****

硕士学位论文

|  |
| --- |
| 基于协同候鸟前以算法的全自动免疫分析 |
| 检测设备的分批优化调度问题研究 |

|  |  |
| --- | --- |
| 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

# 1.2 免疫分析检测设备调度问题的国内外研究现状

# 1.3 论文主要研究内容及其章节安排

二、免疫分析检测设备分批优化调度问题分析及建模

# 2.1 免疫分析检测设备相关知识

## 2.1.1 免疫检验基本原理

## 2.1.2 免疫检验设备基本组成

## 2.1.3 免疫检验流程

# 2.2 分批调度问题特点分析

## 2.2.1 问题描述

## 2.2.2 问题特点分析

# 2.3 柔性作业车间分批调度问题

## 2.3.1 柔性作业车间分批调度问题背景

## 2.3.2 柔性作业车间分批调度问题分类

# 2.4 免疫检测设备的柔性作业车间分批调度模型

# 2.5 柔性作业车间调度问题求解方法

三、候鸟迁移算法理论分析

# 3.1 基本候鸟迁移算法

候鸟迁移算法（Migrating Bird Optimization, MBO）是2012年【Duman】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。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。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。

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

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；

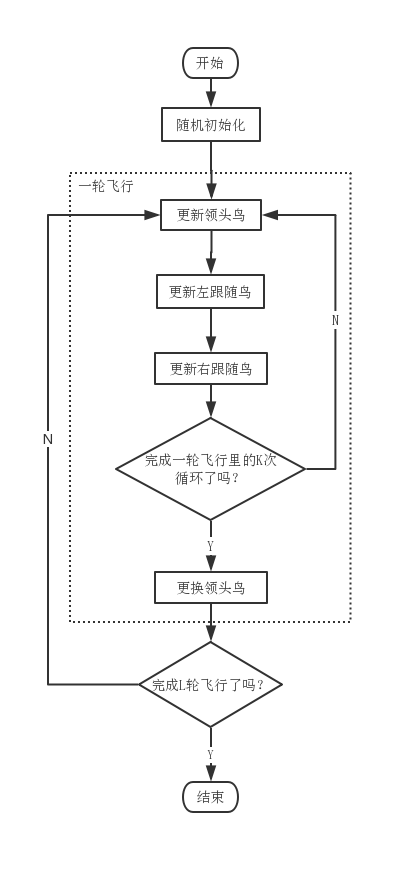
step3.5：令候选集，如果，则令，否则进行step5；

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：



# 3.2 候鸟迁移算法研究现状

# 3.3 候鸟迁移算法性能分析

## 3.3.1 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。

下面详细分析基本MBO算法中加剧多样性损失的一些地方。

3.2.1.1 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。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。

3.2.1.2 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。

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。

（考虑画图示意）

## 3.3.2 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。

3.3.2.1 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。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。

3.3.2.2 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。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。（考虑插图示意拓扑怎么固定）

## 3.3.3 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问题是不等量分批，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。

四、针对分批调度问题的约束处理策略

# 4.1 编码解码方案设计

## 4.1.1 免修补双矩阵编码

## 4.1.2 基于机器柔性灵活指数的解码方式

## 4.1.3 实验分析

# 4.2 搜索算子的设计

## 4.2.1 粗细粒度配合搜索

## 4.2.2 粗粒度交叉搜索算子

## 4.2.3 细粒度邻域搜索算子

4.2.3.1 单一邻域

4.2.3.2 启发式邻域

4.2.3.3 细粒度全邻域

4.2.3.4 邻域搜索策略

4.2.3.5 实验分析

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。然后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，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

## 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。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。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：

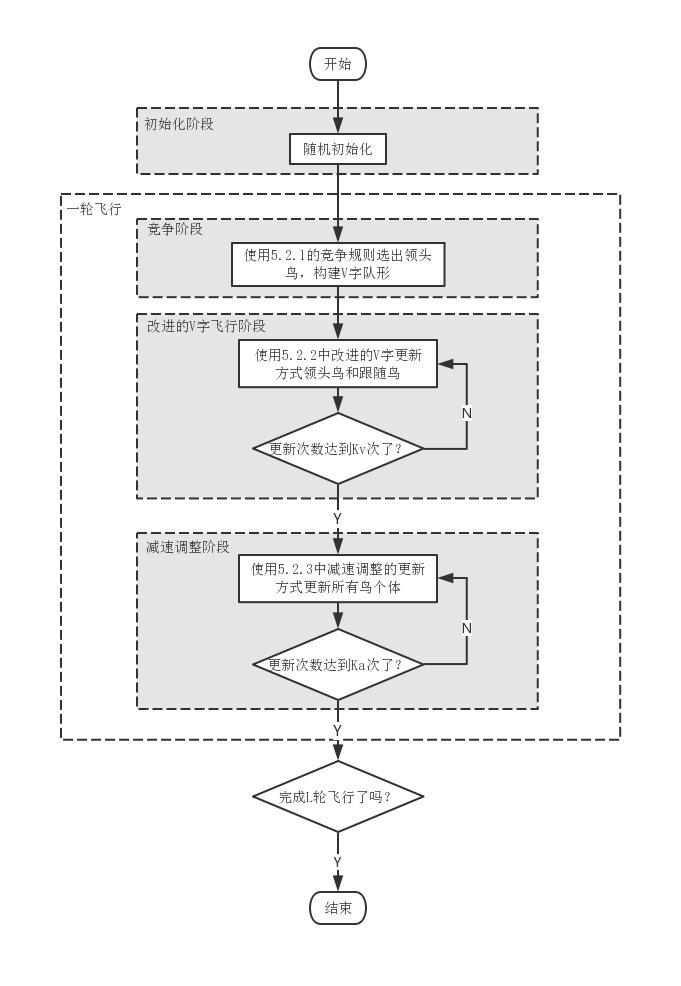
（1）初始化阶段。为了体现算法的鲁棒性，使用随机初始化，不使用加入启发式的初始化。

（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，本节首先把CCMBO应用在标准的FJSP分批调度问题上，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，并且与使用该标准问题的其他算法进行对比。然后把CCMBO算法应用到免疫分析检测设备分批调度问题上，验证分批调度的效果。

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

## 5.3.1 测试例子

5.3.1.1 FJSP分批调度标准测试用例

此处使用【赵】 中的测试用例，一共四个测试用例，类型都属于混排，带有准备操作，准备操作可分离，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，代表每个批的工件总数。一下是四个测试用例的规模：

P1：规模是，即4个批，6台机器，每一批都包含8个工件，这属于中小规模的分批调度问题。每类工件的每个工序在不同机器上的加工时间以及准备操作时间如表所示；

P2：规模是，批数与机器数跟P1相同，每一批都包含20个工件，这属于中小规模的分批调度问题。加工时间与准备操作时间与P1也相同，如表所示；

P3：规模是，即6个批，6台机器，每一批都包含10个工件，这属于中小规模的分批调度问题。每类工件的每个工序在不同机器上的加工时间以及准备操作时间如表所示；

P2：规模是，批数与机器数跟P3相同，每一批都包含20个工件，这属于大规模的分批调度问题。加工时间与准备操作时间与P3也相同，如表所示；

（不知道要不要把具体数据列在正文）

5.3.1.2 免疫分析检测设备分批调度问题

## 5.3.2 实验结果与分析

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[1]。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[2]。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[3]。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。其次设计了基于年龄特征的邻域搜索机制，对不同年龄层的个体执行不同的邻域搜索策略，在个体进入停滞年龄之后，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[4]。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。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。

（画图示意）

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。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；

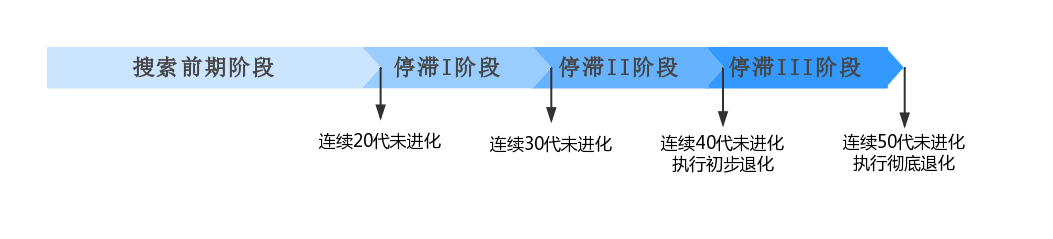
step2：分别从三个子种群G1，G2，G3中未被选择的个体中随机选择个个体，分别是G1r，G2r，G3r；

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4.2.3.4中提到的其他搜索策略，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。

停滞阶段的划分如下图所示。当个体连续20代以上没有进化，则进入“停滞I”阶段，使用2greedy-1step邻域搜索策略，提高邻域搜索的强度。当个体连续30代以上没有进化，则进入“停滞II”阶段，使用2greedy-2step邻域搜索策略，尝试使用更大的步幅来促进半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，个体进行初步退化，即退化到某个比当前差的状态，以尝试跳出局部最优，同时使用2greedy-2step邻域搜索策略。当个体连续50代都没有进化时，个体要进行彻底退化，即重新初始化该个体，彻底逃离局部最优的位置。



## 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。令字鸟群大小为，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

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

[2] 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.

[3] 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.

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