****

硕士学位论文

|  |
| --- |
| 基于候鸟迁移算法的全自动免疫 |
| 检验设备的分批调度及协同优化 |

**Lot Splitting Schedule and Cooperative Optimization of the Immunoassay Equipment Based on Migrating Bird Optimization Algorithm**

**基于候鸟迁移算法的全自动免疫检验设备的分批调度及协同优化**

|  |  |
| --- | --- |
| 作者姓名 |  |
| 学科专业 | 控制理论与控制工程 |
| 指导教师 |  |
| 所在学院 | 自动化科学与工程学院 |
| 论文提交日期 | 2019年4月 |

一、绪论

# 1.1 课题背景和意义

免疫检验，又称为临床免疫学检验（Clinical Laboratory Immunology），是利用免疫学理论来诊断疾病的一种手段，是一种常见且重要的医学检验，目前在国内外医院和检验检疫机构都得到广泛的应用。免疫检验利用抗原抗体之间的结合反应来实现临床上蛋白物质的定性、定量检测。它可以用于检验多种疾病，包括超敏反应性疾病、自身免疫性疾病、免疫增殖病、免疫缺陷病、肿瘤等等，免疫检验对人类健康发展具有重要意义。其中，乙肝（HBsAg、HBsAb、HBcAb等）、丙型肝炎抗体（ant-HCV）、性激素（LH、FSH、Progest等）、胰岛素、肿瘤标记物（CEA、SCC、FPSA等）、梅毒血清（RPR、TPPA等）都是很常见的免疫检验项目，在临床诊断中、术前检查、日常体检等场合应用非常广泛。

随着国内医疗条件的不断提高，患者对于免疫检验的需求日益增加，免疫检验市场规模逐年上升。根据《2018年中国免疫诊断行业分析报告》的调查，自2014年以来，我国免疫检验市场规模一直以高于15%的年增速水平平稳上升，2014年市场规模不足80亿元，只用了四年就增长到140亿元以上。根据该报告的预测，2019年和2020年我国的免疫检验市场依然会以超过15%的年增速水平上升，保持稳定增长。因此有必要加快该领域的研发，以满足国内免疫检验的日益增长的需求。

免疫检验的灵敏度非常高，能对微量的物质进行精准的检测，然而对检验操作和仪器设备有比较高的要求，而且检验步骤非常复杂、检验过程及其费时费力。随着国内医疗条件的不断提高，患者对于免疫检验的需求日益增加，患者对检验的时效性要求也越来越高，因此有必要提高临床中免疫检验的效率。免疫检验由初期的人工检验发展为如今的设备自动检验，越来越精密、规模越来越大、效率越来越高的免疫检验设备在医疗器械市场中面世并得到应用。

全自动免疫检验设备是把不同功能的仪器集成在一台检验设备中，所有仪器在系统调度下互相配合，使所有检验步骤都可在设备内自动完成，实现自动化进样、加试剂、移板、温育震荡、洗板、检测。它可代替人工检验的全过程，而且还能避免主观因素的干扰，实现检验流程的高度标准化、自动化。对于患者来说，它可以在保证检验可靠度的同时，减少患者的等待时间。对于医生来说，它有助于实现快速诊断。对于医院检验科来说，它能在保证检验精度的同时，大量节省人力，减少检验人员的劳动强度，提高免疫检验的效率，增加检验科的样本容纳量，提高医院的竞争力。

然而当前市面上绝大多数全自动免疫检验设备都是进口设备，进口设备在可靠度、灵敏度、自动化程度方面都领先于国内的设备，许多医院和机构的免疫检验设备仍然依赖进口。根据《2018年中国免疫诊断行业分析报告》的调查，当前免疫检验设备的市场依然是以国外产品为主，罗氏公司占有率高达35.5%，雅培、丹纳赫、西门子三家公司总共占据了46%以上的市场。在国外公司研发水平遥遥领先、市场口碑依然很有保证的环境下，加快国内全自动免疫检验设备的研究越来越迫切。

国内关于全自动免疫检验设备的研究非常少，特别是对该设备在批量检验过程中的分批调度问题研究几乎处于空白。由于医院检验科每天接收的样本量日益增加，检验样本呈现出批量性的特点，这对于检验调度是一个巨大的挑战。不同种类的检验样本大批大批地进入检验科，给检验科造成巨大的压力。批量检验导致了更长的等待时间、更低的检验效率。如何解决批量检验背景下的全自动免疫检验设备分批调度，是一个亟待解决的问题，对于临床免疫检验来说，具有很重要的现实意义和应用价值。

为此，本文针对批量检验应用背景下的全自动免疫检验设备分批调度问题进行了深入的研究，针对该复杂问题的特点设计了多种寻优的算子，根据全自动免疫检验设备的特点综合考虑分批方案优化和调度顺序优化，在最大程度上提高批量检验的效率。这对于进一步提高免疫检验的效率和质量、提高医院检验科服务水平、提高患者就医体验都有很重要的意义。

# 1.2 免疫检验设备的国内外研究现状

免疫检验的关键步骤是标记免疫复合物定量检测。由于免疫复合物质非常难以识别和定位，自从免疫检验提出以来，发展出了不同的标记免疫技术。1941年Coons提出了荧光免疫分析（Fluorescent Immunoassay），把不影响活性的荧光物质加入抗体或抗原，使反应后的免疫复合物也带有荧光物质，最后测定荧光强度即可计算检验物质的浓度[1]。该方法安全无毒，敏感性高，但是存在非特异性染色体问题，而且荧光抗体样本并不能长期保存。1956年Yalow和Berson提出了放射免疫分析（Radio Immunoassay），用放射性核素标记抗原或抗体，能实现超微量的物质检测，但是反应过程复杂，需要离心分离之后才能进行放射性测量，放射性标记物的有效时间非常短，要求极其精确的操作，而且该方法的原料有放射性，还需要额外处理核废料[2]。1966年Nakene和Pierce发现了一些酶能使底物显色，达到与荧光标记相似的效果，很快酶免疫分析（Enzyme-labeled Immunoassay）得到了广泛的应用，其中目前应用最多的是酶联免疫法（Enzyme-Linked Immuno Sorbent Assay，ELISA）。由于是在固态载体而非血样液体内反应，省去了离心分离的操作，而且能长期保存，检测速度快，适用于大批量样本检测，得到了国内外临床的认可和广泛应用，是目前应用最广泛的免疫检验方法，也是本文免疫检验设备中采用的检验方法[3]。

酶免疫检验设备的发展直至现在，分为了三个发展阶段：

（1）单功能实验仪器研发，替代了部分人工检验步骤。酶联检测是酶免疫检验过程的核心技术，关系到检测效果的精度，因此研发了以光电反应为原理的酶联分析仪来替代低精度的人工检测比色，极大提高了分析的精度和可靠度。此外，还研发了自动温育震荡器来加速免疫反应过程，研发了自动洗板机来替代人工洗涤操作，研发了试剂盒来替代检验前期包被的准备工作。该阶段专注于提高各模块仪器的精密度和易用度，替代部分检验人力。





(a) 爱康洗板机

(b) 赛默飞iEMS HT微孔板孵育振荡器

(c) 帝肯Infinite F50酶标仪

(d) R&D公司Quantikine系列比色度夹心法ELISA试剂盒

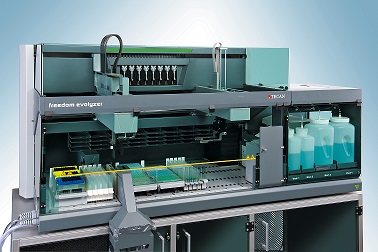
（2）半自动检验设备，把两种或多种仪器集成在一台设备上。随着医疗器械的不断发展，把温育震荡、洗涤、酶联检测等功能部分集成在一台设备中，并配备化加样头、机械臂等装置，成为了免疫检验设备发展的趋势。然而由于单台仪器功能有限，部分检验过程还需要人工操作，例如检验板在各个模块仪器之间需要人工转移。此外，未集成在设备上的部分仪器还需要自行配备、自行人工移板。



(a) 默塞飞Multiskan热电FC温育酶标仪

(b) 倍爱康MAGLIA 60 化学发光免疫分析仪

（3）全自动检验一体化设备，所有仪器和装置在系统控制下配合工作，实现检验过程全自动化，完全代替人工操作。除了自动温育震荡器、自动洗板机、自动酶联检测仪之外，设备还配备灵活的机械臂，方便样本在各个仪器之前的转移，还配备了自动加样头，免除了人工加样的操作，在最大程度上替代人工检验操作。除了设备维护、设备初始化以外，基本不需要人工介入，极大地提高了免疫检验的效率，是当前免疫检验设备的研究趋势[4]。



(a) 帝肯Freedom EVOlyzer 全自动酶免工作站

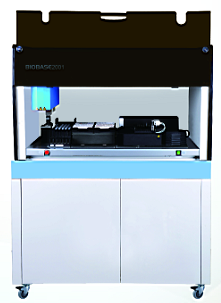
(b) 欧蒙EUROIMMUN Analyzer I-2P 全自动酶免分析仪

全自动免疫检验设备的研发需要多种学科的交叉，需要多种领域的技术，包括生化检验技术、自动化装备技术、光电转换技术、自动控制技术等等，而且需要及其精密的制造和控制才能满足免疫检验过程的标准[5]。因此它的研发和制造难度比较大。

目前国内市场上的全自动免疫检验设备大多数都是进口产品，主要品牌包括雅培、贝克曼、帝肯、欧蒙等等。贝克曼库尔特的UniCel DxI 800免疫分析系统内置30个样本位和50个试剂位，可以进行多种免疫检验项目，达到了的检测灵敏度，可以实现不停机的试剂补充。雅培的ARCHITECT *ci*16200全自动生化免疫分析系统内置365个样本位和155个试剂位，整合了多个功能相同的模块，每个小时能完成200个免疫测试，能为大型医院提供高效的免疫检测，但是其试剂和耗材非常昂贵。

国内的全自动免疫检验技术还不成熟，相关产品还比较少，主要品牌有爱康、博科等等。其中博科的BIOBASE2001全自动酶免工作站内置2个加样通道、6个温育震荡位，属于轻量型的检验设备，采用模块式结构设计而非集成一体的结构设计，可匹配不同厂家的试剂盒。爱康的URANUS AE全自动酶免仪具有高精度、高速度的加样模块，开发了灵活易操作的中文软件，对国内检验工作者比较友好。

随着国内医疗水平的不断提高，患者对免疫检验的需求不断增加，加快国内基于酶联免疫的全自动免疫检验设备的自主研发显得尤为迫切。 分利用设备的对于提高以在不停机的状态西一组内之前覆盖好反应板。

(a) 爱康URANUS AE全自动酶免仪

(b)产-博科BIOBASE2001全自动酶免工作站

# 1.3 免疫检验设备调度问题的国内外研究现状

对于全自动免疫检验设备来说，除了对仪器装置有极高的精密性要求以外，还对检测过程调度有较高的时效性要求。在实际临床检验中，免疫检验设备面对的往往是大量不同种类的检验样本，如何安排设备内各模块装置的配合工作，如何安排样本的检验顺序，才能保证每个样本检验步骤的完整，同时减少样本的等待时间，增加样本检测的并行水平，这就需要对设备的作业进行合理的调度。调度问题是提高设备检验效率的关键问题。

国外大多关于免疫检验设备调度问题的详细资料还未公开，国内关于这方面的研究也非常欠缺。大部分研究针对的是生化检测设备，对基于酶联免疫的免疫检验设备的调度研究更加少。该领域研究的发展经历了多个阶段：

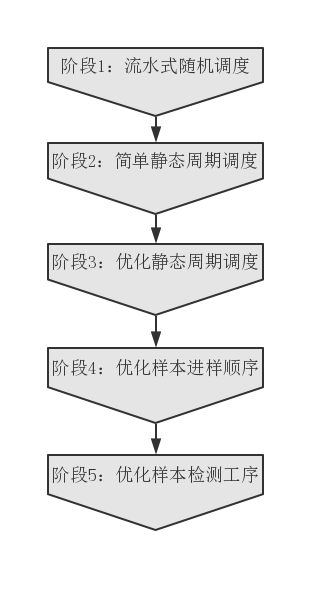
（1）流水式随机调度。市面上多数免疫检验设备都是使用简单的流水式随机进样。贝克曼库尔特公司的UniCel DxI 800免疫分析系统和ACCESS 2全自动免疫分析系统都采用了“随机组合，随机进样”的策略，没有对调度进行优化。帝肯公司的Freedom EVO  ELISA全自动酶免分析系统只能实现普通的连续进样，也没有对调度进行优化。这一类免疫检验设备只能节省劳动力，而并不能有效提高检验效率。

（2）简单静态周期调度。少部分的设备使用了简单的静态周期调度。该方法用相同的周期来调度不同种类的检验样本，让检测样本按照固定的时间间隔进入设备开始检测。该方法忽视了不同样本所需时长的差异性，从而使一部分本身检测时长较短的样本在设备内等待较多时间。而且只有前面的样本完成该步骤并转移代下一个步骤之后，下一个样本才能进行该步骤。检验过程中会消耗较长时间，有些步骤则较快能完成，因此样本在等待前面的样本完成时长较长的步骤时，也需要较长的等待时间。因此静态周期调度虽然简单、容易实现，而且能保证调度的安全原则，但是大量的等待时间造成调度顺序低下。

（3）优化静态周期调度。对于周期调度的优化，一部分研究者做出了不同的改进尝试。刘志辉等人根据临床检验上的经验设计检验计划工作表，对不同种类样本的进样顺序和进样间隔时间进行了多次实验评估，从而确定较优的调度方案[6]。谷成祥也设计了计划工作表，而且针对了不同规模的作业都设计了相应的计划表[7]。然而依靠临床经验进行调度难度大，优化效果也不能保证。杨勇毅等人对进样时间间隔进行了优化，以适应不同检验项目的特点，而不是使用固定的间隔，同时还优化了不同检验项目进样的顺序，对4个样本的所有24种排列顺序进行实验，从而得到最优的进样顺序[8]。但其流水周期式的调度依然比较低效然，这种穷举优化的方法只能用于较小规模的调度。

（4）优化样本进样顺序。少数研究者跳出周期调度的范畴，尝试对所有检测样本的检测顺序进行优化。张晶等人把多样本检验调度问题视为非对称形态推销员问题，从而对检测顺序进行调度。把每个检验样本当做一个城市，遍历所有城市的最短路径即为优化的检测顺序[9]。然而该方法只局限于对检测进样顺序进行优化。

（5）优化样本检测工序。上述方法都属于比较粗糙的调度优化，并没有对检验步骤的顺序进行细致的优化。由于设备中包含多个并行设备，可同时对多个样本进行检验，所有样本的每个检测工序都需要调用不同的并行设备，因此存在多个样本同时需要同一个设备的情况，如果不进行合理的调度，会出现阻塞、等待时间长的后果。因此，进一步优化免疫检验设备的调度，在检验工序层面上进行细致的优化调度，把设备的工作效率最大化，是未来研究的一个重点方向。



如今，大多数免疫检验设备的调度还处于阶段3、阶段4，而学术研究对于免疫检验设备调度的研究已经发展到阶段5了。对免疫检验设备的调度研究朝着更精细化、更自动化、更规模化的方向发展。然而，在医院检验科检验呈批量性的背景下，针对批量检验工序优化的研究还是一片空白，批量检验样本的处理以及调度还未有相关研究。

# 1.4 论文主要研究内容及其章节安排

还未写

二、全自动免疫检验设备分批调度分析及建模

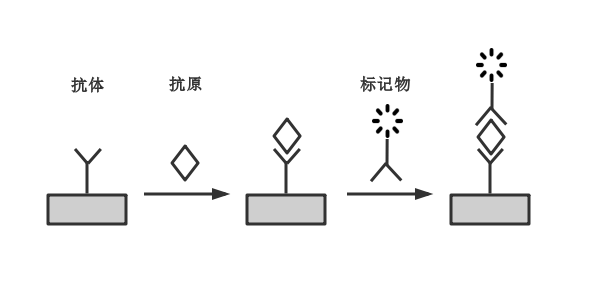
# 2.1 引言

本章主要是明确待求解的问题模型。首先介绍全自动免疫检验设备的原理和工作流程，阐述其分批调度在实际应用中对提高检验效率的重要影响，分析其特点和难点。然后把全自动免疫检验设备的分批调度问题提炼为一种柔性作业车间的分批调度问题，并用数学语言准确描述该问题的目标，明确所有约束条件。

# 2.2 全自动免疫检测设备原理

## 2.2.1 免疫检验原理及流程

免疫检验利用免疫反应，让特定的抗原与抗体会结合产生反应，形成相应的免疫复合物。通过标记棉衣复合物，使其被仪器识别，就能检测复合物的浓度，从而计算待测物的浓度，该原理如图所示。复合物的含量与抗体或抗原物的含量有关，因此能通过某种特定的抗原来检测血样中抗体的含量，或者通过某种特定的抗体来检测血样中抗原的含量。由于抗原和抗体的结合具有特异性，因此只要使用不同种类的抗原或抗体与血样混合，就可以对血样中的某些微量物质进行定性和定量的检验，可以达到非常高的检验精度，但是对反应步骤和反应环境都有比较高的要求。



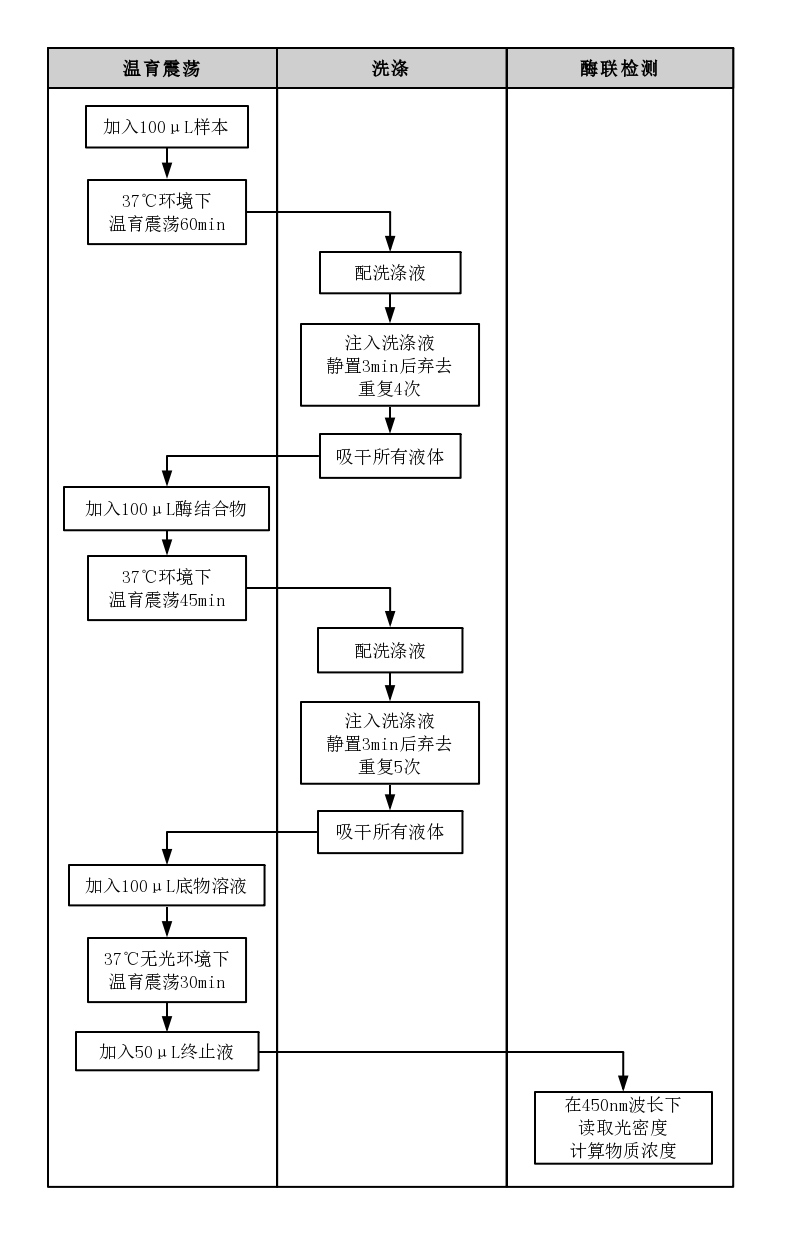
酶联免疫法是目前临床最常用、最可靠的一种免疫检验方法。首先把特异性抗体或抗原作为检验的试剂，固定在某种固态载体表面，这个过称为包被。在临床中，一般使用已包被好的成品试剂盒，省去人工包被的步骤。然后加入待测血样，与试剂在固态载体表面上反应，生成免疫复合物，反应过程可以通过温育震荡加速完成。将反应过后的残留物质洗涤干净之后，就可以加入酶标记的特异性抗体或抗原试剂，继续反应，该过程也可以使用温育震荡加速完成。把残留物质洗涤干净之后，加入底物显色剂，经过一定时间的反应后显色完成，此步骤也可使用轻微的温育震荡加速完成。最后把显色的免疫复合物放入酶标仪，通过比色法对反应结果进行测定和分析。以下是三个主要步骤的介绍：

（1）温育震荡：通过在一定温度下连续震动抗原和抗体，促进二者之间的反应，通常在4℃、37℃或43℃的恒温环境下，震荡30分钟到两个小时就可以产生稳定的免疫复合物。通常使用水浴提供恒温环境，而且为了避免反应物在震动中流出反应板，需要在温育震荡之前覆盖好反应板；

（2）洗涤：洗涤未结合的抗原或抗体等残留物质。反应板中的残留物质会导致包被抗体与二抗间产生交叉反应，还会给酶标仪的检测带来很大的干扰，导致灵敏度降低、或者假阳性及负读数。因此需要使用特定的洗液把残留物质清洗干净，保证极低的残留量，最后还要把反应板上的洗涤剂拍干或吸干。人工洗板存在清洗不够彻底、交叉污染的问题；

（3）酶联检测：测量酶标记检测免疫复合物的含量。由于底物显色剂的作用，反应生成的免疫复合物呈现出一定深度的颜色，颜色越深，代表待检物质的浓度越高。测量通过复合物的光信号，把光信号转化为电信号，由此计算被复合物吸收的光电子，从而得到复合物含量。

基于酶联免疫法的免疫检验完整过程主要由上述三个过程组成，以人类IgG免疫检验为例，整个过程包含三次不同的温育震荡，每次加入不同的试剂，进行不同时长的温育震荡，某些种类的免疫检验还需要不同温度的反应环境，最后一次温育震荡后还需要加入终止液来终止反应。由于每次温育震荡前都需要加试剂，因此把加试剂的步骤包含在温育震荡步骤内。其中还包含两次相同的洗涤，洗涤的步骤包括，按需配比洗涤液，重复静置和弃液，吸干所有液体。经过三次温育震荡和两次洗涤之后，就能进行酶联检测了。具体的流程图如下所示：



## 2.2.2 全自动免疫检验设备原理

随着医疗系统的不断完善和医疗保障的不断普及，患者对免疫检验的需求量越来越大，同时对于检验时效性的要求也越来越高。有些医院的检验科每天接收到的免疫检验项目数量大，种类多，呈多种类批量性。而免疫检验流程复杂，对检验环境和操作要求高，这给检验科造成了很大的工作压力。最初的免疫检测都由检测人员手工完成，在操作上十分容易出现误差，由于检验项目种类多且检验流程复杂，人工检测效率比较低下。为了提高免疫检验的效率，一些半自动的免疫检验设备被研制出来，代替一部分人工操作。但是加样操作，以及检测样本在这三种仪器之间的移板操作还需要人工完成。半自动的免疫检验设备不具备自动调度系统，每个样本在哪个时刻进行哪个操作，哪个样本先检验哪个样本后检验，每个样本使用哪一台仪器进行检验，样本要从哪台仪器转移到哪台机器，这些复杂的调度还需要人工依靠经验来安排。在面多大量不同类的复杂检验项目时，人工调度效果十分有限，导致检验效率也不尽如人意。

随着技术的不断发展，全自动免疫分析检测设备被研制使用，进一步提高生化检验的效率，同时提高了检验的准确率。全自动免疫分析检测设备针对免疫检验过程的特点，集成了免疫检验所需的所有仪器，包括温育震荡器、洗板机、分析仪，保证免疫检验所需的所有步骤都能在设备上进行。同时还配备了高精度定位加样头，高效的样本运输系统，和条形码识别器，保证检验过程可以在设备上全自动运行，除了设备维护和进样，不需要额外的人力。设备还具有主控制器，可以生成调度方案，使所有模块和装置都在控制器的统一调度之下准确运行。每个样本的检测状态和检测结果都被自动记录到控制器中，最后快速生成生化检验结果单，将数据传入医院检验科的信息系统，实现检验结果电子化。大量样本的检验结果还可以长期保存在检验科的信息资料库中，方便医生对复诊患者的病史进行快速而准确的查询，还能为以后的医学研究提供数据服务。

全自动免疫分析检测设备可以进行多种免疫检验项目，它通过条形码识别检验项目的种类，然后按照各种类的检验过程标准执行相应的检验步骤。检验样本只要送入全自动免疫分析检测设备，就可以严格按照其检验流程进行全自动检验，加入特定种类的试剂，进行特定次数、特定时长、特定温度的温育震荡，使用试剂进行彻底的洗板，最后由分析仪得到检测结果。在设备的精准调度下，每个检验项目都能得到检验环境、检验流程上的保证，能实现高精度的标准检验过程。姚勇等人通过实验验证，全自动酶免疫分析系统可以有效避免人工检验的误差，使检测的准确性和重复性大大提高[10]。使用全自动免疫分析检测设备可以在保证检验准确度的同时，提高检验科的检验效率，避免血样的浪费，减少患者的等待时间，为患者提供可靠的检验，实现一流的检验水平和高效的就医环境。

设备中最主要的模块有三个，温育震荡模块、洗板模块、酶联分析模块，每个模块分别配备若干个功能相同的温育震荡器、洗板机、酶联分析仪。这样可以实现多个项目的并行检验，提高检验效率，同时也能避免由于单个机器故障导致的设备瘫痪，提高设备的工作稳定性。这三种仪器的结构和功能如下：

（1）温育震荡器，主要由水箱、温度传感器、半导体制冷片、直流无刷电机构成。负责检测过程中温育震荡的步骤，温度传感器和半导体制冷片负责把水箱环境维持在特定的温度，防止温度变化而影响检验精度。直流无刷电机使反应板中的样本与检验试剂保持一定频率的震动，使二者充分混合反应；

（2）自动洗板机，主要由清洗头、震板电机、加液器、洗液瓶、废液瓶、反应板传感器、气溶胶密封盖组成。负责检测过程中洗涤的步骤，清洗头喷出不同种类的洗涤液，配合适量的震板或浸泡，清洗残留液，并通过吸液模式保证极低的残留量，提高检测精度。同时，气溶胶密封盖可防止传染性气溶胶扩散，洗液瓶和废液瓶中的液位传感器可保障仪器的安全。为了保证洗涤彻底，加入洗涤液之后需要静置一定时间，并重复清洗多次，最后再使用残留液体模式吸干所有液体；

（3）多功能酶标仪，全称为酶联检测仪，相当于变相光电比色计或[分光光度计](https://baike.baidu.com/item/%E5%88%86%E5%85%89%E5%85%89%E5%BA%A6%E8%AE%A1)，主要由光源、光电检测器组成，负责在酶联检测。通过发射某种特定波长的光波，检测免疫复合物的吸光值，从而推算出免疫物的含量，分析样本的检验结果。

## 2.2.3 全自动免疫检验设备工作流程

当医院检验科接收医生开出的检验单，就会给该检验项目分配特定的条形码，用于标记该检验项目，并让护士及时采集血样，在血样管贴上该检验项目的条形码。然后使用全自动免疫检验设备的条形码识别器扫描的血样管，识别检验项目的种类等信息，经过扫描的血样管可以放在设备的待检验区域，等待进样检验。

当被扫描的血样达到一定批量之后，全自动免疫检验设备开始生成调度方案。首先设备统计各个种类项目的数量，然后根据检验样本数量和种类的实际情况，以及根据设备的当前状态，实时生成调度方案。该过程主要使用优化算法来优化分批调度方案，确定子批的划分，和每一个检验工序在仪器中的顺序。然后每一个样本就会严格按照调度方案进入设备，让各台仪器互相配合进行设定好的检验步骤。

当样本检验完成之后，设备会自动分析检验结果，并通过网络把结果传输到检验科的系统上，检验科的医生对检验结果进行审核，并直接将审核后的电子检验结果反馈给医生和患者，而不用等待纸质报告的打印，方便医生快速诊断和处理病情。检验结果还能被长期保存在检验科的数据库中，方面医生快速了解患者病史，同时也可以为医学研究提供充足的临床数据。

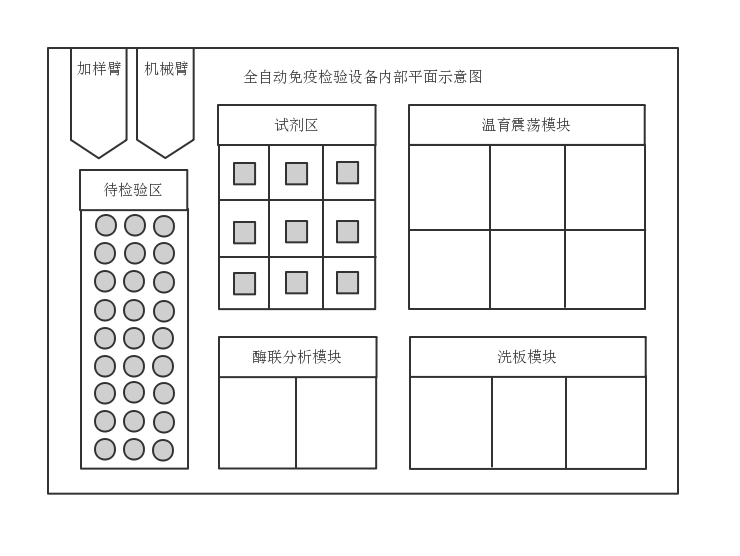


通过以上分析发现，全自动免疫检验设备简化了医院检验科免疫检验的流程，节省了不必要的手续。而调度方案的质量关系到免疫检验的效率，关系到患者和医生需要等待多久才能拿到检验报告。对于批量样本背景下的免疫检验设备分批调度问题来说，合理的分批以及优化的调度顺序是解决该问题的关键。

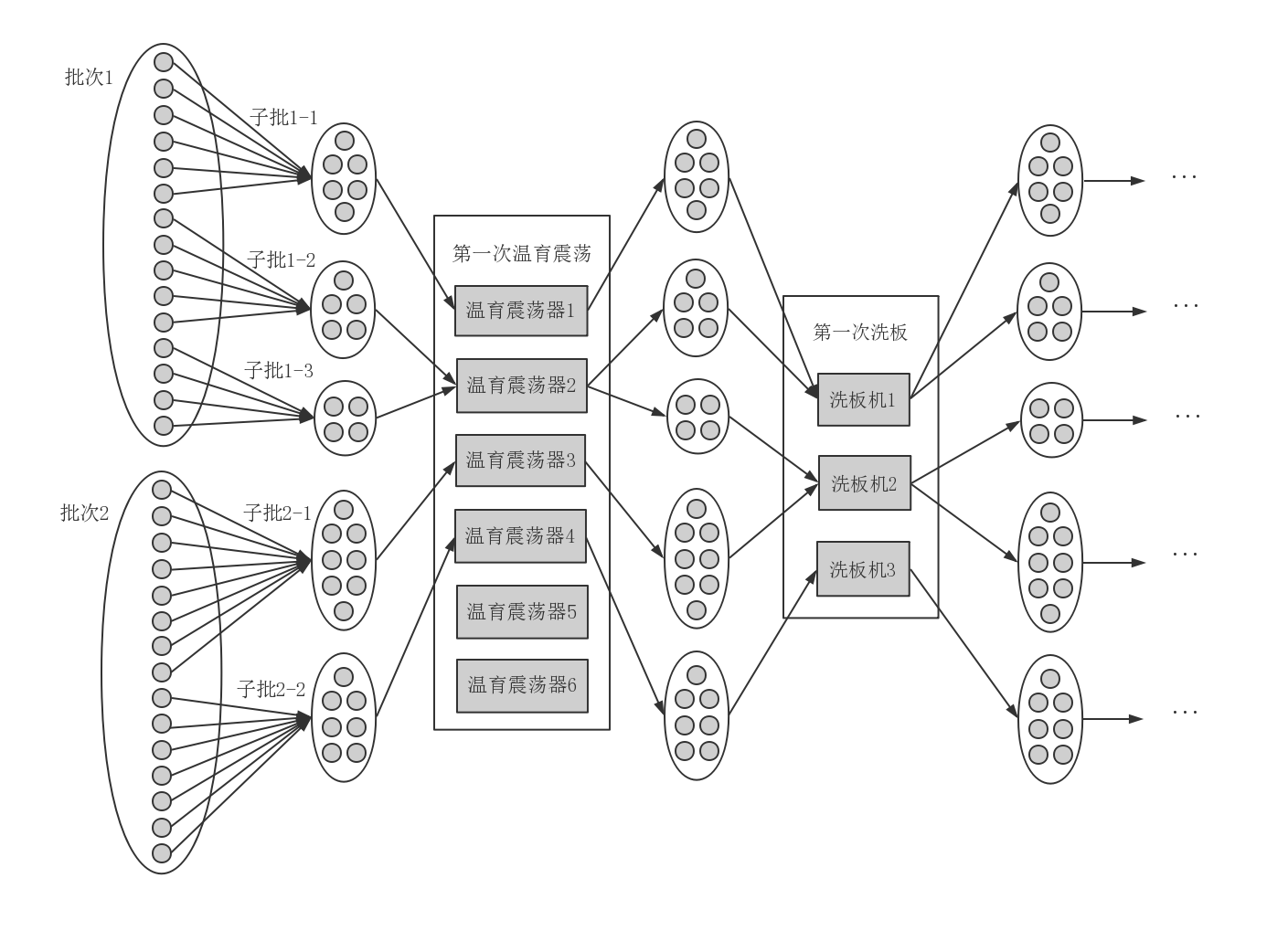
# 2.3 全自动免疫检验设备分批调度问题介绍及分析

## 2.3.1 问题描述

根据全自动免疫检验设备的工作流程，调度方案的生成与检验效率的高低关系密切，解决调度问题是提高全自动免疫检验设备检验效率的关键。根据2.1.2的介绍，全自动免疫检验设备的主要检验模块有三个，包括温育震荡模块、洗板模块、酶联分析模块，每一个模块内都有若干个功能相同的机器。样本进入设备后，根据检验步骤在不同的模块内进行检验工序。以一台小型全自动免疫检验设备为例，其内部结构如图所示。



由于患者对免疫检验的需求日渐增加，有些医院的检验科每天接收能接收到大量不同种类的免疫检验样本，呈现出多种类、批量性的特点。把批次分为若干个较小的子批可以提高检验的灵活性，减少等待时间，如何合理分批需要深入考虑。如何安排每个子批的检验顺序，并确定每个检验工序的起始和结束时间，也需要深入考虑。需要综合考虑全自动免疫检验设备的分批和调度，才能最大程度上提高设备的检验效率。如图所示，是免疫检验中分批调度的示意图，所有批次首先被合理地分为多个子批，以子批为单位进入设备，按照一定顺序，按照在某个时刻分别进入相应仪器，完成各个检验工序。



一个分批调度方案需要确定以下所有信息：

（1）每个批次分为多少个子批

（2）每个子批分别包含多少个检验样本

（3）每个子批的每个工序使用哪台仪器

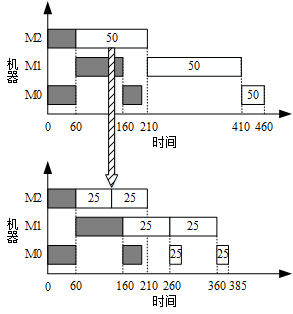
（4）使用每台机器的所有子批的检验顺序

（5）每个子批每个检验工序的起始时间和结束时间

调度是对资源进行合理分配，包括机器资源、加工材料资源、库存资源等等，以高效地完成生产加工任务，满足对市场供应的快速性[11]。良好的生产调度对于提高自动化生产效率来说不容忽视。在对大量不同种类的样本进行检验时，人类依靠经验的调度并不能较好地解决调度问题，需要使用计算机通过高效的调度算法得到调度方案，充分考虑不同检验项目的需求，以及所有资源的特点，合理安排检验的顺序和资源的分配。全自动免疫检验设备能自动生成调度方案，确定每个样本检测的先后顺序，确定每个样本在什么时刻进入哪个模块。由于不同检验项目的检验步骤是固定的，每个步骤的时长也是固定的，因此只要确定了调度方案，就能计算出样本检验结束的时间。设备工作的时候，让所有模块和装置按照调度方案工作，让检测样本严格按照调度顺序进出各个模块。

调度问题在车间加工领域内非常常见，它能直接影响生产加工的时间和成本，传统的调度问题有单机调度问题（Single Machine Shop Scheduling Problem，SMSP），并行机加工问题（Parallel Machine Scheduling Problem，PMSP），置换流水车间调度问题（Permutation Flow Shop Scheduling Problem，PFSP），流水车间调度问题（Flow Shop Scheduling Problem，FSSP）。更复杂的调度问题有开放车间调度问题（Open Shop Scheduling Problem，OSP），作业车间调度问题（Job Shop Problem，JSSP），带装配作业车间调度问题（Assembly job shop problem，AJSP），柔性作业车间调度问题（Flexible Job Shop Problem，FJSP），它们具有更多更复杂的约束。

分批调度是Reiter提出的一种提高车间生产效率的技术，通过把批量工件分为多个子批，不同子批的不同工序可以重叠加工，从而缩短加工时间[12]。Low通过分析和实验证明，把批量的工件划分为子批进行加工能有效缩短完工时间[13]。在一些实际的生产环境中，工件加工往往会呈现批量性。经典的调度一般是把一整批同类的工件视为一个整体，以整个批次为单位放入生产系统，整个批次里所有工件的上一个工序加工完成之后才能进行批次的下一个工序，导致较长的等待时间。而且考虑到实际生产系统中往往会有功能相似、可以并行加工的机器，如果把呈批量性的样本的批次划分为子批，以子批为单位进入生产系统。一个子批不需要等待其他子批完成当下工序的加工就能进行下一个工序，因此同一种工件的不同工序可以同时在不同子批上进行，既能充分发挥并行机器的生产能力，又能减少工件的等待时间。如下图所示，对于2批次的待检验样本，灰色代表批次1，白色代表批次2，若不进行子批划分，批次2的工序2需要等批次50个样本的工序1完成后才能开始，机器的等待时间比较长。把批次2划分为两个子批之后，批次2的工序2可以只需要等待25个样本完成工序1之后就能开始，减少了等待时间，对于批次2的工序3来说也一样减少了等待时间，让部分子批的工序提前开始，使效率显著提升。



分批前后的甘特图

分批调度最早被广泛地应用于FSSP问题中，因为FSSP问题比较简单，大多数对分批调度的研究都是基于FSSP问题的[14][15][16]。后来分批调度也逐渐地被应用于更复杂的生产环境中，例如PMSP问题[17][18]，JSSP问题[19][20]，和AJSP问题。FJSP的分批调度问题是更复杂的问题，具有更复杂的约束，和更复杂的解，它也更贴近实际的生产环境。已经有少部分学者开始研究FJSP的分批调度问题了，但是相关研究还是相对比较少[21][22]。

## 2.3.2 问题难点分析

合理的子批划分对提高生产效率有非常重要的影响。分批子问题是个很复杂的问题，子批量并不是越大越好或者越小越好。当子批量比较大的时候，会导致后面的子批需要等待更长的时间才能开始各个工序的加工。当子批量比较小的时候，会导致较多的子批数，需要更多次数的准备操作，消耗更多的加工时间。子批数也并不是越多越好或者越少越好。当子批数较多的时候，调度问题的复杂度随之提高，寻找最优调度的难度会更高，除此之外还需要消耗更多的准备操作时间[23]。当子批数较少的时候，就难以充分发挥重叠加工的优势。子批划分应该适应不同加工环境的需求，如何合理安排子批数和各子批批量，从而最大程度发挥分批的优势，这是分批调度问题的第一个难点。

调度问题本身是一种复杂的组合优化问题，研究初期通常使用经典数学优化方法来解决该问题，包括混合整数规划法[24]、动态规划法[25]、分支定界法[26]等等，后来启发式方法也[27]被逐渐应用到该问题上，但是都只能解决规模较小的调度问题。Lenstra通过研究发现，这些调度问题本质上都是NP难（Non-deterministic Polynomial Hard）的问题，这类问题的大型实例不可能通过有限步的计算求得精确解，而只能找到近似解[28]。随着优化算法的不断发展，智能进化算法由于在近似求解的高效性，越来越多被适用于调度问题，包括禁忌搜索（Tabu Search，TS）[29]、模拟退火（Simulated Annealing，SA）[30]、遗传算法（Genetic Algorithm，GA）[31]、粒子群算法（Particle Swarm Optimization，PSO）[32]等等。而分批调度问题也属于NP难的问题，比单纯的调度问题具有更加复杂的性质。如何设计算法，技能保证分批的复杂约束，又能高效地求得更优的分批调度解，这是第二个难点。

对于分批调度问题，除了需要优化调度顺序以外，还需要优化子批划分。调度顺序与子批划分存在互相影响的耦合关系。最优调度顺序与最优子批划分是相对而言的。对于某个调度顺序来说，其最优子批划分只是针对它自身而言，对于其他调度顺序未必是最优。反之，对于某个子批划分来说，其最优的调度顺序也只是针对其自身而言，对于其他子批划分而言也未必是最优。当调度顺序改变了，其相应的子批划分问题值域也改变了。当子批划分改变了，其相应的调度顺序搜索空间大小和结构也会发生改变。子批划分与调度顺序的耦合增大了分批调度问题的难度因此，如何协调这两个问题的求解，是分批调度问题的第三个难点。



# 2.4 免疫检测设备的柔性作业车间分批调度模型

根据前文对免疫分析检测设备的分析，把其分批优化调度问题归为一类准备操作可分离的柔性作业车间分批调度问题（Flexible Job Shop Scheduling Problem with Lot Splitting，LS-FJSP），它属于柔性作业车间调度问题的一种拓展，是一个比较复杂的问题，同时具有作业车间、柔性、分批的诸多属性和约束。

柔性作业车间分批调度的定义如下：有个批次的工件待加工，一个批次内有若干个类型相同的工件，要使用台机器完成所有的加工，每种工件都有不同的加工工序。每个工序都有不同的可选加工机器，不同机器的加工时间不一样。由于批次的工件量比较大，把每个批次都划分为多个数量不等的子批，以子批为单位按照工序顺序进入不同机器进行加工。分批调度问题的任务是，找到最优的子批划分方案，即每个批次分成多少个子批，每个子批包含多少个工件，还要找到最优的调度方案，即每个子批在机器上的加工顺序，从而使所有批次的完工时间最小化。

该问题还有以下特点：

（1）以子批为加工单位。一个子批里所有工件的上一个工序都完成后，才能进入下一台机器，以批为单位开始下一个工序。一个子批在某个工序的加工时间等于该子批所有工件的该工序加工时间之和；

（2）子批量约束。一个批次里所有子批的子批量之和等于该批次的批量。

（3）考虑准备操作。机器在加工一类工件之前还需要进行准备操作。机器连续加工多个同类工件之前只需要进行一次准备操作，如果机器连续加工同一个批次内的不同子批，那么只需要在同批的第一个子批加工前进行一次准备操作；

（4）准备操作时间与工件类型有关。不同种类的工件有不同的准备操作时间；

（5）准备操作可与工件分离。在工件到达机器之前，只要机器空闲了，就可以提前进行准备操作，不需要等工件到达之后才进行准备操作；

（6）准备操作时间与工序加工时间分开考虑。工序加工时间不包含准备操作时间；

（7）一致分批。子批的划分在所有工序加工过程中是不变的；

（8）不允许抢占。一个子批在某台机器未完成加工的时候，其他子批不允许抢占用该机器加工；

（9）同一时刻每台机器最多只能加工一个工件；

（10）同一时刻每个子批最多只能进入一台机器进行加工；

（11）保证工序约束。每类工件的工序顺序都是固定的，每个子批完成上一个工序的加工之后，才能开始下一个工序；

（12）所有批次的工件都在零时刻释放，即所有工件的第一个工序在零时刻都进入待加工状态；

（13）机器容量限制。每台机器可容纳的子批量是有上限的，因此在划分子批的时候，每个子批的子批量不可以超过机器的容量限制；

（14）假设每个批次内所有工件都是同种类的工件。

对于全自动免疫检验设备来说，把不同检验种类的批样本视为LS-FJSP中不同的批次，把所有温育震荡器、洗板机、酶联检测仪视为LS-FJSP中的加工机器，把检验步骤的每个步骤视为LS-FJSP中的工序。接下来把免疫检测设备的柔性作业车间分批调度问题用数学模型表达出来，以下是该问题的一些变量：

——批次总数

——表示批次的序号，

——第个批次

——批次的工件总数

——批次的子批总数

——表示子批的序号，

——的第个子批

——的工件总数，即的子批量

——中一个工件的工序总数

——表示工序的序号，

——的第个工序

——机器总数

——表示机器的序号，

——第台机器

——一个工件的工序的可选机器的序号集合

——中一个工件在上的加工时间

——中的工件在上的准备操作时间

——的加工开始时刻

——的加工完成时刻

——的准备操作开始时刻

——的准备操作完成时刻

——的加工机器上，前一个子批工序的完成时间

以下是该问题的决策变量：

 (2-1)

 (2-2)

 (2-3)

数学模型如下：

 (2-4)

s.t.

 (2-5)

 (2-6)

 (2-7)

 (2-8)

 (2-9)

 (2-10)

 (2-11)

 (2-12)

 (2-13)

 (2-14)

式子(2-4)表示该问题的目标是所有批次的子批完工时间最小化。式子(2-5)至式子(2-14)描述该问题的约束条件。式子(2-5)表示子批划分的子批量约束，对每个批次来说，其子批量的和等于该批次的批量。式子(2-6)表示每个子批都在零时刻释放，等待加工。式子(2-7)表示每个子批的每个工序只能从可选机器集合中选择一台机器来加工。式子(2-8)表示每个子批的每个工序最多只需要一次准备操作。式子(2-9)表示如果需要准备操作，那么的准备操作结束时刻等于开始时刻与准备操作时长，如果不需要准备操作，那么的准备操作结束时刻等于开始时刻。式子(2-10)表示不可抢占约束，的加工结束时刻等于开始时刻加上该子批个工件的加工时间。式子(2-11)表示准备操作与工序加工的约束，如果需要准备操作，那么工序加工要在准备操作之后开始。式子(2-12)是计算的加工机器上，前一个子批工序的完成时间。式子(2-13)体现了准备操作可分离，对于的加工机器来说，上一个子批离开之后，就可以开始的准备操作了，而不一定要等待该子批的上一个工序完成。式子(2-14)表示工序的链式约束，的工序要在该子批的上一个工序完成之后才能开始加工。

# 2.5 柔性作业车间分批调度问题求解方法

分批调度要同时求解分批方案和调度方案，但是求解两个有耦合关系的问题，比单纯求解调度或者单纯求解分批问题更复杂一些。Potts曾经尝试把两个问题分开求解，先生成调度顺序，再生成分批方案，实验证明对于多种类工件问题效果并不好[33]。有些学者尝试重复迭代求解这两个问题，先固定调度顺序，使用解析法求解最优的子批划分，再固定子批划分，使用解析法求解最优的调度顺序，不断迭代重复直至收敛，但是对于规模稍大的问题来说，这种方法很容易在局部最优收敛[23]。Wang使用两层遗传算法来迭代求解，上层遗传算法优化分批，下层遗传算法优化调度顺序，每次使用上层种群的一个个体来初始化下层种群，相当于固定分批方案，对调度顺序寻优，由于上层遗传算法的迭代次数远不及下层，因此该算法对分批的寻优是不足的[34]。近年来，一部分学者开始尝试同时优化这两个问题，而不是孤立地看待这两个问题，得到了较为满意的结果。Wong使用了遗传算法，把分批方案和调度方案的信息集合到一条染色体上，对 JSSP的分批和调度同时优化[35]。Han也是使用了类似的策略，使用候鸟迁移算法同时对分批编码和调度编码做邻域搜索，对FSSP的分批和调度同时优化，得到了不错的结果[36]。然而对于LS-FJSP问题的研究还是比较少。同时对分批和调度进行优化能达到精度较高的结果，本文采用这种方法求解LS-FJSP问题。

解决子批划分的方法有多种，按照是否等量，解决子批划分的方法可以分为两类。第一类是等量分批，每个批次都按照等量的原则被划分为若干个子批，每个批次内所有子批的子批量相等。第二类是不等量分批，不需要按照等量的原则划分子批，每个批次内的各个子批批量可以不相等。许多研究采用了等量分批，因为它是最直接、最简单、最容易实现的方法，它只需要确定每个批次的子批数就可以了，不需要额外确定每个子批的子批量，因此它所需要的计算也是最少的[37][13]。但是等量分批不能同时优化子批数和子批量，只能做很粗糙的子批划分，不能根据机器特点灵活分批，得到的分批方案可能比不等量分批差。同时，由于不同机器的性能不一样，不同机器在加工批量相同的同类子批的加工时间不一样，可能会导致机床的空闲时间增加[38]。虽然不等量分批能更灵活地分批，但是搜索不等量分批的最优解比搜索等量分批的最优解难度大，因为其搜索域更大更复杂。只要配合寻优能力较强的求解算法，不等量分批就能得到更优、更精细的分批方案。按照分批是否一致，解决子批划分的方法分为两类。第一类是不一致分批，同一个批次在不同的工序中有不同的子批划分方案，例如子批数可以不同，子批量也可以不同。第二类是一致分批，同一个批次在所有工序中都使用相同的子批划分方案，即在所有工序的加工过程中，所有子批都保持不变。不一致分批是灵活性最高的子批划分方法，但是在实际生产过程中，它需要比较复杂的操作以及管理手段，对生产系统有较高的要求，较少被应用。综上，为了保证分批的精度和保证高应用性，本文采用一致的不等量分批。

# 3.6 本章小结

本章明确了本文待求解的问题模型，并对其进行分析。发现该问题包含两个复杂的子问题，且两个子问题存在耦合关系。同时，该问题变量多、约束数量多、可行域复杂。这些因素导致了问题难以高效优化，这对求解方法提出了很高的要求，因此需要设计既能符合约束，又能同时高效优化两个子问题的算法。

三、候鸟迁移算法理论分析

# 3.1 引言

根据第二章的分析，LS-FJSP问题带有多重复杂约束，求解比较困难。候鸟迁移算法是一种基于邻域搜索的算法，其较强的局部搜索能力可以提高在复杂可行域寻优的能力。本章对候鸟迁移算法进行介绍，分析其缺点和优点，探讨它值得改进的地方，和有利于LS-FJSP问题问题求解的地方。为后面章节的算法改进作理论铺垫。

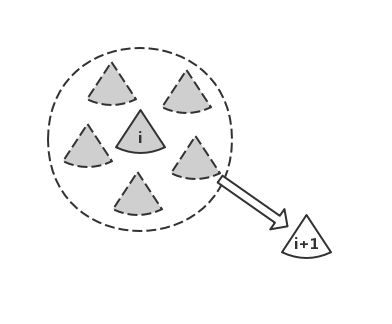
# 3.2 基本候鸟迁移算法

候鸟迁移算法（Migrating Bird Optimization, MBO）是2012年提出的一种优化算法。这种算法模拟候鸟迁移的群体性行为。鸟类在飞行的时候，由于翅膀的结构特点，左右翅尖会产生一定的漩涡气流，可以给后面的鸟提供抬升力，从而节省跟随鸟的能量。候鸟群体经常使用V字型的队伍来飞行，即由一个领头鸟来带领整个鸟群，领头鸟的左右翅膀分别跟着一列跟随鸟，形状跟字母“V”十分相似。Duman认为，候鸟使用V字队形飞行可以节省能量，还可以防止鸟之间的碰撞，并互相保持视觉上的联系[39]。

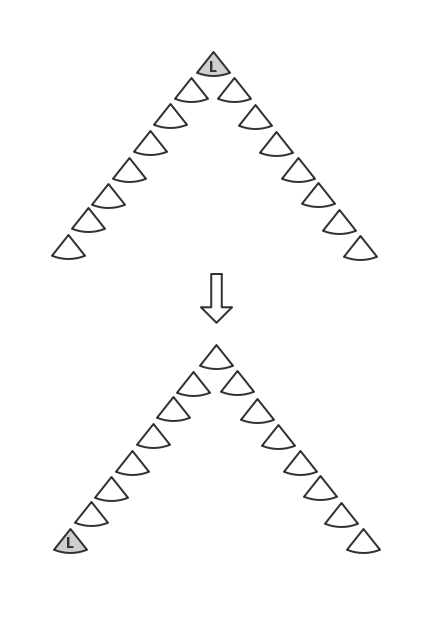


在V字形队伍中，领头鸟需要最多的能量，在左右翼的跟随鸟可以从它们前面那只鸟获得一定的抬升力。比起单独飞行，这种飞行方式可以节省大多数鸟的能量。由于领头鸟需要消耗很多能量，所以领头鸟会定期更换。

MBO是一种基于邻域搜索的算法。从领头鸟，到后面每一只跟随鸟，它们都会通过搜索邻域来更新自身。同时，利益机制还会让前鸟帮助它的跟随鸟进化。具体来说，前鸟会把自己未使用的优秀邻域解分享给它的跟随鸟。如图所示，鸟的跟随鸟是鸟，鸟的一个优秀邻域解会被分享给鸟。



鸟群会保持同一个V字型来更新迭代多次，直到领头鸟累了需要替换，原来V字型才会改变。领头鸟从最前面的位置退到队伍末尾，鸟群的一轮飞行至此结束。左右翼第一只跟随鸟的其中一只会成为新的领头鸟，鸟群的新一轮飞行从此开始，按照新的V字型更新迭代多次。如图所示，领头鸟退到队伍末尾，然后队伍中第一只跟随鸟就成为新的领头鸟。



以下是MBO算法的一些参数：

：鸟群里鸟的数量

：每只鸟邻域候选解的个数

：前鸟给其跟随鸟分享的邻域解的个数

：一轮飞行中迭代的次数

：飞行的总轮数，每轮飞行包含次迭代

以下是描述MBO算法所需要的一些变量：

：鸟群中第只鸟

：领头鸟

：左翼跟随鸟中的第只鸟

：右翼跟随鸟中的第只鸟

：的候选集，里面包含的个候选解

MBO算法的具体流程为：

step1：随机初始化。随机生成个鸟个体，随机选择一直鸟为领头鸟，其余鸟为跟随鸟，并随机将它们组织为V字型，令，；

step2：更新领头鸟。对邻域搜索，得到个邻域解放入候选解集中，如果最优的个体比更优，则用替代；未被使用的邻域解之中最优的个放入左右翼第一只跟随鸟的候选集和中；

step3：更新左跟随鸟；

step3.1：令

step3.2：如果，则需要把的前鸟未使用的个最优邻域解放入候选集；如果，那么进行step3.3;

step3.3：对进行邻域搜索，得到的个邻域解放入候选集；

step3.4：如果候选集中最优的个体比更优，那么用替换；

step3.5：令候选集，如果，则令，否则进行step4；

step4：更新右跟随鸟；

step4.1：令

step4.2：如果，则需要把的前鸟未使用的个最优邻域解放入候选集；如果，那么进行step4.3;

step3.3：对进行邻域搜索，得到的个邻域解放入候选集；

step3.4：如果候选集中最优的个体比更优，那么用替换；

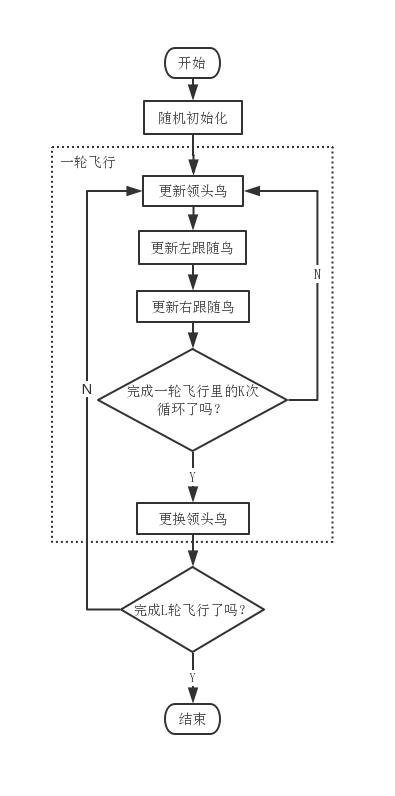
step3.5：令候选集，如果，则令，否则进行step5；

step5：如果，那么，回到step2，否则进行step6；

step6：领头鸟更换，对比左右翼的第一只跟随鸟，如果比更优，那么领头鸟退到左翼的队伍末尾，成为新的领头鸟，否则领头鸟退到右翼的队伍末尾，成为新的领头鸟；

step7：如果，那么进行新一轮的飞行，，，回到step2，否则结束，输出寻优结果。

基本MBO算法的流程图如下所示：



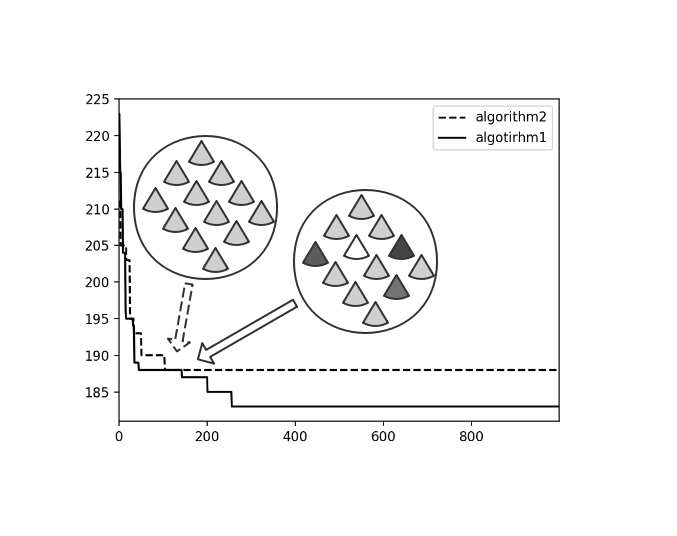
# 3.3 候鸟迁移算法研究现状

# 3.4 候鸟迁移算法性能分析

## 3.4.1 多样性损失分析

对于基于种群的智能进化算法来说，“种群收敛”代表种群的大多数个体进化到一定程度之后趋于一致。种群收敛并不是越快越好，它需要兼顾算法全局搜索和局部搜索的平衡。如果种群收敛得太快，在寻优的前期就严重损失了多样性，在全局搜索还未足够的时候过早地专注于局部搜索了，那么就算种群收敛了，也只能收敛到一个比较差的解。通常这称为“早熟收敛”，或者“停滞”。种群收敛快的算法通常很容易陷入局部最优。出现这种情况是由于多样性损失严重，种群失去了继续寻优的活力。

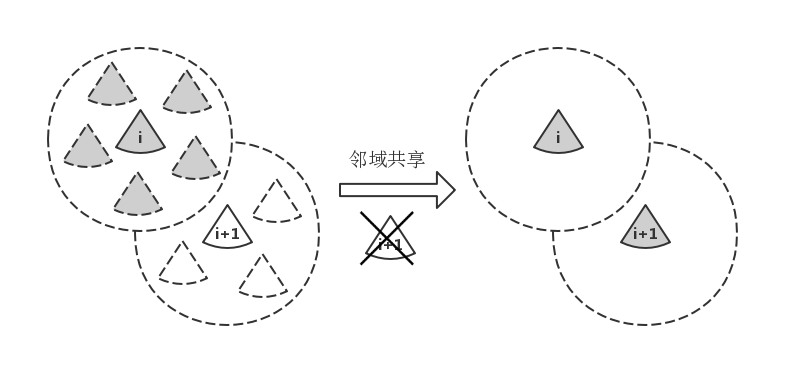
如图所示的最小化优化，算法2，大约在100代的时候开始进入停滞状态，只能在目标值188处收敛，而算法1在100代的时候仍保留着多样性，一直保持搜索的活力知道230代，收敛到目标值183处。



下面详细分析基本MBO算法中加剧多样性损失的一些地方，主要有两个原因，包括基于邻域共享的利益机制，和种群异步更新。

（1）基于邻域共享的利益机制

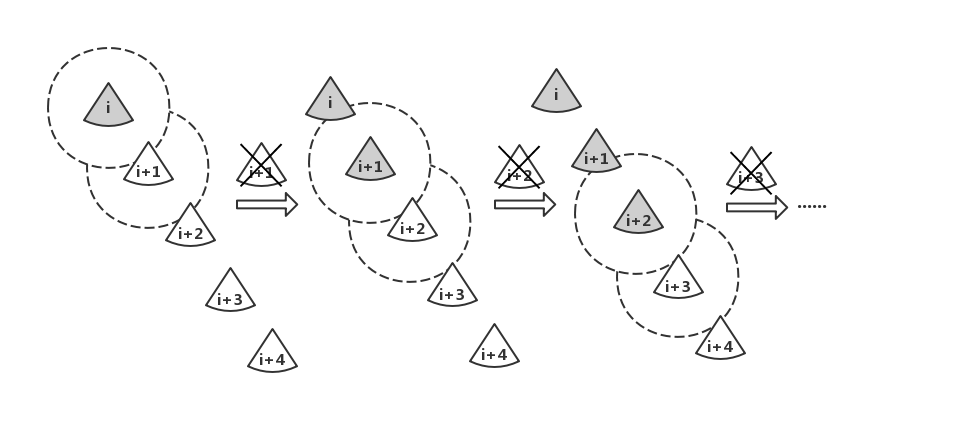
在基本MBO算法中，为了体现前鸟的翅尖对跟随鸟提供抬升力的这种利益机制，前鸟会把自己未使用的优秀邻域分享给其跟随鸟，即进行邻域共享。它的好处是，当某只跟随鸟不能通过自身邻域搜索来提升自己，此时它还可以通过前鸟分享给它的优秀邻域解来提升自己。通过邻域共享，比较差的个体可以在前鸟的协助下，通过一次或者多次迭代快速进化为比较优秀的个体，因此邻域共享有助于种群快速收敛。



基本MBO算法的邻域共享在加快种群收敛的过程中，同时也在大大减少种群的多样性，容易导致早熟收敛，陷入局部最优。如图所示，鸟是鸟的前鸟。前鸟的邻域解替换了当前跟随鸟之后，对于前鸟邻域的挖掘是有利的，有更多个体能集中在前鸟的邻域附近搜索。但是当前跟随鸟所在的邻域却失去了被挖掘的机会，随着迭代不断进行，越来越多有潜力的区域被忽略了，种群中越来越多的个体被局部最优个体替代，种群的多样性损失越来越严重，种群越来越难以跳出局部最优。

（2）种群异步更新

在基本MBO算法中，整个种群所有鸟的更新是异步更新。异步更新指的是，对于V字型的左翼队列或者右翼队列，都是从第一个跟随鸟开始，按照队形顺序，前面一只鸟生成候选集并更新自己之后，后面一只鸟才能根据已更新的前鸟来生成自己的候选集并更新，按顺序直到排末尾的跟随鸟更新，整个种群的更新才完成。如图所示，鸟更新之后，跟随鸟才更新，鸟跟新之后，跟随鸟才能更新，以此类推。



因为异步更新是在前鸟更新之后才去更新的，所以在一次种群的更新过程，前鸟的优秀邻域分享给第一只跟随鸟并更新之后，第一只跟随鸟落入了前鸟的邻域范围，当更新后的第一只跟随鸟把自己的邻域个体分享给第二只跟随鸟并更新之后，第二只跟随鸟也很有可能落入前鸟的邻域范围。使用异步更新模式的邻域共享不仅可以让第一只跟随鸟进入前鸟的邻域，还有可能让后面多只跟随鸟都落入前鸟的邻域，特别是当前鸟都优于后面的多只跟随鸟的时候，这种情况时常发生。在整个种群的一次更新过程中，左翼或者右翼队伍可能会有连续几个个体被其某只前鸟的邻域个体替换，这表示在一次迭代的过程中，异步更新加剧了种群多样性的损失。虽然异步更新加快了种群的收敛速度，但是却让算法更快陷入局部最优。

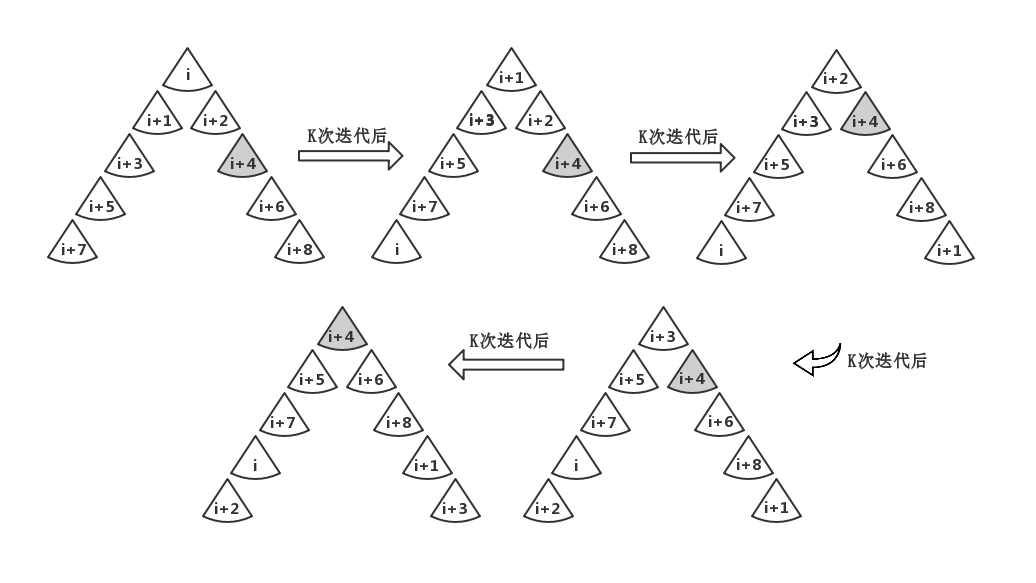
## 3.4.2 群体寻优效率分析

基于群体的进化计算方法的最大的特点是，可以通过个体之间互相协作达到共同进化的效果。个体间协作的效率对算法寻优效率有着很重要的影响。协作的方式一般是把个体按照某种拓扑结构组织起来，然后按照规则互相传播有效消息，互相学习优秀编码。不同的拓扑结构对群体进化有不同影响。高效的拓扑结构可以充分发挥群体协作的潜力，有助于算法的高效搜索。丰富多样的拓扑结构可以让群体有多种进化方向，让种群保持全局寻优的活力。

MBO就属于这种基于群体的进化计算方法。下面分析MBO算法中影响群体寻优效率的一些地方，主要有两个，包括轮替式的领头鸟更换，和过于固定的协作拓扑。

（1）轮替式的领头鸟更换

在基本MBO中，领头鸟的替换是轮替式的，跟随鸟按照V字队形的前后顺序，轮流替换旧的领头鸟，成为新的领头鸟。在领头鸟疲劳了之后，它会退到V字型队伍的队尾，此时只有紧跟领头鸟的两只跟随鸟有机会成为新的领头鸟。无论排在队伍中间和队伍后面的鸟有多优秀，无论它们是否比新的领头鸟优秀，它们在本轮领头鸟替换过程中是没有机会成为领头鸟的。假如种群中有只鸟，那么排在中间的优秀个体最多要等待轮才能成为领头鸟，而且平均需要再平均轮才能再次成为领头鸟，每轮飞行需要迭代次，所以等待时间总计平均需要次迭代。如图中鸟，排在中间的位置，离领头鸟的位置只差一只鸟，但也需要等待次迭代。这种轮替式的领头鸟更换机制并没有让优秀个体在种群中发挥带领作用，影响种群的寻优效率。



（2）过于固定的协作拓扑

在3.3.3.1所分析的轮替制下，领头鸟更替之前和更替之后，几乎所有鸟在V字型队伍里的顺序是没有变化的。假如种群中有只鸟，对于除了领头鸟个左右翼第一只跟随鸟以外的其他鸟来说，它们各自的前鸟，平均每轮飞行之后才可能变化一次，每轮飞行需要迭代次，这就导致了平均次迭代过程中，一只鸟的前鸟都是固定不变的。如图所示，鸟在次迭代之的过程中，其前鸟一直是，跟随鸟一直是。鸟群队伍的排序比较固定和单一，每只鸟的眼光都只能长期局限于某一只前鸟，鸟个体之间协作的拓扑结构比较固定，种群长期只能按照同一个方向去进化，没有充分发挥个体间协作的潜力，不利于种群的高效寻优。

## 3.4.3 局部搜索能力分析

MBO是一种基于邻域搜索的进化计算方法，它的局部搜索能力非常强。对于跟随鸟来说，每一次迭代每个个体都会进行次的邻域搜索，产生个邻域解，每一个个体都会通过自身或者前鸟的邻域解来进化，因此每一只鸟对于自身邻域的探索是比较充足的。对于领头鸟来说，每次迭代都会进行次邻域搜索，产生个邻域解，比跟随鸟产生的邻域解更多，因此领头鸟能得到更充分的邻域探索。总的来说，每一只鸟所在的邻域都可以得到充分的局部搜索。在算法前期，种群多样性还比较高的时候，这种能力有助于同时挖掘多个区域的潜力，在算法后期，种群开始收敛的时候，这种能力有助于深入挖掘目标区域的最优解。

在局部搜索能力方面，相比于以下几种经典群体进化计算方法，MBO与它们的区别在于：

（1）PSO的搜索步长是根据个体在搜索域之间的距离决定的，在搜索中后期种群逐渐收敛的时候，由于个体趋于相近，个体之间的距离非常小，导致搜索步长变得非常小，此时局部搜索的效率可能比较低。而MBO的搜索步长不会因为中后期收敛而变得很小，因此它能保持高效的局部搜索；

（2）ABC主要是跟随蜂在做局部搜索，但是跟随蜂会根据蜜源适应度的大小来选择蜜源，它会偏向于选择适应度高的蜜源。因此适应度低的蜜源很可能得不到局部搜索，这些蜜源得不到有效的局部搜索之后，很快就会被丢弃。而MBO会给每只跟随鸟相同的局部搜索机会，给予充分的局部搜索；

（3）TLBO的教师阶段是最优个体引领整个种群进化，相当于所有个体朝着最优个体的方向做局部搜索。当最优个体多代不变化的时候，整个种群多次迭代都朝着同一个方向去搜索，容易忽略其他方向的潜力。而MBO虽然也有领头鸟带领种群进化，但是领头鸟会定期更换，而且MBO的邻域搜索不会限制个体的进化方向，所以能挖掘更多区域的潜力。

# 3.4 应用于分批调度问题的优势分析

根据后面第四章的设计，本文使用一个矩阵来代表分批方案，其中矩阵的每一个向量称为分批向量，分别代表每个批次的具体分批方案。每一个分批向量都有批量总数约束，如果对两条分批向量进行交叉寻优，那么很大概率会破坏约束，产生不可行的子代，使交叉无效。如果对两条分批向量进行加减运算，得到的和或者差是无意义的，而且也很有可能破坏约束，得到不可行的结果，使运算无效。因此，对分批向量的搜索不能依靠交叉，也不能依靠加减运算，只能依靠基于变异的邻域搜索。

由于本文的分批调度使用不等量分批来解决，所以分批子问题的复杂度非常高，不仅要确定每一个批次的子批的数量，还需要确定每一个子批的子批量。例如工件数为10的批次可以有35种不同的分批方式，工件数为15的批次可以有110种不同的分批方式，工件数为20的批次则可以有434种不同的分批方式。因此该问题对算法的邻域搜索能力要求非常高，分批向量的寻优是分批调度问题中的一个难点。

而根据3.3.3所分析，MBO是一种基于邻域搜索的群体智能进化算法，它刚好契合了分批调度寻优的特点，它的邻域搜索为分批向量的寻优提供了强有力的支持。邻域搜索在不破坏约束的前提下，能对分批向量进行有效的搜索。因此本文使用MBO算法来解决分批调度问题。

# 3.6 本章小结

本章不仅介绍了基本MBO算法的原理，还根据算法结构和特点，从种群多样性损失和群体寻优效率这两方面分析这个算法的缺点。最后分析了它在局部搜索能力方面的优势，并指出它的局部搜索能力恰好契合了LS-FJSP问题的需求，因此得出MBO算法适用于求解本问题的结论。

本章对基本MBO的缺点分析，是后面算法改进的理论依据。对于造成种群多样性损失的两个因素，和造成群体寻优效率的两个因素，在后面的算法设计中都会得到相应的改进。

四、针对分批调度问题的约束处理策略

# 4.1 引言

在把MBO算法应用于LS-FJSP问题之前，还需要考虑问题约束的处理，具体体现在编码解码方案的设计，和搜索算子的设计。根据2.2的分析，LS-FJSP问题同时具有作业车间、柔性、分批的属性和约束，约束条件多而且复杂，这对问题求解产生了两方面的问题：

（1）不可行解导致寻优效率低下。由于约束条件比较多，任何约束在搜索过程中都很容易被打破，得到不可行解。对于不可行解，一般的处理方法包括解码时修补，直接丢弃或者是适应度惩罚。如果采用解码时修补的方法，每次解码时都需要执行一次修补算法，增加解码的时间复杂度，同时会大量改变原始解的编码信息，对该解所做的搜索就白白浪费了，搜索效率低下。如果采用直接丢弃，对于这种约束条件严格且数量多的问题，得到不可行解的概率非常高，如果都直接丢弃的话，群体进化难以进行。如果使用适应度惩罚的话，任由不可行解在种群内进化，可能导致整个种群都是不可行解。总的来说，不可行解的出现会导致算法效率的低下，因此需要避免不可行解的出现[40]。

在本章的4.2节，针对分批调度这一复杂问题设计了一套编码解码方案，它既可以有效表达一个分批调度方案，由于编码结构的独特性，还杜绝了不可行解的出现，提高寻优的效率；

（2）可行域太复杂，导致寻优困难。符合约束的解域是可行域，复杂的可行域具有很不规则的非凸边界，而且可能具有多个峰值，搜索的难度非常大。这要通过增加算法的搜索能力来解决。

在本章的4.3节，针对分批调度这一复杂问题设计了多种搜索算子，使用粗粒度和细粒度配合搜索，融入了启发式及其他复合搜索策略，提高算法在复杂可行域搜索的能力。

# 4.2 编码解码方案设计

编码解码方案是分批调度问题和优化算法之间的桥梁。在使用优化算法求解LS-FJSP问题时，对求解效果有影响的因素有两个，第一个是算法的寻优能力，第二个是编码解码方案的优劣。因此编码解码方案的设计非常重要。

在分批调度问题中，编码代表一个完整分批调度方案的信息，在基于群体的优化算法里，它代表种群内的一个个体。良好的编码方案能有效地表示分批调度方案的信息，还能简化算法的搜索域，避免不可行解的出现，提高寻优的效率。

解码方案是由一个个体的编码求解目标函数的方法。通过解码方案，能够计算得出每个工件在机器上的加工起始时刻和加工结束时刻，可以计算每一台机器最有一个工件的加工结束时刻，从而得到目标函数值。良好的解码方案可以在原始编码的基础上，加入合适的启发式规则来优化调度方案，有助于提高寻优精度。

## 4.2.1双矩阵编码

柔性作业车间的分批调度问题分解为两个子问题，分批问题（SP1）和调度问题（SP2），一个完整的分批调度方案应该包含这两方面的信息，因此一个个体的编码也必须包含这两方面的信息。

分批编码代表SP1的信息，它要确定每一个批次分别要分多少个子批，即子批数，而且每个子批包含多少个工件，即子批量。调度编码代表SP2的信息，它要确定所有子批应该以怎样的顺序在机器上进行加工。分批方案与调度方案存在着互相影响的关系，因此两个编码的设计不能被割裂开来，而是要兼顾两方面的特点和需求。相关学者在这方面设计了不同的方法，做了不同的试验，下面对不同的方法进行分析。

一部分研究为了简化分批的难度，使用等量分批的编码。只需要确定每个批次的子批数，根据等量分批的原则就能确定每个子批的子批量，而不需要在编码里体现子批量[41][36][37]。由此可以简化编码的设计，减小编码的长度。由于编码长度减小了，寻优难度也有所减小，但是这种编码只能优化子批数，不能优化每个子批的批量，因此寻优的结果可能不太理想，本文不采用等量分批编码。

白俊杰提出了一种基于游标的柔性分批编码，同时配合基于工件的调度编码[38]。让每一位基因的每一位数作为分批的游标，这种编码方法虽然能实现不等量分批，而且可以把分批编码的长度控制为与批次数相同，但是基因表示非常累赘。为了保证基因的可行还需要额外多加其他约束，对于批量大于10的实例，还需要使用多位十进制数表示一个基因，对于批量大于10且不为10的倍数的批次，情况就更为复杂，使用非常不方便。在使用搜索算子的时候，非常容易产生不可行解。白俊杰配合使用了传统的基于工序的编码作为调度编码，在编码里体现每个子批的每个工序的顺序。但是对在本问题中，子批数是不一定，不同分批方案下的调度编码长度是不一样的，个体之间进行交叉会产生不可行解。因此基于游标的分批编码与基于工序的编码组合实用性不强。

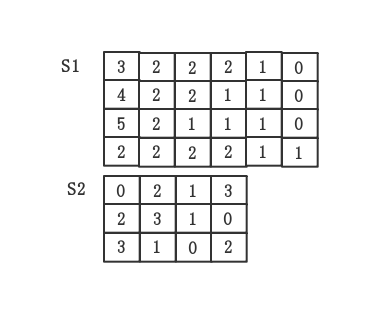
王万良使用了基于分批方案号的编码方法，同时也配合了基于工序的调度编码。首先对每个批次的所有子批划分方案罗列出来并编号，编码分为两部分，用无意义的“0”隔开，第一部分用来指定每个批次的子批划分方案编号，第二部分用来指定每一个子批的大小[42][43]。由于每个批次的子批数不是确定的，该编码第二部分的长度也是不确定的，因此只有编码的第一部分可以进行个体之间的交叉，而对方案编号进行交叉属于间接改变分批方案，难以把握改变的跨度。同时配合使用了基于工序的调度编码，同样是因为不同分批方案下的调度编码长度不一，个体之间交叉会产生不可行解。为了解决这个问题，一些研究规定只有分批方案完全相同的个体才能做调度编码的交叉[41][44]。虽然这能避免进化产生的不可行解，但是可交叉的个体可能很少，特别是在搜索前期，因此严重限制了算法寻优能力。

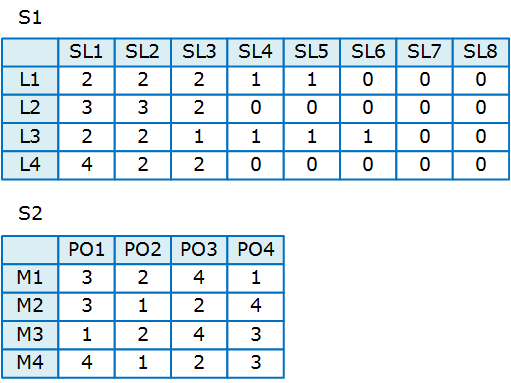
Defersha使用一种实数分批编码，配合基于工序的调度编码[21][45]。每个基因都在之间取值，其值占所有基因总和的比例就是该子批数占批次工件总数的比例。基于工序的调度编码都按照最大子批的规格来构造，以此保证调度编码的长度和批次参数相同，使不同分批方案的调度编码都可交叉，保证解可行。但是实数分批编码使用实数搜索域做离散问题的搜索会出现大量冗余，导致搜索难度增大。

Wong提出了一种3D编码方式，其中的分批编码由多个向量组成，第一个向量表示每个批次的子批数，其余向量分别表示每个批次里每个子批的子批量[46]。虽然分批编码里每个向量的长度不固定，但是向量个数是固定的，例如对于有个批次的实例，就一定有个向量来组成分批编码。而且这种编码方式非常直观，把每个子批量都放在编码里，各种搜索算子可以直接改变子批量的大小。同时配合基于偏好顺序的调度编码，在编码里体现机器对于不同批次工件的偏好顺序。Han也使用了基于偏好顺序的编码，但是他让所有机器都使用同一个偏好顺序[36]。对于柔性车间来说，不同机器的加工特点是不相同的，例如它们可加工的工序不相同，可加工的批次不相同，加工时间也不尽相同。因此让所有机器都使用同一个偏好顺序不能让机器发挥其自身优势，甚至会导致较长的等待时间。而Wong让每台机器都有各自的偏好顺序，解决了这一问题。相比于基于工序的编码，基于偏好的调度编码的优点是能避免不可行解。

根据以上分析，为了避免其他编码方式中出现的各种问题，本文借鉴Wong的思路，改进Wong的3D编码方式，提出一种针对分批调度问题的双矩阵编码。

首先设计一种矩阵式分批编码（S1）。矩阵内每一个向量（分批向量，SV1）都代表着一个批次所有子批的批量，对于有个批次的实例，如果最大子批数为，那么矩阵大小为。相比起Wong的编码，这种编码呈现固定大小的矩阵形式。对于4批次4机器的问题，如果每个批次工件数为8，那么最大子批数为8，调度编码矩阵大小为，每一行代表每个批次的子批量，如图所示，L1~L4分别代表4个批次。对于子批数不足的批次，在向量中填充“0”以达到的长度。通过统计向量中非“0”的基因个数就能知道该批次的子批数。相比于Wong的编码，本编码不需要额外增加一条向量来记录每个批次的子批数。在本文的编码解码方案里，一个批次向量里面每一个基因代表一个子批量，子批量的顺序是无意义的，向量内基因不同的排列方式都代表同一个分批方案，因此会产生冗余。为了减小冗余，需要对每一个批次向量内的基因进行降序的排序，保证每一个编码的唯一性。使用该编码之后，就可以同时优化子批数和各个子批的批量了。





同时配合使用基于偏好顺序的调度编码（S2）。这种编码是Davis在1985年提出的，在车间调度问题里是一种常用的编码方法[47]。Ponnambalam曾经对作业车间的四种常见调度编码做了对比，包括基于工序的编码、基于工序的编码、基于偏好顺序的编码、基于有限规则的编码，通过实验和分析发现，基于偏好顺序的编码是最高效的编码[48]。这种编码在分批调度问题里面也有被使用[40]。当某台机器处于空闲状态，而且有多个待加工的工件，此时机器就按照编码中的偏好顺序，选择待加工工件中偏好度最前的工件[31][32]。基于偏好度的调度编码并不能直接得到工件加工的顺序，而是需要通过解码才能得到具体可行的调度顺序。另一种常用的基于工序的调度编码虽然能表示工件加工的顺序，似乎比基于偏好顺序的编码更直观和简单，但其实编码里的顺序也并不具体可行的调度顺序，它也是需要通过解码才能得到具体可行的调度顺序。在解码过程中，为了保证作业车间的各种约束，编码中许多基因都需要往后移，需要改变原有顺序（建议找一篇基于工序编码的解码文章）。因此换一种角度来看，它与基于偏好顺序的编码实质上都是表示优先顺序的编码，并不直接代表调度过程中的加工顺序。对于分批调度问题来说，使用基于工序的编码很容易出现不可行解，而使用基于优先顺序的编码可以避免不可行解[31]。根据上面的分析，不可行解的出现会限制算法的能力，会降低搜索的效率，因此本文使用基于偏好顺序的调度编码。如图所示，S2矩阵中包含若干个向量， M1~M4分别代表4台机器，每一行是一个向量（调度向量，SV2），代表一台机器的偏好顺序。例如对M1来说，加工的优先顺序是批次3优于批次2优于批次4优于批次1。

## 4.2.2 基于机器柔性指数的解码方式

双矩阵编码中的S1和S2配合使用能有效表达一个分批调度方案的信息，但是如果想得到每个子批的完工时间，还需要进行解码。对于本问题来说，解码是利用调度编码中的信息，根据确定的分批方案，为每个子批的每个工序指定合适的加工机器，确定每个子批的每个工序的加工其实时间和加工结束时间，即生成一个完整的调度方案，并由此计算目标函数值。解码方式有很多种，同一个编码可以使用不同的解码方式，得到不同的调度方案，从而得到不同的目标函数值[40]。因为在不同的规则下，工序的顺序可以不同，工序的等待时间会有差异，工序的加工机器选择也不一定相同。好的解码方式可以把编码解为更优的调度方案，从而提高寻优的精度。

对于作业车间调度来说，特别在使用基于偏好顺序的编码时，解码过程中通常会使用Giffler and Thompson启发式，其思想是，首先选择一台机器，优先选择最早能空闲的机器，然后对该机器可选的待加工工件进行选择，优先选择偏好度高的待加工工件[49][50][32]。但是对于柔性作业车间的分批调度问题来说，需要考虑更多因素，例如属于同一个批次的不同子批，它们的偏好度在同一台机器上是一样的，应该优先加工哪个批次？例如某个子批的某个工序有多个可选机器，其加工时间和准备操作时间各不相同，其空闲时刻也不相同，应该选择哪一台机器来加工？该工序完工时间的最小化是否与所有批次完工时间最小化是否存在矛盾？上述问题如何解决关系到解码效果，对寻优精度有重要影响。为此，本小节提出一种基于机器柔性指数的解码方式，可以在保证问题约束的前提下，引入机器柔性指数，使用针对分批调度问题的规则进行调度。

在柔性作业车间问题里，每一个工件的每一个工序都可以从一台或以上的机器之中选择一台来加工，各台机器之间是有差异的。有些机器能加工的工序多一些，导致会有许多工件同时等待这种机器。而有些机器能加工的工序比较少，导致机器经常处于等待工件的空闲状态，在解码过程中优先给此类机器分配工序。对于机器的这一性质，本节引入柔性指数（Flexibility Index，FI）来定量描述。

 (4-1)

为机器的柔性指数，其中

 (4-2)

本文使用的基于机器柔性指数的解码所采取的思路是，先选择一台机器，选择机器时使用柔性指数来指导选择。FI值较小的机器相对于FI较大的机器，可选择加工的子批的工序总数比较少，因此在解码的时候优先给FI值较小的机器分配工序，以防FI值较小的机器经常处于无可加工工序的空闲状态。然后再生成可在该机器加工的待加工子批工序集合，选出对于该机器来说优先顺序最高的子批。然后根据一步向前看的规则，预计所得子批工序的准备操作及工序加工完成时刻，选择最早完成的子批工序分配给该机器。

在解码的过程中，由于使用了基于FI值的机器选择规则和一步向前看的子批工序选择规则，所以在对基于偏好顺序的调度编码解码的过程中，调度方案也能得到一定的优化。这在一定程度上能帮助该问题的优化。下面对该解码过程详细介绍。

基于机器柔性指数的解码过程中使用的变量如下：

——被选择的机器序号

——被选择的批次序号

——被选择的子批序号

——被选择的工序序号

——机器开始空闲的时刻

——表示每个子批的每个工序的完成状态，如果已经完成加工，则为1，否则为0

——表示所有子批的所有工序的完成状态，如果都完成了，则为1，否则为0

——如果在机器上加工，预计的工序完成时间

——在编码中，对机器来说，批次的优先顺序，1为最高优先级

基于机器柔性指数的解码具体过程如下：

step1：初始化所有机器开始空闲的时刻和每个工序的完成状态，令

 (4-3)

 (4-4)

step2：选择一台合适的机器

step2.1：最早空闲的机器的序号组成集合

 (4-5)

step2.2：如果，那么

 (4-6)

否则要从中选择柔性指数最小的机器

 (4-7)

step3：生成可在中加工的待加工工序集合，对于还未完成加工的工序来说，只要在中需要准备操作且此刻离完成加工所需的时间不大于下一个工序在中的准备操作时间，也可被视作待加工工序

 (4-8)

step4：选择一个合适的子批工序

step4.1：找到中偏好度最前的批次

 (4-9)

step4.3：计算批次的待加工工序在机器上加工预计的完成时间

 (4-10)

step4.3：找到最早能完工的子批工序

 (4-11)

step5：更新一些变量，包括的准备操作开始和结束时刻，的工序加工开始和结束时刻，更新的空闲时刻，更新的完成状态

 (4-12)

 (4-13)

 (4-14)

 (4-15)

 (4-16)

 (4-17)

step6：计算，如果则返回step2，否则调度完成

 (4-18)

# 4.3 搜索算子的设计

根据2.3的分析，LS-FJSP问题在多重复杂约束下的可行域非常复杂，使寻优很有难度。而且由于编码使用了矩阵式的编码，比常见的一维编码或二位编码复杂得多。因此需要有针对性地对LS-FJSP问题设计搜索算子。本节针对不同层面的搜索分别设计了粗粒度和细粒度的算子，粗细粒度算子配合可以优势互补，从编码的不同层面充分搜索。除此之外还融入了启发式及其他复合搜索策略，提高算法在复杂可行域搜索的能力。

## 4.3.1 粗细粒度配合搜索

4.2所设计的编码使用了矩阵的形式，即每个解由两个矩阵表示，而每个矩阵由若干条向量组成，包含了大量的信息，再加上多重复杂约束的限制，进行搜索可谓是戴着镣铐跳舞。根据编码的矩阵形式特点，可以从两个层面进行搜索：

（1）粗粒度层面，即矩阵层面，以S1或S2为单位进行搜索，每次搜索改变矩阵内的多条向量，实现粗粒度的搜索；

（2）细粒度层面，即向量层面，以SV1或SV2为单位进行搜索，每次搜索只改变矩阵内的一条向量，实现细粒度的搜索。

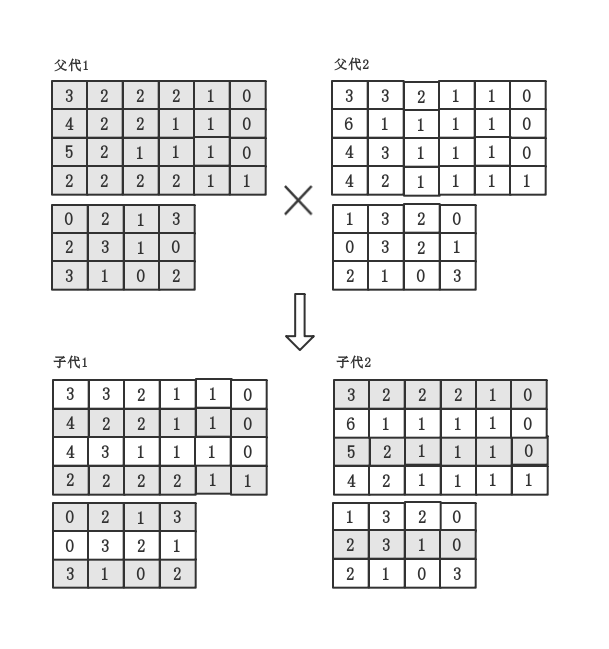
由于粗粒度搜索的步长相对较大，因此有更好的全局视野。而细粒度搜索的步长相对较小，因此有更优的局部挖掘能力。让粗粒度搜索和细粒度搜索配合使用可以同时从不同层面出发进行搜索，有助于平衡全局和局部搜索能力，提高对矩阵式编码寻优的效率。

对高度约束的离散编码来说，常用的搜索算子包括交叉搜索和邻域搜索。交叉搜索通过交换两个解的信息来得到新的解，需要大量的信息交换，符合粗粒度搜索的特点，因此设计了粗粒度交叉搜索算子。邻域搜索算子通过对解自身邻域范围进行探索而得到新的解，需要对解自身做精细的调整和试探，符合细粒度搜索的特点，因此设计了细粒度邻域搜索算子。以下对两种算子进行详细的介绍。

## 4.3.2 粗粒度交叉搜索算子

对于4.2.1的分批调度编码来说，两个由多个分批向量组成，每个向量包含每个批次的分批信息，向量中非零的位数为子批数，向量的每一位代表每个子批的子批量。如果两个分批向量进行交叉，很有可能得到不可行的分批向量，需要额外进行修补。为了避免这种情况，不以向量为单位进行交叉，而以矩阵为单位进行交叉。在两个解交叉的过程中，分批向量内部不改变，但是矩阵内的分批向量在两个解之间互相交换。对调度顺序编码来说，使用同样的规则进行交叉。这种交叉能在保证约束的前提下，在粗粒度层面上有效率地交换两个解的信息。

如图所示，灰色和白色的编码分别表示需要交叉的父代1和父代2。对于4批次3机器的LS-FJSP问题来说，其分批编码包含4个分批向量，调度顺序编码包含3个调度向量。交叉的时候，对这7个向量分别生成7个[0,1]的随机数，如果随机数小于0.5，那么交叉两个父代的对应向量。例如分别生成的随机数为[0.49,0.59, 0.08,0.57, 0.79, 0.04, 0.6]，那么批次1、批次3的分批向量，以及机器2的调度向量都要交换。交换后的子代如图所示，两个个体交换了部分批次的完整分批信息和部分机器的完整调度顺序，得到了新的两个子代。



## 4.3.3 细粒度邻域搜索算子

配合粗粒度算子使用的是细粒度邻域搜索算子，它的目的是从编码中向量的层面上进行寻优，利用其精细的局部搜索能力帮助问题求解。由于LS-FJSP问题约束的复杂性，邻域的设计需要在符合约束的前提下最大程度挖掘邻域的潜力，需要设计具有针对性的邻域。

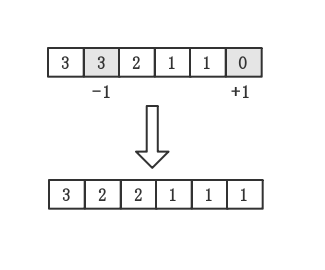
最常见的邻域有基因交换（Swap）、插入（Insert）、倒置（Inverse）、单点变异（Single Point Mutation）。但是交换、插入、反向在分批编码这里完全失去了作用，因为根据分批向量SV1的定义，子批量的顺序是无意义的。而且，单点变异也无法在SV1中使用，因为它只改变了一个子批的子批量，一定会破坏LS-FJSP问题的批量约束。而对于调度向量SV2来说，单点变异也无法使用，因为S2需要包含所有批次的序号，如果只改变一个序号，SV2内一定会出现重复的序号，而且被改变的序号不在SV2中，使SV2无法解码。适用于LS-FJSP问题的邻域需要重新设计，要同时兼顾约束和效率。

本节首先设计了完全符合约束的多种简单邻域，然后针对LS-FJSP问题的调度特点设计了多种启发式邻域，并且针对问题的复杂可行域设计了细粒度全邻域。简单邻域、启发式邻域、细粒度全邻域组成LS-FLSP问题的复合邻域，供解个体随机选择，丰富了该问题的邻域结构。

最后，本节还设计了不同强度和深度的邻域搜索策略，使邻域搜索可在不同策略引导下，表现出不同的特性。不同的策略适用于不同的搜索阶段，适应不同阶段的需求。

（1）简单邻域，针对SV1和SV2，分别设计了多种保证约束的简单邻域。下面详细介绍：

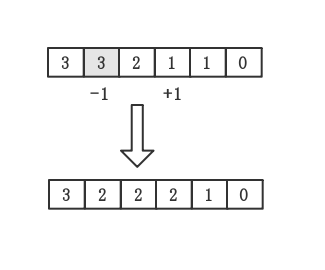
子批对调整邻域（SV1-SPN），是针对分批向量SV1的邻域。由于LS-FJSP问题具有批量约束，一个批次所有子批批量和要与该批次的批量相等，单点子批量调整并不适用，因此设计了子批对调整，保证每次调整之后一定能保证约束。随机选择两个子批，随机增加或减少其中一个子批的子批量，并把改变的量相应地减少或增加到另一个子批。值得注意的是，SV1中的0也可以被选择到子批对中，对0增加一定工件数之后，还达到了改变子批数的效果。因此SV1-SPN同时具有动态调节子批量和子批数的功能。



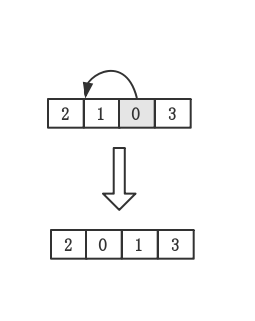
针对调度向量SV2的简单邻域使用了随机插入邻域（SV2-RIN），其原理与传统的插入相同。对于SV2这种基于顺序的编码来说，SV2-RIN和SV2-RSN具有局部改变顺序的良好效果，因此SV2采用了这两个传统的邻域。

（2）启发式邻域

关键子批批量邻域（SV1-CSN），是针对分批向量SV1的邻域。对于LS-FJSP问题而言，关键子批是最后一个完成加工的子批，如果能减小该关键子批的完工时间，那么就有可能减少所有批次的完工时间。基于此规则，可减少关键子批的批量。为了保证约束，需要把子批减少了的工件数加到该批次的其他子批里。如图所示，判断某批次的第二个子批为所有批次加工的关键子批。图中的向量用灰色标记的子批即为关键子批。那么使用CSN邻域的时候，要随机减少该子批的批量，例如这里减少1个工件，并且把减少了的工件量随机分配给其他子批。

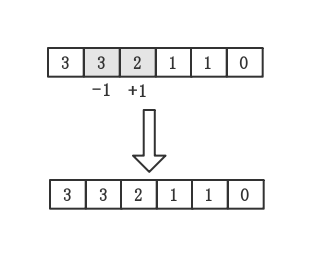


关键子批偏好邻域（SV2-CSN），是针对调度向量SV2的邻域。根据上面的分析，关键子批对完工时间有重要的影响，因此除了调整关键子批的批量以外，还可以调整关键子批所属批次的调度顺序。具体做法是，找到关键子批使用的机器，关键子批所属批次的在该机器的偏好度提前一个位置。



（3）细粒度全邻域

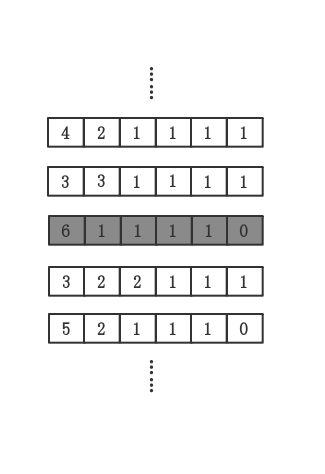
上述的简单邻域和启发式邻域在不同方面具有其各自优势，但是在使用过程中，由于编码的复杂性，可能会出现邻域调整无效的情况，即调整后的个体与调整前个体完全一样。如图所示，使用SV1-SPN邻域来调整某个SV1，得到的结果与原来的完全一样。为了改善这种无效邻域调整的情况，设计了细粒度全邻域，能保证邻域调整的有效性。



SV1细粒度全邻域（SV1-FFN），针对分批向量SV1。在开始优化之前，首先罗列出一个批次中，所有符合批量约束的分批向量，并按照相似性排列，较相似的分批向量排列在相近的位置，组成SV1细粒度全邻域表，如表所示，是含10工件的批次的所有分批向量。在使用SV1-FFN邻域的时候，就可以寻找当前SV1向量在表中的位置，把该位置附近的所有个体都属于邻域个体，可随机选择一个。

生成细粒度全邻域表需要使用排列组合的方法，只需要在优化开始前生成一次，后面就可以一直使用，因此几乎不会增加寻优的时间成本。SV1-FFN使用起来也非常方便，如图所示，灰色的向量是需要做邻域搜索的向量，只要找到它在整个列表中的位置，就可以知道其细粒度全邻域范围有哪些个体，可从邻域中随机选取一个个体。

|  |  |  |  |
| --- | --- | --- | --- |
| **index** | **SV1** | **index** | **SV1** |
| 1 | [1, 1, 1, 1, 2, 4] | 18 | [1, 2, 7, 0, 0, 0] |
| 2 | [1, 1, 1, 1, 3, 3] | 19 | [1, 3, 3, 3, 0, 0] |
| 3 | [1, 1, 1, 1, 6, 0] | 20 | [1, 3, 6, 0, 0, 0] |
| 4 | [1, 1, 1, 2, 2, 3] | 21 | [1, 4, 5, 0, 0, 0] |
| 5 | [1, 1, 1, 2, 5, 0] | 22 | [1, 9, 0, 0, 0, 0] |
| 6 | [1, 1, 1, 3, 4, 0] | 23 | [10, 0, 0, 0, 0, 0] |
| 7 | [1, 1, 1, 7, 0, 0] | 24 | [2, 2, 2, 2, 2, 0] |
| 8 | [1, 1, 2, 2, 2, 2] | 25 | [2, 2, 2, 4, 0, 0] |
| 9 | [1, 1, 2, 2, 4, 0] | 26 | [2, 2, 3, 3, 0, 0] |
| 10 | [1, 1, 2, 3, 3, 0] | 27 | [2, 2, 6, 0, 0, 0] |
| 11 | [1, 1, 2, 6, 0, 0] | 28 | [2, 3, 5, 0, 0, 0] |
| 12 | [1, 1, 3, 5, 0, 0] | 29 | [2, 4, 4, 0, 0, 0] |
| 13 | [1, 1, 4, 4, 0, 0] | 30 | [2, 8, 0, 0, 0, 0] |
| 14 | [1, 1, 8, 0, 0, 0] | 31 | [3, 3, 4, 0, 0, 0] |
| 15 | [1, 2, 2, 2, 3, 0] | 32 | [3, 7, 0, 0, 0, 0] |
| 16 | [1, 2, 2, 5, 0, 0] | 33 | [4, 6, 0, 0, 0, 0] |
| 17 | [1, 2, 3, 4, 0, 0] | 34 | [5, 5, 0, 0, 0, 0] |

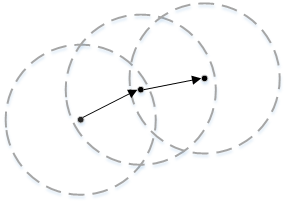


SV2细粒度全邻域（SV2-FFN），针对分批向量SV2。与SV1-FFN类似，生成SV2细粒度全邻域表，当前SV2向量在表中的位置附近的个体都属于淋雨个体。

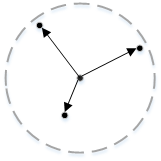
（4）邻域搜索策略

前面所介绍的都是针对LS-FJSP问题设计的邻域结构，使用不同的邻域算子可以在不同邻域范围内搜索。而合适的邻域搜索策略可以帮助邻域算子在其邻域范围内高效搜索。

K步策略（K-Step）。即连续使用K次邻域搜索，每次邻域搜索后得新的解，在新解的基础上进行下一次邻域搜索，一步一步地从一个邻域跳到另一个邻域，实现一定深度的邻域搜索。每次邻域搜索随机选择上面介绍的任一种邻域算子。对于本节的细粒度邻域来说，每次只针对SV1或SV2的邻域搜索，对于整个个体来说，单次的搜索范围相对有限。有时局部最优解与全局最优解之间的距离会超过单个邻域的范围，局部最优需要跨过多个邻域范围才能到达全局最优的位置，导致陷入局部最优的个体难以跳出。K-Step邻域搜索可以跨过一个以上的邻域，能在停滞阶段提高寻优的能力。



K贪心策略（K-Greedy）。即对当前解独立使用K次邻域搜索，得到K个新的解，根据贪心的原则选择其中最好的个体。每次邻域搜索随机选择上面介绍的任一种邻域算子。



# 4.4 本章小结

本章首先设计了适用于LS-FJSP问题的编码解码方案，并把基于柔性指数的规则引入解码方案中，既能避免不可行解的出现，提高寻优效率，又能在规则引导下提高寻优的能力。然后针对LS-FJSP问题编码的特点设计了粗粒度和细粒度的搜索算子，配合使用，从而从不同层面挖掘个体潜力。最后设计了不同搜索策略，用于指引邻域搜索算子的使用，适应不同优化阶段对搜索强度和深度的需求。

本章所设计的编码解码方案及搜索算子，使MBO算法具有应用于LS-FJSP问题的条件，因此将会在后面的算法设计中直接使用。

五、基于竞争式协同候鸟迁移算法的优化调度

# 5.1 引言

根据3.3对MBO算法性能的分析，如果想提高算法的寻优能力，应该从以下两方面着手：

（1）缓解种群多样性的急剧减少

（2）提高群体协同的效率

本章基于这两个出发点，提出了竞争式协同候鸟迁移算法（Competitive Cooperative Migrating Bird Optimization, CCMBO）。CCMBO提出了减速调整阶段和竞争阶段，改进了“轮替式”的领头鸟替换机制，使用竞争式的替换机制。然后改进了基于“邻域共享”的利益机制，改进了异步更新机制，在一定程度上控制了多样性急剧减少的情况。最后丰富了鸟群协作的拓扑结构，提高个体之间协作的效率，提高了算法在复杂空间中的寻优能力。

# 5.2 竞争式协同候鸟迁移算法

CCMBO主要分为三个阶段：竞争阶段、改进的V字飞行阶段、减速调整阶段。经过这三个阶段之后完成一轮飞行，进行一次领头鸟替换，并进入下一轮飞行的三个阶段。

## 5.2.1 竞争阶段

在竞争阶段，要根据竞争的规则确定领头鸟以及V字型队伍的排序。根据3.3.2.1的分析，基本MBO领头鸟的替换是轮替式，优秀个体在种群中难以发挥带领作用，影响寻优效率。而且鸟个体之间协作的拓扑结构比较固定，没有充分发挥个体间协作的潜力，不利于种群的高效寻优。

为了让优秀的个体在种群中充分发挥个体优势的影响作用，同时为了吩咐鸟群的协同拓扑结构，本章给MBO引入竞争阶段。鸟群在减速飞行调整阶段结束之后，鸟群的队形依然是散乱的，经过竞争阶段之后，鸟群可以按照实力优劣竞争领头鸟的位置，确定下一轮飞行的V字型队形，可以形成与上一轮飞行完全不一样的协同拓扑，丰富鸟个体之间互相协作的关系。

在竞争阶段的过程中，所有鸟凭借自身实力角逐领头鸟的位置，越优秀的鸟越有可能成为新的领头鸟。同时个体之间也会互相竞争队伍中间的位置，越优秀的鸟越有可能排在队伍的前面，越差的鸟越有可能排在队伍后面。经过这样的竞争之后，新形成的V字型队形具有一定程度的优劣梯度，这样可以让更优秀的个体有更多机会发挥引领作用，提高寻优的效率，而不是像基本MBO中轮替式的机会均等，优秀个体可能要等待好久才能成为领头鸟。

在大自然的复杂环境下，偶然因素不可避免，竞争阶段也会受到偶然随机因素的影响，具体表现为，最优秀的个体并不一定能成为下一轮飞行的领头鸟，最差的个体也不一定是排在队伍末尾的那个。竞争阶段的准则是，更优秀的个体排在前面的可能性更大，越优秀的个体成为新领头鸟的可能性越大，而不是按照严格适应度排序来构成队形。这种偶然因素的加入可以防止领头鸟的位置被某一个陷入局部最优的个体长期垄断，提高跳出局部最优的能力。即使不是最优秀的个体，只要足够优秀了，也有机会成为领头鸟，这样一来，除了最优秀的个体所在的区域，其他优秀个体所在的区域的潜力，在不同领头鸟的带领作用下，也能得到挖掘。这种加入偶然因素的排序被称为模糊排序。竞争阶段鸟群竞争形成新的V字队形具体过程如下：

step1：对所有鸟进行模糊排序，得到；

step2：成为新的领头鸟，；

step3：如果为偶数，那么加入左翼队列的末尾，即成为，否则加入右翼队列的末尾，即成为；

step4：如果，那么回到step3，否则新的V字队形构建完成。

## 5.2.2 改进的V字飞行阶段

根据3.3.1.1的分析，基本MBO里面基于“邻域共享”的利益机制会加剧种群多样性的损失，让算法更容易陷入局部最优。这种共享模式还有另一个不足的地方。跟随鸟如果被前鸟的优秀邻域个体替换，那么跟随鸟自身所包含的所有信息都会被丢弃。虽然跟随鸟不如前鸟的邻域个体优秀，但是它的编码里也许会包含有价值的代码片段，这些代码片段也被统统丢弃了。

为了缓解“邻域共享”所带来的多样性急剧减少，也为了适当保留跟随鸟自身的优秀代码片段，本章提出使用“代码片共享”以取代“邻域共享”。具体来说，让跟随鸟与前鸟通过交叉，用二者的代码片重新构成一个新的解。其中交叉算子使用4.3.2中设计的粗粒度交叉算子。新的解里面带有前鸟的代码片，也带有跟随鸟的代码片，体现“代码片共享”。这个新的解处于前鸟的邻域范围内，同时也处于跟随鸟的邻域范围内，即两个鸟邻域范围的交集。这种方式既能够体现利前鸟对于跟随鸟的利益机制，又能避免跟随鸟的信息被完全丢弃，使种群保持一定的多样性，保持一定活力，同时还避免跟随鸟所在的邻域被完全废弃，保持对解域的全局搜索，防止早熟收敛。

在CCMBO的V字飞行阶段里，每一只跟随鸟首先与前鸟进行次代码片共享，即交叉。每次交叉得到两个子代，选择较优的子代作为代码片共享的结果。次代码片共享之后得到个新的解，放入候选集里面待跟随鸟选择。接下来，与基本MBO相同，领头鸟是把次邻域搜索得到的邻域解放入候选集，然后择优更新，跟随鸟做次邻域搜索，得到的个邻域解也放入候选集里面。其中邻域搜索使用4.3.3中设计的细粒度邻域算子。此时候选集里就有了个供当前跟随鸟选择的解，如果这个解之中最优解优于当前跟随鸟，那么就用最优解来更新当前跟随鸟。一只跟随鸟更新的具体过程如下，以左翼队列中某一只跟随鸟为例：

step1：令当前鸟的候选集；

step2：令当前鸟与其前鸟交叉，得到两个子代，较优的子代放入候选集；

step3：如果候选集里解个体的数量少于，那么回到step2，否则进行step4；

step4：对当前鸟进行一次邻域搜索，得到一个邻域解，放入候选集；

step5：如果候选集里解个体的数量少于，那么回到setp4，否则进行step6；

step6：找出候选集 最优的解，如果比更优，那么用替换。

值得注意的是，对于种群更新来说，有两种更新方式，异步更新和同步更新。根据3.2.2.2的分析，异步更新会加剧种群多样性的损失，基本MBO属于异步更新。同步更新是指，所有跟随鸟，无论排在队伍中的第几个位置，都是在同一时刻，根据上一次迭代中它的前鸟来生成各自的候选集，然后再使用各自的候选集更新当前跟随鸟。在一次种群的同步更新过程中，前鸟的信息只会分享给紧跟其后的一只跟随鸟，不会对其后多只跟随鸟产生影响，不会加剧种群多样性损失。因此CCMO的种群更新采用同步更新的方式。

V字飞行阶段需要进行次的迭代更新，整个种群采用同步更新一代的具体过程如下：

step1：领头鸟生成候选集，择优更新；

step2：对左翼队列所有鸟， 使用它们各自的前鸟生成各自的候选集

step3：使用各自的候选集对更新；

step4：对右翼队列所有鸟，用它们各自的前鸟生成各自的候选集，

step5：使用各自的候选集对更新。

## 5.2.3 减速调整阶段

当鸟群在领头鸟的带领下加速或者匀速飞行时，鸟群呈现出标准的V字队形。但是当领头鸟更替过程中导致鸟群减速的时候，鸟群的V字队形会被打乱。当领头鸟感到疲劳的时候，它会减慢飞行速度，退到鸟群里面，不再飞在鸟群的最前方。跟随鸟也各自调整自己的速度，由于每只鸟速度不一致而无法保持原来的V字队形。根据这一个鸟类飞行现象，本章提出减速飞行调整阶段，来模拟V字队形被打乱后，鸟个体在散乱队形中自由试探自由飞行的行为。



在这个阶段里，鸟个体两两之间原本比较固定的跟随与被跟随的关系被打破，每只鸟都会不断试探，跟随不同的鸟，调整自己飞行的方向和速度。与V字飞行阶段类似，减速飞行调整阶段也需要几个循环才能完成，循环次数设为。在减速飞行调整阶段的每个循环里，每只鸟都会选择另一只鸟去跟随，同一只鸟在不同的循环里会跟随不同的鸟，以体现鸟个体在调整速度和试探方向的过程中，比较无序的行为特点和散乱的队形结构。在这个阶段，一只鸟跟随另一只鸟所代表的搜索与V字飞行阶段是一样的，都是进行信息共享。只不过减速飞行调整阶段中，跟随与被跟随的关系的动态变化的，在V字飞行阶段，跟随与被跟随的关系是固定不变的。减速调整阶段需要进行代的迭代更新，整个种群每次更新的过程如下：

step1：令；

step2：令鸟个体的候选集，为随机选择另一只鸟，与交叉得到两个子代，选择较好的子代放入候选集；

step3：如果候选集里面个体的数量小于，那么回到step2，否则进行step4；

step4：鸟个体进行一次邻域搜索，把邻域解放入候选集；

step5：如果候选集里面个体的数量小于，那么回到step4，否则进行step6；

step6：如果，那么，否则结束本次迭代。

## 5.2.4 算法流程及复杂度分析

CCMBO主要由四个阶段组成，分别是：

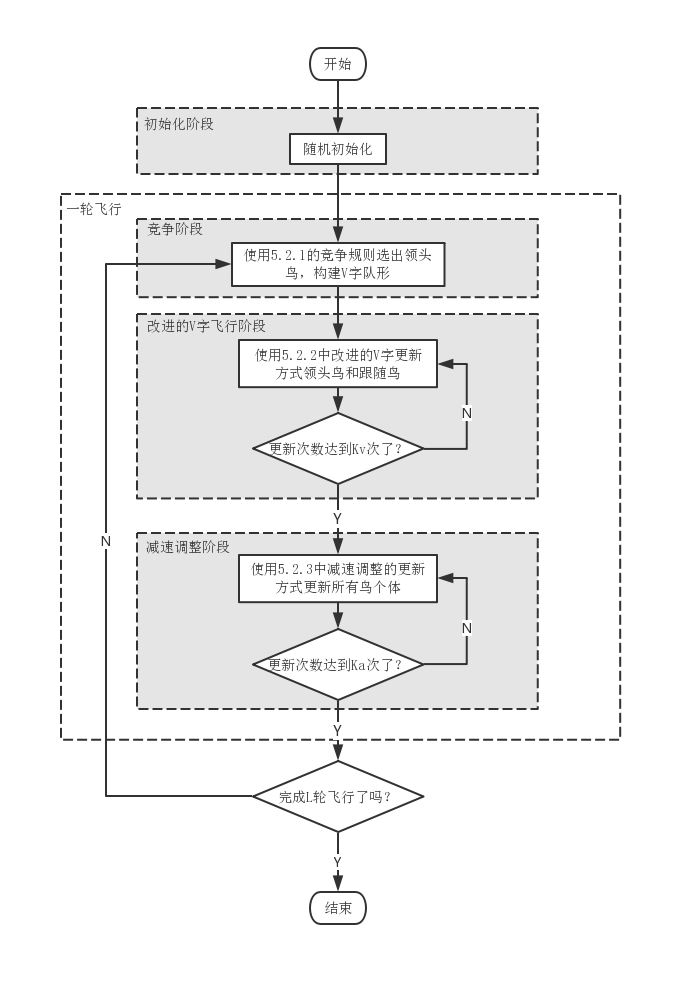
（1）初始化阶段。为了体现算法的鲁棒性，使用随机初始化，不使用加入启发式的初始化。

（2）竞争阶段。每只鸟使用5.2.3的方法选出领头鸟，并构成V字队形，准备下一轮飞行。

（3）改进的V字飞行阶段。保持V字队形不变，所有鸟使用5.2.1的更新方法更新代。

（4）减速调整阶段。打破原有的V字队形，所有鸟使用5.2.2的更新方法在散乱的鸟群中更新代。

算法流程图如下：



初始化阶段结束之后，种群进入一轮飞行。首先进入竞争阶段，按照5.2.1的竞争的规则选出领头鸟，确定V字队形的排列顺序，然后进入V字飞行阶段，使用5.2.2的更新方法对领头鸟和跟随鸟更新代，最后进入减速调整阶段，使用5.2.3的更新方法对每只鸟更新代。竞争阶段只是改变队形，并没有对个体进行更新，因此在一轮飞行中，整个种群一共进行次的更新。完成一轮飞行之后，如果还未达到搜索结束条件，就使用新的V字队形进行下一轮的飞行。

令鸟群大小为，染色体维度为，在最坏的情况下，CCMBO的时间复杂度为：

（1）初始化阶段，对个个体做随机初始化，时间复杂度是

（2）竞争阶段，个个体进行排序，选出领头鸟，构建新的V字型队伍，本阶段时间复杂度是；

（3）改进的V字飞行阶段，个个体通过交叉或者邻域搜索的方法各自生成个候选解，如此更新代，本阶段的时间复杂度是；

（4）减速调整阶段，个个体通过邻域搜索各自生成个候选解，如此更新代，本阶段的时间复杂度是。

综合以上的分析，和代表着优化问题的规模，、和是与和无关的常量，因此对任何规模的优化问题来说，CCMBO算法的时间复杂度为：



# 5.3 实例仿真以及性能评价

为了验证CCMBO算法的效果，本节把CCMBO应用在标准的LS-FJSP分批调度问题，以及全自动免疫检验设备的实例上。首先对4.3.3所设计的邻域搜索算子进行实验，验证其有效性。然后对CCMBO各个改进的部分进行实验，与基本MBO进行对比以验证改进的效果。

本节使用Python3.6编写算法，在3.20GHz，16.0GB的计算机上进行实验。

实验用例一共8个， P1~P4是来自Zhao的LS-FJSP实例，P5~P8是全自动免疫检验设备的应用实例[51]。所有实验用例类型都属于可混排，带有准备操作，以及准备操作可分离，并且以完工时间最小化作为目标。使用来描述测试用例的规模，其中代表批的数量，代表机器数，代表每个批次的工件总数。表展示了8个测试用例的信息，包括批次数、工件数、工序数、机器数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **problem No.** | **批次总数** | **每批次的工件数** | **每批次的工序数** | **机器数** |
| **P1** | 4 | 8,8,8,8 | 3,3,3,3 | 6 |
| **P2** | 4 | 20,20,20,20 | 3,3,3,3 | 6 |
| **P3** | 6 | 10,10,10,10,10,10 | 6,6,5,6,6,6 | 6 |
| **P4** | 6 | 20,20,20,20,20,20 | 6,6,5,6,6,6 | 6 |
| **P5** | 3 | 8,8,8 | 6,6,6 | 8 |
| **P6** | 4 | 15,15,15 | 6,6,6 | 8 |
| **P7** | 5 | 8,8,8,8,8 | 6,6,6,6,6 | 8 |
| **P8** | 5 | 15,15,15,15,15 | 6,6,6,6,6 | 8 |

## 5.3.1 验证邻域搜索算子的有效性

本小节对4.3.3节设计的简单邻域算子（SV1-SPN，SV2-RIN）、启发式邻域算子（SV1-CSN，SV1-CSN）、细粒度全邻域算子（SV1-FFN，SV2-FFN）的效果进行验证。本节设计了两个实验，从不同角度验证邻域算子的作用。

实验一是验证6个邻域算子算法寻优过程中发挥作用的能力。把这6种邻域搜索算子应用到基本MBO算法中，MBO的参数取，，，，。每次需要邻域搜索时从这6种邻域算子随机选取一个，因此在收敛过程中，每种邻域算子的使用机会是均等的。实验记录8个测试用例的寻优过程中，每种邻域算子的成功次数，记为S.T.，即使邻域搜索得到的新个体比原个体更优的次数。为了使数据更有说服力，本实验对每个测试用例都执行20次优化算法，每个测试用例在此过程中，每种邻域算子S.T.如表所示。表中加黑的数字表示每个测试用例的S1邻域算子和S2邻域算子中S.T.的最大值。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **S.T.** | **S1** | | |  | **S2** | | |
| **SV1-SPN** | **SV1-CSN** | **SV1-FFN** |  | **SV2-RIN** | **SV2-CSN** | **SV2-FFN** |
| **P1** | 605 | 609 | **800** |  | 355 | **369** | 383 |
| **P2** | 1145 | **1492** | 896 |  | 443 | 426 | **553** |
| **P3** | 951 | 924 | **1132** |  | 800 | **920** | 790 |
| **P4** | 1541 | **1777** | 1328 |  | 924 | **1059** | 811 |
| **P5** | 714 | 723 | **815** |  | 397 | 426 | **434** |
| **P6** | 1048 | **1289** | 1152 |  | 681 | 664 | **728** |
| **P7** | 817 | **834** | 806 |  | **463** | 452 | 419 |
| **P8** | 1235 | 1304 | **1368** |  | **943** | 854 | 890 |
| **total** | 8056 | **8952** | 8297 |  | 5006 | **5170** | 5008 |
| 25305 | | |  | 15184 | | |

对于S1的邻域算子来说，在P2、P4、P6、P7这4个算例中，SV1-CSN的S.T.值最大，而在其余4个算例中，SV1-FFN的S.T.值最大。SV1-SPN没有在任何算例中达到最大的S.T.值，但是其S.T.总数与其他两个算子差距不超过11%。对于S2的邻域算子来说，在P1、P3、P4这3个算例中，SV2-CSN的S.T.值最大，在P2、P5、P6这3个算例中，SV2-FFN的S.T.值最大，而SV2-SPN在P7和P8这两个算例中S.T.值最大。这说明基于关键子批的启发式邻域，和细粒度全邻域，对于LS-FJSP问题寻优的帮助比简单邻域更多。其中，对于S1和S2来说，引入了启发式的算子SV1-CSN、SV2-CSN，在8个用例中的总S.T.值是最多的，说明引入启发式的邻域算子对于问题求解有最好的效果。

对比S1和S2，可以发现S1的邻域算子的S.T.值总体比S2的邻域算子的S.T.值稍大，在不同算例中，两者的差距不一样，但是总体来说差距明显。造成这样的原因并不是S1的邻域算子搜索能力比S2的邻域算子强，而是因为在LS-FJSP问题中，分批子问题比较复杂，造成S1编码结构复杂，该编码的优化更加复杂，因此需要经过更多次数的S.T.值才能得到满意的优化结果。这也验证了2.3.2中认为分批子问题复杂的结论。

图展示了SV1-SPN、SV1-CSN、SV1-FFN各自的S.T.值占S1所有邻域算子S.T.值总和的比例，图展示了SV2-SPN、SV2-CSN、SV2-FFN各自的S.T.值占S2所有邻域算子S.T.总和的比例。对于S1和S2来说，三种算子的占比相差不大，说明由简单邻域、启发式邻域、细粒度全邻域在寻优过程中都有发挥各自的优势，并互相配合，这三者组成的复合邻域可以在不同方面，针对LS-FJSP的特点取长补短。

实验二是从寻优结果上验证所设计的邻域算子的效果。比较只使用普通邻域的MBO算法（记为MBO）和同时使用普通邻域、启发式邻域、细粒度全邻域的MBO算法（记为MBO+NN）。两个算法的参数与实验一相同。每个算法对每个算例执行30次的优化，结果如表所示。30次优化结果的平均值记为“Avg”，标准差记为“Std”，30次的最优结果和最差结果分别记为“Best”和“Worst”，找到已知最优解的次数记为“BKN”。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **problem No.** | | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
| **MBO** | **Avg** | 86.57 | 190.23 | 201.27 | 386.97 | 126.37 | 219.30 | 195.13 | 342.37 |
| **Std** | 2.03 | **4.10** | **2.63** | **4.07** | **0.60** | 1.81 | 2.29 | 1.83 |
| **Best** | 83 | 185 | 198 | 378 | 125 | 217 | 189 | 340 |
| **Worst** | 91 | 200 | 207 | 392 | 127 | 222 | 198 | 345 |
| **BKN** | 4 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| **MBO+NN** | **Avg** | **85.83** | **189.40** | **200.60** | **385.43** | **126.23** | **218.70** | **194.20** | **341.57** |
| **Std** | **1.69** | 4.57 | 3.03 | 4.50 | 0.67 | **1.49** | 2.29 | **1.82** |
| **Best** | 83 | **184** | **196** | 378 | 125 | 217 | 189 | **339** |
| **Worst** | **89** | 200 | 207 | 392 | 127 | 222 | **196** | 345 |
| **BKN** | **5** | 0 | 0 | 0 | **4** | 0 | 0 | 0 |

使用三种邻域的MBO+NN对8个算例的Avg比单纯使用简单邻域的MBO要小。对于P2、P3、P8来说，MBO+NN的Best比MBO更优，即在30run里找到了更优的结果，在其他算例的Best值相同。对于P1、P7来说，MBO+NN的Worst比MBO的更小，即在30run里最差的结果更优了，在其他算例中的Worst值相同。对于P1和P5来说，MBO+NN找到已知最优解的次数更多。因此实验证明MBO+NN在寻优能力方面比MBO更强，验证了所设计的启发式邻域和细粒度全邻域确实可以寻优的精度。

表只能反映两个算法对不同问题的求解表现，而表通过统计Df值来反映两个算法的整体表现。Df值的定义是，某种算法对所有算例的30次优化结果相比于目前已知最优解的平均偏差，用例的平均偏差计算方法如下：

 (5-1)

其中为算例的目前已知最优解，代表使用某种算法，对算例第次优化求解的结果。该指标可以衡量总体优化结果距离已知最最优解的百分比，差距Df值越小证明该算法的寻优能力更强，越有能力找到最优解。而且，不同算例，使用不同算法得到的Df值的量纲相同，可以直接求和或者求平均，因此使用Df值可以对比不同算法对所有算例的综合优化能力。

根据表中的数据，MBO+NN对P1~P8的Df值稍小于MBO。根据表中Df平均值所绘制的柱状图，MBO+NN对8个算例Df的平均值也小于MBO。由此可见，在使用了基于关键子批的启发式邻域和细粒度全邻域之后，算法的总体寻优能力有所提升。

|  |  |  |
| --- | --- | --- |
| **problem No.** | **MBO** | **MBO+NN** |
| **P1** | 1.95% | 1.55% |
| **P2** | 3.95% | 3.50% |
| **P3** | 4.83% | 4.48% |
| **P4** | 4.30% | 3.89% |
| **P5** | 1.09% | 0.99% |
| **P6** | 2.00% | 1.72% |
| **P7** | 3.79% | 3.30% |
| **P8** | 1.59% | 1.36% |
| **Avg** | 2.94% | **2.60%** |

## 5.3.2 验证CCMBO的有效性

本小节对5.2所做3处改进的效果进行验证。3处改进包括使用竞争阶段、使用改进的V字飞行阶段、使用减速调整阶段。首先，3个改进被分别引入到基本MBO中，引入了竞争阶段的MBO记为MBO+compete，引入了改进V字飞行阶段的MBO记为MBO+improvedV，引入了减速调整阶段的MBO记为MBO+adjust。这三个算法与基本MBO进行对比，以验证改进的有效性。最后，引入了这三种改进的CCMBO算法与上述几种算法进行对比，以验证该算法的有效性。其中MBO 的参数与5.3.1相同，CCMBO的参数取，，，，，，为了公平起见，所有算法的迭代次数相同。MBO、MBO+compete、MBO+improvedV、MBO+adjust、CCMBO这五种算法对8个算例的30次求解结果如表所示。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **problem No.** | | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
| **MBO** | **Avg** | 86.57 | 190.23 | 201.27 | 386.97 | 126.37 | 219.30 | 195.13 | 342.37 |
| **Std** | 2.03 | 4.10 | 2.63 | 4.07 | **0.60** | 1.81 | 2.29 | 1.83 |
| **Best** | 83 | 185 | 198 | 378 | 125 | 217 | 189 | 340 |
| **Worst** | 91 | 200 | 207 | 392 | 127 | 222 | 198 | 345 |
| **BKN** | 4 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| **MBO +compete** | **Avg** | 85.57 | 189.07 | 200.40 | 383.10 | 126.07 | 218.73 | 193.60 | 341.13 |
| **Std** | **1.52** | 3.96 | 3.38 | 3.68 | 0.73 | 1.97 | 2.37 | 2.06 |
| **Best** | 83 | 184 | 197 | 376 | 125 | 216 | 189 | **337** |
| **Worst** | **88** | 198 | 207 | 390 | 127 | 224 | **196** | 344 |
| **BKN** | 4 | 0 | 0 | 0 | 3 | 0 | 0 | 1 |
| **MBO +improvedV** | **Avg** | 85.53 | 188.70 | 199.93 | 382.60 | 126.10 | 218.47 | 193.57 | 340.53 |
| **Std** | 1.73 | 3.84 | 2.91 | 5.38 | 0.70 | 1.61 | **2.25** | 1.93 |
| **Best** | 83 | **183** | **196** | 376 | 125 | 216 | **188** | **337** |
| **Worst** | 89 | **195** | 206 | 394 | 127 | 224 | **196** | 344 |
| **BKN** | 5 | 3 | 0 | 0 | 6 | 0 | 1 | 2 |
| **MBO +adjust** | **Avg** | 85.27 | 188.83 | 199.17 | 382.30 | 125.93 | 218.07 | 192.97 | 340.40 |
| **Std** | 1.53 | 4.12 | 2.90 | 4.45 | 0.63 | **1.48** | 2.82 | 1.85 |
| **Best** | 83 | **183** | **196** | 376 | 125 | 216 | **188** | **337** |
| **Worst** | **88** | 197 | 206 | 394 | 127 | **221** | **196** | 344 |
| **BKN** | 6 | 3 | 0 | 0 | 7 | 0 | 3 | 2 |
| **CCMBO** | **Avg** | **85.13** | **188.10** | **197.87** | **380.73** | **125.80** | **217.90** | **192.07** | **340.10** |
| **Std** | 1.54 | **3.03** | **1.33** | **3.51** | 0.65 | 2.15 | 2.79 | **1.74** |
| **Best** | 83 | **183** | **196** | **374** | 125 | **215** | **188** | **337** |
| **Worst** | **88** | **195** | **200** | **386** | 127 | **221** | **196** | **343** |
| **BKN** | **7** | **4** | 0 | 0 | **10** | **1** | **5** | **4** |

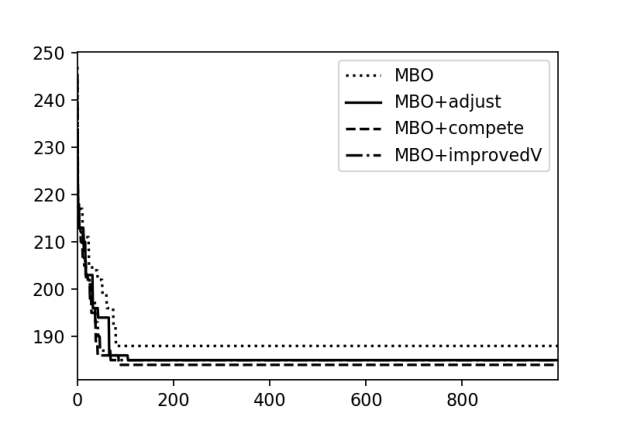
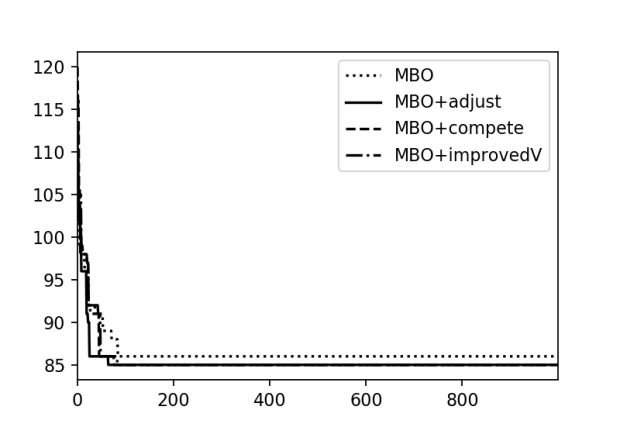
首先用MBO+compete、MBO+improvedV、MBO+adjust分别与MBO进行对比。MBO+compete对于P1~P8的Avg均比MBO更小，说明平均结果得到了优化，然而对于P3、P5、P6、P7、P8来说，Std值有所上升，说明把轮替式领头鸟更换机制改为竞争式之后，寻优的稳定性有所下滑，因此不能只使用竞争阶段。MBO+improvedV对所有测试用例的Avg与MBO相比都更小了，而且对于P2、P7、P8来说，MBO无法求得已知最优解，BKN均为0，而MBO+improvedV求出了已知最优解，BKN分别达到了3、1、2，而P1、P5的BKN也比MBO的更高了，分别上升了1和4。这说明改进后的V字飞行阶段求取最优解的能力有所提升。MBO+adjust对所有测试用例的Avg相比MBO有比较明显的减小，而且对于P1、P2、P5、P7、P8来说，BKN相比于MBO算法有明显的增加，对P2~P4、P6~P8求得的最优结果Best也比MBO更小了。这说明加入减速调整阶段后，算法的寻优能力有了显著提高。

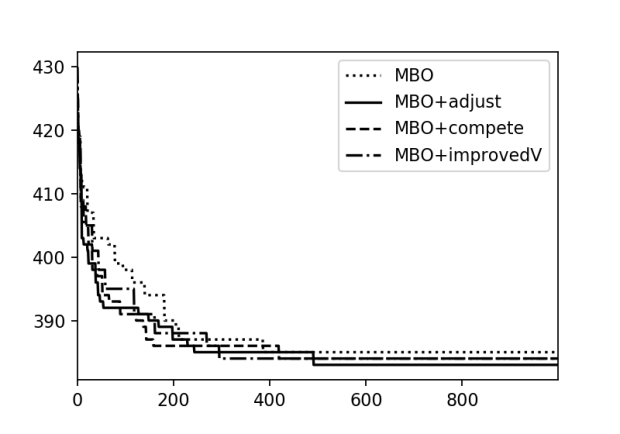
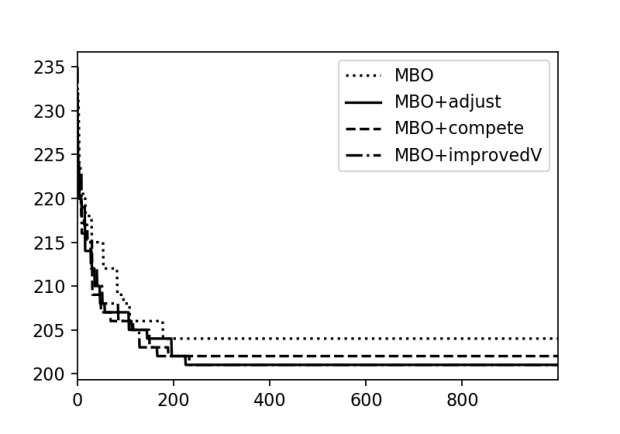
然后把CCMBO与MBO进行对比。根据表，从Avg来看，CCMBO对所有测试用例的优化精度都比MBO显著提高了，从Std角度来看，CCMBO对P1~P4、P8的求解稳定性也有所提高，从Best角度来看，除了P1和P5这两个问题规模最小、求解最容易的问题以外，CCMBO对其他测试用例30次求解得到的最优结果都比MBO更优，从Worst角度看，除了P5之外，CCMBO30次求解的最差结果都比MBO更优了，从BKN来看，除了对P3、P4这两个大规模问题以外，CCMBO找到已知最优解的次数比MBO都有明显提高。这说明引入三个改进之后，算法在寻优精度、稳定性等方面都得到了显著的提升。

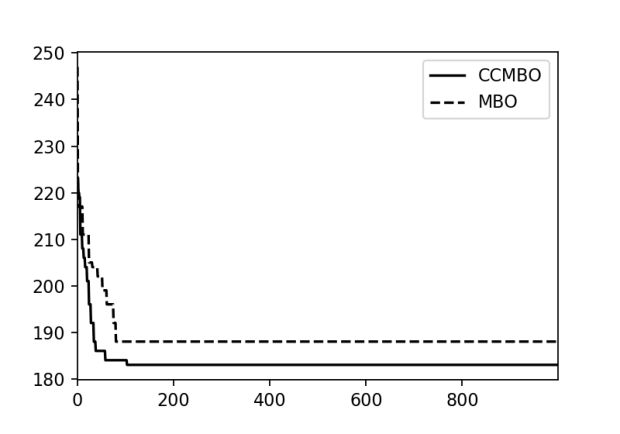
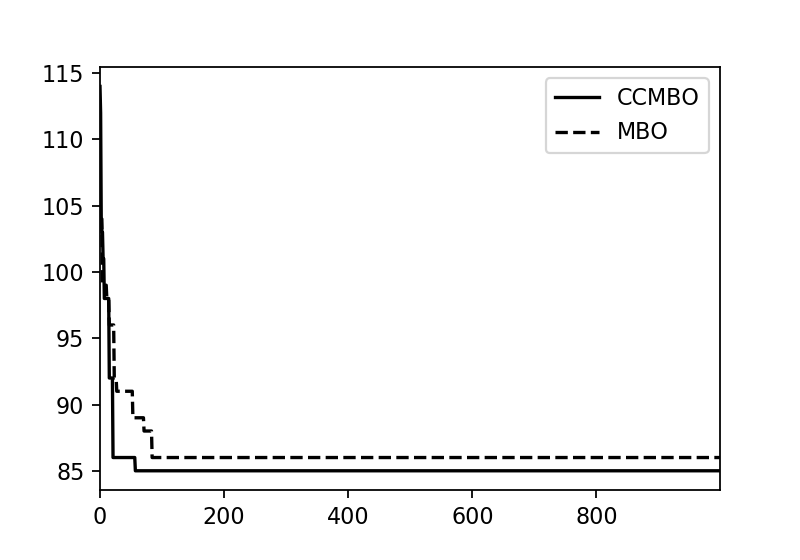
接下来计算Df值，对比这五个算法的综合性能，针对每个测试用例的计算结果及其平均Df值见表，使用Df平均值绘制的柱状图见图。从平均Df值（即Avg）来看，MBO+compete、MBO+improvedV、MBO+adjust的Avg比MBO有不同程度的减小，说明三个改进在不同程度上提升了算法的综合寻优能力。其中MBO+compete、MBO+improvedV、MBO+adjust之中，MBO+adjust的Avg是最小的，说明减速调整阶这一改进所发挥的作用比最大。CCMBO的Avg比其他四个算法明显小很多，说明三个改进结合到CCMBO之后，三个阶段互相配合，有效提高了算法的寻优能力。

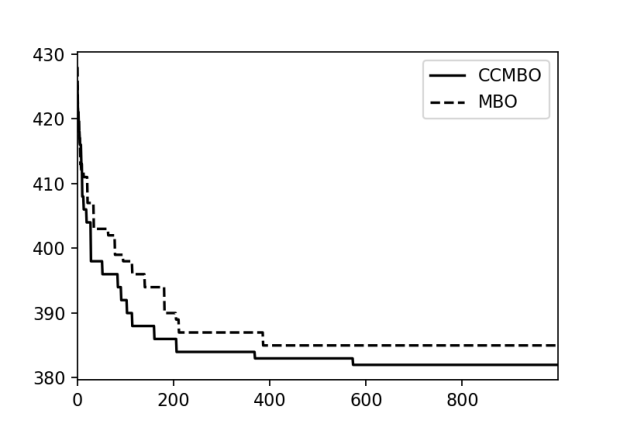
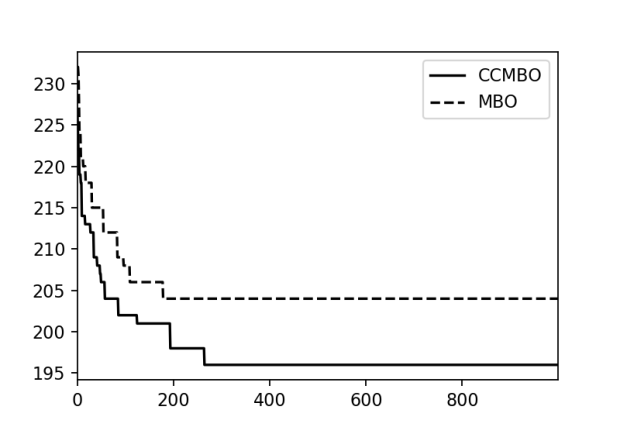
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **problem No.** | **MBO** | **MBO+compete** | **MBO+improvedV** | **MBO+adjust** | **CCMBO** |
| **P1** | 1.95% | 1.40% | 1.38% | 1.24% | 1.17% |
| **P2** | 3.95% | 3.32% | 3.11% | 3.19% | 2.79% |
| **P3** | 4.83% | 4.38% | 4.13% | 3.73% | 3.06% |
| **P4** | 4.30% | 3.26% | 3.13% | 3.05% | 2.62% |
| **P5** | 1.09% | 0.85% | 0.88% | 0.75% | 0.64% |
| **P6** | 2.00% | 1.74% | 1.61% | 1.43% | 1.35% |
| **P7** | 3.79% | 2.98% | 2.96% | 2.64% | 2.16% |
| **P8** | 1.59% | 1.23% | 1.05% | 1.01% | 0.92% |
| **Avg** | 2.94% | 2.39% | 2.28% | 2.13% | **1.84%** |

最后从根据收敛曲线图来验证改进的效果，图展示了MBO、MBO+compete、MBO+improvedV、MBO+adjust某对P1~P4单次求解的收敛图，其中实线表示本章所提出的CCMBO算法。MBO+compete由于使用了竞争式的领头鸟替换，种群由优秀个体引领寻优，其收敛速度相比MBO有了明显的提升。MBO+adjust由于加入了允许自由学习的减速调整阶段，在中后期的寻优能力上有所提升，如图所示，在其他算法已经收敛的时候，MBO+adjust还能继续优化。图展示了CCMBO与MBO对P1~P4单次求解的收敛图。引入了三种改进之后，算法的种群多样性和协助效率得到了提高，因此收敛前期的速度提升了，后期跳出局部最优的能力也提升了。









六、基于多领头鸟分化协同候鸟迁移算法的优化调度

# 6.1 引言

经过第五章的改进和实验，CCMBO算法的寻优性能已经比基本MBO的性能好了，但是寻优精度还有待提升，寻优的稳定性和鲁棒性也稍显不足。

Nathan和Barbosa指出，在候鸟群的长途迁徙过程中，鸟群的队形有时也会形成除了V字型以外的编队，有时还会形成多个组群[52]。对于基于群体的进化算法来说，多种群协同进化是一种有效提升算法寻优能力的手段[53]。Defersha指出，多种群协同进化并不是算法的简单并行版本，而是一种与传统方式不同的方式来搜索解空间[21]。多种群协同允许不同特征的子种群形成，可以进化出不同特征的个体，有更好的多样性，同时子种群内部也能形成很好的局部搜索环境。

本章在CCMBO的基础上，对算法进行多种群协同的改进，提出多领头鸟分化协同候鸟迁移算法（Multi-Leader Competitive Cooperative Migrating Bird Optimization，ML-CCMBO）。首先设计了多领头鸟的分化协同机制，不同的领头鸟带领不同的子鸟群，不同的子鸟群分化出不同的进化方向，通过基于交换的子种群迁移来进行协同进化。其次设计了阶段性邻域搜索，对不同阶段的个体执行不同的邻域搜索策略，在停滞阶段，执行不同程度的退化，以激发个体的搜索能力。

# 6.2 多领头鸟分化协同候鸟迁移算法

## 6.2.1 多领头鸟分化协同机制

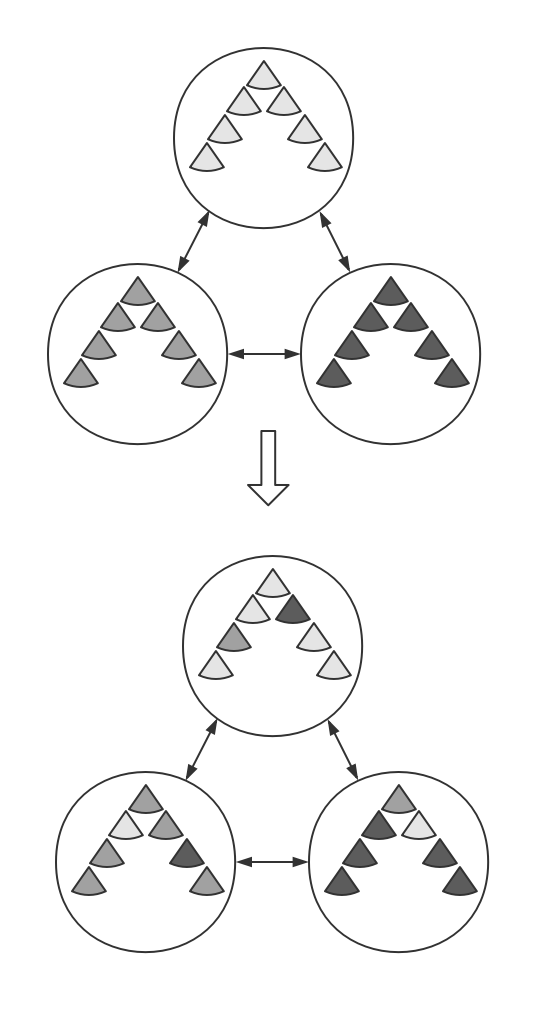
多种群协同不是简单地把一个大种群分为多个种群，而是通过子种群独立进化来维护多样性，通过子种群间的信息交流来提高效率，通过各种种群迁移拓扑来实现高效的进化模式。

Kurdi提出，可以对不同子种群使用不同的算子，以产生不同的子种群环境[54]。不同子种群中的个体根据所处环境进化，可以产生不同特征的个体。

由于分批调度问题可以被分解为两个子问题，即分批子问题和调度子问题，因此本章打算让第一个子鸟群（G1）侧重解决分批子问题，朝着寻找最优分批方案的方向去进化，同时让第二个子鸟群（G2）侧重解决调度子问题，朝着寻找最优调度方案的方向去进化。具体来说，就是让G1使用针对分批矩阵（S1）的邻域搜索算子，让个体在分批搜索域上得到充分的搜索，然后让G2使用针对调度矩阵（S2）的邻域搜索算子，让个体在调度顺序搜索域上得到充分的搜索。

两个子问题并不是完全独立的，它们之间有时存在复杂耦合。例如，针对原来的分批方案有一个最优的调度方案，当分批方案发生变化，原来最优的调度方案对于新的分批方案来说可能已经不是最优的了。因此还需要把这两个子问题当成一个整体来优化。本章打算让第三个子鸟群（G3）同时解决两个子问题，同时朝着最优分批方案和最优调度方案去进化。具体来说，就是让G3使用同时针对S1和S2的邻域搜索算子，使每次邻域搜索都能在分批搜索域和调度搜索域上同时搜索。

三个子鸟群具有不同的功能，分化出不同的搜索环境和特点，适用于不同类型的个体，如下图所示。有些个体只需要改变分批方案，就能得到进化，有些个体只需要改变调度顺序，就能得到进化，而有些个体需要同时改变分批方案和调度顺序才能得到进化。特别是在尝试跳出局部最优的阶段，这三种不同的功能可以给不同的个体提供多样的尝试，帮助跳出局部最优。



三个功能分化的子鸟群不是孤立种群，它们通过一定的拓扑结构和种群迁移策略组织成一个有机的整体，从而实现子种群之间的协同。有了合适的拓扑结构和种群迁移，三个种群的分化功能才能得到最大的发挥。拓扑结构就是子种群连接的方式，ML-CCMBO使用的是环状的拓扑，三个子鸟群分布在一个环上，互相连接，每一个子鸟群都与另外两个子鸟群有连接。种群迁移策略是各个子种群进行信息交流的方式。当某个子种群陷入局部最优了，适当的种群迁移可以引入局部最优以外的个体，刺激子种群跳出局部最优。当某个个体在它的子种群内得不到进化，迁移到另外一个子种群可以给它一个新的进化环境，促进它继续进化。经典的多种群协同进化算法会在一个子种群中选择某些个体去替换另一个子种群的部分个体。如图所示，种群迁移之后，子鸟群引入了不同特性的个体，提高了子种群内的多样性，促进子种群的进化，而迁移到其他子种群的个体则在新的环境里尝试不同方向的进化。考虑到直接替换部分个体会导致部分个体的消失和部分个体的重复出现，会加剧多样性的损失，因此ML-CCMBO不使用替换的方法，而是使用交换的方法。

具体来说，ML-CCMBO的子鸟群会选出最优的个个体，与另一个子鸟群的任意个个体交换。这样能保证每个个体都有迁移到其他子鸟群的机会，同时也能让最优个体停滞不前的时候有更多机会在不同环境的刺激下继续进化。ML-CCMBO使用同步的多种群进化模式，即每个子鸟群独立进化代之后就进行种群迁移。ML-CCMBO子鸟群迁移的步骤如下：

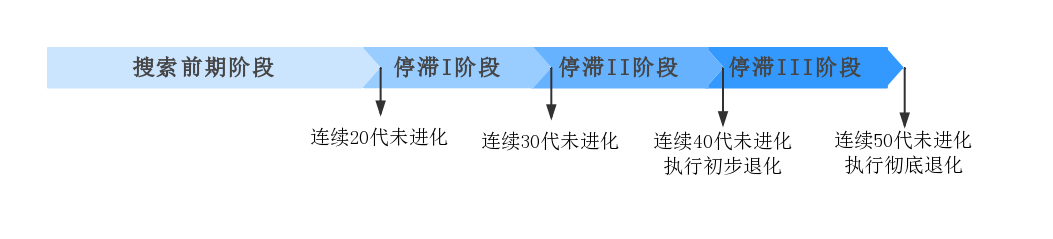
step1：分别选出三个子种群G1，G2，G3最优的个个体，分别是G1b，G2b，G3b；

step2：分别从三个子种群G1，G2，G3中未被选择的个体中随机选择个个体，分别是G1r，G2r，G3r；

step3：令G1b与G2r交换，令G2b与G3r交换，令G3b与G1r交换。

## 6.2.2 阶段性邻域搜索策略

CCMBO算法虽然使用了不同的邻域算子，但是只使用了一种邻域搜索策略。为了适应不同阶段个体的特点，ML-CCMBO使用了4.3.3.4中提到的其他搜索策略，提出了阶段性邻域搜索，可以提高算法在复杂搜索域里面的搜索能力。在个体进行邻域搜索之前，要判断个体处于什么阶段，针对不同阶段的个体使用不同强度的邻域搜索策略。在停滞阶段后期，执行不同程度的退化，帮助个体跳出局部最优。各个阶段的划分如下图所示。



当个体20代以内无进化时，认为这是搜索前期阶段，搜索策略与CCMBO相同，都使用1G-1S的简单策略，有活力的个体一般在20代以内就能得到再一次的进化，因此处于这个阶段的不需要特殊处理。

如果个体连续20代以上没有进化，则进入“停滞I”阶段，被视为有停滞的倾向，需要加强邻域搜索的强度。此时使用2G-1S邻域搜索策略，即每次做两次邻域搜索，按照贪心的法则选取较优者作为邻域搜索的结果。

当个体连续30代以上没有进化，则进入“停滞II”阶段，此时处于更高的停滞水平，此阶段的个体更难得到进化。除了继续保持搜索的强度以外，还需提高邻域搜索的深度。此时使用2G-2S邻域搜索策略，尝试使用更大的步幅做更深入的搜索，促进停滞的个体继续进化。

当个体连续40代以上没有进化，则进入“停滞III”阶段。到达此阶段的个体在经历了“停滞I”和“停滞II”之后还未能进化，已经到达很严重的停滞了，个体似乎遇到了进化的天花板。因此可以在第40代无进化的时候，尝试让个体退几步，进行初步的退化。即退化到某个比当前差的状态，以尝试跳出局部最优。在本阶段同样使用2G-1S邻域搜索策略，保持搜索的强度。大多数个体在本阶段都能得到进化了。

当个体在第50代都无进化时，可以认为该个体彻底停滞，只有少数个体能达到这样的状态。这样的个体并不具备进化的能力，因此可以进行彻底的退化。即重新初始化该个体，彻底逃离局部最优的位置，重新开始搜索，回到搜索前期阶段。

## 6.2.3 算法流程及复杂度分析

ML-CCMBO算法的每个子鸟群会并行运行具有不同邻域针对性的CCMBO算法，同时对每个个体都执行阶段性的邻域搜索策略，三个子鸟群朝着不同的方向进化，经过竞争阶段、改进的V字飞行阶段、减速调整阶段，完成一轮飞行之后，就进行一次种群迁移，迁移结束后进行下一轮的飞行。算法的流程图如下所示：



下面分析ML-CCMBO算法的时间复杂度。令子鸟群大小为，染色体维度为，在最坏的情况下，ML-CCMBO的单个子鸟群寻优的时间复杂度为：

（1）初始化阶段，对个个体做随机初始化，时间复杂度是

（2）竞争阶段，个个体进行排序，选出领头鸟，构建新的V字型队伍，本阶段时间复杂度是；

（3）使用阶段性邻域搜索策略后，改进的V字飞行阶段，在最坏情况下，使用最复杂的邻域策略，个个体通过交叉或者邻域搜索的方法各自生成个候选解，如此更新代，本阶段的时间复杂度是；

（4）使用阶段性邻域搜索策略后，减速调整阶段，在最坏情况下，使用最复杂的邻域策略，个个体通过邻域搜索各自生成个候选解，如此更新代，本阶段的时间复杂度是。

（5）种群迁移阶段，需要选出个个体中最优的个体去迁移，排序所需要的时间复杂度是。

综合以上的分析，和代表着优化问题的规模，、和是与和无关的常量，因此对任何规模的优化问题来说，ML-CCMBO的三个子鸟群寻优的时间复杂度为：



# 6.3 实例仿真以及性能评价

为了验证ML-CCMBO的效果，本节把ML-CCMBO应用到标准LS-FJSP问题和全自动免疫检验设备的实际问题中，测试用例使用5.3中的P1~P8。首先验证6.2阐述的两种改进的有效性，然后验证多种群思想的效果，探讨不同算法多种群化的效果。

## 6.3.1 验证改进的效果

本小节针对6.2节所提出的两个改进进行效果验证。由于ML-CCMBO是在CCMBO的基础上进行多种群化，再进行其他改进得来的，所以为了公平地对比6.2节的改进，需要对CCMBO算法进行简单的多种群化，得到PCCMBO，然后再与引入改进的算法进行对比。PCCMBO由三个CCMBO子种群构成， 子种群迁移使用最简单的随机选择的方法，除此之外并无任何改进。引入多领头鸟分化协同机制的PCCMBO记为PCCMBO+diff，引入阶段性邻域搜索策略的PCCMBO记为PCCMBO+stage。算法参数取，，，，，。表是PCCMBO、PCCMBO+diff、PCCMBO+stage、ML-CCMBO对P1~P8经过30次优化后得到的结果。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **problem No.** | | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
| **PCCMBO** | **Avg** | 84.53 | 186.27 | 194.93 | 375.40 | 125.53 | 217.70 | 191.53 | 339.57 |
| **Std** | **0.85** | 2.29 | 1.18 | 2.29 | 0.53 | 1.92 | 2.72 | 1.56 |
| **Best** | 83 | 183 | 192 | 371 | 125 | 215 | 188 | 337 |
| **Worst** | 85 | 190 | 197 | 379 | 126 | 224 | 196 | 343 |
| **BKN** | 7 | 7 | 1 | 3 | 14 | 2 | 6 | 5 |
| **PCCMBO +diff** | **Avg** | 84.47 | 185.23 | 194.40 | 375.23 | 125.43 | 218.23 | 190.53 | 339.10 |
| **Std** | 0.88 | 1.76 | 1.20 | 1.84 | 0.51 | 2.20 | 2.50 | 1.80 |
| **Best** | 83 | 183 | 192 | 371 | 125 | 215 | 188 | 337 |
| **Worst** | 85 | 188 | 197 | 378 | 126 | 222 | 196 | 342 |
| **BKN** | 8 | 7 | 3 | 3 | 17 | 6 | 11 | 11 |
| **PCCMBO +stage** | **Avg** | 84.20 | 185.03 | 193.83 | 373.43 | 125.43 | 217.30 | 188.87 | 338.77 |
| **Std** | 0.98 | 2.23 | **1.00** | 1.84 | 0.50 | 2.02 | 1.02 | 1.50 |
| **Best** | 83 | 183 | 192 | 371 | 125 | 215 | 188 | 337 |
| **Worst** | 85 | 190 | **195** | **376** | 126 | 220 | **191** | 341 |
| **BKN** | 12 | 10 | 4 | 5 | 17 | **11** | 16 | 12 |
| **ML-CCMBO** | **Avg** | **84.00** | **184.37** | **193.57** | **373.10** | **125.33** | **216.23** | **188.67** | **338.20** |
| **Std** | 1.00 | **1.40** | 1.02 | **1.64** | **0.47** | **1.33** | **0.98** | **1.17** |
| **Best** | 83 | 183 | 192 | 371 | 125 | 215 | 188 | 337 |
| **Worst** | 85 | **187** | **195** | **376** | 126 | **219** | **191** | **340** |
| **BKN** | **15** | **13** | **7** | **7** | **20** | **11** | **19** | **14** |

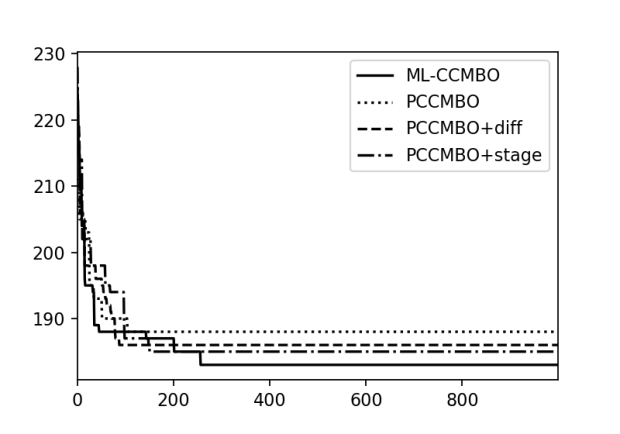
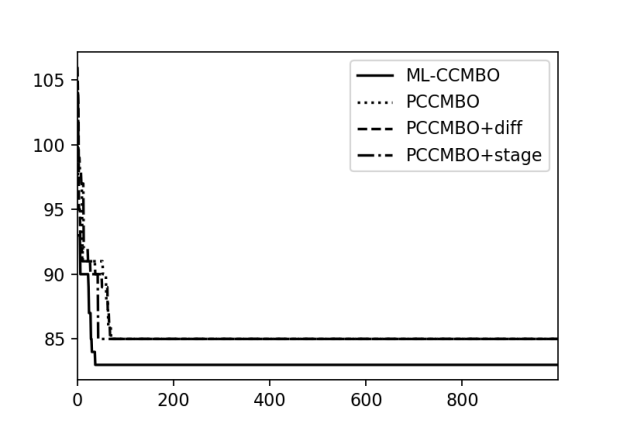
首先观察PCCMBO+diff、PCCMBO+stage与PCCMBO的区别。根据表，PCCMBO+diff在8个算例的Avg都稍微小于PCCMBO，对P2、P4来说，其BKN值与PCCMBO的持平，对其他算例来说，其BKN值都有不同程度的提高，说明子种群功能分化之后，算法的寻优精度提升了一些。PCCMBO+stage在8个算例的Avg也都小于PCCMBO，而且在30次求解过程中得到的最差解比PCCMBO的最差解更优，说明使用阶段性邻域搜索策略之后，算法在不同停滞阶段执行不同的邻域搜索，使得算法在中后期跳出局部最优的能力更强了，因此找到的最差的解更好了。除了P1和P6以外，PCCMBO+stage求解的Std都比PCCMBO的小，说明该阶段性邻域搜索策略还有效提高了算法的稳定性。

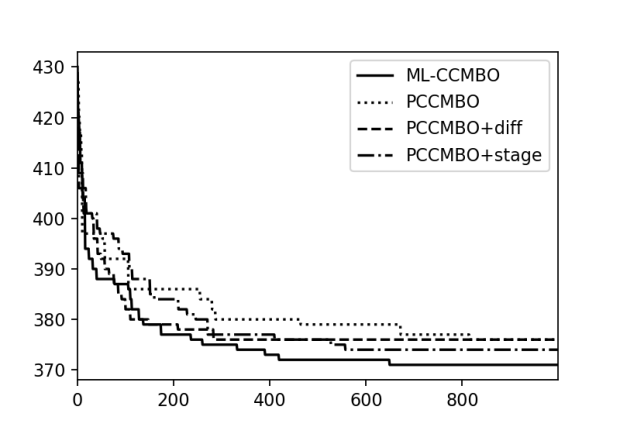
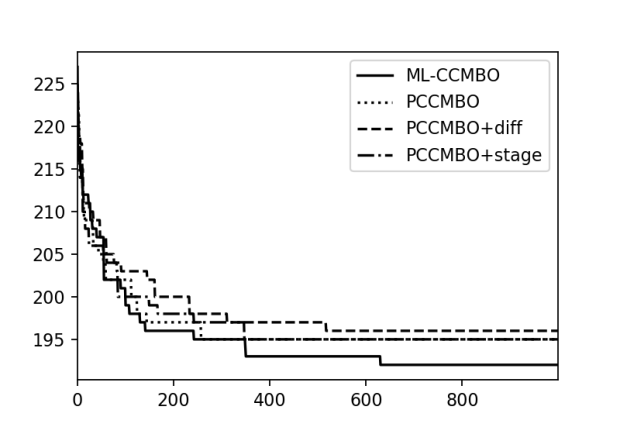
然后观察ML-CCMBO与PCCMBO的区别。根据表，从AVG来看，ML-CCMBO对P1~P8的平均优化精度明显比PCCMBO好。从Std来看，除了P1与P3以外，ML-CCMBO的求解稳定性都比PCCMBO都有显著的降低，而且PCCMBO对P1求解是稳定地求出较差的解，而不是求出较好的解。从Worst来看，除了规模最小的P1和P5以外，30次求解的最差解都比PCCMBO的更优，即ML-CCMBO求到较差解的几率更小。从BKN来看，PCCMBO在30run中求到已知最优解的次数比PCCMBO显著提高了很多，说明在两个改进的共同作用下，算法寻优的能力得到了大大的提高。

接下来计算这四种算法的Df值，以评价它们的综合性能，详细数据见表，Df平均值如图所示。其中PCCMBO+diff和PCCMBO+stage的平均Df值都在不同程度上小于PCCMBO。特别是PCCMBO+stage，对每个算例的Df值都小于PCCMBO，验证了阶段性邻域搜索策略的有效性。而ML-CCMBO无论是针对每个算例的Df值，还是Df平均值，都远远小于PCCMBO，说明同时引入分化协同机制和阶段性邻域搜索之后，算法解出最优解的能力大大提升了。

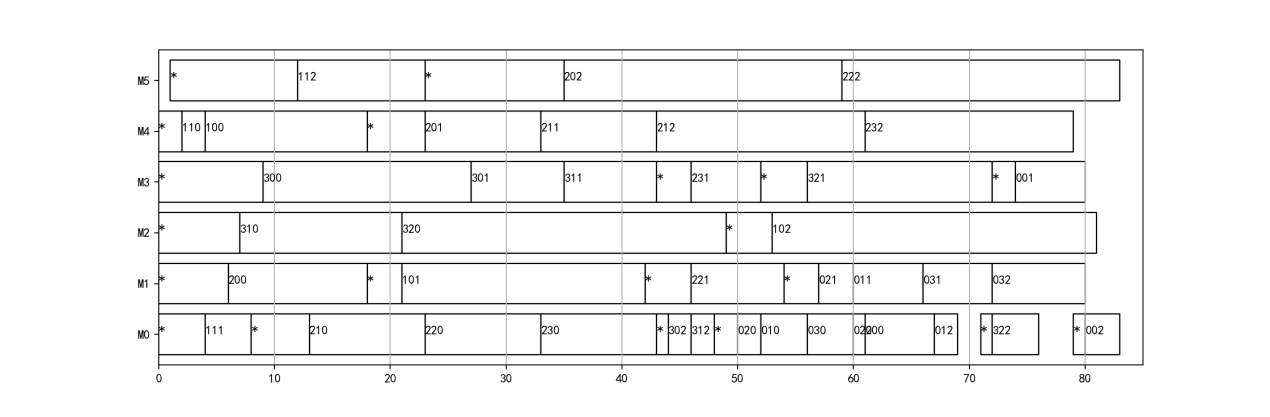
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **problem No.** | **PCCMBO** | **PCCMBO+diff** | **PCCMBO+stage** | **ML-CCMBO** |
| **P1** | 0.84% | 0.80% | 0.66% | 0.55% |
| **P2** | 1.79% | 1.22% | 1.11% | 0.75% |
| **P3** | 1.53% | 1.25% | 0.95% | 0.82% |
| **P4** | 1.19% | 1.14% | 0.66% | 0.57% |
| **P5** | 0.43% | 0.35% | 0.35% | 0.27% |
| **P6** | 1.26% | 1.50% | 1.07% | 0.57% |
| **P7** | 1.88% | 1.35% | 0.46% | 0.35% |
| **P8** | 0.76% | 0.62% | 0.52% | 0.36% |
| **Avg** | 1.21% | 1.03% | 0.72% | **0.53%** |

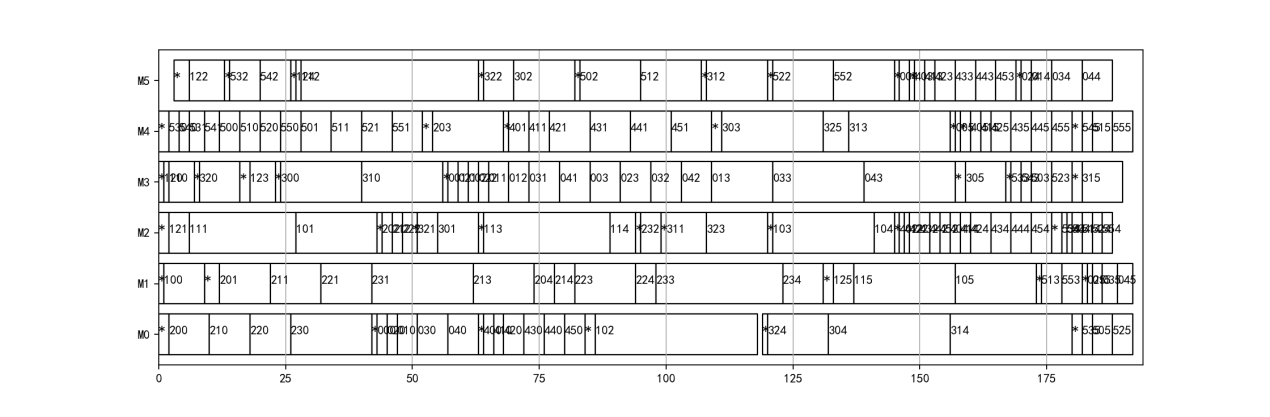
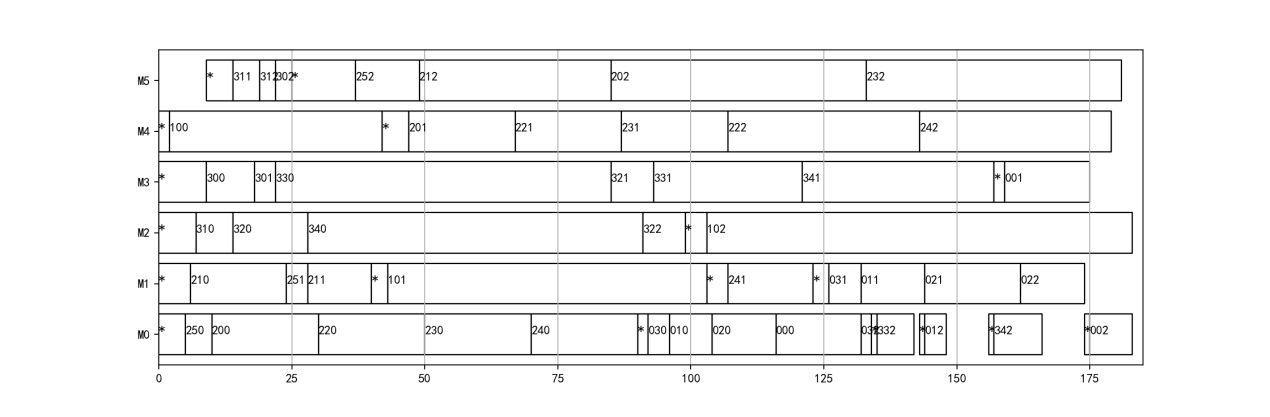
最后从根据收敛曲线图来验证改进的效果，图展示了PCCMBO、PCCMBO+diff、PCCMBO+stage、ML-CCMBO对P1~P4单次求解的收敛图，其中实线表示本章所提出的ML-CCMBO算法。对于P2来说，PCCMBO+diff和PCCMBO+stage的收敛速度比PCCMBO稍慢了一些，但是可以发现，PCCMBO只是收敛得比较早，有早熟收敛的倾向，而PCCMBO+diff和PCCMBO+stage具有一定跳出局部最优的能力。对于P1~P4来说，ML-CCMBO的收敛都是最快，而且收敛精度最高的。

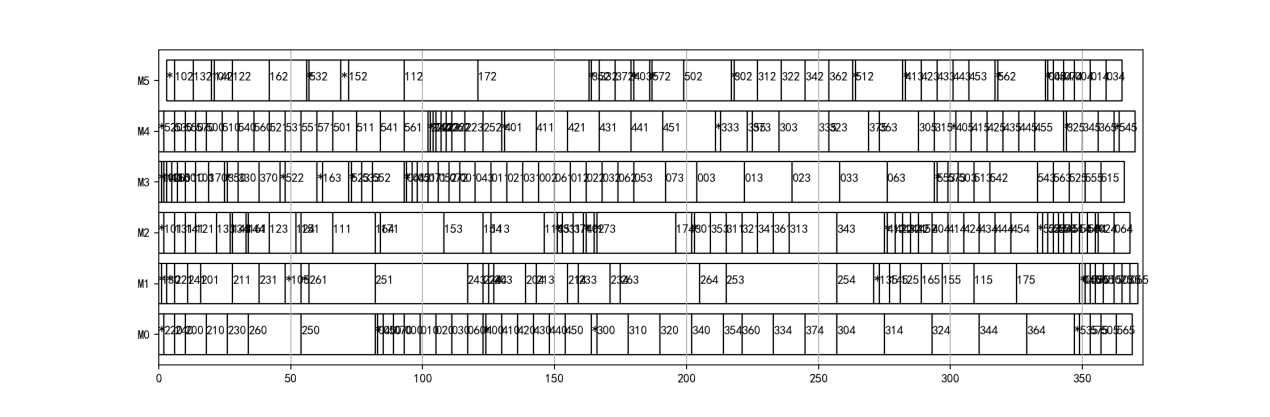


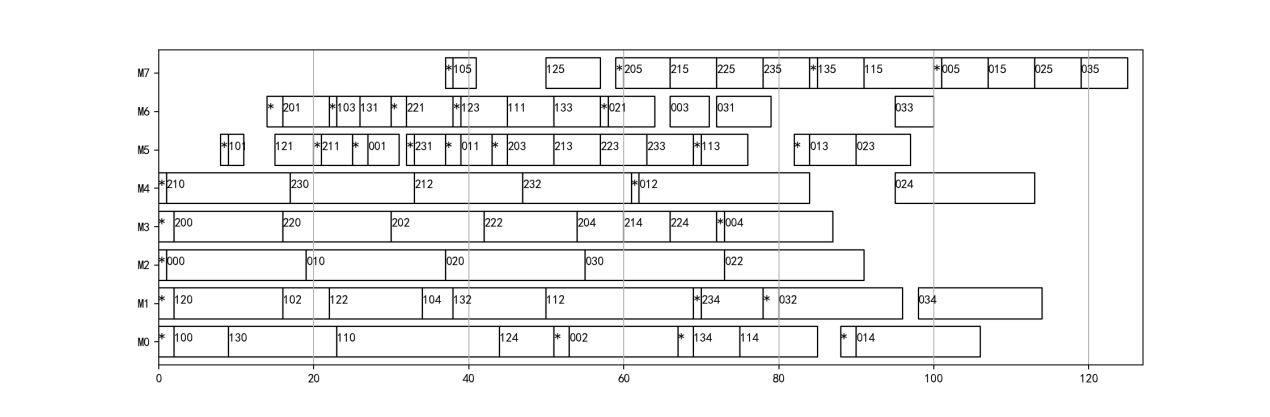


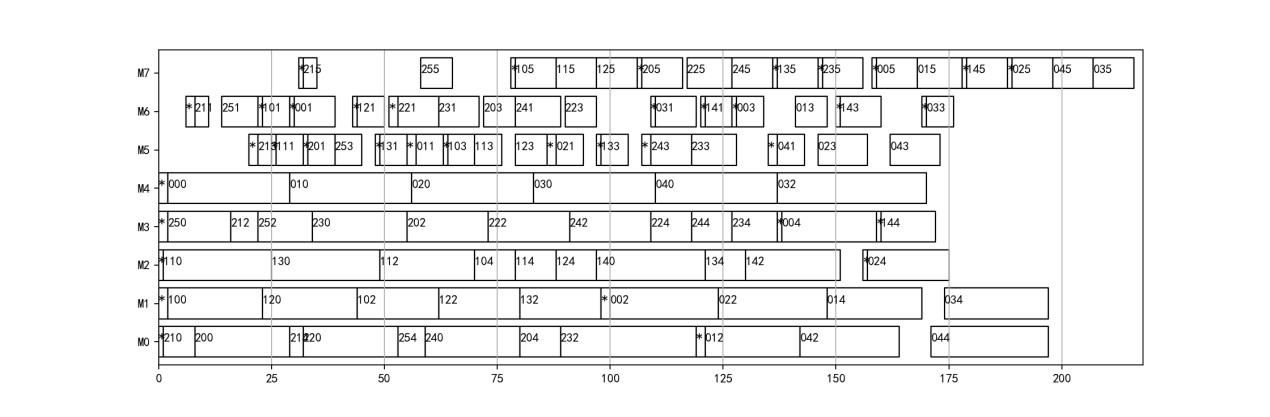
图展示了算例使用ML-CCMBO对P1~P8求解得到的分批调度方案的甘特图。其中标有“\*”的方框代表准备操作，其他方框上都标有三个数字，分别代表批次序号、子批序号、工序序号。甘特图的横坐标代表时间，纵坐标代表不同的机器。

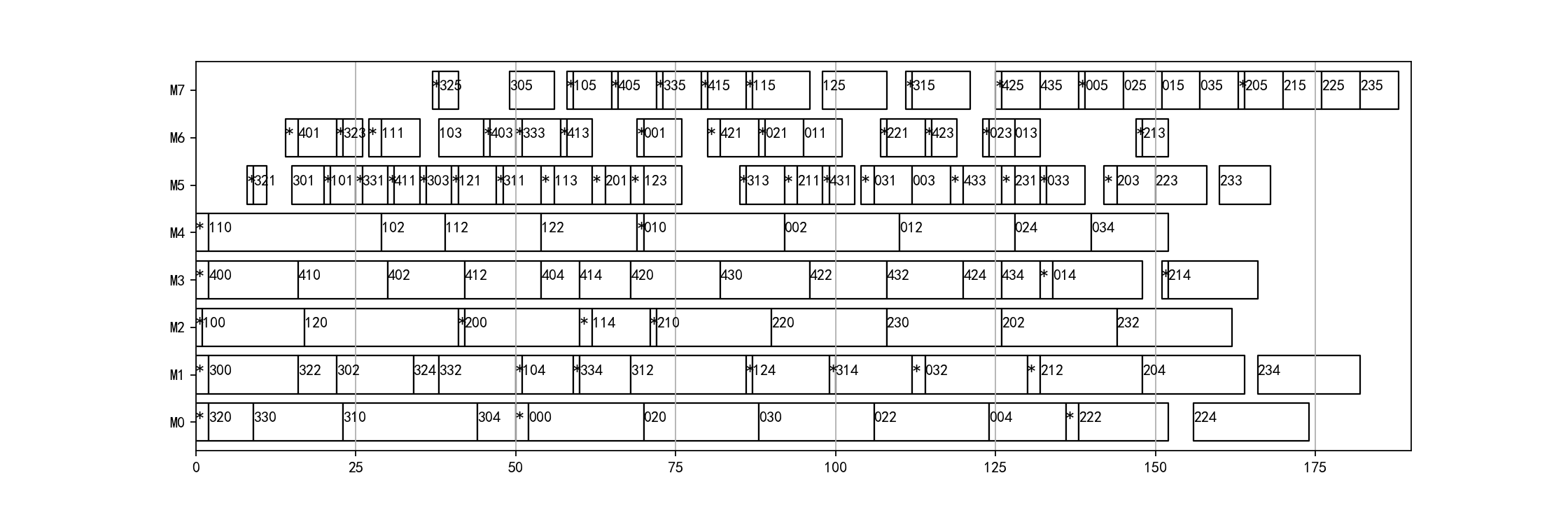


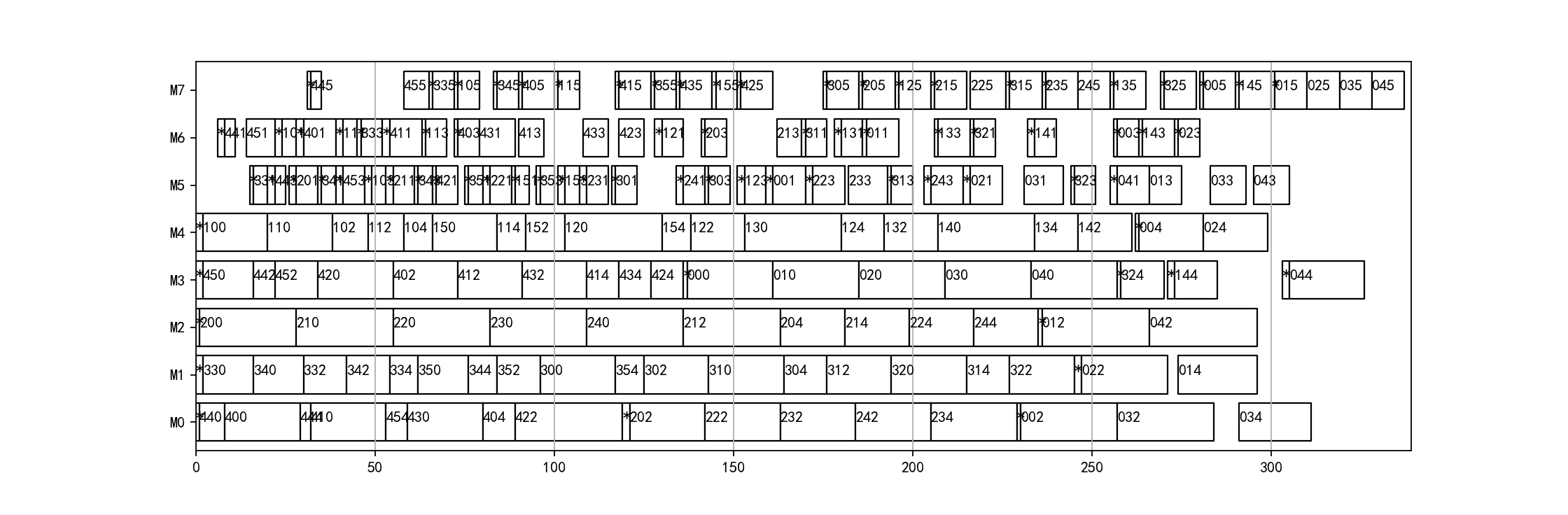












## 6.3.2 分析多种群的效果

本章提出的ML-CCMBO算法是一种基于多种群化思想的算法，但是ML-CCMBO并不是简单的多种群化版本，而是根据LS-FJSP问题特征设计了子种群进化及交互的方式，使多个种群组成一个有机体。本小节主要分析对算法进行不同形式的多种群化对求解LS-FJSP有什么效果，主要包括两组对比。

首先对MBO、MBO的简单的多种群化算法PMBO、本章提出的改进多种群算法ML-CCMBO进行对比。为了公平起见，三个算法所使用的种群个体总数都为51，而且都使用了相同的迭代次数。对算例P1~P8求解30次的结果如表所示。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **problem No.** | | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
| **MBO** | **Avg** | 86.57 | 190.23 | 201.27 | 386.97 | 126.37 | 219.30 | 195.13 | 342.37 |
| **Std** | 2.03 | 4.10 | 2.63 | 4.07 | 0.60 | 1.81 | 2.29 | 1.83 |
| **Best** | 83 | 185 | 198 | 378 | 125 | 217 | 189 | 340 |
| **Worst** | 91 | 200 | 207 | 392 | 127 | 222 | 198 | 345 |
| **BKN** | 4 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| **PMBO** | **Avg** | 85.87 | 188.70 | 199.57 | 382.30 | 125.90 | 219.37 | 193.27 | 340.27 |
| **Std** | 2.05 | 2.73 | 3.15 | 4.82 | 0.65 | 2.17 | 2.53 | 1.79 |
| **Best** | 83 | 184 | 195 | 376 | 125 | 216 | 188 | **337** |
| **Worst** | 91 | 194 | 207 | 396 | 127 | 224 | 196 | 343 |
| **BKN** | 4 | 0 | 0 | 0 | 8 | 0 | 3 | 3 |
| **ML-CCMBO** | **Avg** | **84.00** | **184.37** | **193.57** | **373.10** | **125.33** | **216.23** | **188.67** | **338.20** |
| **Std** | **1.00** | **1.40** | **1.02** | **1.64** | **0.47** | **1.33** | **0.98** | **1.17** |
| **Best** | 83 | **183** | **192** | **371** | 125 | **215** | 188 | **337** |
| **Worst** | **85** | **187** | **195** | **376** | **126** | **219** | **191** | **340** |
| **BKN** | **15** | **13** | **7** | **7** | **20** | **11** | **19** | **14** |

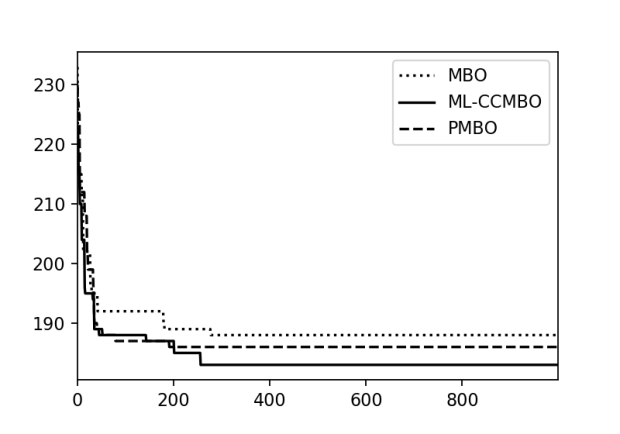
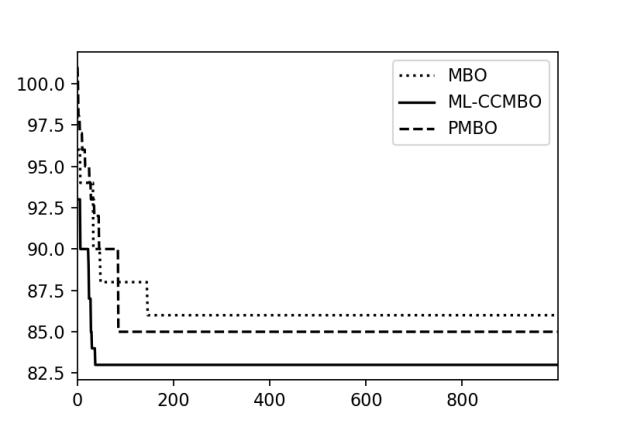
PMBO是MBO的简单多种群化，没有设计特殊的子种群进化算子，也没有设计特殊的子种群迁移方式，与MBO的区别是能保持更高的种群多样性，因此就Avg来看，PMBO求解结果相比于MBO的结果仅有较少的提升。而且对于P1、P3~P7来说，其Std反而比MBO更高了，意味着求解稳定性更差了。对于MBO无法求得最优解的P2~P4、P6，PMBO依然也不能求出最优解。据此得出，虽然简单多种群化可以稍微提高算法的寻优精度，但是提高比较不明显，对于一些实例依然无法求出最优解，反而损失了一定的稳定性。

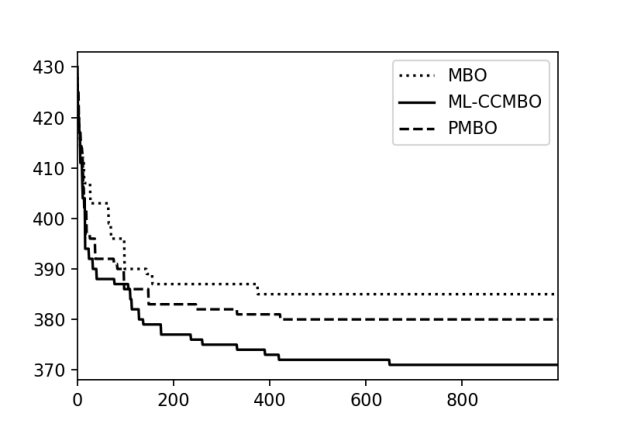
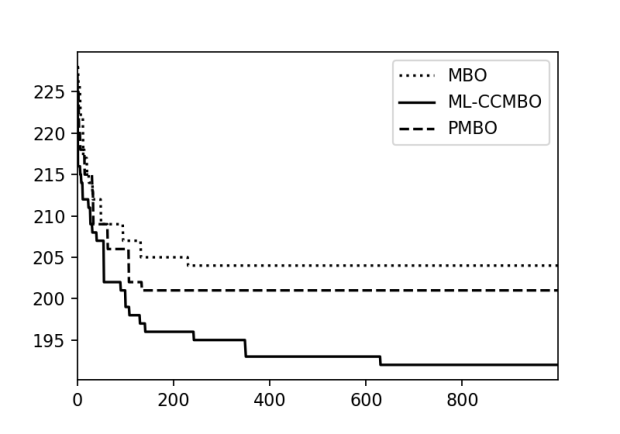
ML-CCMBO不是简单的多种群化，而是根据LS-FJSP的问题特点，引入了有针对性的子种群分化协同机制，为不同子种群设计了不同的进化算子，为子种群迁移设计了有效的迁移方式，因此其优化结果无论是从Avg、Std还是BKS方面看，相对MBO都有很明显的提升，说明算法的寻优精度、稳定性、求出最优解的能力都同时变强了。这说明，经过有针对性的设计之后，多个种群形成一个有机体，是可以在很大程度上提高求解LS-FJSP的能力的。

表和图展示了这三个算法求解P1~P8的Df值及其平均值。可以发现，无论是针对单个算例的Df值还是Df平均值，ML-CCMBO的该指标都远比MBO和PMBO的小，说明ML-CCMBO求出最优解的能力远比MBO和PMBO强。

|  |  |  |  |
| --- | --- | --- | --- |
| **problem No.** | **MBO** | **PMBO** | **ML-CCMBO** |
| **P1** | 1.95% | 1.57% | 0.55% |
| **P2** | 3.95% | 3.11% | 0.75% |
| **P3** | 4.83% | 3.94% | 0.82% |
| **P4** | 4.30% | 3.05% | 0.57% |
| **P5** | 1.09% | 0.72% | 0.27% |
| **P6** | 2.00% | 2.03% | 0.57% |
| **P7** | 3.79% | 2.80% | 0.35% |
| **P8** | 1.59% | 0.97% | 0.36% |
| **Avg** | 2.94% | 2.27% | **0.53%** |

图展示了MBO、PMBO、ML-CCMBO对P1~P4某次求解的收敛过程，其中实线表示ML-CCMBO算法。可以发现，PMBO的收敛速度和收敛精度比MBO好一些，但还是陷入了局部最优，求解能力有限。而ML-CCMBO的前期收敛速度都远远比MBO和PMBO的好，在收敛后期的时候，当其他算法陷入局部最优而无法继续优化的时候，它还有优化的活力，并逐步收敛到已知的最优解，这证明了ML-CCMBO经过精心的多种群化设计后，达到了较为满意的优化效果。

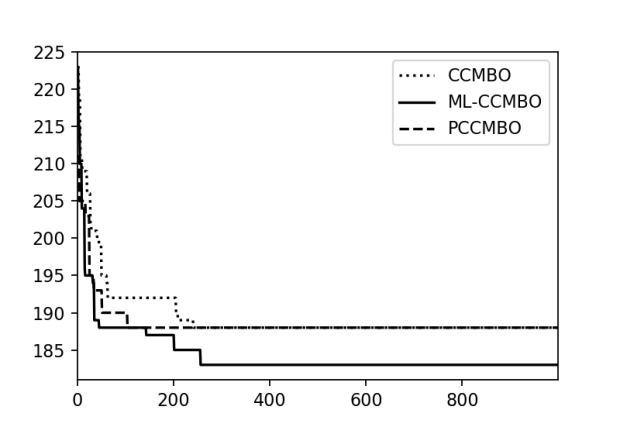
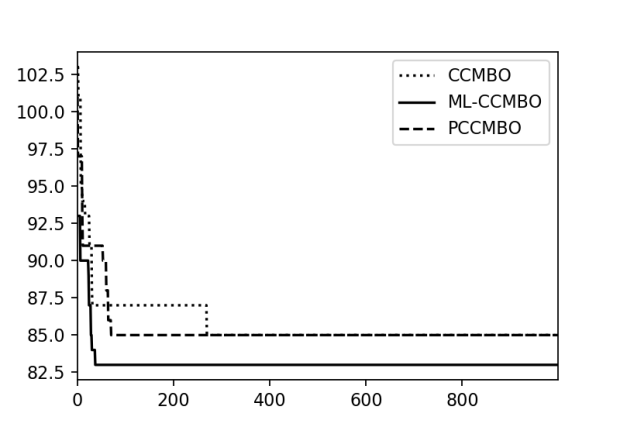


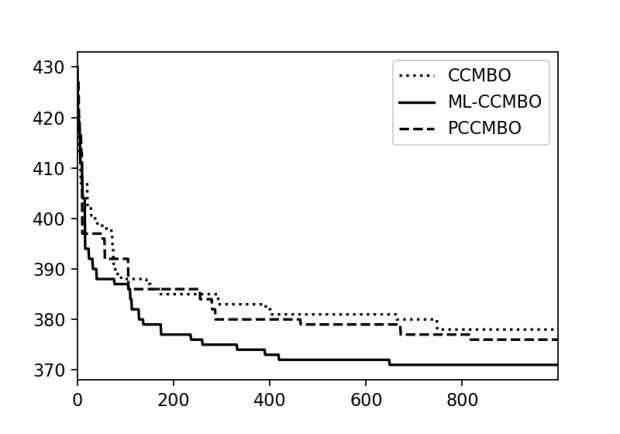
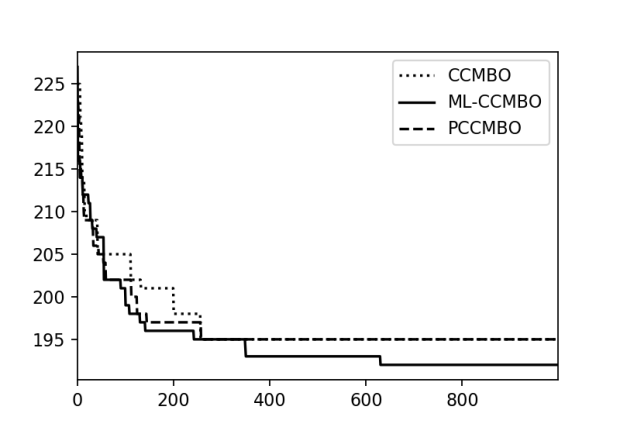


然后对CCMBO、CCMBO的简单的多种群化算法PCCMBO、本章提出的改进多种群算法ML-CCMBO进行对比。为了公平起见，三个算法所使用的种群个体总数都为51，而且都使用了相同的迭代次数。对算例P1~P8求解30次的结果如表所示。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **problem No.** | | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
| **CCMBO** | **Avg** | 85.13 | 188.10 | 197.87 | 380.73 | 125.80 | 217.90 | 192.07 | 340.10 |
| **Std** | 1.54 | 3.03 | 1.33 | 3.51 | 0.65 | 2.15 | 2.79 | 1.74 |
| **Best** | 83 | 183 | 196 | 374 | 125 | 215 | 188 | 337 |
| **Worst** | 88 | 195 | 200 | 386 | 127 | 224 | 196 | 343 |
| **BKN** | 7 | 4 | 0 | 0 | 10 | 1 | 5 | 4 |
| **PCCMBO** | **Avg** | 84.53 | 186.27 | 194.93 | 375.40 | 125.53 | 217.70 | 191.53 | 339.57 |
| **Std** | 0.85 | 2.29 | 1.18 | 2.29 | 0.50 | 1.72 | 2.72 | 1.50 |
| **Best** | 83 | 183 | 192 | 371 | 125 | 215 | 188 | 337 |
| **Worst** | 85 | 190 | 197 | 379 | 126 | 224 | 196 | 343 |
| **BKN** | 7 | 7 | 1 | 3 | 14 | 2 | 6 | 5 |
| **ML-CCMBO** | **Avg** | 84.00 | 184.37 | 193.57 | 373.10 | 125.33 | 216.23 | 188.67 | 338.20 |
| **Std** | 1.00 | 1.40 | 1.02 | 1.64 | 0.47 | 1.33 | 0.98 | 1.17 |
| **Best** | 83 | 183 | 192 | 371 | 125 | 215 | 188 | 337 |
| **Worst** | 85 | 187 | 195 | 376 | 126 | 219 | 191 | 340 |
| **BKN** | 15 | 13 | 7 | 7 | 20 | 11 | 19 | 14 |

|  |  |  |  |
| --- | --- | --- | --- |
| **problem No.** | **CCMBO** | **PCCMBO** | **ML-CCMBO** |
| **P1** | 1.17% | 0.84% | 0.55% |
| **P2** | 2.79% | 1.79% | 0.75% |
| **P3** | 3.06% | 1.53% | 0.82% |
| **P4** | 2.62% | 1.19% | 0.57% |
| **P5** | 0.64% | 0.43% | 0.27% |
| **P6** | 1.35% | 1.26% | 0.57% |
| **P7** | 2.16% | 1.88% | 0.35% |
| **P8** | 0.92% | 0.76% | 0.36% |
| **Avg** | 1.84% | 1.21% | 0.53% |





七、基于多微鸟群协同候鸟迁移算法的多目标分批优化调度

参考文献

[1] 许金钩, 王尊本. 荧光分析法[M]. 科学出版社, 2006.

[2] 程绍钧, 余裕民. 检验核医学[M]. 重庆大学出版社, 2001.

[3] 贺淹才. 基因工程概论[M]. 清华大学出版社, 2008.

[4] 李延斌, 孙志华. 全自动生化分析仪的分类、结构和选购[J]. 中国医疗器械信息, 2008, 14(3): 34–40.

[5] 王炜. 全自动生化分析仪关键技术进展[J]. 中国医疗器械信息, 2010, 16(12): 18–19.

[6] 刘志辉, 葛秀君, 朱海龙. 进板顺序及洗涤模式对FAME全自动酶免分析系统检测乙型肝炎标志物效率的影响[J]. 现代医药卫生, 2010, 26(1): 51–52.

[7] 谷成祥. 全自动酶免疫分析系统检测乙型肝炎标志物的工作模式的优化研究[J]. 检验医学, 2006, 21(1): 78–81.

[8] 杨勇毅, 王军, 罗蓉. 全自动酶免分析系统工作流程的优化[J]. 中国输血杂志, 2007, 20(2): 133–135.

[9] 张晶, 祝连庆, 王君等. 一种全自动酶免分析仪多任务调度方法[J]. 计算机仿真, 2014, 31(6): 4–8.

[10] 姚勇, 相恒泉, 石圆圆. 全自动酶免分析系统与手工操作方法的应用比较[J]. 临床血液学杂志, 2012, 25(8): 482–487.

[11] MACCARTHY B L, LIU J Y. Addressing the Gap in Scheduling Research - a Review of Optimization and Heuristic Methods in Production Scheduling[J]. International Journal of Production Research, 1993, 31(1): 59–79.

[12] Reiter S. A system for managing job-shop production[J]. The Journal of Business, 1966, 39(3): 371–393.

[13] Low C, Hsu C M, Huang K I. Benefits of lot splitting in job-shop scheduling[J]. International Journal of Advanced Manufacturing Technology, 2004, 24(9–10): 773–780.

[14] Chen J, Steiner G. On discrete lot streaming in no-wait flow shops[J]. IIE Transactions, 2003, 35(2): 91–101.

[15] Kalir A A, Sarin S C. Optimal Solutions for the Single Batch, Flow Shop, Lot-streaming Problem with Equal Sublots[J]. Decision Sciences, 2007, 32(2): 387–398.

[16] Liu J. Single-job lot streaming in m-1 two-stage hybrid flowshops[J]. European Journal of Operational Research, 2008, 187(3): 1171–1183.

[17] Kim Y D, Shim S O, Kim S B等. Parallel machine scheduling considering a job-splitting property[J]. International Journal of Production Research, 2004, 42(21): 4531–4546.

[18] Nait Tahar D, Yalaoui F, Chu C等. A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times[J]. International Journal of Production Economics, 2006, 99(1–2): 63–73.

[19] Cheng M, Mukherjee N J, Sarin S C. A review of lot streaming[J]. International Journal of Production Research, Taylor & Francis, 2013, 51(23–24): 7023–7046.

[20] Lei D, Guo X. Scheduling job shop with lot streaming and transportation through a modified artificial bee colony[J]. International Journal of Production Research, 2013, 51(16): 4930–4941.

[21] Defersha F M, Chen M. A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming[J]. Proceedings - 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, 2009, 1: 201–208.

[22] Defersha F M, Bayat Movahed S. Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming[J]. Computers and Industrial Engineering, Elsevier, 2018, 117: 319–335.

[23] Dauzère-Pérès S, Lasserre J-B. Lot Streaming in Job-Shop Scheduling[J]. Operations Research, 2008, 45(4): 584–595.

[24] Wagner H M. An integer programming model for machine scheduling[J]. Naval Research Logistics Quarterly, 1959, 6: 131–140.

[25] Held M, Karp R M. An dynamic programming approach to sequencing problems[J]. Journal ofSIAM, 1962, 10: 196–210.

[26] Ignall E, Schrage L. Application of the branch and bound technique to some flow-shop problems[J]. Operations Research, 1965, 13: 400–412.

[27] Palmer D S. Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining a near optimum[J]. Operational Research Quarterly, 1965, 10: 101–107.

[28] J.K.Lenstra, A.H.G.Rinnooy, KanP.Brucker. Complexity of machine scheduling problems[J]. Annals of Discrete Mathematics, 1977, 1: 343–362.

[29] Li J-Q, Pan Q-K, Suganthan P N等. A Hybrid Tabu Search Algorithm With an Efficient Neighborhood Structure for the Flexible Job Shop Scheduling Problem[J]. The International Journal of Advanced Manufacturing Technology, 2011, 52(5–8): 683–697.

[30] van Laarhoven P J M, Aarts E H L, Lenstra J K. Job Shop Scheduling by Simulated Annealing[J]. Operations Research, 2008, 40(1): 113–125.

[31] Roce F, Tadei R C, Volta G. A genetic algorithm for the job shop problem[J]. Computers & Operations Research, 1995, 22(1): 15–24.

[32] Sha D Y, Hsu C-Y. A hybrid particle swarm optimization for job shop scheduling problem[J]. Computers & Industrial Engineering, 2006, 51(4): 791–808.

[33] C.N.Potts, K.R.Baker. Flow shop scheduling with lot streaming[J]. Operations Research Letters, 1989, 8(6): 297–303.

[34] Jian W, Hong Z. Integrated Optimization of Lot Streaming and Sublot-intermingling Scheduling for a Kind of Flow Shop[J]. Journal of System Simulation, 2008, 20(4): 1011–1015.

[35] Chan F T S, Wong T C, Chan L Y. The application of genetic algorithms to lot streaming in a job-shop scheduling problem[J]. International Journal of Production Research, 2009, 47(12): 3387–3412.

[36] Han Y, Li J, Sang H等. An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem[J]. Swarm and Evolutionary Computation, Elsevier B.V., 2018, 38: 64–78.

[37] Huang R H, Yu T H. An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting[J]. Applied Soft Computing, Elsevier B.V., 2017, 57: 642–656.

[38] 白俊杰，龚毅光，王宁生，唐敦兵. 多目标柔性作业车间分批优化调度[J]. 计算机集成制造系统, 2010, 16(2): 0–7.

[39] Duman E, Uysal M, Alkaya A F. Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem[J]. Information Sciences, Elsevier Inc., 2012, 217: 65–77.

[40] Demir Y, Işleyen S K. An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations[J]. International Journal of Production Research, 2014, 52(13): 3905–3921.

[41] Zhijun S, Jin A. 作业车间多工艺路线批量作业计划优化[J]. 中国机械工程, 2006, 19(2): 183–187.

[42] 王万良，范丽霞，徐新丽，赵燕伟，张静. 多目标差分进化算法求解柔性作业车间批量调度问题[J]. 2013, 19(10): 2481–2492.

[43] Yang X, Fan L, Li L等. Hybrid Discrete Differential Evolution Algorithm for Lot Splitting with Capacity Constraints in Flexible Job Scheduling[J]. Mathematical Problems in Engineering, 2013: 1–10.

[44] 王云，冯毅雄，谭建荣，高一聪. 柔性作业车间分批调度多目标优化方法[J]. 浙江大学学报（工学版）, 2011, 45(4): 719–764.

[45] Defersha F M, Chen M. Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time[J]. International Journal of Production Research, 2012, 50(8): 2331–2352.

[46] Wong T C, Ngan S C. A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop[J]. Applied Soft Computing Journal, Elsevier B.V., 2013, 13(3): 1391–1399.

[47] Davis L. Job Shop Scheduling with Genetic Algorithms[A]. international conference on genetic algorithms[C]. 1985: 136–140.

[48] Ponnambalam S G, Aravindan P, Sreenivasa Rao P. Comparative evaluation of genetic algorithms for job-shop scheduling[J]. Production Planning and Control, 2001, 12(6): 560–574.

[49] B Giffler G L T. Algorithms for solving production-scheduling problems[J]. Operations Research, 1959, 8(4): 487–503.

[50] Lei D. A Pareto archive particle swarm optimization for multi-objective job shop scheduling[J]. Computers and Industrial Engineering, 2008, 54(4): 960–971.

[51] ZHAO Y. New Hybrid Parallel Algorithm for Variable-sized Batch Splitting Scheduling with Alternative Machines in Job Shops[J]. Chinese Journal of Mechanical Engineering, 2010, 23(04): 484–495.

[52] Nathan A, Barbosa V C. V-like formations in flocks of artificial birds[J]. Artificial Life, 2008, 14(2): 179–188.

[53] Asadzadeh L. A parallel artificial bee colony algorithm for the job shop scheduling problem with a dynamic migration strategy[J]. Computers and Industrial Engineering, Elsevier Ltd, 2016, 102: 359–367.

[54] Kurdi M. An effective new island model genetic algorithm for job shop scheduling problem[J]. Computers & Operations Research, Elsevier, 2016, 67: 132–142.