现代排序论与作业调度优化专题

DOI:10.11721/cqnuj20260202

基于自我学习的混合 CHIO 算法在准时化作业车间调度中的应用*

亓祥波[1]，赵品威[1]，宋　岩[2]，王　润[1]
(1. 沈阳大学 机械工程学院，沈阳 110044；2. 辽宁大学 物理学院，沈阳 110036)

摘要:随着客户定制化需求的增加以及对交货时间的关注,准时化生产成为提高企业竞争力的关键因素之一,面向准时化生产的车间调度值得深入研究。针对作业车间调度中拖期严重、准时化程度低等问题,提出了以最小拖期、最小提前期和最小化最大完工时间为目标的车间调度模型;针对该模型的求解,基于冠状病毒群免疫优化(coronavirus herd immunity optimizer, CHIO)算法提出了一种自我学习的混合 CHIO 算法(hybrid CHIO algorithm based on self-learning, HCHIO)。首先,设计了一种具备得分评价机制的自我学习算子库,使得算法能够针对不同问题进行自我学习从而选择最优算子以提升算法的全局寻优性能;其次,通过对最优解进行邻域搜索,增强了算法的局部搜索能力;最后,在基准测试与实际案例上对 HCHIO 进行了实验,验证了该算法在解决车间调度问题上良好的寻优能力。实验结果证明了 HCHIO 在求解准时化作业车间调度问题上的有效性。

关键词:准时化生产;作业车间调度;冠状病毒群免疫算法;自我学习;算子库
中图分类号:O221.4;TP391　　**文献标志码**:A　　**文章编号**:1672-6693(2026)02-0009-17

在现代制造业中,车间作业是生产过程中至关重要的环节之一。作业车间调度(job shop scheduling, JSP)则是制造业企业中的一项关键任务,目的是通过合理安排和调度作业流程,最大程度地提高企业关注的关键绩效指标。对 JSP 问题的研究在提升企业的竞争力、降低生产成本、提高客户满意度等方面具有重要意义。作为非确定多项式(non-deterministic polynomial, NP)完全问题,JSP 问题属于最难的约束组合优化问题[1],在多项式时间范围内没有一个有效的算法能求得最优解。随着全球制造业的发展和竞争的加剧,对调度的优化变得越来越重要,设计科学合理的调度方案可以提高生产效率、增加经济效益。因此,调度优化算法成为调度领域的研究热点之一。

近些年来,受自然界生物群体行为、物理现象和自然现象等启发设计出来的各种智能优化算法,因具有较强的全局搜索能力、自适应性、鲁棒性、并行计算能力和易实现等优点而被广泛应用于离散调度的问题求解中。亓祥波等人[2]针对以最小化最大完工时间为目标的置换作业车间调度问题提出了一种改进布谷鸟算法,首先运用 NEH、Logistic 混沌映射等方法生成初始解,其次引入了动态改变步长和差分进化的交叉策略,最后通过实验验证了该算法能够有效地求解离散调度问题。陈照辉等人[3]提出了一种改进的算术优化算法,引入了 6 种邻域搜索策略和非线性数学优化加速函数对算法进行了改进,通过实验验证了所提算法在解决离散调度问题的可行性。亓祥波等人[4]针对离散作业车间调度问题的特性提出了一种基于量子计算和 Weibull 分布的混合冠状病毒群免疫算法,增强了算法在离散调度问题上的全局搜索能力。

在当前制造业中,追求最小化最大完工时间是较为常见的优化目标。虽然提前完工在某些情况下看似有利,但如果不仔细管理和考虑相关风险,可能会带来库存沉余、资源浪费、供应链不协调和质量缺陷等一系列问题。最小提前期与最小拖期、最小化最大完工时间属于相互冲突的目标,研究如何在这些目标之间取得平衡、找到最优解的方法对于解决实际问题具有重要意义。目前随着客户定制化需求的不断增加以及对交货时间的关注,准时化生产(just-in-time,JIT)也成为车间调度需要考虑的重要方面。Baptiste 等人[5]在 2008 年首次将工序交货期的概念引入到车间调度问题中,即准时化作业车间调度问题(just-in time job shop scheduling problem,

* **收稿日期**:2024-07-25　**修回日期**:2025-05-27　**网络出版时间**:2026-04-13T13:51
资助项目:国家自然科学基金面上项目(No. 92067110);辽宁省教育厅高等学校基本科研项目(No. LJKQZ2021164);辽宁省自然科学基金项目(No. 2022-KF-12-11)
第一作者简介:亓祥波,男,副教授,博士,研究方向为智能优化、车间调度,E-mail: ustcdragon@syu.edu.cn;**通信作者简介**:赵品威,男,E-mail:1012993863@qq.com
网络出版地址:https://link.cnki.net/urlid/50.1165.n.20260411.1841.002

JIT-JSP)。王大志等人[6]设计了多智能体进化算法对最小化总拖期时间的流水车间调度问题进行求解。涂雪平等人[7]设计了一种混合禁忌搜索和数学规划的策略算法来求解准时化订单动态产品替补问题。Kurdi[8]提出了一种改进的岛屿模型模因算法,求解了以完工时间、总延迟时间和总提前时间为优化目标的调度问题。Zhang等人[9]针对车间调度问题提出了一种结合特定局部改进策略的多目标遗传算法,以最小能源消耗和最小化总拖期时间为求解目标,并通过改进2个受限子问题的数学模型来提高了解的质量。裴小兵等人[10]提出了一种博弈人工蜂群算法,求解了以最大完工时间、总空闲时间和总拖期时间为目标的作业车间调度问题。史双元等人[11]以最小拖期和最小最大完工时间为目标提出了一种多目标差分进化算法,并通过在基准算例上的实验验证了该算法的有效性。魏光艳等人[12]结合分布估计和禁忌搜索提出了一种混合算法,求解了目标为最小化总成本和最小拖期的车间调度问题。

JIT强调零库存、按需生产,并力求避免拖期或提前,通过上述文献分析可知,关于准时化车间调度的研究集中于以最小化拖期时间为求解目标却忽略了提前期的问题。在准时化的框架下,提前期同样是一个关键的调度指标。提前完成可能会导致库存积压等不必要的生产成本,这与准时化研究以避免库存过剩和减少资源浪费为目标是矛盾的。因此,仅追求最小化拖期时间的调度模型并不完全符合准时化的实际需求。

冠状病毒群免疫优化(coronavirus herd immunity optimizer, CHIO)算法是2020年Al-Betar等人[13]提出的一种新的元启发式优化算法,受新冠病毒群体免疫策略的启发。作为一种较新的智能优化算法,该算法具有收敛快、参数少等优点,已有不少学者对它作了改进并应用于各类实际问题上,例如路径规划[14]、医疗检测[15]、设计优化[16]等,但在离散调度问题上的应用相对较少。因此本文以最小总拖期、最小提前期和最小化最大完工时间的JIT-JSP为研究对象,并针对该问题在标准CHIO算法的基础上设计了一种能自我学习的混合冠状病毒群免疫优化算法(hybrid CHIO algorithm based on self-learning, HCHIO)。

1 相关基础知识

1.1 问题描述

传统生产模式与JIT模式的核心区别在于:传统生产模式是预测驱动,根据市场预测制定大批量生产计划,容易产生过剩库存;JIT模式是订单驱动,按需生产,仅在客户需要的时间生产所需数量的产品。JIT-JSP可以理解为在m台机器上对n个工件进行加工,并且每个工件都需按照特定的加工顺序在各个机器上依次进行加工,且每个工件在每台机器上的加工时间都是已知的。调度的主要目的是在m台机器之间合理安排加工任务的顺序,以实现最小拖期、最小提前期和最小化最大完工时间的目标。问题满足以下假设:

1) 机器之间相互独立;

2) 加工准备时间已经包含在加工时间中;

3) 所有机器均从0时刻开始可用;

4) 同一工件的不同工序之间有先后约束且不同工件之间的优先级相同;

5) 每道工序在同一时刻只能由1台机器进行加工;

6) 在加工过程中不会出现缺料或机器故障等情况。

1.2 数学模型

本文使用整数规划模型[1,17]对JIT-JSP进行描述,在确定每个工件的总处理时间的基础上增加一个交货松弛度,从而确定该工件的交货时间。本文模型的优化目标如下:

$$\min(F(x))=\min(w_1 f_1+w_2 f_2+w_3 f_3) \tag{1}$$

其中:$F(x)$表示目标函数;f_1表示最小化最大完工时间,$f_1=\min\max\{\max c_{ik}\}$,$c_{ik}$为工件$i$在机器$k$上的完成时间,$1\leqslant k\leqslant m$,$1\leqslant i\leqslant n$;$f_2$表示最小拖期,$f_2=\sum_{i=1}^{n}\max\{c_i-D_i,0\}$,$c_i$表示工件$i$完工时间,$D_i$表示工件$i$的应交期时间;$f_3$表示最小提前期,$f_3=\sum_{i=1}^{n}|\min\{c_i-D_i,0\}|$;$w_1$、$w_2$、$w_3$均为权重系数,可由企业根据实际需求设置且$w_1+w_2+w_3=1$。

问题约束为:

$$c_{ik}-p_{ik}+M(1-x_{ihk})\geqslant c_{ih},i=1,2,\cdots,n;h,k=1,2,\cdots,m。 \tag{2}$$

$$c_{jk}-c_{ik}+M(1-x_{ijk})\geqslant p_{jk},i,j=1,2,\cdots,n;k=1,2,\cdots,m。\tag{3}$$

$$c_{ik}\geqslant 0,i=1,2,\cdots,n;k=1,2,\cdots,m。\tag{4}$$

$$D_i=\theta\sum_{k=1}^{m}p_{ik}。\tag{5}$$

式(2)为工艺之间的约束，表示当前工序与上一道工序之间的约束关系；式(3)为机器之间的约束，表示同一机器在同一时刻只能加工 1 个工件。其中：c_{ik}，c_{jk} 分别为工件 i，j 在机器 k 上完成时的时间；c_{ih} 表示工件 i 在机器 h 上的完工时间；p_{ik}，p_{jk} 分别为工件 i，j 在机器 k 上需要加工的时间；M 是一个充分大的正数；D_i 表示该工件的应交期时间；θ 表示交货松弛度，设置为 1.5；x_{ihk} 和 x_{ijk} 分别为指示系数、指示变量，表示为：

$$x_{ihk}=\begin{cases}1,\text{机器 } h \text{ 先于机器 } k \text{ 加工工件 } i;\\0,\text{其他情况。}\end{cases}$$

$$x_{ijk}=\begin{cases}1,\text{若工件 } i \text{ 先于工件 } j \text{ 在机器 } k \text{ 上加工;}\\0,\text{其他情况。}\end{cases}$$

1.3　基于工序的编码方式

标准的 CHIO 算法只用于求解连续问题，无法直接求解离散问题，因此在求解离散的作业车间调度问题时需要将连续变量转换为离散变量。在这个转换过程中，本文使用了基于工序的编码方式。该编码方式由一系列工件号组成，其中编码的长度等于总加工工序的数量。在解码的过程中，按照从左到右的顺序对工序编码进行加工。当第 1 个工件号第 1 次出现时，表示该工件正在进行第 1 道工序的加工；当该工件号第 2 次出现时，表示该工件正在进行第 2 道工序，以此类推，直到所有的工序都被完成加工。以一个 3×2 规模问题为例，该问题表示有 3 个工件需要依次在 2 台机器上进行完成，共计需要 6 道工序。则该例连续变量转换离散变量的实现步骤如下。

步骤 1，假设算法的搜索空间为[0，1]，在搜索空间内随机生成 1×6 的连续变量。

步骤 2，对生成的随机变量按从小到大排序，得到中转变量，各中转变量在连续变量所处的位置索引即为离散变量。

步骤 3，将离散变量除以机器总数并向上取整即可得到工序编码。

表 1 给出了一个连续变量转换为离散变量的实例。

表 1　编码对应规则实例

Tab. 1　Example of coding correspondence rules

连续变量	中转变量	离散变量	连续变量	中转变量	离散变量
0.95	0.16	2	0.91	0.91	4
0.16	0.39	5	0.39	0.95	1
0.99	0.79	6	0.79	0.99	3

从表 1 可知，该加工工序的编码为[1，3，3，2，1，2]，其中编码的第 1 个[1]表示为第 1 个工件的第 1 道工序，第 2 个[1]则表示第 1 个工件的第 2 道工序，以此类推进行解码。

1.4　CHIO 算法

CHIO 算法中种群由易感者、感染者和免疫者 3 类个体构成。传播速度取决于感染者与其他个体的接触情况。算法主要模拟了 2 种防疫策略：一是社会距离，即通过限制个体间的接触来减缓传播；二是群体免疫，即当种群中获得免疫的个体达到一定比例时，传播自然中止。CHIO 算法通过对这 2 种策略的建模，来平衡优化过程中的全局搜索与局部开发。具体流程如下。

步骤 1，初始化种群。在初始化种群 $\boldsymbol{H}$ 后，随机产生与种群个数 H_s 一样多的一组病例(即个体)。生成的病例以二维矩阵 $H_s\times n$ 形式存储，且每个个体均对应一个状态向量 $\boldsymbol{S}_i$ 和年龄向量 $\boldsymbol{A}_j$，$\boldsymbol{H}$ 的计算公式为：

$$\boldsymbol{H}=\begin{bmatrix}x_1^1 & x_2^1 & \cdots & x_n^1\\ x_1^2 & x_2^2 & \cdots & x_n^2\\ \vdots & \vdots & & \vdots\\ x_1^{H_s} & x_2^{H_s} & \cdots & x_n^{H_s}\end{bmatrix}。$$

其中:n 为问题维度。

步骤2,个体进行迭代更新。在算法迭代过程中,x_i 病例的 j 基因要么保持不变,要么受社会距离的变化而变化,该变化通过与病毒传播率 B_r 的占比比较来体现,B_r 的取值为0.05,变化的公式为:

$$x_i^j(t+1)=\begin{cases}x_i^j(t), r\geqslant B_r;\\ C(x_i^j(t)), r<\dfrac{1}{3}B_r;\\ N(x_i^j(t)), r<\dfrac{2}{3}B_r;\\ R(x_i^j(t)), r<B_r。\end{cases}$$

其中:r 表示(0,1)之间的随机数,$x_i^j(t+1)$表示 $x_i^j(t)$更新后的基因值;

$$C(x_i^j(t))=x_i^j(t)+r\times(x_i^j(t)-x_i^c(t)),$$

其中:$x_i^c(t)$表示从感染个体中选择的个体,此时 $c=\{i\mid S_i=1\}$;

$$N(x_i^j(t))=x_i^j(t)+r\times(x_i^j(t)-x_i^m(t)),$$

其中:$x_i^m(t)$表示从易感个体中选择的个体,此时 $m=\{i\mid S_i=0\}$;

$$R(x_i^j(t))=x_i^j(t)+r\times(x_i^j(t)-x_i^v(t)),$$

其中:$x_i^v(t)$表示从免疫个体中选择的最好的个体,此时 $v=\{i\mid S_i=2\}$。

步骤3,适应度值更新。随后更新群体免疫群体,计算新个体 $x_i(t+1)$的免疫率(即适应度值),如果 $f(x_i(t+1))>f(x_i(t))$,则 $x_i(t+1)$取代 $x_i(t)$。且如果该个体为感染个体,则年龄向量 $\boldsymbol{A}_j$ 增加1。其中个体对应状态向量 $\boldsymbol{S}_i$ 也随适应度值的改变而改变,公式为:

$$\boldsymbol{S}_i=\begin{cases}1, f(x_i(t+1))<\dfrac{f(x)_i(t+1)}{\Delta f(x)}\wedge\boldsymbol{S}_i=0\wedge C_a(x_i(t+1));\\ 2, f(x_i(t+1))>\dfrac{f(x)_i(t+1)}{\Delta f(x)}\wedge\boldsymbol{S}_i=1。\end{cases}$$

其中:$f(x_i(t+1))$表示在 $t+1$ 代时第 i 个个体的适应度;$\Delta f(x)$表示当前种群的平均免疫率,C_a 表示当前个体是否继承过感染个体基因,如果继承过感染个体的基因,当前个体为易感人群且免疫率小于平均免疫率,则该个体感染病毒,此时该个体的 $\boldsymbol{S}_i=1$;如果当前个体免疫率大于平均免疫率且当前个体为感染个体,则该个体为免疫个体,此时该个体的 $\boldsymbol{S}_i=2$。

步骤4,感染个体死亡重置。最后,当感染的个体年龄超过最大年龄后,该个体将死亡重置,公式为:

$$x_i=l_b+(u_b-l_b)\times\mathrm{rand}$$

其中:u_b、l_b 分别表示问题的搜索空间的上界和下界,rand 表示(0,1)之间的随机数。

1.5 算子介绍

算子是智能优化算法中实现特定功能的操作单元,类似于编程中的函数,但具有更复杂的实现逻辑。例如,在神经网络中,卷积操作、激活函数等均可视为算子;在优化算法中,算子通过封装复杂算法如交叉、变异等操作来实现解空间的动态调整。算子的本质是通过数学或物理规律对解空间进行操作,推动算法向最优解收敛。

1.5.1 白鲸优化算子

白鲸优化算子源于 Zhong 等人[18]于2022年提出的白鲸优化算法(beluga whale optimization, BWO),该算法灵感主要源于白鲸的群体捕食行为。BWO 包括探索、开发和鲸落3个主要阶段。

1) 算子1:探索算子

该算子的位置更新公式为:

$$\begin{cases}x_{i,j}^{t+1}=x_{i,p_j}^t+x_{r,p_1}^t-x_{i,p_j}^t\times(1+r_1)(\sin 2\pi r_2), j=\text{偶数};\\ x_{i,j}^{t+1}=x_{i,p_j}^t+x_{r,p_1}^t-x_{i,p_j}^t\times(1+r_1)(\cos 2\pi r_2), j=\text{奇数}。\end{cases}$$

其中:t 表示为当前迭代次数,$x_{i,j}^{t+1}$ 是第 i 只白鲸在 $t+1$ 代第 j 维的更新后的新位置,$p_j(j=1, 2, \cdots, d)$是从 d 维中选择的随机整数,x_{i,p_j}^t 是第 i 只白鲸在 t 代时第 p_j 维的值,x_{r,p_1}^t 表示第 t 代随机选择的白鲸个体位置,r 表示随机选择的索引,r_1 和 r_2 均表示为(0,1)之间的随机数。

2）算子 2：开发算子

该算子个体更新公式为：

$$x_i^{t+1}=r_3\times x_{\text{best}}^t-r_4\times x_i^t+C_1\times L_{\text{F}}\times(x_r^t-x_i^t)。$$

其中：t 为当前迭代次数，x_i^{t+1} 表示第 i 只白鲸根据其他白鲸共享的位置更新后的位置，x_{best}^t 表示当前白鲸种群中白鲸的最优位置，x_i^t 表示第 i 只白鲸在 t 代的位置，r_3 和 r_4 均表示(0,1)之间的随机数，C_1 是衡量 Levy 飞行的随机跳跃强度，表达式为 $C_1=2r_4\times(1-t/T_{\max})$，$L_{\text{F}}$ 表示 Levy 飞行函数，x_r^t 表示在 t 代时随机选择一只白鲸的位置。

3）算子 3：鲸落算子

该算子的更新方式如下所示：

$$x_i^{t+1}=r_5x_i^t-r_6x_r^t+r_7x_{\text{step}}。$$

其中：r_5，r_6，r_7 均是(0,1)之间的随机数；x_{step} 表示鲸鱼坠落的步长，定义为：$x_{\text{step}}=(u_{\text{b}}-l_{\text{b}})\exp\left(-C_2\times\dfrac{t}{T_{\max}}\right)$，$C_2$ 是与鲸落概率 W_{f} 和种群规模 n 相关的跃阶因子，表达式为 $C_2=2W_{\text{f}}\times n$，$u_{\text{b}}$ 和 l_{b} 分别表示变量的上界和下界。

1.5.2　麻雀优化算子

麻雀优化算子源于 2020 年提出的麻雀优化算法(sparrow search algorithm，SSA)[19]，该算法主要受到了麻雀觅食行为和反捕食行为的启发。在 SSA 中，麻雀群体中存在着不同的角色分工，包括探索者、追随者和警戒者。

1）算子 4：探索者算子

该算子位置更新公式为：

$$x_{i,j}^{t+1}=\begin{cases}x_{i,j}^t\exp\left(-\dfrac{t}{\alpha T_{\max}}\right), R_1<S_T;\\ x_{i,j}^t+\boldsymbol{QL}, R_1\geqslant S_T。\end{cases}$$

其中：$x_{i,j}^{t+1}$ 表示在 $t+1$ 代时第 i 只麻雀在第 j 维中的更新位置，t 为当前迭代次数，$j=1,2,\cdots,d$；$T_{\max}$ 为最大迭代次数；α 是(0,1]中的一个随机数；Q 为服从高斯分布的随机数；$\boldsymbol{L}$ 表示 $1\times d$ 维的矩阵；R_1、S_T 分别为预警值和警戒值，且 $R_1\in[0,1]$，$S_T\in[0.5,1]$。

2）算子 5：追随者算子

该算子位置更新公式为：

$$x_{i,j}^{t+1}=\begin{cases}Q\exp\left(\dfrac{x_{\text{worst}}-x_{i,j}^t}{i^2}\right), i>n/2;\\ x_p^{t+1}+\left|x_{i,j}^t-x_p^{t+1}\right|\boldsymbol{A}^+\boldsymbol{L}, i\leqslant n/2。\end{cases}$$

其中：x_{worst} 表示当前种群中的全局最差位置，x_p^{t+1} 是在 $t+1$ 代时麻雀种群中探索者的全局最优位置，$\boldsymbol{A}$ 为 $1\times d$ 维的矩阵，矩阵中每个元素随机赋值 1 或 -1 且 $\boldsymbol{A}^+=\boldsymbol{A}^{\text{T}}(\boldsymbol{AA}^{\text{T}})^{-1}$。

3）算子 6：警戒者算子

警戒者位置更新公式为：

$$x_{i,j}^{t+1}=\begin{cases}x_{\text{best}}^t+\beta\left|x_{i,j}^t-x_{\text{best}}^t\right|, f_i>f_{\text{g}};\\ x_{i,j}^t+K\left(\dfrac{\left|x_{i,j}^t-x_{\text{worst}}^t\right|}{(f_i-f_{\text{w}})+\varepsilon}\right), f_i=f_{\text{g}}。\end{cases}$$

其中：x_{best}^t 是在 t 代时当前种群中全局最优位置；β 表示均值为 0、方差为 1 的高斯分布随机数；K 表示[-1,1]之间的随机数；f_i 为当前麻雀个体的适应度值；f_{g} 表示当前全局最优适应度值；f_{w} 则为当前全局最差适应度值；ε 为避免分母出现 0 这一情况的极小常数。

1.5.3　其他算子

1）算子 7：正余弦算子

正余弦算子源于正余弦优化算法(sine cosine algorithm，SCA)[20]，SCA 是一种随机优化算法，寻优过程主要分为 2 个阶段：探索阶段和开发阶段。算子位置更新方式为：

$$x_i^{t+1}=\begin{cases}x_i^t+r_1\times\sin r_2\times|r_3P_i^t-x_i^t| & r_4<0.5;\\ x_i^t+r_1\times\cos r_2\times|r_3P_i^t-x_i^t| & r_4>0.5。\end{cases}$$

其中:x_i^{t+1} 是个体 i 的更新位置,x_i^t 是个体 i 在第 t 代的当前位置,r_2 表示(0,2)之间的随机数,r_1,r_3,r_4 分别为(0,1)之间的随机数,P_i^t 表示 t 代时的最优个体。

2) 算子8:量子旋转门算子

量子旋转门算子源于量子计算(quantum computing, QC)[21],QC由Feynman于1982年首先提出,现已成为国内外学者关注的前沿学科。在QC中,信息的基本存储单位称为量子比特(quantum bit, QB),QB按类型可分为单量子比特、双量子比特、多量子比特。其中单量子比特的叠加状态为:

$$|\varphi\rangle=\alpha|0\rangle+\beta|1\rangle,$$
$$|\alpha|^2+|\beta|^2=1。\tag{6}$$

其中:$|\varphi\rangle$表示量子比特的叠加态,α 表示量子比特处于基态$|0\rangle$的概率幅,β 表示量子比特处于基态$|1\rangle$的概率幅。

量子门是QC中的基本操作单元,用于在QB上执行特定的逻辑变换。常见的量子门包括Hadamard门、相位门和量子旋转门等。其中量子旋转门表达式为:

$$R(\theta)=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix},$$
$$\begin{bmatrix}\widetilde{\alpha}\\ \widetilde{\beta}\end{bmatrix}=R(\theta)\begin{bmatrix}\alpha\\ \beta\end{bmatrix}。$$

其中:θ 为量子旋转角且$\theta=\text{rand}\times2\pi$;$\begin{bmatrix}\widetilde{\alpha}\\ \widetilde{\beta}\end{bmatrix}$为更新后的概率幅。由式(6)可知 $\beta=\sqrt{1-\alpha^2}$,则量子旋转门的更新表达式为:

$$\widetilde{\alpha}=\left|\alpha\times\cos\theta-\sqrt{1-\alpha^2}\times\sin\theta\right|。$$

因此,经过量子旋转门变换后的个体更新表达式为:

$$Q_i=\left|x_i\times\cos\theta-\sqrt{1-x_i^2}\times\sin\theta\right|。$$

其中:Q_i 表示 x_i 个体通过量子旋转门转换后产生的新个体。

3) 算子9:Weibull分布算子

Weibull分布是一种连续概率分布,在可靠性工程和寿命预测中具有广泛应用。概率密度函数(probability density function, PDF)表达式为:

$$f(x)=\begin{cases}\dfrac{B}{A}\left(\dfrac{x}{A}\right)^{(B-1)}\exp\left(-\left(\dfrac{x}{A}\right)^B\right), & x\geqslant0;\\ 0, & x<0。\end{cases}$$

其中:A 为尺度参数,表示函数的走势;B 为形状参数,表示函数的形状。图1表示尺度为 A,形状为 B 的Weibull分布的PDF。Sahoo等人[22]将Weibull分布算子应用到飞蛾扑火算法上,证实了该算子可以在当前位置搜索到有用的信息以增加算法种群的多样性。该算子的具体实现如下所示:

$$S_{\text{step}}=\text{sign}(\text{rand}-0.5)\times\text{wbl}(1,1)。$$
$$s_{\text{step}}=\begin{cases}0.5\times\text{sign}(\text{rand}-0.5)\times\text{wbl}(0.5,1)\times\text{norm}(x_{\text{best}}-x_i), & x_{\text{best}}\neq x_i;\\ 0.1\times\text{sign}(\text{rand}-0.5)\times\text{wbl}(0.5,1), & x_{\text{best}}=x_i。\end{cases}$$
$$x_{\text{new}}^j=\begin{cases}x_i^j+S_{\text{step}}, & \text{rand}\leqslant0.25;\\ x_i^j+s_{\text{step}}, & \text{rand}>0.25。\end{cases}$$

其中:S_{step} 表示Weibull分布算子的大步长,s_{step} 表示为小步长,sign(rand−0.5)为符号函数,可随机取得值为的−1或1的向量,wbl(·,·)表示尺度 A、形状 B 不同取值时的Weibull分布的随机数,norm为范数函数,x_i^j 表示第 i 个个体第 j 维的值,x_{best} 表示当前种群中最优个体,x_{new}^j 表示 x_i 生成的新个体的第 j 维的值。

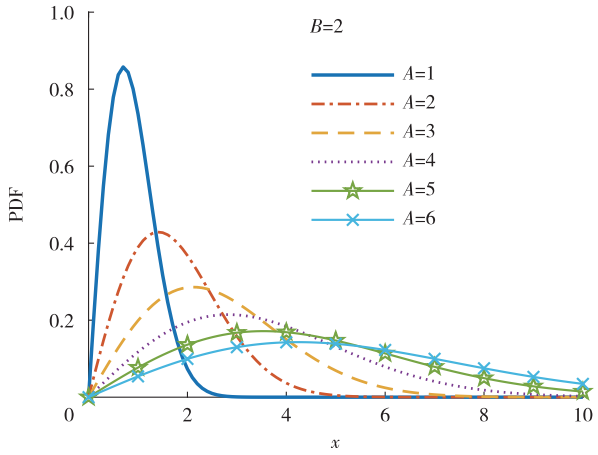


图 1　Weibull 分布示意图

Fig. 1　The schematic diagram of the Weibull distribution

4）算子 10：改进高斯分布算子

高斯分布（Gaussian distribution，GD）[23]最早由棣莫弗发现，已被多次运用到各种元启发式算法的改进中。在本算法中 $\sigma^2(t)=\exp(t/T_{\max})$。根据公式得到方差为 $\sigma^2(t)$ 的高斯变异算子 $G(t)$。高斯变异扰动公式为：

$$x_G=x_i^t+G(t)\times(x_{r1}^t-x_{r2}^t)。$$

其中：x_G 表示经过改进高斯算子变异后的个体，x_i^t 表示 t 代时第 i 个个体，x_{r1}^t 和 x_{r2}^t 分别表示在 t 代时从种群中随机选择的 2 个不相同的个体，$r1$，$r2$ 分别表示 2 个个体的索引。

5）算子 11：萤火虫变异算子

萤火虫变异算子源于萤火虫优化算法（fire-fly algorithm，FA）[24]。该算子主要模拟了萤火虫根据个体亮度而相互吸引的行为，实现公式为：

$$x_f=x_i^t+\beta_0\exp(-\gamma r^2)\times(x_{\text{best}}-x_i^t)+\alpha\varepsilon_0。$$

其中：x_i^t 表示在 t 代时第 i 个个体；β_0 表示 2 只萤火虫相互接触时的吸引度，即为初始吸引度，取值为 1；γ 表示介质对光的吸收系数，取值为 1；$r=\sqrt{\sum_{j=1}^{\dim}(X_{i,j}-X_{\text{best},j})^2}$，dim 表示所求解问题的维度大小，$X_{\text{best},j}$ 表示最优个体在 j 维的位置。x_{best} 是最优个体；α 为扰动步长因子，取值为 0.5；ε_0 表示在[−0.5，0.5]之间均分分布的随机数。

6）算子 12：变邻域搜索算子

变邻域搜索算法（variable neighborhood search，VNS）是一种改进型的局部搜索算法。在 VNS 的基础上设计了一种基于正态分布的变邻域搜索算子，对当前解根据搜索半径 R 进行局部扰动，扰动公式为：

$$x_i^t=x_{i,\text{new}}^t+R\times\text{randn}。$$

其中：$x_{i,\text{new}}^t$ 表示扰动后的新 x_i 个体，x_i^t 表示在 t 代时第 i 个个体，R 表示搜索半径，randn 表示服从正态分布的随机数。扰动后，若新个体适应度优于原个体，则对该个体进行贪婪选择；否则扩大 R 的搜索半径。

2　HCHIO 算法

在智能优化算法中，探索通常指的是在搜索空间中广泛地进行搜索，目的是覆盖尽可能多的区域，寻找全局最优解或至少找到多个可能的优解。开发是指利用当前的解来进一步优化，目标是集中搜索当前已知的局部最优区域[25]。在 CHIO 算法中，由于原始算法的算子迭代方式较为单一，且算子对不同问题的适应性较差，面对复杂优化问题时，算子无法根据问题特点动态调整，容易导致算法陷入局部最优解，无法找到全局最优解。为此，算法在前期引入了基于自我学习算子库的策略，以增强全局探索能力；在后期，则通过引入多邻域搜索来提升局部开发能力。通过引入这 2 种策略平衡了算法前后期的搜索能力，增强了算法的性能。

2.1　基于自我学习算子库的全局探索策略

文献[26]提出了“无免费的午餐”定理，该定理指出对于优化问题而言，不存在一个单一的优化算法能够在所有可能的优化问题中都始终优于其他算法。换言之，如果一个算法在某些特定类型的问题上表现优异，那么

它可能在另一些类型的问题上表现不佳。并且不同算法之间的差异主要由算子之间的差异引起的。结合该定理,可以理解为不同算法对不同问题的求解效率、效果均不同,这主要是由算子的差异所带来的。不同的算子对问题结构产生不同程度的适应性,一些算子可能对某些问题的特定特征更加敏感。例如,在遗传算法中,选择、交叉和变异等算子会影响搜索空间的探索方式和优化过程。这些算子的设计会对算法的效率和效果产生重要影响。因此,对于不同问题的求解,选择合适的算子对于算法的性能至关重要。受该理论启发,本文设计了一种能够自我学习的算子库策略,其中算子库主要用于存储所需算子,包含了 12 种算子。这 12 种算子各自针对收敛性、种群多样性和邻域搜索进行不同的性能设计,互为补充,在面对大规模、复杂的问题时,能平衡算法的探索与开发能力,是解决作业车间调度复杂约束的关键。评价库则用于存储对应算子的得分情况,算子得分越高则意味着该算子对于问题的适应程度越高,求解效果也就越好;反之则适应程度较差,求解效果也越差。随着算法不断地迭代,评价库中算子的得分不断累积,使得算法能够针对不同问题选择最优算子以提升性能,从而使得算法实现自我学习,增加全局探索能力。算子库的自我学习过程如图 2 所示。

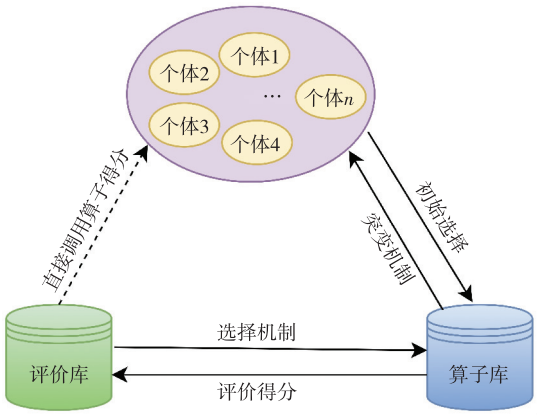


图 2　自我学习示意图

Fig. 2　The schematic diagram of self-learning

具体算子描述和实现步骤如下所示:

步骤 1,初始化算子库和评价库。

步骤 2,在算法进入迭代前期,对每个个体从算子库中随机选择 1 个算子。

步骤 3,对算法中适应度较差的个体易感者和感染者按照该个体选择的算子进行更新。

步骤 4,如果算子使更新后的个体得到了提升,则在评价库中记录该算子得分;否则不得分。

步骤 5,如果个体在当前算子下更新 2 次且未得到提升,则触发突变机制,调用评价库中算子得分情况重新为该个体选择算子,选择方法可以为轮盘赌、锦标赛、线性排序、指数排序等[27],本文的选择方法为轮盘赌选择法(即分数越高的算子被选择的概率越大);否则转到步骤 3。

2.2　基于多邻域搜索的局部开发策略

文献[28]指出全局最优解是在所有邻域结构中的局部最优解中找到的,并且某种邻域结构的局部最优解通常接近于另一种邻域结构的局部最优解。基于此理论,在本文中,经过迭代过程后,对种群中的最优个体设计了交换、逆转和倒置 3 种邻域结构,在邻域内寻找比当前解更好的解,并逐步优化得到最佳解,以增强算法的局部挖掘能力。多邻域搜索的示意图如图 3 所示。

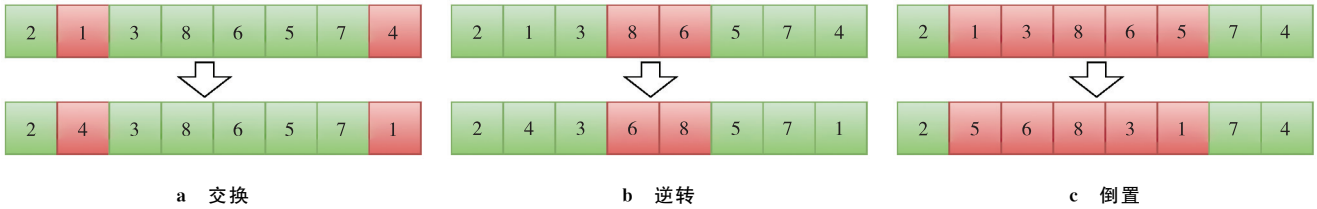


图 3　多邻域搜索示意图

Fig. 3　The schematic diagram of a multi-neighborhood search

多邻域搜索的具体方法如下：

1）交换操作：假设加工任务的总加工工序数为 N，在 1～N 中随机选择 2 个不同的位置，然后交换这 2 个位置的加工工序。

2）逆转操作：假设加工任务的总加工工序数为 N，在 1～N 中随机选择 1 个位置 O_a，然后将该位置与它后面的工序位置 O_{a+1} 进行交换。如果选择的位置是最后一个位置，则将该位置的工序与工序列表中的第 1 个位置的工序进行交换。

3）倒置操作：假设加工任务的总加工工序数为 N，在 1～N 中随机选择 2 个不同的整数为 a 和 $a+p$，假设未进行倒序操作前的工序编码为 $O=[O_1,O_2,O_3,\cdots,O_a,O_{a+p},\cdots,O_N]$，则执行倒序操作后的工序编码为 $O_{new}=[O_1,O_2,\cdots,O_{a+p},O_{a+p-1},\cdots,O_a,\cdots,O_N]$。

2.3　HCHIO 算法流程

图 4 给出了 HCHIO 的算法流程。

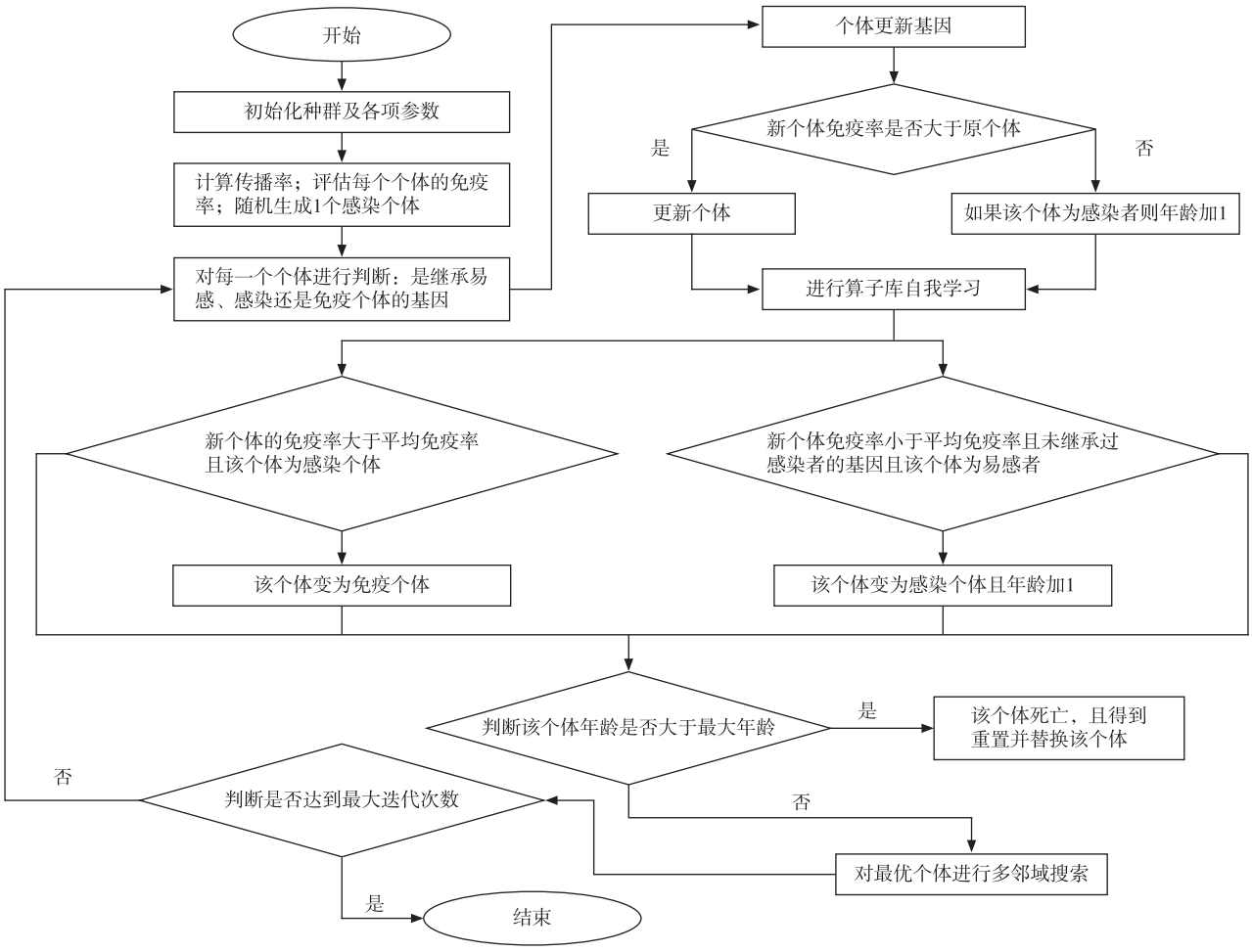


图 4　HCHIO 算法流程图

Fig. 4　The algorithm flowchart of HCHIO

3　实验仿真与结果分析

为了验证 HCHIO 的性能，采用了车间调度上典型的 FT 算例和 LA 算例进行了仿真测试，算例的规模见表 2。实验所用计算机设备参数如下：CPU 为 AMD Ryzen7 5700U、主频为 1.8 GHz、运行内存为 16 GB、操作系统为 Windows 11，编程环境为 Matlab 2018b。

3.1　对比算法与参数设置

为了便于对比分析，选取了几种较为新颖先进的智能优化算法作为对比算法，其中有混合萤火虫-粒子群算法（HFPSO）[29]、自适应对立黏菌算法（AOSMA）[30]、CHIO[13]、BWO[18]、SSA[19]、蜣螂算法（DBO）[31]以及基于

QC和Weibull分布的混合冠状病毒群免疫算法(QCHIO)[4]。为了公平起见,各算法初始化时采用相同大小的种群,所有算法参数均设置种群个数为50,最大迭代次数为500,算法独立运行次数为30次;目标函数的w_1、w_2、w_3均设置为1/3,HCHIO中的算子在个体得到改进时,在评价库中得1分,否则不得分。分别记录每个算例的平均值、最优值作为实验结果,结果如表3所示。为了反映算子库的有效性,给出了算法运行30次后,各算子在评价库的得分情况,如表4所示。同时,考虑到各企业对于各目标的权重设置有所差异,因此针对目标权重的不同情况做了相应的仿真实验,参数除权重不同外,均与上文保持一致,实验结果如表5所示。

表2 算例规模

Tab. 2 Example scale

算例	规模	算例	规模	算例	规模	算例	规模	算例	规模
FT06	6×6	FT20	20×5	LA11	20×5	LA21	15×10	LA31	30×10
FT10	10×10	LA01	10×5	LA16	10×10	LA26	20×10	LA36	15×15

表3 HCHIO与对比算法在算例上对比

Tab. 3 The comparison between HCHIO and the contrast algorithm on the example cases

算例	HCHIO		CHIO		HFPSO		AOSMA		BWO		SSA		DBO		QCHIO	
	最优	平均	最优	平均	最优	平均	最优	平均	最优	平均	最优	平均	最优	平均	最优	平均
FT06	**27**	**28**	28	29	28	31	29	31	28	30	29	31	28	31	28	29
FT10	**724**	**887**	844	1 005	814	1 053	941	1 177	1 120	1 355	1 029	1 380	1 046	1 285	890	985
FT20	**3 062**	**3 452**	4 082	4 427	3 766	4 232	4 412	4 922	4 854	5 330	4 811	5 595	4 251	4 790	3 983	4 283
LA01	**569**	**626**	601	649	573	676	592	609	761	750	672	763	625	710	575	633
LA11	**2 978**	**3 371**	3 699	3 940	3 555	3 920	3 979	4 347	4 279	4 587	4 226	4 556	3 629	4 091	3 634	3 839
LA16	**472**	**554**	586	664	479	716	601	814	693	882	749	907	650	816	570	643
LA21	**1 083**	**1 292**	1 623	1 729	1 163	1 490	1 591	1 965	1 925	2 287	1 924	2 488	1 680	2 077	1 466	1 701
LA26	**2 428**	**2 826**	3 757	4 037	2 777	3 410	3 973	4 565	4 406	4 978	4 262	5 116	3 715	4 383	3 340	3 875
LA31	**7 186**	**7 708**	10 867	11 270	9 077	10 067	10 889	11 667	11 579	12 782	11 245	13 293	10 355	11 667	10 343	11 078
LA36	**1 087**	**1 301**	1 676	1 913	1 260	1 617	1 480	2 032	2 062	2 402	1 958	2 593	1 848	2 379	1 639	1 835

注:加粗的数据表示最优值。

表4 各算子总得分

Tab. 4 Total score of each operator

算例	算子1	算子2	算子3	算子4	算子5	算子6	算子7	算子8	算子9	算子10	算子11	算子12
FT06	2 061	750	808	0	3 753	1 741	335	830	861	903	74	2 005
FT10	1 838	314	541	2	6 527	975	351	219	1 591	492	834	3 460
FT20	1 844	378	744	2	7 010	1 295	373	373	2 261	663	2 120	3 501
LA01	1 835	662	693	1	7 031	1 237	313	715	1 524	818	197	2 738
LA11	1 296	454	564	2	7 035	989	344	348	2 273	578	1 848	3 403
LA16	1 757	377	501	1	5 469	892	324	282	1 401	561	492	3 346
LA21	1 350	351	498	2	4 794	696	405	193	1 742	492	1 794	3 938
LA26	1 440	274	506	1	4 732	649	372	98	1 766	406	2 278	4 168
LA31	1 135	213	377	8	3 783	443	469	48	2 087	301	3 642	4 374
LA36	1 206	313	323	2	5 009	473	366	35	2 072	358	2 218	4 016
总得分	15 762	4 086	5 555	21	55 143	9 390	3 652	3 141	17 578	5 572	15 497	34 949

表 5　权重不同时 HCHIO 在各算例上的实验结果

Tab. 5　The experimental results of HCHIO on each example with different weights

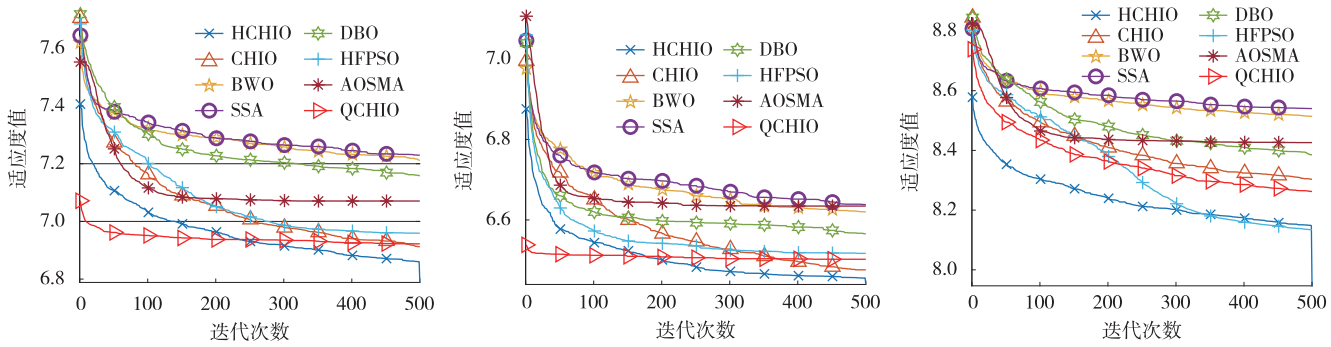
算例	(0.33,0.33,0.33)		(0.1,0.45.0.45)		(0.3,0.35,0.35)		(0.5.0.25,0.25)		(0.7,0.15,0.15)		(0.9,0.05,0.05)	
	最优值	平均值	最优值	平均值	最优值	平均值	最优值	平均值	最优值	平均值	最优值	平均值
FT06	27	28	28	29	28	29	28	31	30	31	30	31
FT10	637	844	721	876	711	908	776	944	761	953	681	955
FT20	3 062	3 452	3 139	3 438	3 182	3 519	3 162	3 548	3 146	3 549	2 983	3 606
LA01	529	595	539	631	539	633	556	630	530	651	603	689
LA11	3 133	3 458	3 161	3 411	3 262	3 448	3 000	3 470	3 197	3 464	2 926	3 540
LA16	472	554	526	584	464	610	526	627	561	636	457	644
LA21	1 143	1 339	1 235	1 310	1 195	1 365	1 139	1 379	1 121	1 364	1 174	1 377
LA26	2 445	2 751	2 597	2 753	2 350	2 765	2 534	2 790	2 566	2 806	2 581	2 874
LA31	7 186	7 708	7 328	7 668	7 145	7 707	7 338	7 808	7 223	7 840	7 062	7 939
LA36	1 087	1 301	1 074	1 244	1 048	1 306	1 011	1 298	1 080	1 391	1 008	1 407

3.2　仿真结果与分析

从表 3 的实验结果可以看出，HCHIO 在 10 个算例中，无论是最优值，还是平均值，均取得了最优，尤其在大规模问题 LA31、LA36 上，相较于原始 CHIO 算法在平均值上均提升了 32%，相较于 HFPSO 分别提升了 24% 和 19%，相较于 AOSMA 提升更大，分别为 34% 和 43%。从表 4 的算子得分可以推出，本文算子库的 12 个算子中，求解离散调度问题效果最优的为算子 5，其次为算子 12，效果最差的为算子 4，这也从反面验证了“无免费的午餐”定理，即没有一个算法在所有的问题上表现都好。

表 5 中的(·,·,·)表示 HCHIO 算法中各权重大小，括号内依次为权重 w_1，w_2 和 w_3。从表 5 可以看出，在 FT06、FT10、FT20、LA11、LA16、LA26、LA31、LA36 算例上随着拖期和提前期的权重不断增大，实验结果却不断减小。在 LA01 和 LA21 算例上虽然出现波动，但整体上仍呈现减小趋势。由此可以推断出，在实际生产中，随着对 JIT 越来越重视，也即对拖期和提前期权重的增大，生产消耗的总工时将会不断减小。

为了反映 HCHIO 在求解调度问题的动态收敛特性，本文选择 3 个测试算例(FT10、LA01、LA26)作为代表，分别给出了平均迭代曲线对比图，如图 5 所示。



a　FT10　　b　LA01　　c　LA26

图 5　在部分算例上的迭代曲线图

Fig. 5　Iterative curve graphs of some examples

从图 5 可以看出，在相同迭代次数下，HCHIO 在 FT10 与 LA26 算例上，无论是收敛速度还是寻优精度均远大于其他算法。在 LA01 算例上，从迭代速度上看，在迭代前期 HCHIO 的速度虽略逊于 HFPSO，但随着迭代数据的不断积累，算子选择的不断优化，在迭代到 50 代左右时，收敛速度和寻优精度均已远超过其他算法。

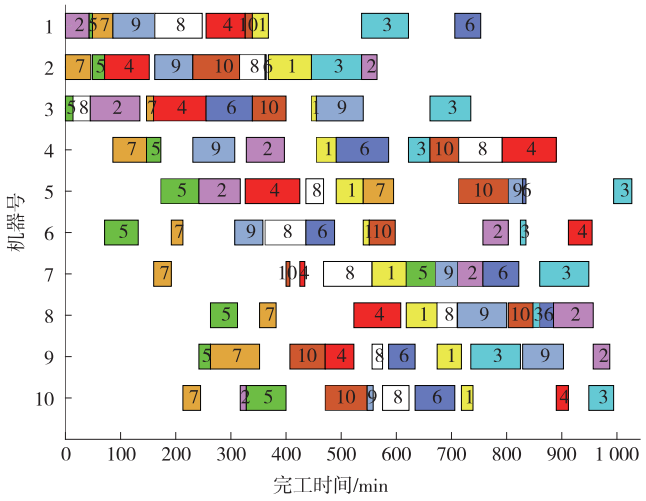
表 6 给出了 7 种算法在算例 FT10、LA16、LA36 算例上最优解的最小拖期、最小提前期和最小化最大完工时间，图 6 为 HCHIO 获取最优解对应的甘特图。

表 6 算法在部分算例上最优解的各目标值

Tab. 6 The objective values of the optimal solutions of the algorithm in some examples min

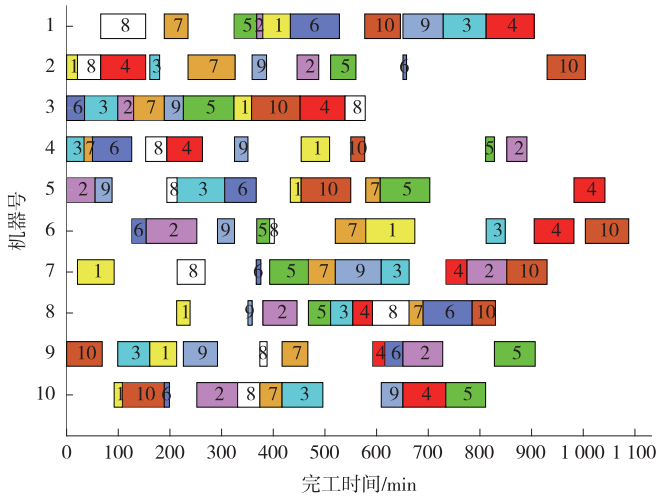
算例	CHIO			HCHIO			HFPSO			AOSMA		
	拖期	提前期	完工时间	拖期	提前期	完工时间	拖期	提前期	完工时间	拖期	提前期	完工时间
FT10	1 626.50	70.00	1 176.00	811.50	73.00	1 027.00	1 365.00	9.50	1 119.00	1 557.00	13.50	1 036.00
LA16	665.00	20.50	1 072.00	309.50	18.00	1 088.00	672.00	33.50	1 090.00	673.00	33.50	1 097.00
LA36	3 182.00	329.50	1 517.00	1 795.00	4.50	1 462.00	1 885.50	0.00	1 432.00	3 671.00	47.50	1 691.00

算例	BWO			SSA			DBO		
	拖期	提前期	完工时间	拖期	提前期	完工时间	拖期	提前期	完工时间
FT10	2 205.00	30.50	1 098.00	1 853.00	61.50	1 140.00	1 386.00	87.50	1 154.00
LA16	912.50	155.00	1 011.00	1 168.50	0.00	1 077.00	841.00	5.50	1 102.00
LA36	4 545.50	11.00	1 628.00	4 239.50	0.00	1 635.00	3 993.50	16.00	1 535.00



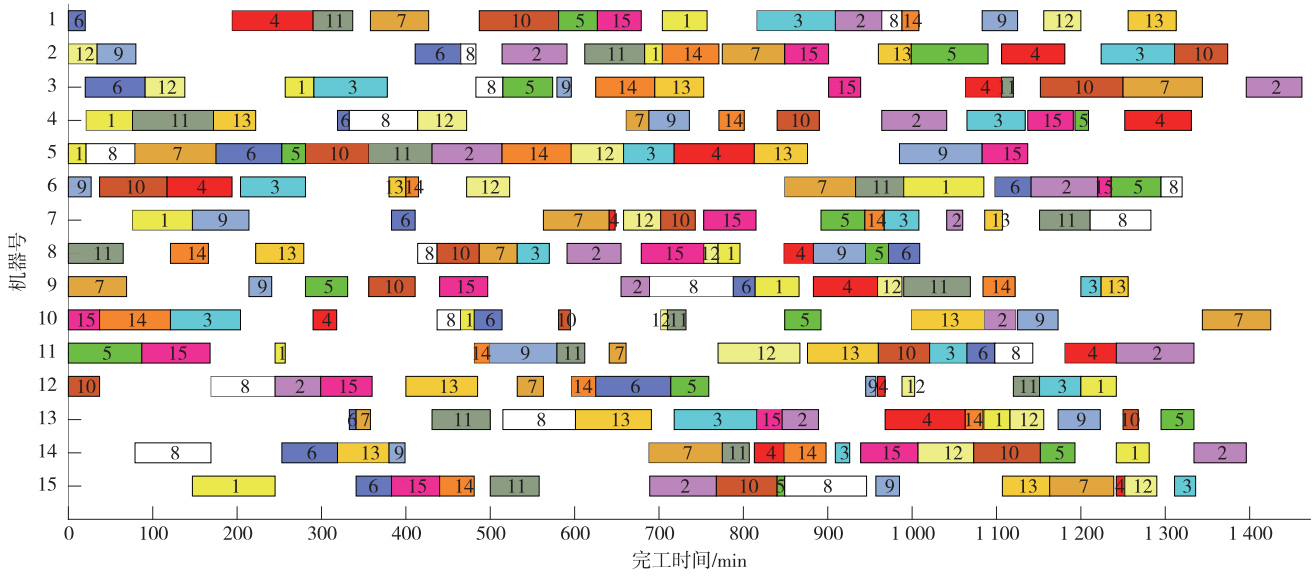
注:完成时间为 1 027 min。

a FT10



注:完成时间为 1 088 min。

b LA16



注:完成时间为 1 462 min。

c LA36

图 6 在部分算例上的甘特图

Fig. 6 The Gantt diagram of some examples

从表 6 可知，在 FT10、LA16、LA36 算例的最优解上，从提前期方面看，HFPSO 在 FT10 上取得了最优，SSA 在 LA16 算例上取得了最优，在 LA36 算例上，HFPSO 和 SSA 均取得了最优提前期；从最大完工时间上看，HCHIO 在 FT10 算例上取得了最优，LA36 算例上为第 2 名；从拖期上看，HCHIO 均取得了最优拖期且相较于其他算法存在较大优势，综合考虑下，HCHIO 在 JIT-JSP 中仍具有较高优越性。

3.3　Friedman 检验

为了进一步验证本文算法与其他算法在统计学上是否存在显著性差异，对各算法在 10 个算例上的平均值整体进行非参数 Friedman 检验，检验结果如表 7 所示。假设 HCHIO 与其他算法样本上不存在显著性区别，若 $p>0.05$，表示接受原假设，即样本之间不存在统计学意义上的差异；若 $p<0.05$，则表示拒绝原假设，即样本之间存在统计学意义上的差异。结果显示 p 值远小于 0.01，说明了 HCHIO 与其他算法的结果存在统计学意义上的差异。秩平均数越小说明该组样本数据在整个样本中相对较小，由于目标函数为最小化问题，所以如果某个样本的秩平均值越小，那么该样本的得分排名就越高，即在各个样本中表现最好。通过该检验验证了本文算法较其他对比算法在解决调度问题上具有更高的可靠性和更强的优越性。

表 7　各算法 Friedman 检验结果

Tab. 7　Friedman test results of each algorithm

算法	个案数	平均值	标准偏差	百分位数(第 50 个)	秩平均值	rank	检验统计
HCHIO	10	2 204.5	2 283.503 5	1 296.5	1.1	1	
CHIO	10	2 966.3	3 316.035 4	1 821	3.65	4	
HFPSO	10	2 721.2	2 964.862 9	1 553.5	3.25	3	
AOSMA	10	3 212.9	3 452.267 5	1 998.5	5.1	5	$p=2.342\ 6e-10$
BWO	10	3 538.3	3 750.424 1	2 344.5	6.8	7	
SSA	10	3 672.2	3 891.756 4	2 540.5	7.75	8	
DBO	10	3 222.9	3 398.281 1	2 228	5.7	6	
QCHIO	10	2 890.1	3 254.719 1	1 768	2.65	2	

4　算法在实例上的验证

为了进一步验证 HCHIO 求解车间调度问题的可行性和有效性，对某公司汽车发动机生产的作业排产问题进行了求解，根据文献[32]整理了某型号的发动机零部件的生产工件工艺和相关数据，各工件的加工顺序如表 8 所示，加工时长如表 9 所示。

表 8　各工件加工工艺

Tab. 8　The processing technology of each workpiece

工序	气缸体	气缸盖衬垫	轴承座	油底壳	气缸盖	飞轮	曲轮轴架	曲轮轴盖	曲轴	正时带上盖
工序 1	粗车加工	粗车加工	粗车加工	粗车加工	粗车加工	粗车加工	粗车加工	粗车加工	粗车加工	粗车加工
工序 2	精车加工	精车加工	精车加工	精车加工	精车加工	精车加工	精车加工	精车加工	精车加工	精车加工
工序 3	镗床	镗床	镗床	铣床	钻床	磨床	铣床	钻床	镗床	铣床
工序 4	铣床	钻床	钻床	磨床	镗床	钻床	镗床	铣床	铣床	钻床
工序 5	钻床	磨床	铣床	镗床	磨床	铣床	钻床	磨床	钻床	磨床
工序 6	磨床	铣床	磨床	钻床	铣床	镗床	磨床	镗床	磨床	镗床

根据表 8 可以看出，需要排产的发动机工件生产过程需要 10 个工件，每个工件需要 6 道工序，为 10×6 规模的问题，解集规模为 $(10!)^6$。运用本文算法及对比算法分别对该问题进行求解，设置种群规模为 50，最大迭代次数为 100，w_1、w_2、w_3 均设置为 1/3，算法独立运行 10 次，记录各算法最优解、平均值和标准差，并记录最优解的最小拖期、最小提前期和最小化最大完工时间，如表 10 所示。最优加工方案对应的甘特图如图 7 所示。

表 9 各工件加工时长

Tab. 9 The processing time of each workpiece min

机床	气缸体 J1	气缸盖衬垫 J2	轴承座 J3	油底壳 J4	气缸盖 J5	飞轮 J6	曲轮轴架 J7	曲轮轴盖 J8	曲轴 J9	正时带上盖 J10
粗加工车 M1	6	4	3	6	6	5	7	4	8	4
精加工车 M2	12	8	9	11	10	11	13	8	13	7
镗床 M3	12	6	8	10	7	5	9	8	10	6
铣床 M4	6	5	7	8	9	7	10	6	8	5
磨床 M5	4	3	3	5	4	8	4	4	9	3
钻床 M6	8	3	5	4	6	3	3	3	4	3
总时长	48	29	35	44	42	39	46	33	52	28

表 10 实例求解结果

Tab. 10 Results of instance solving min

算法	最优值	平均值	标准差	拖期	提前期	完工时间
HCHIO	122.67	135.00	6.48	229	3	136
CHIO	166.33	188.97	10.31	357	0	142
HFPSO	134.67	154.97	14.52	257	11	136
AOSMA	159.67	159.67	0.00	320	24	135
BWO	155.33	158.80	1.83	315	18	133
SSA	159.67	159.67	0.00	320	24	135
DBO	144.33	162.17	10.08	284	10	139
QCHIO	140.33	153.54	6.43	289	3	129

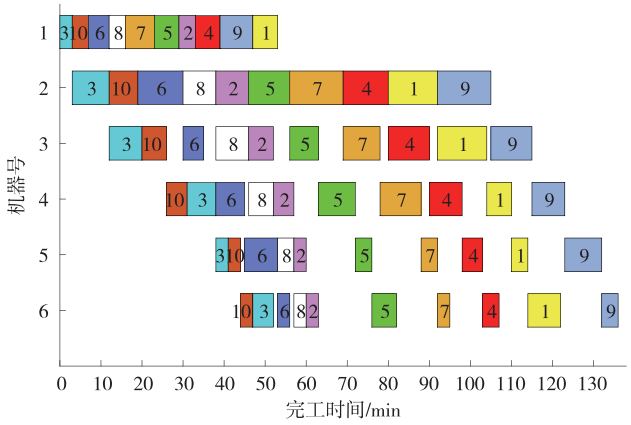


图 7 最优加工方案对应的甘特图

Fig. 7 The Gantt chart corresponding to the optimal processing plan

所得最优加工序列为[3,10,3,10,6,8,7,6,8,5,2,3,10,2,5,10,6,3,4,8,6,8,7,2,3,10,4,9,1,10,3,1,5,7,6,6,8,2,9,2,4,1,5,7,5,4,1,9,9,8,7,4,2,5,7,4,1,9,1,9],在所有算法中,HCHIO 的最优解和平均值均表现最优,最优解的加工总时长 136 min,最小拖期为 229 min,最小提前期为 3 min,特别是在拖期目标上远优于其他算法。

5　结束语

随着全球制造业的发展以及竞争的日益加剧，面向 JIT 的车间调度在提高生产车间效率方面扮演着关键的角色。本文构建了以最小化拖期、最小化提前期和最小化最大完工时间为目标的 JIT-JSP 模型，针对该模型设计了 HCHIO，并在 10 个经典算例与 1 个实际问题上进行了实验，得出以下结论：

HCHIO 具有较强的全局搜索能力、寻优精度等特点。通过算子库策略，算法在迭代过程中自我学习进化并能够选择出适合当前问题的算子，针对最优解进行的多邻域局部搜索策略，有效避免了算法陷入局部最优，提升了算法的局部搜索能力。

本文所研究对象为基于作业车间调度问题的调度模型，作为实际生产的简化模型，除本文实例中的发动机零部件生产，还有弹簧、金属冲压件等生产调度都符合作业车间调度问题的特点，将本文算法用于求解这些行业的调度问题将是下一步的研究方向。

参考文献：

[1] Yazdani M，Aleti A，Khalili S M，et al. Optimizing the sum of maximum earliness and tardiness of the job shop scheduling problem[J]. Computers & Industrial Engineering，2017，107：12-24.

[2] 亓祥波，王宏伟，王浩毅，等. 面向饲料加工的排产优化方法研究[J]. 重庆师范大学学报（自然科学版），2024，41（2）：65-73.
Qi Xiangbo，Wang Hongwei，Wang Haoyi，et al. Study on production scheduling optimization method for feed processing [J]. Journal of Chongqing Normal University (Natural Science)，2024，41 (2)：65-73.

[3] 陈照辉，刘海杨，夏倩，等. 改进算术优化算法求解考虑机器老化效应和工件释放时间的作业车间调度问题[J]. 重庆师范大学学报（自然科学版），2024，41（2）：56-64.
Chen Zhaohui，Liu Haiyang，Xia Qian，et al. Improved arithmetic optimization algorithm for solving job shop-scheduling problems with machine aging effects and workpiece release time[J]. Journal of Chongqing Normal University (Natural Science)，2024，41 (2)：56-64.

[4] 亓祥波，赵品威，王润. 基于量子计算和威布尔分布的混合 CHIO 算法求解 JSP 问题[J]. 制造技术与机床，2024（3）：178-187.
Qi Xiangbo，Zhao Pinwei，Wang Run. Solving the JSP problem using a hybrid CHIO algorithm based on quantum computing and Weibull distribution[J]. Manufacturing Technology & Machine Tool，2024 (3)：178-187.

[5] Baptiste P，Flamini M，Sourd F. Lagrangian bounds for just-in-time job-shop scheduling[J]. Computers & Operations Research，2008，35 (3)：906-915.

[6] 王大志，刘士新，郭希旺. 求解总拖期时间最小化流水车间调度问题的多智能体进化算法[J]. 自动化学报，2014，40 (3)：548-555.
Wang Dazhi，Liu Shixin，Guo Xiwang. A multi-agent evolutionary algorithm for solving total tardiness permutation flow-shop scheduling problem[J]. Acta Automatica Sinica，2014，40 (3)：548-555.

[7] 涂雪平，李铁克，施灿涛. 基于准时制生产的订单动态产品替补模型及算法[J]. 计算机集成制造系统，2016，22（7）：1670-1678.
Tu Xueping，Li Tieke，Shi Cantao. Dynamic substitute model and algorithm for just-in-time production[J]. Computer Integrated Manufacturing Systems，2016，22 (7)：1670-1678.

[8] Kurdi M. An improved island model memetic algorithm with a new cooperation phase for multi-objective job shop scheduling problem[J]. Computers & Industrial Engineering，2017，111：183-201.

[9] Zhang R，Chiong R. Solving the energy-efficient job shop scheduling problem：a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption[J]. Journal of Cleaner Production，2016，112：3361-3375.

[10] 裴小兵，王诗慧. 基于博弈人工蜂群算法的多目标车间调度研究[J]. 武汉大学学报（工学版），2023，56（3）：362-370.
Pei Xiaobing，Wang Shihui. Research on multi-objective shop scheduling based on game artificial bee colony algorithm[J]. Engineering Journal of Wuhan University，2023，56 (3)：362-370.

[11] 史双元，熊禾根. 考虑外协的作业车间无拖期调度问题多目标差分进化算法[J]. 计算机集成制造系统，2024，30 (12)：4352-4368.

Shi Shuangyuan, Xiong Hegen. Multi-objective differential evolution algorithm for no-tardiness job shop scheduling problem with outsourcing option[J]. Computer Integrated Manufacturing Systems, 2024, 30 (12): 4352-4368.

[12] 魏光艳，叶春明. 混合分布估计算法求解分布式柔性作业车间调度问题[J]. 运筹与管理，2024，33 (8)：51-57.
Wei Guangyan, Ye Chunming. Hybrid estimation of distribution algorithm for distributed flexible job shop scheduling[J]. Operations Research and Management Science, 2024, 33 (8): 51-57.

[13] Al-Betar M A, Alyasseri Z A A, Awadallah M A, et al. Coronavirus herd immunity optimizer (CHIO) [J]. Neural Computing and Applications, 2021, 33 (10): 5011-5042.

[14] Dalbah L M, Al-Betar M A, Awadallah M A, et al. A modified coronavirus herd immunity optimizer for capacitated vehicle routing problem[J]. Journal of King Saud University. Computer and Information Sciences, 2022, 34 (8): 4782-4795.

[15] Yan C H, Razmjooy N. Kidney stone detection using an optimized Deep Believe network by fractional coronavirus herd immunity optimizer[J]. Biomedical Signal Processing and Control, 2023, 86: 104951.

[16] Kumar C, Magdalin Mary D, Gunasekar T. MOCHIO: a novel Multi-Objective Coronavirus Herd Immunity Optimization algorithm for solving brushless direct current wheel motor design optimization problem[J]. Automatika, 2022, 63 (1): 149-170.

[17] 王永明，尹红丽，秦开大. 作业车间调度理论及其优化方法研究[M]. 北京：科学出版社，2013.
Wang Yongming, Yin Hongli, Qin Kaida. Research on job shop scheduling theory and its optimization method[M]. Beijing: Science Press, 2013.

[18] Zhong C T, Li G, Meng Z. *Beluga* whale optimization: a novel nature-inspired metaheuristic algorithm[J]. Knowledge-Based Systems, 2022, 251: 109215.

[19] Xue J K, Shen B. A novel swarm intelligence optimization approach: sparrow search algorithm[J]. Systems Science & Control Engineering, 2020, 8 (1): 22-34.

[20] Mirjalili S. SCA: a Sine Cosine Algorithm for solving optimization problems[J]. Knowledge-Based Systems, 2016, 96: 120-133.

[21] 高洪元，刁鸣. 量子群智能及其在通信技术中的应用[M]. 北京：电子工业出版社，2016.
Gao Hongyuan, Diao Ming. Quantum swarm communication application[M]. Beijing: Publishing House of Electronics Industry, 2016.

[22] Sahoo S K, Sharma S, Saha A K. A novel variant of moth flame optimizer for higher dimensional optimization problems [J]. Journal of Bionic Engineering, 2023, 20 (5): 2389-2415.

[23] Meng K, Chen C, Xin B. MSSSA: a multi-strategy enhanced sparrow search algorithm for global optimization[J]. Frontiers of Information Technology & Electronic Engineering, 2022, 23 (12): 1828-1847.

[24] Yang X S. Firefly algorithm, stochastic test functions and design optimisation[J]. International Journal of Bio-Inspired Computation, 2010, 2 (2): 78-84.

[25] Hussain K, Salleh M N M, Cheng S, et al. On the exploration and exploitation in popular swarm-based metaheuristic algorithms[J]. Neural Computing and Applications, 2019, 31 (11): 7665-7683.

[26] Wolpert D H, Macready W G. No free lunch theorems for optimization[J]. IEEE Transactions on Evolutionary Computation, 1997, 1 (1): 67-82.

[27] Shukla A, Pandey H M, Mehrotra D. Comparative review of selection techniques in genetic algorithm[C]//2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE). IEEE, 2015: 515-519.

[28] 潘全科，王文宏，朱剑英，等. 基于粒子群优化和变邻域搜索的混合调度算法[J]. 计算机集成制造系统，2007，13 (2)：323-328.
Pan Quanke, Wang Wenhong, Zhu Jianying, et al. Hybrid heuristics based on particle swarm optimization and variable neighborhood search for Job Shop scheduling[J]. Computer Integrated Manufacturing Systems, 2007, 13 (2): 323-328.

[29] Aydilek İ B. A hybrid firefly and particle swarm optimization algorithm for computationally expensive numerical problems [J]. Applied Soft Computing, 2018, 66: 232-249.

[30] Naik M K, Panda R, Abraham A. Adaptive opposition slime mould algorithm[J]. Soft Computing, 2021, 25 (22): 14297-14313.

[31] Xue J K, Shen B. Dung beetle optimizer: a new meta-heuristic algorithm for global optimization[J]. The Journal of

Supercomputing, 2023, 79 (7): 7305-7336.

[32] 王宁. 基于改进蚁群算法的 Job-shop 车间调度研究[D]. 大连：大连海事大学，2021.
Wang Ning. Job shop scheduling based on improved ant colony algorithm[D]. Dalian: Dalian Maritime University, 2021.

Special Section on Modern Sorting Theory and Job Scheduling Optimization

Application of Self-Learning Hybrid CHIO Algorithm in Just-In-Time Job Shop Scheduling

QI Xiangbo[1], ZHAO Pinwei[1], SONG Yan[2], WANG Run[1]

(1. School of Mechanical Engineering, Shenyang University, Shenyang 110044;
2. School of Physics, Liaoning University, Shenyang 110036, China)

Abstract: With the increase in customer customization demands and the focus on delivery times, just-in-time production has become a key factor in enhancing enterprise competitiveness, making job shop scheduling for just-in-time production worthy of in-depth study. To address the issues of severe delays and low just-in-time performance in job shop scheduling, a job shop scheduling model is proposed with the objectives of minimizing delays, minimizing lead times, and minimizing the maximum completion time. To solve this model, a self-learning hybrid CHIO algorithm (HCHIO) based on the coronavirus herd immunity optimizer (CHIO) is proposed. Firstly, a self-learning operator library with a scoring evaluation mechanism is designed, enabling the algorithm to self-learn and select the optimal operator for different problems to enhance the global optimization performance of the algorithm. Secondly, by conducting neighborhood search on the optimal solution, the local search ability of the algorithm is strengthened. Experiments on benchmark tests and real cases were conducted on HCHIO, verifying the excellent optimization ability of the proposed algorithm in solving job shop scheduling problems. The experimental results demonstrate the effectiveness of the self-learning hybrid CHIO algorithm in solving just-in-time job shop scheduling problems.

Keywords: just-in-time production; job-shop scheduling; coronaviruses group immune algorithm; self-learning; operator library

(责任编辑 陈 乔)