控制与决策
Control and Decision

基于改进近端策略优化算法的柔性作业车间调度

王艳红, 付威通, 张俊, 谭园园, 田中大

引用本文:
王艳红, 付威通, 张俊, 等. 基于改进近端策略优化算法的柔性作业车间调度[J]. 控制与决策, 2025, 40(6): 1883-1891.

在线阅读 View online: https://doi.org/10.13195/j.kzyjc.2024.0904

基于改进近端策略优化算法的柔性作业车间调度

王艳红[†]，付威通，张 俊，谭园园，田中大

(沈阳工业大学 人工智能学院，沈阳 110870)

摘 要: 柔性作业车间调度是经典且复杂的组合优化问题, 对于离散制造系统的生产优化具有重要的理论和实际意义. 基于多指针图网络框架和近端策略优化算法设计一种求解柔性作业车间调度问题的深度强化学习算法. 首先, 将“工序-机器”分配调度过程表征成由选择工序和分配机器两类动作构成的马尔可夫决策过程; 其次, 通过解耦策略解除动作之间的耦合关系, 并设计新的损失函数和贪婪采样策略以提高算法的验证推理能力; 在此基础上扩充状态空间, 使评估网络能够更全面地感知与评估, 从而进一步提升算法的学习和决策能力. 在随机生成算例及基准算例上进行仿真和对比分析, 验证算法的良好性能及泛化能力.

关键词: 柔性作业车间调度；近端策略优化算法；双动作耦合网络；损失函数优化；贪婪采样；深度强化学习

中图分类号: TP18 **文献标志码:** A

DOI: 10.13195/j.kzyjc.2024.0904

引用格式: 王艳红, 付威通, 张俊, 等. 基于改进近端策略优化算法的柔性作业车间调度 [J]. 控制与决策, 2025, 40(6): 1883-1891.

Flexible job-shop scheduling based on improved proximal policy optimization algorithm

WANG Yan-hong[†]，*FU Wei-tong*，*ZHANG Jun*，*TAN Yuan-yuan*，*TIAN Zhong-da*

(College of Artificial Intelligence，Shenyang University of Technology，Shenyang 110870，China)

Abstract: Flexible job-shop scheduling is a classical and complex combinational optimization problem, which has important theoretical and practical significance for the production optimization of discrete manufacturing systems. A deep reinforcement learning algorithm for the flexible job-shop scheduling problem is designed based on a muti-pointer graph networks framework and a proximal policy optimization algorithm. Firstly, the operation-machine assignment scheduling is represented as a Markov decision process which is composed of two kinds of actions, namely selection operation and allocation machine. Then, the coupling relationship between actions is removed using a decoupling strategy, and a new loss function and a greedy sampling strategy are designed to improve the verification inference performance. Moreover, the state space is expanded to enable the critic network to perceive and evaluate the state more comprehensively, thereby further improving the learning and decision-making capabilities of the algorithm. Simulations and comparations on randomly generated examples and benchmarks show the superior performance and generalization ability of the proposed algrithm.

Keywords: flexible job-shop scheduling；proximal policy optimization algorithm；double-action coupling network；loss function optimization；greedy sampling；deep reinforcement learning

0 引 言

柔性作业车间调度问题 (FJSP) 是组合优化领域的一个重要分支[1]. 作为作业车间调度问题 (JSP) 的扩展, FJSP 增加了路径的柔性特征[2], 呈现出更显著的 NP 难特性, 使得经典的数学规划或约束规划等精确算法在求解大规模 FJSP 问题时, 会面临计算时间过长的问题[3]. 因此, 研究者只能寻求计算较简便的近似算法.

求解 FJSP 的近似算法主要有规则调度算法和智能优化算法等. 规则调度算法具有求解时间短的

收稿日期: 2024-07-27；**录用日期:** 2024-12-17.

基金项目: 国家自然科学基金青年科学基金项目 (62003221)；辽宁省教育厅科研基金重点攻关计划项目 (LJKZZ20220021)；辽宁省科技计划联合基金项目 (2023-MSLH-255)；辽宁省教育厅科研基金面上项目 (LJKMZ20220509)；东北大学流程工业综合自动化国家重点实验室开放课题 (2023-kfkt-02).

责任编委: 王凌.

[†]通信作者. E-mail: wangyh@sut.edu.cn.

本文附带电子附录文件,可登录本刊官网该文“资源附件”区自行下载阅览.

优点, 但其调度结果的鲁棒性差, 如何设计和选择合适的调度规则是解决问题的瓶颈. 智能优化算法主要有遗传算法[4]、人工蜂群算法[5] 等, 是迄今发展最为完备的方法. 然而, 由于智能优化算法是基于迭代搜索的, 缺少对状态信息的利用和搜索策略的评价[6], 随着问题规模的增大, 算法的性能往往会显著下降.

近年来, 随着人工智能的发展, 深度强化学习 (DRL) 逐渐被应用于车间调度等组合优化领域. DRL 可以通过对交互搜索中获得的信息及搜索策略的学习, 获得知识并将其用于后续迭代搜索, 从而加快收敛速度、提高求解质量. 现有基于 DRL 求解车间调度问题的算法研究可以分为 3 类: 1) 使用 DRL 优化元启发式算法参数[7-8]; 2) 使用 DRL 选择调度规则[9-11]; 3) 使用 DRL 拟合调度规则[12-14].

使用 DRL 优化元启发式算法参数的方法, 可以获得更好的迭代结果. Cai 等[8] 利用Q学习算法, 动态选择混合蛙跳算法搜索策略, 用于求解分布式流水车间调度问题, 提高了解的质量. 但是算法本质上仍属于元启发式算法, 解的随机性高、求解速度慢等问题仍然无法从根本上得到解决.

使用 DRL 选择调度规则的方法降低了规则选择的难度. 王艳红等[11] 针对作业随机到达的动态作业车间调度问题, 使用Q学习来选择调度规则. 算法兼顾了性能与可解释性, 但受制于规则本身的特性, 难以保证算法的全局最优和鲁棒性.

使用 DRL 拟合调度规则的方法摆脱了既有规则的限制. Zhang 等[12] 针对 JSP, 基于图神经网络 (GNN) 设计了一个端到端自动学习优先级规则的方案, 该算法可以从零开始学习优先级调度规则, 并在更大规模的问题上泛化. 在其基础上, Lei 等[14] 针对 FJSP, 设计了一个多指针图网络 (MPGN)架构, 构造了选择工序和分配机器两个动作网络, 并使用多近端策略优化算法 (Multi-PPO) 训练两个动作网络, 获得了良好的性能. 然而, 该算法将两个动作网络独立进行优化, 没有充分考虑动作之间的相互作用和耦合关系, 使得其性能仍有很大提升空间. 为此, 本文在 MPGN 基础上, 提出一种求解 FJSP 的改进双动作 PPO 调度算法 (IDAPPO), 主要工作包括以下 3 方面: 1) 针对工序与机器之间的复杂耦合关系, 设计动作解耦方法; 2) 改进损失函数, 并设计新的双动作贪婪采样策略来提升算法的验证和推理能力; 3) 结合 PPO 与 FJSP 的特性, 为选择工序网络和分配机器网络重新设计状态空间, 提升算法的学习和决策能力.

1 问题的描述

本文针对标准的 FJSP, 考虑共有n个工件$J = \{J_1, J_2, \ldots, J_n\}$需要在$m$台机器$M = \{M_1, M_2, \ldots, M_m\}$上进行加工. 每个工件$J_i$包含一道或多道($k^i$) 工序$O^i = \{O^i_1, O^i_2, \ldots, O^i_{k^i}\}$, 可以在多台不同的机器上进行加工, 但同一时刻只能由一台机器完成. 性能指标取为最小化最大完工时间, 即$J = \min(C_{\max})$.

为求解该组合优化问题, 本文定义了选择工序 $\mathrm{Agent_{job}}$ 和分配机器 $\mathrm{Agent_{mch}}$ 两个智能体, 并将该问题转化为一个马尔科夫决策过程. 于是, n个工件在m台机器上的调度任务转化成为N个决策步上双智能体与环境的交互过程. 将该交互过程建模为一个 7 元组$(s^j_t, s^m_t, a^j_t, a^m_t, r_t, s^j_{t+1}, s^m_{t+1})$. 其中: s^j_t、s^m_t分别表示在第t决策步 $\mathrm{Agent_{job}}$ 的状态、$\mathrm{Agent_{mch}}$ 的状态; a^j_t、a^m_t分别表示 $\mathrm{Agent_{job}}$ 的动作、$\mathrm{Agent_{mch}}$ 的动作, r_t表示这两个动作获得的奖励; s^j_{t+1}、s^m_{t+1}分别表示在状态s^j_t、s^m_t下, 采取a^j_t和a^m_t这两个动作后到达的下一个状态, $1 \leqslant t \leqslant N$, N表示工序的总数量.

2 改进双动作 PPO 调度算法

PPO 算法相较于其他强化学习算法具有对学习率的敏感度低、需调整的参数较少的优点[13]. 因此本文基于 PPO 算法设计一种改进双动作 PPO 调度算法, 首先设计动作网络和评估网络, 再设计损失函数引导两个动作网络收敛.

2.1 神经网络结构

神经网络模型由动作网络与评估网络构成. 其中, 动作网络又包括选择工序、分配机器两个网络; 评估网络嵌入于选择工序网络中, 利用其编码器输出的图池化特征gpl^j估计状态值. 网络结构及参数如表 1 所示.

表1 神经网络结构

网络名称	组件名称	组成	输入、输出节点个数
选择工序网络	编码器	3层GNN, 每层GNN网络有2个全连接网络	(12,256), (256,128) (128,256), (256,128) (128,256), (256,128)
	解码器	3层全连接网络	(128×3,128) (128,128), (128,1)
	评估网络	2层全连接网络	(128,64), (64,1)
分配机器网络	编码器	1个可训练张量矩阵	(2,128)
	解码器	3层全连接网络	(128×3,128) (128,128), (128,1)

1) 状态编码器.

本文利用 GNN[12] 构成选择工序网络的编码器. GNN 汇集工序邻居节点的特征, 查询特征间的隐含关系, 并将它们抽象为更高维度的特征, 输出每个工

序节点的抽象特征ab^j以及图池化特征gpl^j.

分配机器网络的编码器为 1 个 1 层全连接网络. 它将机器特征以矩阵运算形式转换成高维抽象特征, 输出每个机器的高维抽象特征ab^m以及机器池化特征pl^m.

2) 动作解码器.

两个网络的动作解码器皆为 3 层全连接网络, 输出每个动作的得分, 并通过掩码ms^j、ms^m将不可选择的工序和机器掩盖.

2.2　双智能体与环境交互过程

本文设计的 IDAPPO 是一种离线 DRL 算法. 它先通过与环境Env的交互获得经验, 再利用经验进行网络更新. 在每个决策步t, Agent与Env的交互流程如图 1 所示, 每个交互过程可以分解为如下步骤.

step 1: $\mathrm{Agent_{job}}$观察Env得到当前的s_t^j和ms_t^j; $\mathrm{Agent_{mch}}$观察Env得到s_t^m和ms_t^m.

step 2: $\mathrm{Agent_{job}}$编码器对s_t^j编码获得ab_t^j和gpl_t^j, 并将gpl_t^j传递到$\mathrm{Agent_{mch}}$; 之后, $\mathrm{Agent_{job}}$解码器解码gpl_t^j、pl_{t-1}^m、ab_t^j, 计算得到每个侯选工序的概率pi_t^j; 通过ms_t^j掩盖加工完毕工件的工序, 再经过采样获得a_t^j.

step 3: $\mathrm{Agent_{mch}}$编码器对s_t^m进行编码获得ab_t^m和pl_t^m, 并将pl_t^m传给$\mathrm{Agent_{job}}$; $\mathrm{Agent_{mch}}$解码器解码gpl_t^j、pl_t^m和ab_t^m, 计算每台机器的概率pi_t^m; 通过ms_t^m掩盖掉不可加工该工序的机器, 再经过采样获得a_t^m.

step 4: 两个Agent的动作a_t^j和a_t^m作用于Env, Env从s_t^j、s_t^m转移到s_{t+1}^j、s_{t+1}^m.

step 5: 将s_t^j, s_t^m, a_t^j, a_t^m, ms_t^j, ms_t^m, r_t等信息作为经验存放于经验池中, 完成一次交互.

step 6: 若所有工序都被安排到可加工机器上, 即整个调度过程完毕, 取出经验计算损失函数, 更新神经网络, 否则返回 step 1.

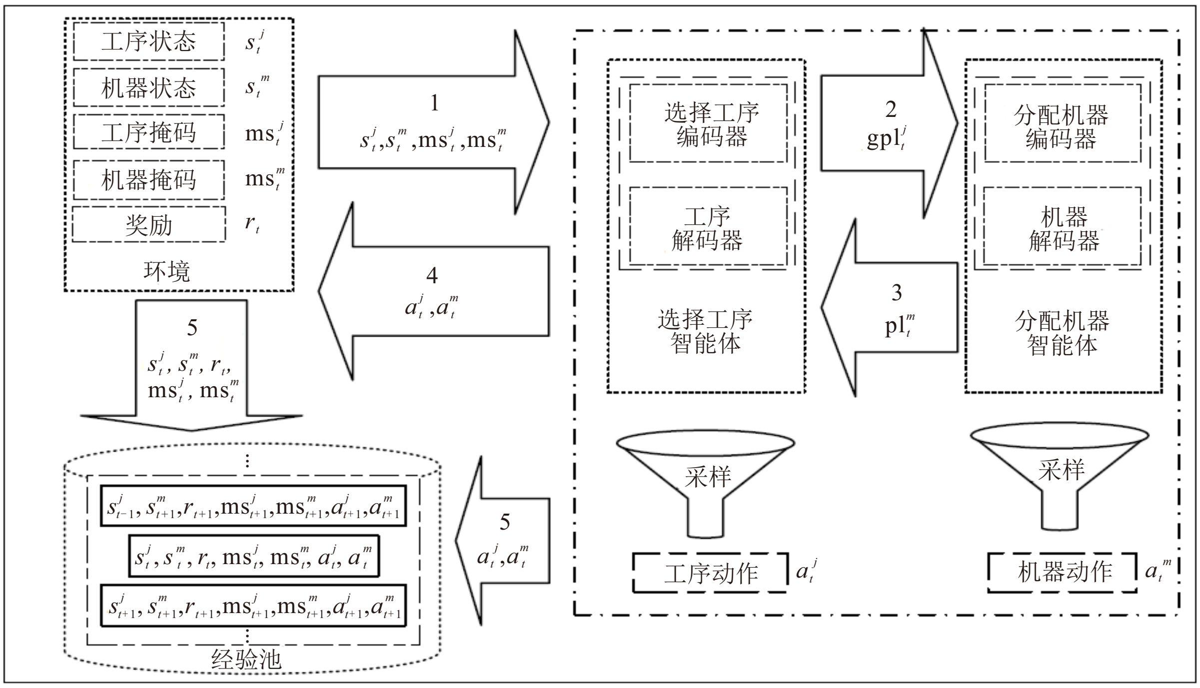


图1　智能体与环境的交互

2.2.1　状态空间

状态空间是智能体感知环境的重要来源. 对应于$\mathrm{Agent_{job}}$、$\mathrm{Agent_{mch}}$两个智能体, 状态空间也由两部分组成, 包括$\mathrm{Agent_{job}}$的工序状态s^j和$\mathrm{Agent_{mch}}$的机器状态s^m.

1) 工序状态.

为充分表征作业车间中工序的状态, 本文为$\mathrm{Agent_{job}}$精心设计了 14 个状态特征.

① 工序的预期最低完成时间LB[15]. 对于已经调度的工序$O_{j,o}$, $\mathrm{LB}(O_{j,o})$就是其完工时间; 对于尚未调度的工序$O_{j,o}$, $\mathrm{LB}(O_{j,o})$为同一工件的直接前续工序的完工时间 (没有直接前续时取为零) 加上所有可加工机器上的最短加工时间.

② 已调度标志I. 已调度的工序$I(O_{j,o})$为 1, 未调度的工序$I(O_{j,o})$为 0.

③ 工序在可用机器上的最短加工时间$\min_p_{j,o}$.

④ 工序在可用机器上的平均加工时间$\mathrm{mean}_p_{j,o}$.

⑤ 工序在可用机器上加工时间的跨度$\mathrm{sp}_p_{j,o}$.

⑥ 工序可被加工的机器数量num_$O_{j,o}$.

⑦ 工件候选工序的可以开始加工时间 (ope_time$_o$), 即工件直接前继工序完工时间.

⑧ 工件尚未调度工序 (job_re$_j$) 的数量num_job_re$_j$.

⑨ 工件尚未调度工序的最低加工总耗时min_re_load$_j$.

⑩ 工件尚未调度工序的平均加工总耗时mean_re_load$_j$.

⑪ 工件尚未调度工序的加工时间总耗时的跨度sp_re_load$_j$.

⑫ 工序等待机器时间的最小值min_ope_wait_mch$_{o,m}$.

⑬ 工件等待机器时间的平均值mean_ope_wait_mch$_{o,m}$.

⑭工件等待机器时间的最大值max_ope_wait_mch$_{o,m}$.

以上, 特征①～⑥是与工序直接相关的特征. 其中: 特征①和②沿用了文献 [14] 中表现良好的特征, 为解决不同调度算例中工序加工时间跨度不同的问题, 引入特征③～⑤, 为优先考虑加工机器较少的工序引入特征⑥.

本文还引入与工件以及机器有关的特征 (特征⑦～⑭) 以丰富状态信息, 使评估值更加准确. 其中: 特征⑦～⑪是与工件有关的特征, 特征⑦为工件时间特征, 为了优先处理尚未调度的、工序较多的工件引入特征⑧, 为了优先处理剩余负载最大的工件设置特征⑨～⑪; 特征⑫～⑭则代表了工序等待机器的时间.

2) 机器状态.

考虑到在工序状态中已包含工件时间和工件等待机器时间这两个特征, 通过它们可求得机器完工时间, 并将在gpl^{j}中传递到Agent$_{\rm mch}$. 因而, 本文采用机器等待时间替代文献 [14] 使用的机器完工时间特征. 于是Agent$_{\rm mch}$具有以下两个状态特征:

1) 机器上等待选择的工序的等待时间mch_wait_ope$_{m,o}$.

2) 工序的加工时间$p_{j,o,m}$. 若某工序不可在某台机器上加工, 则其加工时间取为其他可用机器上加工时间的平均值.

2.2.2 动作空间

本文采用直接映射的方式选择动作. 选择工序网络的动作空间定义为所有工件的已调度工序的后续工序, 大小等于工件数量. 分配机器网络的动作空间包括所有机器, 大小等于机器数量.

2.2.3 状态转移

在每个决策步t, 双智能体作出动作a_t^j、a_t^m后, 环境会根据a_t^j、a_t^m将状态s_t^j、s_t^m更新为s_{t+1}^j、s_{t+1}^m, 并将其传递给智能体. 现有文献在设计状态转移时, 常使用“允许左移”操作[13,16], 即在调度工序时, 查看机器是否有时间间隔可插入该工序, 以提高算法的性能. 然而, 本文研究发现“允许左移”操作会消耗大量计算时间, 为此本文的状态转移取消了“允许左移”, 将工序安排在加工机器上已调度的工序之后.

2.2.4 奖励函数

由文献 [10] 可知, 稀疏奖励在大规模的调度算例中表现较差, 而密集奖励难以设计, 为此本文构造了密集-稀疏奖励机制, 即在状态之间使用密集奖励, 在同一个状态的两个动作之间使用稀疏奖励. 针对$C_{\max}$最小化这一优化目标, 本文设计两种密集-稀疏奖励, 具体如下:

1) 工序的预期最低完成时间LB形式. 每个决策步的奖励为$r_t={\rm LB}(s_{t-1}^j,s_{t-1}^m)-{\rm LB}(s_t^j,s_t^m)$. 其中: ${\rm LB}(s_t^j,s_t^m)$的计算公式为

$$\begin{aligned}&{\rm LB}(s_t^j,s_t^m)=\max({\rm LB}(O_{j,o})),\\&\forall j\in J,\ o\in O^j.\end{aligned}\tag{1}$$

2) 机器完工时间F形式. F为动作完成后所有机器完成时间的最大值, 每个决策步的奖励为$r_t=F(s_{t-1}^j,s_{t-1}^m)-F(s_t^j,s_t^m)$.

在LB、F形式下, 动作a_t^j、a_t^m的预期后续累积奖励分别为${\rm LB}(s_{t-1}^j,s_{t-1}^m)-C_{\max}$、$F(s_{t-1}^j,s_{t-1}^m)-C_{\max}$.

两种形式奖励函数的累计奖励最大化都与$C_{\max}$最小化一致, 但在实际训练测试中发现LB形式优于F形式. LB形式的奖励相较于F形式更加密集. 更加密集的奖励有助于智能体获得更好的性能. 为此, 本文后续采用LB形式密集-稀疏奖励.

2.2.5 采 样

采样可以将解码器输出的动作概率转化为动作. 本文在训练阶段, 使用随机采样作为采样策略, 在验证阶段使用改进贪婪采样作为采样策略. 贪婪采样原指两个智能体都依据各自智能体输出的动作概率选择概率最大的动作. 本文改进为选择概率最大的动作组合, 逻辑伪代码如表 2 所示.

表 2 中: step 1 表示选择工序网络输出每个候选工序的概率; step 2～step 5 表示遍历候选工序, 为每个候选工序计算被机器加工的概率; step 6 表示选择最大概率的动作组合作为采样结果.

表2　贪婪采样伪代码

贪婪采样伪代码
step 1:　选择工序网络对每个候选动作(cand)计算概率pi^j、初始化存储动作组合概率的数组
step 2:　for ja = cand do
step 3:　　机器网络计算工序ja对应使用每台机器加工的概率$\{\mathrm{pi}_k^m\}, k \in M$
step 4:　　计算工序ja与机器组成的动作组合的概率$\{\mathrm{pi}_{\mathrm{ja}}^j \times \mathrm{pi}_k^m\}$, $\mathrm{ja} \in \mathrm{cand}, k \in M$并存储在数组中
step 5:　end for
step 6:　选择数组中概率最大的动作组合 $\max(\{\mathrm{pi}_{\mathrm{ja}}^j \times \mathrm{pi}_k^m\}, \mathrm{ja} \in \mathrm{candidate}, k \in M)$

在决策步t, 新贪婪采样选择概率最大的动作组合为$\{a_t^j$、$a_t^m\}$, 即新的贪婪采样会更加强调整体收益. 在多智能体系统中, 考虑整体收益比考虑单个智能体的收益更高.

2.3　动作解耦策略

智能体$\mathrm{Agent}_{\mathrm{job}}$、$\mathrm{Agent}_{\mathrm{mch}}$的动作$a^j$、$a^m$对彼此调度结果都有影响, 存在耦合关系, 使得在 DRL 中单个动作a^j或者a^m的累积奖励难以计算. 但动作的后续累计奖励是损失函数中的一部分, 影响着算法最终的收敛效果. 因此, 本文设计解耦策略来解除动作耦合关系, 包括两方面:

1) 状态间的动作解耦. 动作a_t^j、a_t^m与a_{t+1}^j、a_{t+1}^m之间的耦合关系是通过动态规划解除的. 状态s_{t1}^j、s_{t1}^m的动作a_{t1}^j、a_{t1}^m与状态s_{t2}^j、s_{t2}^m的动作a_{t2}^j、a_{t2}^m之间调度结果的影响可以量化为$R(a_{t1}^j, a_{t1}^m) - R(a_{t2}^j, a_{t2}^m) = \sum_{t1}^{t2} r_t$.

2) 状态内两个动作的解耦. 动作a_t^j与a_t^m的耦合关系是通过动作组合解除的. 不考虑每个动作的影响, 转而考虑动作组合的影响. 每个动作组合对后续调度结果的影响可以量化为$R(a_t^j, a_t^m) - R(a_{t+1}^j, a_{t+1}^m) = r_t$.

2.4　损失函数

损失函数指引算法的优化方向. PPO 用于多智能体训练时, 优化目标为每个智能体分别获得最大累计奖励. 但每个智能体的最佳动作难以保证系统总体的$C_{\max}$最小. 本文将两个智能体在同一损失函数下进行优化, 改进优化目标为双智能体总体最大收益.

PPO 损失函数包含clip损失 (loss_clip, LC)、评估网络的均方差损失 (loss_critic, LCR), 以及动作网络的交叉熵损失 (loss_entropy, LE)[16] 3 部分, 即

$$\mathrm{loss} = \mathrm{LC} + 0.5\mathrm{LCR} + 0.01\mathrm{LE}. \tag{2}$$

本文针对 FJSP 问题对其进行改进, clip损失改进为

$$\mathrm{LC} = E[\varDelta_{3-1} + \varDelta_{3-2}]. \tag{3}$$

其中: $\varDelta_{3-1}$和$\varDelta_{3-2}$分别定义为

$$\begin{cases} \varDelta_{3-1} = \min(\mathrm{clip}(\varDelta_{4-1}, 1-\varepsilon, 1+\varepsilon), \varDelta_{4-1}) \times \\ \qquad \mathrm{Adv}(a_t^j, a_t^m), \\ \varDelta_{3-2} = \min(\mathrm{clip}(\varDelta_{4-2}, 1-\varepsilon, 1+\varepsilon), \varDelta_{4-2}) \times \\ \qquad \mathrm{Adv}(a_t^j, a_t^m), \\ \varDelta_{4-1} = \dfrac{\pi_\theta^j(a_t^j|s_t^j)}{\pi_{\theta_{\mathrm{old}}}^j(a_t^j|s_t^j)}, \\ \varDelta_{4-2} = \dfrac{\pi_\theta^m(a_t^m|s_t^m)}{\pi_{\theta_{\mathrm{old}}}^m(a_t^m|s_t^m)}, \end{cases} \tag{4}$$

这里$\mathrm{Adv}(a_t^j, a_t^m)$为优势函数.

评估网络均方差损失改进为

$$\mathrm{LCR} = E[(V(s_t^j, s_t^m) - R(s_t^j, s_t^m))^2]. \tag{5}$$

其中: $V(s_t^j, s_t^m)$表示评估网络对状态s_t^j、s_t^m估计的状态值, $R(s_t^j, s_t^m)$表示状态s_t^j、s_t^m到终点状态的累积奖励.

动作的交叉熵损失改进为

$$\mathrm{LE} = E[\mathrm{Ent}(\pi_\theta^j(\cdot|s_t^j))] + E[\mathrm{Ent}(\pi_\theta^m(\cdot|s_t^m))], \tag{6}$$

其中$E[\cdot]$表示取均值.

改进后, 使用统一的损失函数训练网络的优点主要为: 1) 有助于使两个智能体之间传递的张量具有更完整的梯度信息; 2) 张量的梯度信息更加准确. 此外, 本文设计的损失函数也可以应用到其他多动作、但无法为每个动作分配奖励函数的问题中.

3　仿真测试与对比分析

本文基于 Python3.10, 在 Nvidia 3090ti 显卡、i9-12900 k CPU 的 Windows 主机上完成损失函数的性能、算法的性能以及泛化性等测试. 使用的主要超参数如表 3 所示.

表3　超参数

超参数名	超参数值
学习率	2e-4
k_epochs	2
clip系数ε	0.2
Adam.eps	1e-5
epochs	4

3.1　损失函数性能测试

为验证本文设计的新损失函数的收敛效果, 仅使用改进后的损失函数加上新的贪婪算法, 在与文献 [14] 相同的条件下进行训练和验证, 表 4 给出了测试的$C_{\max}$结果.

表4 损失函数性能测试结果

算例规模	文献[14]	本文
6 × 6	271.34	272.32
10 × 10	320.45	316.59
15 × 15	347.99	340.86

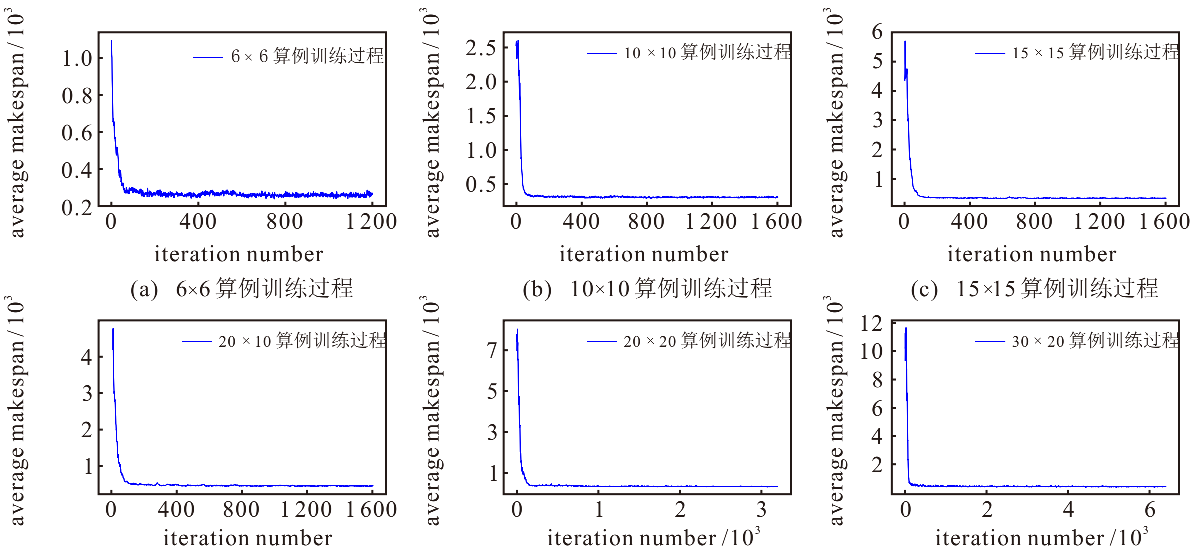
由表 4 可以看出, 与文献 [14] 比较, 本文的$C_{\max}$仅在6×6规模的调度算例上略逊色, 在更大规模问题上均取得了更好的结果. 分析其原因为: 小规模算例下, 损失函数对性能提升的空间有限, 加之取消“允许左移”操作, 导致性能下降; 随着问题规模的增大, 新损失函数对收敛性提升的优势愈加明显.

3.2 算法性能测试

为了进一步验证算法的性能, 在相同随机种子生成的算例[14-15] 上进行训练、测试, 并与基准算法进行比较.

所对比的算法中, Or-tools 是 Google 研发的软件工具包; 基准调度规则选取了 8 组复合规则[14], 分别由 4 条选择工序规则, 包括 FIFO(first in, first out)、LWKR(least work remaining first)、MOPNR (most operation number remaining first)、MWKR(most work remaining first), 以及 2 条分配机器规则, 包括 EET(earliest end time first) 和 SPT(shortest processing time first) 组成.

图 2 给出了不同规模算例的训练收敛过程, 可以看出, 本文设计的新损失函数可以收敛到理想的效果.



(a) 6×6 算例训练过程 (b) 10×10 算例训练过程 (c) 15×15 算例训练过程

(d) 20×10 算例训练过程 (e) 20×20 算例训练过程 (f) 30×20 算例训练过程

图2 训练曲线

表5 101 号随机种子训练测试结果

算例规模	调度性能	调度算法							
		OR_TOOLS	FIFO+SPT	FIFO+EET	MOPNR+SPT	MOPNT+EET	MWKR+SPT	MWKR+EET	**IDAPPO**
6 × 6	$C_{\max}$	213.59	305.36	263.38	323.36	262.20	326.24	262.2	251.58
	t / s	0.101	0.0132	0.0139	0.0140	0.0142	0.0132	0.0142	0.031
10 × 10	$C_{\max}$	246.12	379.43	321.46	422.55	331.23	417.85	316.86	299.280
	t / s	1.36	0.051	0.050	0.050	0.048	0.047	0.050	0.137
15 × 15	$C_{\max}$	250.33	407.03	342.05	468.78	357.50	462.80	344.90	320.01
	t / s	47.96	0.158	0.150	0.149	0.167	0.146	0.149	0.452
20 × 10	$C_{\max}$	—	553.00	487.01	616.78	491.05	620.02	476.35	434.57
	t / s	—	0.143	0.141	0.142	0.146	0.138	0.142	0.473
20 × 20	$C_{\max}$	—	416.76	350.14	493.59	372.19	487.08	359.11	328.74
	t / s	—	0.362	0.353	0.348	0.344	0.359	0.351	1.002
30 × 20	$C_{\max}$	—	519.49	437.41	621.00	461.21	616.56	444.74	398.47
	t / s	—	1.168	0.683	0.675	0.664	0.660	0.676	2.357
50 × 20	$C_{\max}$	—	712.63	651.60	844.18	659.95	848.99	643.28	576.50
	t / s	—	1.658	1.636	1.676	1.640	1.637	1.631	7.744
100 × 20	$C_{\max}$	—	1191.12	1245.81	1361.41	1241.47	1377.57	1231.93	1059.10
	t / s	—	6.775	6.712	6.575	6.617	6.794	6.728	31.26

为了验证算法的性能, 在 101 号 Numpy 种子上进行实验验证, 结果如表 5 所示. 可以看出: IDAPPO 求解质量优于所有复合调度规则; Or-tools 求解器在 1 800 s 内无法求解大规模算例, 而 IDAPPO 与复合调度规则的求解时间在同一数量级, 远快于 Or-tools. 因此, 本文设计的 IDAPPO 算法兼备了求解速度和求解质量的优势.

3.3 算法泛化性测试

算法泛化性的测试数据集包括随机生成的调度算例和 Hurink[17] 基准算例.

3.3.1 随机生成算例测试

本文在 101 号 Numpy 种子和 60 号 Torch 种子上训练验证, 在 200 号 Numpy 种子上测试对比, 测试算法的泛化性, 结果如表 6 所示, 其中 50×20、100×20 算例由 30×20 算例所得智能体验证.

由表 6 可以看出: IDAPPO 在除 100×20 外的算例上, 性能均优于文献 [14] 的结果, 并且在求解时间上与其相近. 研究结果表明: 本文的 IDAPPO 算法兼有较强的泛化能力与较快的求解速度. IDAPPO 在 100×20 算例上的表现略逊色, 分析原因为, 100×20 算例工序数 (即状态数量) 远多于原 30×20 规模算例, 使智能体区分状态的能力略有下降.

表6 200 号随机种子测试结果

算力规模	调度性能	调度算法	
		文献[14]	IDAPPO
6×6	$C_{\max}$	272.32	260.948
	t/s	0.041	0.027
10×10	$C_{\max}$	320.45	296.758
	t/s	0.14	0.183
15×15	$C_{\max}$	347.99	316.14
	t/s	0.39	0.456
20×10	$C_{\max}$	454.85	431.667
	t/s	0.34	0.466
20×20	$C_{\max}$	361.75	330.52
	t/s	1.08	1.020
30×20	$C_{\max}$	433.42	398.80
	t/s	1.97	2.072
50×20	$C_{\max}$	587.48	576.78
	t/s	4.12	6.326
100×20	$C_{\max}$	1 054.70	1 058.48
	t/s	18.34	25.465

3.3.2 基准测试

本文在 101 号 Numpy 种子和 60 号 Torch 种子上训练, 在 Hurink 基准算例中验证, 结果如表 7 和表 8 所示. 两种用于对比的元启发式算法中, RGA 为遗传算法, 2SGA 是两阶段遗传算法, 皆源于文献 [4], 而 DRL 包括文献 [14] 和本文设计的 IDAPPO.

表7 Hurink 基准算例测试结果

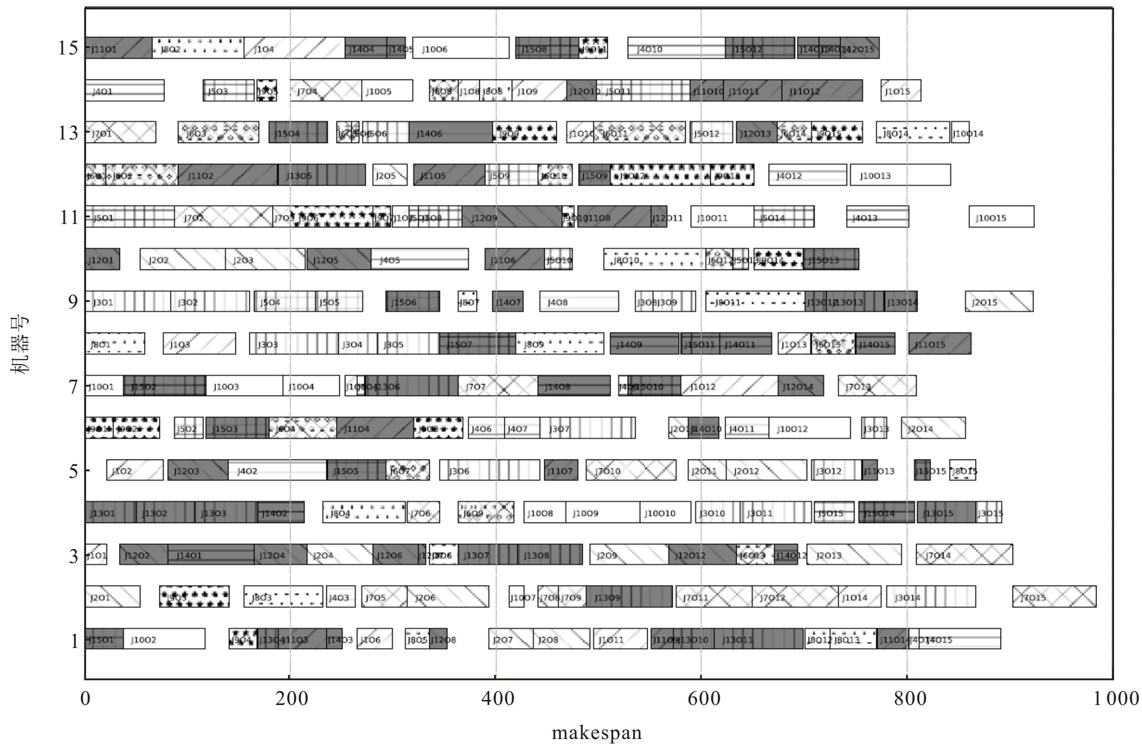
算例规模	算例	FIFO+EET	FIFO+SPT	LWKR+EET	MOPNR+EET	MOPNR+SPT	MWKR+EET	RGA	2SGA	文献[14]	**IDAPPO**
10×10	La16	747	2 491	1 617	760	2 492	717	717	717	717	717
	La17	663	2 269	1 395	649	2 247	646	646	646	647	646
	La18	673	2 732	1 435	706	2 732	691	663	663	663	663
	La19	656	2 212	1 412	747	2 212	672	617	617	626	634
	La20	757	1 922	1 471	756	1942	756	756	756	756	756
15×15	La36	992	4 975	2 489	1 036	4 894	1 512	948	948	985	984
	La37	1 043	5 785	2 822	1 065	5 785	1 710	988	988	1 028	1 027
	La38	949	5 181	2 698	961	5 181	1 430	943	943	948	943
	La39	975	5 291	2 418	990	5 291	1 628	931	922	979	970
	La40	979	5 072	2 398	973	4 966	1 596	955	955	968	975

表8 求解时间比较

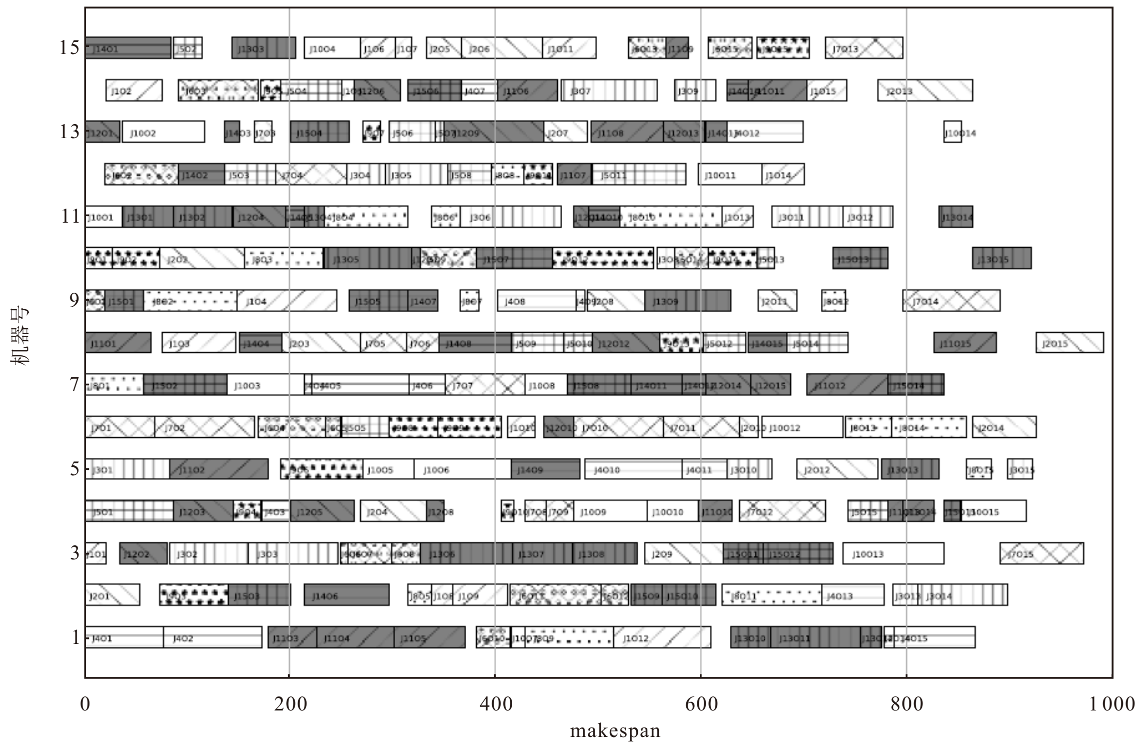
算法类型	复合调度规则	DRL	元启发式算法[4]
耗时	0.05 ~ 0.2 s	0.1 ~ 0.5 s	1 ~ 30 min

由表 7 和表 8 可以看出: DRL 在求解质量上优于复合调度规则, 在求解速度上远快于元启发式算法. 这意味着 DRL 兼备了求解质量和求解速度的优势. 同时, 在 DRL 中, 对比文献 [14], 在所有 10 个基准算例上, IDAPPO 在 5 个算例上表现优秀, 有 3 个持平, 有 2 个略逊色, 表明了算法具有良好的鲁棒性, IDAPPO 可以在训练后直接用于求解其他 FJSP 问题, 不必重新训练. IDAPPO 在很大程度上优于文献 [14], 然而却并未保留在 101 号随机种子和 200 号随机种子验证集上的巨大领先优势, 换言之, IDAPPO 具备泛化能力, 但泛化到与训练算例有较大差异的算例时, 会伴随轻微的性能下降.

为了更加直观地对比 IDAPPO 对复合调度规则的性能优势, 图 3(a) 和图 3(b) 给出了 La36 算例在表现最好的复合调度规则以及 IDAPPO 求解结果上的甘特图. 对比可以看出, 两者在前期都可以拥有较为密集的机器排班, 但在后期, IDAPPO 在 12 以及 13 号机器上排班更加密集, 得到了更小的 $C_{\max}$.



(a) La36 IDAPPO甘特图



(b) La36 Fifo+Eet甘特图

图3 甘特图

4 结 论

本文提出了一种用于求解 FJSP 的 IDAPPO 算法, 设计了解耦策略解除动作耦合关系, 并改进了损失函数、设计了贪婪采样策略来提升算法性能. 在此基础上, 结合 PPO 与 FJSP 特性, 为选择工件网络和分配机器网络重新设计了状态空间. 测试结果表明, IDAPPO 算法在基准算例和随机生成的算例上的性能都得到了提高, 并且具备良好的鲁棒性和泛化能力. 然而, 算法在泛化到状态数量、数据结构差别较大的算例时, 解的质量会有所下降, 未来将考虑通过改进网络模型等途径解决这一问题.

参考文献 (References)

[1] Du Y, Li J, Li C, Duan P. A reinforcement learning

approach for flexible job-shop scheduling problem with crane transportation and setup times[J]. IEEE Transactions on Neural Networks and Learning Systems, 2024, 35(4): 5695-5709.

[2] Wang R Q, Wang G, Sun J, et al. Flexible job-shop scheduling via dual attention network-based reinforcement learning[J]. IEEE Transactions on Neural Networks and Learning Systems, 2024, 35(3): 3091-3102.

[3] Song W, Chen X Y, Li Q Q, et al. Flexible job-shop scheduling via graph neural network and deep reinforcement learning[J]. IEEE Transactions on Industrial Informatics, 2023, 19(2): 1600-1610.

[4] Defersha F M, Rooyani D. An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time[J]. Computers & Industrial Engineering, 2020: 106605-106623.

[5] 王静, 雷德明. 考虑批处理机的绿色模糊混合流水车间调度问题[J]. 控制与决策, 2024, 39(10): 3413-3421.
(Wang J, Lei D M. Research on energy-efficient fuzzy hybrid flow shop scheduling with batch processing machines [J]. Control and Decision, 2024, 39(10): 3413-3421.)

[6] 王凌, 潘子肖. 基于深度强化学习与迭代贪婪的流水车间调度优化[J]. 控制与决策, 2021, 36(11): 2609-2617.
(Wang L, Pan Z X. Scheduling optimization of flow shop based on deep reinforcement learning and iterative greed[J]. Control and Decision, 2021, 36(11): 2609-2617.)

[7] Wang J, Lei D M, Cai J C. An adaptive artificial bee colony with reinforcement learning for distributed three-stage assembly scheduling with maintenance[J]. Applied Soft Computing, 2022, 117: 108371-108383.

[8] Cai J C, Lei D M, Wang J, et al. A novel shuffled frog-leaping algorithm with reinforcement learning for distributed assembly hybrid flow shop scheduling[J]. International Journal of Production Research, 2023, 61(4): 1233-1251.

[9] Han B A, Yang J J. Research on adaptive job shop scheduling problems based on dueling double DQN[J]. IEEE Access, 2020, 8: 186474-186495.

[10] Samsonov V, Ben H K, Meisen T. Reinforcement learning in manufacturing control: Baselines, challenges and ways forward[J]. Engineering Applications of Artificial Intelligence, 2022, 112: 104868-104882.

[11] 王艳红, 尹涛, 谭园园, 等. 基于规则与Q学习的作业车间动态调度算法[J]. 计算机集成制造系统, 2023, 11(2): 1-17.
(Wang Y H, Yin T, Tan Y Y, et al. Dynamic job-shop scheduling algorithm based on rule and Q learning[J]. Computer Integrated Manufacturing Systems, 2023, 11(2): 1-17.)

[12] Zhang C, Song W, Cao Z, et al. Learning to dispatch for job shop scheduling via deep reinforcement learning[J]. arXiv, 2020, 2010: 12367-12378.

[13] Che G, Zhang Y, Tang L, et al. A deep reinforcement learning based multi-objective optimization for the scheduling of oxygen production system in integrated iron and steel plants[J]. Applied Energy, 2023, 345: 121332-121351.

[14] Lei K, Guo P, Zhao W, et al. A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem[J]. Expert Systems with Applications, 2022, 205: 117796-117813.

[15] Taillard E D. Benchmarks for basic scheduling problems[J]. European Journal of Operational Research, 1993, 64(2): 278-285.

[16] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv, 2017, 1707: 06347-06358.

[17] Hurink J, Jurisch B. Tabu search for the job-shop scheduling problem with multi-purpose machines[J]. Operations Research Spektrum, 1994, 15(4): 205-215.

作者简介

王艳红 (1967−), 女, 教授, 博士生导师, 主要研究方向为生产调度与优化、智能工厂及数字孪生系统, E-mail: wangyh@sut.edu.cn;

付威通 (1999−), 男, 硕士生, 主要研究方向为生产调度、深度强化学习, E-mail: 17866536530@163.com;

张俊 (1986−), 男, 副教授, 博士生导师, 主要研究方向为复杂工业生产过程建模、优化与控制以及神经网络、深度学习, E-mail: zhangjunroger@163.com;

谭园园 (1983−), 女, 副教授, 博士生导师, 主要研究方向为生产计划与调度, E-mail: tanyuanyuan83@sina.com;

田中大 (1978−), 男, 教授, 博士生导师, 主要研究方向为复杂工业过程建模与控制、网络控制系统的协同控制、时间序列的建模与预测, E-mail: tianzhongda@126.com.