5264	5293	6509	6846	6871
92	B	2730	2886	5294	5323	6539	6876	6901
93	B	2760	2916	5324	5353	6569	6906	6931
94	A	2790	2946	3554	3583	325	662	687
95	A	2820	2976	3584	3613	3929	4266	4291
96	A	2850	3006	3614	3643	3959	4296	4321
97	A	2880	3036	3644	3673	3989	4326	4351
98	A	2910	3066	3674	3703	4019	4356	4381
99	C	2940	3096	4604	4633	5249	5586	5611
100	A	2970	3126	3734	3763	4079	4416	4441
101	A	3000	3156	3764	3793	4109	4446	4471
102	A	3030	3186	3794	3823	4139	4476	4501
103	C	3060	3216	4724	4753	5369	5706	5731
104	B	3090	3246	5654	5683	6899	7236	7261
105	A	3120	3276	3884	3913	325	662	687
106	B	3150	3306	5714	5743	6959	7296	7321
107	C	3180	3336	4844	4873	5489	5826	5851
108	C	3210	3366	4874	4903	5519	5856	5881
109	C	3240	3396	4904	4933	5549	5886	5911
110	A	3270	3426	4034	4063	4379	4716	4741
111	A	3300	3456	4064	4093	4409	4746	4771
112	A	3330	3486	4094	4123	4439	4776	4801
113	B	3360	3516	5924	5953	7169	7506	7531
114	C	3390	3546	5054	5083	5699	6036	6061
115	C	3420	3576	5084	5113	5729	6066	6091
116	B	3450	3606	6014	6043	7259	7596	7621

117	A	3480	3636	4244	4273	325	662	687
118	C	3510	3666	5174	5203	5819	6156	6181
119	A	3540	3696	4304	4333	4649	4986	5011
120	A	3570	3726	4334	4363	4679	5016	5041
121	C	3600	3756	5264	5293	5909	6246	6271
122	B	3630	3786	6194	6223	7439	7776	7801
123	B	3660	3816	6224	6253	7469	7806	7831
124	B	3690	3846	6254	6283	7499	7836	7861
125	C	3720	3876	5384	5413	6029	6366	6391
126	A	3750	3906	4514	4543	325	662	687
127	A	3780	3936	4544	4573	4889	5226	5251
128	A	3810	3966	4574	4603	4919	5256	5281
129	A	3840	3996	4604	4633	4949	5286	5311
130	A	3870	4026	4634	4663	4979	5316	5341
131	B	3900	4056	6464	6493	7709	8046	8071
132	C	3930	4086	5594	5623	6239	6576	6601
133	C	3960	4116	5624	5653	6269	6606	6631
134	C	3990	4146	5654	5683	6299	6636	6661
135	C	4020	4176	5684	5713	6329	6666	6691
136	A	4050	4206	4814	4843	325	662	687
137	C	4080	4236	5744	5773	6389	6726	6751
138	A	4110	4266	4874	4903	5219	5556	5581
139	C	4140	4296	5804	5833	6449	6786	6811
140	A	4170	4326	4934	4963	5279	5616	5641
141	A	4200	4356	4964	4993	5309	5646	5671
142	C	4230	4386	5894	5923	6539	6876	6901
143	A	4260	4416	5024	5053	5369	5706	5731
144	A	4290	4446	5054	5083	5399	5736	5761
145	C	4320	4476	5984	6013	6629	6966	6991
146	A	4350	4506	5114	5143	325	662	687
147	C	4380	4536	6044	6073	6689	7026	7051
148	A	4410	4566	5174	5203	5519	5856	5881
149	C	4440	4596	6104	6133	6749	7086	7111
150	C	4470	4626	6134	6163	6779	7116	7141
151	C	4500	4656	6164	6193	6809	7146	7171
152	A	4530	4686	5294	5323	5639	5976	6001
153	C	4560	4716	6224	6253	6869	7206	7231
154	A	4590	4746	5354	5383	5699	6036	6061
155	C	4620	4776	6284	6313	6929	7266	7291
156	A	4650	4806	5414	5443	325	662	687
157	A	4680	4836	5444	5473	5789	6126	6151
158	A	4710	4866	5474	5503	5819	6156	6181
159	C	4740	4896	6404	6433	7049	7386	7411
160	C	4770	4926	6434	6463	7079	7416	7441

附录 C：主要程序/关键代码

代码环境	操作系统: macOS Mojave (Version 10.14.3) 编程语言: Python 3.7.1 (Anaconda Navigator 1.9.2) 编辑器: PyCharm 2018.3.2 (Professional Edition) 代码详见: Code/Combine_Pyecharts_with_igraph.py

代码清单 1 第二问代码

```
#调度优化函数
    def schedule_optimization(s):
        list_index=s
        start=-30
        start_jincang=[]
        start_jincang.append(0)
        first_wenrun=[]
        cizhujiayang=[]
        second_wenrun=[]
        start_qingxi=[]
        start_jiance=[]
        end_jiance=[]
        slide_array=[]
        slide_wenrun=[]
        for i in range(len(list_index)):
            if i>=1:
                start_jincang.append(start_jincang[i-1]+30)
            if i>=8:
                slide_array=start_qingxi[i-8:i]
                if 0 in slide_array:
                    slide_array[slide_array.index(0)]=start_qingxi[i]
            if i>=40:
                slide_wenrun=first_wenrun[i-40:i]
                if 0 in slide_wenrun:
                    slide_wenrun[slide_wenrun.index(0)]=first_wenrun[i]
            if list_index[i]=='A':
                first_wenrun.append(start_jincang[i]+156)
                cizhujiayang.append(first_wenrun[i]+608)
                second_wenrun.append(cizhujiayang[i]+29)
                start_qingxi.append(second_wenrun[i]+316)
                start_jiance.append(start_qingxi[i]+337)
                end_jiance.append(start_jiance[i]+25)
            else:
                first_wenrun.append(start_jincang[i]+156)
                cizhujiayang.append(first_wenrun[i]+2408)
                second_wenrun.append(cizhujiayang[i]+29)
                start_qingxi.append(second_wenrun[i]+1216)
                start_jiance.append(start_qingxi[i]+337)
                end_jiance.append(start_jiance[i]+25)
```

```
            if i>=8 and start_qingxi[i]-min(slide_array)<325:
                start_qingxi[i]=min(slide_array)+325
                start_jiance[i]=start_qingxi[i]+337
                end_jiance[i]=start_jiance[i]+25
                slide_array[slide_array.index(min(slide_array))]=0
            if i>=40 and first_wenrun[i]-min(slide_wenrun)<1200:
                first_wenrun[i]=min(slide_wenrun)+1200
                slide_wenrun[slide_wenrun.index(min(slide_wenrun))]=0
                if list_index=='A':
                    cizhujiayang[i]=first_wenrun[i]+608
                    second_wenrun[i]=cizhujiayang[i]+29
                    start_qingxi[i]=second_wenrun[i]+316
                    start_jiance[i]=start_qingxi[i]+337
                    end_jiance[i]=start_jiance[i]+25
                else:
                    cizhujiayang[i]=first_wenrun[i]+2408
                    second_wenrun[i]=cizhujiayang[i]+29
                    start_qingxi[i]=second_wenrun[i]+1216
                    start_jiance[i]=start_qingxi[i]+337
                    end_jiance[i]=start_jiance[i]+25
    return end_jiance,start_jincang,first_wenrun,cizhujiayang,second_wenrun,start_qingxi,start_jiance
    initial_list=80*'A'+80*'B'
    #打乱字符串序列
    from random import shuffle
    def shuffle_str(s):
        # 将字符串转换成列表
        str_list = list(s)
        # 调用random模块的shuffle函数打乱列表
        shuffle(str_list)
        # 将列表转字符串
        return ''.join(str_list)
    str_ab=[]
    # 调用
    if __name__ == '__main__':
        for i in range(10000):
            str_ab.append(shuffle_str(initial_list))
    min_jiance=max(schedule_optimization(str_ab[0])[0])
    for j in range(1,len(str_ab)):
        if min_jiance>max(schedule_optimization(str_ab[j])[0]):
            min_jiance=max(schedule_optimization(str_ab[j])[0])
            result_str=str_ab[j]
            result_jincang=schedule_optimization(str_ab[j])[1]
            result_first_wenrun=schedule_optimization(str_ab[j])[2]
            result_cizhujiayang=schedule_optimization(str_ab[j])[3]
            result_second_wenrun=schedule_optimization(str_ab[j])[4]
            result_start_qingxi=schedule_optimization(str_ab[j])[5]
            result_start_jiance=schedule_optimization(str_ab[j])[6]
            result_end_jiance=schedule_optimization(str_ab[j])[0]
    import pandas as pd
```

```
    data_result=pd.DataFrame({'芯片序号':range(1,161),'芯片类型':list(result_str),'进仓时间':result_jincang,'第一次温育起始时间':result_first_wenrun,'磁珠加样起始时间':result_cizhujiayang,'第二次温育起始时间':result_second_wenrun,'清洗起始时间':result_start_qingxi,'检测起始时间':result_start_jiance,'检测结束时间':result_end_jiance})
```

代码清单 2 第三问代码

```
#调度优化函数
    def schedule_optimization(s):
        list_index=s
        start=-30
        start_jincang=[]
        start_jincang.append(0)
        first_wenrun=[]
        cizhujiayang=[]
        second_wenrun=[]
        start_qingxi=[]
        start_jiance=[]
        end_jiance=[]
        slide_array=[]
        slide_wenrun=[]
        for i in range(len(list_index)):
            if i>=1:
                start_jincang.append(start_jincang[i-1]+30)
            if i>=8:
                slide_array=start_qingxi[i-8:i]
                if 0 in slide_array:
                    slide_array[slide_array.index(0)]=start_qingxi[i]
            if i>=40:
                slide_wenrun=first_wenrun[i-40:i]
                if 0 in slide_wenrun:
                    slide_wenrun[slide_wenrun.index(0)]=first_wenrun[i]
            if list_index[i]=='A':
                first_wenrun.append(start_jincang[i]+156)
                cizhujiayang.append(first_wenrun[i]+608)
                second_wenrun.append(cizhujiayang[i]+29)
                start_qingxi.append(second_wenrun[i]+316)
                start_jiance.append(start_qingxi[i]+337)
                end_jiance.append(start_jiance[i]+25)
            else:
                first_wenrun.append(start_jincang[i]+156)
                cizhujiayang.append(first_wenrun[i]+2408)
                second_wenrun.append(cizhujiayang[i]+29)
                start_qingxi.append(second_wenrun[i]+1216)
                start_jiance.append(start_qingxi[i]+337)
                end_jiance.append(start_jiance[i]+25)
            if i>=8 and start_qingxi[i]-min(slide_array)<325:
                slide_array[slide_array.index(min(slide_array))]=0
                start_qingxi[i]=min(slide_array)+325
                start_jiance[i]=start_qingxi[i]+337
```

```
                    end_jiance[i]=start_jiance[i]+25
                if i>=40 and first_wenrun[i]-min(slide_wenrun)<1200:
                    first_wenrun[i]=min(slide_wenrun)+1200
                    slide_wenrun[slide_wenrun.index(min(slide_wenrun))]=0
                    if list_index=='A':
                        cizhujiayang[i]=first_wenrun[i]+608
                        second_wenrun[i]=cizhujiayang[i]+29
                        start_qingxi[i]=second_wenrun[i]+316
                        start_jiance[i]=start_qingxi[i]+337
                        end_jiance[i]=start_jiance[i]+25
                    else:
                        cizhujiayang[i]=first_wenrun[i]+2408
                        second_wenrun[i]=cizhujiayang[i]+29
                        start_qingxi[i]=second_wenrun[i]+1216
                        start_jiance[i]=start_qingxi[i]+337
                        end_jiance[i]=start_jiance[i]+25
    return end_jiance,start_jincang,first_wenrun,cizhujiayang,second_wenrun,start_qingxi,start_jiance
    #60个A和50个B、50个C
    initial_list=60*'A'+50*'B'+50*'C'
    #打乱字符串序列
    from random import shuffle
    def shuffle_str(s):
        # 将字符串转换成列表
        str_list = list(s)
        # 调用random模块的shuffle函数打乱列表
        shuffle(str_list)
        # 将列表转字符串
        return ''.join(str_list)
    str_ab=[]
    # 调用
    if __name__ == '__main__':
        for i in range(10000):
            str_ab.append(shuffle_str(initial_list))
    min_jiance=max(schedule_optimization(str_ab[0])[0])
    for j in range(1,len(str_ab)):
        if min_jiance>max(schedule_optimization(str_ab[j])[0]):
            min_jiance=max(schedule_optimization(str_ab[j])[0])
            result_str=str_ab[j]
            result_jincang=schedule_optimization(str_ab[j])[1]
            result_first_wenrun=schedule_optimization(str_ab[j])[2]
            result_cizhujiayang=schedule_optimization(str_ab[j])[3]
            result_second_wenrun=schedule_optimization(str_ab[j])[4]
            result_start_qingxi=schedule_optimization(str_ab[j])[5]
            result_start_jiance=schedule_optimization(str_ab[j])[6]
            result_end_jiance=schedule_optimization(str_ab[j])[0]
    import pandas as pd
    data_result=pd.DataFrame({'芯片序号':range(1,161),'芯片类型':list(result_str),'进仓时间':result_jincang,'第一次温育起始时间':result_first_wenrun,'磁珠加样起始时间':result_cizhujiayang,'第二次温育起始时间':result_second_wenrun,'清洗起始时间':result_start_qingxi,'检测起始时间':result_start_jiance,'检测结束时间':result_end_jiance})
```

代码清单 3 第四问遗传算法

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 26 18:52:06 2023

@author: gmy
"""
import numpy as np
import random
DNA_SIZE=160
POP_SIZE=200
CROSSOVER_RATE=0.8
MUTATION_RATE=0.005
N_GENERATIONS=500
def schedule_optimization(s):
    list_index=s
    start=-30
    start_jincang=[]
    start_jincang.append(0)
    first_wenrun=[]
    cizhujiayang=[]
    second_wenrun=[]
    start_qingxi=[]
    start_jiance=[]
    end_jiance=[]
    slide_array=[]
    slide_wenrun=[]
    for i in range(len(list_index)):
        if i>=1:
            start_jincang.append(start_jincang[i-1]+30)
        if i>=8:
            slide_array=start_qingxi[i-8:i]
            if 0 in slide_array:
                slide_array[slide_array.index(0)]=start_qingxi[i]
        if i>=40:
            slide_wenrun=first_wenrun[i-40:i]
            if 0 in slide_wenrun:
                slide_wenrun[slide_wenrun.index(0)]=first_wenrun[i]
        if list_index[i]==1:
            first_wenrun.append(start_jincang[i]+156)
            cizhujiayang.append(first_wenrun[i]+608)
            second_wenrun.append(cizhujiayang[i]+29)
            start_qingxi.append(second_wenrun[i]+316)
            start_jiance.append(start_qingxi[i]+337)
            end_jiance.append(start_jiance[i]+25)
        else:
            first_wenrun.append(start_jincang[i]+156)
            cizhujiayang.append(first_wenrun[i]+2408)
```

```
                    second_wenrun.append(cizhujiayang[i]+29)
                    start_qingxi.append(second_wenrun[i]+1216)
                    start_jiance.append(start_qingxi[i]+337)
                    end_jiance.append(start_jiance[i]+25)
                if i>=8 and start_qingxi[i]-min(slide_array)<325:
                    start_qingxi[i]=min(slide_array)+325
                    slide_array[slide_array.index(min(slide_array))]=0
                    start_jiance[i]=start_qingxi[i]+337
                    end_jiance[i]=start_jiance[i]+25
                if i>=40 and first_wenrun[i]-min(slide_wenrun)<1200:
                    first_wenrun[i]=min(slide_wenrun)+1200
                    slide_wenrun[slide_wenrun.index(min(slide_wenrun))]=0
                    if list_index==1:
                        cizhujiayang[i]=first_wenrun[i]+608
                        second_wenrun[i]=cizhujiayang[i]+29
                        start_qingxi[i]=second_wenrun[i]+316
                        start_jiance[i]=start_qingxi[i]+337
                        end_jiance[i]=start_jiance[i]+25
                    else:
                        cizhujiayang[i]=first_wenrun[i]+2408
                        second_wenrun[i]=cizhujiayang[i]+29
                        start_qingxi[i]=second_wenrun[i]+1216
                        start_jiance[i]=start_qingxi[i]+337
                        end_jiance[i]=start_jiance[i]+25
            return max(end_jiance)
        '''def f(x):#求目标函数值
            return 10*np.sin(5*x)+7*np.cos(4*x)'''
        def translateDNA(pop):#求出种群中各个解生成列表
            x=[]
            #print('pop是',pop.shape[1])
            for i in range(POP_SIZE):
                num=0
                list1=pop[i]
                num=schedule_optimization(list1)
                x.append(num)
            return x
            '''x=pop.dot(2**np.arange(DNA_SIZE)[::-1])/float(2**DNA_SIZE-1)*(X_BOUND[1]-X_BOUND[0])+X_BOUND[0]
            print('x=',x[10,2])
            print(len(x))
            return x'''
        def get_fitness(pop):
            x=translateDNA(pop)
            pred=x
            #pred=f(x)
            #print('pred为',pred)#适应度，
            return (pred-np.min(pred))+1e-3
        def crossover_and_mutation(pop,CROSSOVER_RATE=0.8):#得到矩阵
            new_pop=[]
            for father in pop:
```

```
            child = father
            if np.random.rand() < CROSSOVER_RATE:
                mother = pop[np.random.randint(POP_SIZE)]
                cross_points=np.random.randint(low=0,high=DNA_SIZE)
                child[cross_points:]=mother[cross_points:]
            mutation(child)
            if(sum(child)==80):
                new_pop.append(child)
            else:
                i=0
                while(sum(child)!=80):
                    child[i]=child[i]^1
                    i+=1
                new_pop.append(child)
        return new_pop
    def mutation(child,MUTATION_RATE=0.005):#得到字符串
        if np.random.rand() < MUTATION_RATE:
            mutate_point=np.random.randint(0,DNA_SIZE)
            child[mutate_point]=child[mutate_point]^1
    def select(pop,fitness):#抽样函数。1到100的列表中按照概率给可能抽到的基因排序
        idx=np.random.choice(np.arange(POP_SIZE),size=POP_SIZE,replace=True,p=(fitness)/float(fitn
ess.sum()))
        return pop[idx]
    def print_info(pop):
        fitness=get_fitness(pop)
        max_fitness_index=np.argmax(fitness)#最大自变量
        print("max_fitness:",fitness[max_fitness_index])
        x=translateDNA(pop)
        print("最优基因型：",pop[max_fitness_index])
        print("x:",x[max_fitness_index])
    def main():
        l=[]
        for i in range(200):
            liat4=[]
            for i in range(80):
                liat4.append(1)
            for i in range(80):
                liat4.append(0)
            random.shuffle(liat4)
            l.append(liat4)
        pop=l
        #pop = np.random.randint(2,size=(POP_SIZE,DNA_SIZE))
        #print('pop初始化',pop[2][4]+3)
        for i in range(N_GENERATIONS):
            x=translateDNA(pop)
        pop=np.array(crossover_and_mutation(pop,CROSSOVER_RATE))#构建矩阵
        fitness=get_fitness(pop)#列表
        pop=select(pop,fitness)
        print_info(pop)
    main()
```