分类号：TP24.6　　学校代码：10363
密　级：公　开　　学　　号：2220110173

安徽工程大学
Anhui Polytechnic University

硕士学位论文

题目　作业车间调度优化及其方法研究

论 文 作 者　胡孔夫
指 导 教 师　王雷　教授
学科（专业）　机械
研 究 方 向　智能制造系统调度与优化

论文提交日期：2025 年 5 月 18 日

分类号：TP242.6　　单位代码：10363
密　级：公　开　　学　　号：2220110173

作业车间调度优化及其方法研究

Research on Optimization and Methods of Job Shop Scheduling

学 生 姓 名 ： 胡孔夫
校 内 老 师 ： 王雷 教授
校 外 导 师 ： 张茂杉
专 业 学 位 类 别 ： 机械
研究方向（领域）： 智能制造系统调度与优化

论文答辩日期： 2025 年 5 月 12 日

安徽工程大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律后果由本人承担。

学位论文作者签名：胡弘史

日期：2025 年 5 月 18 日

安徽工程大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权安徽工程大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在　　年解密后适用本版权书。

本学位论文属于

不保密☑。

学位论文作者签名：胡乳夫　　指导教师签名：

日期：2025年 5 月 18 日　　日期：2025年 5 月 18 日

作业车间调度优化及其方法研究

摘　要

随着科技的迅猛发展，世界正迈入工业 4.0 时代，工业生产规模迅速扩大，工厂对高效生产方法的需求日益增加。其中，调度作为影响生产效率的关键因素之一，受到广泛关注。

作业车间调度问题(job shop scheduling problem, JSP)作为最基本的生产调度问题之一，在汽车制造、机械加工、纺织服装工业等领域有着广泛的应用，其研究具有重要的学术价值和工程意义。近年来，针对 JSP 的解决方法层出不穷，其中智能优化算法和深度强化学习方法因其独特的优势和应用前景成为研究的热点。本文聚焦经典 JSP 及其扩展问题——柔性作业车间调度问题(flexible job shop scheduling problem, FJSP)，分别提出了一种混合元启发式算法和一种深度强化学习方法进行求解，并通过基准数据集验证所提方法的有效性和鲁棒性。本文的主要工作包括以下几个方面：

首先，针对 JSP，以最小化完工时间(Makespan)为目标，提出了一种混合遗传禁忌搜索算法(hybrid genetic tabu search algorithm，HGTSA)。在问题的编码阶段，采用了基于工序的编码方式，以便于进行后续的搜索操作。在遗传算法阶段，设计了一种改进的 POX 交叉算子，以及一种基于单台机器邻域搜索的变异算子，帮助个体跳出局部最优。在禁忌搜索阶段，通过分析现有的邻域结构，从理

论上建立了关键路径上工序无效移动的判据，并设计了一种基于 N8 邻域的包含三对工序联合移动的多工序精确联动邻域结构(N8-transpose with 2-machine-transpose, N8T+2MT)。此外，还提出了一种基于调度部分重建的搜索方法，进一步增加算法的搜索能力。最后通过实验验证了所提算法的有效性。

随后，针对 FJSP，同样以最小化完工时间为目标，提出了一种端到端的深度强化学习(Deep reinforcement learning, DRL) 框架。该框架设计了一种新的神经元架构，在传统神经元结构的基础上引入了二次项权重，并将其应用于基于长短期记忆网络(Long Short-Term Memory, LSTM)的 Actor 网络。此外，采用双 Critic 网络分别拟合状态值函数和动作值函数。使用近端策略优化算法(Proximal policy optimization algorithm, PPO)训练智能体。考虑 FJSP 中作业层和机器层的决策需要，设计了新的复合调度规则，并建立了合理的状态空间和奖励机制。将车间的实时状态信息编码成多个矩阵，神经网络通过接收状态信息输出执行不同动作的概率。实验表明，该框架在随机生成的小规模数据集上训练后，能够高效解决任意规模的实例，具有较强的鲁棒性和泛化能力，并在求解时间上优于其他启发式算法。

最后，总结了全文的工作，并展望了未来的研究方向。

关键词：作业车间调度问题；柔性作业车间调度问题；遗传算法；禁忌搜索算法；深度强化学习；近端策略优化算法

RESEARCH ON OPTIMIZATION AND METHODS OF JOB SHOP SCHEDULING

ABSTRACT

With the rapid development of technology, the world is entering the era of Industry 4.0, leading to the rapid expansion of industrial production and an increasing demand for efficient production methods. Scheduling, as a critical factor influencing production efficiency, has been widely studied.

The job shop scheduling problem (JSP), one of the fundamental production scheduling problems, is extensively applied in industries such as automotive manufacturing, mechanical processing, and textile and garment production. Research on JSP is considered to hold significant academic value and engineering importance. In recent years, numerous approaches have been proposed to address JSP, among which intelligent optimization algorithms and deep reinforcement learning methods have been recognized as research hotspots due to their unique advantages and promising applications. This study focuses on the classical JSP and its extended version—the flexible job shop scheduling problem (FJSP). A hybrid metaheuristic

algorithm and a deep reinforcement learning (DRL) approach are proposed to solve these problems, and their effectiveness and robustness are validated using benchmark datasets.

First, for JSP, a hybrid genetic tabu search algorithm (HGTSA) is proposed to minimize makespan. In the encoding stage, an operation-based encoding scheme is adopted to facilitate subsequent search operations. During the genetic algorithm phase, an improved POX crossover operator and a mutation operator based on single-machine neighborhood search are designed to help individuals escape local optima. In the tabu search phase, criteria for the invalid movement of operations on the critical path are theoretically established based on an analysis of existing neighborhood structures. Additionally, a multi-operation joint movement neighborhood structure, N8-transpose with 2-machine-transpose (N8T+2MT), is introduced, incorporating joint movements of three pairs of operations within the N8 neighborhood. Furthermore, a search method based on scheduling partial reconstruction is developed to enhance the algorithm's search capability. The effectiveness of the proposed algorithm is verified through experimental results.

Second, for FJSP, an end-to-end deep reinforcement learning (DRL) framework is proposed, also aiming to minimize makespan. The framework incorporates a novel neuron architecture in which quadratic term weights are introduced into the traditional neuron structure and applied to

the Actor network based on a Long Short-Term Memory (LSTM) network. Additionally, a dual-Critic network is adopted to approximate the state-value function and the action-value function, and the agent is trained using the Proximal Policy Optimization (PPO) algorithm. Considering the decision-making requirements at both the job and machine levels in FJSP, a new composite scheduling rule is designed, along with a reasonable state space representation and reward mechanism. The real-time state information of the shop floor is encoded into multiple matrices, and the neural network is trained to output the probabilities of executing different actions based on the received state information. Experimental results demonstrate that after being trained on small-scale randomly generated datasets, the proposed framework can efficiently solve instances of arbitrary scales. Strong robustness and generalization capability are exhibited, while computational efficiency surpasses that of other heuristic algorithms.

Finally, the conducted research is summarized, and potential directions for future work are discussed.

KEY WORDS: job shop scheduling problem; flexible job shop scheduling problem; genetic algorithm; tabu search algorithm; deep reinforcement learning; proximal policy optimization algorithm

目　录

第1章 绪论

1.1 研究背景、目的及意义

2013年，德国提出“工业4.0”概念，该战略旨在通过充分利用信息化技术促进产业变革，促使制造业向智能化转型，开启了人类历史上的第四次工业革命。“中国制造2025”与德国“工业4.0”的合作对接渊源已久。2015年5月，国务院正式印发《中国制造 2025》，部署全面推进实施制造强国战略。智能制造是工业 4.0 与中国制造 2025 的共同核心，是实现两大战略目标的关键抓手。它通过将自动化技术、信息技术与先进制造技术相结合，旨在打造柔性化、敏捷化和智能化的生产体系。在智能制造体系中，生产调度是实现资源优化配置、提高生产效率的核心环节，其直接关系到工厂生产目标的实现，如缩短交货时间、提高设备利用率等，在现代制造业中起着至关重要的作用，广泛应用于汽车制造、机械加工、纺织服装等行业[1][2][3]。合理高效的调度方案能够大幅提高生产效率、降低资源浪费、减少仓储物流成本，因此，研究车间调度问题具有重要的工程实际意义。

随着智能制造的快速发展，车间调度问题面临更复杂的挑战和更高的要求，传统的流水车间显然不能应对生产复杂性和动态环境，满足消费者的个性化需求将成为主流，作业车间和柔性作业车间也应运而生。作业车间调度问题(job shop scheduling problem, JSP)是传统流水车间调度问题的扩展，每个工件都具有不同的工艺路线，在每台机器上的加工顺序和时间都不相同。而柔性作业车间调度问题(flexible job shop scheduling problem, FJSP)则是JSP的扩展，每道工序都可以在多台可选机器上加工，相比JSP更加灵活。JSP与FJSP都是典型的NP-hard(Nondeterministic Polynomial - hard) 组合优化问题，随着问题规模的扩大，求解难度会随之增加，传统的人工调度的方法逐渐变得不可行，因此急需开发

出一种能够在短时间内给出优质调度方案的方法，来满足现代化工厂生产的需要。

此类问题的理论研究已经持续数十年之久，很多学者根据不同问题类型提出了很多基准数据集，通过使用基准数据集进行测试，可以衡量算法的性能，测试结果也是业内公认评价算法性能的重要指标之一。现阶段解决车间调度问题的理论方法有很多，其中一些方法可以找到调度方案的最优解，但是需要花费非常多的时间，而另一些方法则是可以在短时间内给出一个相对较好的可行解，这些方法各有优劣，具体使用哪种方法则需要根据实际生产需求来确定。本文的目的是开发出两种高效的求解方法，并分别将它们应用于 JSP 和 FJSP 的求解，从理论上进行生产计划的制定和流程的优化，以满足不同生产实际的需求。

综上所述，从理论上研究经典作业车间调度问题及其扩展问题，不仅在实际应用中能够根据不同的需求，为企业提供优质的生产计划解决方案，提高企业的生产效率，同时在学术上，也为算法的优化做出了贡献，为解决其他类似组合优化问题提供了新思路，促进了计算机与人工智能领域的进步。

1.2 国内外研究现状

本文主要研究 JSP 及其扩展问题 FJSP 这两类调度问题，优化的目标是调度方案的最大完工时间(Makespan)，本节内容将分两个部分进行全面综述，第一部分描述了关于 JSP 的常用求解方法，以及根据问题特征其他学者提出的有效策略；第二部分则是描述了 FJSP 的研究现状，并分析了每种方法的优缺点。

1.2.1 作业车间调度问题的研究现状

解决 JSP 的关键是在机器上为每一道工序安排合适的加工顺序。现阶段解决 JSP 的方法可分为四类：数学优化方法、精确方法、机器学习方法、启发式与元启发式算法。数学优化方法通常先建立相关数学模型，如整数线性规划(Integer Linear Programming, ILP)和混合整数线性规划模型(Mixed Integer Linear Programming, MILP)等[4][5]，然后使用数学求解器（如 CPLEX 和 Gurobi）求解，这类方法的计算效率很大程度上取决于求解器的性能，在大规模或复杂的问题

的求解上可能需要耗费相当长的时间。精确方法例如分支定界法等虽然可以找到最优解，但只能处理问题规模的较小的实例。随着问题规模的扩大，精确算法的运行时间将呈指数级增长，这变得不切实际。因此，已经开发了近似方法。虽然近似方法不一定能找到最优解，但它们可以在一定时间内提供相对较好的解，近年来得到了广泛的应用。机器学习方法一般可分为两类：端到端的深度强化学习方法(deep reinforcement learning, DRL)和强化学习与启发式算法的结合方法[6][7][8]。端到端 DRL 框架是一种基于策略的学习方法，它直接从现有数据中学习，并以端到端的方式做出决策。这种方法避免了手动进行特征设计的局限性，可以更好地适应不同的任务和环境。然而，这种 DRL 框架需要大量的训练数据，且其性能高度依赖于超参数设置。在解决随机 JSP 实例时，其性能往往不稳定，这意味着它无法在不同规模和类型的实例中始终如一地提供出色的结果。另一方面，将强化学习与启发式算法相结合在近些年越来越受欢迎。例如，Q 学习可以与遗传算法相结合，以决定在不同状态下使用哪些交叉和变异算子。然而，这种方法在很大程度上取决于状态和动作空间的设计，并且 Q 学习受到 Q 表规模的限制，无法处理高维复杂状态和动作空间的问题。启发式算法作为近似方法的一个重要分支，在 JSP 中得到了快速发展，并取得了显著成功。遗传算法(genetic algorithm, GA)[9][10][11]、粒子群优化算法(particle swarm optimization, PSO)[12][13][14]、蚁群优化算法(ant colony optimization, ACO)[15][16][17]作为群优化算法，具有很强的全局搜索能力，可以显著减少高维优化问题的计算时间。然而，它们通常对种群的初始化和关键参数的控制很敏感，这使得它们容易陷入局部最优。另一方面，模拟退火(simulated annealing, SA)[18][19][20]和禁忌搜索(tabu search, TS)[21][22]等算法表现出强大的局部搜索能力，并表现出良好的鲁棒性。此外，还有一些其他的算法，例如分布估计算法(estimation of distribution algorithm, EDA)[23][24]，这是一种基于统计原理的新兴随机优化算法，也被应用于求解 JSP。

然而，每种经典的启发式算法都有一定的局限性。通过整合多种启发式方法开发混合算法可以补充它们的优势并减轻弱点，从而提高整体性能。Lin 等人[25]提出了一种混合群智能算法，该算法由 PSO、SA 以及多类型个体增强方案组成。Li 等人[26]提出了一种将 GA 和 TS 相结合的混合算法，用于解决柔性车间调

度问题。Chang等人[27]提出了一种混合GA来解决分布式柔性车间调度问题，并提出了一个新的编码机制来解决无效的作业分配问题。Sun 等人[28]提出了一种混合合作协同进化算法，该算法将 PSO 与 GA 相结合，以提高算法的收敛能力。Zhang 等人[29]提出了一种混合 SA，该方法结合了一种新的免疫机制，并引入了瓶颈水平指标来评估过程的关键性。此外，一些研究人员致力于引入新的搜索策略来提高算法的性能，并取得了一些成功。例如， Chen等人[30]提出了一种基于局部优化策略和改进的旋转角度优化的混合量子算法。Goncalves 等人[31]使用一种名为扩展的 Akers 图形方法，该方法将调度方案转换为一种图形表示，从而将调度问题重新表述为路径规划任务。

基本启发式算法的另一个重要问题是其简单的搜索方法和不确定的有效性。换句话说，他们的搜索方法不是根据 JSP 的问题特征专门设计的, 例如，在遗传算法中，交叉方法大多涉及在两个个体之间随机交换一段遗传密码，而变异方法大多涉及随机选择两个操作来交换处理顺序，或将操作插入随机位置进行处理等。这些搜索方法涉及高度的随机性，缺乏可行性分析，这意味着新生成的解决方案有时甚至不如原方案。这极大地影响了算法的性能。因此，急需开发出一种方法能够根据 JSP 的问题特性，有效指导工序移动，提高搜索的高效性，邻域结构概念的提出解决了此问题。在 JSP 中，邻域结构指的是基于关键路径上工序块的移动方法，关键路径指的是一个完整调度方案中的最长路径，理论上来说，如果能够通过移动关键路径上的工序来减小关键路径的长度，整体的完工时间也会减少，目前对于邻域结构的研究已经取得了不错的进展。Laarhoven等人[32]最初引入了N1邻域结构(Neighborhood 1)，该结构旨在通过交换关键路径上两个相邻工序来生成邻域解。然而，N1 邻域不仅规模庞大，而且包含大量的非改进操作。Matsou 等人[33]提出了 N2 邻域结构，即同时交换两对工序，该结构被集成到 SA 中以供使用。Amico 等人[34]介绍了 N3 和 N4 邻域结构。更先进的发展是 Nowicki 等人于 1996 年提出的 N5 邻域结构[35]。它通过交换关键路径上的头工序及其相邻工序和尾工序及其相邻工序来生成新的解决方案。上述邻域结构在一个块中只生成两个邻域解，它们的规模很小，扩展的邻域解的数量也相对较少。如果邻域结构的规模扩大，随之扩展出的邻域解的数量也会随之增加。N6 邻域结构[36]在 N5 的基础上引入了两个额外的操作：将块

内的操作移动到块的头部或尾部。Nasiri 等人开发的混合禁忌搜索算法使用 N6 邻域结构。类似地，N7 邻域结构[37]通过引入两个附加操作来扩展 N6：将操作从块的头部或尾部移动到块本身。N8 邻域结构是 Gao 等人[38]在 2023 年提出的一项最新研究成果。基于 N7 邻域的结构，它引入了四个附加操作：将内部操作移动到块外，将头部操作移动到临界块外的操作之前，以及将头部操作在临界块外操作之后。这是一项非常大胆的创新，因为 N8 社区涵盖了关键路径之外的流程。这种方法进一步扩大了搜索范围。

然而，邻域结构仅考虑关键路径上的操作移动，相对于非关键路径上大量工序则未被考虑。尽管 N8 邻域考虑了关键路径之外的操作移动，但所有移动都限制在同一台机器上。事实上，这些操作动作受到其作业前操作和作业后操作的约束。如果工件的前道或后道工序的位置不理想，则移动可能会增加完工时间，从而导致无效移动。因此，有必要开发一种新的邻域结构，使关键路径上的操作与其他机器上的操作同时进行。这种方法进一步扩大了邻域解的搜索范围，可以帮助避免低质量的结果。目前，关于多工序精确联动邻域结构的研究很少。赵诗奎等人[39]在 N2 邻域结构的基础上增加了两对运动进行运算，命名为 ICET+2MT，而巴智勇等人[40]在 N7 邻域结构基础上提出了 PN7+2MT。然而，尽管 N8 邻域结构是最新的邻域结构，但尚未有人在此基础上进一步研究。

1.2.2 柔性作业车间调度问题的研究现状

FJSP 作为 JSP 的扩展，不仅要在机器上为工序安排合适的加工顺序，还需要为每一道工序在可选的机器组中挑选出合适的加工机器。现阶段解决 FJSP 的方法可分为四类：数学优化方法、精确方法、机器学习方法、启发式与元启发式算法，这与 JSP 的求解方法基本类似。

在数学优化方法中，许多学者专注于开发 ILP 和 MILP 模型。Özgüven 等人[41]开发了一种 MILP-1 模型，并将其计算效率与另一种模型进行了比较，证明了其优越性。Birgin 等人[42]提出了 MILP 的扩展版本，使用任意的有向无环图定义了操作之间的优先级，与 Özgüveni 等人的模型相比，该模型表现更好。V. Roshanaei 等人[43]开发了两个基于位置和序列的 MILP 模型，并对其大小复杂度、求解有效性和计算效率进行了数值探索。结果表明，由于其结构效率高，所提

出的模型优于比较文献中的其他模型。在精确方法中，B&B 通常用于求解 FJSP。Zhou 等人[44]建立了一个 Petri 网模型，将调度计划转化为从初始标记到最终标记的 Petri 网射击序列，并使用 B&B 方法获得最优调度。Lloyd 等人[44]使用 Petri 网建模，提出了一种改进的 B&B 搜索。Hansmann 等人[45]提出了一种比商业求解器 CPLEX 12.4 和 Gurobi 5.0 更具竞争力的方法。Ahn 等人[47]提出了一种 B&B 方法，通过使用基于转换索引标记的优先级规则和基于定时 Petri 网的各种死锁预防条件，有效地减少了搜索空间。实验结果表明，该方法在各种柔性制造系统实例中的表现优于数学公式和先前的算法。综上所述，数学优化方法和精确方法可以处理 FJSP，在工业和管理中具有重要的实际应用价值。然而即使这些学者针对数学优化方法与精确方法进行了优化，提高了效率，但其仍然受限于问题规模，在解决大规模实例上不可行。

目前启发式与元启发式算法应用较为广泛。包括禁忌搜索算法[48]、模拟退火算法[49]、遗传算法[50][51]、粒子群优化算法[52]、人工蜂群算法[53][54]、差分进化算法[55]和灰狼优化算法[56]等。近年来，大多数学者根据 FJSP 的特点对算法进行了改进。Zarrouk 等人[52]提出了一种两级粒子群算法，用于处理操作到机器的映射和机器上操作的排序。他们还对最优目标函数值实施了下限检查策略，以减少解码次数并节省运行时间。Yuan 等人[57]开发了结合和谐搜索和大邻域搜索的混合模块，分别代表了基于进化和基于约束的方法。混合和谐搜索可以快速生成高质量的解决方案，而大邻域搜索具有很强的强化能力。González 等人[58]介绍了一种应用于 TS 的有效邻域结构，使用快速估计方法来评估邻域解的质量。这种方法类似于 JSP 中的邻域结构研究，是一种高效的搜索方法。Hmida 等人[59]提出了一种求解 FJSP 的爬升微分搜索方法的变体。这种基于微分的方法还利用了块的概念来寻找邻域结构以进行局部搜索。大多数研究人员专注于改进启发式和元启发式算法，例如使用有效的方法来提高初始解的质量，或开发新的邻域结构来提高搜索效率。这些方法确实减少了算法执行时间并提高了性能。但是由于 FJSP 相较于 JSP 更为复杂，求解时时间复杂度更高，由于启发式和元启发式算法的本质仍然是通过连续迭代生成新的解决方案，因此解决大规模问题仍然需要相当长的时间。并且当加工信息发生变化时，则需要重新解决，如果

调度问题的数量增加，理论上求解时间会接近无穷大，这在实际生产中是不可接受的。

那么如何在有限的时间内获取高质量的可行调度方案呢？一些研究人员引入了优先级调度规则(Priority Dispatching Rules, PDRs)用于 FJSP 的求解。系统为每个操作分配优先级，并按顺序安排。调度规则有很多种，如先进先出(First-In First-Out, FIFO)、最短加工时间(Shortest Processing Time, SPT)、剩余工作量最大(Most Work Remaining, MWKR)等。然而，单独应用一个或多个调度规则并不适合解决 FJSP。首先，如果使用基于贪婪的思想来应用调度规则，那么这可能是目光短浅的。其次，决策的产生主要源自于每个时间步中满足条件的工序和机器信息，而全局信息在很大程度上被忽略了。此外，调度规则的应用在很大程度上依赖于从专家系统中获取知识，或通过广泛的试错实验来获得经验。但这与启发式算法的工作原理没有本质上的区别，启发式算法的运行也会随着时间的推移来提高解的质量。这促使本文开发了一种新的 FJSP 求解方法，能够使用现有系统，对于任意的加工信息都能够实时决策出高质量调度方案。随着机器学习的出现，这个问题似乎有了解决方案。

随着计算机科学的进步，近年来越来越多的研究人员将强化学习(RL)和 DRL 应用于解决组合优化问题，取得了值得称赞的结果，特别是在处理大型复杂问题时。端到端的 DRL 框架将 FJSP 的求解过程转化为一个马尔科夫决策过程(Markov Decision Process, MDP)，智能体通过不断与环境交互并从中接收反馈来改善其行为。传统的 RL 在处理高维状态空间和动作空间时表现无力，而 DRL 通过引入深度神经网络作为代理，有效地改善了这一问题，它可以从高维状态空间中提取特征，并有效地处理各种数据。这使得深度强化学习在复杂环境中得到了更广泛、更有效的应用。表 1-1 总结了近年来使用 RL 和 DRL 方法解决调度问题的相关文献。

表 1-1 近年来使用 RL 和 DRL 方法解决调度问题的相关文献

方法	文献作者与年份	机器学习方法	神经网络类型	问题
端到端的 DRL 框架	[61] Junyoung Park, et al., 2020	PPO	GNN	JSP
	[61] Hameed MSA, et al., 2023	PPO	GNN	JSP
	[62] Bao-an Han, et al., 2020	Deep Q-learning	CNN	JSP
	[63] Wenquan Zhang, et al., 2024	Deep Q-learning	GAT	FJSP
	[64] Runqing Wang, et al., 2023	Actor-Critic	DAN	FJSP
	[61] Wen Song, et al., 2023	PPO	GNN	FJSP
	[66] Kun Lei, et al., 2022	PPO	GNN	FJSP
	[61] Cong Zhao, et al., 2023	Actor-Critic	DNN	FJSP
DRL 与启发式算法结合方法	[68] Yibing Li, et al., 2023	Q-learning	-	FJSP
	[68] Atefeh Momenikorbekandi, et al., 2023	Q-learning, SARSA	-	JSP
	[70] Xiaojun Long, et al., 2021	Q-learning	-	FJSP
	[71] Ronghua Chen, et al. 2020	Q-learning, SARSA	-	FJSP
	[72] Shu Luo. 2020	Deep Q-learning	DNN	FJSP

相关方法大致可分为两类：一类是端到端的 DRL 框架，另一类是将启发式算法集成到 DRL 中。Li 等人[68]将 RL 与人工蜂群算法相结合，将环境划分为不同的状态，并在每个时间步时根据这些状态选择行动，通常涉及不同的搜索方法。与随机选择相比，这种方法具有明显的效率优势。Momenikorbekandi 等人[68]将 Q-learning 和 SARSA 强化学习应用于 JSP，并进行了比较，以确定它们各自的优势。Long 等人[70]也将 RL 与人工蜂群算法相结合，基于 RL 智能地选择蜂群更新维度，从而提高了收敛速度。Chen 等人[71]将 RL 与 GA 相结合，以调整算法的关键参数。Luo 等人[72]使用双深度 Q 网络来学习调度规则，并将其应用于求解动态 FJSP，证明了其优于传统的 Q-learning。近年来，对端到端 DRL 框架的研究越来越多。Park 等人[60]使用图神经网络(Graph Neural Network, GNN)来学习，将 JSP 的空间结构表示为图的节点特征，使用析取图来表示调度计划。MSA 等人[61]、Song 等人[64]和 Lei 等人[66]也采用了类似的方法。MSA 使用 GNN 提取特征，为欧几里德空间中的当前生产状态提供丰富的编码，其核心是应用于 GNN 的自定义消息传递算法。Song 使用基于 GNN 的架构来捕捉作业和机器之间的复杂关系。Lei 则是采用了一种多目标策略优化算法来学习与操作选择和机器选择相对应的两个子策略。在 Han 等人[62]的框架中，调度过程被表示为析取图，从而转化为一个多阶段序贯决策问题，使用卷积神经网络近似状态-动作值。Zhang 等人[63]使用图注意力网络从析取图中提取全局状态信息，并使用 Transformer 编码器定量捕捉机器之间的竞争关系，从而改善动作和状态的表示。Wang 等人[64]将用于深度特征提取的自关注模型与用于可扩展决策的 DRL 相结合，准确简洁地表示了操作和机器之间的复杂关系。他们提出了一种由多个相

互连接的操作消息注意力块和机器消息注意力块组成的双注意力网络。Zhao 等人[67]采用了受传统调度规则启发的角色关键架构，利用综合特征集准确表示系统状态，并设计了八组动作，在测试中表现出优异的性能。

1.3 现有研究的问题分析与总结

从现有的研究中可以看出，JSP 与 FJSP 是当今作业车间调度问题的研究热点，其理论研究具有重要意义。JSP 与 FJSP 的研究方法可分为四类：数学优化方法、精确方法、机器学习方法、启发式与元启发式算法。前两种方法显然不足以满足现代智能化工厂的制造需求，故机器学习方法、启发式与元启发式算法应成为现阶段研究的重点。JSP作为最基础的问题，其求解的时间复杂度相对较低，故使用启发式与元启发式算法来求解，目标是计算出相关实例的最优解。而 FJSP 是 JSP 的扩展，计算的时间复杂度相对较高，故使用端到端的 DRL 方法，在现代工厂中个性化的柔性生产中，面对不断变化的生产信息，能够快速给出高质量的调度方案，提高生产效率。然而，现有的研究还存在以下几点不足：

(1) 针对 JSP 问题，启发式算法求解的关键在于算法的搜索方法，邻域结构是业内较为先进的研究，现有邻域结构中存在邻域解数量少和存在大量无法改进 makespan 的邻域解，即工序的无效移动，如何通过理论分析来建立工序移动的有效性，从而提升算法的效率，是一个有前途的研究方向。因此，进一步研究邻域结构，对 JSP 的求解有很大帮助。此外，单一的算法往往受其架构的限制，并不能总是表现出优秀的性能。例如，群智能算法有很强的全局搜索能力，而局部搜索算法则是侧重于局部搜索，在小范围内有很高搜索精度，如果开发出结合群智能与局部搜索算法的混合算法，理论上算法综合性能会有提升。

(2) 针对 FJSP 问题，DRL 框架在很大程度上依赖于状态空间的设计和表示。在大规模问题中，生产信息极其复杂，提取有用信息来表示车间状态是一项重大挑战。另一方面，动作空间的设置也很重要。它确定代理在特定状态下采取的具体行动是否可以在节省生产资源的同时平衡机器负载，从而最大限度地提高总回报。常见的方法是将调度规则直接应用于动作空间的设计。然而，由于调度规则种类繁多，学习所有这些规则显然是不切实际的。大多数学者选择选

择一批经典且具有代表性的调度规则来形成动作空间，而独立开发新的调度规则的研究相对较少。在 FJSP 求解中，决策分为工序和机器两个层级，单一的调度规则往往过于简单，无法同时考虑这两个方面。这是需要解决的问题之一。此外，根据 1.2.2 节中讨论的相关文献显示，该领域的研究主要集中在以下几个方面：其一是注意力机制或多策略网络的应用，用于选择工作和机器。这些方法中的动作空间通常很简单，调度规则直接应用于动作空间。另一方面，新的复合调度规则的发展：受传统调度规则的启发，一些学者开发了新的组合调度规则作为动作空间。然而，关于神经网络架构的创新工作有限。注意力机制不再是一种非常新颖的方法。通过调试超参数来调整隐藏层维度和不同层神经元数量以提高神经网络整体性能的做法已经过时。此外，大多数研究都没有披露详细的超参数。因此，仍然有很多创新的空间。

1.4 本文的主要工作与论文结构

针对 JSP，对现阶段最新的 N8 邻域结构进行了有效性分析，并设计了一种多工序精确联动邻域结构，将其应用于基于遗传算法与禁忌搜索算法的混合算法。此外，还设计了一种基于调度部分重建的全局搜索策略，提升算法的性能。

针对 FJSP，在现有神经元架构的基础上引入了二次项权重参数，设计了一种新神经元架构，将其应用于基于长短期记忆网络的演员网络中，使用双评论家网络分别用于拟合状态值函数和动作值函数。此外，还设计了 7 种复合调度规则应用于动作空间，以及 8 种状态指标应用于状态空间，使用近端策略优化算法进行模型训练。

图 1-1 展示了本文的总体研究框架。

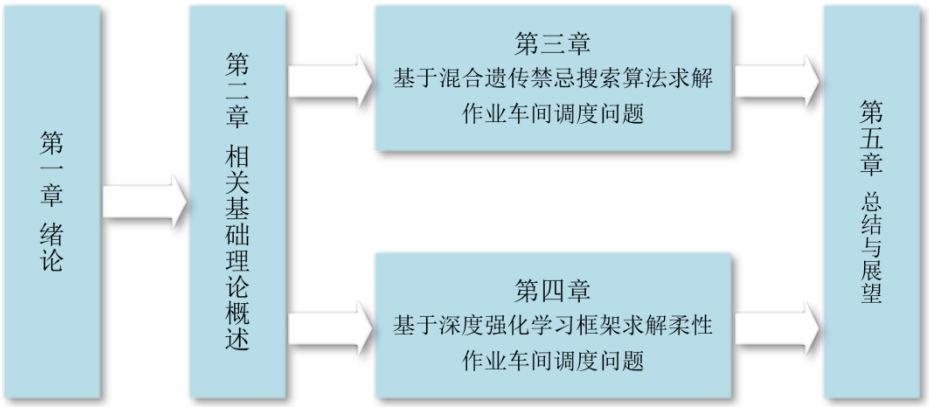


图 1-1 本文总体研究框架

本文剩余章节的主要内容如下：第二章概述了与本文相关的基础理论，包括作业车间调度问题、启发式与元启发式算法、强化学习与深度强化学习；第三章详细描述了基于混合遗传禁忌搜索算法求解作业车间调度问题，包括对现有最新邻域结构中工序移动的有效性分析，所提多工序精确联动邻域结构的具体内容，以及一种基于调度部分重建的全局搜索策略；第四章详细介绍了基于深度强化学习框架求解柔性作业车间调度问题，包括所提新神经元架构与工作原理，以及所提强化学习的相关内容；第五章对全文工作进行总结并展望了未来的研究方向。

第2章 相关基础理论概述

2.1 作业车间调度问题

生产调度，即对生产过程进行作业计划，作为一个关键模块，是整个先进生产制造系统实现管理技术、运筹技术、优化技术、自动化与计算机技术发展的核心。调度问题通常指对生产过程的作业计划，例如工件在机器上的加工顺序、生产批量的划分等[72]。作业车间调度问题是最典型最重要也是最基础的调度问题，其概念最初由美国学者 Johnson[74]于 1954 年提出，他研究的是如何在多个机器上安排多个作业，使得整个生产过程的完工时间(Makespan)最小化。Johnson 提出的主要问题是在两台机器上如何最优地安排作业顺序，解决的是二机作业车间问题的调度方案。随着生产工业的日益复杂，作业车间调度问题逐渐扩展到更一般化的情况，特别是在多个机器和复杂的作业类型下。经典的作业车间调度问题开始成为组合优化领域的一个重要问题。上世纪 60 年代，Garey 和 Johnson 提出了作业车间调度问题的 NP 完全性(NP-completeness)。随着制造业的发展和技术的进步，传统的作业车间调度问题逐渐暴露出一些局限性。作业车间调度问题通常假设每个作业的每个操作只能在特定的机器上完成，这种假设不一定适用于现代复杂的制造环境。例如，在现代制造系统中，某些操作可以由多台机器中的任意一台进行，而不是固定在某台机器上。这种机器的选择灵活性成为了研究的一个新方向。柔性作业车间调度问题正是为了解决这种灵活性需求而提出的，它是作业车间调度问题的扩展，强调了机器选择的柔性。上世纪 80 年代末期，Brucker 等[75]人在研究车间调度时指出，在柔性制造系统中，操作可以由多个机器共同完成，因此作业调度问题应考虑机器的选择灵活性，并提出了柔性作业车间调度问题的模型。进入 21 世纪后，随着信息技术、自动化技术以及智能制造的飞速发展，FJSSP 逐渐成为制造领域中的重要研究课题。

本文的主要研究内容是开发解决 JSP 和 FJSP 的理论方法，首先给出了 JSP 和 FJSP 的问题模型，然后提出了一种混合智能算法用于 JSP 的求解，以及一种深度强化学习方法用于 FJSP 的求解，最后进行了仿真实验，验证了所提方法的有效性和高效性。上述具体内容将在第三章和第四章介绍。

2.2 启发式与元启发式算法

启发式算法和元启发式算法是优化问题中常用的求解方法，在作业车间调度这样的复杂问题中应用广泛。启发式算法是一种通过经验或规则来寻求问题解答的算法，其目标是找到一个“足够好”的解决方案，而不是精确的最优解。启发式算法在 JSP 中的应用通常包括贪心算法、最短加工时间优先、先进先出等，这些方法通过选择某个标准来安排工序在机器上加工，从而获得一个较为有效的调度方案。这种方法的缺点也很明显，即非常依赖于所使用的启发式规则，且通常只能找到局部最优解，该局部最优解可能与全局最优相差甚远。元启发式算法是一类较为通用的优化算法，旨在通过某些启发式策略引导搜索过程，从而高效地寻找问题的近似最优解。元启发式算法通常是非确定性的，它通过对解空间的全局搜索进行引导，避免陷入局部最优解。元启发式算法按照种群规模的设置又可分为群体智能算法和局部搜索算法。遗传算法作为经典群体智能算法在 JSP 求解上应用十分广泛，该方法受达尔文生物进化论的启发，首先将调度方案按照一定规则编码作为个体，并且生成数量较多的个体作为初始种群，能够覆盖大部分解空间，再通过通过模仿适者生存的进化过程，即选择、交叉和变异的操作不断演化，探寻不同区域，从而趋向于全局最优，最终得到较优的调度方案。禁忌搜索是一种局部搜索算法，通过在搜索过程中引入记忆结构，避免算法陷入局部最优解，进而探索到更优的解。禁忌搜索算法的核心思想是通过对当前解的邻域解进行迭代搜索，同时维护一个禁忌表来存储和控制解空间中不可访问的区域。禁忌表主要记录当前搜索过程中不可重复的操作或解，从而避免搜索过程中的循环和局部最优解。

目前，JSP 及其求解方法的研究已经取得了显著进展，但仅仅追求“足够好”的解决方案已无法满足当前的需求。因此，本研究旨在开发一种更加高效的求解方法，力求在基准案例中尽可能地得到最优解。为此，本文结合了遗传算法

在全局搜索中的优势和禁忌搜索算法在局部搜索中的能力，提出了一种混合遗传禁忌搜索算法。该方法的具体内容将在第三章中进行详细介绍。

2.3 强化学习与深度强化学习

强化学习(Reinforcement Learning, RL)是一种基于智能体与环境交互的学习方法，它通过试错的方式来学习决策策略，寻找能够进行最优决策的策略模型，通常以最大化奖励值为目标。在 FJSP 中，选择合适的工序在合适的机器上进行加工可以视为一项决策，强化学习可以用来优化决策过程，找到近似最优的调度方案。在强化学习的框架下，马尔科夫决策过程(Markov Decision Process, MDP)为其提供了数学模型和理论基础。在理解强化学习时，首先需要了解马尔科夫决策过程的基本概念，因为强化学习正是通过 MDP 中智能体与环境的交互来进行学习和决策的，具体内容将在第 2.3.1 节中进行介绍。在学习的过程中，需要设置多项指标用于表征车间的状态，这些指标的交叉组合则会形成高维状态空间，而传统的强化学习如 Q-learning、SARSA 等无法处理这种情况，此时需要引入一种数学方法来拟合数据。深度强化学习(Deep Reinforcement Learning, DRL)结合了深度学习和强化学习的优势，使用深度神经网络作为智能体，逼近策略或价值函数，使得强化学习能够处理更高维度、更复杂的决策问题。DRL 将策略参数化，通过神经网络来表示调度策略，从而提升了强化学习的表达能力。神经网络的主要工作原理是构建一个多层次的、非线性映射函数，将输入数据，即特征，映射到输出结果，它通过学习输入数据与输出结果之间的映射关系，逐步调整其参数，以实现对数据的逼近或预测，其训练过程本质上是一个优化过程。为了更好地理解神经网络是如何进行拟合的，需要了解神经网络的基本架构，即神经元，以及其数学原理。基础的神经元架构及其扩展形式：多层神经网络的具体内容将在第 2.3.2 节中进行介绍。

2.3.1 马尔科夫决策过程

RL 是一种机器学习范式，用于训练智能体在与环境交互的过程中通过试错的方式学习来实现某些目标。MDP 则是强化学习问题的数学形式化，描述了智能体与环境之间的交互模型。MDP 可以用五元组表示法建模：$\{\mathcal{S},\mathcal{A},P,r,\gamma\}$，

$\mathcal{S}$ 为状态空间，表示系统所有可能的状态，在 FJSP 中，状态可以是工件的加工进度，工序的加工时长等；$\mathcal{A}$ 为动作空间，表示智能体在某一状态下可以采取的所有动作的集合，在 FJSP 中，动作可以是安排某一可选工序在某一可选机器上进行加工；P 为状态转移概率，表示智能体从当前状态转移到下一状态的概率；r 为奖励函数，表示智能体在给定状态 s 下执行动作 a 后获得的奖励；γ 为折扣因子，表示未来奖励的重要程度，γ 的取值范围为[0,1]，当 γ 趋近于 0 时，智能体更关注当前时间步的奖励，而当 γ 接近 1 时，智能体则更关注长期回报。

一个完整的 MDP 描述如下：首先，智能体在每个时间步 t 时都存在状态 s_t，并根据状态 s_t 从动作集中选择一个动作 $a_t \in \mathcal{A}$，根据状态转换函数 P，智能体将从当前状态 s_t 转换到下一个状态 s_{t+1}，并同时从环境中获得奖励 r_t，重复上述过程，直至完成每个时间步的决策。$\pi_\theta(s,a)$ 表示智能体在给定状态 s 下选择动作 a 的概率分布函数。RL 的目标是找到一个策略 π_θ，使智能体在与环境交互中的累积奖励最大化，该最大化累积奖励可以用式(2-1)来描述：

$$J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_t \gamma^t r_t] \tag{2-1}$$

其中，r_t 表示时间步 t 时从环境中获得的奖励，$\mathbb{E}_{\pi_\theta}$ 表示在策略 π_θ 下的期望奖励。

MDP 为强化学习提供了一个强大的理论框架，它通过定义状态空间、动作空间、转移概率、奖励函数等要素，描述了智能体与环境的交互过程。强化学习则通过与环境的不断交互和策略更新，逐步学习到最优的决策策略。MDP 中的“马尔科夫性”假设是强化学习的基础，它使得强化学习能够通过当前状态和动作来决定未来的行为，而不需要依赖于历史信息。通过强化学习，智能体可以在 FJSP 等复杂优化问题中，通过智能决策不断提升调度质量。

2.3.2 基础神经元架构与神经网络

受动物中枢神经系统（尤其是大脑）的启发，研究人员开发了一种数学模型，该模型模拟了用于估计或近似函数的生物神经网络的结构和功能，称为人

工神经网络。在人工神经网络中，神经元是最基本的结构单元。每个神经元都与其他神经元相连，传输和处理复杂的信息。神经元能够接收多个输入信号，并通过加权求和后，经过一个激活函数产生输出。图2-1为一个标准神经元模型的示意图。

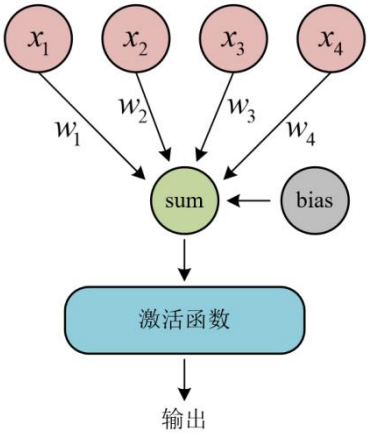


图 2-1 神经元模型示意图

其中，每个输入 x_1, x_2, x_3, x_4 通过相应的权重 w_1, w_2, w_3, w_4 连接到神经元，其工作原理如下：首先，将每个输入 x_i 乘以其对应的权重 w_i，得到加权输入 $w_i \cdot x_i$，接下来，将所有加权输入相加得到线性和 $sum = \sum_{i=1}^{4} w_i \cdot x_i$。然后，将偏置项 $bias$ 添加到线性和中，为模型提供更大的灵活性和更强的拟合能力。最后，将结果输入到激活函数中，激活函数为结果引入了非线性，使模型能够捕捉和理解复杂的映射关系。最后，输出 y 被传递给下一层神经元。通过这种架构，神经元可以对输入执行复杂的非线性变换，从而实现函数的有效估计或近似。式(2-2)描述了此过程的数学模型：

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{2-2}$$

其中，y 为神经元的输出，x_i 为神经元的输入，w_i 为 x_i 的权重，b 为偏置项，f 则是激活函数，如 sigmoid、ReLU 等，引入非线性。简化起来，单一神经元的计算方式可以类似于线性函数，如式(2-3)所示：

$$y = ax + b \tag{2-3}$$

然而，上述神经元架构只是一个非常简单的线性模型，而神经网络的优势在于它由非常多的神经元组成，多个神经元构成的集合称为“层”，层是神经网络的基本构成单位，一个完善的神经网络由多个层构成，通过多层结构进行深度学习，例如多层感知机(Multilayer perceptron, MLP)。MLP 由多个神经元组成，这些神经元分布在不同的层中，首先，输入层负责接收输入的数据，并将其传递至下一层隐藏层，隐藏层通过多层神经元进行计算，提取输入数据的高阶特征，最后传递至输出层，生成最终的预测结果。其中每一层的神经元都接受上一层的输出，并通过加权求和以及激活函数生成输出。通过多层神经网络的堆叠，网络能够捕捉复杂的输入与输出之间的非线性关系。图 2-2 给出了 MLP 示意图。

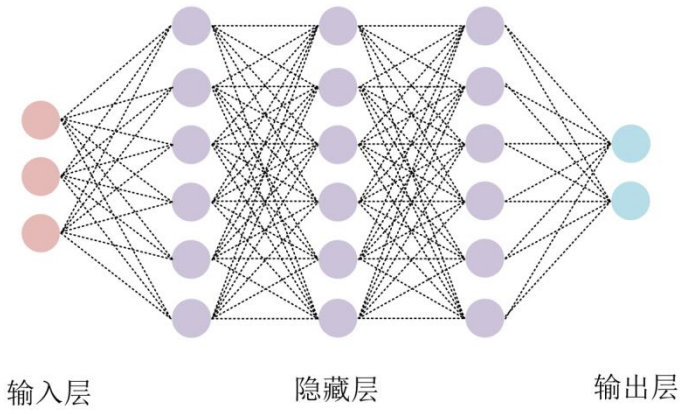


图 2-2 MLP 示意图

第3章 基于混合遗传禁忌搜索算法求解作业车间调度问题

3.1 引言

制造业是国民经济的支柱。车间调度问题（JSP）是生产制造领域的一个关键问题，具有重要的研究价值。介绍了一种混合遗传禁忌搜索算法（HGTSA），该算法结合了遗传算法（GA）的全局搜索能力和禁忌搜索（TS）的局部搜索能力，以最小化最大完成时间为目标。基于对突出邻域结构的分析，本文从理论上建立了识别关键块内操作无效移动的标准，设计了一种涉及三对操作的多操作联合移动邻域结构，并将其表示为 N8 转置和 2 机传输（N8T+2MT），可以指导操作的有效移动。此外，为了避免过早收敛，引入了一种基于调度部分重建和其他一些增强遗传算子的搜索方法。通过在 JSP 基准上与其他最先进的算法进行比较，验证了 HGTSA 的有效性，实验结果证明了 N8T+2MT 的具体优势。

3.2 问题描述

JSP 可以描述如下，给定含有 n 个工件的集合 $\mathcal{J}=\{J_1,J_2,...J_n\}$ 和 m 台机器的集合 $\mathcal{M}=\{M_1,M_2,...M_m\}$，每个工件 $J_i\in\mathcal{J}$ 都有 n_{J_i} 道工序，这些工序必须按照既定的顺序 $\mathcal{O}_{J_i}=\{O_{J_i,1},O_{J_i,2},...,O_{J_i,n_{J_i}}\}$ 进行加工，每道工序都需要在指定机器上加工，$T_{J_i,j}$ 表示工序 $O_{J_i,j}$ 的加工时间。此外，还有以下约束：

(1)每台机器一次只能处理一项作业。

(2)操作处理开始后不能中断。

(3)所有流程之间没有准备时间。

(4)所有机器和所有作业从时间 0 开始可用。

(5)同一作业的操作应符合工艺路线的限制。

(6)强制所有操作的开始时间都是非负的。

在本文中，JSP 优化的目标是最小化完工时间(Makespan)，如式(3-1)所示。

$$C_{\max} = \max\{C_i\}, i \in [1,2,...,m] \tag{3-1}$$

表 3-1 展示了一个包含 4 个工件和 4 台机器的 JSP 实例加工信息。每个单元格中的数字表示每个工序的加工时间和相应的机器。图 3-1 显示了在这些处理约束下一个随机生成的可行调度的 Gantt 图。Gantt 图的水平轴表示时间，而垂直轴表示所在机器。相同的颜色表示同一工件的不同工序，而不同的颜色用于区分不同的工件。机器上的每个工序都由一个矩形块表示，矩形内显示工件编号。矩形两端的数字表示相应工件的开始和结束时间。

表 3-1 一个包含 4 工件 4 机器的 JSP 实例信息

Job / J_i	加工时间/机器编号			
	$O_{J_i,1}$	$O_{J_i,2}$	$O_{J_i,3}$	$O_{J_i,4}$
1	3/1	3/2	2/3	6/4
2	1/1	5/4	3/3	4/2
3	3/2	2/1	3/4	5/3
4	3/4	2/3	4/2	1/1

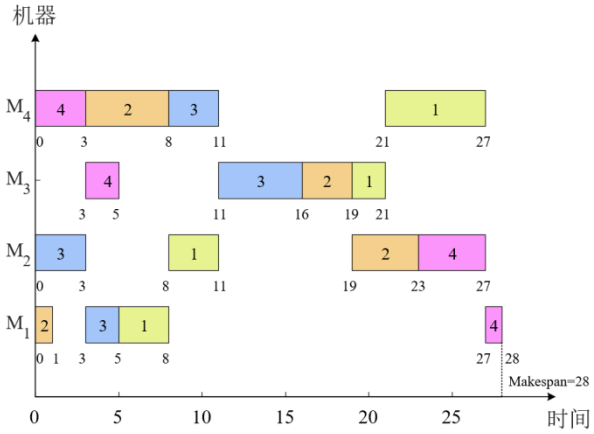


图 3-1 调度方案 Gantt 图

3.3 多工序精确联动邻域结构

关键路径的概念在 JSP 中至关重要。关键路径表示在生产加工中完成所有工序所需的最短时间。它由一系列工序组成，每个工序之间没有空闲时间。任何工序都不能在不影响进度的情况下延迟。这意味着关键路径上的每一项工序都会直接影响整个项目的完工时间。关键路径上的任何工序延迟都将导致整个

调度完工时间的延迟。在实际的制造环境中，关键路径有助于识别生产过程中的瓶颈工序或机器，并确定哪些工序对整个生产进度的影响最大。它帮助管理者识别和管理生产过程中的关键要素，促进更有效的生产计划和资源利用。最终，这将降低成本，提高生产效率，并加强资源管理。

在理论研究中，开发一种通过工序块的移动来缩短关键路径长度的搜索方法，在调度优化中起着至关重要的作用。首先根据不同的机器将关键路径划分为几个部分，位于关键路径上同一机器上的相邻工序称为关键块。此时，引入了邻域结构的概念，邻域结构的主要思想是利用关键块中工序的移动来利用机器空闲时间，从而缩短关键路径的长度，最终缩短完工时间。邻域结构的引入标志着 JSP 的搜索方法从盲目搜索和随机搜索转向了工序层面的有序搜索。在其他学者的研究中，N5、N6、N7 和 N8 邻域结构非常经典，在第 1.2.1 节中，详细介绍了它们邻域结构中包含的内容，其指导工序移动的示意图如图 3-2 所示。

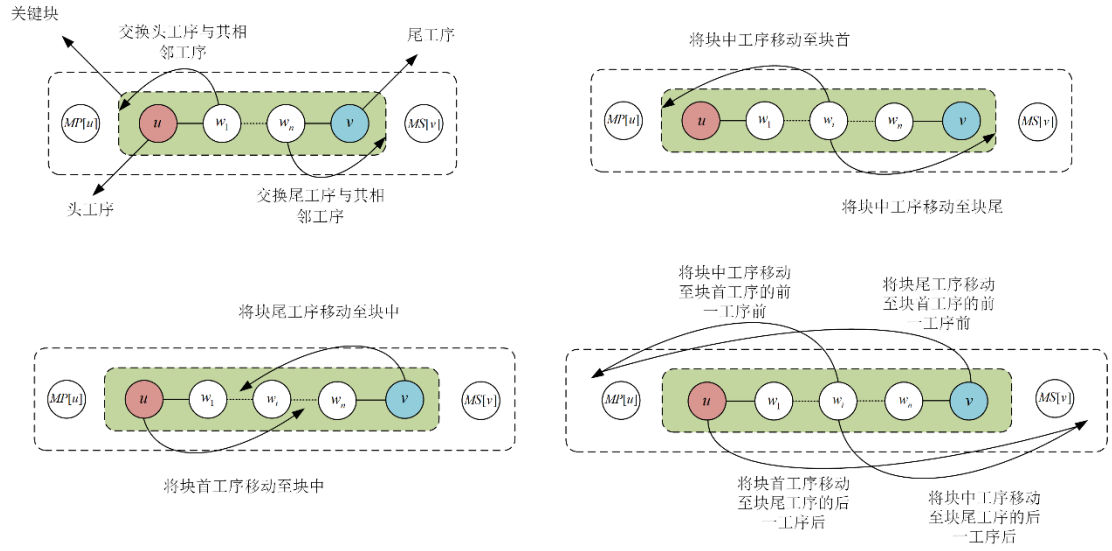


图 3-2 N5~N8 邻域工序移动示意图

N8 邻域结构因其多种工序移动模式而显得较为复杂，可能会产生众多邻域解。然而，由于缺乏足够的理论分析，这种结构在搜索过程中可能会产生大量无效的解决方案。本文对 N8 邻域结构进行了理论分析，提出了临界操作动作有效的条件，并提供了证明。随后，引入了 N8 邻域结构的扩展优化，提出了一种基于 N8 邻域的新型多工序精确联动邻域结构。表 3-2 给出了邻域结构中常用的符号及其说明。

表 3-2 邻域结构中常用的符号及其说明

符号	描述	计算公式
$MP[u]$	工序 u 的机器前一工序	-
$MS[u]$	工序 u 的机器后一工序	-
$JP[u]$	工序 u 的工件前一工序	-
$JS[u]$	工序 u 的工件后一工序	-
d_u	工序 u 的加工时间	-
r_u	工序 u 的头长度	-
q_u	工序 u 的尾长度	$q_u = makespan - r_u$
$s^E[u]$	工序 u 的最早开工时间	-
$c^E[u]$	工序 u 的最早完工时间	$c^E[u] = s^E[u] + d_u$
$s^L[u]$	工序 u 的最晚开工时间	-
$c^L[u]$	工序 u 的最晚完工时间	$c^L[u] = s^L[u] + d_u$

3.3.1 N8 邻域结构的有效性分析

从上述内容来看，N8 邻域结构的大小明显大于其他邻域结构。然而，并非所有的邻域解都能缩短完工时间。为了解决这个问题，本文提供了一些分析，包括 6 个命题和证明。这里使用了一种近似评估方法来几段受移动影响的所有工序总长度，记为 $C'_{\max}$，如果 $C'_{\max}$ 比这些工序的原总长度 $C_{\max}$ 还要大，那么此次工序移动并不能减小完工时间。

命题 1：如果工序 u 和 v 在同一机器上的同一个关键块内，u 是块首工序，v 是块尾工序，对于任意的工序 $w \in [w_1, MS[v]]$，如果 $r_{JP[w]} + d_{JP[w]} + d_u \geq r_w$，那么将 u 移动到 $MS[v]$ 后不能减小完工时间。示意图如图 3-3 所示。

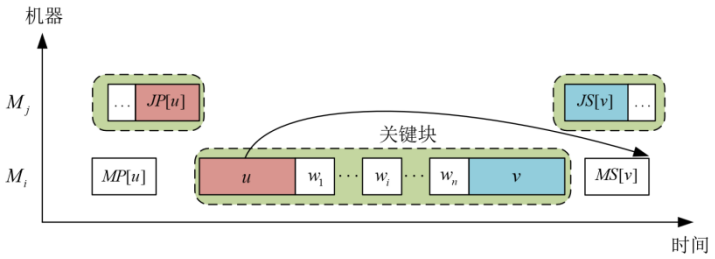


图 3-3 命题 1 工序移动示意图

证明： 当 u 移动到 $MS[v]$ 后时，受影响的工序集为 $\{w_1,...,w_i,...,w_n,v,MS[v],u\}$，每个受影响工序的开始和结束时间以及 $C'_{\max}$ 的计算过程如式(3-2)和式(3-3)所示。

$$\begin{cases} r'_{w_1}=\max\{r_{MP[u]}+d_{MP[u]},r_{JP[w_1]}+d_{JP[w_1]}\} \\ ... \\ r'_{w_n}=\max\{r'_{w_{n-1}}+d_{w_{n-1}},r_{JP[w_n]}+d_{JP[w_n]}\} \\ r'_v=\max\{r'_{w_n}+d_{w_n},r_{JP[v]}+d_{JP[v]}\} \\ r'_{MS[v]}=\max\{r'_v+d_v,r_{JP[MS[v]]}+d_{JP[MS[v]]}\} \\ r'_u=\max\{r'_{MS[v]}+d_{MS[v]},r_{JP[u]}+d_{JP[u]}\} \\ q'_u=\max\{q_{JS[u]},q_{MS[MS[v]]}\}+d_u \\ q'_{MS[v]}=\max\{q_{JS[MS[v]]},q'_u\}+d_{MS[v]} \\ q'_v=\max\{q_{JS[v]},q'_{MS[v]}\}+d_v \\ q'_{w_n}=\max\{q_{JS[w_n]},q'_v\}+d_{w_n} \\ ... \\ q'_{w_1}=\max\{q_{JS[w_1]},q'_{w_2}\}+d_{w_1} \end{cases} \tag{3-2}$$

$$C'_{\max}\geq\max\{r'_{w_1}+q'_{w_1},...,r'_{w_n}+q'_{w_n},r'_v+q'_v,r'_{MS[v]}+q'_{MS[v]},r'_u+q'_u\} \tag{3-3}$$

以工序 w_i 的视角来看，移动之前的完工时间为：

$$C_{\max}\geq r_{w_i}+d_{w_i}+...+d_{w_n}+d_v+d_{MS[v]}+q_{MS[MS[v]]} \tag{3-4}$$

移动之后的完工时间为：

$$C'_{\max}\geq r'_{w_i}+q'_{w_i} \tag{3-5}$$

此时可以计算出 w_i 的新头尾长度：

$$r'_{w_i}\geq r_{JP[w_i]}+d_{JP[w_i]} \tag{3-6}$$

$$q'_{w_i} \geq q_{MS[MS[v]]} + d_{MS[v]} + d_v + d_{w_n} + ... + d_{w_i} \tag{3-7}$$

综合式(3-6)和式(3-7)可得：

$$C'_{\max} \geq r_{JP[w_i]} + d_{JP[w_i]} + d_{w_i} + ... + d_{w_n} + d_v + d_{MS[v]} + d_u + q_{MS[MS[v]]} \tag{3-8}$$

如果$r_{JP[w]} + d_{JP[w]} + d_u \geq r_w$，则有：

$$C'_{\max} \geq r_{w_i} + d_{w_i} + ... + d_{w_n} + d_v + d_{MS[v]} + q_{MS[MS[v]]} \tag{3-9}$$

综上所述：$C'_{\max} \geq C_{\max}$，命题 1 成立。

命题 2：如果工序u和v在同一机器上的同一个关键块内，u是块首工序，v是块尾工序，对于任意的工序$v \in [v_1, v_i]$，如果$r_{JP[v]} + d_{JP[v]} + d_u \geq r_v$，那么将$u$移动到$v$后不能减小完工时间。示意图如图 3-4 所示。

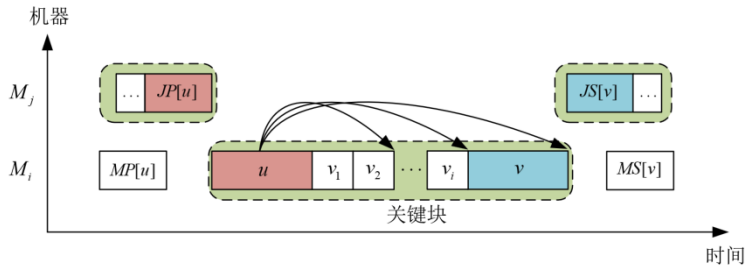


图 3-4 命题 2 工序移动示意图

此命题的证明与命题 1 类似，这里将不再赘述。

命题 3：如果工序u和v在同一机器上的同一个关键块内，u是块首工序或块中工序，v是块尾工序，对于任意的工序$u_i \in u$，如果$q_{JS[u_i]} \geq q_{JS[v]}$，那么将$u_i$移动到$v$后不能减小完工时间。示意图如图 3-5 所示。

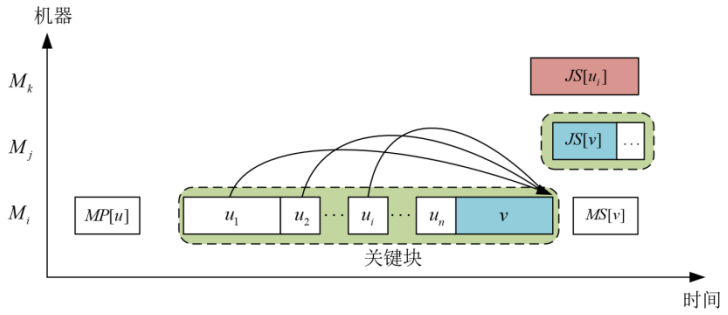


图 3-5 命题 3 工序移动示意图

证明：移动之前的完工时间$C_{\max} = r'_{u_i} + q'_{u_i}$，当$u_i$移动到$v$后，新的头尾长度为：

$$r'_{u_i} \geq r_{MP[u_i]} + d_{MP[u_i]} + d_{u_{i+1}} + ... + d_{u_n} + d_v \tag{3-10}$$

$$q'_{u_u} \geq q_{JS[u_u]} + d_{u_i} \tag{3-11}$$

由式(3-10)和式(3-11)可得：

$$C'_{\max} \geq r_{MP[u_i]} + d_{MP[u_i]} + d_{u_i} + d_{u_{i+1}} + ... + d_{u_n} + d_v + q_{JS[u_i]} \tag{3-12}$$

如果$q_{JS[u_i]} \geq q_{JS[v]}$，则有：

$$C'_{\max} \geq r_{MP[u_i]} + d_{MP[u_i]} + d_{u_i} + d_{u_{i+1}} + ... + d_{u_n} + d_v + q_{JS[v]} \tag{3-13}$$

综上所述：$C'_{\max} \geq C_{\max}$，命题 3 成立。

命题 4：如果工序u和v在同一机器上的同一个关键块内，u是块首工序或块中工序，v是块尾工序，如果$r_{JP[v]} + d_{JP[v]} < r_u - d_{MP[u]}$，那么将$v$移动到$MP[u]$前不能减小完工时间。示意图如图 3-6 所示。

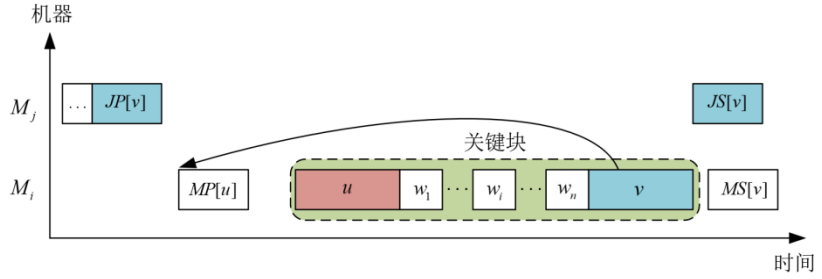


图 3-6 命题 4 工序移动示意图

证明：从工序$JP[v]$的结束时间到工序v的结束时间，该路径段被称为关键路径上的扩展路径。工序集$\{MP[u], u, w_1, ..., w_i, ..., w_n, v\}$构成扩展路径上的工序块，称为扩展块，扩展路径由这些扩展块和空闲时间形成。在某种程度上，扩展块和关键块具有相似的特征。如果可以通过在扩展块内移动操作来减少扩展路径的长度，从而利用扩展路径上的空闲时间进行操作处理，并同时减少关键路径的长度的话，那么这种工序移动的有效性就成立了。

当v移动到$MP[u]$前，会出现一条新的扩展路径。扩展路径上的扩展块为：$\{v, MP[u], u, w_1, ..., w_i, ..., w_n\}$，现在需要按如下方式重新计算新扩展路径($L$)的长度。基于扩展块，移动前扩展路径的长度为：

$$L = d_v + d_{MP[u]} + d_u + d_{w_i} + ... + d_{w_n} + L_f \tag{3-14}$$

空闲时间 L_f 在移动之前为：

$$L_f = [r_{MP[u]} - (r_{JP[v]} + d_{JP[v]})] + [r_u - (r_{MP[u]} + d_{MP[u]})] \tag{3-15}$$

工序移动后的扩展路径长度为：

$$L' = d_v + d_{MP[u]} + d_u + d_{w_i} + ... + d_{w_n} + L'_f \tag{3-16}$$

空闲时间 L'_f 在移动之后为：

$$L'_f = L_f - d_v \tag{3-17}$$

显然，时间的值一定是非负的。如果计算结果显示 $L'_f \le 0$，则表明移动前扩展路径上的空闲时间已被充分利用。如果 $L'_f > 0$，这意味着扩展路径中仍有空闲时间。如果 $r_{JP[v]} + d_{JP[v]} < r_u - d_{MP[u]}$，则 $L_f > 0$，且 $d_v > 0$ 是恒成立的，故 $L'_f < L_f$。综上所述，命题 4 成立。

命题 5：如果工序 u 和 v 在同一机器上的同一个关键块内，u 是块首工序或块中工序，v 是块尾工序，对于任意的工序 $u_i \in u$ 满足 $q_{JS[u_i]} + d_v \ge q_{u_i} - d_{u_i}$，那么将 v 移动到 u_i 前不能减小完工时间。示意图如图 3-7 所示。

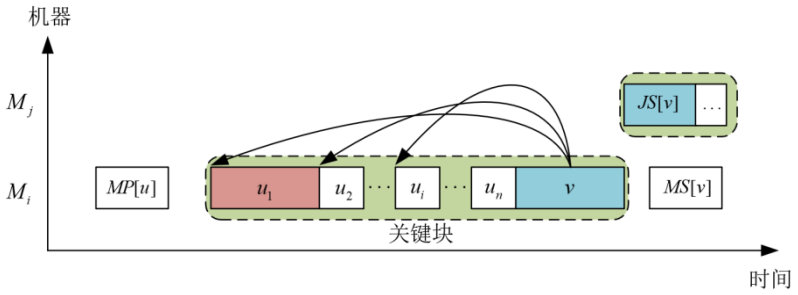


图 3-7 命题 5 工序移动示意图

证明：将工序 v 移至 u_i 前，受影响的工序为 $\{v, u_i, ..., u_n\}$，每个受影响工序的开始和结束时间以及 $C'_{\max}$ 的计算过程如式(3-18)和式(3-19)所示。

$$\begin{cases} r'_v = \max\{r_{MP[u]} + d_{MP[u]}, r_{JP[v]} + d_{JP[v]}\} \\ r'_{u_1} = \max\{r'_v + d_v, r_{JP[u_1]} + d_{JP[u_1]}\} \\ \ldots \\ r'_{u_i} = \max\{r'_{u_{i-1}} + d_{u_{i-1}}, r_{JP[u_{i-1}]} + d_{JP[u_{i-1}]}\} \\ \ldots \\ r'_{u_n} = \max\{r'_{u_{n-1}} + d_{u_{n-1}}, r_{JP[u_n]} + d_{JP[u_n]}\} \\ q'_{u_n} = \max\{q_{MS[v]}, q_{JS[u_n]}\} + d_{u_n} \\ \ldots \\ q'_{u_i} = \max\{q'_{u_{i+1}}, q_{JS[u_{i+1}]}\} + d_{u_{i+1}} \\ \ldots \\ q'_{u_1} = \max\{q'_{u_2}, q_{JS[u_1]}\} + d_{u_1} \\ q'_v = \max\{q'_{u_1}, q_{JS[v]}\} + d_v \end{cases} \tag{3-18}$$

$$C'_{\max} \geq r'_{u_i} + q'_{u_i} \tag{3-19}$$

根据式(3-18)推导得到工序u_1的近似头尾长度为：

$$r'_{u_i} \geq r_{MP[u]} + d_{MP[u]} + d_v + d_u + \ldots + d_{u_{i-1}} = r_{u_i} + d_v \tag{3-20}$$

$$q'_{u_i} \geq q_{JS[u_i]} + d_{u_i} \tag{3-21}$$

由式(3-20)和式(3-21)可知：

$$C'_{\max} \geq r_{u_i} + d_v + q_{JS[u_i]} + d_{u_i} \tag{3-22}$$

若$q_{JS[u_i]} + d_v \geq q_{u_i} - d_{u_i}$成立，则有：

$$C'_{\max} \geq r_{u_i} + (q_{u_i} - d_{u_i}) + d_{u_i} = r_{u_i} + q_{u_i} = C_{\max} \tag{3-23}$$

综上所述，$C'_{\max} \geq C_{\max}$，命题 5 成立。

命题 6：如果工序u和v在同一机器上的同一个关键块内，u是块首工序或块中工序，v是块中工序或块尾工序，若工序u之后的任意工序v_i满足$r_{JP[v_i]} + d_{JP[v_i]} \geq r_u$，那么将$v_i$移动到$u$前不能减小完工时间。示意图如图 3-8 所示。

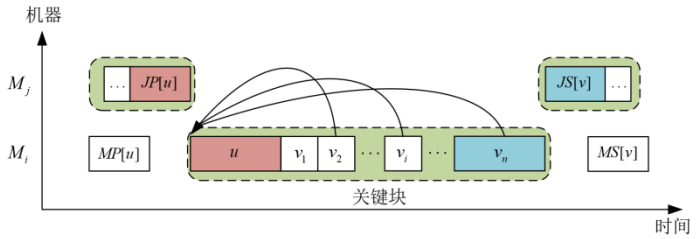


图 3-8 命题 6 工序移动示意图

证明：以工序 v_i 的视角来看，移动前的完工时间为：

$$C'_{\max} \geq r'_v + q'_v \tag{3-24}$$

由式(3-24)可推导得：

$$r'_{v_i} \geq r_{JP[v_i]} + d_{JP[v_i]} \tag{3-25}$$

$$q'_{v_i} \geq q_{MS[v_i]} + d_{v_{i-1}} + ... + d_{v_1} + d_u + d_v \tag{3-26}$$

如果 $r_{JP[v_i]} + d_{JP[v_i]} \geq r_u$，结合式(3-25)和式(3-26)可得：

$$C'_{\max} \geq r_u + q_u + d_{v_1} + ... + d_{v_i} + q_{MS[v_i]} = C_{\max} \tag{3-27}$$

综上所述，$C'_{\max} \geq C_{\max}$，命题 6 成立。

N8 邻域结构中参与移动的工序数量多且移动范围较 N5-N7 邻域更广，能从多个角度来利用机器的空闲时间。以上基础理论分析，识别了 N8 邻域结构中有效与无效移动操作，同时也为更复杂的多工序精确联动邻域结构的设计提供了理论基础。

邻域结构的根本在于引导关键工序对机器空闲时间进行利用，从而减小 makespan。在图 3-9 中，u 为关键块的块首工序，v 为块尾工序，若将关键块中的工序全部移走，则会形成图中所示的空闲时间 III：$[r_u, r_v + d_v]$，而 u 与其机器前序工序 $MP[u]$ 之间存在空闲时间 I：$[r_{MP[u]} + d_{MP[u]}, r_u]$，此时，若按照 N8 邻域的工序移动规则中将块尾工序移至块首这一操作进行工序块的移动，即将 v 移至 u 前进行加工，似乎可以利用空闲时间 I，但这取决于的 v 的工件前序工序 $JP[v]$ 的所在位置，同时也取决于工序的最早开工时间与最晚开工时间。假设图中 $JP[v]$ 均为最早开工，若 $JP[v]$ 处于①或②位置，将 v 移至 u 前进行加工并不能利用空闲时间 I，若 $JP[v]$ 处于③或④位置，将 v 移至 u 前进行加工则可以利

用空闲时间 I；同理，假设图中 $JS[u]$ 均为最晚开工，若 $JS[u]$ 处于①或②位置，将 u 移至 v 后进行加工并不能利用空闲时间 II，若 $JS[u]$ 处于③或④位置，将 u 移至 v 后进行加工则可以利用空闲时间 II。此时，若 $JP[v]$ 与 $JS[u]$ 均处于理想情况，即 $JP[v]$ 与 $JS[u]$ 均位于④位置，将 v 与 u 互换进行加工，则可以同时利用空闲时间 I 和 II，这意味着原调度在此位置需要用空间 IV 进行加工，调整后只需要用空间 III。同理，N8 邻域扩展的解集，IV 空间可扩展至 V 空间（即 $JP[v]$ 与 $JS[u]$ 位于⑦位置）甚至更广。

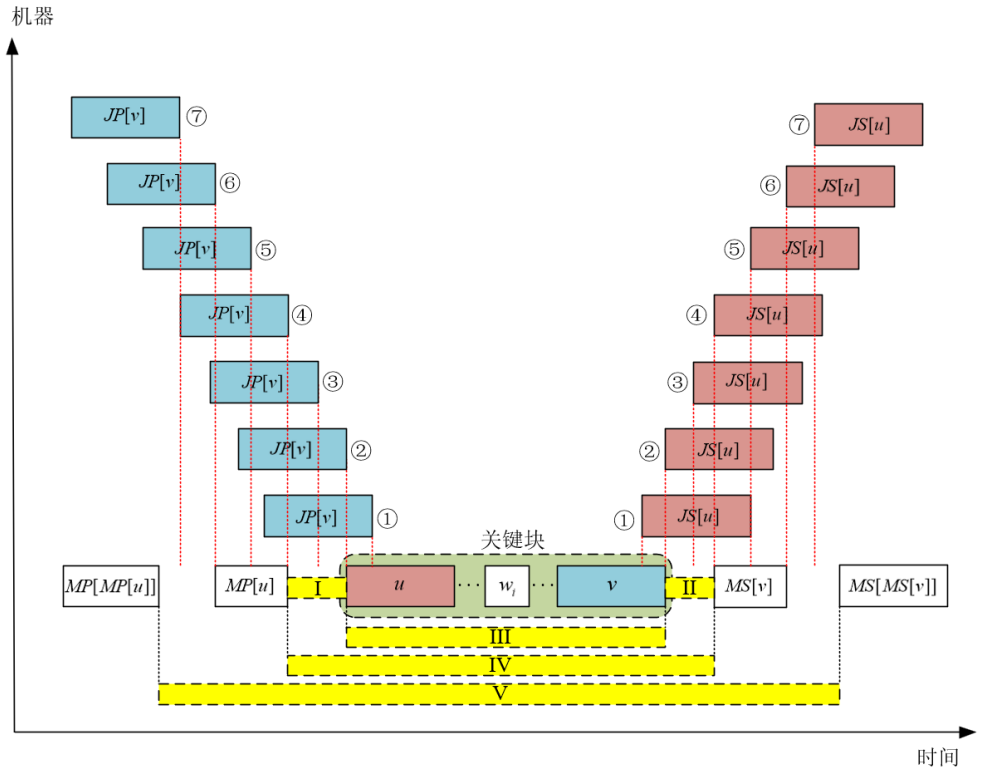


图 3-9 邻域结构中工序移动来利用空闲时间原理解析图

但并不是任何时候都可以利用空闲时间，如当 $JP[v]$ 与 $JS[u]$ 位于①、②位置时，无法利用空闲时间扩展至 III 空间，当 $JP[v]$ 与 $JS[u]$ 位于⑤、⑥位置时，无法利用空闲时间扩展至 V 空间，等等。

当 $JP[v]$ 位于①位置时，且此时 $JP[v]$ 是按照最早开工时间进行加工，若将其与其机器前序工序 $JP[JP[v]]$ 进行交换，如图 3-10 所示，则此时将 v 移至 u 前进行加工便可以利用到空闲时间 I；同理，当 $JS[u]$ 位于①位置时，且此时 $JS[u]$

是按照最晚开工时间进行加工，若将其与其机器后续工序 $MS[JS[u]]$ 进行交换，则此时将 u 移至 v 后进行加工便可以利用到空闲时间 II。上述操作即为多工序精确联动。

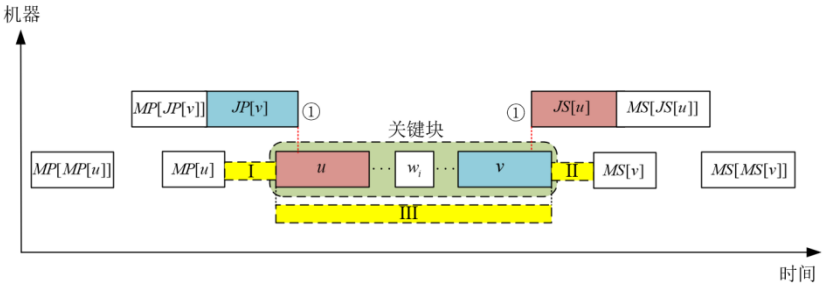


图 3-10 多工序精确联动示意图

3.3.2 多工序精确联动邻域结构 N8T+2MT 的设计

目前，在求解 JSP 的高效算法中，基于邻域结构的搜索方法很常见。一些学者还提出了几种多工序精确联动邻域结构，如 ICET+2MT[81]、PN7+2MT[82]、sCET+2MT[39]等，并对表 3-3 中的缩写进行了解释。

表 3-3 邻域结构相关缩写及其解释

缩写	解释
CET	Transpose the edge operations on critical block (Critical End Transpose)
MT	Transpose the operations on another machine (Machine Transpose)
N8T+2MT	N8 neighborhood structure Transpose and 2 Machine Transpose

通常，工序的选择是随机的，然后对它们进行测试，看看它们是否符合联动的条件。如果不符合条件，这些工序将被放弃。这可能会导致一个问题，即所选工序可能都不符合条件，而符合条件的工序可能不会被选择。在相同的搜索尝试次数下，这可能会降低算法的搜索性能。本研究针对这一问题，在前面分析的基础上，提出了 N8T+2MT 多工序精确联动邻域结构，分为 8 种情况，具体实现如下。

情况 1：块首工序 u 移至块尾工序 v 的机器后序工序 $MS[v]$ 后，工序联动方式如图 3-11 所示，具体步骤如下：

(1)令 msu 为关键块中某块中工序，如果 $JP[msu]\neq\varnothing$，并且满足 $r_{JP[msu]}+d_{JP[msu]}+d_u\geq r_{msu}$，则按照工序的最早完工时间，查找工序 msu 的某一道工件前序工序 msu'，如果存在工序 msu' 满足 $MP[msu']\neq\varnothing$，且 $r_{MP[msu']}+d_{MP[msu']}=r_{msu'}$，则交换工序 msu' 与 $MP[msu']$；

(2) 如果 $JS[u]\neq\varnothing$，并且满足 $q_{JS[u]}>q_{MS[v]}-d_{MS[v]}$，则按照工序的最晚开工时间，查找工序 u 的某一道工件后序工序 u'，如果存在工序 u' 满足条件 $MS[u']\neq\varnothing$，且 $q_{u'}-d_{u'}=q_{MS[u']}$，则交换工序 u' 与 $MS[u']$。

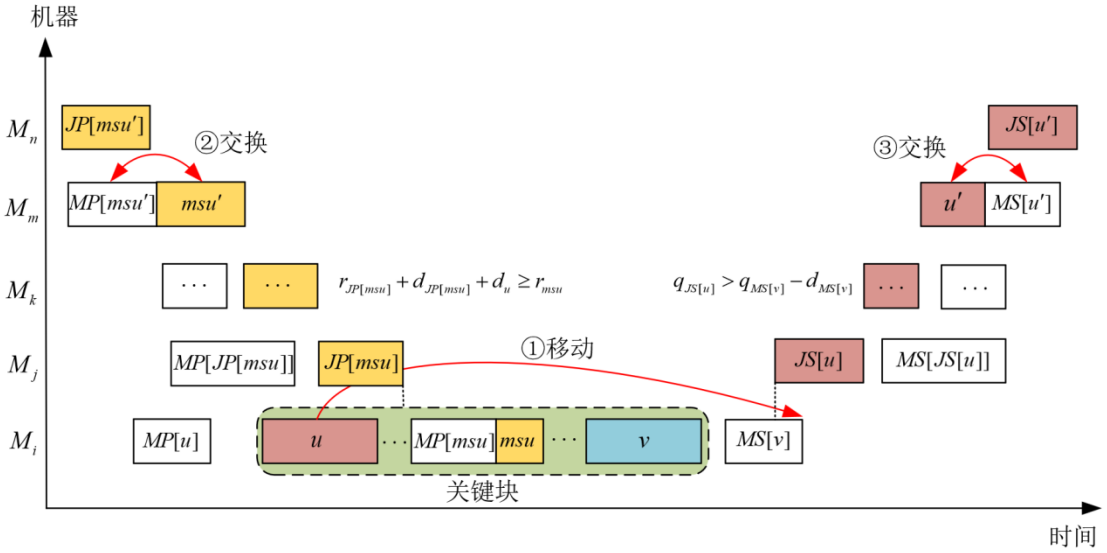


图 3-11 多工序精确联动情况 1 示意图

情况 2：块首工序 u 移至块尾工序 v 后，工序联动方式如图 3-12 所示，具体步骤如下：

(1)令 msu 为关键块中某块中工序，如果 $JP[msu]\neq\varnothing$，并且满足 $r_{JP[msu]}+d_{JP[msu]}+d_u\geq r_{msu}$，则按照工序的最早完工时间，查找工序 msu 的某一道工件前序工序 msu'，如果存在工序 msu' 满足 $MP[msu']\neq\varnothing$，且 $r_{MP[msu']}+d_{MP[msu']}=r_{msu'}$，则交换工序 msu' 与 $MP[msu']$；

(2) 如果 $JS[u]\neq\varnothing$，并且满足 $q_{JS[u]}>q_v-d_v$，则按照工序的最晚开工时间，查找工序 u 的某一道工件后序工序 u'，如果存在工序 u' 满足条件 $MS[u']\neq\varnothing$，

且 $q_{u'}-d_{u'}=q_{MS[u']}$，则交换工序 u' 与 $MS[u']$。

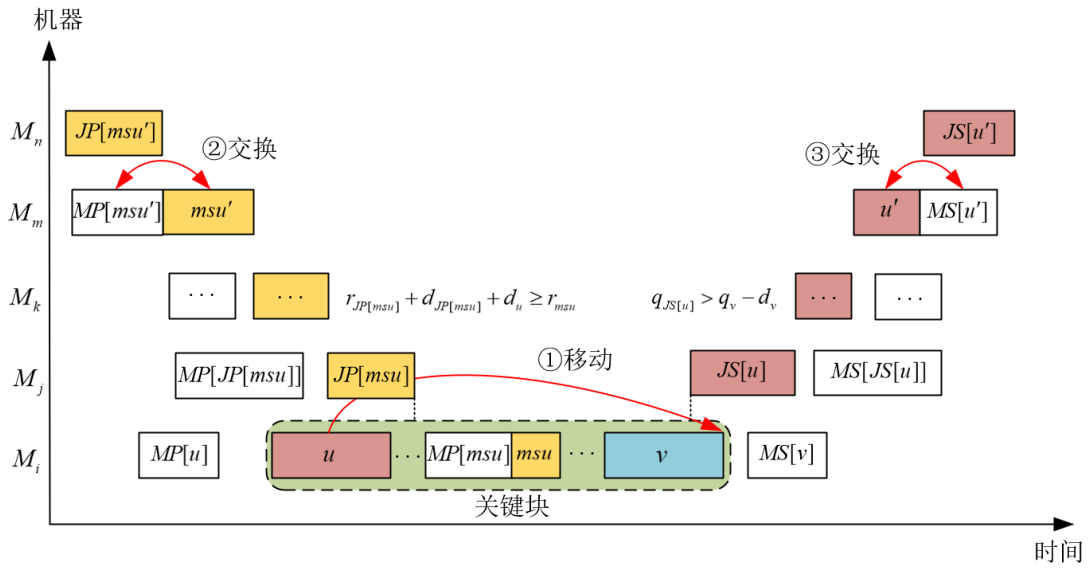


图 3-12 多工序精确联动情况 2 示意图

情况 3：块首工序 u 移至块中工序 v 后，工序联动方式如图 3-13 所示，具体步骤如下：

(1)令 msu 为关键块中某块中工序，如果 $JP[msu]\neq\varnothing$，并且满足 $r_{JP[msu]}+d_{JP[msu]}+d_u \geq r_{msu}$，则按照工序的最早完工时间，查找工序 msu 的某一道工件前序工序 msu'，如果存在工序 msu' 满足 $MP[msu']\neq\varnothing$，且 $r_{MP[msu']}+d_{MP[msu']}=r_{msu'}$，则交换工序 msu' 与 $MP[msu']$；

(2) 若 $MS[u]$ 与 $MS[v]$ 之间的任意工序 w 均满足 $r_{JP[w]}+d_{JP[w]}+d_u < r_w$，且 $JS[u]\neq\varnothing$，并且满足 $q_{JS[u]} > q_v - d_v$，则按照工序的最晚开工时间，查找工序 u 的某一道工件后序工序 u'，如果存在工序 u' 满足条件 $MS[u']\neq\varnothing$，且 $q_{u'}-d_{u'}=q_{MS[u']}$，则交换工序 u' 与 $MS[u']$。

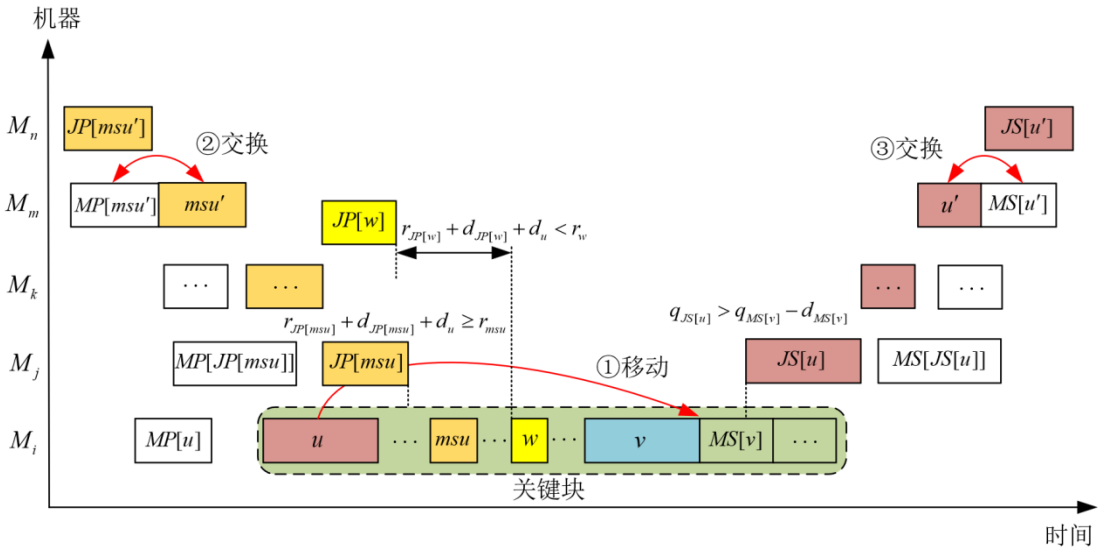


图 3-13 多工序精确联动情况 3 示意图

情况 4：块中工序 u 移至块尾工序 v 后，工序联动方式如图 3-14 所示，具体步骤如下：

(1)令 msu 为关键块中某块中工序，且位于工序 u 与 v 之间，如果 $JP[msu]\neq\varnothing$，并且满足 $r_{JP[msu]}+d_{JP[msu]}>r_u$，则按照工序的最早完工时间，查找工序 msu 的某一道工件前序工序 msu’，如果存在工序 msu’ 满足 $MP[msu']\neq\varnothing$，且 $r_{MP[msu']}+d_{MP[msu']}=r_{msu'}$，则交换工序 msu’ 与 $MP[msu']$；

(2) 如果 $JS[u]\neq\varnothing$，并且满足 $q_{JS[u]}>q_v-d_v$，则按照工序的最晚开工时间，查找工序 u 的某一道工件后序工序 u’，如果存在工序 u’ 满足条件 $MS[u']\neq\varnothing$，且 $q_{u'}-d_{u'}=q_{MS[u']}$，则交换工序 u’ 与 $MS[u']$。

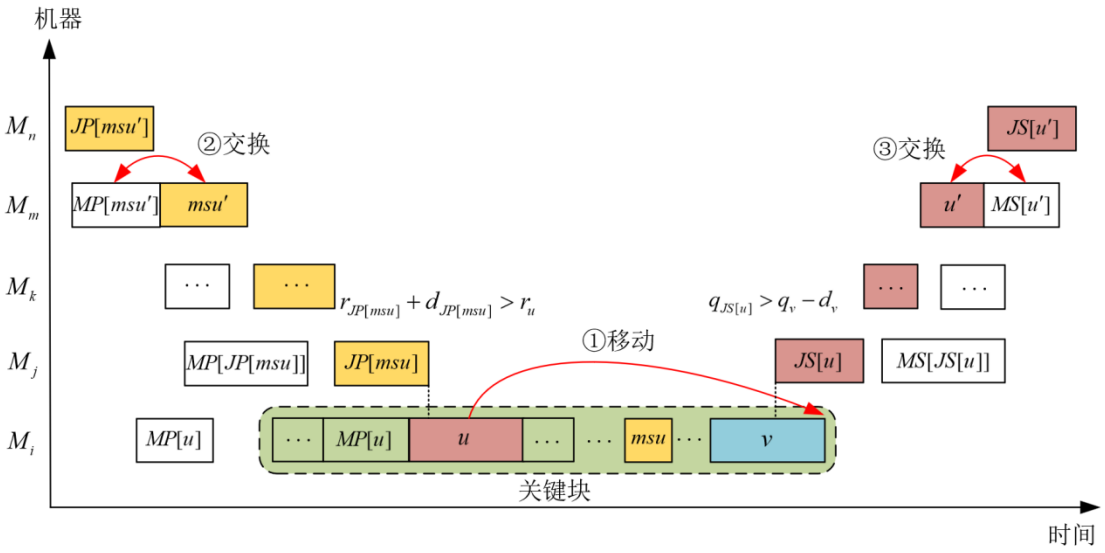


图 3-14 多工序精确联动情况 4 示意图

情况 5：块尾工序 v 移至块首工序 u 的机器前序工序 $MP[u]$ 前，工序联动方式如图 3-15 所示，具体步骤如下：

(1) 令 mpv 为关键块中的某块中工序，如果 $JS[mpv]\neq\varnothing$，并且满足 $q_{JS[mpv]}>q_{MS[v]}$，则按照工序的最晚开工时间，查找工序 mpv 的某一道工件后序工序 mpv’，如果在工序 mpv’ 满足条件 $MS[mpv']\neq\varnothing$，并且 $q_{mpv'}-d_{mpv'}=q_{MS[mpv']}$，则交换工序 mpv’ 和 $MS[mpv']$；

(2) 如果 $JP[v]\neq\varnothing$，并且满足 $r_{JP[v]}+d_{JP[v]}\geq r_{MP[u]}$，则按照工序的最早完工时间，查找工序 v 的某一道工件前序工序 v’，如果存在工序 v’ 满足条件 $MP[v']\neq\varnothing$，并且 $r_{MP[v']}+d_{MP[v']}=r_{v'}$，则交换工序 $MP[v']$ 与 v’。

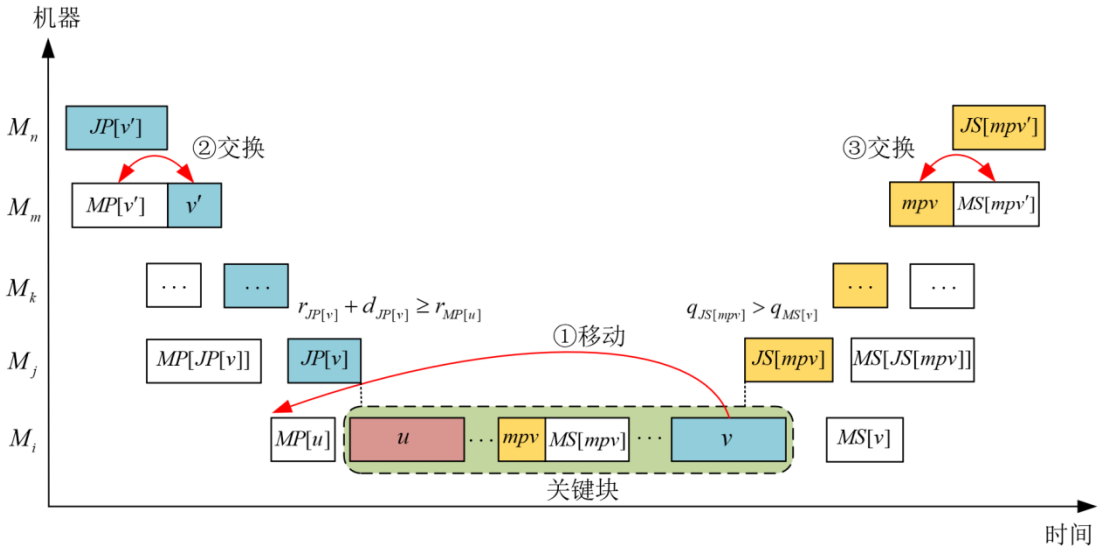


图 3-15 多工序精确联动情况 5 示意图

情况 6：块尾工序 v 移至块首工序 u 前，工序联动方式如图 3-16 所示，具体步骤如下：

(1) 令 mpv 为关键块中的某块中工序，如果 $JS[mpv]\neq\varnothing$，并且满足 $q_{JS[mpv]}>q_{MS[v]}$，则按照工序的最晚开工时间，查找工序 mpv 的某一道工件后序工序 mpv’，如果在工序 mpv’ 满足条件 $MS[mpv']\neq\varnothing$，并且 $q_{mpv'}-d_{mpv'}=q_{MS[mpv']}$，则交换工序 mpv’ 和 $MS[mpv']$；

(2) 如果 $JP[v]\neq\varnothing$，并且满足 $r_{JP[v]}+d_{JP[v]}\geq r_u$，则按照工序的最早完工时间，查找工序 v 的某一道工件前序工序 v’，如果存在工序 v’ 满足条件 $MP[v']\neq\varnothing$，并且 $r_{MP[v']}+d_{MP[v']}=r_{v'}$，则交换工序 $MP[v']$ 与 v’。

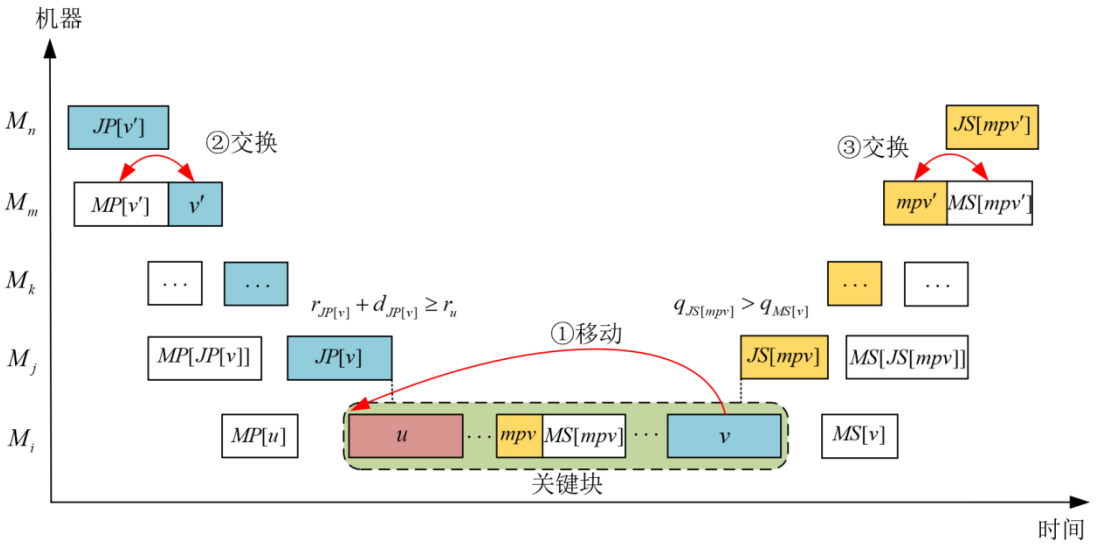


图 3-16 多工序精确联动情况 6 示意图

情况 7：块尾工序 v 移至块中工序 u 前，工序联动方式如图 3-17 所示，具体步骤如下：

(1) 令 mpv 为关键块中的某块中工序，如果 $JS[mpv]\neq\varnothing$，并且满足 $q_{JS[mpv]}>q_{MS[v]}$，则按照工序的最晚开工时间，查找工序 mpv 的某一道工件后序工序 mpv'，如果在工序 mpv' 满足条件 $MS[mpv']\neq\varnothing$，并且 $q_{mpv'}-d_{mpv'}=q_{MS[mpv']}$，则交换工序 mpv' 和 $MS[mpv']$；

(2) 若 $MP[u]$ 与 v 之间的任意工序 w 均满足 $q_w-d_w-d_v>q_{JS[w]}$，且 $JP[v]\neq\varnothing$，并且满足 $r_{JP[v]}+d_{JP[v]}\geq r_u$，则按照工序的最早开工时间，查找工序 v 的某一道工件前序工序 v'，如果存在工序 v' 满足条件 $MP[v']\neq\varnothing$，并且 $r_{MP[v']}+d_{MP[v']}=r_{v'}$，则交换工序 $MP[v']$ 与 v'。

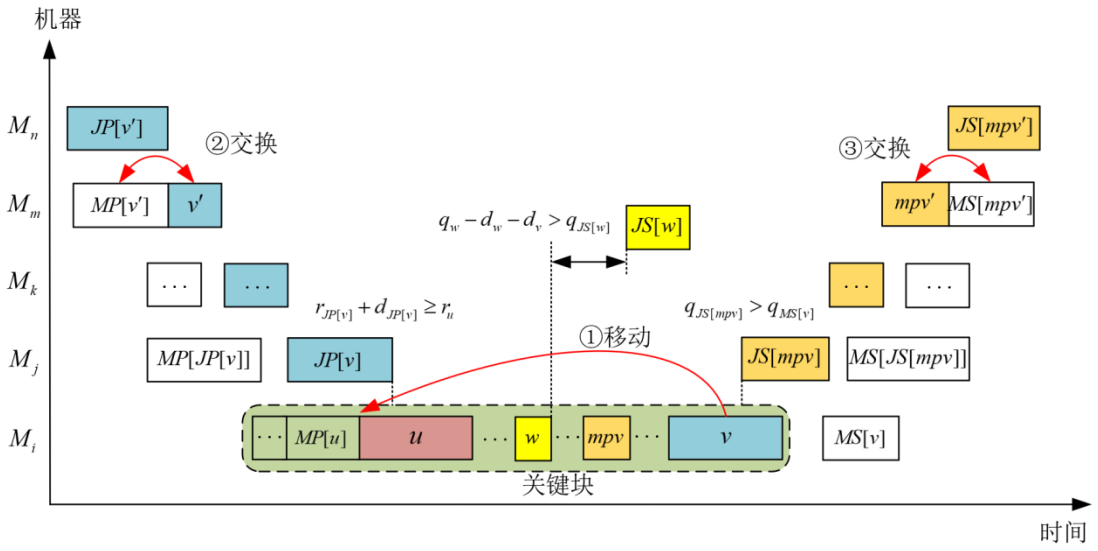


图 3-17 多工序精确联动情况 7 示意图

情况 8：块中工序 v 移至块首工序 u 前，工序联动方式如图 3-18 所示，具体步骤如下：

(1) 令 mpv 为关键块中的某块中工序，如果 $JS[mpv] \neq \varnothing$，并且满足 $q_{JS[mpv]} > q_{MS[v]}$，则按照工序的最晚开工时间，查找工序 mpv 的某一道工件后序工序 mpv'，如果在工序 mpv' 满足条件 $MS[mpv'] \neq \varnothing$，并且 $q_{mpv'} - d_{mpv'} = q_{MS[mpv']}$，则交换工序 mpv' 和 $MS[mpv']$；

(2) 如果 $JP[v] \neq \varnothing$，并且满足 $r_{JP[v]} + d_{JP[v]} \geq r_u$，则按照工序的最早完工时间，查找工序 v 的某一道工件前序工序 v'，如果存在工序 v' 满足条件 $MP[v'] \neq \varnothing$，并且 $r_{MP[v']} + d_{MP[v']} = r_{v'}$，则交换工序 $MP[v']$ 与 v'。

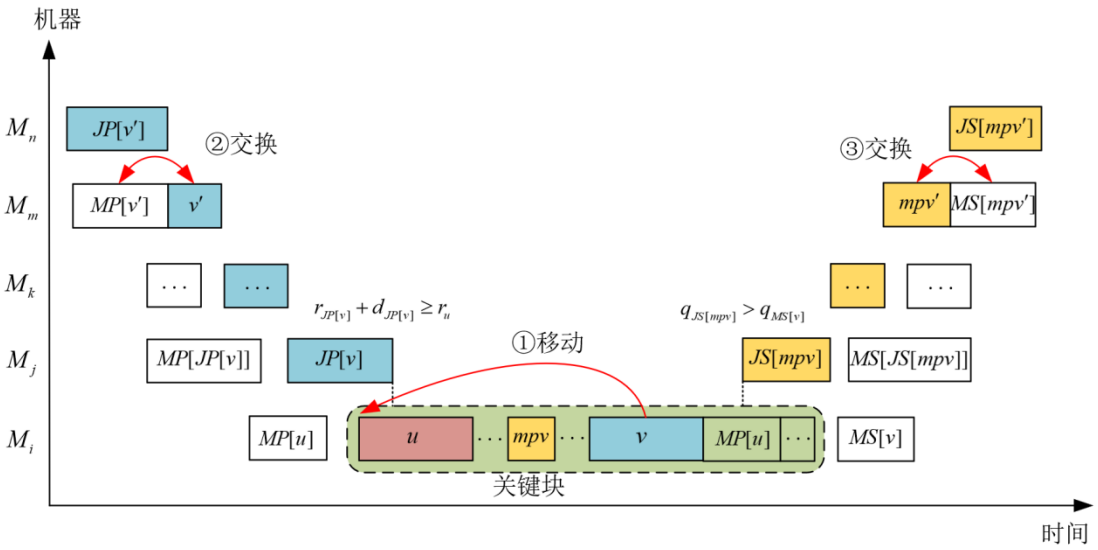


图 3-18 多工序精确联动情况 8 示意图

3.4 混合遗传禁忌搜索算法

3.4.1 初始化

初始化是 HGTSA 的第一步。它应该确保个体在全局解空间中均匀分布，且具有一定的质量。这减小了由于种群初始质量差而导致算法运行时间过长的可能性。本文采用了一种半启发式、半随机的初始化方法。这种方法有助于避免因完全随机初始化造成的时间浪费。首先，对每个工件的第一个工序进行加工，选择工序的顺序是随机的，然后再安排所有工件的第二个工序，重复上述过程，直到安排完全部工序。种群规模 N 是一个重要参数， N 的值影响算法的性能。具体值将通过后续参数实验确定。

3.4.2 编码

编码方法应简洁且方便后续迭代中的遗传操作。本文使用的方法是基于工件的编码，产生一个由数字组成的字符串，作为染色体。从左到右读取染色体串，工件编号的出现对应于该工件的工序，其出现的次数为该工件的工序数。染色体的总长度等于所有工件的工序数综合。这种染色体表示的一个关键好处是，染色体的任何排列都可以被解码为可行解。

以 3.2 节中给出的实例信息为例，首先，对所有工件的第一个工序进行调度，得到一个初始编码串：[1234]；接下来，使用基于系统时间的随机数对编码串进行打乱，得到一个新的编码串，例如：[4312]；重复上述步骤，直到生成完整的染色体串，例如：[4312124314324231]。每条染色体都代表一个完整的可行调度。不断创建新的染色体，直到群体规模达到预设值。图 3-19 显示了染色体串的示意图。

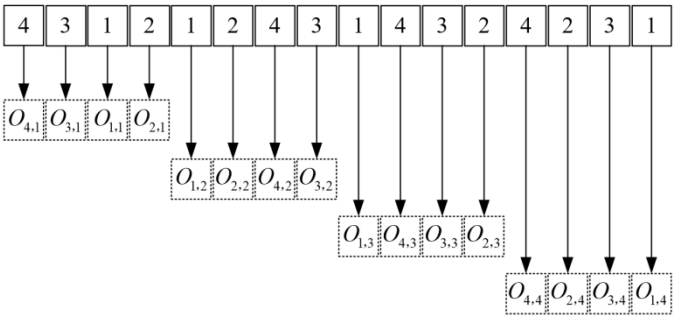


图 3-19 染色体串的示意图

3.4.3 解码

在此步骤中，初始化的染色体将被解码为调度计划表，并计算完工时间。调度的解码可以分为三类：无延迟解码、主动解码和半主动解码。根据相关研究证实，主动解码包含了最优调度，因此本文在解码方法中只使用主动调度，以减少搜索空间。具体步骤描述如下。

Step1：从左向右遍历染色体串，读取工件编号 J_i、工序数 $O_{J_i,j}$、相应的机器编号 $M_{J_i,j}$，相应的加工时间 $T_{J_i,j}$。

Step2：安排 $O_{J_i,j}$ 在机器 $M_{J_i,j}$ 上加工，并获得完工时间 $E_{t_{J_i,j}}$。

Step3：安排 $O_{J_i,j+1}$ 在机器 M_x 上处理，然后在 M_x 上查找出所有的空闲时间，即得到一个或若干个从 S_x 开始到 E_x 上结束的空闲时间间隔 $[S_x, E_x]$。由于同一工件的不同工序之间存在约束，$O_{J_i,j+1}$ 只有在 $O_{J_i,j}$ 完成时才能开始被调度，因此 $O_{J_i,j+1}$ 的最早开始时间 t_b 可以根据式(3-28)计算，且 t_b 需满足式(3-29)的约束。

$$t_b = \max\{Et_{J_i,j}, S_x\} \tag{3-28}$$

$$t_b + T_{J_i,j+1} \le E_x \tag{3-29}$$

Step4：根据式（3-29），如果$[S_x, E_x]$有足够的时间跨度来完成$O_{J_i,j+1}$的加工，则时间间隔$[S_x, E_x]$可用于$O_{J_i,j+1}$。否则，$O_{J_i,j+1}$将在M_x上的最后一个工序结束时分配，这意味着$O_{J_i,j+1}$的加工会从T_x开始。图 3-20 解释了此过程。

Step5：重复 Step3~Step4，直到每个操作都在相应的机器上处理完毕。

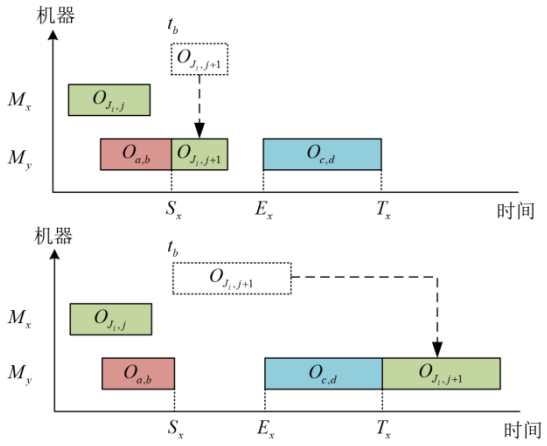


图 3-20 主动解码过程示意图

3.4.4 选择算子

选择算子的目的是过滤掉种群中质量较低的个体，为了更好地指导种群的更新，本文提出了一种类似蒙特卡洛方法的选择策略。首先，直接选择群体中的最优个体进入下一代，精英保留有助于避免在迭代过程中丢失最优解；接下来，计算并记录种群的中位数；最后，采用一种基于概率的选择函数来选择个体。表 3-4 列出了该方法一些相关参数及其解释。

表 3-4 选择策略相关参数及其解释

符号	描述	计算公式
T_i	种群中个体完工时间的中位数	-
T_j	个体的完工时间	-
t_k	衰减因子	$t_k = \frac{T_j}{T_i}$
ΔT	温度	$\Delta T = T_j - T_i$
p	蒙特卡洛函数	$p = \begin{cases} 1, \Delta T < 0 \\ \exp(-\frac{\Delta T}{t_k}), \Delta T \geq 0 \end{cases}$

首先，需要计算T_i。其次，随机选择一个人并计算T_j。然后可以得到ΔT。如果$\Delta T < 0$，即$p = 1$，则接受该个体。否则，计算 p 并生成随机数$q \in [0,1)$。如果$q \geq p$，个体将被接受进入下一代种群中；否则，该个体将被淘汰。蒙特卡洛函数在概率上可以接受劣解，能够增加种群多样性。

3.4.5 分类算子

本节描述了一种进化策略，该策略涉及对种群中具有不同质量的个体应用不同的搜索方法。对于完工时间较小的个体，进行更详细的局部搜索以实现局部最优。相反，对于较差的个体，重点转向了全局搜索，旨在探索尽可能多的解空间。然后，这些个体在后续步骤中与其他个体合作，最终追求全局最优。

经过多次试验，使用基于 N8T+2MT 的禁忌搜索应用于一些精英个体，而对其余个体使用调度部分重建搜索方法。具体分配比例将通过参数实验确定。这两种搜索方法将在第 3.4.6 节和第 3.4.7 节中详细描述。

3.4.6 基于 N8T+2MT 的禁忌搜索

禁忌搜索(Tabu search, TS)是本文所提混合算法的一个组成部分，因此这一步不会分配太多时间。禁忌搜索的迭代次数是提前预设的，当算法达到指定的迭代次数时，无论当前解改进到何种程度，它都将退出并继续下一步。

作为 TS 的短期记忆，禁忌表在防止算法迂回搜索方面起着重要作用。本文介绍了一种由主工序移动和副工序移动组成的禁忌列表结构。具体来说，在 N8T+2MT 邻域结构中，一个完整的多工序联动包括一个主工序移动(N8T)和两个副工序移动(2MT)。如果一次完整的联动没有导致完工时间的延长，则该移动操作涉及的六个工序，都应该在禁忌表中完整记录。在后续搜索过程中，只有当涉及的所有六个工序都与禁忌列表中记录的信息匹配时，才会被禁止。如果只有一部分工序与禁忌表中的记录重合，那么此次联动是被允许的。

禁忌表的长度是算法的一个重要参数。研究表明，禁忌表的长度应随着工件数量与机器数量之比的增加而适当增加。本研究中使用的方法由 Zhang 等人[40]提出，禁忌表的最小长度设置为 $L=10+\frac{n}{m}$。动态禁忌表的长度在 $[L,1.5L]$ 的范围内选择。

禁忌搜索算法的一个关键优点是它能够避免冗余搜索。当判定工序移动是无效时，需要将其存储在禁忌表中，以防止算法再次选择此移动。随着搜索的进行，染色体的特征会发生变化，以前被认为无效的移动可能会变得有效。为了防止禁忌表无限增长，其大小是根据问题规模设置。这意味着它智能地存储了有限数量的禁忌操作，当满足特定条件时，禁忌移动会减少。在本研究中，如果一次完整的移动不能提高完成时间，则将其相关工序存储在禁忌列表中。当存储在禁忌表中的操作数量达到预设水平时，禁忌表前半部分的禁忌信息将被释放。

在 TS 运行时，算法会优先处理禁忌表中不存在的工序移动操作。对于主工序移动，它根据 N8 邻域结构从关键块中随机选择两个工序。对于副工序移动，使用循环结构遍历调度计划表。该算法选择满足多工序联动的条件且具有较小完工时间的邻域解。如果有两个或多个具有相同完工时间的邻域解，则随机选择其中一个。然而，可能存在一种特殊情况，即所有候选解都包含在禁忌表中。在这种情况下，从所有可能的工序移动中随机选择一个移动，并立即重置禁忌表。

作为一个有 n 个作业和 m 台机器的 JSP 实例，基于 N8 邻域结构内的 6 种移动，当关键块上有 n 个操作时，交换次数最多，为 $(n-1)\times 6\times m$，计算复杂度此

时为$O(nm)$。对于 N8T+2MT，在 N8 邻域结构的基础上它通过寻找两对额外的工序交换操作，对两个交换的最大搜索次数为$2\times(m-1)$，此外，只需要计算一次，因为头工序在尾工序之后移动，尾工序在头工序之前移动，N8T+2MT 的最大搜索数量为$[(n-1)\times3\times m\times2\times(m-1)]+[(n-1)\times3\times m]+[3\times m\times2\times(m-1)]$，计算复杂度为$O(nm^2)$。

3.4.7 调度部分重建搜索

解决 JSP 的本质是在给定约束信息的情况下找到合理的工序安排。上一节的内容研究了通过改变关键路径上的工序排列来缩短完工时间。然而，还有一些工序并不在关键路径上，由于在初始化时的不合理安排，机器上会出现大量空闲时间，导致其他工序的延迟。因此，需要一种新的方法来优化这些工序。本节内容介绍了一种调度部分重建的搜索策略来解决这个问题。该策略详细描述如下。

首先，基于当前的调度方案，随机选择一个工件并将其从调度计划中移除；然后按照最早开完工时间重新排列其他工序；最后，将删除的工件按照一定规则重新添加回调度计划中，从而获得了一个新调度方案。具体的规则将分为 2 中情况描述，图 3-21 给出了相关添加规则的示意图。

情况 1：如果被添加的工序$O_{i,j}$是该工件的第一道工序$O_{1,j}$，$O_{1,j}$在机器$M_{O_{1,j}}$上加工。

(1)如果$M_{O_{1,j}}$上没有工序在 0 时刻开始加工，将$O_{1,j}$添加在 0 时刻。

(2) 如果$M_{O_{1,j}}$上有工序$O_{m,n}$在 0 时刻开始加工，将$O_{1,j}$添加在$O_{m,n}$后。

情况 2：如果被添加的工序$O_{i,j}$不是该工件的第一道工序，$O_{i,j}$在机器$M_{O_{i,j}}$上加工，该工序的工件前一工序为$O_{i-1,j}$。

(1) 按时间顺序搜索机器$M_{O_{i,j}}$上的现有工序。如果有一个工序$O_{m,n}$的开始时间大于$O_{i-1,j}$的结束时间，将$O_{i,j}$添加在$O_{m,n}$后。

(2) 按时间顺序搜索机器 $M_{O_{i,j}}$ 上的现有工序。如果没有任何工序的开始时间大于 $O_{i-1,j}$ 的结束时间，将 $O_{i,j}$ 添加在机器 $M_{O_{i,j}}$ 的末尾。

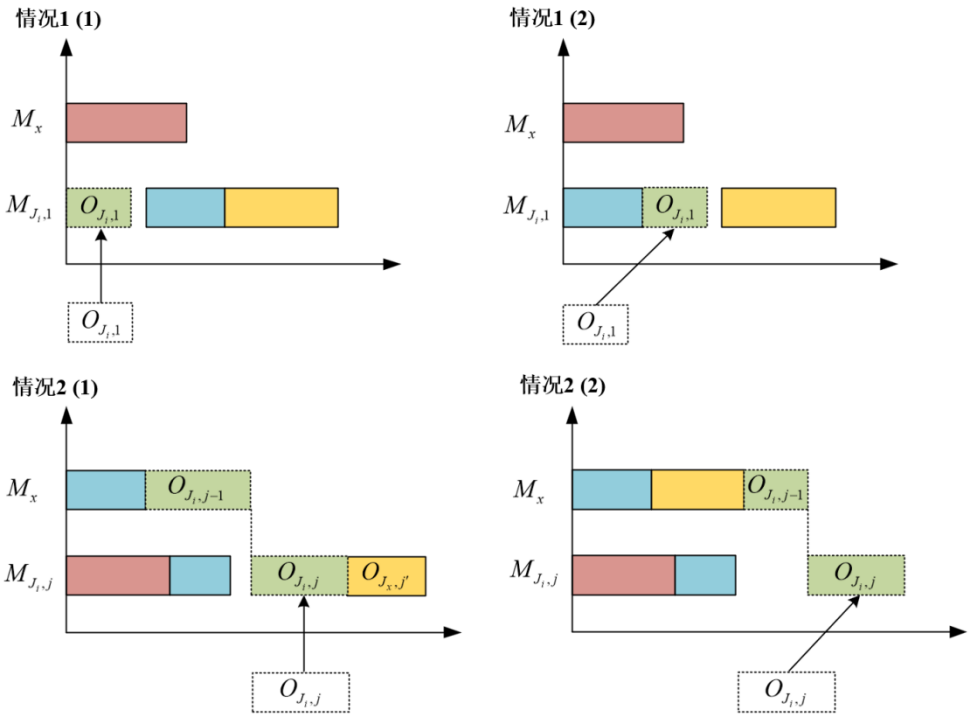
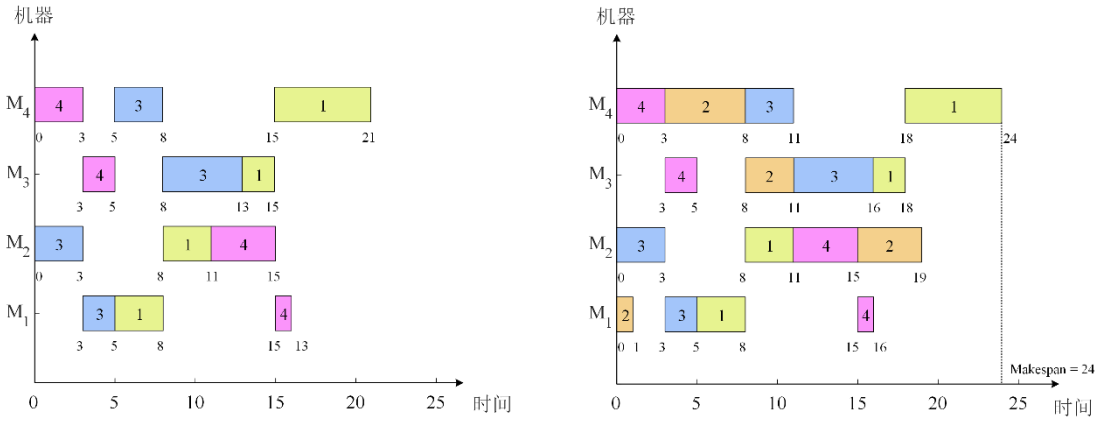


图 3-21 工件重新添加回调度不同情况示意图

需要注意的是，任何策略都有其局限性，这种策略的潜在副作用目前尚不清楚。此外，这种策略可能会对染色体造成严重干扰。因此，这种策略不会在同一染色体上广泛重复。相反，一次只会选择一个工件进行移除和重新添加。此举的目的是为了防止由于过度搜索而产生具有类似缺点的后代，这可能会导致算法陷入局部最优。

该策略的有效性将通过一个简单的实例进行验证。以表 3-1 中给出的实例为例，图 3-1 说明了一种可行的调度解决方案。此时，以工件 2 为例，随机选择工件 2 移除。剩余的工件按照最早开完工时间重新排列，再将作业 2 添加回调度中，从而产生新的调度，如图 3-22 所示。通过比较图 3-1 和图 3-22，原始计划的完成时间为 28，而使用此策略优化的计划的新完成时间为 27。这证明了这一战略的有效性。



(1) 工件2被移除，剩余工件按照最早开完工时间重新排列

(2) 工件2被重新添加回调度中，产生新调度方案

图 3-22 调度部分重建策略的实例演示

3.4.8 交叉算子

交叉是一种全局搜索方法，通过在两条染色体之间交换部分片段生成新个体。两个选定的父代个体将根据概率决定是否执行交叉算子。Zhang 等[83]人介绍了一种创新的交叉方法，称为优先运算（POX）交叉算子。这种方法巧妙地继承了亲本特征，同时减轻了潜在的基因损伤，稳定的产生新解。Hu 等[84]人通过采用随机选择交叉工件集的做法改进了 POX 交叉，这有助于防止后代继承类似的缺点。在传统方法中，通过交叉产生的后代总是被接受的，即使它们的质量会比父代个体差。这可能会导致最优解在迭代中丢失，本研究在此基础上进行了一些改进。具体来说，在涉及交叉算子的两个父代个体和生成的两个子代个体中，选择并保留了两个具有较小完工时间的个体。这种方法既为下一代保留最好的染色体，同时保持种群中个体之间的多样性。

3.4.9 变异算子

变异算子是遗传算法中常用的一种小规模局部搜索方法。被选中的个体将根据一定的概率进行变异。典型的变异方法包括交换法和插入法等，这些方法通过改变染色体内的基因序列来产生新个体。然而，需要注意的是，同一调度方案和染色体的编码序列并不是一一对应的，也就是说相同的调度方案可以用不同的染色体串来表示。这意味着当染色体的编码序列发生变化时，调度解决方案可能保持不变，这是常规变异方法效率低的原因之一。另一方面，由于选

择变异节点的随机性，交换法可能会影响多台机器上的多个工序，对染色体造成重大破坏，这是不理想的。使用插入法时，可能会在机器上引入大量空闲时间，从而增加总完工时间。

在本文中，使用了一种基于单台机器相近工序的变异算子。具体操作如下。首先在一台机器上随机选择一个工序，并将其与相邻的工序交换，无论是在该工序之前还是之后。这样生成新的解决方案时，染色体变化的程度相对较小，避免了个体质量的大幅滑落。以表 3-1 中的调度信息为例，图 3-1 说明了一个可行的调度解决方案。在这种情况下，选择机器 M_3 上的随机作业 J_3，并将其与 J_2 交换。这种变异产生了一个新解，其完工时间为 24，如图 3-23 所示。

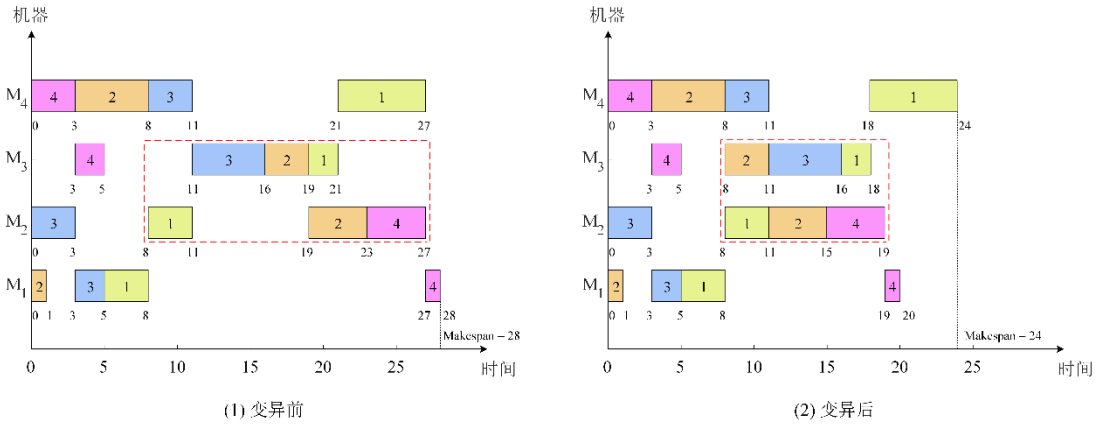


图 3-23 变异算子的实例演示

3.4.10 算法架构

HGTSA 的算法运行顺序描述如下。首先，初始化种群，并对每个个体进行编码。经过解码后，可以获得种群中每个个体的完工时间和调度计划表。此时，算法评估是否满足终止条件，如果是肯定的，则算法终止；否则，它将进入迭代阶段。迭代阶段首先运行选择算子，通过一些方法过滤较差的个体，保留较优个体，形成新的群体。随后，将种群分为两部分，一部分执行基于 N8T+2MT 邻域结构的禁忌搜索算法，另一部分采用调度部分重建策略进行重构。完成后，这两个部分重新合并为一个种群，根据相关概率继续执行交叉和变异算子。此时，迭代阶段结束，并评估是否满足终止条件，如此重复直至运行结束。图 3-

24 给出了算法流程图。

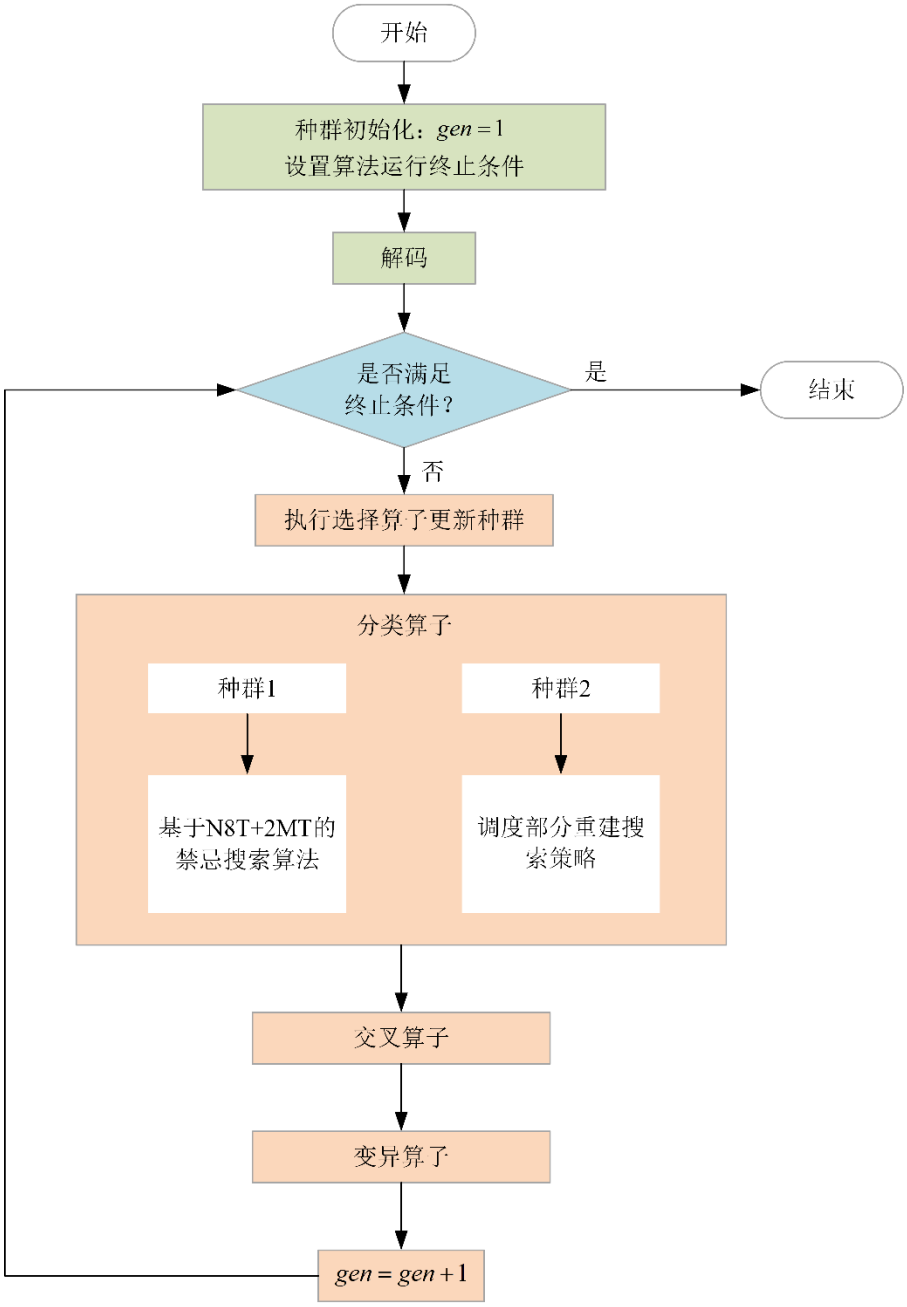


图 3-24 HGTSA 运行流程图

3.5 实验与仿真

所有测试程序均使用 C++编程语言实现，并在具有 Windows 11 操作系统的计算机上使用 Microsoft Visual Studio 2022 软件运行（CPU 基准速度为 2.3GHz，RAM 为 16GB）。

本文使用著名基准实例用于测试 HGTSA。这些车间调度实例如下：Fisher 和 Thompson[85]表示为 FT06、FT10 和 FT20 的 3 个实例，Lawrence[86]表示为 LA01-LA40 的 40 个实例，Adams 等人[83]表示为 ABZ5-ABZ9 的 5 个实例，Applegate 等人[88]示为 ORB01-ORB10 的 10 个实例，Storer 等人[89]表示为 SWV01-SWV10 的 10 个实例，Taillard[90]表示为 TA01-TA50 的 50 个实例，Demirkol 等人[91]表示为 DMU01-DMU40 的 40 个示例。这些基准实例包含小型、中型和大型 JSP，很难解决。这些实例可以在此网站上下载：(https://www.eii.uva.es/elena/Elena%20Perez%20Vazquez_archivos/files_optimaJSSP/jobshop1.txt)，为了消除随机性的影响，所有实例都在十次运行中独立测试。

3.5.1 参数设计

算法中一些关键参数的值对其性能有重大影响，仅根据经验为这些参数赋值可能是不准确的。因此，使用 DOE 方法确定参数的具体取值[92]。目标是选择一种合适的参数组合，以产生具有低可变性的稳定实验结果。

需要实验确定的主要参数如下：N 表示种群大小。r 代表精英人口分配比例。$Iter_{\max}$ 代表禁忌搜索迭代次数。C_p 代表交叉概率。M_p 代表突变概率。这些因素之间没有相互作用。经过初步筛选实验，确定了参数实验的范围。总共有 5 个参数，为了保证实验精度，选择了 4 个水平。每个参数的级别如表 3-5 所示。

表 3-5 参数及其水平

参数	水平			
	1	2	3	4
N	50	100	150	200
r	25%	30%	35%	40%
$Iter_{\max}$	50	75	100	125
C_p	60%	65%	70%	75%
M_p	5%	10%	15%	20%

本实验中使用的基准实例是 LA40。考虑到程序执行的随机性，每个解决方案使用相应的程序独立运行 5 次。CPU 运行时间设置为 10 秒作为终止条件。M_{av} 是实验结果的平均值。表 3-6 给出了正交阵列 $L_{16}(4^5)$。使用 Minitab 软件对实验结果进行分析，平均值的响应表如表 3-7 所示。图 3-25 给出了 M_{av} 和 M_{av} 的信噪比的结果，其中信噪比定义为 $-10\times\log10(\frac{\sum_{i=1}^{n}M_{av}{}^2}{n})$。Mav 代表算法的平均性能，较低的值表示在该参数设置下的平均性能良好。信噪比反映了性能稳定性，较高的信噪比表明在该参数设置下相对稳定，对随机变化的敏感性较低。通过实验得出的参数最佳设置为：$N=200$，$r=30\%$，$Iter_{\max}=50$，$C_p=75\%$，$M_p=5\%$。可以发现，$Iter_{\max}$ 是影响算法性能的最关键参数，这也符合预期。

表 3-6 参数及其水平

编号	水平					M_{av}
	N	r	$Iter_{\max}$	C_p	M_p	
1	1	1	1	1	1	1716.0
2	1	2	2	2	2	1749.2
3	1	3	3	3	3	1792.2
4	1	4	4	4	4	1733.2
5	2	1	2	3	4	1769.0
6	2	2	1	4	3	1695.4
7	2	3	4	1	2	1811.4
8	2	4	3	2	1	1767.0
9	3	1	3	4	2	1774.4
10	3	2	4	3	1	1765.8
11	3	3	1	2	4	1730.2
12	3	4	2	1	3	1785.8
13	4	1	4	2	3	1772.2
14	4	2	3	1	4	1730.8
15	4	3	2	4	1	1703.2
16	4	4	1	3	2	1715.6

表 3-7 均值响应表

水平	因子				
	N	r	$Iter_{\max}$	C_p	M_p
1	1748	1758	1714	1761	1738
2	1761	1735	1752	1755	1763
3	1764	1759	1766	1761	1761
4	1730	1750	1771	1727	1741
Delta	34	24	56	34	25
Ranking	3	5	1	2	4

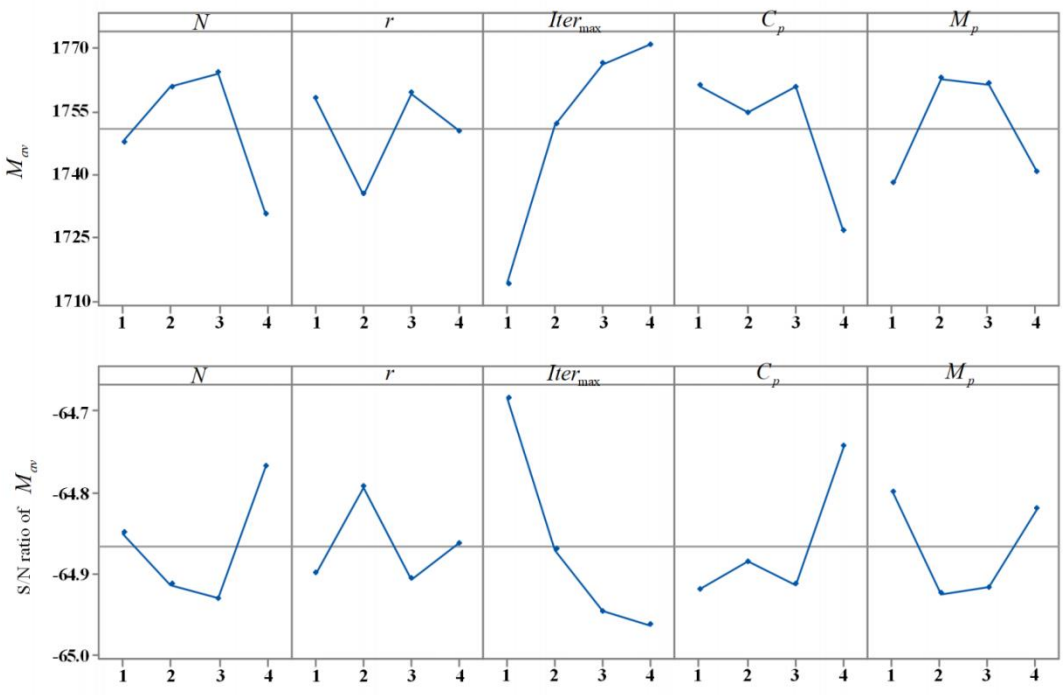


图 3-25 M_{av} 与 M_{av} 的信噪比

3.5.2 性能测试

本节内容使用著名基准案例评估 HGTSA 的整体性能。使用了以下指标：UB 为目前已知的最佳 makespan，LB 为理论下界。$Best$、M_{av}、T_{Best} 分别表示所求最优完工时间、平均完工时间和获得最优完工时间的计算时间（单位：秒）。该算法的终止标准包括达到 LB 或连续迭代 500 秒不产生更优解。

首先将 HGTSA 与经典的元启发式算法对比，在 FT、LA 和 ABZ 实例中选择了 23 个实例进行性能测试。表 3-8 提供了详细的结果。选择了 3 种与 HGTSA 相关的经典启发式算法：遗传算法(GA)、禁忌搜索(TS)、模拟退火(SA)。可以看出，HGTSA 能够找到全部实例的最优解，且在大多数实例计算上时间非常短，多数小于 10 秒，最长也只需要 39 秒。GA、TS 和 SA 在这些实例的计算中表现不佳，其中大多数都未求得最优解。这一结果也是意料之中的。HGTSA 表现出了良好的性能，这归功于其优化的算法框架和使用 N8T+2MT 邻域结构作为其主要搜索方法。HGTSA 的整体性能是优于经典启发式算法。

表 3-8 与经典元启发式算法的实验结果对比

Prob-lem	Size	UB(LB)	HGTSA			GA	TS	SA
			Best	M_{av}	T_{Best}	*Best*	*Best*	*Best*
FT06	6×6	55	**55**	55	0.13	69	65	71
FT10	10×10	930	**930**	930	5.34	1126	1095	1168
FT20	20×5	1165	**1165**	1165	3.65	13.4	1288	1326
LA01	10×5	666	**666**	666	0.16	694	698	705
LA02	10×5	655	**655**	655	1.64	696	682	695
LA03	10×5	597	**597**	597	3.25	635	642	655
LA04	10×5	590	**590**	590	1.93	652	641	664
LA05	10×5	593	**593**	593	2.51	683	665	702
LA06	15×5	926	**926**	926	0.99	1135	1172	1165
LA07	15×5	890	**890**	890	0.88	1112	1136	1125
LA08	15×5	863	**863**	863	0.82	1098	1142	1126
LA09	15×5	951	**951**	951	0.53	1302	1256	1272
LA10	15×5	958	**958**	958	1.21	1298	1309	1349
LA11	20×5	1222	**1222**	1222	3.45	1525	1563	1597
LA12	20×5	1039	**1039**	1039	2.63	1263	1286	1326
LA13	20×5	1150	**1150**	1150	2.96	1426	1463	1445
LA14	20×5	1292	**1292**	1292	4.54	1653	1673	1682
LA15	20×5	1207	**1207**	1207	3.79	1598	1632	1658
LA16	10×10	945	**945**	949.3	39.46	1233	1259	1308
LA17	10×10	784	**784**	784	3.86	992	963	954
LA18	10×10	848	**848**	848	12.37	1132	1140	1085
LA19	10×10	842	**842**	842	26.32	1098	1122	1156
LA20	10×10	902	**902**	902	9.83	1230	1287	1302

在 FT、LA 和 ABZ 实例中选择了 48 个实例进行性能测试。表 3-9 提供了详细的结果。将 HGTSA 与该领域最近的高级研究论文中的其他流行算法进行了比较。这些算法包括：改进的遗传算法(Improved Genetic Algorithm, IGA)[84]、混合粒子群优化算法(Hybrid Particle Swarm Optimization, HPSO)[93]、混合遗传算法(Hybrid Genetic Algorithm, HGA)[94]、带路径重连的禁忌搜索算法(Tabu Search with Path Relinking, TS/PR)[95]、基于深度强化学习的算法(Deep Reinforcement Learning, DRL)[96]、带偏置随机密钥遗传算法的扩展 Akers 图形方法(Biased Random-Key Genetic Algorithm, BRKGA)[30]。

从表 3-9 中可以看出，HGTSA 可以求得 3 个 FT 和 40 个 LA 实例的最优值。

而 HPSO 在 2 个实例上无法求得最优，TS/PR 和 BRKGA 在 LA29 实例上表现不佳，HGA 在 12 个实例上难以达到最优，这可能是由于算法的搜索效率较低。对于 DRL，其受到深度强化学习框架的限制，缺乏类似于邻域结构的有效搜索方法，导致 JSP 上的结果不佳。深度强化学习在解决动态问题时通常更稳健，在处理静态问题时可能不如启发式算法有效。HGTSA 得益于其多工序联动邻域结构，与其他改进算法相比，综合性能更好，在某些实例的表现超越了 TS/PR 和 BRKGA。在 ABZ 实例的测试中，HGTSA 在 3 个实例上没有达到 UB。尽管如此，它仍然优于基于 IGA 和 DRL。

表 3-9 FT/LA/ABZ 实例的实验结果对比

	Size	UB(LB)	HGTSA			IGA	HPSO	HGA	TS/PR	DRL	BRKGA
			Best	M_{av}	T_{Best}	*Best*	*Best*	*Best*	*Best*	*Best*	*Best*
FT06	6×6	55	**55**	55	0.13	55	55	-	55	60	55
FT10	10×10	930	**930**	930	5.34	930	930	-	930	1102	930
FT20	20×5	1165	**1165**	1165	3.65	1165	1165	-	1165	1332	1165
LA01	10×5	666	**666**	666	0.16	666	666	666	666	670	666
LA02	10×5	655	**655**	655	1.64	655	655	655	655	756	655
LA03	10×5	597	**597**	597	3.25	597	597	597	597	807	597
LA04	10×5	590	**590**	590	1.93	590	590	590	590	716	590
LA05	10×5	593	**593**	593	2.51	593	593	593	593	593	593
LA06	15×5	926	**926**	926	0.99	926	926	926	926	932	926
LA07	15×5	890	**890**	890	0.88	890	890	890	890	1012	890
LA08	15×5	863	**863**	863	0.82	863	863	863	863	962	863
LA09	15×5	951	**951**	951	0.53	951	951	951	951	985	951
LA10	15×5	958	**958**	958	1.21	958	958	958	958	958	958
LA11	20×5	1222	**1222**	1222	3.45	1222	1222	1222	1222	1228	1222
LA12	20×5	1039	**1039**	1039	2.63	1039	1039	1039	1039	1050	1039
LA13	20×5	1150	**1150**	1150	2.96	1150	1150	1150	1150	1150	1150
LA14	20×5	1292	**1292**	1292	4.54	1292	1292	1292	1292	1292	1292
LA15	20×5	1207	**1207**	1207	3.79	1207	1207	1207	1207	1477	1207
LA16	10×10	945	**945**	949.3	39.46	945	945	945	945	1051	945
LA17	10×10	784	**784**	784	3.86	784	784	784	784	857	784
LA18	10×10	848	**848**	848	12.37	848	848	848	848	947	848
LA19	10×10	842	**842**	842	26.32	842	842	844	842	979	842
LA20	10×10	902	**902**	902	9.83	902	902	907	902	990	902
LA21	15×10	1046	**1046**	1052.6	93.64	1046	1046	1046	1046	1079	1046
LA22	15×10	927	**927**	927	34.86	-	927	935	927	1057	927
LA23	15×10	1032	**1032**	1032	13.96	1032	1032	1032	1032	1145	1032
LA24	15×10	935	**935**	937.4	6.68	-	935	953	935	1074	935
LA25	15×10	977	**977**	981.3	56.41	-	977	981	977	1099	977
LA26	20×10	1218	**1218**	1218	32.97	-	1218	1218	1218	1383	1218
LA27	20×10	1235	**1235**	1241.2	164.50	-	1235	1236	1235	1515	1235
LA28	20×10	1216	**1216**	1216	23.46	-	1216	1216	1216	1451	1216

LA29	20×10	1152	**1152**	1160.5	126.36	-	1163	1160	1153	1389	1153
LA30	20×10	1355	**1355**	1355	47.81	-	1355	1355	1355	1477	1355
LA31	30×10	1784	**1784**	1784	0.91	1784	1784	1784	1784	1894	1784
LA32	30×10	1850	**1850**	1850	2.68	1850	1850	1850	1850	1902	1850
LA33	30×10	1719	**1719**	1719	3.34	1719	1719	1719	1719	1826	1719
LA34	30×10	1721	**1721**	1721	1.26	1721	1721	1721	1721	1854	1721
LA35	30×10	1888	**1888**	1888	2.49	1888	1888	1888	1888	2026	1888
LA36	15×15	1268	**1268**	1268	76.49	-	1268	1287	1268	1426	1268
LA37	15×15	1397	**1397**	1397	46.38	-	1397	1407	1397	1649	1397
LA38	15×15	1196	**1196**	1201.4	161.89	-	1196	1196	1196	1398	1196
LA39	15×15	1233	**1233**	1235.9	183.46	-	1233	1233	1233	1469	1233
LA40	15×15	1222	**1222**	1236.3	364.91	-	1224	1229	1222	1400	1222
ABZ5	10×10	1234	1235	1250.6	109.6	1250	-	-	-	1360	-
ABZ6	10×10	943	**943**	949.7	50.8	943	-	-	-	980	-
ABZ7	20×15	656	658	672.1	380.9	670	-	-	657	767	656
ABZ8	20×15	665(645)	667	669.3	243.1	682	-	-	667	805	667
ABZ9	20×15	678(661)	**678**	681.2	90.6	695	-	-	678	874	678

从 ORB 实例中选择了 10 个问题，这些问题相较于 LA、ABZ 实例更加复杂，可能需要更多时间。选择的对比算法包括：具有移动瓶颈的引导式局部搜索算法(SBGLS)[35]，具有特定邻域的禁忌搜索技术(TSAB)[37]。表 3-10 提供了详细的结果。其中，HGTSA 在 ORB08 实例中没有达到最优，而 SBGLS 在 7 个实例中未能求得最优解，TSAB 在 4 个实例表现不佳。从求解结果的角度来看，HGTSA 具有一定优势。*Total time* 表示求得这些解所需的最小 CPU 时间的总和，HGTSA 的运行时间仅为 266.33 秒，优于 SBGLS 的 4871 秒和 TSAB 的 801.7 秒。

表 3-10 ORB 实例的实验结果对比

Problem	Size	UB	HGTSA		SBGLS		TSAB	
			Best	T_{Best}	*Best*	T_{Best}	*Best*	T_{Best}
ORB01	10×10	1059	**1059**	13.20	1059	548	1059	17.3
ORB02	10×10	888	**888**	5.34	890	376	890	88.4
ORB03	10×10	1005	**1005**	3.65	1005	356	1005	16.2
ORB04	10×10	1005	**1010**	98.6	1011	427	1013	285.6
ORB05	10×10	887	**887**	6.43	889	389	889	15.2
ORB06	10×10	1010	**1010**	4.19	1013	472	1013	124.8
ORB07	10×10	397	**397**	12.43	397	642	397	69.2
ORB08	10×10	899	901	103.5	913	568	899	97.6
ORB09	10×10	934	**934**	7.96	941	426	934	73.2
ORB10	10×10	944	**944**	11.03	946	667	944	14.2
Total time (s)			266.33		4871		801.7	

从 SWV 实例中选择了 10 个问题，由于其复杂的数据和庞大的规模，SWV 实例中仍有部分问题尚未求得最优解。选择的对比算法包括：全局均衡搜索和禁忌搜索相结合的混合算法(GESTS)[21]，禁忌搜索和模拟退火相结合的杂交算法(TSSA)[97]。表 3-11 提供了详细的结果。从结果来看，HGTSA 在 9 个问题上取得了最好的结果，仅在 SWV04 实例上表现不佳，而 GESTS 在 6 个实例的结果上不如 HGTSA，TSSA 在 5 个问题上结果较差，因此 HGTSA 相比之下能够得出更好的结果。*MSE* 表示平均稳定性误差，用于衡量算法的稳定性。它可以通过式(3-30)计算：

$$MSE = \frac{\sum_{i=1}^{n} Best}{\sum_{i=1}^{n} M_{av}} \tag{3-30}$$

根据式(3-30)，MSE 的范围值为(0,1]。MSE 值越大，表示算法获得的结果与最佳结果之间的差距越小。根据相关指标的计算值，HGTSA 具有最大的 MSE 值，为 0.9958，GESTS 为 0.9930，TSSA 的 MSE 值最小，为 0.9869，相比之下 HGTSA 稳定性表现最好。

表 3-11 SWV 实例的实验结果对比

Prob-lem	Size	UB(LB)	HGTSA		GESTS		TSSA	
			Best	M_{av}	*Best*	M_{av}	*Best*	M_{av}
SWV01	20×10	1407	**1407**	1412.3	1412	1417.5	1412	1423.7
SWV02	20×10	1475	**1475**	1477.5	1475	1477.2	1475	1480.3
SWV03	20×10	1398(1369)	**1398**	1402.3	1398	1398.4	1398	1417.5
SWV04	20×10	1470(1450)	1474	1475.6	1471	1476.3	1470	1483.7
SWV05	20×10	1424	**1425**	1427.1	1426	1436.5	1425	1443.8
SWV06	20×15	1672(1591)	**1675**	1674.9	1677	1678.4	1679	1700.1
SWV07	20×15	1594(1446)	**1595**	1610.3	1595	1622.6	1603	1631.3
SWV08	20×15	1752(1640)	**1752**	1758.4	1766	1785.7	1756	1786.9
SWV09	20×15	1656(1604)	**1659**	1664	1660	1681.8	1661	1689.2
SWV10	20×15	1743(1631)	**1743**	1755	1760	1775.5	1754	1783.7
MSE			0.9958		0.9930		0.9869	

从 TA 实例中选择了 50 个问题，这是过去 20 年来最困难的 JSP 实例的一部分。表 3-12 显示了使用 HGTSA 求解 TA 实例集的结果，并与一些其他算法（包

括 BRKGA、SBGLS 和 TSAB）进行了比较。可以看出，HGTSA 能够为 50 个实例中的 37 个找到当前已知最优的解决方案，并对 4 个实例求出了新的上限：TA32-1784、TA43-1846、TA44-1979、TA50-1923。

表 3-12 TA 实例的实验结果对比

Prob-lem	Size	UB(LB)	HGTSA		BRKGA		SBGLS-N6	TSAB-N5
			Best	M_{av}	*Best*	M_{av}	*Best*	*Best*
TA01	15×15	1231(1231)	1231	1231	1231	1231	-	1231
TA02	15×15	1244(1244)	1244	1244	1244	1244	1244	1244
TA03	15×15	1218(1218)	1218	1218	1218	1218	1222	1218
TA04	15×15	1175(1175)	1175	1175	1175	1175	-	1181
TA05	15×15	1224(1224)	1224	1224	1224	1224.9	1233	1233
TA06	15×15	1238(1238)	1238	1238	1238	1238.9	-	1241
TA07	15×15	1227(1227)	1228	1228	1228	1228	-	1228
TA08	15×15	1217(1217)	1217	1217	1217	1217	1220	1217
TA09	15×15	1274(1274)	1274	1275.6	1274	1277	1282	1274
TA10	15×15	1241(1241)	1241	1241	1241	1241	1259	1241
TA11	20×15	1357(1323)	1357	1363.5	1357	1360	-	1364
TA12	20×15	1367(1351)	1369	1371.9	1367	1372.6	1377	1367
TA13	20×15	1342(1282)	1344	1347.2	1344	1347.3	-	1350
TA14	20×15	1345(1345)	1345	1345	1345	1345	1345	1345
TA15	20×15	1339(1304)	1339	1342.6	1339	1348.9	-	1353
TA16	20×15	1360(1304)	1360	1360	1360	1362.1	-	1369
TA17	20×15	1462(1462)	1463	1468.6	1462	1470.5	-	1478
TA18	20×15	1396(1369)	1396	1396	1396	1400.9	1413	1396
TA19	20×15	1332(1304)	1332	1332	1332	1333.2	1352	1341
TA20	20×15	1348(1318)	1348	1353.4	1348	1350.4	1362	1359
TA21	20×20	1642(1573)	1642	1652.3	1642	1647	-	1659
TA22	20×20	1600(1542)	1600	1660	1600	1600	-	1603
TA23	20×20	1557(1474)	1557	1559.7	1557	1562.6	-	1558
TA24	20×20	1644(1606)	1644	1649.4	1646	1650.6	-	1659
TA25	20×20	1595(1518)	1595	1595	1595	1602	-	1615
TA26	20×20	1643(1558)	1647	1656.3	1643	1652.3	1657	1659
TA27	20×20	1680(1617)	1680	1686.7	1680	1685.6	-	1689
TA28	20×20	1603(1591)	1603	1609.2	1603	1611.7	-	1615
TA29	20×20	1625(1525)	1625	1625	1625	1627.4	1629	1629
TA30	20×20	1584(1485)	1584	1592.4	1584	1588.5	-	1604
TA31	30×15	1764(1764)	1764	1765.1	1764	1764.4	1766	1766

TA32	30×15	1785(1774)	**1784**	1789.4	1785	1794.1	1841	1803
TA33	30×15	1791(1778)	1791	1795.4	1791	1793.7	1832	1796
TA34	30×15	1829(1828)	1831	1836.5	1829	1832.1	-	1832
TA35	30×15	2007(2007)	2007	2007	2007	2007	-	2007
TA36	30×15	1819(1819)	1819	1823.1	1819	1822.9	-	1823
TA37	30×15	1771(1771)	1774	1783.4	1771	1777.8	1815	1784
TA38	30×15	1673(1673)	1675	1678.9	1673	1676.7	1700	1681
TA39	30×15	1795(1795)	1795	1801.3	1795	1801.6	1811	1798
TA40	30×15	1669(1631)	1670	1675.1	1669	1678.1	1720	1686
TA41	30×20	2006(1874)	2006	2013.3	2008	2018.7	-	2026
TA42	30×20	1937(1867)	1939	1962	1937	1949.3	-	1967
TA43	30×20	1848(1809)	**1846**	1848.6	1852	1863.1	-	1881
TA44	30×20	1983(1927)	**1979**	1988.7	1983	1992.4	-	2004
TA45	30×20	2000(1997)	2000	2000	2000	2000	-	2008
TA46	30×20	2004(1940)	2006	2013.6	2004	2015.5	-	2040
TA47	30×20	1894(1789)	1889	1901.4	1894	1902.1	-	1921
TA48	30×20	1943(1912)	1943	1949.7	1943	1959.2	2001	1982
TA49	30×20	1964(1915)	1967	1976.4	1964	1972.6	-	1994
TA50	30×20	1924(1807)	**1923**	1925.3	1925	1927	-	1967

最后使用 DMU 实例进行测试，DMU 实例与 TA 实例难度相当，同样被认为是最难的 JSP 实例之一。从 DMU 实例中选择了 40 个问题，新增了一种对比算法：i-TSAB[98]。表 3-13 给出了对比结果。对比结果表明，HGTSA 能够在 37 个实例上求得当前已知的最优解，并刷新了 3 个实例的结果：DMU11-3440、DMU12-3505、DMU17-3826。尽管 BRKGA 在所有 40 种情况下都能找到最著名的解决方案，但它没有找到更好的上限，这意味着 HGTSA 的探索能力更强。

表 3-13 DMU 实例的实验结果对比

Prob-lem	Size	UB(LB)	HGTSA		BRKGA		GES-TS	i-TSAB
			Best	M_{av}	*Best*	M_{av}	*Best*	*Best*
DMU01	20×15	2563(2501)	2563	2653	2563	2563	2566	2571
DMU02	20×15	2706(2651)	2706	2715.3	2706	2714.5	2706	2715
DMU03	20×15	2731(2731)	2731	2735.1	2731	2736.5	2731	–
DMU04	20×15	2669(2601)	2669	2675.6	2669	2672.4	2669	–
DMU05	20×15	2749(2749)	2749	2756.4	2749	2755.4	2749	–
DMU06	20×20	3244(2998)	3246	3252.3	3244	3246.6	3250	3265
DMU07	20×20	3046(2815)	3046	3060.1	3046	3058.6	3053	–
DMU08	20×20	3188(3051)	3188	3189.4	3188	3188.3	3197	3199
DMU09	20×20	3092(2956)	3092	3092.6	3092	3094.4	3092	3094

DMU10	20×20	2984(2858)	2984	2989.5	2984	2984.8	2984	2985
DMU11	30×15	3445(3395)	**3440**	3444.6	3445	3445.8	3453	3470
DMU12	30×15	3513(3481)	**3505**	3511.3	3513	3518.9	3518	3519
DMU13	30×15	3681(3681)	3682	3688.1	3681	3690.6	3697	3698
DMU14	30×15	3394(3394)	3394	3394	3394	3394	3394	3394
DMU15	30×15	3343(3343)	3343	3343	3343	3343	3343	–
DMU16	30×20	3751(3734)	3753	3760.3	3751	3758.9	3781	3787
DMU17	30×20	3830(3709)	**3826**	3846.2	3830	3850.6	3848	3854
DMU18	30×20	3844(3844)	3844	3838.4	3844	3845.4	3849	3854
DMU19	30×20	3770(3669)	3770	3788.6	3770	3791.8	3807	3823
DMU20	30×20	3712(3604)	3712	3716.2	3712	3715.3	3739	3740
DMU21	40×15	4380(4380)	4380	4380	4380	4380	4380	–
DMU22	40×15	4725(4725)	4725	4725	4725	4725	4725	–
DMU23	40×15	4668(4668)	4668	4668	4668	4668	4668	–
DMU24	40×15	4648(4648)	4648	4648	4648	4648	4648	–
DMU25	40×15	4164(4164)	4164	4164	4164	4164	4164	–
DMU26	40×20	4647(4647)	4647	4648.6	4647	4658.4	4667	4679
DMU27	40×20	4848(4848)	4848	4848	4848	4848	4848	4848
DMU28	40×20	4692(4692)	4692	4692	4692	4692	4692	–
DMU29	40×20	4691(4691)	4691	4691	4691	4691	4691	4691
DMU30	40×20	4732(4732)	4732	4732	4732	4732	4732	4732
DMU31	50×15	5640(5640)	5640	5640	5640	5640	5640	–
DMU32	50×15	5927(5927)	5927	5927	5927	5927	5927	–
DMU33	50×15	5728(5728)	5728	5728	5728	5728	5728	–
DMU34	50×15	5385(5385)	5385	5385	5385	5385	5385	–
DMU35	50×15	5635(5635)	5635	5635	5635	5635	5635	–
DMU36	50×20	5621(5621)	5621	5621	5621	5621	5621	–
DMU37	50×20	5851(5851)	5851	5851	5851	5851	5851	5851
DMU38	50×20	5713(5713)	5713	5713	5713	5713	5713	–
DMU39	50×20	5747(5747)	5747	5747	5747	5747	5747	–
DMU40	50×20	5577(5577)	5577	5577	5577	5577	5577	–

3.5.3 邻域结构有效性测试

邻域结构是目前 JSP 求解中最为先进的搜索方法之一。它能够根据问题的特征进行有针对性的搜索。为展示本文邻域结构的先进性，设计了相关实验，并与当前最先进的几种邻域结构进行对比。

首先，为了评估不同邻域结构在改进同一染色方面的效率，选用了LA01实例，通过使用随机算子生成 5 个初始可行解，并同时记录它们的初始完成时间。选择了 2 个先进的经典邻域结构，N8 和 N7 邻域结构，以及最新提出的两个多工序精确联动邻域结构：PN7+2MT 和 ICET+2MT（所选的比较邻域结构是根据

原始作者的描述尽可能复现的）。为了消除不同智能算法、算法设计和参数设置对测试结果的影响，这些邻域结构并没有集成到智能算法中，而是使用其固有的工序移动方法来产生邻域解。考虑到程序执行中的随机性，在实验中均独立求解 10 次。每次运行中，执行 50 次工序移动，每次移动都是从邻域结构覆盖的解空间中随机选择的。此次试验设计了两项指标用于评估结果，*Am* 代表完工时间的平均改进，*Ai* 代表平均改进的百分比，具体定义见式(3-31)。其中，$M_{initial}$ 表示未得到改善的个体完工时间。最终的测试结果如表 3-14 所示。

$$Am_{Best}=\frac{\sum_{i=1}^{5}Best}{5},\quad Am_{M_{av}}=\frac{\sum_{i=1}^{5}M_{av}}{5},\quad Ai=\frac{\sum_{i=1}^{5}Initial\ makespan-\sum_{i=1}^{5}M_{av}}{\sum_{i=1}^{5}Initial\ makespan} \tag{3-31}$$

表 3-14 不同邻域结构的实验结果对比

Problem	Initial makespan	N8T+2MT		PN7+2MT		ICET+2MT		N8		N7	
		Best	M_{av}	*Best*	M_{av}	*Best*	M_{av}	*Best*	M_{av}	*Best*	M_{av}
1	1203	807	872	827	932	972	1063	671	732	824	861
2	1056	697	725	835	855	723	884	765	794	794	857
3	946	716	762	702	782	788	802	724	765	814	834
4	1138	709	738	757	765	712	749	753	801	749	795
5	988	674	723	725	771	838	884	678	771	783	851
Am		720.6	764	769.2	821	806.6	876	718.2	772.6	792.8	839.5
Ai		-	28.3%	-	22.9%	-	17.8%	-	27.5%	-	21.3%

从表 3-14 中可以看出，N8T+2MT 的性能与 N8 非常接近，N8 在寻找最佳结果方面略优于 N8T+2MT，但 N8T+2MT 则表现出更好的总体平均结果和更大的稳定性。PN7+2MT 是基于 N7 邻域结构设计的多工序精确联动邻域结构，在测试中略优于 N7，这也是意料之中的。相比之下，ICET+2MT 是基于 N5 邻域结构的改进，在测试结果中排名较低。可以看出，N8T+2MT 的邻域结构与其他邻域结构相比具有一定的优势。

对于经典的 JSP，尽管单独使用复杂的邻域不一定能带来更好的解决方案，但解决方案空间的范围和改进邻域解决方案的效率仍然是评估邻域结构性能的关键标准。从 LA 中选择了 8 个不同大小的问题进行此测试。对于每个问题，使用随机算子生成 1 个初始解，每个邻域结构独立用于 100 轮工序移动。减少完工时间的次数被记录下来，并绘制出了条形图进行比较，如图 3-29 所示。竖轴表示减少完工时间的操作移动次数，而横轴则是不同实例的名称，不同的颜色

用于区分不同的邻域结构。其中竖轴上的数字表示在这 100 次实验中缩短完工时间的次数。N8T+2MT 在大多数情况下都有更多的改进次数。此外，随着案例大小和数据复杂性的增加，以及关键路径中关键块数量的增加，N8T+2MT 在提供邻域解决方案方面的效率也在提高。N8 邻域结构由于其包含了更大的邻域解空间，在提供邻域解决方案方面优于 N7。PN7+2MT 类似于 ICET+2MT，在与其基础邻域结构的对比中显示出一些优势。

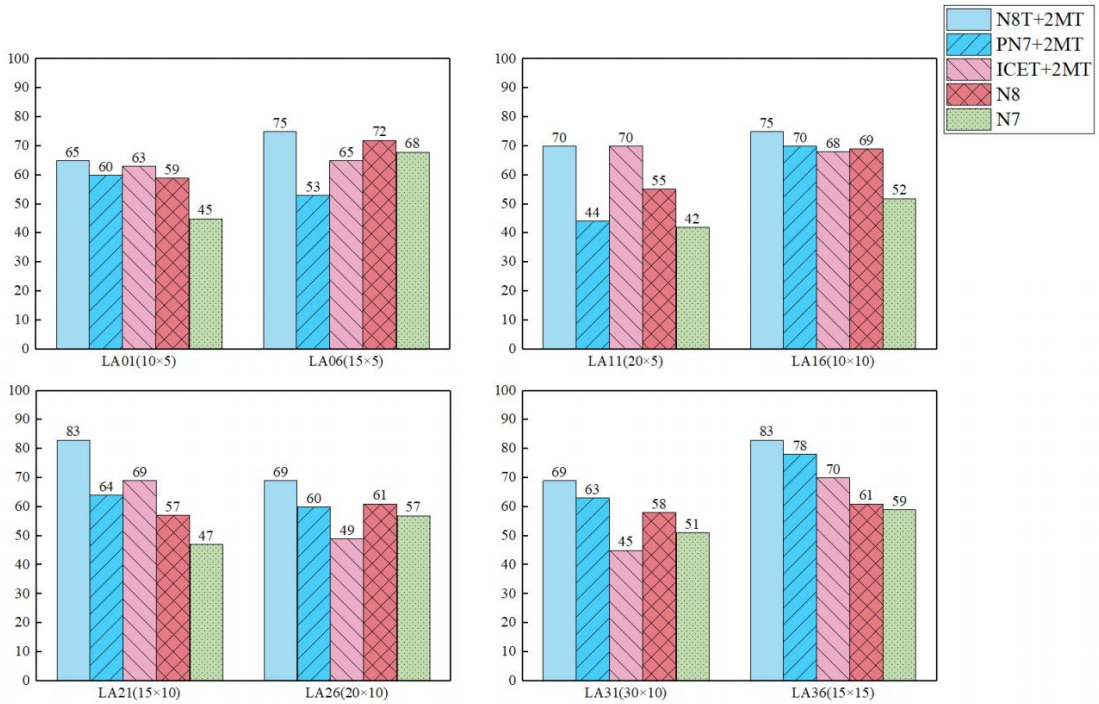


图 3-26 不同邻域结构改进同一实例的结果对比

在多工序精确联动邻域结构的开发过程中，会涉及对工序移动的有效性进行了详细分析，因此在与其基本邻域结构相比，求解效率会更高。然而，这并不一定意味着多工序精确联动邻域结构就一定更好，这是因为邻域结构只提供强大的局部搜索指导方法，仍然需要搭配有效的全局搜索来使用。只有当局部和全局搜索协同工作时，才能获得更好的性能。

3.6 本章小结

本章提出了一种基于多工序精确联动邻域结构的混合遗传禁忌搜索算法。在分析 N8 邻域结构中无效工序移动的基础上，提出了 8 种多操作联合运动，进一步丰富了 JSP 中邻域结构的研究。此外，还提出了一种调度部分重建策略来

增强全局搜索能力。为了解决传统进化算法在求解 JSP 时经常出现的种群多样性降低和过早收敛等问题，将遗传算法与禁忌搜索算法混合。此外，还使用了改进的 POX 交叉算子、基于类似 Metropolis 方法的选择算子、基于单台机器相邻工序移动的变异算子等方法。本文对邻域结构的发展做出了贡献，并为后续研究提供了新的思路。

此外本文也存在一些局限性。虽然HGTSA取得了较好的结果，但邻域结构的相对较大规模和算法结构的复杂性，可能导致运行时间较长，特别是在大规模实例测试的情况下，因为其每一次的工序移动之后，都需要重新计算完工时间，且在下一次邻域结构执行之前，都要重新计算关键路径，这是十分耗时的。未来的研究重点之一将是寻求近似评估方法来代替这些操作。另一个有趣的研究方向是将本文提出的邻域结构与不同制造环境中调度问题的其他特征相结合，并可能将其应用于各种车间调度问题，如多约束多目标的问题等。这可能需要调整和扩展算法以解决特定的问题特征或约束，并在更广泛的背景下评估其性能。

第4章 基于深度强化学习框架求解柔性作业车间调度问题

4.1 引言

FJSP 是 JSP 的扩展，在确定工序在机器上的加工顺序的同时，需要在每个工序的可用机器集合内挑选一个合适的机器，以最小化 makespan。由于决策的复杂度，即使是当前最高效的元启发式方法，在面对大规模案例时，也很难在较短时间内求出近似最优解，此外如果调度案例发生了更换，则需要耗费大量时间重新求解。本章提出了一种端到端的深度强化学习框架来求解 FJSP。在强化学习阶段，考虑到在 FJSP 中需要在作业和机器级别进行决策，设计了新的复合调度规则，并建立了合理的状态空间和奖励机制。在深度学习阶段，设计了基于长短期记忆网络的 actor 网络，以及双 critic 网络分别用于拟合状态值函数和动作值函数，提出了一种新的神经元架构，该架构通过引入二次项权重，来增加神经元表达能力。在决策时，车间的调度信息和实时状态信息被编码为多个矩阵进入神经网络，训练数据被存储在经验缓冲区，使用近端策略优化算法进行训练，最后使用 Adam 优化器进行参数更新。所提出的端到端框架通过在小规模数据集上进行训练，训练完成后可以用于解决任意规模的实例。实验结果表明，与其他深度强化学习框架相比，训练好的智能体具有很强的泛化能力，并在求解时间方面优于元启发式方法。

4.2 问题模型

FJSP 可以描述如下：给定含有 n 个工件的集合 $\mathcal{J}=\{J_1,J_2,...J_n\}$ 和 m 台机器的集合 $\mathcal{M}=\{M_1,M_2,...M_m\}$，每个工件 $J_i\in\mathcal{J}$ 都有 n_i 道工序，这些工序必须按照既定的顺序 $\mathcal{O}_i=\{O_{i,1},O_{i,2},...,O_{i,n_i}\}$ 进行加工，每道工序 $O_{i,j}\in\mathcal{O}_i$ 都可以在可用机器集合 $\mathcal{M}_{i,j}\subseteq\mathcal{M}$ 中的任意一台机器 M_k 上加工，$p_{i,j,k}$ 表示工序 $O_{i,j}$ 在机器

M_k 上的加工时间。此外，所有机器和工件在开始时都是可用的，每个工序一旦开始加工就不能中断，每台机器一次只能处理一个工序。FJSP 的优化目标是最小化完工时间 $C_{\max}$，如式(4-1)所示，其中 C_i 表示工件 J_i 的完工时间。

$$C_{\max} = \max\{C_i\}, i \in [1,2,...,n] \tag{4-1}$$

FJSP 的数学模型与 Dauzère-Pérès 等[77]人的模型相当。首先，给出了两个定义的二元变量，见式(4-2)和(4-3)。

$$x_{i,j,k} = \begin{cases} 1, & \text{if machine } k \text{ is selected for the operation } O_{i,j}, \\ 0, & \text{otherwise.} \end{cases} \tag{4-2}$$

$$y_{i,j,k,l} = \begin{cases} 1, & \text{if } O_{i,j} \text{ is performed on machine } k \text{ in priority } l, \\ 0, & \text{otherwise.} \end{cases} \tag{4-3}$$

定义一些符号如下：$S_{i,j}$ 为工序 $O_{i,j}$ 的开始时间，q_k 为分配给机器 k 的次数，$S_{k,l}^m$ 为在优先级 l 中对机器 k 执行操作的开始时间，l 是一个足够大的数。FJSP 的数学模型如式(4-4)至(4-11)所示。

$$\begin{aligned} &S_{i,j} + \sum_{k \in M_{i,j}} (p_{i,j,k} \cdot x_{i,j,k}) \le S_{i,j+1}, \\ &i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i - 1\} \end{aligned} \tag{4-4}$$

$$\begin{aligned} &\sum_{k \in M_{i,j}} x_{i,j,k} = 1, \\ &i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\}, k \in \{1,2,...,m\} \end{aligned} \tag{4-5}$$

$$\begin{aligned} &\sum_{l} y_{i,j,k,l} = x_{i,j,k}, \\ &i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\}, k \in \{1,2,...,m\}, l \in \{1,2,...,q_k\} \end{aligned} \tag{4-6}$$

$$\begin{aligned} &S_{k,l}^m + p_{i,j,k} \cdot y_{i,j,k,l} \le S_{k,l+1}^m, \\ &i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\}, k \in \{1,2,...,m\}, l \in \{1,2,...,q_k - 1\} \end{aligned} \tag{4-7}$$

$$\begin{aligned} &S_{k,l}^m + (1 - y_{i,j,k,l}) \cdot L \ge S_{i,j}, \\ &i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\}, k \in \{1,2,...,m\}, l \in \{1,2,...,q_k\} \end{aligned} \tag{4-8}$$

$$\begin{aligned} &S_{k,l}^m \le S_{i,j} + (1 - y_{i,j,k,l}) \cdot L, \\ &i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\}, k \in \{1,2,...,m\}, l \in \{1,2,...,q_k\} \end{aligned} \tag{4-9}$$

$$x_{i,j,k} \in \{0,1\},\quad i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\},\ k \in \{1,2,...,m\} \tag{4-10}$$

$$y_{i,j,k,l} \in \{0,1\},\quad i \in \{1,2,...,n\},\ j \in \{1,2,...,n_i\},\ k \in \{1,2,...,m\},\ l \in \{1,2,...,q_k\} \tag{4-11}$$

其中，约束(4-4)定义了同一作业的连续操作之间的优先级关系。约束(4-5)确保每个工序仅分配给其候选机器集中的一台机器。约束(4-6)强制在一台机器上以一个优先级处理每个工序。约束(4-7)确保每台机器一次只能处理一个工序。约束(4-8)和(4-9)防止同一机器 k 上的两个工序重叠。

4.3 马尔科夫决策过程

State：状态空间的表示必须非常严谨，充分表征与决策相关的工序和机器。对于工序，有必要了解工件的处理进度和剩余的处理时间。由于每个时间步的决策仅取决于车间的当前生产状态，因此工厂中机器的状态也至关重要。有必要了解机器的当前负载和系统的整体状态。时间步的全局状态表示为由 7 个状态函数水平关联形成的矩阵（见式(4-12)）。应当注意，在初始状态下，没有选择作业，并且所有 7 个状态函数的值都设置为 0。

$$s_t = [s_t^1 \quad s_t^2 \quad s_t^3 \quad s_t^4 \quad s_t^5 \quad s_t^6 \quad s_t^7] \tag{4-12}$$

状态 s_t^1 为工件 J_i 的下一个工序的加工时间。假设工件 J_i 的当前工序为 j。函数表示如式(4-13)所示。

$$s_t^1 = p_{i,j+1,k} \tag{4-13}$$

状态 s_t^2 为工件 J_i 剩余工序的加工时间总和，如果不同机器上的工序加工时间不同，则取平均加工时间。假设工件 J_i 的当前工序为 j，共有 h 个工序。函数表示如式(4-14)所示。

$$s_t^2 = \sum_{j'=j+1}^{h} p_{i,j',k} \tag{4-14}$$

状态 s_t^3 为工件 J_i 的处理进度，表示为 P_{J_i}，假设工件 J_i 的当前工序为 j，共有 h 个工序。函数表示如式(4-15)所示。

$$s_t^3 = P_{J_i} = \frac{j}{h} \tag{4-15}$$

状态 s_t^4 为当前机器的利用率，表示为 $U_{M_k}(t)$，即机器上的总加工时间 $\sum p_{i,j,k}$ 除以机器 M_k 在时间步 t 处的完工时间 $C_{\max}(M_k)$。函数表示如式(4-16)所示。

$$s_t^4 = U_{M_k}(t) = \frac{\sum p_{i,j,k}}{C_{\max}(M_k)} \tag{4-16}$$

状态 s_t^5 为所有机器的平均利用率，表示为 $U_{avg}(t)$。函数表示如式(4-17)所示。

$$s_t^7 \ s_t^5 = U_{avg}(t) = \frac{\sum_{k=1}^{m} \sum p_{i.j.k}}{\sum_{k=1}^{m} C_{\max}(M_k)} \tag{4-17}$$

状态 s_t^6 为当前时间所有机器的利用率与平均利用率之间的标准差。函数表示如式(4-18)所示。

$$s_t^6 = D_{std}(t) = \sqrt{\frac{\sum_{k=1}^{m} (U_{M_k}(t) - U_{avg}(t))^2}{m}} \tag{4-18}$$

状态 s_t^7 为当前机器完成时间与全局完成时间的比值。函数表示如式(4-19)所示。

$$s_t^7 = r(C_{\max}) = \frac{C_{\max}(M_k)}{C_{\max}(t)} \tag{4-19}$$

Action：现有的调度规则大多是从处理时间或到达时间等单一角度设计的。单独应用一个或多个调度规则显然是不合理的，有必要综合考虑作业和机器两个维度。本文将作业选择和机器分配视为一个复合决策。动作空间的具体描述如下：

(1) 选择加工时间最短的工序。如果有多个这样的工序，优先选择不会导致机器产生额外等待时间的工序。

(2) 选择加工时间最长的工序。如果有多个这样的工序，优先选择最早开始的工序。

(3) 选择剩余处理时间（平均）最长的工序。如果有多个这样的工序，优先

选择可最早完成的工序。

(4) 选择剩余处理时间（平均）最短的工序。如果有多个这样的工序，优先选择可最早开工的工序。

(5) 选择一个可以在当前最早完成时间的机器上加工的工序。如果有多个这样的工序，优先选择可最早开工的工序。

(6) 选择一个可以在当前最早完成时间的机器上加工的工序。如果有多个这样的工序，优先选择不会导致机器产生额外等待时间的工序。

(7) 选择任意一个可用的工序和机器。

Transition: 在当前状态s_t，智能体执行一个动作，一旦动作完成，环境将确定性地转换到新的状态s_{t+1}，智能体会获得奖励r_t。本文通过状态矩阵中的特定值来区分不同的状态。

Reward：奖励的设计至关重要，因为它决定了智能体在与环境交互的过程中，能否客观公平地评估在特定状态下采取的动作的质量。本文采用 Song 等人[64]提出的方法。具体来说，奖励设置为两个时间步$t-1$和t的完工时间之差，定义为$r_t = C_{\max}(t-1) - C_{\max}(t)$，当时间步为 0 时，奖励$r_0 = 0$。因此，一次完整训练的总奖励如式(4-20)所示。

$$\begin{aligned} R &= r_0 + r_1 + r_2 + ... + r_t \\ &= 0 + C_{\max}(0) - C_{\max}(1) + C_{\max}(1) - C_{\max}(2) + ... + C_{\max}(t-1) - C_{\max}(t) \\ &= -C_{\max}(t) \end{aligned} \quad (4\text{-}20)$$

易知总奖励为完工时间的相反数，即最大化总奖励的目标与最小化 makespan 的目标是一致的，因此该奖励的设置是合理的。

Policy：在本文中，策略$\pi_\theta(s_t, a_t)$表示对每个状态s_t执行不同动作的概率分布。该策略被参数化为一个神经网络，它以状态为输入，输出概率分布。随着训练的进行，参数会被调整，政策也会朝着最大化预期累积奖励的目标进行更新。

4.4 一种带有二次项权重的神经元架构

从数学角度来看，神经网络训练的实际过程可以看成函数拟合过程，从而

近似目标函数的真实值。然而，在训练过程中，真实值通常是未知的，因此经常通过随机初始化或使用中间值等方法来设置初始参数。这些初始参数构成了一个模拟策略，用以替代实际值。训练优化的主要目标是通过计算损失来衡量模拟策略和真实策略之间的差异，然后调整参数以最小化此误差并减少损失，这个过程通常会非常漫长。梯度的概念能够指导参数往最优的方向逼近，当使用基于梯度的方法更新参数时，梯度的方向可以不断变化。如果更新幅度太大，可能会超过最佳点或发散。相反，如果更新幅度过小，可能会导致收敛缓慢，甚至陷入局部最优。能否开发出一种方法来控制梯度对参数更新的影响，在尽可能不随意改变梯度大小的情况下，使参数能够平稳的更新，从而收敛至最优，这就是引入二次项权重的原因。

如图4-1所示，浅绿色区域表示模糊坐标范围，而真实坐标位于该范围内的某个地方，由红点表示。函数表达式可简化为式(4-21)。

$$\begin{aligned} y_1 &= a_1 x + c \\ y_2 &= a_2 x^2 + bx + c \end{aligned} \tag{4-21}$$

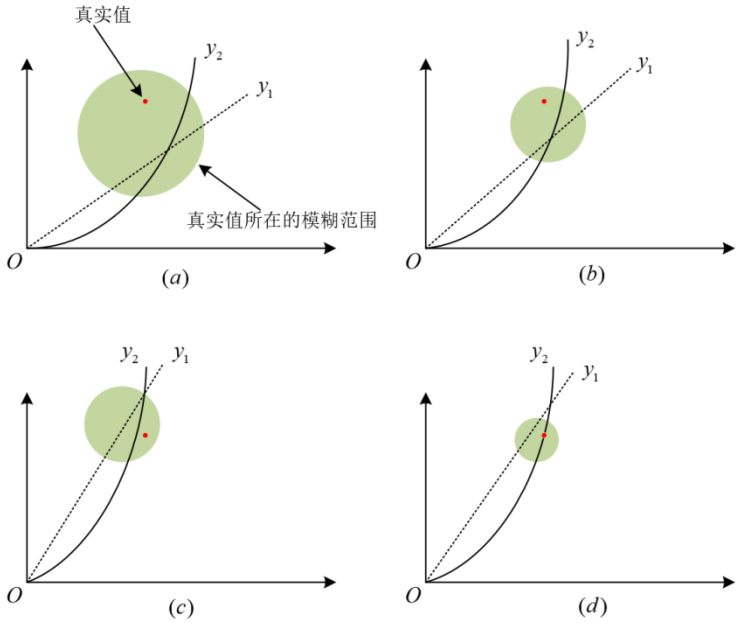


图 4-1 函数拟合的示意图

在参数初始化后如图 4-1 (*a*)所示，函数 y_1 和 y_2 都可以拟合模糊坐标，但不能拟合真实坐标。经过一些训练，如图 4-1(*b*)所示，模糊坐标范围逐渐变窄，函数 y_1 和 y_2 都更接近真实坐标。由于函数 y_1 只有一个参数 a_1，如果调整幅度太

大，更新过程中的轻微失调可能会超过真实坐标。相比之下，函数 y_2 有两个参数 a_2 和 b，所以即使如果整体调整幅度较大，则每个参数的幅度会减小，使其更接近真实坐标，如图 4-1 (*c*)所示。在图 4-1 (*d*)中，在训练进行了若干次后，函数 y_2 成功地拟合了真实坐标，而函数 y_1 由于其结构限制，拟合效果不如 y_2。

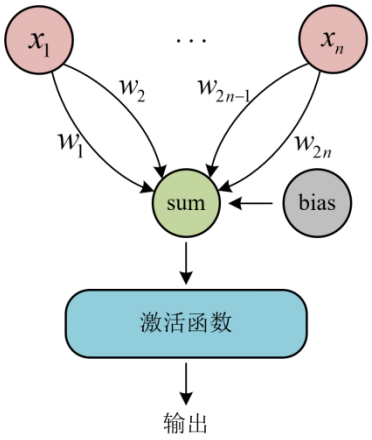


图 4-2 一种带有二次项权重的神经元架构示意图

图 4-2 给出了所提新神经元架构的示意图。神经元有 n 个输入，每个输入通过 $2n$ 个权重参数连接。加权和的计算为 $\text{sum}=\sum_{i=1}^{n}(w_{2i-1}\cdot x_i^{\,2}+w_{2i}\cdot x_i)$，其中 w_{2i-1} 和 w_{2i} 分别是与输入 x_i 的二次项和一次项相对应的权重。神经元的输出由式(4-22)表示。

$$y=f(\sum_{i=1}^{n}(w_{2i-1}\cdot x_i^{\,2}+w_{2i}\cdot x_i)+b) \tag{4-22}$$

4.5 深度强化学习框架

4.5.1 智能体与环境交互过程描述

一个完整的调度决策过程由若干个时间步组成，时间步的数量等于需要调度的工序总数。该过程大致如下：首先，导入训练数据集并初始化处理信息矩阵，包括可用工序矩阵、调度方案矩阵、可用机器矩阵和对应机器上的加工时间矩阵。接下来，建立初始状态矩阵，调用 actor 网络的前向传播函数，根据输出的概率分布选择动作编号并执行相应的动作。然后更新可用工序矩阵、调度

方案矩阵和状态矩阵，计算当前时间步获得的奖励值，并更新完成指标（即时间步）。最后，原始的状态、动作、奖励、更新后的状态和完成指标将被打包并存储在经验缓冲区中，以供后续进行参数更新使用。重复上述步骤，直到时间步达到预设，即可得到一个完整的调度方案。以下内容将详细描述每个步骤的操作。

表 4-1 一个包含 3 工件 4 机器的 FJSP 实例信息

Job	Operation	Machine and processing time		
		M_1	M_2	M_3
J_1	$O_{1,1}$	3	4	-
	$O_{1,2}$	2	-	5
	$O_{1,3}$	-	4	6
J_2	$O_{2,1}$	-	2	4
	$O_{2,2}$	3	-	1
	$O_{2,3}$	4	3	-
J_3	$O_{3,1}$	-	2	9
	$O_{3,2}$	1	3	-
	$O_{3,3}$	4	-	5
J_4	$O_{4,1}$	3	-	7
	$O_{4,2}$	3	4	-
	$O_{4,3}$	-	1	3

数据初始化：以表 4-1 提供的一个包含 3 个工件和 4 个机器的 FJSP 实例信息为例，为了便于计算机识别和操作，根据提出的方法创建了四个矩阵，如图 4-3 所示。图 4-3(*a*)显示了可用工序矩阵，该矩阵反映了哪些工序已经安排，哪些工序尚未安排。最初，所有工序都是未安排的，用 0 表示。当工序被调度之后，相应的 0 将被 1 替换。要查找可用的工序，只需从左到右查找对应位置的数字是否为 0 即可。调度方案矩阵（图 4-3(*b*)）显示了机器上计划加工工序的位置，大多数状态向量计算都将依靠该矩阵。图 4-3(*c*)和 4-3(*d*)显示了每个工序的

可用机器及其相应的加工时间。

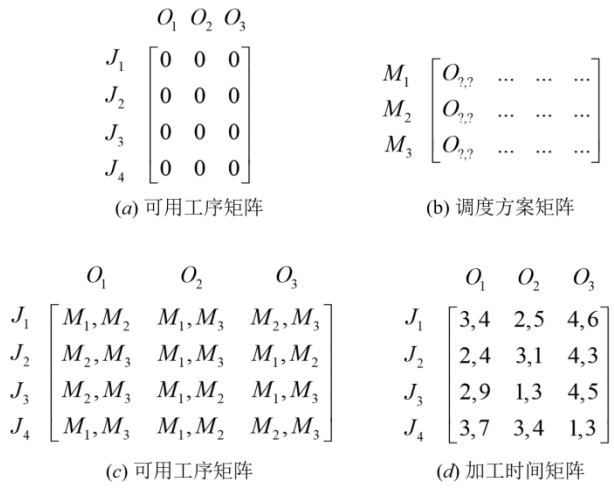


图 4-3 4 种矩阵的初始化

动作执行与环境交互：以时间步 t_1 为例，首先计算状态向量并形成如 4.1 节中描述的状态矩阵 s_0，将其输入到 actor 网络中，然后输出执行不同动作的概率，并使用 $\varepsilon-greedy$ 策略选择动作。假设选择了动作编号为 3，智能体将按照要求筛选出所有可选的工序。此时，所有可选工序为：$O_{1,1}$，$O_{2,1}$，$O_{3,1}$，$O_{4,1}$，然后计算这些工序对应工件的平均剩余加工时间，分别为 12、8.5、12、10.5。因此，$O_{1,1}$ 和 $O_{3,1}$ 被选出。此时，再计算 $O_{1,1}$ 和 $O_{3,1}$ 的完工时间，可以发现 $O_{3,1}$ 的开始时间为 0，对应机器 M_2 上的加工时间为 2，因此完工时间为 2，为可最早完成的工序。故当前时间步的动作 a_1 是选择工序 $O_{3,1}$ 在 M_2 上加工，开始时间为 0，结束时间为 2，调度操作后，将更新可用工序矩阵和调度方案矩阵。计算时间步的奖励：$r_1=0-2=-2$，并更新状态为 s_1。如果时间步数未达到预设值，则继续进行下一个时间步 t_2。当前时间步的训练数据包括 s_0，a_1，r_1，s_1，t_1 将被打包并存入经验缓冲区，以供后续参数更新使用。以上即是一个时间步中智能体与环境交互的完整过程。图 4-4 给出了此过程的示意图。

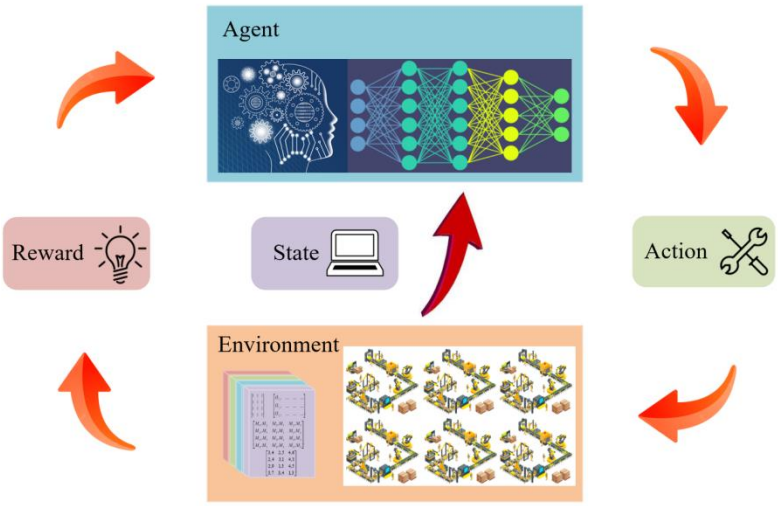


图 4-4 智能体与环境交互过程的示意图

4.5.2 actor 网络设置

对于 actor 网络，输入层需要接收状态作为输入。由于本文的状态空间是由 7 个向量组成的矩阵，因此设置 7 个神经元来分别接收每个状态向量是合理的。需要注意的是，输入层的权重设置为 1，偏置项设置为 0，并且不设置任何激活函数，这样做的目的是将原始数据真实且完整地传递给下一层。

LSTM 是循环神经网络(Recurrent Neural Network, RNN)的一种变体，通常用于处理和预测序列数据。LSTM 通过引入特殊的存储单元，解决了传统 RNN 在处理长序列时出现的梯度消失和爆炸问题。它主要由遗忘门、输入门、单元状态和输出门四部分组成，每个神经元将包含一组完整的门控单元。遗忘门用于控制哪些信息需要从单元状态中被遗忘，而输入门则决定将哪些新信息添加到单元状态中。单元状态负责记录当前内存信息，而输出门控制从单元状态到隐藏状态的信息流，并控制当前时间步的隐藏状态。

由于引入了二次项权重，每个神经元中的权重参数个数为 8。表 4-2 提供了每个门控单元进行数据处理的详细描述。W 和 b 分别表示权向量和偏置向量，σ 为 sigmoid 激活函数，tanh 为双曲正切激活函数，$[h_{t-1}, x_t]$ 表示两个向量的连接，h_t 为历史输出，x_t 为当前输入。与使用普通激活函数的典型隐藏层神经元不同，LSTM 层内部使用不同的门单元。这些门使用 sigmoid 和 tanh 函数控制信息流和内存更新。

表 4-2 每个门控单元的数据处理公式

门控单元	符号	计算公式
遗忘门	f_t	$f_t=\sigma(W_f^{(1)}\cdot[h_{t-1},x_t]+W_f^{(2)}\cdot[h_{t-1},x_t]^2+b_f)$
输入门	C_t	$C_t=\tanh(W_C^{(1)}\cdot[h_{t-1},x_t]+W_C^{(2)}\cdot[h_{t-1},x_t]^2+b_C)$
	i_t	$i_t=\sigma(W_i^{(1)}\cdot[h_{t-1},x_t]+W_i^{(2)}\cdot[h_{t-1},x_t]^2+b_i)$
单元状态	C_t	$C_t=f_t\cdot C_{t-1}+i_t\cdot C_t$
输出门	o_t	$o_t=\sigma(W_o^{(1)}\cdot[h_{t-1},x_t]+W_o^{(2)}\cdot[h_{t-1},x_t]^2+b_o)$
	h_t	$h_t=o_t\cdot\tanh(C_t)$

Actor 网络隐藏层剩余部分由全连接层组成，前一层的所有输入节点都连接到当前层的每个输出节点，形成线性组合。该过程将从前一层提取的局部特征组合为更高级的特征。全连通层的激活函数为 Leaky Rectified Linear Unit（LeakyReLU）。Actor 网络的输出层也是由全连接层组成，激活函数为 Softmax。

4.5.3 critic 网络设置

Critic 网络的主要功能是通过估计状态价值函数来评估和改进策略。它以状态为输入，输出状态值函数$V(s)$，反映了智能体在未来给定状态 s 下可以获得的预期总回报。它还以状态-动作对为输入，并输出动作值函数$Q(s,a)$，反映智能体在给定状态 s 中采取动作 a 在未来可以获得的期望总回报。本文设置双 critic 网络分别用于拟合$V(s)$和$Q(s,a)$。具体来说，critic1 将状态作为输入，并输出状态值函数$V(s)$，而 critic2 将状态-动作对作为输入并输出动作值函数$Q(s,a)$。两个 critic 网络的隐藏层和输出层均由全连接层构成，隐藏层的激活函数为 LeakyReLU，输出层的激活函数为 Softmax。

4.5.4 损失函数

在 PPO 中，actor 网络的损失函数（$L_{actor}(\theta)$）由三部分组成：策略改进项（$L_{clip}(\theta)$），熵正则项（$L_{entropy}(\theta)$）和价值函数误差项（$L_{vf}(\theta)$）。如式(4-23)所示，其中θ为策略参数。

$$L_{actor}(\theta) = L_{clip}(\theta) + L_{entropy}(\theta) + L_{vf}(\theta) \tag{4-23}$$

策略改进项（见式(4-24)）用于确保每个策略的更新幅度不会变得太大。其中，$\mathbb{E}_t$ 表示时间步 t 的预期回报的估计。$\frac{\pi_{\theta_{new}}(s,a)}{\pi_{\theta old}(s,a)}$ 表示新策略与旧策略的比率，表示两个策略之间的差异程度。ε 是裁切系数，若新旧策略之间的差变得太大，则会控制更新幅度。

$$L_{clip}(\theta) = \mathbb{E}_t[clip(\frac{\pi_{\theta_{new}}(s,a)}{\pi_{\theta old}(s,a)}, 1-\varepsilon, 1+\varepsilon) \cdot \sum_{i=1}^{n} A_t] \tag{4-24}$$

在熵正则项中（见式(4-25)），熵是信息论中的一个概念，用于测量随机变量的不确定性。在强化学习中，策略的熵表示在给定状态下采取行动的不确定性。较高的熵意味着策略更随机，而较低的熵表示策略更具确定性。其中 c_1 是权重参数。

$$L_{entropy}(\theta) = \mathbb{E}_t[-\frac{1}{n}\sum_{i=1}^{n} \pi_\theta(s,a) \cdot log\pi_\theta(s,a)] \cdot c_1 \tag{4-25}$$

价值函数误差项（见式(4-26)）的目标是使用均方误差作为损失函数，使估计值 $V_\phi(s_t)$ 和实际预期回报 $V(s_t)$ 之间的平方误差最小化。其中 c_2 是权重参数。

$$L_{vf}(\theta) = \mathbb{E}_t(V_\phi(s_t) - V(s_t))^2 \cdot c_2 \tag{4-26}$$

对于 critic 网络，均方误差(Mean Square Error, MSE)被用作损失函数。本文建立了两个 critic 网络，每个网络对应一个损失函数。式(4-27)是状态值函数的损失函数，式(4-28)是动作值函数的损失函数。

$$L_{critic}(\theta) = \frac{1}{n}\sum_{t=1}^{n} A(\theta)^2 \tag{4-27}$$

$$L_{critic}(\phi) = \frac{1}{n}\sum_{t=1}^{n} A(\phi)^2 \tag{4-28}$$

其中 $A(\theta)$ 和 $A(\phi)$ 都是优势函数，表示当前状态和状态-动作对的优势。n 代表总时间步。

4.5.5 梯度计算与误差反向传播

对于 actor 网络，根据链式规则，损失函数相对于策略参数的梯度等于损失

函数相对于策略的偏导数乘以策略相对于策略参数的偏导数，如式(4-29)所示。

$$\frac{\partial L_{actor}(\theta)}{\partial \theta}=\frac{\partial L_{actor}(\theta)}{\partial \pi_{\theta_{new}}(s,a)}\cdot\frac{\partial \pi_{\theta_{new}}(s,a)}{\partial \theta} \tag{4-29}$$

首先需要计算损失函数关于策略的偏导数，其方法是通过分别计算损失函数中每个项关于策略的偏导数，并将结果相加，如式(4-30)所示。

$$\frac{\partial L_{actor}(\theta)}{\partial \pi_{\theta_{new}}(s,a)}=\frac{\partial L_{clip}(\theta)+\partial L_{entropy}(\theta)+\partial L_{vf}(\theta)}{\partial \pi_{\theta_{new}}(s,a)} \tag{4-30}$$

对于策略改进项，取决于裁切范围与新旧策略的比值，如式(4-31)所示。

$$\frac{\partial L_{clip}(\theta)}{\partial \pi_{\theta_{new}}(s,a)}=\begin{cases}(1+\varepsilon)\cdot A_t, \dfrac{\pi_{\theta_{new}}(s,a)}{\pi_{\theta old}(s,a)}>1+\varepsilon\\ (1-\varepsilon)\cdot A_t, \dfrac{\pi_{\theta_{new}}(s,a)}{\pi_{\theta old}(s,a)}<1-\varepsilon\\ A_t, 1-\varepsilon<\dfrac{\pi_{\theta_{new}}(s,a)}{\pi_{\theta old}(s,a)}<1+\varepsilon\end{cases} \tag{4-31}$$

对于熵正则项，其计算如式(4-32)所示。

$$\frac{\partial L_{entropy}(\theta)}{\partial \pi_{\theta_{new}}(s,a)}=-c_1\sum\log\pi_\theta(s,a) \tag{4-32}$$

对于价值函数误差项，其计算如式(4-33)所示。其中$V(s_t)$是当前时间步对于未来奖励的估计值，使用蒙特卡洛方法计算，如式(4-34)所示。

$$\frac{\partial L_{vf}(\theta)}{\partial \pi_{\theta_{new}}(s,a)}=2c_2(V_\phi(s_t)-V(s_t)) \tag{4-33}$$

$$V(s_t)=G_t=R_t+\gamma R_{t+1}+\gamma^2R_{t+2}+\dots \tag{4-34}$$

使用反向传播算法计算策略相对于策略参数的偏导数。通过将$\frac{\partial L_{actor}(\theta)}{\partial \pi_{\theta_{new}}(s,a)}$与每层网络激活函数的导数相乘，可以得到损失函数相对于每层参数的梯度。

$$\frac{\partial \pi_{\theta_{new}}(s,a)}{\partial \theta}=\frac{\partial \pi_{\theta_{new}}(s,a)}{\partial y}\cdot\frac{\partial y}{\partial \theta} \tag{4-35}$$

对于输出层，激活函数为 Softmax。根据式(4-35)，神经元误差项δ需要乘以激活函数的导数，然后计算神经元输出的偏导数。由于输出层中的神经元也

是全连接的，因此输出的形式为 $y=wx+b$，其中 x 是神经元的输入值。因此，偏置项的梯度 $b=\delta$，权重的梯度 $w=\delta\cdot x$。对于隐藏层，其神经元也是全连接的，计算梯度的过程与输出层相似。但是需要注意的是，隐藏层神经元的激活函数是 LeakyReLU，因此在计算 δ 时，需要乘以激活函数 LeakyReLU 的导数。此外，对于 LSTM 层和全连接隐藏层，它们的神经元结构为 $y=w_1x^2+w_2x+b$，因此二次项权重的梯度 $w_1=\delta\cdot x^2$。

在 critic 网络中，由于实际回报是未知的，因此需要使用时间差分法来计算状态值函数和动作值函数的目标值函数。

式(4-36)为状态值函数的目标值函数，$V_\phi(s_t)$ 和 $V_\phi(s_{t+1})$ 分别是 critic 网络使用当前状态和下一状态来估算的状态值函数，r_t 是当前时间步的真实奖励，γ 是折扣因子。

$$A(\theta)=(r_t+\gamma V_\phi(s_{t+1}))-V_\phi(s_t) \tag{4-36}$$

式(4-37)为动作值函数的目标值函数，其中 $Q(s_t,a_t)$ 是 critic 网络分别使用当前状态动作对和下一个状态动作对来估算的动作值函数。r_t 是当前时间步的实际奖励，γ 是折扣因子。

$$A(\phi)=Q(s_t,a_t)-(r_t+\gamma\cdot Q(s_{t+1},a_{t+1})) \tag{4-37}$$

下一步是计算损失函数相对于 critic 网络输出的偏导数。本文有两个 critic 网络分别用于输出状态值函数和动作值函数。它们的偏导数也是分别计算的，如方程式(4-38)和方程式(4-39)所示。

$$\begin{aligned}\frac{\partial L_{critic}(\theta)}{\partial V_\theta(s)}&=\frac{1}{n}\sum_{t=1}^{n}\cdot 2\cdot A(\theta)\cdot\frac{\partial A(s_t,a_t)}{\partial V_\theta(s)}\\&=\frac{1}{n}\sum_{t=1}^{n}\cdot 2\cdot A(\theta)\cdot[\frac{\partial}{\partial V_\theta(s)}(r_t+\gamma V_\theta(s_{t+1})-V_\theta(s_t))]\\&=-2\cdot\frac{1}{n}\sum_{t=1}^{n}\cdot A(\theta)\end{aligned} \tag{4-38}$$

$$\frac{\partial L_{critic}(\phi)}{\partial V_\theta(s)}=2\cdot\frac{1}{n}\sum_{t=1}^{n}\cdot A(\phi) \tag{4-39}$$

Critic 网络的反向传播过程与 actor 网络的原理相同，这里将不再赘述。

4.5.6 Adam 优化器与参数更新

Actor 网络和 critic 网络的参数更新都是通过 Adam 优化器进行的。表 4-3 给出了使用的一些参数及其解释。

表 4-3 Adam 优化器相关参数及解释

符号	解释
β_1	一阶矩估计的衰减率
β_2	二阶矩估计的衰减率
ε	一个非常小的正数
l_r	学习率
m_t	一阶矩估计
v_t	二阶矩估计
m_t	一阶矩估计的偏差校正项
$\hat{v}_t$	二阶矩估计的偏差校正项

首先， m_t 和 v_t 被初始化为 0，并使用基于计算出的参数梯度 g_t 进行更新，如式(4-40)所示。然后，对 m_t 和 $\hat{v}_t$ 进行偏差校正，如式(4-41)所示。最后，更新权重和偏置项，如式(4-42)所示。

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1)\cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2)\cdot g_t^{\,2} \end{aligned} \tag{4-40}$$

$$\begin{aligned} m_t &= \frac{m_t}{1-\beta_1^t} \\ mv_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{4-41}$$

$$\begin{aligned} w_{t+1} &= w_t - l_r \cdot \frac{m_t^{w}}{\sqrt{\hat{v}_t^{w}} + \varepsilon} \\ b_{t+1} &= b_t - l_r \cdot \frac{m_t^{b}}{\sqrt{\hat{v}_t^{b}} + \varepsilon} \end{aligned} \tag{4-42}$$

4.5.7 DRL 总体框架

本文提出的端到端 DRL 框架运行过程可以描述如下。在训练阶段，首先根

据数据集中提供的处理信息建立初始矩阵，并根据相关指标计算出初始状态矩阵。状态矩阵被输入到 actor 网络中，预测在该状态下采取不同动作的概率，并选择一个动作，即分配一个待加工的工序给某台机器。加工完成后进入新的状态，并从环境中获得奖励。一个完整时间步的训练信息被存储在经验缓冲区中。在访问经验缓冲区之前，首先检查其存储的经验数量是否已达到预设的限制。如果缓冲区已满，将删除最早存储的经验，为后续新的经验腾出空间。这些步骤会重复执行，直到所有工序都完成加工。训练使用的数据集在若干次训练后会被更换，当训练迭代次数达到预设值时，进行参数更新。在参数更新阶段，首先从经验缓冲区中提取一批训练数据。对于这一批数据，需要计算每条数据关于神经元输出的损失函数的偏导数，然后取平均。利用链式法则，通过反向传播算法计算出每个参数的梯度。最后，使用 Adam 优化器更新网络参数。在训练过程中，网络参数更新若干次后，使用验证数据集进行性能评估，并记录结果以评估当前性能。训练完成后，程序会自动导出并保存参数，供决策使用。DRL 框架运行流程图如图 4-5 所示。

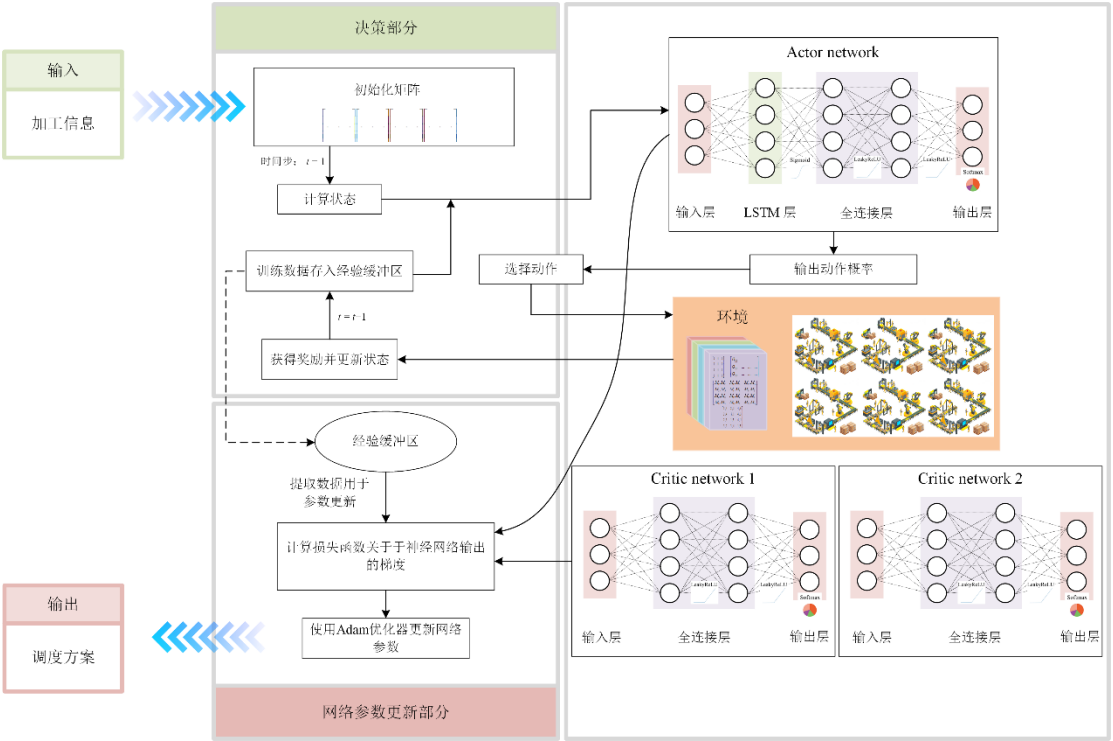


图 4-5 DRL 框架运行流程图

4.6 实验与仿真

所有测试程序均使用 C++语言编写，在 Microsoft Visual Studio 2022 环境下编译运行，使用安装 Windows 11 系统的计算机，CPU 频率为 2.3GHz，RAM 为 16GB。代码中未使用任何开源库，训练过程未使用 GPU 加速。

4.6.1 数据集与基准测试集

本文提出的 DRL 框架在大小为 10×5 和 15×10 的数据集上进行训练，每个数据集的训练数据集大小为 2000。所有训练集都是随机生成的。实验表明，10×5 数据集的训练时间为 28 分钟，15×10 数据集为 52 分钟。从训练时间来看，随着问题规模的增加，训练时间适度增长。由于训练是离线进行的，训练后的模型具有良好的泛化性能，因此时间是可以接受的。在测试中，本文首先使用随机生成的实例（10×10、15×15、20×10、20×20、30×20、50×20 和 100×20）进行验证。此外，作者在著名的公共 FJSP 基准上测试了所提出框架的性能，包括 Hurink[78]的实例和 Behnke[79]的实例。基准实例可以在 Monaldo Mastrolilli 的网页(https://people.idsia.ch/~monaldo/fjsp.html)上找到。

为了评估所提出框架的有效性，本文将其与多种 PDR 进行了比较。由于评估所有调度规则显然不可能，因此本文测试了 20 种流行的 PDR，并最终选择了 4 种表现最好的 PDR：FIFO(先进先出)、最多剩余操作(MOR)、最短处理时间(SPT)和加权最近工作剩余时间(MWKR)。对于随机数据集，本文使用 OR-Tools(一个广泛使用的商业约束编程求解器，以其鲁棒性著称)进行了基准测试，并对 OR-Tools 施加了 1800 秒的时间限制以确保公平。通过 OR-Tools 获得的最优解作为参考。此外，本文使用 Gap 值和决策时间作为评估指标。

4.6.2 消融实验

本节内容讨论了引入二次项权重对 DRL 框架性能的影响。在传统的神经元架构中，每个输入向量都有一个偏置项和一个权重。如果神经元的输入数量为 n，则神经元中的参数总数为 $n+1$。当引入二次项权重时，神经元中的参数数量增加到 $2n+1$。显然，随着神经元数量的增加，神经网络每个 epoch 的训练时

间也会增加。

测试了三种神经元架构，每种架构分别含有一个一次项权重、两个一次项权重以及一个二次项权重和一个一次项权重，同时保持其他设置不变。使用相同的随机数据集进行训练，设置 60 个 epoch，并在每个 epoch 结束后验证一次性能。不同神经网络结构的训练效果的收敛线图如图4-6所示。最初，与仅使用一个一次项权重或两个一次项权重的神经元架构相比，包含一个二次项权重和一个一次项权重的神经元架构表现出更优的性能和更快的收敛速度。随着训练进入后期阶段，具有一个二次项权重和一个一次项权重的神经元结构的运行结果仍然相对出色。从训练时间的角度来看，由三种神经元结构组成的网络完成了相同的 60 个 epoch 的时间分别为 802s、823s、821s，尽管参数计数有所增加，但训练负荷并没有显著增加。这是因为用于参数更新的损失函数及其在网络输出上的偏导数是相同的，引入的二次项权重并没有增加新的反向传播算法而造成额外负荷。综上所述，本文提出的神经元架构是可靠的。

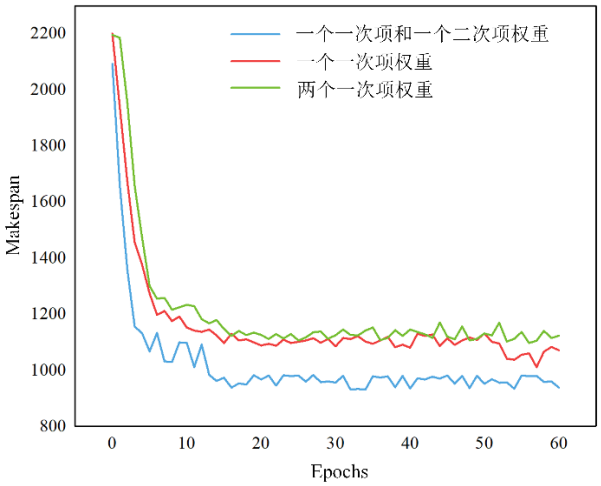


图 4-6 三种神经元架构训练效果对比

4.6.3 参数设计

超参数的取值对于 DRL 框架性能的影响很大。适当的超参数设置可以加速训练过程，使模型更快地达到收敛，从而节省计算资源。此外，适当的超参数调整有助于避免训练过程中的不稳定，使过程更平稳可控。考虑到一些超参数的值范围很广，测试所有值显然是不可能的。基于相关实验，本文通过定义一个合理的测试范围对超参数值进行了预处理。这些参数的值是使用 DOE(Design

of Experiments)方法确定的。目标是选择一种参数组合，以产生具有低可变性的稳定实验结果。由于超参数数量众多，本文进行了分组测试，以提高实验的可靠性。表 4-4 给出了超参数及其水平取值。

表 4-4 超参数及其水平取值

超参数名称	符号	因子水平			
		1	2	3	4
隐藏层维度	n_{hl}	1	**2**	3	4
隐藏层每层神经元数量	hl_{neuron}	16	32	**48**	64
裁切系数	ε	0.1	0.2	0.3	**0.4**
熵正则项权重系数	c_1	0.01	0.03	0.05	**0.07**
价值函数误差项系数	c_2	**0.5**	0.7	0.9	1.1
折扣因子	γ	0.8	0.85	**0.9**	0.95
学习率	lr	**1e-4**	3e-4	1e-3	3e-3
经验缓冲区规模	erb	5000	10000	15000	**20000**
每次更新使用数据数量	exp_{train}	30	**50**	70	90
每个训练集的使用周期	$num_{exp-train}$	**10**	20	30	40
动量参数 1	β_1	**0.9**	0.99	-	-
动量参数 2	β_2	0.99	**0.999**	-	-
指数衰减率	edr	**0.9**	0.99	-	-

根据 Minitab 软件生成的正交表，使用统一的随机数据集进行训练，进行超参数实验。终止条件设置为 60 个 epoch。由于数据量很大，这里就不列出具体正交阵列和均值响应表。图 4-7 给出了 M_{av} 和 M_{av} 的信噪比的结果，其中信噪比定义为 $-10\times\log10(\frac{\sum_{i=1}^{n}M_{av}^{\ 2}}{n})$。$M_{av}$ 代表算法的平均性能，较低的值表示在该参数设置下的平均性能良好。信噪比反映了性能稳定性，较高的信噪比表明在该参数设置下相对稳定，对随机变化的敏感性较低。根据图 4-7，最佳的参数选择在表 4-4 中以粗体标出。

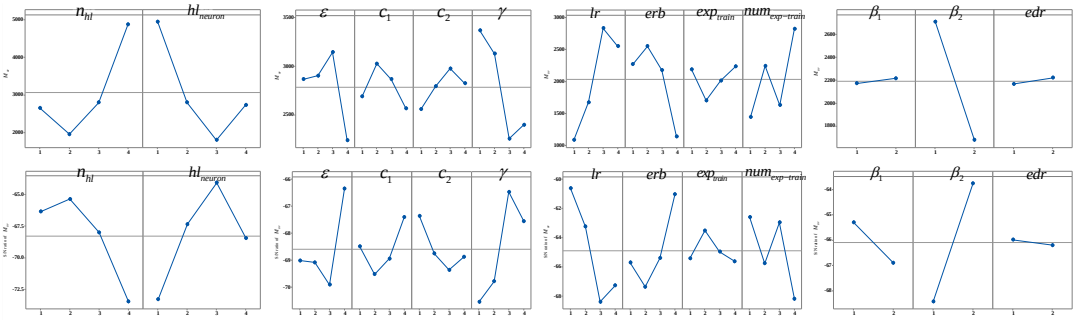


图 4-7 M_{av} 的均值与信噪比

4.6.4 随机实例的测试结果

本节内容使用了 70 个随机生成的实例测试集，包含 7 种不同规模的实例，规模范围从 10×10 到 100×20 不相同，每种规模分别有 10 个实例，根据一些学者提出的基准实例设置，其加工时间的生成范围为 1-99。这些实例用于测试本文框架在不同规模的随机实例上的泛化能力。对于本文提出的方法，分别使用 10×5 和 15×10 的随机训练集用于框架训练。作为比较，本文还列出了不同 PDR 方法和 OR-Tools 的六种组合，测试结果如表 4-5 所示，其中最佳结果使用粗体字标出。

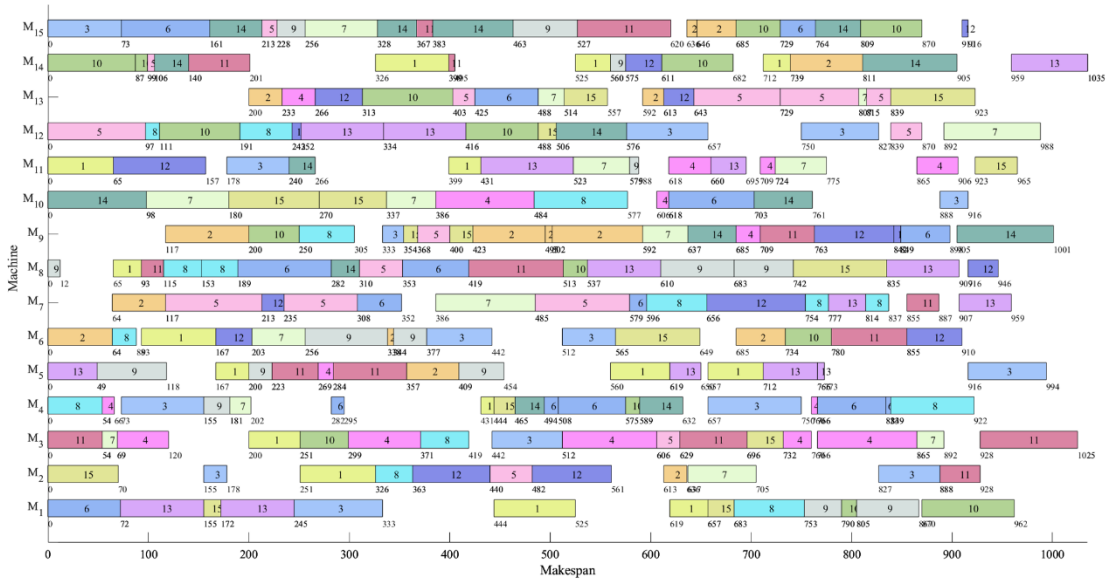
表 4-5 随机实例上的实验结果与比较

规模		OR-Tools	MOR+SPT	MOR+FIFO	MOR+MWKR	SPT+FIFO	SPT+MWKR	FIFO+MWKR	本文 10×5	本文 15×10
10×10	C_{max}	238.06	372.88	395.35	428.61	377.49	440.62	362.18	**255.62**	275.65
	Gap	0.00%	56.63%	66.07%	80.04%	58.57%	85.09%	52.14%	**7.38%**	15.79%
	Time	213.23s	0.36s	0.38s	0.43s	0.31s	0.33s	0.45s	0.33s	0.31s
15×15	C_{max}	404.45	569.62	657.19	618.84	572.65	685.46	635.85	**432.16**	438.67
	Gap	0.00%	40.84%	62.49%	53.01%	41.59%	69.48%	57.21%	**6.85%**	8.46%
	Time	1800s	0.35s	0.36s	0.44s	0.33s	0.36s	0.50s	0.59s	0.60s
20×15	C_{max}	532.45	788.78	725.84	772.73	886.15	798.16	879.46	**575.45**	589.60
	Gap	0.00%	48.14%	36.32%	45.13%	66.43%	49.90%	65.17%	**8.08%**	10.73%
	Time	1800s	0.38s	0.38s	0.39s	0.35s	0.35s	0.44s	0.49s	0.48s
20×20	C_{max}	688.33	1012.16	1134.42	1213.01	1084.18	1056.48	1123.48	**735.51**	781.61
	Gap	0.00%	47.05%	64.81%	76.23%	57.51%	53.48%	63.22%	**6.85%**	13.55%
	Time	1800s	0.40s	0.38s	0.39s	0.36s	0.35s	0.39s	0.57s	0.55s
30×20	C_{max}	723.13	1005.15	965.42	1078.17	1123.46	978.69	1235.24	**787.37**	788.96
	Gap	0.00%	39.00%	33.51%	49.10%	55.36%	35.34%	70.82%	**8.88%**	9.10%
	Time	1800s	0.41s	0.44s	0.41s	0.40s	0.39s	0.55s	0.88s	0.83s
50×20	C_{max}	1228.45	1702.36	1623.24	1672.31	1769.84	1698.49	1949.53	**1325.27**	1332.73
	Gap	0.00%	38.58%	32.14%	36.13%	44.07%	38.26%	58.70%	**7.88%**	8.49%

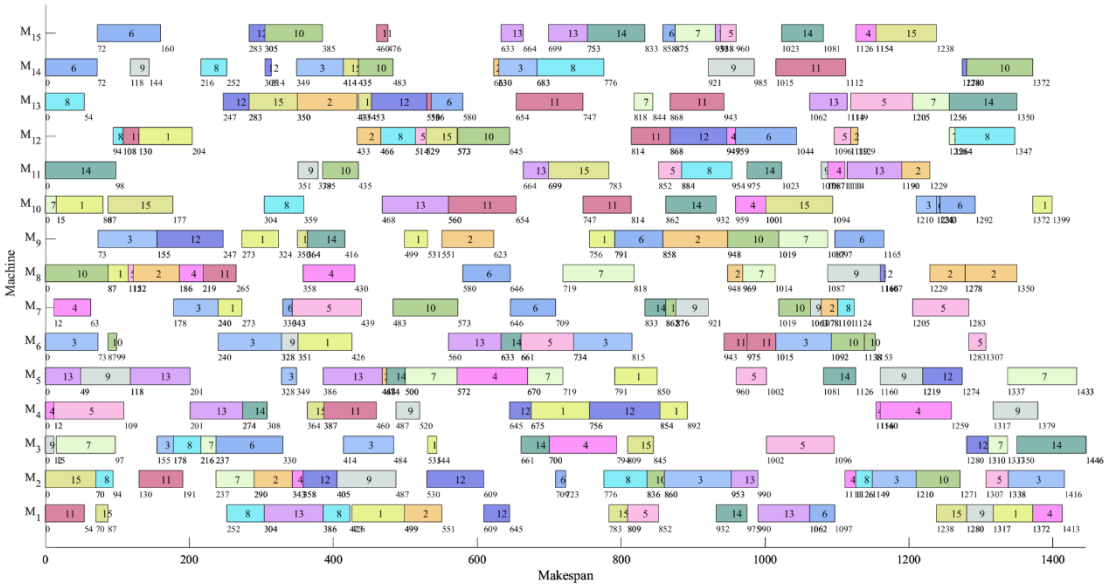
100×20	*Time*	1800s	0.66s	0.65s	0.50s	0.45s	0.46s	0.72s	0.96s	1.02s
	C_{max}	1932.42	2675.43	2629.92	2746.86	2563.51	2671.35	2866.46	2221.16	**2106.37**
	Gap	0.00%	38.45%	36.09%	42.15%	32.66%	38.24%	48.34%	14.94%	**9.00%**
	Time	1800s	0.71s	0.77s	0.79s	0.83s	0.67s	1.02s	1.56s	1.63s

为消除随机性对实验的影响，本文进行了 10 次独立测试。其中，$C_{\max}$ 为测试结果的均值，*Gap* 为平均差距的百分比，*Time* 为运行时间的均值。结果表明，OR-Tools 只能为最小的实例（10×10）提供最优解，但耗时 213.23 秒，这在实际生产过程中是无法接受的。在 10×10 的实例测试中，本文方法比最佳 PDR 方法（MWKR+SPT）高出 39.72%。对于 15×15 的实例，本文方法比 MOR+SPT 的结果高出 35.99%。而在 20×15、20×20 和 30×20 的实例中，本文方法比 PDR 组合平均提高了约 30%。对于更大的实例，50×20 和 100×20，尽管本文的方法和 OR-Tools 之间的性能差距有所增加，但本文的方法仍然比 PDR 保持优势。从求解时间的角度来看，尽管一些 PDR 提供的求解时间比本文的方法略短，但差异也远小于 1 秒，与显著提高的求解质量相比，这并不重要。此外，在 10×5 数据集上训练的模型显示出较优的性能。

图 4-8 显示了在 15×15 实例求解上使用本文方法的得出最优结果和最优 PDR 得出最优结果的 Gantt 图对比。通过比较可以看出，本文方法提供的解决方案在机器上的空闲时间更少，这意味着与 PDR 生成的最优解决方案相比，本文办法可以更有效地节省生产资源，提高效率。



(a) 使用本文方法得出结果绘制的Gantt图 (makespan = 1035)



(b) 使用最优PDR得出结果绘制的Gantt图 (makespan = 1446)

图 4-8 本文方法与最优 PDR 方法在 15×15 实例上的结果对比

总体而言，这些结果表明，本文提出的方法在不同规模的随机 FJSP 实例中是有效和可靠的，始终优于传统的 PDR，并且具有更好的可扩展性。

4.6.5 Brandimart 实例的测试结果

本节使用 Brandimart 的数据集中的 10 个实例进行测试，以评估其在随机数据集上训练并应用于这些实例时的泛化能力。实例的大小从 10×6 到 20×15 不等。测试结果如表 4-6 所示，其中最佳结果以粗体字标示。将本文方法与六种不同的

PDR 组合进行了比较。很明显，本文提出的方法优于所有 PDR。此外，将本文的结果与 Zhang 等[63]人和 Wen 等[64]人提出的框架进行了比较。本文方法在 10 个实例中有 8 个优于 Zhang 的方法，只有 2 个实例的结果略逊色。从平均结果来看，与 Zhang 的方法相比，本文的方法在所有实例中都得出了更好的结果。与 Wen 的方法相比，本文的框架在几乎所有情况下都表现出色。唯一的例外是 MK08 实例，Wen 的方法（在 20×10 的数据集上训练）比本文的方法好 0.3%左右。总体而言，本文所提框架在 Brandimart 的实例上表现出了卓越的性能，显示出有效性和可靠性。

表 4-6 Brandimart 实例的实验结果与比较

案例	规模	LB	UB	MOR+SPT	MOR+FIFO	MOR+MWKR	SPT+FIFO	SPT+MWKR	FIFO+MWKR	[63]	[65] 20×5	[65] 20×10	本文 10×5
MK01	10×6	36	40	60	63	60	48	58	60	44	55	48	**43**
MK02	10×6	24	26	42	52	41	46	42	40	31	42	43	**30**
MK03	15×8	204	204	335	235	303	238	315	319	**207**	232	213	208
MK04	15×8	48	60	172	82	71	81	121	168	69	82	75	**65**
MK05	15×4	168	171	265	188	223	186	213	259	177	205	185	**175**
MK06	10×15	33	58	104	100	81	102	123	105	77	110	103	**65**
MK07	20×5	133	139	219	218	238	215	230	222	**151**	215	214	156
MK08	20×10	523	523	598	535	578	532	555	587	531	525	**523**	525
MK09	20×10	299	307	468	416	464	375	392	441	**334**	424	333	**332**
MK10	20×15	165	197	352	382	289	272	211	368	245	266	266	**238**
Avg				261.5	227.1	234.8	209.5	226	256.9	186	216	200	**183.7**

4.6.6 Hurink 实例的测试结果

Hurink 的基准案例是一组基于现实世界生产调度问题的数据集，其特征是问题规模的多样性和不同程度的柔性。这些数据集包括从小到大的不同规模和柔性的实例，旨在评估算法的可扩展性和泛化性能。本文从中选择了 vdata、rdata 和 edata 三种数据集，每种数据集包含了 40 个实例，从 LA01 到 LA40，总共 120 个实例。表 4-7 显示了测试结果，其中最佳结果以粗体字表示。除了使用 OR-Tools 和 4 个著名的 PDR 进行比较外，本文还选择了两种最先进的启发式算法：基于规则的遗传算法（RegGA）和两阶段遗传算法（2SGA），这两种算法都是 Rooyani 和 Defersha[80]提出的，用于解决 FJSP。Wen 的方法也被添加用于比较。可以看出，OR-Tools 可以提供非常好的解决方案，但它仍然需要很长时间。在 vdata 实例上，2SGA 和 RegGA 的结果优于 OR-Tools，时间分别显著缩

短了 51.43 秒和 191.40 秒。2SGA 所需的时间更短是由于其初始解的质量更好。本文提出的方法在 10×5 的数据集上训练，在 vdata 测试中取得了良好的结果，优于 Wen 的方法，差值仅为 3.51%，耗时仅 0.33s。当在 15×10 的数据集上训练时，本文的方法在 edata 实例上表现出较好的性能，在 *Gap* 值上领先 Wen 的方法 0.77%。四种 PDR 组合在测试中的性能较差，与 OR-Tools、两种启发式算法和两种 DRL 框架相比存在很大差距。在决策时间方面，本文方法也优于 PDR 组合。总之，对于 Hurink 实例的测试结果，本文方法表现出了良好的性能，结果与预期一致。

表 4-6 Hurink 实例的实验结果与比较

方法	rdata(LA01~LA40)			edata(LA01~LA40)			vdata(LA01~LA40)		
	C_{max}	*Gap*	*Time*	C_{max}	*Gap*	*Time*	C_{max}	*Gap*	*Time*
OR-Tools	938.38	0.28%	1203.89	1026.70	-0.19%	226.53	924.40	0.38%	1367.77
[50] RegGA	-	-	-	-	-	-	836.13	3.20%	191.40
[50] 2SGA	-	-	-	-	-	-	812.20	0.39%	51.43
[40] 10×5	**1030.83**	**11.15%**	1.03	1187.48	15.53%	1.00	955.90	4.25%	1.00
[40] 15×10	1031.33	11.14%	0.97	1182.08	15.00%	1.04	954.33	4.02%	0.96
本文 10×5	1043.65	11.73%	0.34	1194.00	16.71%	0.33	**951.83**	**3.51%**	**0.33**
本文 15×10	1042.68	11.64%	**0.33**	**1170.95**	**14.23%**	**0.33**	956.37	4.01%	0.32
MOR	1064.04	14.71%	0.45	1211.23	17.91%	0.45	969.25	6.09%	0.45
SPT	1184.80	27.70%	0.46	1305.35	26.19%	0.46	1085.46	18.20%	0.46
FIFO	1082.08	16.63%	0.44	1255.46	22.07%	0.45	980.69	7.50%	0.44
MWKR	1046.20	12.53%	0.51	1179.90	14.92%	0.50	962.01	5.11%	0.50

4.6.7 讨论

DRL 对初学者来说非常抽象，在程序开发过程中，他们可能会遇到许多奇怪的问题。在本节中，作者将分享本文研究过程中遇到的一些问题和解决方案，希望对同行研究有所帮助。

研究过程中一个常见且令人不安的问题是，actor 网络输出的动作概率是一个无穷大的值，而初始训练阶段输出的概率值是正常的。有几种方法可以排除这个问题。一个可能的原因是学习率过高。在本文程序的早期测试版本中，将学习率从 1e-2 降低到 3e-3 和 1e-3 有效地缓解了这种症状。当学习率为 1e-2 时，在 42 个 epoch 处出现了概率无穷大的情况。当学习率为 3e-3 和 1e-3 时，无穷大概率分别出现在 161 和 356 个 epoch。虽然这并没有完全解决问题，但降低学习

率确实是一种有效的方法。另一个可能的原因是 Softmax 函数引起的数值溢出。标准的 Softmax 函数涉及以 e 为基数的求幂运算。当值非常大时，这可能会导致数值溢出，导致无穷大或溢出，如果超过程序的变量限制，则会导致无限值或负无限值。为了确保数值稳定性，可以在执行幂运算之前减去输入向量中的最大值。此修改可以解决此问题。

关于损失函数，重要的是要注意，虽然它是计算梯度的基础，但它并不直接参与参数的更新。实际需要的是损失函数相对于网络输出的导数，即梯度。然而，这种梯度不会直接用于更新参数，而是通过反向传播算法使用链式法则计算的，以确定每个参数的梯度。以输出层为例，该过程涉及计算误差项，该项等于损失函数相对于输出的导数乘以输出层的激活函数。随后，神经元偏差的梯度等于误差项。对于神经元中的每个权重，梯度等于神经元的梯度乘以权重的相应输入，以此类推。梯度在网络中从输出层到输入层的有条不紊的传播。

以上内容希望对后续相关研究人员有所帮助。

4.7 本章小结

本文提出了一种基于端到端的 DRL 框架来解决 FJSP，目标是最小化完工时间。在强化学习方面，设计了一种新的复合调度规则作为动作空间，以及合理的状态空间和奖励措施。调度过程使用矩阵序列表示，从中可以计算出局部状态。在深度学习方面，引入了一种新的神经元架构，并将其应用于基于 LSTM 的 actor 网络中。此外，还实现了双 critic 网络，分别用于拟合状态值函数和动作值函数。训练完成的框架可以用于求解任意规模的实例，该框架在小规模数据集上训练后能够在大规模实例的求解上表现出良好的泛化能力。所提方法在基准案例上进行了测试，显示出优异的泛化性能。使用不同大小的随机实例的实验结果也表明，所提出的方法优于现有的 PDR 及其组合，并表现出一定的优势。

未来一个有价值的研究方向是开发解决动态 FJSP（Dynamic FJSP）的框架。在现实生产中，工厂的状态很少按计划发展。各种变化都可能引发意想不到的问题，制造系统中不可预测的实时中断可能会改变进度的有效性。即使是轻微的干扰，如机器故障、原材料短缺或新订单的插入，也会使时间表不那么准确。

必须迅速处理这些意外事件，以保持生产效率。目前，用于解决 DFJSP 的 DRL 框架的开发还处于起步阶段，这使其成为一个有趣的研究方向。此外，在此基础上，可以考虑多目标 FJSP（Multi-objective FJSP），解决瓶颈机器负载和机器能耗等其他目标。这可以为解决更广泛的实际问题提供有价值的见解。

第5章 总结与展望

5.1 总结

本文分别为作业车间调度问题和柔性作业车间调度问题开发了一种有效的方法。在JSP求解方面，本文提出了一种混合遗传禁忌搜索算法（HGTSA），通过融合遗传算法（GA）与禁忌搜索（TS）的优势，提升了搜索的全局探索能力和局部优化精度。遗传算法部分利用改进的部分映射交叉（POX）算子和单台机器邻域搜索变异算子，使个体能更高效地跳出局部最优，而禁忌搜索部分则结合最新的N8邻域结构，进一步增强搜索能力。为提高算法的搜索效率，本文首先对现有的N8邻域结构进行了深入分析，并发现其中存在较多无效工序移动，即部分工序移动并不能有效缩短调度方案的最大完工时间（Makespan）。基于此，本文提出了一种多工序精确联动邻域结构（N8T+2MT），该结构不仅考虑了关键路径上的工序调整，还引入了跨机器的协同工序移动策略，以提高局部搜索的有效性。本文对该邻域结构进行了系统的理论推导，并证明了其有效性，从数学角度确保了其在求解过程中的合理性。在FJSP求解方面，本文提出了一种端到端的深度强化学习（DRL）框架，以最小化完工时间（Makespan）为优化目标，并构建了一个结合神经网络和强化学习的智能调度系统。首先基于马尔科夫决策过程（MDP）对FJSP建模，定义了状态空间、动作空间和奖励函数，并采用近端策略优化（PPO）算法对智能体进行训练。在神经网络架构方面，本文提出了一种新型神经元结构，在传统神经网络的基础上引入了二次项权重，并应用于基于长短期记忆（LSTM）的 Actor 网络。此外，为增强策略学习的稳定性和泛化能力，本文采用了双Critic网络来分别拟合状态值函数和动作值函数，避免了策略梯度的剧烈波动。在动作空间设计方面，本文提出了一种复合调度规则，结合传统调度规则（如 SPT、FIFO 等）与强化学习的决策机制，使得智能体能在不同生产场景下自动学习最优调度策略。同时，本文还构

建了一种基于矩阵编码的状态表示方法，以提升神经网络对车间状态的感知能力，并优化了奖励机制，以确保智能体能够朝着减少 Makespan 的方向进行优化。

5.2 展望

未来的研究可以在以下几个方向上进一步拓展：

JSP的更高效求解策略，进一步研究更具理论支持的邻域结构，减少无效搜索，提高计算效率。探索自适应搜索策略，使算法能够动态调整搜索方向，提升收敛速度。结合强化学习，使得算法能够根据问题实例自动学习最优的搜索策略。

FJSP 的动态调度优化现实生产环境中，调度计划常因机器故障、订单变更等不确定因素受到干扰，因此开发适用于动态 FJSP（Dynamic FJSP, DFJSP）的求解框架十分重要。当前 DFJSP 的 DRL 求解方法仍处于起步阶段，未来可以探索更具鲁棒性的实时优化策略。结合多目标优化（如能源消耗、设备维护成本等），提升调度方案的实际适用性。

算法的可解释性与工业应用当前深度强化学习框架存在一定的“黑箱”问题，未来研究可结合可解释性 AI 技术，提高调度决策的透明度。研究如何将所提方法部署到实际生产环境中，考虑生产线的动态调整，实现智能工厂的自动化调度。

通过这些改进，未来的作业车间调度优化方法将在理论研究和工业应用中发挥更大的作用，进一步提升智能制造环境下的生产效率。

参考文献

[1] Benitez G B, Ayala N F, Frank A G. Industry 4.0 innovation ecosystems: An evolutionary perspective on value cocreation[J]. International journal of production economics, 2020, 228: 107735.

[2] Zhong R Y, Xu X, Klotz E, et al. Intelligent manufacturing in the context of industry 4.0: a review[J]. Engineering, 2017, 3(5): 616-630.

[3] Brettel M, Klein M, Friederichsen N. The relevance of manufacturing flexibility in the context of Industrie 4.0[J]. Procedia Cirp, 2016, 41: 105-110.

[4] Liu A, Luh P B, Yan B, et al. A novel integer linear programming formulation for job-shop scheduling problems[J]. IEEE Robotics and Automation Letters, 2021, 6(3): 5937-5944.

[5] Yao Y J, Liu Q H, Li X Y, et al. A novel MILP model for job shop scheduling problem with mobile robots[J]. Robotics and Computer-Integrated Manufacturing, 2023, 81: 102506.

[6] 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.

[7] Talbi E G. Machine learning into metaheuristics: A survey and taxonomy[J]. ACM Computing Surveys (CSUR), 2021, 54(6): 1-32.

[8] Chen R, Yang B, Li S, et al. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem[J]. Computers & industrial engineering, 2020, 149: 106778.

[9] De Giovanni L, Pezzella F. An improved genetic algorithm for the distributed and flexible job-shop scheduling problem[J]. European journal of operational research, 2010, 200(2): 395-408.

[10] Wei F F, Cao C Y, Zhang H P. An improved genetic algorithm for resource-constrained flexible job-shop scheduling[J]. Int. J. Simul. Model, 2021, 20(1): 201-211.

[11] Chiou C W, Chen W M, Liu C M, et al. A genetic algorithm for scheduling dual flow shops[J]. Expert Systems with Applications, 2012, 39(1): 1306-1314.

[12] Huang X, Guan Z, Yang L. An effective hybrid algorithm for multi-objective flexible job-shop scheduling problem[J]. Advances in Mechanical Engineering, 2018, 10(9): 1687814018801442.

[13] Moslehi G, Mahnam M. A Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search[J]. International Journal of Production Economics, 2011, 129(1): 14-22.

[14] Sha D Y, Lin H H. A multi-objective PSO for job-shop scheduling problems[J]. Expert Systems with Applications, 2010, 37(2): 1065-1070.

[15] Xing L N, Chen Y W, Wang P, et al. A knowledge-based ant colony optimization for flexible job shop scheduling problems[J]. Applied Soft Computing, 2010, 10(3): 888-896.

[16] Seo M, Kim D. Ant colony optimisation with parameterised search space for the job shop scheduling problem[J]. International Journal of Production Research, 2010, 48(4): 1143-1154.

[17] Zhao B, Gao J, Chen K, et al. Two-generation Pareto ant colony algorithm for multi-objective job shop scheduling problem with alternative process plans and unrelated parallel machines[J]. Journal of Intelligent Manufacturing, 2018, 29: 93-108.

[18] Cruz-Chávez M A, Martínez-Rangel M G, Cruz-Rosales M H. Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem[J]. International Transactions in Operational Research, 2017, 24(5): 1119-1137.

[19] Wang B, Xie H, Xia X, et al. A NSGA-II algorithm hybridizing local simulated-annealing operators for a bi-criteria robust job-shop scheduling problem under scenarios[J]. IEEE Transactions on Fuzzy Systems, 2018, 27(5): 1075-1084.

[20] Vahedi Nouri B, Fattahi P, Ramezanian R. Hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities[J]. International Journal of Production Research, 2013, 51(12): 3501-3515.

[21] Nasiri M M, Kianfar F. A GES/TS algorithm for the job shop scheduling[J]. Computers & industrial engineering, 2012, 62(4): 946-952.

[22] Prot D, Bellenguez-Morineau O. Tabu search and lower bound for an industrial complex shop scheduling problem[J]. Computers & Industrial Engineering, 2012, 62(4): 1109-1118.

[23] Liu B, Fan Y, Liu Y. A fast estimation of distribution algorithm for dynamic fuzzy flexible job-shop scheduling problem[J]. Computers & Industrial Engineering, 2015, 87: 193-201.

[24] Du Y, Li J, Luo C, et al. A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations[J]. Swarm and Evolutionary Computation, 2021, 62: 100861.

[25] Lin T L, Horng S J, Kao T W, et al. An efficient job-shop scheduling algorithm based on particle swarm optimization[J]. Expert Systems with Applications, 2010, 37(3): 2629-2636.

[26] Li X, Gao L. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem[J]. International Journal of Production Economics, 2016, 174: 93-110.

[27] Chang H C, Liu T K. Optimisation of distributed manufacturing flexible job shop scheduling by using hybrid genetic algorithms[J]. Journal of Intelligent Manufacturing, 2017, 28: 1973-1986.

[28] Sun L, Lin L, Gen M, et al. A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling[J]. IEEE Transactions on Fuzzy Systems, 2019, 27(5): 1008-1022.

[29] Zhang R, Wu C. A hybrid immune simulated annealing algorithm for the job shop scheduling problem[J]. Applied Soft Computing, 2010, 10(1): 79-89.

[30] Chen W, Yang H, Hao Y. Scheduling of dynamic multi-objective flexible enterprise job-shop problem based on hybrid QPSO[J]. IEEE Access, 2019, 7: 127090-127097.

[31] Gonçalves J F, Resende M G C. An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling[J]. International Transactions in Operational Research, 2014, 21(2): 215-246.

[32] Van Laarhoven P J M, Aarts E H L, Lenstra J K. Job shop scheduling by simulated annealing[J]. Operations research, 1992, 40(1): 113-125.

[33] Matsuo H, Juck SUH C, Sullivan R S. A controlled search simulated annealing method for the single machine weighted tardiness problem[J]. Annals of Operations Research, 1989, 21: 85-108.

[34] Dell'Amico M, Trubian M. Applying tabu search to the job-shop scheduling problem[J]. Annals of Operations research, 1993, 41(3): 231-252.

[35] Nowicki E, Smutnicki C. A fast taboo search algorithm for the job shop problem[J]. Management science, 1996, 42(6): 797-813.

[36] Balas E, Vazacopoulos A. Guided local search with shifting bottleneck for job shop scheduling[J]. Management science, 1998, 44(2): 262-275.

[37] Zhang C Y, Li P G, Guan Z L, et al. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem[J]. Computers & Operations Research, 2007, 34(11): 3229-3242.

[38] Xie J, Li X, Gao L, et al. A new neighbourhood structure for job shop scheduling problems[J]. International Journal of Production Research, 2023, 61(7): 2147-2161.

[39] 赵诗奎,黄林,吕杰.Job shop 强化多工序联动邻域结构与近似评价研究[J].机械工程学报,2023,59(04):318-331.

[40] 巴智勇,袁逸萍,裴国庆,等.作业车间调度的多工序精确联动邻域结构混合进化算法[J].计算机集成制造系统, 2024, 30(02): 537-552.

[41] Özgüven C, Özbakır L, Yavuz Y. Mathematical models for job-shop scheduling problems with routing and process plan flexibility[J]. Applied Mathematical Modelling, 2010, 34(6): 1539-1548.

[42] Birgin E G, Feofiloff P, Fernandes C G, et al. A MILP model for an extended version of the flexible job shop problem[J]. Optimization Letters, 2014, 8: 1417-1431.

[43] Roshanaei V, Azab A, ElMaraghy H. Mathematical modelling and a meta-heuristic for flexible job shop scheduling[J]. International Journal of Production Research, 2013, 51(20): 6247-6274.

[44] Zhou M C, Chiu H S, Xiong H H. Petri net scheduling of FMS using branch and bound method[C]//Proceedings of IECON'95-21st Annual Conference on IEEE Industrial Electronics. IEEE, 1995, 1: 211-216.

[45] Lloyd S, Yu H, Konstas N. FMS scheduling using Petri net modeling and a branch & bound search[C]//Proceedings. IEEE international symposium on assembly and task planning. IEEE, 1995: 141-146.

[46] Hansmann R S, Rieger T, Zimmermann U T. Flexible job shop scheduling with blockages[J]. Mathematical Methods of Operations Research, 2014, 79: 135-161.

[47] Ahn J, Kim H J. A branch and bound algorithm for scheduling of flexible manufacturing systems[J]. IEEE Transactions on Automation Science and Engineering, 2023.

[48] Vilcot G, Billaut J C. A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem[J]. International Journal of Production Research, 2011, 49(23): 6963-6980.

[49] Cruz-Chávez M A, Martínez-Rangel M G, Cruz-Rosales M H. Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem[J]. International Transactions in Operational Research, 2017, 24(5): 1119-1137.

[50] Zhang G, Gao L, Shi Y. An effective genetic algorithm for the flexible job-shop scheduling problem[J]. Expert Systems with Applications, 2011, 38(4): 3563-3573.

[51] Chaudhry I A, Rafique A F, Elbadawi I, et al. Integrated scheduling of machines and automated guided vehicles (AGVs) in flexible job shop environment using genetic algorithms[J]. International Journal of Industrial Engineering Computations, 2022, 13(3): 343-362.

[52] Zarrouk R, Bennour I E, Jemai A. A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem[J]. Swarm Intelligence, 2019, 13: 145-168.

[53] Li J Q, Pan Q K, Gao K Z. Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems[J]. The International Journal of Advanced Manufacturing Technology, 2011, 55: 1159-1169.

[54] Xie Z, Yang D, Ma M, et al. An improved artificial bee colony algorithm for the flexible integrated scheduling problem using networked devices collaboration[J]. International Journal of Cooperative Information Systems, 2020, 29(01n02): 2040003.

[55] Wang H, Sheng B, Lu Q, et al. A novel multi-objective optimization algorithm for the integrated scheduling of flexible job shops considering preventive maintenance activities and transportation processes[J]. Soft Computing, 2021, 25: 2863-2889.

[56] Zhu Z, Zhou X. An efficient evolutionary grey wolf optimizer for multi-objective flexible job shop scheduling problem with hierarchical job precedence constraints[J]. Computers & Industrial Engineering, 2020, 140: 106280.

[57] Yuan Y, Xu H. An integrated search heuristic for large-scale flexible job shop scheduling problems[J]. Computers & Operations Research, 2013, 40(12): 2864-2877.

[58] González M A, Vela C R, Varela R. Scatter search with path relinking for the flexible job shop scheduling problem[J]. European Journal of Operational Research, 2015, 245(1): 35-45.

[59] Hmida A B, Haouari M, Huguet M J, et al. Discrepancy search for the flexible job shop scheduling problem[J]. Computers & Operations Research, 2010, 37(12): 2192-2201.

[60] Park J, Chun J, Kim S H, et al. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning[J]. International journal of production research, 2021, 59(11): 3360-3377.

[61] Hameed M S A, Schwung A. Graph neural networks-based scheduler for production planning problems using reinforcement learning[J]. Journal of Manufacturing Systems, 2023, 69: 91-102.

[62] 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.

[63] Zhang W, Zhao F, Li Y, et al. A novel collaborative agent reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for flexible job shop scheduling problem[J]. Journal of Manufacturing Systems, 2024, 74: 329-345.

[64] Wang R, 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, 2023, 35(3): 3091-3102.

[65] Song W, Chen X, Li Q, et al. Flexible job-shop scheduling via graph neural network and deep reinforcement learning[J]. IEEE Transactions on Industrial Informatics, 2022, 19(2): 1600-1610.

[66] 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.

[67] Zhao C, Deng N. An actor-critic framework based on deep reinforcement learning for addressing flexible job shop scheduling problems[J]. Math. Biosci. Eng, 2024, 21(1): 1445-1471.

[68] Li Y, Liao C, Wang L, et al. A reinforcement learning-artificial bee colony algorithm for flexible job-shop scheduling problem with lot streaming[J]. Applied Soft Computing, 2023, 146: 110658.

[69] Momenikorbekandi A, Abbod M. Intelligent scheduling based on reinforcement learning approaches: applying advanced q-learning and state–action–reward–state–action reinforcement learning models for the optimisation of job shop scheduling problems[J]. Electronics, 2023, 12(23): 4752.

[70] Long X, Zhang J, Qi X, et al. A self-learning artificial bee colony algorithm based on reinforcement learning for a flexible job-shop scheduling problem[J]. Concurrency and Computation: Practice and Experience, 2022, 34(4): e6658.

[71] Chen R, Yang B, Li S, et al. A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem[J]. Computers & industrial engineering, 2020, 149: 106778.

[72] Luo S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning[J]. Applied Soft Computing, 2020, 91: 106208.

[73] 王凌. 车间调度及其遗传算法[M]. 清华大学出版社, 2003.

[74] Johnson S M. Optimal two-and three-stage production schedules with setup times included[J]. Naval research logistics quarterly, 1954, 1(1): 61-68.

[75] Hartmanis J. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)[J]. Siam Review, 1982, 24(1): 90.

[76] Brucker P, Schlie J. A branch and bound algorithm for the flexible job-shop scheduling problem[J]. European Journal of Operational Research, 1988, 34(2): 258-268.

[77] Dauzère-Pérès S, Ding J, Shen L, et al. The flexible job shop scheduling problem: A review[J]. European Journal of Operational Research, 2024, 314(2): 409-432.

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

[79] Behnke D, Geiger M J. Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers[J]. Institut Für Betriebliche Logistik Und Organisation, 2012.

[80] Rooyani D, Defersha F M. An efficient two-stage genetic algorithm for flexible job-shop scheduling[J]. Ifac-Papersonline, 2019, 52(13): 2519-2524.

[81] 赵诗奎. 作业车间调度问题的多工序联动邻域结构研究[J]. 机械工程学报, 2020, 56(13): 192-206.

[82] 巴智勇,袁逸萍,裴国庆,等.作业车间调度的多工序精确联动邻域结构混合进化算法[J]. 计算机集成制造系统, 2024, 30(02): 537-552.

[83] 张超勇,饶运清,刘向军,等.基于 POX 交叉的遗传算法求解 Job-Shop 调度问题[J].中国机械工程,2004,(23):83-87.

[84] Hu K, Wang L, Cai J, et al. An improved genetic algorithm with dynamic neighborhood search for job shop scheduling problem[J]. Math. Biosci. Eng, 2023, 20: 17407-17427.

[85] John F. Muth, Gerald L. Thompson. Industrial scheduling[M]. New Jersey: Englewood Cliffs, N.J. : Prentice-Hall, [1963], 1963.

[86] Lawrence S. Resource constrained project scheduling[J]. an experimental investigation of heuristic scheduling techniques, 1984.

[87] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling[J]. Management science, 1988, 34(3): 391-401.

[88] Applegate D, Cook W. A computational study of the job-shop scheduling problem[J]. ORSA Journal on Computing, 1991, 3(2): 149–156.

[89] Storer R H, Wu S D, Vaccari R. New search spaces for sequencing problems with application to job shop scheduling[J]. Management Science, 1992, 38(10): 1495–1509.

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

[91] Demirkol E, Mehta S, Uzsoy R. Benchmarks for shop scheduling problems[J]. European Journal of Operational Research, 1998, 109(1): 137–141.

[92] Xie J, Li X, Gao L, et al. A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems[J]. Journal of Manufacturing Systems, 2023, 71: 82–94.

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

[94] Wang Y. A new hybrid genetic algorithm for job shop scheduling problem[J]. Computers & Operations Research, 2012, 39(10): 2291-2299.

[95] Peng B, Lü Z, Cheng T C E. A tabu search/path relinking algorithm to solve the job shop scheduling problem[J]. Computers & Operations Research, 2015, 53: 154-164.

[96] Yuan E, Cheng S, Wang L, et al. Solving job shop scheduling problems via deep reinforcement learning[J]. Applied Soft Computing, 2023, 143: 110436.

[97] Zhang C Y, Li P G, Rao Y Q, et al. A very fast TS/SA algorithm for the job shop scheduling problem[J]. Computers & Operations Research, 2008, 35(1): 282–294.

[98] Nowicki E, Smutnicki C. An advanced tabu search algorithm for the job shop problem[J]. Journal of Scheduling, 2005, 8: 145–159.

攻读学位期间所取得的科研成果

一、发表学术论文

[1] Kongfu Hu, Lei Wang, Jingcao Cai, et al. "An improved genetic algorithm with dynamic neighborhood search for job shop scheduling problem." Mathematical Biosciences and Engineering 20.9(2023): 17407-17427. (中科院四区，IF=2.5)

[2] Lei Wang, Kongfu Hu, Jingcao Cai, et al. " A hybrid genetic tabu search algorithm based on a multi-operation joint movement neighborhood structure for job shop scheduling problems." Complex & Intelligent Systems (中科院二区，accept)

[3] Kongfu Hu, Lei Wang, Jingcao Cai, et al. " Deep reinforcement learning with a new neuron architecture for solving flexible job shop scheduling." Alexandria Engineering Journal (中科院二区，投稿中)

二、获得的荣誉

[1] 安徽工程大学第四届研究生数学建模大赛二等奖

致谢

时光飞逝，转眼间我的研究生生涯即将画上圆满的句号。在这段充满挑战与成长的旅程中，得到了许多人的帮助与支持。在此，我怀着感激之情，向所有关心、支持、帮助过我的人们致以最诚挚的谢意！

首先，我要衷心感谢我的导师王雷教授，感谢您在我研究生期间的悉心指导与无私帮助。您不仅在学术上给予我巨大的帮助，开阔了我的研究视野，教会我严谨的科研态度和独立思考的能力，更在生活上给予了我莫大的关怀。每当遇到学术瓶颈和困惑时，您总是耐心地为我解答，指引我走出困境。您的宽容与智慧深深感染了我，也让我在科研道路上不断前进。感谢您无时无刻不在为我提供宝贵的指导与支持，我将终生铭记。

其次，我还要特别感谢课题组的所有同学，大家在一起讨论、学习和共同进步的日子，使我的研究生生涯充满了温馨与动力。正是你们的支持和鼓励，才让我在科研的道路上更加坚定和勇敢。

另外，我要感谢我的父母。在我成长的每一个阶段，您们总是给予我无私的关爱与支持。在我求学的道路上，您们为我提供了坚定的后盾和无尽的动力。每当我疲惫和困惑时，您们的鼓励和支持总是让我重新振作。没有你们的理解与陪伴，我无法专注于学术研究，也无法走到今天。感谢你们无条件的爱和付出，我将永远铭记在心。

最后，我要感谢所有帮助和关心过我的朋友们。在我遭遇低谷时，是你们的陪伴与鼓励让我重新找回了前行的力量。我将永远铭记你们的帮助与支持，继续努力，不辜负大家的期望。未来的道路依然充满挑战，但有了你们的陪伴和鼓励，我相信自己能够不断突破自我，迈向更高的目标。

尚德敏学　唯实惟新