文章编号:1671-4229(2024)01-0038-11

基于改进乌鸦搜索算法的动态柔性作业车间调度应用研究

彭 凯,岳 磊,徐 庆,邹 涛
(广州大学 机械与电气工程学院,广东 广州 510006)

摘 要:车间运作中常会面临许多随机事件扰动,这会扰乱原始调度方案,使得车间生产混乱,生产效率降低。文章以随机机器故障可恢复、不可恢复或需较长事件恢复、紧急工件插入以及紧急工件插入时机器发生故障等4种情况作为随机事件,使用一种混合重调度驱动方式来响应随机事件。同时,构建出动态柔性作业车间调度模型,将本用于处理连续性问题的乌鸦搜索算法进行改进,采用基于机器和基于工序的离散编码方式,设计了3种子代生成方法来增强全局搜索能力,以一定概率发生变异避免产生局部最优。另外,采用IG迭代贪婪算法增加算法的开拓能力,采用完工时间偏差以及序列偏差作为评价标准,在多个测试案例中对右移重调度和完全重调度进行比较分析,同时将文章提出的算法与遗传算法GA和差分进化算法DE进行对比。实验证明,在测试不同的调度方法时,所提出的算法具有优越性与高效性。

关键词:动态调度;机器故障;柔性作业车间;乌鸦搜索算法
中图分类号:TP18;TH186 **文献标志码**:A

Research on dynamic flexible job shop scheduling based on improved crow search algorithm

PENG Kai, YUE Lei, XU Qing, ZOU Tao
(School of Mechanical and Electrical Engineering, Guangzhou University, Guangzhou 510006, China)

Abstract: Workshop operations are often faced with many random event perturbations, which can disrupt the original scheduling scheme and cause chaos and productivity loss in the workshop. In this paper, a hybrid rescheduling driver is used to respond to the random events by taking four scenarios as random machine failures that can be recovered, cannot be recovered or take a long time to recover, emergency workpiece insertion, and machine failures that occur during emergency workpiece insertion. Meanwhile, a dynamic flexible job shop scheduling model is constructed, and the crow search algorithm, which is originally used to deal with the continuity problem, is improved by adopting machine-based and process-based discrete coding, and designing the generation method of the three subgenerations to enhance the global searching capability, and avoiding the generation of local optimums with a certain probability of mutation. The IG iterative greedy algorithm is also used to increase the pioneering ability of the algorithm. The completion time deviation and sequence deviation are used as evaluation criteria to compare and analyze the right-shift rescheduling and complete rescheduling in several test cases, and at the same time, the algorithm proposed in this paper is compared with the genetic algorithm GA and differential evolution algorithm DE, and the experiments prove that the algorithm pro-

收稿日期:2023-09-21;**修回日期**:2023-12-21
作者简介:彭凯(1999—),男,硕士研究生. E-mail:1161245326@qq.com
引文格式:彭凯,岳磊,徐庆,等. 基于改进乌鸦搜索算法的动态柔性作业车间调度应用研究[J]. 广州大学学报(自然科学版),2024,23(1):38-48.

posed in this paper has the superiority and high efficiency when testing different scheduling methods.
Key words: dynamic scheduling; machine failure; flexible job shop; crow search algorithm

柔性作业车间调度问题(Flexible Job Shop Scheduling Problem, FJSP)是经典的 NP-HARD 问题,其难点在于要在生产资源受限的条件下,通过排列一批工件的加工顺序和为每个工件指派加工机器来最大化达到调度目标。在过去一段时间,绝大部分学者都将目光聚焦在静态车间调度问题上,此类问题的求解需要假设许多条件。而车间运转过程中常常遇到机器故障、订单插队等随机事件,这些事件会打乱原始的调度方案,导致调度方案不再是最优解,甚至造成生产混乱。为了减少损失,则需要采用新的调度方案。当意外事件发生时,此时问题转化为动态柔性作业车间调度问题(Dynamic Flexible Job Shop Scheduling Problem, DFJSP)[1-2]。

近年来,诸多学者投身于复杂动态车间调度模型问题的求解,其中一部分学者致力于对车间模型动态事件的构建。Al-Hinai 等[3]将随机机器发生故障作为动态事件,提出一种两阶段的混合遗传算法,先对 makespan 进行静态调度优化,在此基础上再进行双目标函数的动态模型优化,提出了一种分阶段求解的思路,使得第二阶段求解效率大幅度提升。Ning 等[4]模拟在加工过程中有紧急工件插入的情况,提出一种改进的混合多阶段量子粒子群算法,将粒子群算法的更新机制进行改进,最终在测试集上展现出该算法的优越性,其更新机制可以使得算法收敛速度加快。Zadeh 等[5]提出可变加工时间的动态模型,指出作业的处理时间可能和估计时间存在偏差,并基于人工蜂群的思想提出一种启发式模型,不过该算法过于依赖预估时间这一特征,其鲁棒性不够。以上案例提供了各种不同的车间动态事件,为模型的建立提供了新思路。

除了对动态车间调度模型的研究以外,还有部分学者致力于研究多目标的动态调度优化。Zeiträg 等[6]为了更好地逼近非支配前沿,提出一种基于多目标遗传规划的超启发式规则,采用代理模型来近似适应度,以新工件插入作为随机事件,最终在测试集上证明了所得到的 Pareto 前沿的收敛性与多样性,为多目标解的优越性提供了一种新的对比思路。Chen 等[7]采用 NSGA-II 对具有随机机器故障的多目标动态柔性车间调度问题进行求解,以 makespan 和机器总工作量为目标,设计了最优自适应调度策略,证明了算法对于减少机器故障所带来影响的有效性,为调度策略的选择提供了新思路。Liu 等[8]采用一种改进的基于分解的多目标进化算法(IMOEA/D),以模糊加工时间作为动态事件进行求解,使用 3 种进化策略和变邻域搜索,最终证明了算法的收敛性与多样性。

随着算法的不断发展,群智能算法因其有着易于实现、鲁棒性强等诸多优点不断地被学者研究。如刘微等[9]结合灰狼算法和差分算法各自的特点对灰狼算法进行改进,提出一种混合灰狼算法用于求解 DFJSP,用 4 种随机事件作为扰动因素,有效地证明了算法的鲁棒性。对于开放式车间的动态调度问题,刘乐等[10]将随机机器出现故障作为动态随机事件,设置了 3 种重调度方法,并通过比较它们的性能来确定在特定情况下采用哪种调度方案更优,为动态车间调度提供了另一种优化角度。王艳等[11]设计出改进的多目标差分进化算法,使用 Nash 均衡计算出熵权法的总权重,将其和 TOPSIS 的评价体系相集合,多种不同的调度方案进行评价,证明了该算法求解多目标问题的先进性。孙丽珍等[12]结合启发式规则产生初始种群,同时引入工件批量加入和机器故障的特征,设计基于多约束贪婪插入的解码方式,最后通过标准数据集测试验证了算法的可行性。以上案例都说明群智能算法在求解车间调度问题上实效性很强。

随着群智能算法的不断发展,越来越多基于自然现象的群智能算法不断被提出,乌鸦搜索算法就是其中之一。乌鸦搜索算法(Crow Search Algorithm,CSA)是 Askarzadeh[13]于 2016 年首次提出的一种群智能优化算法,该算法的思想主要是通过模拟自然界中乌鸦的部分习性而设计的,乌鸦具有记忆、交流、储存食物以及窃取食物等社会性行为。

相较于其他经典群智能优化算法,乌鸦搜索算法有着参数少、鲁棒性强、扩展性好和易于实现的优势。该算法在图像分割、故障诊断、特征提取

等领域已成功应用,展现出其优化的潜力。Kumar等[14]将CSA算法使用在云计算的任务调度模型上,通过找到最适合的虚拟机(Virtual Machine, VM),最大程度地缩短了任务完成时间。Zhang等[15]将CSA算法用于微电网的多目标调度模型中,针对其在高维目标下开发能力差的情况,提出了两种改进的开发模型,最终将微电网的总运营成本降低了21.5%。通常情况下,研究CSA的学者更关注其在连续问题上的应用。近年来,一些学者逐步将其用于传统静态车间调度问题中,如闫红超等[16]设计了一种混合乌鸦搜索算法,采用局部搜索增强的方式,用于求解置换流水车间的调度问题,但鲜少有动态车间调度下的相关应用研究或报道。

因此,本文以柔性作业车间为车间模型,设计出一种改进的乌鸦搜索算法(Improved Crow Search Algorithm, ICSA),该算法考虑了4种突发事件,通过采取两种不同的重调度方案,来对比分析各自的优劣,为该调度模型提供了不同的调度选择。

1 问题描述

动态柔性作业车间调度问题关键在于其场景更为复杂,引入了随机扰动事件或干扰因素[1-2]。本文主要以随机机器故障可恢复、不可恢复或需较长事件恢复、紧急工件插入以及紧急工件插入时机器发生故障等来作为扰动因素。该问题可以被描述为有 n 个工件 $J=\{J_1,J_2,J_3,\cdots,J_n\}$,$m$ 台机器 $M=\{M_1,M_2,M_3,\cdots,M_m\}$,其中,每个工件有 j 道工序,每道工序需要从可选机器集合中选择1台进行加工。在加工过程中,如果随机事件发生,则需要重新对调度方案进行调整来最小化损失。

下面对DFJSP作出以下假设:

(1)在初始时刻,任意机器都可用;

(2)每台机器只能同时加工一个工件;

(3)机器加工过程不可中断,若该机器发生故障,则该工件需要重新分配至其他机器上加工。

(4)机器故障后可修复,且修复的时长是已知的。

如公式(1)所示,本文以最小化最大完工时间*makespan*作为目标函数。

$$\min C=\max_{1\leqslant i\leqslant n}\{C_i\}, \tag{1}$$

其中,C_i 为第 i 个工件的完工时间。

表1是两个工件和6台机器的加工信息[17]。表中每一行表示当前工序在各个机器上的加工时间,"-"符号表示当前工件的当前工序不可以在这台机器上加工。

表1 加工信息表

Table 1 Processing information table

工件	工序	加工时间					
		M_1	M_2	M_3	M_4	M_5	M_6
J_1	O_{11}	2	-	5	-	4	-
	O_{12}	-	3	-	5	-	2
	O_{13}	5	-	3	6	-	4
J_2	O_{21}	-	4	-	-	5	4
	O_{22}	3	-	5	5	-	5
	O_{23}	-	5	-	6	4	-

2 改进的乌鸦搜索算法

2.1 乌鸦搜索算法

乌鸦搜索算法是通过模拟自然界中乌鸦的习性而设计的,乌鸦具有记忆能力,能够记住食物存放的位置,也具有一定的沟通能力,能够将食物存放的位置分享给同伴,会通过散布假信息来欺骗同伴,乌鸦还会跟随在同伴身后,伺机偷取对方的食物,并将自己的食物储藏起来[18]。

该算法主要依据4个基本规则:

(1)乌鸦是群居生活的动物;

(2)乌鸦有着出色的记忆能力,可以记住食物储藏的位置;

(3)乌鸦会跟随同伴,等待机会窃取其食物;

(4)乌鸦可以通过欺骗的方式,保护自己的食物不被窃取。

假设乌鸦种群数量为 N,乌鸦在 D 维空间内活动。因此,在 t 时刻乌鸦 i 的位置是 $x_i^t=[x_{i,1}^t, x_{i,2}^t, x_{i,3}^t,\cdots,x_{i,D}^t]$,其中,$i\in[1,2,\cdots,N]$,$t\in[1,2,\cdots,D]$。

根据基本规则(3),乌鸦 c 在 t 时刻要访问藏食地点 m_c^t,乌鸦 i 跟随乌鸦 c 以窃取其食物,若未被发现,则乌鸦 i 按照公式(2)更新自身位置:

$$x_i^{t+1}=x_i^t+r_i\times fl\times(m_c^t-x_i^t), \tag{2}$$

其中,r_i 为(0,1)中的随机数,fl 为乌鸦飞行的欧式距离,$c\in\{1,2,\cdots,N\}$ 且 $c\neq i$。

根据规则(4),如果乌鸦 c 发觉被乌鸦 i 跟随,

为了保护食物,则会飞向一个随机的位置以欺骗乌鸦 i。所以,乌鸦 i 按照公式(3)更新自身位置:

$$x_i^{t+1}=\begin{cases}x_i^t+r_i\times fl\times(m_c^t-x_i^t), & \text{if } r_c\geqslant AP,\\ \text{a random position, otherwith,}\end{cases} \tag{3}$$

其中,AP 为乌鸦的认知概率,r_c 为(0,1)随机数。

乌鸦 i 按照公式(4)更新食物的位置:

$$m_i^{t+1}=\begin{cases}x_i^{t+1}, & \text{if } f(x_i^{t+1}) \text{ is better than } f(m_i^t),\\ m_i^t, & \text{otherwith}\end{cases} \tag{4}$$

其中,$f(x_i^{t+1})$ 表示 $t+1$ 时刻乌鸦 i 的适应度值,初始时刻,有 $m_i^0=x_i^0$。

CSA 的算法流程如图 1 所示:

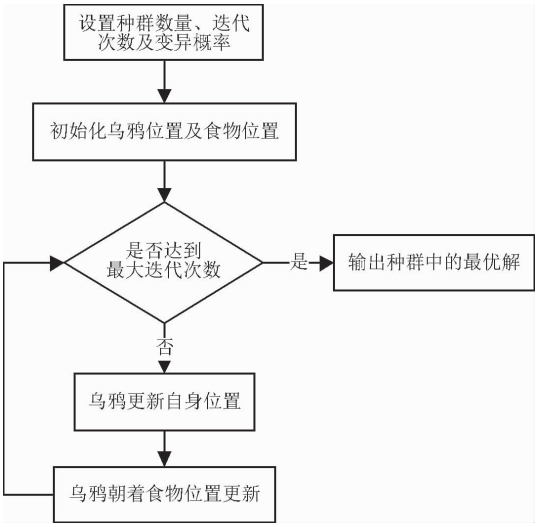


图 1　CSA 算法流程图

Fig. 1　Flow chart of crow search algorithm

最初的乌鸦搜索算法用于处理连续型问题,该算法在迭代过程中生成解的方式都是采用连续型变量的计算方式,然而车间调度问题是离散的组合优化问题,故需要对算法进行改进[16]。本文重新定义了乌鸦位置 $x_i^t=[x_{i,1}^t,x_{i,2}^t,x_{i,3}^t,\cdots,x_{i,D}^t]$,将其定义为工件加工顺序,如[2,3,1,5,4]表示依次加工工件的编号。在最初的乌鸦搜索算法中,乌鸦位置的更替是根据乌鸦和食物、乌鸦和乌鸦之间的位置距离来更新的[19],由于将连续性问题离散化,因此,CSA 算法中后续步骤全部要进行修改。

2.2　编码方式

本文使用双层编码,以适应动态柔性作业车间。个体编码由两个序列串组成,分别是操作顺序序列(OS)和机器选择序列(MS),编码方式见图 2。

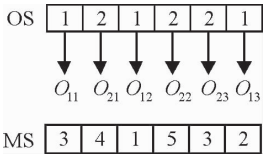


图 2　编码方式

Fig. 2　Coding method

如图 2 所示,其中,OS 的长度与每个工件的工序数相等,OS 中的数字表示工件的序号,从左往右出现的次数表示该工件的第几道工序;MS 的长度与 OS 相同,其中的数字表示当前工件的当前工序需要在第几台机器上加工。

2.3　解码方式

对于种群中的个体,为了进行后续的操作,需要对个体进行解码操作,才能获得相关信息。读取 OS 列表,通过对应位置在 MS 中确定它的加工机器,再通过数据表读取加工时间。

根据图 2 的调度方案进行解码,可以得到在表 2 中看到的工件处理信息,包括开始处理及完成处理时间等。

表 2　解码信息表

Table 2　Decoding information table

OS	工序	MS	机器编号	加工时间
1	O_{11}	3	M_3	5
2	O_{21}	4	M_4	2
1	O_{12}	1	M_1	5
2	O_{22}	5	M_5	4
2	O_{23}	3	M_3	3
1	O_{13}	2	M_2	2

2.4　初始化方式

为了增加算法的收敛速度,本文采用 3 种操作来为种群初始化,分别是全局选择(Global Selection,GS)、局部选择(Local Selection,LS)和随机选择(Random Selection,RS)。GS 可以增大初始种群在解空间中的覆盖率,LS 主要聚焦于局部位置,RS 可以使算法具有随机性,可能会覆盖到部分难以搜索到的位置。

LS 的操作步骤如下所示:

Step1:按工件顺序选择一个工件;

Step2:执行这个工件的每一步工序;

Step3:选择负荷最小的机器;

Step4:在每个工件加工完成以后,将机器负荷归0;

Step5:循环执行Step1至Step4,直至所有工件均分配完成。

RS的操作步骤如下所示:

Step1:随机选择一个工件;

Step2:执行这个工件的每一步工序;

Step3:选择负荷最小的机器;

Step4:循环执行Step1至Step3,直至所有工件均分配完成。

在动态调度和静态调度中,初始化方法有些许不同(图3)。在动态调度中,已经进行加工的工件工序顺序不会被改变。当发生随机事件时,为了将尚未完成的工件工序排列成最优顺序,需要进行重新调度。因此,编码可以分为两个部分,即已经完成的工序和需要重新开始的工序。这样可以确保在重新调度时,只对尚未完成的工序进行调整。

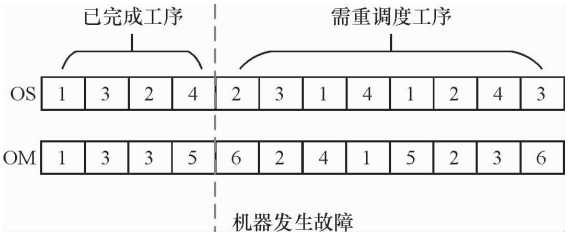


图3 初始化方法

Fig. 3 Initialization method

2.5 交叉方式

编码方式分为OS和MS两种,因此也需要单独进行交叉操作,OS采用PPX交叉方式,MS采用Complete Crossover交叉方式,两者都可以采用Swap-Insert-Reverse交叉方式。

2.5.1 PPX交叉操作

PPX交叉方法如图4所示,具体步骤如下:

Step1:随机生成一段由01数字组成且长度与OS相等的序列串;

Step2:遍历该序列串,当遇到数字1时执行步骤3,当遇到数字0时执行步骤4;

Step3:从父代1中选取第一项,并将其赋值给与序列串下标相同的子代序列串位置,再将父代1和父代2中此数字删除;

Step4:从父代2中选取第一项,并将其赋值给与序列串下标相同的子代序列串位置,再将父代1和父代2中此数字删除;

Step5:重复执行Step2至Step4,直到遍历完整个01序列串。

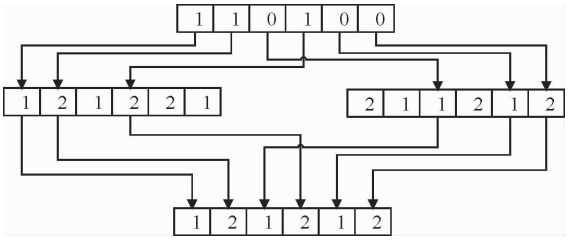


图4 PPX交叉方法

Fig. 4 PPX crossover method

2.5.2 Complete Crossover交叉操作

Complete Crossover(CCX)交叉方式如图5所示,操作步骤如下:

Step1:随机生成一个与目标序列串长度相等且由随机01元素组成的子序列;

Step2:遍历该序列,当遇到数字1时执行步骤3,当遇到数字0时执行步骤4;

Step3:从父代1中选取对应下标的数字,并将其赋值给与01序列串下标相同的子代序列串位置;

Step4:从父代2中选取对应下标的数字,并将其赋值给与01序列串下标相同的子代序列串位置;

Step5:重复执行步骤2至步骤4,直到遍历完整个01序列串。

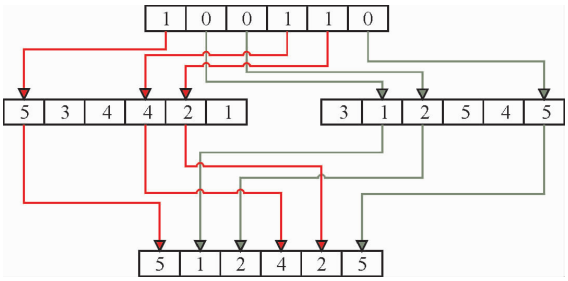


图5 Complete Crossover交叉方式

Fig. 5 Complete Crossover crossover method

2.5.3 Swap-Insert-Reverse交叉操作

Swap-Insert-Reverse(SIR)交叉方式见图6,操作的具体步骤如下:

Step1:随机生成两个不同的下标,将其对应元素交换;

Step2:随机选择一个元素和一个位置,将元素插入到该位置;

Step3:随机选择一段子序列串,将其逆序。

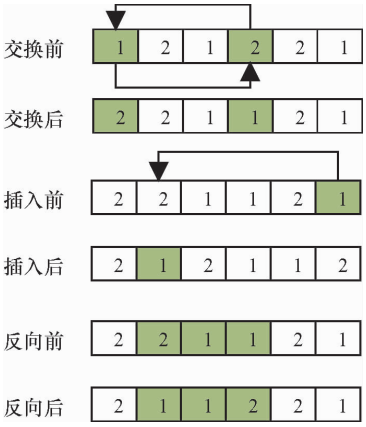


图 6　Swap-Insert-Reverse 交叉方式

Fig. 6　Swap-Insert-Reverse crossover method

2.5.4　变异操作

变异操作如图 7 所示,在需要进行重调度的序列段中,随机挑选两个不重叠的子序列,交换其对应下标的元素。

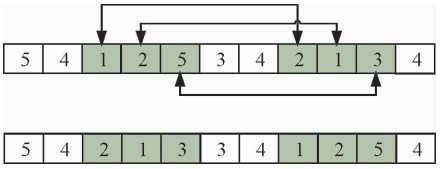


图 7　变异操作

Fig. 7　Mutation operation

2.6　局部搜索

为了改善乌鸦搜索算法中局部搜索不充分以及解空间挖掘不够深的情况,本文采取 IG 迭代贪婪的局部搜索方法,具体操作如下:

Step1:从一个完整的工件排序 π 中随机去除 d 个工件,从而产生两个子序列,包含 d 个被去除工件的子序列 π_R 和包含 $n-d$ 个剩余工序的子工序 π_D;

Step2:将 π_R 中的工件依次插入 π_D 中,最终获得完整的工件排序 π';

Step3:遍历 π'的每一个可插入位置,选择适应度最高的位置插入;

Step4:若最终 π'的适应度值更高,则更新解,否则就放弃 π'。

2.7　ICSA 算法流程

改进的乌鸦搜索算法如图 8 所示。当加工过程中发生突发事件时,算法会将未完成的工序进行重排,算法具体流程为:初始化乌鸦和食物的位置,将种群中的解使用 3 种不同的子代生成方式生成新解,同时对于种群中适应度值为前 1/10 的种群采取 IG 迭代贪婪搜索方式,增强局部搜索。为了避免算法陷入局部最优,在局部搜索完成以后,再以 pc 的概率发生变异,以求跳出局部解。最终进行优胜劣汰法则,更新食物的位置。

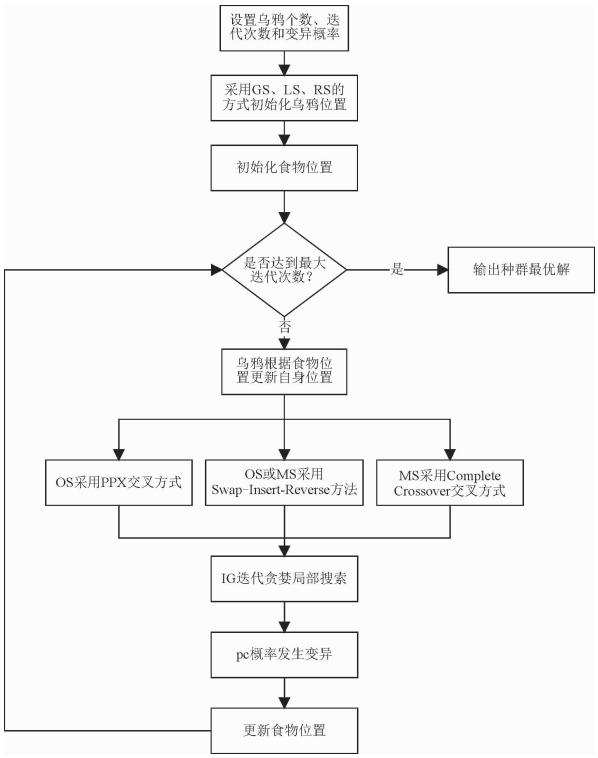


图 8　改进 CSA 算法流程图

Fig. 8　Flowchart of the improved CSA algorithm

3　重调度的驱动策略及方法

3.1　重调度的驱动策略

重调度方案的启动往往是因为在执行原始方案的过程中出现了突发事件,此时应根据不同的突发事件作出相对应的重调度策略。

基于扰动事件发生而启动的事件性驱动,其优点是实时性更高,即第一时间就能做出反应。但缺点是会堆积许多隐藏的不合理调度,若遇到连续突发事件,则会不断触发重调度,使得策略的稳定性下降。

周期性驱动的重调度机制在实际应用中易于实现,但可能无法选择合适的周期值,且对突发事件响应速度较慢。

综合考量上述两种驱动方式,结合其各自的优缺点,本文采用一种混合驱动策略,如图 9 所

示,即当未发生突发事件时使用周期重调度,突发事件发生时触发事件性重调度,这种方法可以增加制造过程中的稳定性,也有能力响应车间突发的随机事件,从而提高制造过程中应对变化的能力。

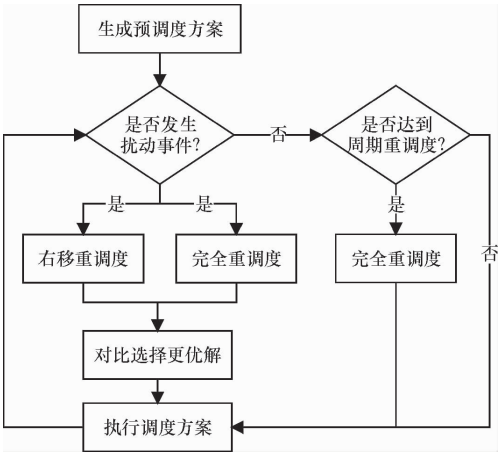


图9 混合重调度流程图

Fig. 9 Hybrid rescheduling flow chart

3.2 重调度的方法

为了降低因突发事件发生而导致的生产成本增加,本文将从以下3种不同的重调度方案来分别介绍。

(1)右移重调度是指受随机事件影响的工件工序开始加工时间向后移动,且其对应的加工机器不发生改变。

(2)局部重调度是指在原计划调度中,不调整各工件的加工工序所对应的加工机器,但需要改变部分加工机器上工件工序的顺序。

(3)完全重调度是指当突发事件发生后,重新组合当前和未加工的工件,生成全新的调度方案。

右移重调度的优点在于响应迅速、执行简单,然而该方法仅仅只是将生产计划向后推迟,这会导致完工时间大幅增加。而局部重调度的优势在于它可以只通过调整部分加工机器上的工件工序来降低完工时间,但是由于各个工件对应的加工机器仍然没有改变,同样也会导致完工时间向后延期。对于完全重调度而言,它将未完成工序全部重排,这会使得完工时间大幅下降,虽然算法复杂度会增加,但对于企业而言可以接受。

3.3 重调度的评价指标

3.3.1 完工时间偏差

当重调度方案启动以后,部分或者全部工件的工序会得到一个新的完成加工时间,本文引入了完工时间偏差来计算结果,即通过将重调度方案中新的完工时间与原始调度方案中新的完工时间做差,得出完工时间偏移量,总计每个受影响的工件完工时间偏移量,将其累加即可得完工时间总偏差量,见公式(5)。

$$t_D = \max_{1\leqslant i\leqslant n} t'_i - \max_{1\leqslant i\leqslant n} t_i, \tag{5}$$

其中,t'_i为重调度以后工件i的完工时间。

3.3.2 序列偏差

在执行重调度方案的过程中,会存在部分工件需要更换加工机器的情况。由于这些工件在原始调度方案中已经被分配给了初始机器,所以需要由人或者搬运小车将这些工件运送至相应的加工机器,在这期间会产生人工成本以及材料运输成本。本文引入序列偏差,通过统计工件工序对应的加工机器发生改变的数量,来衡量重调度方案的优越性,公式(6)表示工件i的工序j对应的加工器是否发生改变。

$$Q_{ij} = \begin{cases} 0, M_{ij} = M'_{ij'}, \\ 1, M_{ij} \neq M'_{ij'}, \end{cases} \tag{6}$$

其中,M_{ij}表示工件i的工序j所指定的加工机器序号,$M'_{ij'}$为重调度以后工件i的工序j所指定的加工机器序号。

公式(7)表示序列偏差评价指标。

$$SD = \sum_{i=1}^{n} \sum_{j=1}^{num(j)} Q_{ij}, \tag{7}$$

其中,$num(j)$为工件总的工序数量。

将其合并为同一个评价指标即可得到公式(8):

$$EVA = w_1 t_D + w_2 / SD, \tag{8}$$

其中,w_1,w_2为权重,且$w_1 + w_2 = 1$。

4 实验及分析

本文使用Python 2.7对动态柔性作业车间问题进行了仿真,运行环境为Intel Core i5 - 9300H CPU @2.4 GHz,RAM 16 GB。

4.1 实验算例设置

采用10×5×6实例进行仿真测试,数据如表3所示,在该测试数据中,一共有10个待加工工件,每个工件有5道加工工序,每道工序最多可在6台机器中选择。其中,会有部分工件不可以在指定机器上加工,在数据中用“-”符号表示,表中其余数字表示机器的加工性能。

表 3　仿真数据集

Table 3　Simulation dataset

工件	工序	加工时间					
		M_1	M_2	M_3	M_4	M_5	M_6
J_1	O_{11}	3	6	2	4	2	5
	O_{12}	5	5	–	5	3	2
	O_{13}	5	2	5	5	6	2
	O_{14}	–	5	4	–	3	6
	O_{15}	2	4	2	2	2	–
J_2	O_{21}	6	2	5	–	3	5
	O_{22}	–	2	6	3	5	5
	O_{23}	6	3	4	3	–	3
	O_{24}	5	4	2	5	6	–
	O_{25}	2	3	–	–	4	2
J_3	O_{31}	–	4	–	–	6	5
	O_{32}	6	–	3	4	4	6
	O_{33}	5	6	5	6	2	5
	O_{34}	3	–	5	5	–	3
	O_{35}	4	6	–	–	–	4
J_4	O_{41}	2	5	–	6	2	3
	O_{42}	6	5	4	5	–	2
	O_{43}	5	2	4	–	6	6
	O_{44}	6	5	–	3	3	6
	O_{45}	3	2	3	6	6	3
J_5	O_{51}	5	6	4	6	4	5
	O_{52}	4	–	6	6	–	2
	O_{53}	5	–	6	3	6	6
	O_{54}	3	5	2	5	4	6
	O_{55}	6	3	6	5	5	4
J_6	O_{61}	5	4	2	6	6	6
	O_{62}	6	4	5	6	2	3
	O_{63}	–	3	6	6	3	2
	O_{64}	6	4	2	–	2	2
	O_{65}	2	5	2	4	3	4
J_7	O_{71}	2	6	3	4	4	2
	O_{72}	3	3	4	6	3	–
	O_{73}	4	–	–	4	5	–
	O_{74}	4	5	5	2	2	4
	O_{75}	5	4	5	3	4	2
J_8	O_{81}	4	–	6	3	6	5
	O_{82}	2	3	2	5	3	2
	O_{83}	–	3	5	–	6	–
	O_{84}	5	6	3	–	–	6
	O_{85}	5	3	6	–	2	5
J_9	O_{91}	–	6	4	–	–	5
	O_{92}	2	–	4	3	3	2
	O_{93}	4	2	2	4	4	–
	O_{94}	3	5	6	4	3	2
	O_{95}	6	2	6	3	6	5
J_{10}	O_{101}	3	–	6	6	2	5
	O_{102}	3	4	2	3	6	–
	O_{103}	5	6	3	5	2	–
	O_{104}	5	4	6	5	2	4
	O_{105}	6	5	4	2	3	3

4.2　算法参数设置

在 ICSA 算法中,种群数量设定为 50 个,最大迭代次数为 500 次,每个个体之间都存在交叉,发生变异的概率 pc = 0.1。本文将两个重调度评价指标的权重设置为 w_1 = 0.9 和 w_2 = 0.1。

本文采用了 4 种不同的动态事件进行实验,如表 4 所示。其中,DE1 表示加工机器发生损坏,但可以修复且修复时间已知;DE2 表示加工机器发生故障,机器不可修复或者修复时间过长;DE3 表示在加工过程中有紧急工件插入;DE4 表示紧急工件插入与机器故障两件随机事件共同发生。

表 4　动态事件

Table 4　Dynamic events

动态事件	动态事件描述
DE1	在 t = 10 时,机器 4 损坏,预计维修时间为 10
DE2	在 t = 15 时,机器 2 损坏,预计不可恢复或需要较长时间恢复
DE3	在 t = 7 时,紧急工件插入,该工件的加工信息如表 5 所示
DE4	在 t = 6 时,有新工件 11 插入,同时机器 2 在 t = 15 的时候发生故障,预计维修时间为 10

表 5　新插入工件加工信息

Table 5　Newly inserted workpiece processing information

工件	工序	加工时间					
		M_1	M_2	M_3	M_4	M_5	M_6
J_{11}	O_{111}	5	4	3	2	–	2
	O_{112}	4	3	5	2	6	4
	O_{113}	3	4	5	6	2	2
	O_{114}	6	4	2	3	–	2
	O_{115}	4	2	3	3	4	2

图 10 展示了原始调度方案中的甘特图。

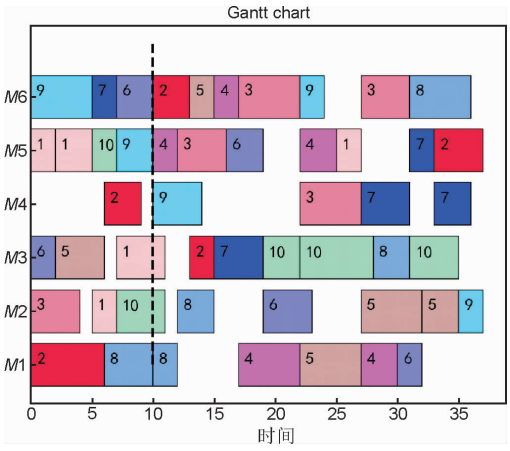
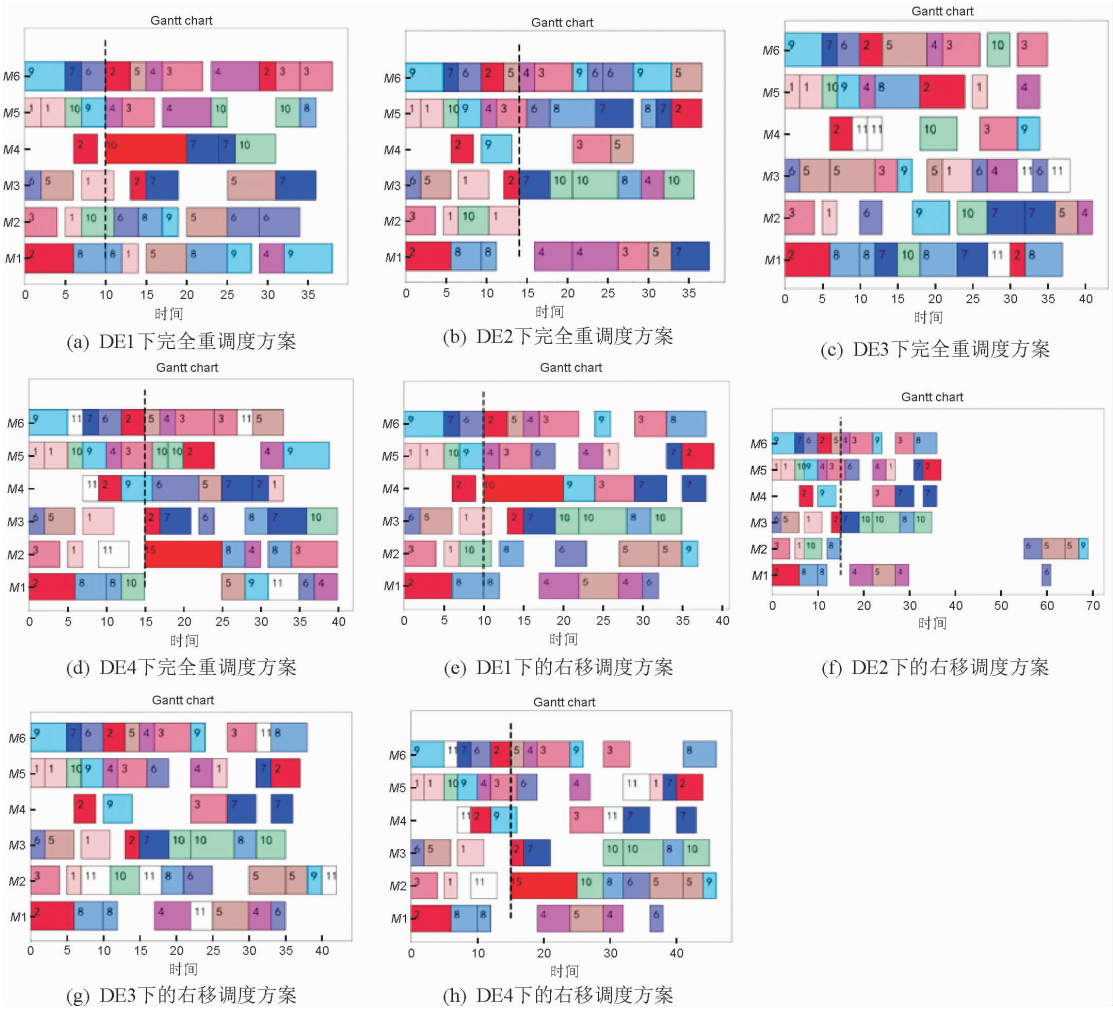


图 10　原始调度方案

Fig. 10　Original scheduling scheme

当车间发生随机事件时,对比分析不同的重调度方法所得到的甘特图,见图 11。表 6 表示在不同的动态事件下,各个重调度结果之间的比较,从实验结果可以分析出,当加工过程中随机扰动事件发生时,采用完全重调度方案获得的结果比采用右移重调度所使用的 makespan 更少,但是完全重调度会大幅度打乱原有的调度安排,导致工件工序对应的加工机器变动比较大,人力运输或者转移资源成本增加。值得注意的是,如果加工过程中,机器发生故障,且该故障需要修复的时间过长或者不可修复,则完全重调度方案远远胜于右移重调度方案。



(a) DE1下完全重调度方案 (b) DE2下完全重调度方案 (c) DE3下完全重调度方案

(d) DE4下完全重调度方案 (e) DE1下的右移调度方案 (f) DE2下的右移调度方案

(g) DE3下的右移调度方案 (h) DE4下的右移调度方案

图 11 不同随机事件下完全重调度与右移重调度方案

Fig. 11 Complete rescheduling scheme and right shift scheduling scheme under different dynamic events

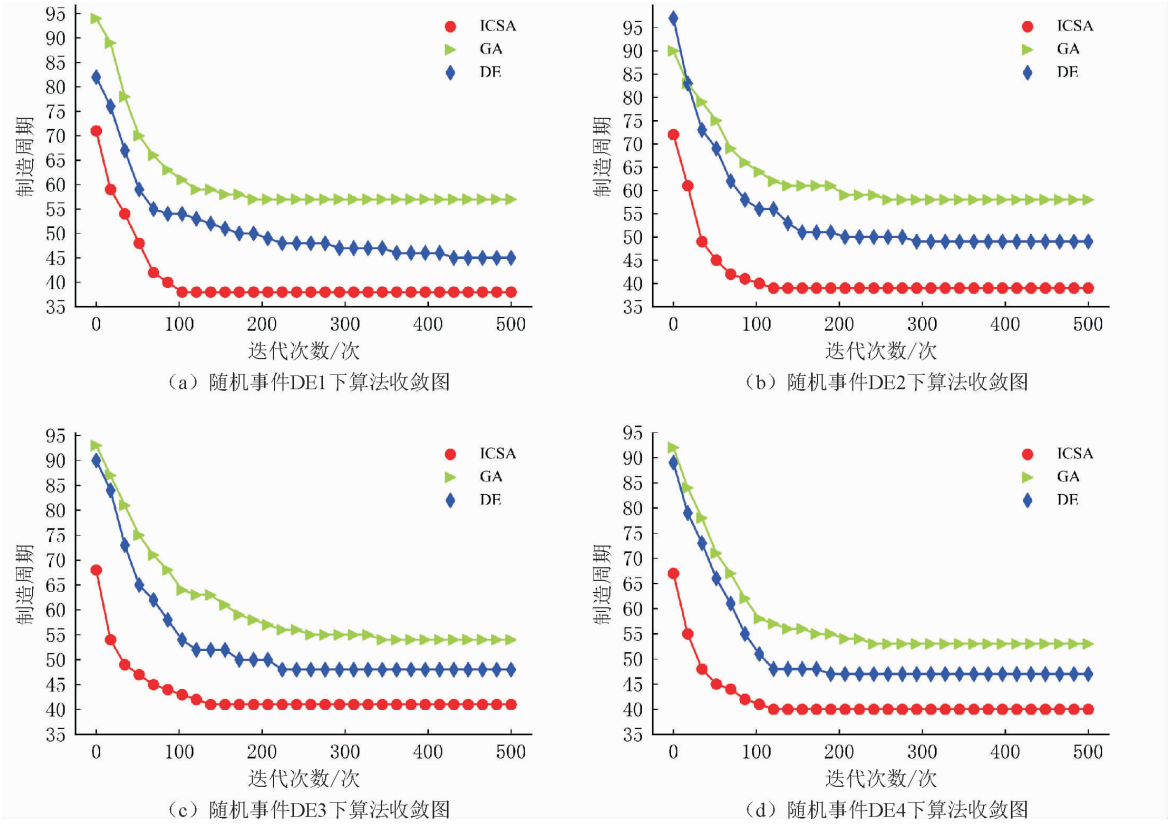
表 6 动态重调度结果比较

Table 6 Comparision of dynamic rescheduling results

动态事件	原始调度	右移重调度			完全重调度		
	makespan	makespan	*SD*	*EVA*	makespan	*SD*	*EVA*
DE1	37	39	26	1.8	38	2	0.95
DE2	37	69	18	28.8	39	1	2.8
DE3	37	42	29	4.5	41	9	3.61
DE4	37	46	27	8.1	40	3	2.73

图 12 是 4 种随机事件发生以后执行完全重调度方案完工时间的算法收敛图。本文将改进的乌鸦搜索算法与经典的遗传算法 GA 和差分进化算法 DE 进行比较,可以看出在最初搜索过程中,

需打乱未完成的工序,所以初始的 makespan 值比较高。随着迭代次数的增加,算法逐渐找到了最优个体,此时算法收敛速度较快。当种群聚集在最优个体附近时,开始进行局部搜索,此时算法的收敛速度放缓。通过不断地交叉、变异、局部搜索操作,算法逐渐收敛到最优解位置。在整个迭代过程中,本文提出的算法在收敛速度、收敛效果上都优于经典的进化算法,验证了算法的优越性。同时,该案例采用的是经典柔性作业车间模型,因此该算法具有很好的推广性。



(a) 随机事件DE1下算法收敛图　(b) 随机事件DE2下算法收敛图

(c) 随机事件DE3下算法收敛图　(d) 随机事件DE4下算法收敛图

图 12　算法收敛图

Fig. 12　Convergence diagram of the algorithm

5　结　论

本文提出了一种混合重调度驱动方案来处理 DFJSP,同时将乌鸦搜索算法进行改进,针对其收敛速度慢的缺点,提出了 3 种初始解的生成方式增加收敛速度;针对算法开拓能力差的缺点,提出了一种 IG 迭代贪婪的局部搜索方法;为了克服容易陷入局部最优的缺点,采用了 3 种交叉方式和变异操作,以使其能够跳出局部最优。最后,通过测试数据对 4 种模型进行求解,将该算法与 GA 和 DE 算法进行对比,根据求解结果可知,当机器发生故障且修复时间不长时,若该机器上的待处理工件数量不多,则采用右移重调度和完全重调度对于完工时间的影响差别不大,但是对于工序的序列偏差差别很大,因此,企业需要根据自身的利益对方案做出选择。当机器发生故障且需要较长时间修复时,此时采取完全重调度是更优的选择。而对于新工件插入而言,采取完全重调度可以比右移重调度所需要的完工时间更短。

参考文献:

[1] 尤一琛, 王艳, 纪志成, 等. 基于随机机器故障的柔性作业车间动态调度[J]. 江苏大学学报(自然科学版), 2021, 42(6): 648-654,714.

[2] 陈超,王艳,严大虎,等. 面向能耗的柔性作业车间动态调度研究[J]. 系统仿真学报, 2017, 29(9): 2168-2174, 2181.

[3] Al-Hinai N, Elmekkawy T Y. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm[J]. International Journal of Production Economics, 2011, 132(2): 279-291.

[4] Ning T, Huang M, Liang X, et al. A novel dynamic scheduling strategy for solving flexible job-shop problems[J]. Journal of Ambient Intelligence and Humanized Computing, 2016, 7(5): 721-729.

[5] Zadeh M S, Katebi Y, Doniavi A. A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times[J]. International Journal of Production Research, 2019, 57(10): 3020-3035.

[6] Zeiträg Y, Figueira J R, Horta N, et al. Surrogate-assisted automatic evolving of dispatching rules for multi-objective dynamic job shop scheduling using genetic programming[J]. Expert Systems with Applications, 2022, 209: 118194.

[7] Chen C, Ji Z C, Wang Y. NSGA-II applied to dynamic flexible job shop scheduling problems with machine breakdown[J]. Modern Physics Letters B, 2018, 32: 1840111.

[8] Liu Z G, Liang X, Hou L Y, et al. Multi-strategy dynamic evolution-based improved MOEA/D algorithm for solving multi-objective fuzzy flexible job shop scheduling problem[J]. IEEE Access, 2023, 11: 54596-54606.

[9] 刘微, 兰图, 孙梓华, 等. 混合狼群算法在动态车间调度中的应用[J]. 吉林师范大学学报(自然科学版), 2021, 42(4): 66-73.

[10] 刘乐, 周泓. 一种常见干扰条件下的开放车式车间重调度研究[J]. 管理科学学报, 2014, 17(6): 28-48.

[11] 王艳, 丁宇. 动态柔性作业车间优化调度与决策方法[J]. 系统仿真学报, 2020, 32(11): 2073-2083.

[12] 孙丽珍, 毕利. 多约束条件下的动态柔性作业车间调度研究[J]. 控制工程, 2020, 27(11): 1921-1929.

[13] Askarzadeh A. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm[J]. Computers & Structures, 2016, 169: 1-12.

[14] Kumar P K R, Kousalya K. Amelioration of task scheduling in cloud computing using crow search algorithm[J]. Neural Computing and Applications, 2020, 32(10): 5901-5907.

[15] Zhang Z F, Zhao C, Chen D F,et al. Research on microgrid scheduling based on improved crow search algorithm[J], International Transactions on Electrical Energy Systems, 2022, 2022: 4662760.

[16] 闫红超, 汤伟, 姚斌. 基于新混合乌鸦搜索算法的置换流水车间调度[J/OL]. 计算机集成制造系统, 2022, http://kns.cnki.net/kcms/ditail/11.5946.TP20220802.1204.002.html.

[17] Fan H L, Xiong H G, Goh M. Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints[J]. Computers & Operations Research, 2021, 134: 105401.

[18] Meraihi Y, Gabis A B, Ramdane-Cherif A, et al. A comprehensive survey of crow search algorithm and its applications [J]. Artificial Intelligence Review, 2021, 54(4): 2669-2716.

[19] Gholami J, Mardukhi F, Zawbaa H M. An improved crow search algorithm for solving numerical optimization functions[J]. Soft Computing, 2021, 25(14): 9441-9454.

【责任编辑: 卓祯雨】