2022 届研究生硕士学位论文

分类号：_____ 学校代码：10269

密 级：_____ 学 号：51194501076

華東師範大學

East China Normal University

硕士学位论文

MASTER'S DISSERTATION

论文题目：带有设置时间和资源约束的柔性作业车间调度问题的算法研究

院　　系：	软件工程学院
专　　业：	软件工程
研 究 方 向：	组合优化
指 导 教 师：	卜天明（副）教授
学位申请人：	叶孙飞

2022 年 4 月

Dissertation for Master's Degree in 2022

University Code: 10269

Student ID: 51194501076

East China Normal University

Title: A study of algorithms for the Flexible Job Shop Scheduling Problem with Setup Times and Resource Constraint

Department/School:	School of Software Engineering
Major:	Software Engineering
Research direction:	Combinatorial Optimization
Supervisor:	Asso Prof. Tian-Ming Bu
Candidate:	Sunfei Ye

April, 2022

叶孙飞 硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
杨智应	教授	上海海事大学	主席
彭超	副教授	华东师范大学	
杜育根	副教授	华东师范大学	

中文摘要

柔性作业车间调度问题（Flexible Job Shop Scheduling Problem，FJSP）是工业生产制造和流程规划过程中最关键的问题之一，是一个 NP-hard 问题，在生产调度领域中十分热门。在该问题中，若干个作业需要在若干台机器上进行处理，每个作业由若干道工序构成。同一作业的工序之间存在先后约束，每道工序可以在其可选机器集中任选一台进行处理，每台机器同时只能处理一道工序。传统的 FJSP 忽略了序列相关设置时间和资源约束，然而在实际生产过程中可能出现需要考虑这些约束的需求场景。因此本文提出了带有设置时间和资源约束的 FJSP。为了解决该问题，本文引入了哈里斯鹰优化（Harris Hawk Optimization，HHO）算法并结合强化学习算法（Reinforcement Learning，RL）提出了一种自学习哈里斯鹰优化（Self-learning Harris Hawks Optimization，SLHHO）算法。本文工作如下：

（1）实现了一个离散版本的 HHO 算法来解决带有序列相关设置时间和资源约束的 FJSP。HHO 算法自 2019 年被提出以来主要用于解决数值优化问题，并取得了非常好的性能表现，而 FJSP 是一种组合优化问题。本文首次使用 HHO 算法来解决带有设置时间和资源约束的 FJSP。HHO 算法的寻优空间是一个连续的空间，但是 FJSP 中的机器分配和工序顺序都是离散形式的，因此需要将算法中得到的实数表示的位置向量转换成离散整数表示的机器分配向量和工序顺序向量，才能用于解决 FJSP。

（2）设计了一种解码方式来尽量避免资源约束。本文的资源约束条件主要是指同一时刻给机器更换模具的操作员数量有限，因此在将机器分配向量和工序顺序向量解码成调度方案时需要尽可能避免机器因等待操作员而空闲的情况。

（3）使用强化学习算法来智能地调整算法的关键参数，从而提升算法的性能。本文根据问题模型的特征，合理地划分强化学习算法的状态集和动作集，同时设置合适的奖励方法和动作选择策略。

关键词: 柔性作业车间调度问题，序列相关设置时间，资源约束，哈里斯鹰优化算法，强化学习

ABSTRACT

The flexible job shop scheduling problem (FJSP) is a popular research topic in the field of production scheduling. It is an NP-Hard problem, and it is also one of the most critical problems in the process of industrial manufacturing and process planning. This problem requires assigning a group of workpieces to a group of machines for processing, and each workpiece is composed of multiple processes. There are certain sequential constraints between the processes. Each process can be assigned to multiple machines. Traditional FJSP ignores sequence-dependent setup times and resource constraints. However, these constraints should be taken into account during the manufacturing process. So this paper proposes the FJSP with setup times and resource constraints. This paper applies harris hawk optimization (HHO) algorithm and proposes a self-learning harris hawks optimization (SLHHO) algorithm combined with reinforcement learning algorithm for this problem. The works of this paper is as follows:

(1) A discrete version of the HHO algorithm is implemented to solve FJSP with sequence-dependent setup time and resource constraints. The HHO algorithm has been mainly used to solve numerical optimization problems since it was proposed in 2019, and has achieved very good performance, while FJSP is a combinatorial optimization problem. This paper is the first one to use the HHO algorithm for FJSP with setup time and resource constraints. The optimization space of HHO algorithm is a continuous space, but the machine assignment and operate sequence in FJSP are discrete. Therefore, the position vector of real number obtained by the algorithm should be converted into machine assignment vector and process sequence vector represented by discrete integer to solve FJSP.

(2) A decoding method is designed to avoid resource constraints as much as possible. The resource constraints in this paper mainly refer to the limited number of operators who can replace the tools for the machines at the same time. Therefore, when decoding the

machine assignment vector and operate sequence vector into the scheduling plan, it is necessary to avoid the situation that the machine waits for the operator to be idle as much as possible.

(3) Reinforcement learning algorithm is used to adjust the key parameters of the algorithm intelligently to improve the performance of the algorithm. According to the characteristics of the problem model of this paper, the state set and action set of reinforcement learning algorithm are reasonably divided, and the appropriate reward method and action selection strategy are set up.

Keywords: *flexible job shop scheduling, sequence-dependent setup times, resource constraints, harris hawks optimization, reinforcement learning*

目 录

插图

表 格

第一章 绪 论

1.1 研究背景与意义

随着社会的发展和科技的进步，世界各国的制造业迅猛发展,生产力的提升导致了竞争也越来越激烈。为了提升本国制造业的竞争力，各个国家相继提出了工业化战略。德国认为现在是利用智能化技术来促进工业发展的时代 [1]，美国想要利用科技来提高生产效率，降低能源消耗成本，从而在制造业占据领先地位 [2]。英国为了发展和复苏本国的的制造业，也提出了《英国工业 2050 战略计划》[3]。在这种严峻的环境下，为了实现制造强国的战略，中国在 2015 年推出了《中国制造 2025》[4]，以此作为第一个十年的行为纲领。该计划倡导促进制造业创新，致力于从消耗能源型转变为绿色发展型，提高生产效率，降低能源消耗。

随着竞争变得愈发激烈，制造业企业除了要尽可能地提高生产效率，减少资源消耗，提高盈利，也要为客户提供个人化、定制化的产品，来满足客户层出不穷的需求，这样才能吸引客户，争夺更多的市场。在实际生产过程中，在加工环节花费的时间实际上是很少的，大多是消耗在资源调配和生产调度环节，所以制定良好的资源管理和生产调度方案对提高企业生产效率和降低能耗成本至关重要。生产调度是生产制造中最关键的环节之一，通过设计优异的调度方案，可以更好的分配原料、机器、人力等资源，提升生产效率，降低能耗成本，充分利用资源，以此来提高企业的竞争力。

生产调度问题可以表示为在一定时间内，合理分配机器、人力、原料等资源给多个制造任务，排列出一个可行的调度方案并执行，最后得到满足客户需求的产品，交付给客户。作业车间调度问题（Job Shop Sheduling Problem, JSP) 是一个经典的生产调度问题。在 JSP 中，若干个作业需要被安排在有限数量的机器上进行处理。每个作业包含若干道工序，这些工序必须在相应的机器上进行处理并且同一作业的不同工序之间存在先后约束关系 [5]。

但是随着制造业向着个人化、定制化的趋势发展，客户需求越来越多，产品种类也越来越多，JSP 已经不适用于当前的生产模式。FJSP 应运而生，它是在 JSP 的基础上进行扩展得到的变种问题，更能符合实际生产制作过程的需求。与 JSP 相比，FJSP 中的每道工序都可以分配给一组可用的机器而不是特定的一台机器，这样可以保证车间生产的稳定性，即使有机器发生故障，也可以选择其他正常的机器进行加工，避免生产线陷入停滞。这虽然增加了调度的灵活性，但是也让问题变得更加复杂 [6]。

然而与实际制造过程相比，FJSP 还是稍显简单。它没有考虑很多实际制造过程中可能存在的约束条件，比如机器更换模具的设置时间、操作人员的数量限制、考虑到能源消耗的维护活动、模糊处理时间和重叠操作等等。这些约束条件都是实际生产制造过程中可能遇到的，十分具有研究价值。在 FJSP 的诸多约束中，序列相关设置时间在再制造工程中具有相当重要的工程意义。再制造是以废弃零部件为毛坯，重新加工成产品的制作过程，这样可以以较低的能源消耗挖掘出废弃零部件的价值。

在再制造过程中，处理机器在连续处理不同工序之间，需要调整或更换成相应的模具，在这个过程中所花费的时间被称为设置时间。设置时间根据是否与序列有关，可以划分成序列无关设置时间和序列相关设置时间。序列无关设置时间指的是一台机器连续处理同一个作业的不同工序时不需要更换模具，只有前后两道工序分属不同任务时才需要更换模具。而序列相关设置时间指的是一台机器连续处理同一个作业的不同工序时仍需要更换模具 [7]。相比于前者，后者更复杂，需要考虑的更多，但也更贴近实际生产环境。如果在同一时间，多台机器需要更换模具，而操作人员不足，部分机器不可避免地会陷入停滞状态，等待操作人员，这必然会延长作业的完工时间，并影响订单交付。因此在经典 FJSP 的基础上，研究带有序列相关设置时间和资源约束的 FJSP 更符合实际制造过程的某些需求场景，具有一定的研究价值，可以指导生产车间根据客户需求迅速产生高效的生产调度计划，加快车间生产速度，减少资源成本，增强市场竞争力。

1.2 研究现状

JSP 起源于 Johnson[8] 在 1954 年发表的一篇文章，文章中提出了一组作业必须以固定顺序先后经过两台机器加工的调度优化问题。Johnson 设计了一个简单的决策规则来获得最小完工时间，接着讨论并解决了受限制情况下的三机问题。这些虽然都是最简单的生产调度问题，但是可以用来模拟车间生产制造流程，对企业生产进行指导，有一定的研究价值。在过去的几十年的时间里，JSP 的变种问题层出不穷。到了 1990 年，Bruker 和 Schlie[9] 第一次提出了 FJSP。相比于 JSP，FJSP 将处理机器从唯一机器选择扩展到了多个机器选择，这虽然增加了问题的复杂性，使得求解更难，但是也提高了车间调度的灵活性和稳定性，即使有机器发生故障，也可以把工序安排在其他正常的机器上进行加工，降低机器故障带来的恶劣影响，有效的提升了车间生产效率。

1.2.1 FJSP 的变种模型

如果 FJSP 只针对一个性能指标进行优化，那就是单目标优化，目前研究比较多的性能指标有最大完工时间、货物交付延迟时间、机器负载和机器加工能源消耗等。其中，最大完工时间可以直接用来衡量车间生产效率，是最为直观的生产指标，也是目前研究最多的。Gao[10] 等人研究了带有模糊处理时间的单目标优化 FJSP，其优化目的是最小化最大完工时间。该问题中的处理时间还额外附加了不确定性，处理时间的模糊或者不确定性是再制造的七大特征之一。传统的 FJSP 只考虑了机器的灵活性，而一定程度上忽略了工人灵活性。鉴于人力因素在实际生产制造过程中提高生产效率和降低成本的影响和潜力，Gong[11] 等人提出了具有工人灵活性的单目标 FJSP，在所提出的问题中，机器需要工人去操作，所以除了要考虑工序顺序和机器分配外，还要考虑工人的分配。Abdelmaguid[12] 等人研究了具有可分离序列相关设置时间的 FJSP，还提供了具有随机邻域搜索功能的禁忌搜索方法。实验结果表明，对于平均设置时间与处理时间比率较低的实例，禁忌搜索方法平均能够实现低于 10% 的最优差距。

与单目标相比，多目标优化则是综合考虑了多个性能指标，在多个约束情况下寻找最优解，因此需要考虑的因素更多。Li 等人 [13] 研究了具有维护活动的多目标 FJSP 问题，主要针对最大完工时间、机器的执行任务数和关键机器的执行任务数这三个指标，而且分别考虑了机器可用情况和机器不可用情况。Ahmadi[14] 研究了随机机器故障下的多目标 FJSP，希望可以同时提高调度的最大完工时间和调度稳定性指标。Azzouz[15] 等人研究了具有序列相关设置时间的多目标 FJSP，优化目标是最小化两种目标函数：最大完工时间和双标准目标函数，为此还设计了一种遗传算法来解决该问题。在 2020 年，Li 等人 [16] 认为在现实生产要求中应同时考虑序列相关设置时间和传输时间，如果有绿色生产的要求，还要额外考虑机器加工过程中的能源消耗。为了解决该问题，他使用整数规划进行分析和建立模型，然后提出了一种增强的 Jaya 算法来解决。

1.2.2 解决 FJSP 的各种启发式方法

FJSP 是一种组合优化问题，是 NP-hard 的。由于它的这种特性，精确算法对于解决 FJSP 尤其是大规模问题效果很差，所以大多是使用启发式算法来解决 FJSP 的。在过去的几十年中，启发式算法在 FJSP 研究领域中非常流行。Kacem 等人 [17] 使用遗传算法（GA）来解决 FJSP，该算法集成了多种种群初始化以及选择遗传个体和繁殖个体的策略。实验结果表明，在算法中集成更多的策略会产生较优的结果。Du 等人 [18] 提出了一种自适应修改重要参数的遗传算法，该遗传算法中的重要参数是交叉和变异概率参数。在选择这两个参数时，使用自适应策略，每次更新都考虑父染色体的适应度值。如果父染色体的适应度值高于种群的一般水平，就需要降低这两个概率来减少在进化过程中破坏优秀个体的可能性。而如果父染色体的适应度值低于种群的一般水平，就需要提高这两个概率来期望产生适应度值更好的后代。Xing 等人 [19] 提出了一种蚁群优化算法（ACO）来解决 FJSP，该算法使用了知识模型。在该算法中，ACO 的优化过程会产生一些经验供知识模型学习，然后根据这些学习到的经验来指导启发式搜索。Gao 等人 [10] 将和声搜索算

法（HS）应用于带有模糊处理时间的单目标优化 FJSP。该算法模仿音乐创作过程，将音乐中的和声看成是解向量，模拟和声创造和声的过程，利用现有的解决方案来生成新的候选解决方案。

其他常用的算法还有布谷鸟搜索算法（CS）[20]、粒子群优化算法（PSO）[21]、模因算法 [22]、灰狼优化算法（GWO）[23]、模拟退火算法（SA)[24]、萤火虫算法（FF）[25] 和禁忌搜索算法（TS）[26] 等等。虽然已经存在多种多样的启发式算法，但是还没有一个启发式算法可以在各个方面都具有最佳性能。这些启发式算法在某些方面性能突出，但在其他一些方面性能较差，因此仍需要引入新的算法来满足各种需求。

哈里斯鹰优化（HHO）算法是一种新的高效群智能算法，由 Heidari 等人 [27] 在 2019 年提出。该算法源于主要生活在美国亚利桑那州南部地区的哈里斯鹰群的独特的合作狩猎行为的启发，整个算法过程分为六个探索和开发的阶段。HHO 算法在 29 个基准函数上进行仿真实验，实验结果还与其他主流优化器进行对比。实验结果表明 HHO 性能出色，是顶级的优化器之一。除此之外，HHO 算法还在六个著名的工程优化问题上进行了实验，均获得了最佳解决方案。HHO 算法不需要额外的调节参数，具有调节参数少的特点，而且也不需要考虑优化问题的梯度数学信息，具有广泛的适用性。目前它也已经被应用于许多领域，包括滑坡敏感性质 [28]、电力载荷分布 [29]、斜坡稳定性预测 [30]、结构优化 [31]、卫星数据集的图像去噪 [32] 和图像分割 [33] 等等。目前还没有 HHO 算法在 FJSP 上的相关研究，根据 HHO 在基准函数问题和工程优化问题上优良的表现，考虑使用 HHO 算法来解决带有序列相关设置时间和资源约束的 FJSP 还是有一定的研究价值的。

使用启发式算法解决 FJSP 是为了在可接受的计算时间内找到尽可能令人满意的解决方案。根据解决问题的方式，启发式方法可以分为两大类：分层法和集成法。

分层法解决 FJSP

分层法将 FJSP 分解为两个子问题，分别是机器分配和工序排序，然后单独处理它们，以此来降低求解 FJSP 的难度。当每道工序的处理机器分配固定时，剩下的排序子问题就变成了 JSP，所以分层法很自然就产生了。Brandimarte[34] 在 1993 年第一次使用分层法来解决 FJSP，他使用某些调度规则分配机器来解决分配子问题，然后使用 TS 来解决生成的 JSP。Kacem[35] 提出先使用本地化方法来解决机器分配问题，构造合理的分配方案，然后使用该方案控制的 GA 来解决工序排序问题。Xia 和 Wu[36] 提出了一种有效的混合方法来解决多目标 FJSP，他们先使用 PSO 来获得机器分配方案，然后利用 SA 对每台机器分配到的任务进行排序。

集成法解决 FJSP

不同于分层法，集成法同时考虑了机器分配和工序排序。而且过去的研究表明集成法通常比分层法拥有更好的性能，但它也更难设计。Chen 等人 [37] 提出了一种有效的 GA 来解决 FJSP，在他们的算法中，用于表示问题解决方案的染色体是一组数值构成的向量。前半部分表示各工序的机器分配信息，后半部分描述每台机器上的工序的排序顺序。数值实验结果表明他们的方法可以获得高质量的解决方案。Zhang 等人 [38] 提出了一种基于多阶段操作的 GA，将每道工序表示为一个阶段，将每台机器表示为一个状态，这样就将问题表述成多阶段多状态的问题，然后从动态规划的角度去寻找解决方案。Xing[19] 提出了一种基于知识的 ACO 算法，知识模型在 ACO 的运行中进行记录和训练，接着利用训练出来的知识来优化当前的本地化搜索。其中解决方案的表示由工序排序和机器选择两部分组成，学习算子重新组合两个解决方案来产生新的解决方案。

1.2.3 强化学习在组合优化中的应用

近年来，强化学习（RL）在人工智能领域获得了极其显著的成果，其应用的场景也是层出不穷，例如自动驾驶、棋类竞技、电子游戏等等。在强化学习中，代

理以收益最大化为目标，在与环境不断的交互过程中学习如何做决策。强化学习这种可以灵活、实时的响应环境的特性能够很好的应对复杂多变的实际生产制造过程，在调度领域的应用上大有前景。

Emary 等人 [39] 提出了一种 GWO 的变体，它使用强化学习与神经网络相结合来提高性能。该算法使用强化学习来设置算法参数，代理的开发能力取决于代理的经验库和当前搜索空间的状态。经验库基于神经网络构建，主要是将一组代理的状态映射到一组具体影响开发能力的相应动作，并且由所有的搜索代理更新，以此来反馈经验并不断增强未来的动作。Shahrabi 等人 [40] 在 2017 年就使用 Q-learning 算法来增强针对带有随机作业到达和机器故障的动态 FJSP 所提出的可变领域搜索（VNS）方法的性能。该算法通过不断改进来自强化学习的策略来选择重新调度过程的参数，还设计了一种新方法来根据所选参数的质量计算学习过程中的奖励。Chen 等人 [41] 采用 GA 作为基本优化方法，然后使用强化学习算法来智能地调整 GA 的遗传概率和变异概率。为了验证算法的性能，他们在多种规模的基准实例上进行了实验，实验结果中能够看出所提出的算法在解决 FJSP 方面优于竞争对手。

Hsieh 等人 [42] 提出了一种基于 Q-learning 的粒子群优化算法（QSO）。它将优化问题视为一种 RL 问题，将用于搜索最佳解决方案的优化过程视为寻找最佳策略以最大化预期回报的 RL 过程。QSO 算法的优化过程随着每个个体模仿群体中全局最佳个体的行为而进行，其中全局最佳个体是根据其累计表现而不是每次评估时的瞬时表现来选择的。Chen 等人 [43] 提出了一种利用强化学习控制遗传算法的新算法（SCGA），他们认为使用 Q-learning 算法会存在无法很好适应 GA 算法和需要训练时间的缺点，所以选择使用 SARSA 算法来控制 GA 算子的选择，这样既可以无需提前训练，而且在应用中更具适用性和可扩展性。Q-learning 算法和 SARSA 算法具有不同的特点，前者具有更好的学习性能，但需要漫长的训练。而 SARSA 算法具有更快的收敛速度，但很难达到最佳的效果。因此就有研究人员考虑是否可以结合两者的特点，既可以加快速度又可以提高学习效果。Wang 等人 [44] 提出了一种反向 Q-learning，将 Q-learning 和 SARSA 相结合直接调整 Q 值，然后 Q 值

影响动作选择，最终在提升学习速度的同时增强强化性能。

1.3 主要研究内容

本文主要研究带有序列相关设置时间和资源约束的 FJSP，目的是找到最短的最大完工时间。HHO 算法可以高效的解决优化问题，也已经应用于诸多研究领域，在这些领域中，HHO 算法均表现良好。但在 FJSP 研究领域，HHO 算法的相关研究还非常少。HHO 算法是一种基于种群的、无需考虑梯度信息的优化技术，可以用于解决各种优化问题，因此将 HHO 算法应用在 FJSP 领域是有一定研究价值的。近年来，强化学习在调度领域异军突起，它在提升启发式算法性能方面效果显著，也已经有不少研究人员使用强化学习算法来提升启发式算法的性能。因此，本文提出了一种基于哈里斯鹰优化算法和强化学习算法的自学习哈里斯鹰优化（Self-learning Harris Hawks Optimization, SLHHO）算法来解决带有序列相关设置时间和资源约束的柔性作业车间调度问题。本文主要工作如下：

（1）实现了一个离散版本的 HHO 算法来解决带有序列相关设置时间和资源约束的 FJSP。HHO 算法的寻优空间是连续的空间，种群个体的位置向量是用一组一定范围内的实数构成的向量。本文使用集成法的方式同时考虑机器分配和工序顺序，将位置向量的前半部分通过线性公式转换成离散的机器分配向量，将位置向量的后半部分通过编号排列的方式转换成离散的工序顺序向量。最后将这两个离散的向量根据其所蕴含的信息解码成一个可行的调度方案，用以解决 FJSP。

（2）设计了一个解码方法来尽量避免资源约束。机器在处理完一道工序后需要进行更换模具才能继续处理下一道工序，这期间耗费的时间就是设置时间。本文还考虑了更换模具过程中的人力成本。鉴于人力成本的高昂，车间不太可能为每台机器都安排一位操作员来更换模具，所以有可能存在某台机器需要更换模具时发现没有空闲的操作员的情况。因此本文提出了可以同时给机器更换模具的操作员数量有限并且少于机器数量这一约束条件。为此设计了一个解码方法，该方法在机器遇到无操作员可以更换模具时考虑根据某种规则重新调整当前工序的机器

分配，从而在解码过程中尽量避免机器等待空闲的操作员的情况。

（3）使用强化学习算法来智能地调整算法的关键参数，从而增强算法的性能。HHO 算法在搜索阶段会根据某个参数来选择随机搜索策略或根据其他个体位置搜索策略。该参数在设置时需要充分考虑到算法在不同的运行阶段对这两种搜索策略的需求。强化学习不仅会考虑种群过去的和当前的状态，还能适当的预测未来的状态。因此使用强化学习算法可以设置更有效果的参数，增强算法的性能。本文根据问题模型的特征，合理地划分强化学习算法的状态集和动作集，同时确定奖励方法和动作选择策略。

1.4 论文章节安排

第一章是绪论。首先介绍的是生产调度问题的相关研究背景和意义，简单描述了 FJSP 及其变种模型的相关概念，针对 FJSP 中一些很少被考虑到的约束条件，提出了本文研究的问题模型。接着简单介绍了国内外关于 FJSP 的一些研究，主要分为 FJSP 的变种模型、解决 FJSP 的各种启发式算法以及强化学习在组合优化中的应用。最后就是本文的研究工作以及论文的章节安排。

第二章是相关理论与技术介绍。首先是描述了带有序列相关设置时间和资源约束的柔性作业车间调度问题的概念，介绍了一些用到的数学符号的含义，还提供了一个问题实例帮助理解。接着介绍了 Heidari 等人在 2019 年提出的 HHO 算法的具体内容，用图表详细展示了算法的具体步骤和各个阶段。最后介绍了强化学习算法的一些相关知识，展示了强化学习算法的核心策略公式，并对 Q-learning 和 SARSA 这两个经典强化学习算法进行对比，阐述了它们的优缺点。

第三章是 HHO 算法的实现。本文首先将 HHO 算法中用实数组成的位置向量转换为离散的机器分配向量和工序顺序向量，并针对本文所提出问题的约束条件，设计了一种解码方式来尽可能地避免资源约束。最后根据两个 FJSP 的经典数据集生成适合本文问题模型的数据集，并与其他有效算法进行对比实验。实验结果表明了 HHO 算法在解决带有序列相关设置时间和资源约束的柔性作业车间调度问

题上十分有效。

第四章是 SLHHO 算法的实现。首先根据本文提出的问题模型特征来构建环境状态评估模型，接着划分强化学习算法的状态集合和动作集合并确定动作选择策略和激励方法。最后在之前创建的数据集上与未使用强化学习算法的 HHO 算法和其他有效算法进行对比实验。结果表明 SLHHO 可以得到质量更高的解决方案，而且额外产生的代价非常小。

第五章是总结和展望。对本文提出的自学习哈里斯鹰优化算法进行工作总结，并指出一些尚存的缺点，为之后的研究提供新的方向。

第二章　相关理论与技术介绍

2.1　问题描述

带有序列相关设置时间和资源约束的 FJSP 描述如下：某工厂车间有 n 个需要处理的独立作业 $J = J_1, J_2, ..., J_n$ 和 m 台能够使用的机器 $M = M_1, M_2, ..., M_m$。作业 J_i 由一些具有先后约束关系的工序组成 $O_{i1}, O_{i2}, ..., O_{iu_i}$，这些工序需要按固定的次序依次处理。工序 O_{ij} 可以选择其可选的诸多机器 M_{ij} 中的一台机器进行处理，而且不同工序可以分配的机器也不一定是一样的，有些工序只能由部分机器进行处理，而有些工序可以由所有机器进行处理，具体情况视工序而定。每道工序的处理时间由处理的机器决定，机器 M_k 处理工序 O_{ij} 的处理时间为 p_{ijk}。机器 M_k 在处理完一道工序 O_{gh} 之后，需要一名操作员进行更换相应的模具才能继续处理下一道工序 O_{ij}，这期间花费的就是序列相关设置时间 s_{ghijk}。工厂一共有 w 名操作员可以同时给机器更换模具，所以如果当前没有空闲的操作员时，那么需要更换模具的机器就会陷入等待状态，直到等到第一个空闲的操作员给它更换模具以后才能继续运行。表2.1总结了本节中主要使用的一些符号。在处理过程中需要满足如下条件：

(1) 不考虑机器之间的运输时间。

(2) 每台机器同时只能处理一道工序。

(3) 同一时刻，每个操作员最多只能给一台机器更换模具。

(4) 每道工序只能分配给一台机器进行处理，并且处理时是非可抢占的。

(5) 在同个作业的工序之间具有优先级约束, 在不同作业的工序之间则不具有优先级约束。

FJSP 有诸多常用的性能指标，比如最大完工时间、最小延期、最小机器载荷等等。其中最大完工时间指的是从零时开始，到所有作业的全部工序处理完成的总时间。它是 FJSP 研究中使用最多的指标之一，也是用来衡量企业生产效率的一

表 2.1: 符号含义

符号	意义
J	作业集合
M	机器集合
n	需要处理的作业数量
m	可使用的机器数量
u_i	作业 J_i 的工序数量
w	同时可以更换模具的操作员的数量
O_{ij}	作业 J_i 的工序 j
M_{ij}	工序 O_{ij} 的机器集合
C_i	作业 J_i 的最后一道工序处理完成的时间
$C_{i,j}$	工序 O_{ij} 的完成处理时间
ET_k	机器 M_k 上当前任务的结束时间
p_{ijk}	工序 O_{ij} 在机器 M_k 上的处理时间
s_{ghijk}	在机器 M_k 上，先处理工序 O_{gh}，再处理工序 O_{ij} 之间的设置时间

项重要指标。所以本文提出问题的目的就是获得最短的最大完工时间：

$$f = \min(\max_{1\le i\le n} C_i), \tag{2.1}$$

其中 C_i 是作业 J_i 的完工时间，很显然，C_i 就等于作业 J_i 的最后一道工序 O_{iu_i} 处理完成的时间。

为了更加清楚地描述该问题，本文提供了一个有 3 台机器、3 个作业的实例。表2.2展示了各个作业的各道工序在各台机器上的处理时间，没有数字表示该工序无法在该机器上进行处理。表2.3展示的是机器 M_1 上的序列相关设置时间。列名表示在机器 M_1 上先处理的工序，行名表示后处理的工序，start 表示机器 M_1 尚未开始工作，只要操作员更换好模具马上就可以进行处理。例如在机器 M_1 上先处理工序 O_{11}，再处理工序 O_{21} 所需要的设置时间为 2。

由于篇幅限制，这里仅列出了机器 M_1 的设置时间表，其他机器的设置时间表也存在，但没有详细列出。图2.1是该实例的一个解决方案的甘特图。蓝色矩形表示设置时间，白色矩形表示处理时间。该实例中同时可以给机器更换模具的操作员

表 2.2: 处理时间

作业	工序	机器		
		M_1	M_2	M_3
作业 1	O_{11}	5	5	4
	O_{12}	4	2	3
作业 2	O_{21}	2	2	1
	O_{22}	3	4	3
作业 3	O_{31}	2	4	5
	O_{32}	2	-	2

表 2.3: 机器 M_1 上的设置时间

	O_{11}	O_{12}	O_{21}	O_{22}	O_{31}	O_{32}
start	1	1	1	2	1	1
O_{11}	-	1	2	1	1	2
O_{12}	-	-	2	2	2	1
O_{21}	1	2	-	1	1	1
O_{22}	1	2	-	-	2	2
O_{31}	1	2	1	1	-	1
O_{32}	2	2	2	1	-	-

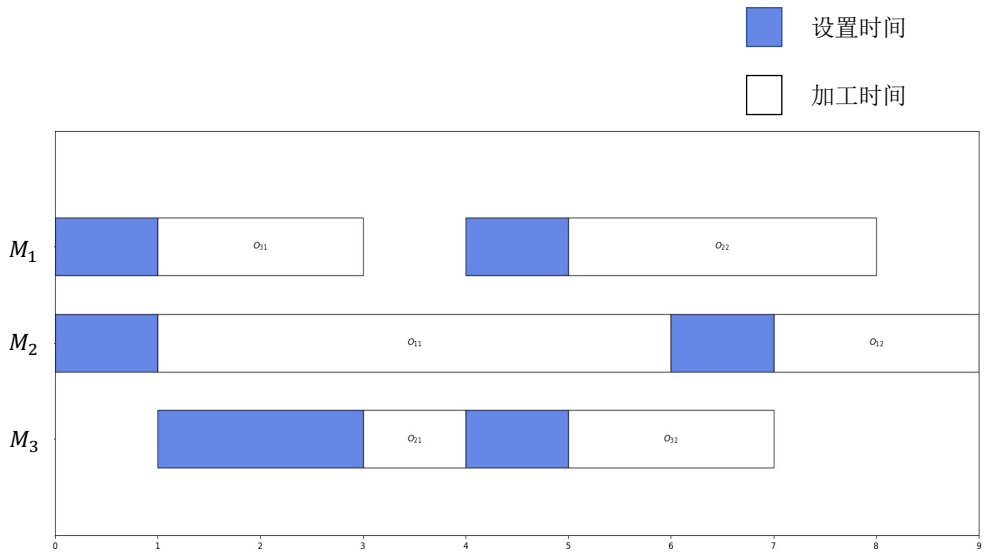

图 2.1: 一个解决方案的甘特图

数量是 2。由于操作员的数量限制，三台机器无法在零时同时更换模具。机器 M_3 只能等操作员更换完机器 M_1 和机器 M_2 的模具之后才能来给它更换模具。如果没有操作员数量限制的话，那么机器 M_3 也可以在零时就更换模具，作业 3 的完工时间可以更短。

2.2 哈里斯鹰优化算法介绍

哈里斯鹰优化算法是 Heidari 等人 [27] 在 2019 年提出的一种新的高效群智能算法。哈里斯鹰主要生活于美国亚利桑那州南部区域。大部分种类的猛禽通常都是独自捕猎的，而哈里斯鹰是与群体一起行动，进行特别的团队捕猎行为。这显得它们如此与众不同，也正因如此，哈里斯鹰群的合作捕猎行为才非常适合模拟成一种群智能优化过程。在捕猎过程中，哈里斯鹰群首先栖息在可能存在猎物的地点的附近区域，用锐利的鹰眼搜寻猎物。发现猎物以后，就对其群起围攻。猎物也不会坐以待毙，会做出各种逃避行为来提高逃生几率。在此过程中，哈里斯鹰群就会根据猎物的逃避行为来相应的采取多种围捕策略，尽量消耗猎物的能量，等到猎物精疲力竭无力躲避时就捕获它们。这些行为可以极大的提升捕猎成功率。HHO 算法也是正是通过模拟这些围捕过程来寻求单目标优化问题的最优解。接下来将介绍 HHO 算法的具体内容，表2.4中展示的是算法中涉及到的符号的含义。图2.2展示了 HHO 算法的各个阶段，算法1为 HHO 算法的总体框架。

2.2.1 搜索阶段

在合作捕猎过程中，哈里斯鹰依靠锐利的双眼来侦查环境和搜寻猎物。但是，广阔的亚利桑那州南部地区遍布沙漠，搜寻猎物比较困难，通常需要花费较长的时间来搜寻和追踪猎物。因此，哈里斯鹰群需要分散开来，扩大搜索面积，来提高搜寻到猎物的概率。鹰群个体随机栖息在一些位置上，主要根据两种策略来选择栖息的位置：

（1）若 $q < q_m$，哈里斯鹰会按照鹰群中其他个体的位置和目标猎物的位置进

表 2.4: HHO 算法中的符号及含义

符号	意义
N	种群数量
t	当前迭代数
T	总迭代次数
D	问题维度
E	逃逸能量
E_0	初始能量
LB, UB	上界，下界
$X(t)$	第 t 次迭代的个体
$X_{rabbit}(t)$	第 t 次迭代中最佳个体
$X_{rand}(t)$	第 t 次迭代中随机选择的个体
$X_m(t)$	第 t 次迭代中个体的平均位置
r_{1-5}, q	(0,1) 范围内的随机数

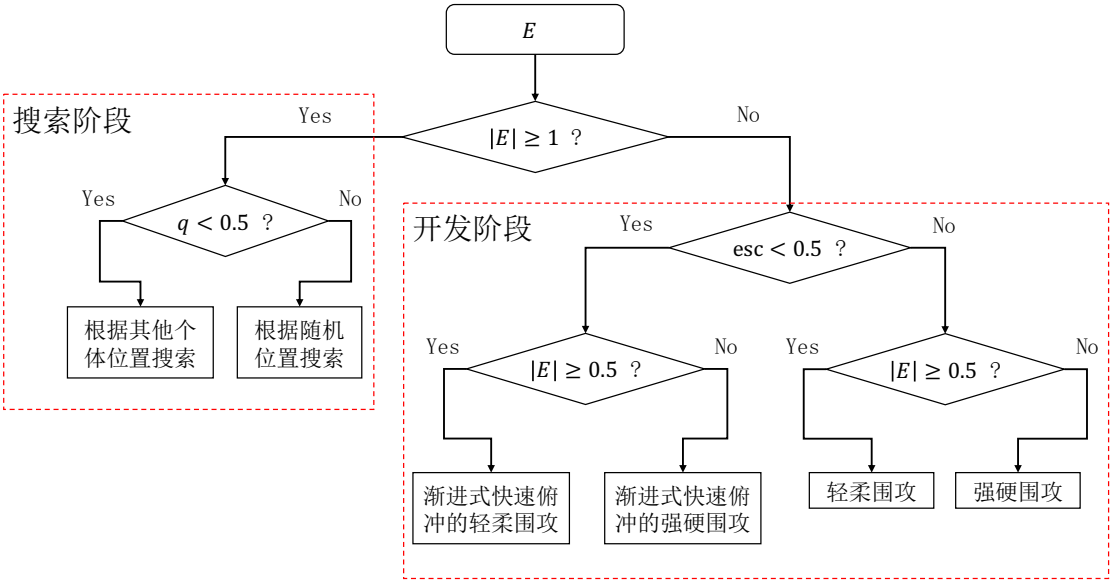

图 2.2: HHO 算法的各个阶段

Algorithm 1 HHO 算法

输入: 种群大小 N 和最多的迭代次数 T.

输出: 最终抓到猎物的位置 X_{rabbit}.

种群初始化 $X_i(i = 1,2,...,N)$。
设置已迭代数 $t \leftarrow 0$。
while $t \leq T$ **do**
  评估各种群个体。
  将种群的最佳个体设置为 X_{rabbit}。
  for 种群个体 X_i **do**
    更新 E_0 和跳跃强度 J。
    用公式（2.4）更新 E。
    if $|E| \geq 1$ **then**
      处于搜索阶段，根据公式（2.2）更新个体位置向量。
    else
      if $esc \geq 0.5$ and $|E| \geq 0.5$ **then**
        采用轻柔围攻策略，根据公式（2.5）更新个体位置向量。
      else if $esc \geq 0.5$ and $|E| < 0.5$ **then**
        采用强硬围攻策略，根据公式（2.6）更新个体位置向量。
      else if $esc < 0.5$ and $|E| \geq 0.5$ **then**
        采用渐进式快速俯冲的轻柔围攻策略，根据公式（2.10）更新个体位置向量。
      else if $esc < 0.5$ and $|E| < 0.5$ **then**
        采用渐进式快速俯冲的强硬围攻策略，根据公式（2.11）更新个体位置向量。
      end if
    end if
  end for
end while

行栖息。

（2）若 $q \geq q_m$，哈里斯鹰会随机在鹰群狩猎领域的某个位置上栖息。

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1 |X_{rand}(t) - 2r_2 X(t)|, & q \geq q_m \\ (X_{rabbit}(t) - X_m(t)) - r_3(LB + r_4(UB - LB)), & q < q_m \end{cases} \tag{2.2}$$

其中 q_m 是哈里斯鹰选择栖息地策略的概率条件，原算法中是固定为 0.5，在后面的章节中，本文提出的 SLHHO 会使用强化学习算法对其进行智能调整。q、r_1、r_2、r_3、r_4 均为 $(0,1)$ 区域内的随机数，UB 是寻优空间的上限，LB 是下限。r_3 是一个比例系数，r_4 的值越接近 1，该策略的随机性就会越高。$X_{rabbit}(t)$ 是猎物的位置向量，也可以理解成是鹰群中最接近目标猎物的个体的位置向量，$X_{rand}(t)$ 是随机在鹰群中选择的个体的位置向量，$X_m(t)$ 是鹰群所有个体的平均位置向量，具体公式如下：

$$X_m(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t) \tag{2.3}$$

2.2.2　从搜索阶段转换至开发阶段

在当前阶段，猎物的逃逸能量会影响到搜索和开发状态的转换。猎物的逃逸能量会在逃脱过程中因为消耗而不断降低，所以可以用如下公式表示：

$$E = 2E_0 \times (1 - \frac{t}{T}) \tag{2.4}$$

其中 E_0 是猎物的初始逃逸能量值，由于不同猎物之间的逃逸能量不同，所以 E_0 是 $[-1,1]$ 之间的一个随机数。当 $E_0 < 0$ 时，猎物由于不断逃跑消耗能量处于萎靡不振的状态。而当 $E_0 \geq 0$ 时，则表示着猎物在不断恢复，有能量继续活动。E 在算法运行过程中是不断下降的。当逃逸能量 $|E| \geq 1$ 时，哈里斯鹰广泛分散，四处搜寻猎物的位置，HHO 处于搜索阶段。而当 $|E| < 1$ 时，HHO 在开发过程中尝试利用解的相邻区域进行局部勘探，处于开发阶段。

2.2.3 开发阶段

在当前的阶段，哈里斯鹰开始围捕之前搜寻到的目标猎物，目标猎物也不会坐以待毙，会想要逃脱当前的区域。因此哈里斯鹰会根据猎物的求生动作而选择不同的追捕行为。HHO 使用了四种行为策略来仿真围捕行为。猎物总是希望能逃离危险的区域，假设猎物的逃脱概率为 esc，当 $esc < 0.5$ 时为成功逃离，$esc \geq 0.5$ 时为逃离失败。而无论猎物怎么跑，哈里斯鹰都不会放弃追捕，它们会根据猎物的逃逸能量来选择轻柔或强硬的方式从不同方向围攻猎物。在捕猎过程中，哈里斯鹰会越来越接近目标猎物，猎物在不断的逃脱过程中会损失越来越多的能量。当猎物精疲力竭时，哈里斯鹰就会发起最后的突袭来捕捉目标猎物。

轻柔围攻

当 $esc \geq 0.5$ 并且 $|E| \geq 0.5$ 时, 猎物的精力充沛，有能力继续尝试逃脱，此时哈里斯鹰发起突袭的话势必会被轻易逃脱。因此，哈里斯鹰会不断徘徊在猎物附近，消耗其能量，让猎物精疲力竭，进而突袭。而且猎物也会不断做一些随机跳跃的假动作来迷惑追击者。具体公式如下：

$$X(t+1) = \Delta X(t) - E|JX_{rabbit}(t) - X(t)| \tag{2.5}$$

其中 $\Delta X(t) = X_{rabbit}(t) - X(t)$，$J$ 是猎物在逃脱过程中的随机跳跃强度，$J = 2(1 - r_5)$。J 会在每次更新中随机产生来仿真猎物的动作。

强硬围攻

当 $esc \geq 0.5$ 并且 $|E| < 0.5$ 时，猎物已经精疲力竭，逃逸能量消耗殆尽，毫无反抗能力，哈里斯鹰可以强硬的包围猎物，直接发起突袭。当前情况下，位置的更新可以如下表示：

$$X(t+1) = X_{rabbit}(t) - E|\Delta X(t)| \tag{2.6}$$

渐进式快速俯冲的轻柔围攻

当 $esc < 0.5$ 并且 $|E| \geq 0.5$ 时，猎物有机会逃脱，且能量充沛。针对这种情形，哈里斯鹰会在发起最后突袭前构成轻柔的包围圈。为了更好的模拟猎物逃逸的行为，HHO 算法使用了 LF 的概念 [45]。LF 用于模拟猎物在逃跑过程中做的一些欺骗动作以及哈里斯鹰在猎物周围采取的各种试探攻击、假装突袭的行为。所以 HHO 在该阶段会考虑基于 LF 的行为模式。哈里斯鹰会根据猎物的行为反应逐步选择更好的俯冲方式来接近猎物。首先会根据以下规则评估下一步的行动：

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X(t)| \tag{2.7}$$

如果这种行动并没有得到更好的结果，哈里斯鹰就会采取基于 LF 模式的行为，比如不规则、突然快速俯冲等。该策略模式如下表示：

$$Z = Y + S \times LF(D) \tag{2.8}$$

其中 S 是 $1 \times D$ 的随机生成的向量，D 是空间维度，$LF(D)$ 是 LF 函数：

$$LF(x) = 0.01 \times \frac{u \times \sigma}{(|v|)^{\frac{1}{\beta}}}, \sigma = \left(\frac{\Gamma(1+\beta) \times \sin(\frac{\pi\beta}{2})}{\Gamma(\frac{1+\beta}{2}) \times \beta \times 2^{(\frac{\beta-1}{2})}} \right)^{\frac{1}{\beta}} \tag{2.9}$$

其中 u, v 是 $(0,1)$ 范围内的随机数，β 是一个常量，这里设置为 1.5。基于以上情况，该围攻方式可以总结为：

$$X(t+1) = \begin{cases} Y, & \text{if } F(Y) < F(X(t)) \\ Z, & \text{if } F(Z) < F(X(t)) \end{cases} \tag{2.10}$$

F 是适应度函数，此外如果 $F(Y)$ 和 $F(Z)$ 的效果都差于 $F(X(t))$，那它们就保持不变。而如果效果都好于 $F(X(t))$ 的话，就选择效果最好的那个。

渐进式快速俯冲的强硬围攻

当 $esc < 0.5$ 并且 $|E| < 0.5$，猎物仍有可能逃脱，但是能量损失殆尽。于是哈里斯鹰在开始攻击前先围成一个强硬的包围圈，防止猎物逃脱，再拉近与猎物的距

离，发起最后的突袭，保证万无一失。具体规则如下：

$$X(t+1)=\begin{cases} Y_m, & \text{if } F(Y_m)<F(X(t)) \\ Z_m, & \text{if } F(Z_m)<F(X(t)) \end{cases} \tag{2.11}$$

其中 Y_m 定义为:

$$Y_m = X_{rabbit}(t) - E|JX_{rabbit}(t) - X_m(t)| \tag{2.12}$$

而 Z_m 表示为：

$$Z_m = Y_m + S \times LF(D) \tag{2.13}$$

2.3 强化学习算法介绍

强化学习算法是一种人工智能学习算法，以收益最大化为目的。RL 执行动作来影响环境状态，然后根据环境返回的反馈而不是通过测试用例来评估动作的效果，再根据这种效果来训练代理去选择更好的动作，从而从环境中获得最大累计奖励 [46]。在 RL 中，代理不断与复杂的环境交互，根据环境的反馈来优化动作选择，最终实现最佳决策 [47]。RL 的模型框架如图2.3所示，其中 RL 代理是 RL 算法的虚拟对象。在 t 时，代理获取 t 时的环境状态 s_t，依靠动作选择策略来选择动作 a_t。接着在 $t+1$ 时刻，环境在执行动作 a_t 后过渡到状态 s_{t+1}，并从环境中获得奖励 r_t，这将使得代理更新动作选择并选择下一个会让结果更好的动作 a_{t+1}，更合适的动作会让代理找到最佳策略 π_t 从而获得最大的累计奖励。策略公式如下：

$$\pi^* = arg\max_{\pi} E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k} \mid s_t\right\}, \forall s_t \in S, \forall t \geq 0 \tag{2.14}$$

其中 γ 表示折扣率,k 表示未来时刻, t 是当前时刻, s_t 是 t 时刻的状态，S 表示状态集合.

Q-learning 算法和 SARSA 算法是两种典型的基于 Q 值的 RL 算法，它们的目的都是为了评估最佳策略的 Q 值。Q 值是动作选择合理性的特征表示，根据环境状态和动作的反馈进行更新。Q 值存储在用于记录代理学习经验的 Q 值表中，初始 Q 值表是算法生成的一个零值矩阵，行数是状态集合的数目，列数是动作集合

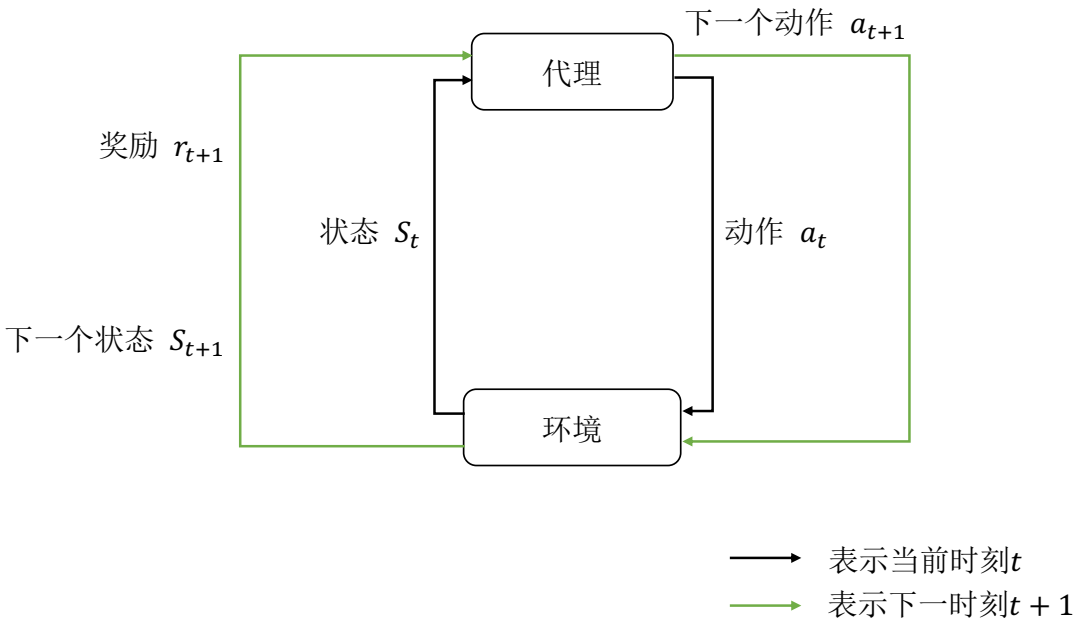

图 2.3: RL 模型框架

的数目。环境状态的变化会引起 Q 值表中 Q 值的变化，而 Q 值的变化会影响动作的选择。单步 Q-learning 算法的 Q 值更新计算方式如下：

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1})) \tag{2.15}$$

其中 a_t 是在 t 时刻选择的动作，$Q(s_t, a_t)$ 表示的是在 s_t 状态时执行 a_t 动作可以得到的 Q 值。α 是学习率，r_{t+1} 是在 s_t 状态时执行 a_t 动作获取到的收益，$Q(s_{t+1}, a_{t+1})$ 表示在通过贪心策略选择 a_{t+1} 动作并在 s_{t+1} 状态时执行之后预期的 Q 值。单步 SARSA 算法的 Q 值更新计算方式如下：

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})) \tag{2.16}$$

这两者不同的地方主要在于更新 Q 值的方式不同。Q-learning 算法是假设先按照动作选择策略来选择动作并执行得到未来状态，然后选取未来状态下预期 Q 值最大的那个动作来真正执行，最后更新 Q 值。而 SARSA 算法是直接按照动作选择策略来选择动作并执行，然后再根据得到的环境状态，更新 Q 值。这两个算法各有千秋，Q-learning 算法在每个状态都是选择预期 Q 值最大的那个动作来更新 Q 值，这

样可以得到更优的解。但是它需要长时间的学习，在算法的早期阶段，它的学习效果和质量要差于 SARSA 算法。而 SARSA 算法虽然有更快的收敛性和较高的学习性能，但是很容易陷入局部最优，难以获得最优解。

2.4 本章小结

HHO 算法源于对哈里斯鹰群合作捕猎过程的模拟，将哈里斯鹰在捕猎过程中针对猎物的反应而采取的不同行为演化为算法的多个策略，也列举了算法中用到的数学公式，并一一解释其含义。HHO 算法的种群个体位置向量是由一组实数组成的，在后续的章节中，会将其转换成离散的机器分配和工序顺序向量，才能用来解决 FJSP。接着简要介绍了强化学习算法的相关概念，还特别介绍了 Q-learning 算法和 SARSA 算法这两个基于 Q 值的强化学习算法。在展示了这两个 RL 算法的 Q 值更新公式的同时，对两者进行了对比，阐述了两者的优缺点。

第三章　哈里斯鹰优化算法实现

3.1　问题分析

Heidari 等人 [27] 在 2019 年提出了 HHO 算法用来解决不同的优化任务，并且使用了 29 个无约束基准问题验证了 HHO 出色的性能。这些基准问题都是连续空间上的数值优化问题，HHO 算法中哈里斯鹰个体位置向量是一组用一定范围内的实数构成的向量。算法的每次迭代运行，都会得到一个代理猎物的最佳位置向量，然后经过转换函数转换成优化问题的解。但是在 FJSP 中，无论是机器分配，还是工序顺序，都是用整数表示的离散形式的向量。这也意味着如果想要得到 FJSP 的解决方案，就必须要将 HHO 算法运行得到的实数表示的个体位置向量使用某种方法转换为成为离散的整数向量，再根据向量中蕴含的机器分配和工序顺序信息，得到一个可行的 FJSP 的解决方案。鉴于集成法可以比分层法拥有更卓越性能的优点，接下来本文将以集成法的形式使用 HHO 算法来解决带有序列相关设置时间和资源约束的 FJSP。

3.2　两段向量编码

HHO 算法中哈里斯鹰群个体的位置向量用一个 $2l$ 维的实数表示的向量来表达，$X=\{x_1,x_2,...,x_l,x_{l+1},...,x_{2l}\}$。维度 $2l$ 必须根据问题满足一定的约束。其中 l 是要求解的 FJSP 中所有工序的数量，向量的元素 x 是在 $[-\epsilon,\epsilon]$ 之间的实数。位置向量的前半部分 $X^1=\{x_1,x_2,...,x_l\}$ 描述了每道工序的机器分配信息，而位置向量的后半部分 $X^2=\{x_{l+1},x_{l+2},...,x_{2l}\}$ 则描述了所有机器上的工序顺序信息。

本文的两段向量编码由两个向量组成：机器分配向量和工序顺序向量，分别蕴含了 FJSP 的机器分配信息和工序顺序信息。机器分配向量 $MA=\{y_1,y_2,...,y_l\}$，是一个由 l 个整数构成的向量，其中 l 是 FJSP 中的工序总数。在向量中，$y_j,1\le j\le l$ 表示工序 j 选择其可选机器集中的第 y_j 台机器。图3.1展示了一个机器分配向量例

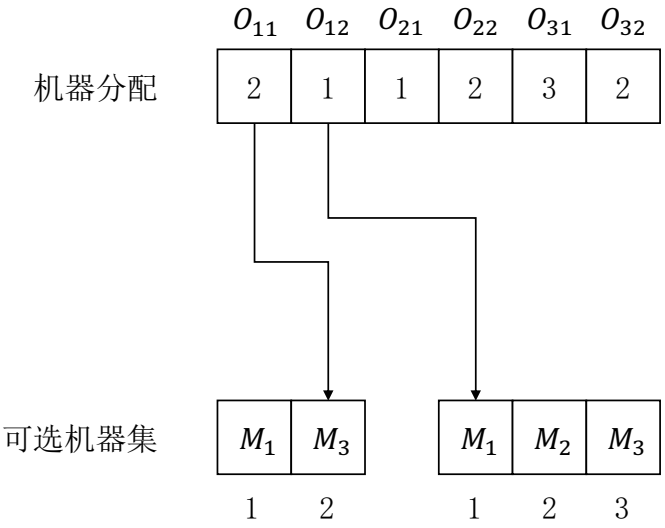

图 3.1: 机器分配向量的实例

子。图中 $y_1 = 2$ 表示工序 O_{11} 选择了它可以选择的诸多机器中的第二台，即 M_3。

至于工序顺序向量，$OS = \{v_1, v_2, ..., v_l\}$，最简单的方法之一就是给每个工序编号，然后直接使用编号排列组合。但是由于同一作业的工序之间存在优先约束，这种方法并不能总是为 FJSP 提供一个可行的调度方案。因此，本文使用基于工序的表示方式，使用相同的编号命名同一作业的所有工序，然后根据给定的编码序列中的出现顺序来解释它们。与使用编号排列组合不同的是，这种方法始终可以得到一个可行的调度方案。首先将同一作业的工序都编号成该作业的序号，作业 J_i 的序号 i 在向量中出现 n_i 次，就表示是该作业的第 n_i 道工序，$O_{i,1}, O_{i,2}, ..., O_{i,n_i}$。一个可能的工序顺序向量如图3.2所示，并且可以转换成工序排序的唯一列表：$O_{21} \succ O_{11} \succ O_{22} \succ O_{31} \succ O_{12} \succ O_{32}$。工序 O_{21} 具有最高优先级并且首先被处理，然后是工序 O_{11}，依此类推。

接下来就要将给定的位置向量 X 转换为机器分配向量 MA 和工序顺序向量 OS，将位置向量的前半部分 X^1 转换为 MA，后半部分 X^2 转换为 OS。对于位置向量前半部分的转换，设 s_j 表示工序 j 的可选机器集的大小，需要将实数 $x_j \in [-\epsilon, \epsilon]$ 映射到整数 $y_j \in [1, ..., s_j]$。具体过程是：首先将 x_j 通过线性变换转换成属于 $[1, s_j]$

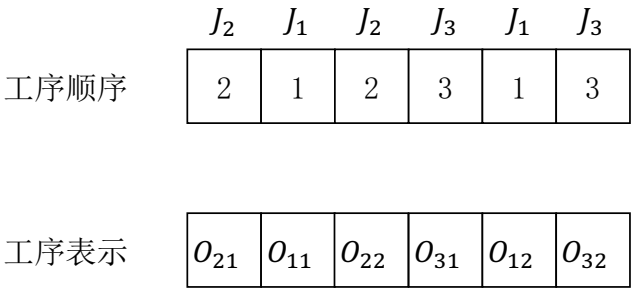

图 3.2: 工序顺序向量的实例

的实数，然后给 y_j 赋予转换后的实数最接近的整数值。具体公式如下:

$$y_j = \text{round}(\frac{1}{2\epsilon}(s_j - 1)(x_j + \epsilon) + 1),\ \ 1 \le j \le l \tag{3.1}$$

其中 round(x) 是将 x 设置为最靠近的整数的函数。在特定情况下 $s_j = 1$，无论 x_j 的值是多少，y_j 始终等于 1。

在工序顺序向量的转换中,首先将位置向量 X^2 的元素按工序顺序 $O_{i1}, O_{i2}, ..., O_{ij}$ 编号，接着将其按照元素值的大小降序排序，然后将每个工序的编号替换成所属的作业编号。一个可能的工序顺序向量转换例子如图3.3所示。这样将位置向量转换成了机器分配向量和工序顺序向量以后，就可以根据这两个向量得到可行的调度方案了。

3.3　两段向量解码

调度问题的调度方式一般分为三大类：主动调度方式、半主动调度方式和非延迟调度方式 [48]。在考虑完工时间的情况下，使用主动调度的方式可以找到最优的调度 [49]。为了说明主动调度，这里先介绍一下左移的概念。左移是一种将工序移动到一个更早的开始处理时间并使得调度尽可能紧凑的策略。如果存在某些工序可以在不改变机器上的工序处理顺序的情况下更早的开始处理，则称之为局部左移。如果存在某些工序可以更早的开始处理，但可能会改变机器上的工序处理顺序，则称之为全局左移。如果一种调度不存在局部左移，则称之为半主动调度。如果一种调度不存在全局左移，则称之为主动调度。主动调度的解空间要小于半

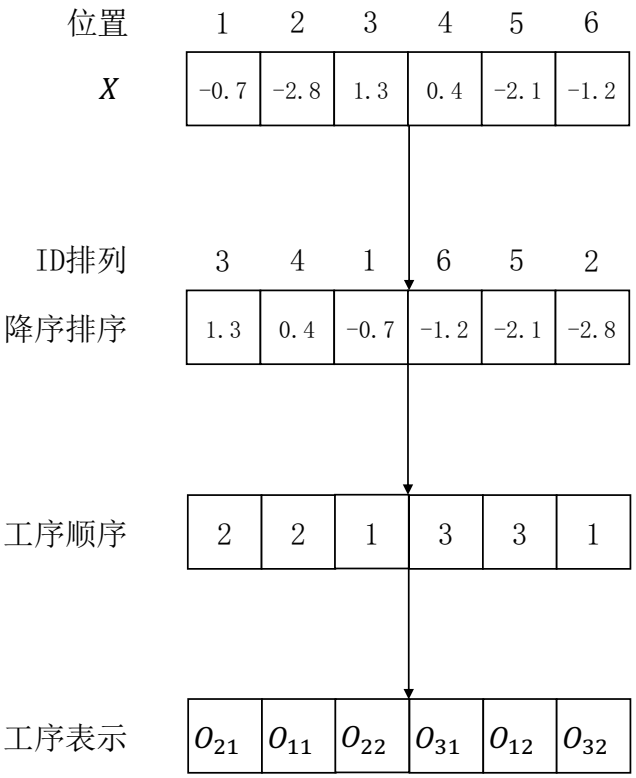

图 3.3: 将位置矢量 X^2 转换成工序顺序向量

主动空间，因此，可以将两段向量解码成主动调度，从而减少搜索空间，而且可以得到质量较高的调度方案。

为了将两段向量解码成主动调度，当一道工序 O_{ij} 被分配在机器 M_k 上处理且处理时间为 p_{ijk} 先搜索机器 M_k 上最早可用的满足条件的空闲时间间隔分配给它。如果不存在这样的时间间隔，则将该工序安排在该机器处理任务的末尾。机器 M_k 的空闲时间间隔表示为 $[S_k, E_k]$，$C_{i,j}$ 表示工序 O_{ij} 的完工时间。若该时间间隔可分配给 O_{ij}，那么就需要满足以下条件：

$$\begin{cases} S_k + s_{ghijk} + p_{ijk} \le E_k, & \text{if } j = 1 \\ \max\{S_k, C_{i,j-1}\} + s_{ghijk} + p_{ijk} \le E_k, & \text{if } j \ge 2 \end{cases} \tag{3.2}$$

如果存在可以分配的空闲时间间隔，那么工序 O_{ij} 的开始处理时间 b_{ij} 定义如下：

$$b_{ij} = \begin{cases} S_k + s_{ghijk}, & \text{if } j = 1 \\ \max\{S_k, C_{i,j-1}\} + s_{ghijk}, & \text{if } j \ge 2 \end{cases} \tag{3.3}$$

如果不存在的话，令 ET_k 表示机器 M_k 上当前任务队列的结束时间，则开始处理时间 b_{ij} 定义如下：

$$b_{ij}=\begin{cases} ET_k + s_{ghijk}, & \text{if } j=1 \\ \max\{ET_k, C_{i,j-1}\} + s_{ghijk}, & \text{if } j \geq 2 \end{cases} \tag{3.4}$$

在所有的工序都被安排以后，完工时间就可以通过 $C_{max} = \max\limits_{1\leq k\leq m}(ET_k)$ 计算得到。MA 和 OS 具体的解码步骤在算法2中展示。

Algorithm 2 MA 和 OS 解码过程

1: 根据工序顺序向量 OS 得到工序处理顺序 $\{O_1, O_2, ..., O_l\}$
2: 根据机器分配向量 MA 得到每道工序 O_i 的三元组信息 (O_i, M_k, p_{ik})
3: **for** $i = 1$ to l **do**
4:　获取工序 O_i 的三元组信息。
5:　在工序 O_i 被分配的机器上搜索最早的可使用的空闲时间间隔。
6:　**if** 存在可使用的空闲时间间隔 **then**
7:　　工序 O_i 就分配到该时间间隔，根据公式（3.3）设置开始处理时间 b_i。
8:　**else**
9:　　将工序 O_i 分配到机器当前任务队列末尾，根据公式（3.4）设置开始处理时间 b_i。
10:　**end if**
11: **end for**
12: **return** 主动调度方案和最大完工时间

3.4　资源约束

在本文提出的问题中，机器在连续处理两道工序之间需要操作员花费一定的时间进行更换模具，而且操作员的数量是有限制的。考虑到人力成本原因，操作员的数量一般是少于机器数量的。所以机器有可能遇到操作员无空闲的情况，从而陷入等待停滞状态，导致产品交付延迟。为了提升车间生产效率，避免产品延迟交付，在 MA 和 OS 解码过程中应尽量减少机器等待操作员的情况。

在上一节中，通过算法2产生了一个没有考虑操作员数量限制这种资源约束的主动调度方案。在这个调度方案中，如果工序分配的机器上存在最早可使用的空

闲时间间隔，就将工序安排在该时间间隔。但如果考虑操作员的情况，那么该时间间隔还要满足在此时间间隔中存在空闲的操作员。如果在该时间间隔中存在空闲的操作员，那么该工序就可以直接安排在该时间间隔中。如果在该时间间隔中不存在空闲的操作员，那最差的情况就是停滞等待，等待任意一个操作员空闲下来，这样肯定会延长工序的完成时间，甚至有可能影响到该工序所属作业的完工时间。

那么就需要寻找一种方法来尽量避免这种情况。假设工序 O_{ij} 一开始分配给机器 M_k 处理，机器 M_k 中存在可使用的空闲时间间隔，但在此时间间隔中不存在空闲的操作员。而在工序 O_{ij} 的可选机器集 M_{ij} 中除了机器 M_k 外还存在另一台机器 M_x，机器 M_x 在处理其他作业的某道工序 O_{gh} 之前存在一段空闲的时间间隔，可能是由于工序 O_{gh} 在等待工序 $O_{g(h-1)}$ 的完成。这段空闲时间间隔满足工序 O_{ij} 的处理条件，而且也存在空闲的操作员。那么将工序 O_{ij} 分配到机器 M_x，就避免了机器停滞等待情形，也没有影响其他工序，有可能缩短该工序所属作业的完工时间，甚至有可能得到质量更好的解决方案。基于以上的这种思想，本文设计了一种解码方法来尽量减少机器等待操作员的情况，具体步骤在算法3中描述。

3.5 实验及结论

为了测试 HHO 算法求解带有序列相关设置时间和资源约束的 FJSP 的性能，该算法用 Python 语言实现，并在具有 2.6 GHz、英特尔 i5 处理器和 8 GB RAM 运行内存的 PC 主机上运行。针对本文所提出问题的特性，根据一些经典数据集创建了一些基准实例。为了评估 HHO 算法的性能，本文会将 HHO 算法在这些基准实例上运行的实验结果与其他多个有效算法进行比较。

3.5.1 数据集

由于没有专门用于带有序列相关设置时间和资源约束的 FJSP 的基准实例，本文就基于两个在 FJSP 研究领域中被广泛使用的经典数据集来构建专门的基准实例来测试 HHO 算法的性能。这两个经典数据集分别为：

Algorithm 3 避免资源约束

获取工序 O_i 的三元组信息 (O_i, M_k, p_{ik})。
在工序 O_i 被分配的机器上搜索最早的可使用的空闲时间间隔。
if 存在可使用的空闲时间间隔 **then**
    寻找空闲的操作员。
    if 存在空闲的操作员 **then**
        工序 O_i 就分配到该时间间隔，更新当前操作员的工作时刻表和当前机器 M_k 的工作时刻表。
    else
        将工序 O_i 的可选机器集中剩下的机器按设置时间和处理时间之和升序排序 $\{M_1, M_2, ..., M_r\}$。
        for $h = 1$ to r **do**
            在当前机器 M_h 上搜索最早的可使用的空闲时间间隔和在此期间空闲的操作员
            if 存在空闲时间间隔和空闲操作员 **then**
                将工序 O_i 分配给该时间间隔，更新机器 M_h 的工作时刻表和当前操作员的工作时刻表，退出循环。
            else
                继续循环。
            end if
        end for
        if 工序 O_i 没有被分配给剩余机器中的任意一台机器 **then**
            将工序 O_i 分配到机器 M_k 当前任务末尾，等待第一个空闲的操作员，更新它们的工作时刻表。
        end if
    end if
else
    将工序 O_i 分配到机器 M_k 当前任务末尾，寻找空闲的操作员。
    if 存在空闲的操作员 **then**
        更新机器 M_k 和操作员的工作时刻表。
    else
        等待第一个空闲的操作员，更新机器 M_k 和操作员的工作时刻表。
    end if
end if

（1）Kacem 数据集：Kacem 数据集由 Kacem 等人 [50] 提出，这个数据集里的基准实例主要有 8×8、10×10 和 15×10 三种规模：

- 8×8 表示该实例一共有 8 个作业，每个作业包含了 27 道工序，可以使用的机器的数量为 8 台。
- 10×10 表示该实例由 10 个作业组成，每个作业包含了 30 道工序，可以使用的机器的数量为 10 台。
- 15×10 表示该实例由 15 个作业组成，每个作业包含了 56 道工序成，可以使用的机器的数量为 10 台。

（2）Brandimarte 数据集：Brandimarte 数据集由 Brandimarte 等人 [34] 提出，该数据集有 10 个实例。这些实例的作业数量在 10 到 20 之间，可使用的机器数量在 4 到 15 台之间，所有作业的工序数量在 55 到 240 之间。

本文基于 Kacem 数据集和 Brandimarte 数据集构建了 12 个基准实例。其中 $01-02$ 是基于 Kacem 数据集产生的小规模的基准实例，$03-12$ 是基于 Brandimarte 数据集产生的较大规模的基准实例。在每个实例中，每台机器的设置时间为 $[1,...,5]$ 中的随机整数。并且为每个实例根据它们的数据规模设置了一个合适的操作员数量。

3.5.2 参数设置

在 HHO 算法中，种群大小数量 N 为 50，总的迭代次数 T 为 100 次。哈里斯鹰选择栖息地策略概率条件 q_m 固定为 0.5, 目的是哈里斯鹰有均等的机会去全局搜寻猎物或局部围攻猎物。由于算法的随机性，每个基准实例都会独立运行 30 次。为了比较算法的整体性能，记录以下四个指标来衡量实验结果：

(1) Best：30 次运行实验的最佳完工时间。

(2) Avg：30 次运行实验的平均完工时间。

(3) SD：30 次运行实验得到的标准差。

表 3.1: 实例 01-12 的实验结果（粗体数字表示四种算法中的最佳结果）

		HHO			PSO-SA			DABC			IJaya		
Instance	$J\times M\times L$	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
01	$4\times5\times2$	**17**	**17.9**	2.6	19	24.4	2.2	20	26	2.5	22	25.3	1.5
02	$8\times8\times2$	**27**	**31.5**	3.3	30	34.1	4.4	30	36	4.5	29	34.3	3.8
03	$10\times6\times3$	**73**	**78.6**	6.1	82	94.9	6.4	89	99.3	4.9	88	97.9	4.4
04	$10\times6\times3$	**59**	**64.2**	6.8	74	86	5.7	81	91.2	4.7	81	92.3	4.9
05	$15\times8\times3$	**290**	**319.5**	19	343	377.3	24.3	373	418.7	24.4	386	432.7	27.7
06	$15\times8\times3$	**122**	**130.5**	7.1	134	143.5	7.3	158	171.6	7.3	142	158.8	8.5
07	$15\times4\times3$	**265**	**274**	11.7	279	309.5	13.6	305	336.7	11.6	314	333	10.9
08	$10\times15\times3$	**165**	**175**	13	194	226.1	13.3	251	277.9	12.6	251	273.2	10
09	$20\times5\times4$	**232**	**242.5**	14.5	276	315.1	22.2	319	350.2	15.2	304	328	15.5
10	$20\times10\times4$	**670**	**701.8**	21.9	706	744.3	18.6	757	815.6	20.6	792	825	18.9
11	$20\times10\times4$	**512**	**536.8**	27.8	581	672.8	30.2	663	732.6	25	632	719.4	33.4
12	$20\times15\times4$	**410**	**433.9**	24.9	491	543.5	23.1	585	641.5	15.9	609	647.6	17.5

(4) Time：30 次运行实验的平均时间（s）。

3.5.3　实验结果

为了评估 HHO 算法的性能，本节在 12 个基准实例上将 HHO 算法的实验结果与重写的 PSO-SA [51]、DABC [13] 和 IJaya [16] 进行对比。实验结果在表3.1中展示，其中 $J\times M\times L$ 分别表示作业的数目、机器的数目和操作员的数目（例如，$4\times3\times2$ 表示这个实例有 4 个作业、3 台机器和 2 个操作员）。

从表3.1中可以看出，与 PSO-SA、DABC 和 IJaya 相比，HHO 算法具有出色的表现。在所有的 12 个基准实例中，HHO 算法均可以获得质量最佳的解决方案，而

表 3.2: 算法运行时间

Instance	$J \times M \times L$	HHO Time(s)	PSO-SA Time(s)	DABC Time(s)	IJaya Time(s)
01	$4 \times 5 \times 2$	3.3	0.4	0.4	0.7
02	$8 \times 8 \times 2$	7.6	0.9	0.7	1.4
03	$10 \times 6 \times 3$	9.8	1.9	1.8	3.1
04	$10 \times 6 \times 3$	12.2	2	1.9	3.1
05	$15 \times 8 \times 3$	32.8	5.2	6.7	8.3
06	$15 \times 8 \times 3$	16.9	2.9	2.9	5
07	$15 \times 4 \times 3$	20.1	5.2	3.4	8.3
08	$10 \times 15 \times 3$	30.1	6.3	4.3	10.1
09	$20 \times 5 \times 4$	22.3	4.6	3.3	6.1
10	$20 \times 10 \times 4$	44.5	11.3	7.5	15.2
11	$20 \times 10 \times 4$	49.4	11.9	8.4	15.3
12	$20 \times 15 \times 4$	52	10.5	8.1	14.2

且实例规模越大，质量提升越多。这得益于算法中使用的渐进式选择策略，它帮助种群个体逐步提升自己的适应度值，在迭代过程中通过选择更好的位置向量来提高解的质量。并且算法在进行局部搜索时使用不同的具有短长度跳跃的 LF 模式来增强 HHO 的开发能力。在所有实例中，解决方案的平均质量也是最高的，但没有最佳质量提升明显，在 SD 上表现也不突出，算法的稳定性还有待提升。HHO 算法中哈里斯鹰个体平均位置的多样化机制虽然可以促进 HHO 在初始迭代中的探索行为，但也使得初始种群之间适应度值差距颇大。

表3.2是算法的运行时间。从表中看出，HHO 虽然可以获得质量较高的解决方案，但算法的运行时间也有所增长。与 PSO-SA、DABC 和 IJaya 相比，HHO 的运行时间不占优势。为了避免机器等待操作员的情况而专门设计的解码方法虽然有效提升了解决方案的质量，但也额外增加了计算开销。如果为了追求解决方案的质量，而不考虑时间消耗，那么这种代价是值得的。

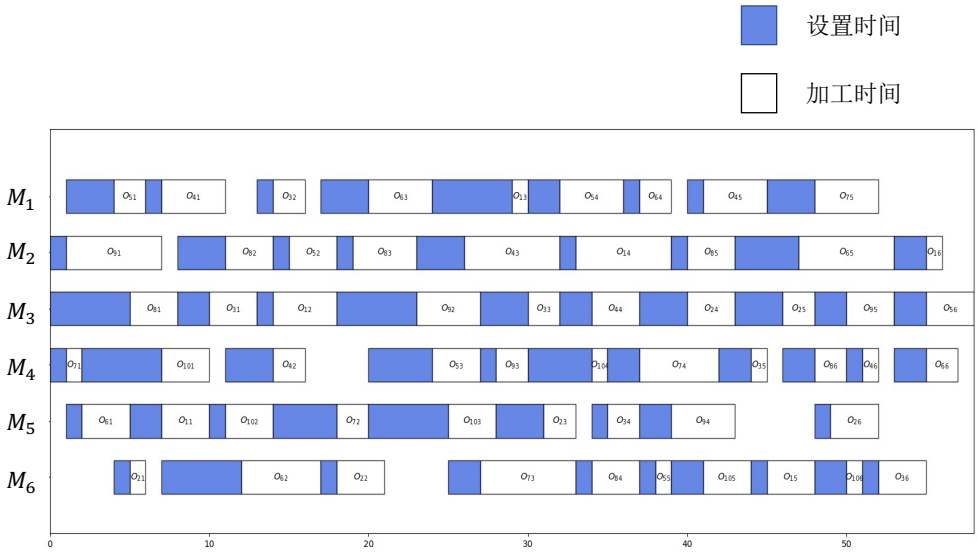

图 3.4: 实例 04 解决方案的甘特图

为了展示 HHO 算法的执行效果，本文绘制了 HHO 算法在实例 04 上得到的一个可行解决方案的甘特图，如图3.4所示。该实例有 10 个作业，每个作业有 5 或 6 道工序，同时可以给机器更换模具的操作员数量是 3，蓝色矩形表示设置时间，白色矩形表示处理时间。

3.6　本章小结

本章使用 Python 语言实现了离散版本的 HHO 算法，使其可以用于解决带有序列相关设置时间和资源约束的 FJSP。首先将 HHO 算法中得到的种群个体位置向量转换成机器分配向量和工序顺序向量，再解码成可行的调度时间表。在解码过程中还要考虑操作员数量限制这种资源约束，为此专门设计一种解码方法来尽量避免机器因等待操作员而空闲的情形。

由于没有专门针对本文提出问题的数据集，本文基于两个 FJSP 经典数据集生成了 12 个基准实例来测试 HHO 算法的性能。本文还将实验产生的结果与其他三种有效算法进行对比，实验结果表示 HHO 算法可以有效的解决本文所提出的问题。相比于其他算法，HHO 算法明显提升了解决方案的质量，并且随着问题规模

越大，提升越明显。但这种性能提升的代价是运行时间的增加，HHO 算法的运行时间长于其他算法，而且在算法稳定性方面也没有很明显的优势。本文还绘制了 HHO 算法在一个基准实例上得到的解决方案的甘特图，来展示 HHO 算法的效果。

第四章　自学习哈里斯鹰优化算法实现

4.1　问题描述

在上一章中，本文使用 HHO 算法来解决带有序列相关设置时间和资源约束的 FJSP，得到了较高质量的解决方案。但这个算法在设计的时候就是比较简单的，几乎没有额外的探索和开发机制，这样就存在可以优化的空间。例如在 HHO 算法的搜索阶段，哈里斯鹰群个体会使用两种策略来改变自己的栖息位置从而搜寻猎物。至于使用哪种策略就取决于参数 q，当 $q < q_m$ 时，哈里斯鹰会按照鹰群中其他个体的位置和目标猎物的位置进行改变。而当 $q \geq q_m$ 时，哈里斯鹰会随机地栖息在一个鹰群活动范围内的位置。

在 HHO 原始算法中，q_m 固定设置为 0.5，这样在算法的迭代过程中可以有均等的机会去随机搜索或根据其他个体位置搜索。但这种方式没有充分的利用算法的搜索能力和开发能力，存在提升的可能性。比如在算法的某次迭代过程中，种群的质量已经较高了，这时候就应该强化算法的邻域开发能力，从而追求质量更好的种群。而当种群质量还比较低时，强化算法的邻域开发能力意义不大，即使得到质量更高的种群，也有可能比不上从质量较高种群中生成的种群。此时应该提高算法的随机搜索能力，搜寻质量更好的初始种群。所以在 HHO 算法的运行过程中，q_m 不应该是个固定值，而是应该根据种群的整体状态（不仅仅是当前的状态，还应该包括过去的状态，甚至包括未来的状态）来变化，从而强化算法的邻域开发能力或者随机搜索能力，以得到质量更好的解决方案。

4.2　自学习哈里斯鹰优化算法模型

哈里斯鹰的初始栖息位置决定了哈里斯鹰个体是随机搜索猎物还是根据其他个体位置进行搜素。栖息策略的选择取决于 q_m。如果 q_m 偏大，则哈里斯鹰个体去趋向于在根据其他个体位置搜索。而如果 q_m 偏小，哈里斯鹰个体就更趋向于在种

群活动范围内随机搜索。在 HHO 算法中，q_m 是固定设置为 0.5，目的是为了个体有均等机会去随机搜索或者根据其他个体位置进行搜索。这种方式是存在缺陷的，它忽视了算法在迭代过程中对随机搜索能力和根据其他个体位置搜索能力的需求变化。但如果只根据当前种群适应度值来设置 q_m，也无法保证其适用性和可靠性。因为这种方式没有考虑前几代或未来几代的可能的种群影响。本节设计了一种自学习哈里斯鹰优化算法（SLHHO），使用 RL 来控制 HHO 算法的关键参数。RL 不仅可以考虑当前和过去种群的状态，还能对未来种群的状态做适当的预测，可以设置更有效果的参数，从而增强算法的性能。

4.2.1　组合模型

RL 的基本组成部分是价值函数、奖励、动作选择策略和环境。在将 RL 和 HHO 算法组合时必须充分考虑这些基本组成部分。在上述前提下本文设计了如图4.1所示的方案，其中 RL 和 HHO 算法的组合模型分为学习模块、强化过程和环境。HHO 可以看成是一个环境，每次迭代都会改变它的状态从 s_t 到 s_{t+1}。学习模块由 RL 代理和 Q 值表组成，其中 RL 代理是 RL 算法的虚拟对象。

组合模型的执行分为四个阶段。第一步，代理获取 HHO 在迭代过程中时刻 t 时的状态 s_t(HHO)。第二步，根据确定的动作选择策略选择并执行对应的动作 a_t(HHO)。在 SLHHO 中，代理所执行的动作就是调整和更新 q_m，HHO 将使用更新后的新的 q_m。第三步，再次执行 HHO 算法，将 HHO 的状态转化为 s_{t+1}(HHO) 并将状态结果反馈给代理。最后，代理将会执行对应的动作 a_{t+1}，并且学习过程也会被代理根据之前经历的状态和动作以及收到的反馈记录下来。在这过程中，代理会使用价值函数来评估环境状态，然后根据对应的状态和动作在 Q 值表中更新相应的 Q 值。

经过 k 次迭代以后，强化过程就被激活，用于选择 q_m 的动作选择就可以根据之前的学习经验来进行优化。如果获得的奖励是正向的，那么动作 a_t(HHO) 的选择就会被增强。但如果获得是负数的奖励，那动作 a_t(HHO) 的选择就会被削弱。

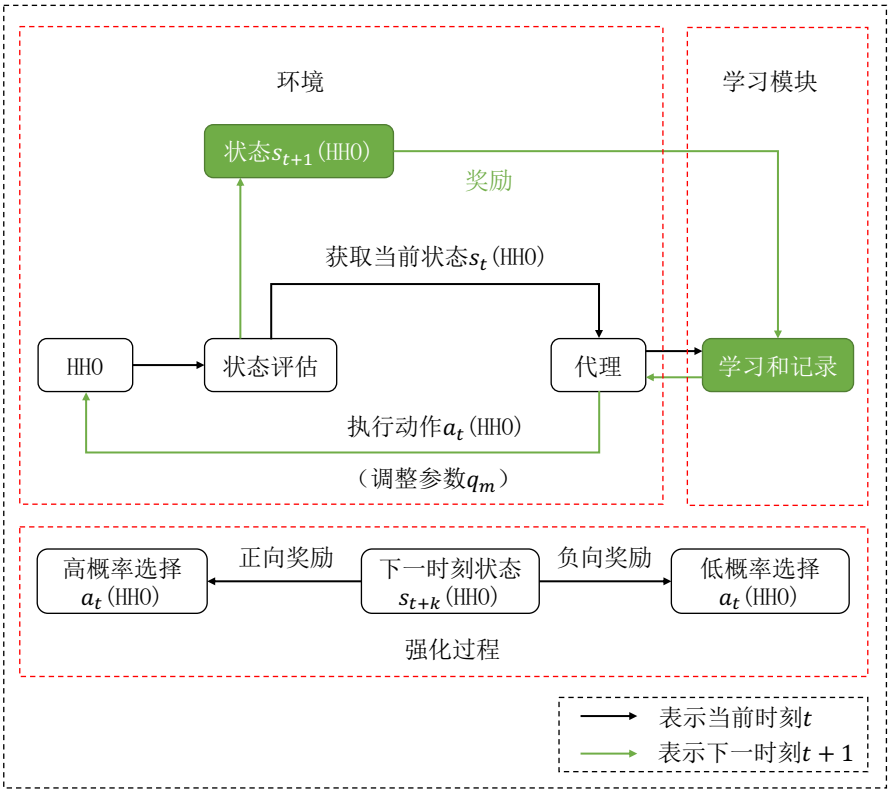

图 4.1: 组合模型

这个不断获取状态、选择动作、执行动作、获得反馈和调整策略的过程就是强化过程，经过学习过程积累学习经验以后，强化过程就可以通过智能调整 q_m 的方式来控制 HHO 算法，从而达到提升算法性能的效果。

4.2.2　构造学习模块

Q-learning 算法和 SARSA 算法各有千秋，Q-learning 算法学习效果强，但需要长时间的训练。SARSA 算法收敛的速度快，但是难以找到最优解。因此学习模块的主要部分就是将 Q-learning 算法和 SARSA 算法融合起来，吸收两者的优点。本文提出了一种融合两种 RL 算法的方法来提高 HHO 的性能，该方法在 HHO 算法迭代过程中的不同阶段应用 Q-learning 算法和 SARSA 算法，融合两者的优点。在 HHO 算法的初始阶段，先使用 SARSA 算法来学习，学习过程中使用公式 (2.16) 来更新 Q 值，并记录在 Q 值表中。这也将被视为 Q-learning 算法的训练阶段。当 HHO 算法运行到一定程度以后，满足了转换条件，就将 SARSA 算法转换成 Q-learning

算法。转换条件需要保证 Q-learning 算法有充足的学习经验，这是由非零 Q 值的数量决定的，而 Q 值的数量与状态和动作的数量有关。因此在 SLHHO 中，转换条件可以由公式 (4.1) 表示。

$$RL = \begin{cases} \text{SARSA}, & t < \dfrac{N_s \times N_a}{2} \\ \text{Q-learning}, & t \geq \dfrac{N_s \times N_a}{2} \end{cases} \tag{4.1}$$

其中，t 为目前的迭代数，N_s 为状态的数量，N_a 为动作的数量。

在初始阶段之后，Q-learning 算法取代 SARAS 算法来控制 HHO 算法，Q 值根据公式 (2.15) 来更新 Q 值，并记录在 Q 值表中。学习模块的具体执行步骤如下所示，示意图如4.2所示。

Step 1 初始化 Q 值表，行为状态，列为动作，初始值均为零值。

Step 2 获取当前状态 s_t，$s \leftarrow s_t$。

Step 3 根据公式 (4.1)，使用 SARSA 或者 Q-learning 的策略从状态 s 对应的动作集合中选择动作 a。

Step 4 执行动作 a。

Step 5 获取奖励 r，并且获取新的状态 s_{t+1}。

Step 6 如果 HHO 的终止条件满足则终止进程，否则跳到 Step 7。

Step 7 参照公式 (2.15) 或者公式 (2.16) 来更新 Q 值。

Step 8 更新状态：$s \leftarrow s_{t+1}$。

Step 9 确定转换条件是否满足，若满足转换条件，将 SARSA 替换成 Q-learning，然后跳到 Step 2。否则，直接跳到 Step 2。

4.2.3 状态集

本文所提出问题的目标是获取最短的最大完工时间，可以认为是 HHO 的适应度值。在 SLHHO 中，环境状态应该基于种群适应度值来构建，需要考虑以下几个方面：

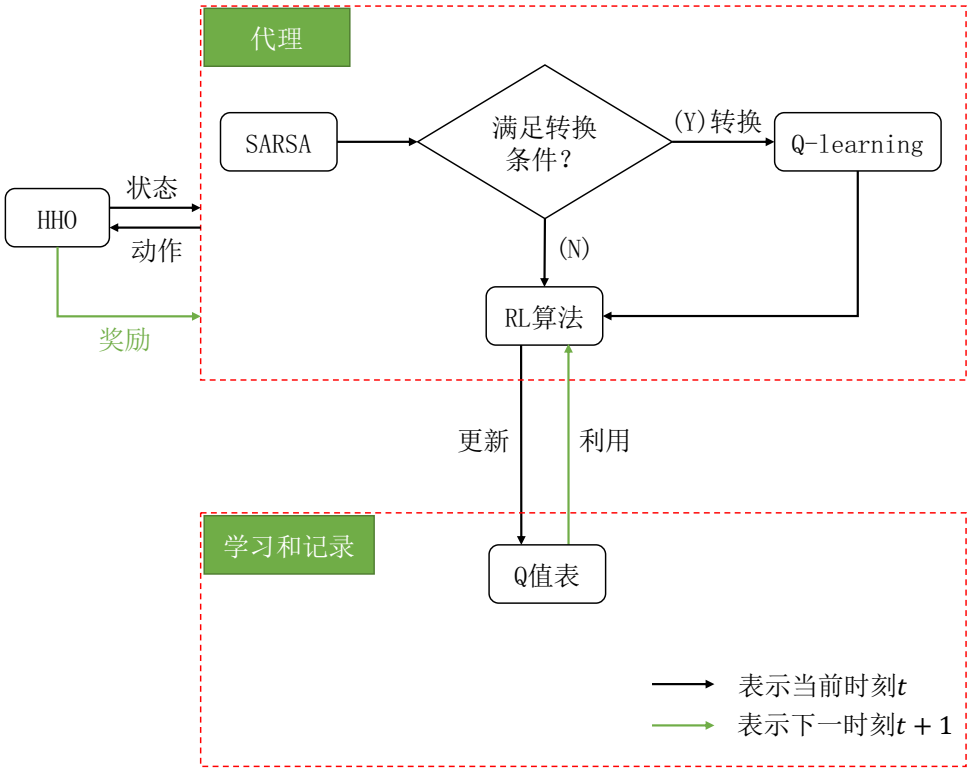

图 4.2: 学习模块

(1) 种群的平均适应度值;

(2) 种群多样化;

(3) 种群中最佳个体的适应度值;

本文使用了 4 个公式来评估环境状态。公式 (4.2) 计算的是当前种群平均适应度值。公式 (4.3) 计算的是当前种群的多样性。公式 (4.4) 计算的是当前种群最佳个体适应度值。公式 (4.5) 计算的是由公式 (4.2)、公式 (4.3) 和公式 (4.4) 加权得到的种群状态值。

$$f^* = \frac{\sum_{i=1}^{N} f(x_i^t)}{\sum_{i=1}^{N} f(x_i^1)}, \tag{4.2}$$

$$d^* = \frac{\sum_{i=1}^{N} \left| f(x_i^t) - \frac{\sum_{i=1}^{N} f(x_i^t)}{N} \right|}{\sum_{j=1}^{N} \left| f(x_j^1) - \frac{\sum_{j=1}^{N} f(x_j^1)}{N} \right|}, \tag{4.3}$$

$$m^* = \frac{\max f(x^t)}{\max f(x^1)}. \tag{4.4}$$

$$S^* = w_1 f^* + w_2 d^* + w_3 m^*, \tag{4.5}$$

其中 $f(x_i^t)$ 表示第 t 代种群的个体 i 的适应度值。$\max f(x^t)$ 表示第 t 代种群的最佳个体的适应度值。w_1, w_2, w_3 是权重系数，$w_1 + w_2 + w_3 = 1$，分别代表了三个影响因子的相对重要性。种群的平均适应度值和种群多样性反映了整体种群的情况，可以用来提升整体种群的质量，更容易获得优秀的个体。在种群的最佳个体的基础上进行开发，有可能得到更好的个体。所以这三者的重要性不同，本文分别设置 w_1，w_2 和 w_3 为 0.3，0.3 和 0.4。

状态的数量是非常重要的，状态数量越多，越适合智能学习，但需要更多的探索，消耗更多的计算资源。而状态数量太少会降低学习效果，导致结果不佳。根据 S^* 的特征，本文将状态集划分成 20 个，$S = \{s_1, s_2, ..., s_{20}\}$，其中 S^* 的区间值设置成 0.05。例如，$S^* \in [0, 0.05], s = s_1$, $S^* \in [0.05, 0.1], s = s_2$。

4.2.4　动作集

在 HHO 的每次迭代过程中，代理都会采取不同的动作来获得相应的 q_m。本文将动作集划分为 10 个，q_m 的取值范围是从 0.4 到 0.6，每个动作之间的间隔值是 0.02。例如，当选择动作 a_1 时，q_m 是 [0.4, 0.42) 范围内的一个随机数。

4.2.5　动作选择策略

在算法的初始阶段，所有的 Q 值都为零值，这意味着代理没有任何学习经验可以参考，只能执行搜索行为，同时学习经验。代理搜索未知环境，并记录学习过程，达到一定程度以后就可以利用获得的学习经验来指导代理的动作选择。RL 的动作选择策略选择使用 ε-贪心策略，因为它同时考虑到了搜索和开发，提供了搜

索和开发之间的权衡。

$$\pi(s_t, a_t) = \begin{cases} \max\limits_{a} Q(s_t, a), & \varepsilon \geq r_{0-1} \\ a(\text{Randomly}), & \varepsilon < r_{0-1} \end{cases} \tag{4.6}$$

其中 ε 是贪心率，r_{0-1} 是 0 到 1 范围内的随机数。当 $\varepsilon \geq r_{0-1}$ 时，预期 Q 值最大的动作就会被选中，所以也叫贪心策略。否则就随机选择。

4.2.6　奖励方法

在 SLHHO 中，奖励方法是根据最佳个体适应度值来设计的。代理一开始并不知道选择哪些动作会更好，而是通过尝试选择这些动作来发现哪些动作会产生更高的奖励。在代理选择了一个动作并执行以后，如果能达到更好的效果，它就会产生一个正向的奖励。奖励方法如公式4.7所示。

$$r = \frac{\max f(x^t) - \max f(x^{t-1})}{\max f(x^{t-1}))} \tag{4.7}$$

其中 $\max f(x^t)$ 和 $\max f(x^{t-1})$ 分别是第 t 代和前一代的最佳个体的适应度值。

SLHHO 算法的具体过程在算法4中描述，具体流程在图4.3中展示。

4.3　实验及结论

在上一章中，由于没有专门用于带有序列相关设置时间和资源约束的 FJSP 的基准实例，本文基于两个在 FJSP 研究领域中被广泛使用的经典数据集 Kacem 和 Brandimarte 构建了 12 个不同规模的基准实例来测试 HHO 算法的性能。本节将继续使用这 12 个基准实例来测试 SLHHO 算法的性能。

4.3.1　参数设置

在 SLHHO 算法中，种群数目 N 为 50，最多迭代的次数 T 为 100。哈里斯鹰选择栖息地策略的概率条件 q_m 将由 RL 算法来智能设置。RL 算法使用的一些学习参数（状态集和工作集的数量在上一节中已经确定）具体如表4.1中所示。由于

Algorithm 4 SLHHO 算法

初始化 HHO: N 为种群的数目, T 为最多迭代的次数
初始化 RL: Q 值表
设置已迭代数 $t \leftarrow 0$
评估各种群个体
随机选择一个动作 a_t, 更新动作 $a \leftarrow a_t$。评估环境状态 s_t, 更新状态 $s \leftarrow s_t$
while $t \leq T$ **do**
    根据公式 (4.7) 计算奖励 r_{t+1}
    if 转换条件未满足 **then**
        使用 ε-贪心策略来选择动作 a_{t+1}
        根据根据公式 (2.16) 来更新 Q 值
    else
        使用贪心策略选择动作 a_{t+1}
        根据根据公式 (2.15) 来更新 Q 值
        使用 ε-贪心策略选择动作 a_{t+1}
    end if
    根据公式 (4.5) 评估 HHO 的状态 s_{t+1}，更新状态 $s \leftarrow s_{t+1}$
    更新当前动作 $a \leftarrow a_{t+1}$
    执行动作 a
    执行 HHO 算法
    $t \leftarrow t+1$
    评估各种群个体
end while
return 最佳解决方案

表 4.1: RL 算法参数设置

参数	数值
折扣率 γ	0.2
初始奖励 r	1
学习率 α	0.75
贪心率 r_{0-1}	0.85
状态数量 N_s	20
动作数量 N_a	10

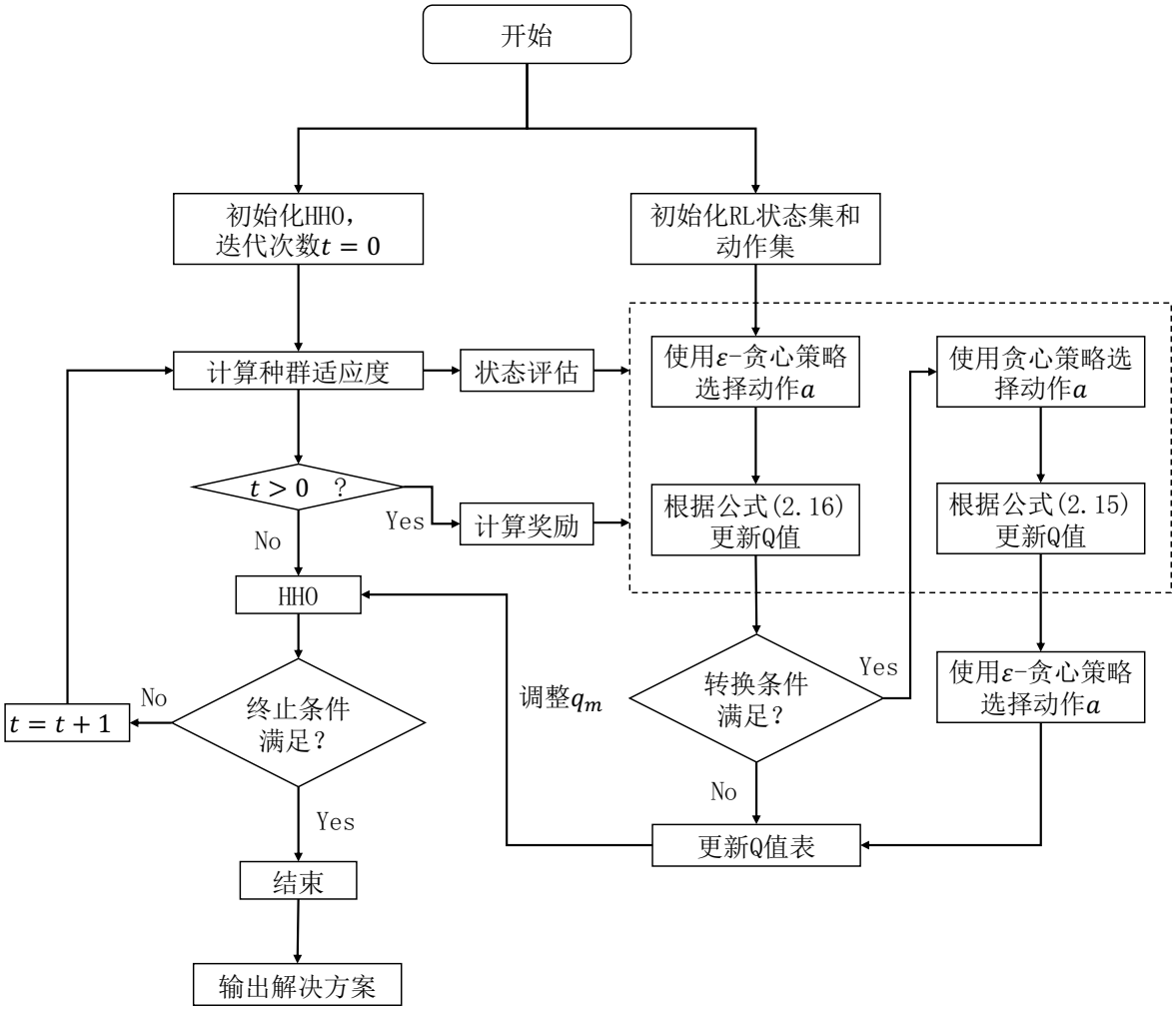

图 4.3: SLHHO 算法流程图

算法的随机特性，12 个基准实例都会独立运行 30 次。为了比较算法的整体性能，记录以下四个指标来衡量实验结果：

(1) Best：30 次运行实验的最佳完工时间。

(2) Avg：30 次运行实验的平均完工时间。

(3) SD：30 次运行实验得到的标准差。

(4) Time：30 次运行实验的平均时间（s）。

4.3.2 实验结果

SLHHO 使用了 RL 算法来智能调整关键参数，而 HHO 算法中的关键参数是固定设置的。为了验证 RL 算法的强化效果，本文将 HHO 算法和未使用 RL 算法的 HHO 算法在 12 个基准实例上进行了实验。实验的结果如表4.2中所示，在所有的 12 个基准实例中，SLHHO 算法都能够获得最佳的解决方案。而 HHO 算法只能在实例 01 和 02 这两个小规模的基准实例上获得最佳的解决方案。在剩下的 10 个规模较大的基准实例上，SLHHO 算法获得的解决方案质量均比 HHO 算法获得的要高。由于 SLHHO 算法中使用了 RL 算法，额外增加了计算资源，所以 SLHHO 算法在 12 个实例上的运行时间均比 HHO 算法要长，但是时间增加的不多。因此可以得出结论，RL 算法以增加了一些运行时间的代价，提升了 HHO 算法获得的解决方案的质量。

SLHHO 与 PSO-SA、DABC 和 IJaya 的实验结果在表4.3中展示。其中 $J \times M \times L$ 分别表示作业的数目、机器的数目和操作员的数目（例如，$4 \times 3 \times 2$ 表示这个实例有 4 个作业、3 台机器和 2 个操作员）。表4.3展示了，SLHHO 在 12 个基准实例中均可以获得最佳的解决方案，而且问题规模越大，解决方案质量提升越明显。在 Avg 方面，SLHHO 也是表现最好的，但是在标准偏差 SD 上，SLHHO 的优势就不明显了，算法稳定性还有所欠缺。

为了测试 SLHHO 的收敛速度，本文在图4.4中描绘了 SLHHO、PSO-SA、DABC 和 IJaya 在实例 06 上的实验结果的收敛曲线。从图4.4中能看出，和其他三种算法

表 4.2: SLHHO 和 HHO 在实例 01-12 上的实验结果

		SLHHO			HHO		
Instance	$J \times M \times L$	Best	Avg	Time(s)	Best	Avg	Time(s)
01	4 × 5 × 2	**17**	17.7	3.5	**17**	17.9	3.3
02	8 × 8 × 2	**27**	31.2	8.3	**27**	31.5	7.6
03	10 × 6 × 3	**70**	77.4	11.5	73	78.6	9.8
04	10 × 6 × 3	**58**	63.4	13.7	59	64.2	12.2
05	15 × 8 × 3	**289**	318.6	35.6	290	319.5	32.8
06	15 × 8 × 3	**119**	130.8	21.7	122	130.5	16.9
07	15 × 4 × 3	**259**	273.7	25.2	265	274	20.1
08	10 × 15 × 3	**155**	173.7	36.9	165	175	30.1
09	20 × 5 × 4	**230**	241.8	22.8	232	242.5	22.3
10	20 × 10 × 4	**667**	701.6	46.6	670	701.8	44.5
11	20 × 10 × 4	**509**	538.5	53	512	536.8	49.4
12	20 × 15 × 4	**401**	431	55.2	410	433.9	52

表 4.3: SLHHO 与其他算法在实例 01-12 的实验结果

		SLHHO			PSO-SA			DABC			IJaya		
Instance	$J \times M \times L$	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
01	4 × 5 × 2	**17**	**17.7**	1.9	19	24.4	2.2	20	26	2.5	22	25.3	1.5
02	8 × 8 × 2	**27**	**31.2**	4.9	30	34.1	4.4	30	36	4.5	29	34.3	3.8
03	10 × 6 × 3	**70**	**77.4**	6.8	82	94.9	6.4	89	99.3	4.9	88	97.9	4.4
04	10 × 6 × 3	**58**	**63.4**	5.9	74	86	5.7	81	91.2	4.7	81	92.3	4.9
05	15 × 8 × 3	**289**	**318.6**	20.3	343	377.3	24.3	373	418.7	24.4	386	432.7	27.7
06	15 × 8 × 3	**119**	**130.8**	6.7	134	143.5	7.3	158	171.6	7.3	142	158.8	8.5
07	15 × 4 × 3	**259**	**273.7**	9.7	279	309.5	13.6	305	336.7	11.6	314	333	10.9
08	10 × 15 × 3	**155**	**173.7**	12.2	194	226.1	13.3	251	277.9	12.6	251	273.2	10
09	20 × 5 × 4	**230**	**241.8**	13.3	276	315.1	22.2	319	350.2	15.2	304	328	15.5
10	20 × 10 × 4	**667**	**701.6**	21.8	706	744.3	18.6	757	815.6	20.6	792	825	18.9
11	20 × 10 × 4	**509**	**538.5**	22.7	581	672.8	30.2	663	732.6	25	632	719.4	33.4
12	20 × 15 × 4	**401**	**431**	25.5	491	543.5	23.1	585	641.5	15.9	609	647.6	17.5

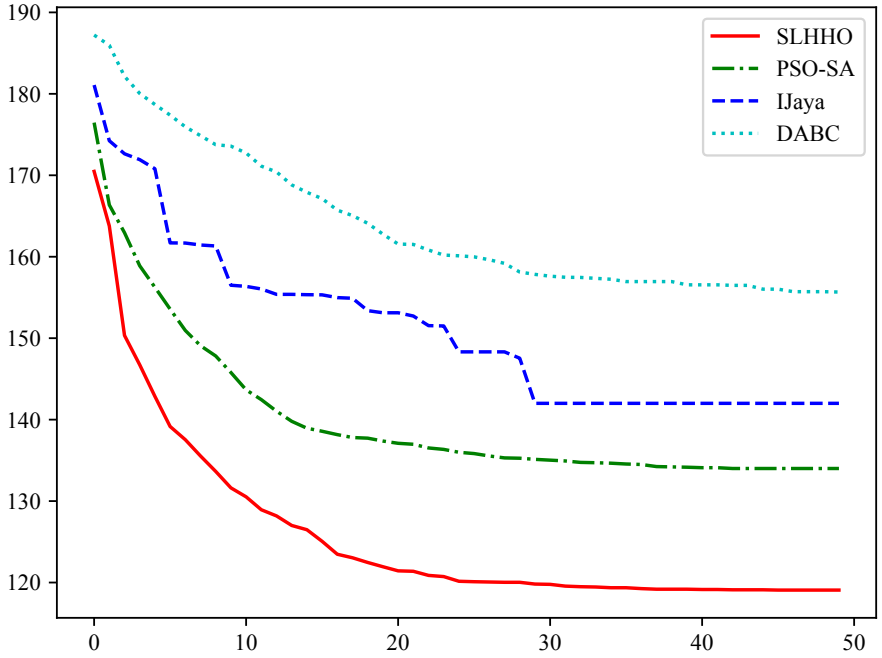

图 4.4: 实例 06 的收敛曲线

相比，SLHHO 的收敛速度更快，而且可以很清楚的看出 SLHHO 获得的解决方案的质量更高。

4.4　本章小结

本章结合了 RL 算法和 HHO 算法提出了一种 SLHHO 算法，主要思想是利用 RL 算法来智能调整 HHO 算法中的关键参数以此提升 HHO 算法的性能。RL 算法和 HHO 算法的组合模型主要分为学习模块、强化过程和环境。在学习模块中，RL 代理将 Q-learning 和 SARSA 算法应用在不同的执行阶段，以结合 SARSA 算法前期收敛速度快和 Q-learning 算法后期强化效果强的优势，从而获得更快的学习速度和更高的求解精度。本文综合考虑了种群最佳个体、种群平均适应度值和种群多样性来评估环境状态，然后根据环境状态的特征，合理地划分状态集和动作集，并设置了动作选择策略和奖励方法。

为了验证 RL 算法的强化效果，本文将 SLHHO 算法和未使用 RL 算法的 HHO 算法在 12 个基准实例上进行对比实验。从实验结果中能够看出，SLHHO 在所有的基准实例上均可以获得最佳的解决方案，而且在大部分的基准实例上都比未使用 RL 算法的 HHO 算法表现出色，除了略微增加运行时间。由此可见，使用 RL 算法智能调整关键参数可以有效提升 HHO 算法的性能。

此外本文还将 SLHHO 与其他三种有效算法在 12 个基准实例上的实验结果进行对比，从实验结果上可以看出 SLHHO 在所有实例上得到的结果都是最佳的。为了对比算法的收敛性，本文绘制了 SLHHO 和其他对比算法在实例 06 上的收敛曲线图。从图中能够看出，与其他算法相比，SLHHO 收敛速度更快，而且可以清楚的看出解决方案的质量有明显的提升。

第五章 总结与展望

5.1 工作总结

传统 FJSP 考虑了机器灵活性，而没有考虑到序列相关设置时间和操作员数量限制等一些实际生产过程中有可能遇到的约束条件。针对 FJSP 的这种不足，本文提出了带有序列相关设置时间和资源约束的 FJSP，其中资源约束表现为同一时间为机器更换模具的操作员的数量有限。该问题是经典 FJSP 的扩展，满足了实际生产过程中可能出现的一些需求场景，具有一定的研究价值。

为了解决该问题，本文引入了 HHO 算法。HHO 算法性能出色，是顶级的优化器之一，而且具有调节参数少、无梯度优点，具有广泛的适用性。HHO 算法还曾在在六个著名的工程优化问题上进行了实验，均获得了最佳解决方案。这些工程优化问题都可以建模成带有约束条件的目标优化模型，与 FJSP 具有一定的相似性，因此本文考虑使用 HHO 算法来解决 FJSP。

HHO 算法中的种群个体位置向量是用实数来表示的，因此本文首先将 HHO 算法中的种群个体位置向量转换成离散整数表示的机器分配向量和工序顺序向量。然后针对本文提出问题中的操作员数量限制约束专门设计了一种解码方法，在将机器分配向量和工序顺序向量解码成可行的调度方案的同时尽可能地避免机器陷入因等待操作员而空闲的停滞状态，从而使得最大完工时间最短。由于没有专门针对带有序列相关设置时间和资源约束的 FJSP 的基准实例，本文基于两个 FJSP 经典数据集构建了 12 个不同规模的基准实例来测试 HHO 算法的性能，并将实验结果与其他有效算法进行对比。实验结果表明，与其他有效算法相比，HHO 算法可以获得更好的解决方案，并且问题规模越大，解决方案质量提升越明显，但运行时间也有所增长。

考虑到算法的关键参数选择对算法性能影响较大，本文提出了一种 SLHHO 算法。SLHHO 使用 RL 算法来智能调整 HHO 算法的关键参数，从而提升 HHO 算法

的性能。在 SLHHO 中，RL 代理将 Q-learning 和 SARSA 应用在不同的执行阶段，以结合 SARSA 前期收敛速度快和 Q-learning 后期强化效果强的优点，从而获得更快的学习速度和更高的求解精度。本文综合考虑了种群最佳个体、种群平均适应度值和种群多样性来评估环境状态，然后根据环境状态的特征，合理划分状态集和动作集，并设置了动作选择策略和奖励方法。最后在 12 个基准实例上将 SLHHO、HHO 和其他有效算法进行对比实验。实验结果表明，SLHHO 可以在所有的基准实例上获得最佳的解决方案，而运行时间只比 HHO 的运行时间略微增加一点。本文还绘制了 SLHHO 和其他对比算法在实例 06 上的收敛曲线图，从图中能够看出，相较于其他对比算法，SLHHO 的收敛速度更快。由此可以得出，本文提出的 SLHHO 算法可以有效地解决带有序列相关设置时间和资源约束的 FJSP，性能表现优异。

5.2　未来展望

本文提出 SLHHO 来求解带有序列相关设置时间和资源约束的 FJSP，相较于其他算法在解决方案的质量上有较大提升，但还存在诸多不足，未来可以在以下方向上继续优化：

（1）SLHHO 良好性能的代价是运行时间的增长，这主要是由于 SLHHO 中的适应度评估过程有较多的遍历操作。因此未来可以考虑优化代码结构，使用辅助空间来减少遍历操作，从而降低算法的运行时间。

（2）本文针对资源约束设计了一种解码方法，但这种方法主要针对机器分配，有可能更改机器上的工序处理顺序。未来可以尝试从操作员方面入手，在不影响机器上工序处理顺序的情况下，利用碎片时间去提前更换模具。

（3）本文提出问题只考虑了序列相关设置时间和操作员数量限制，实际生产过程中还可能遇到其他约束条件，例如不同的操作员做相同的任务需要花费的时间可能也不同、机器故障等。未来在问题模型中可以考虑更多的约束条件。

参考文献

[1] 战戟. 工业 4.0 从自动生产到智能制造 [J]. 智库时代, 2020(6): 2.

[2] 陈醒. 说说美国先进制造业战略计划 [J]. 国际融资, 2019(9): 4.

[3] 钟丽连. 英国现代制造强国发展战略演进及对中国的启示 [D]. [S.l.]: 福州大学, 2018.

[4] 周济. 智能制造——” 中国制造 2025” 的主攻方向 [J]. 中国机械工程, 2015, 26(17): 12.

[5] ÇALIŞ B, BULKAN S. A research survey: review of AI solution strategies of job shop scheduling problem[J]. Journal of Intelligent Manufacturing, 2015, 26(5): 961–973.

[6] CHAUDHRY I A, KHAN A A. A research survey: review of flexible job shop scheduling techniques[J]. International Transactions in Operational Research, 2016, 23(3): 551–591.

[7] LAGUNA M. A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times[J]. IIE transactions, 1999, 31(2): 125–134.

[8] JOHNSON S M. Optimal two- and three-stage production schedules with setup times included[J]. Naval Research Logistics Quarterly, 1954, 1.

[9] BRUCKER P, SCHLIE R. Job-shop scheduling with multi-purpose machines[J]. Computing, 1990, 45(4): 369–375.

[10] GAO K Z, SUGANTHAN P N, PAN Q K, et al. An effective discrete harmony search algorithm for flexible job shop scheduling problem with fuzzy processing time[J]. International Journal of Production Research, 2015, 53(19): 5896–5911.

[11] GONG G, CHIONG R, DENG Q, et al. A hybrid artificial bee colony algorithm for flexible job shop scheduling with worker flexibility[J]. International Journal of Production Research, 2020, 58(14): 4406–4420.

[12] ABDELMAGUID T F. A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times[J]. Applied Mathematics and Computation, 2015, 260: 188–203.

[13] LI J-Q, PAN Q-K, TASGETIREN M F. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities[J]. Applied Mathematical Modelling, 2014, 38(3): 1111–1132.

[14] AHMADI E, ZANDIEH M, FARROKH M, et al. A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms[J]. Computers & Operations Research, 2016, 73: 56–66.

[15] AZZOUZ A, ENNIGROU M, SAID L B. Flexible Job-shop Scheduling Problem with Sequence-dependent Setup Times using Genetic Algorithm.[C] // ICEIS (2). 2016: 47–53.

[16] LI J-Q, DENG J-W, LI C-Y, et al. An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times[J]. Knowledge-Based Systems, 2020, 200: 106032.

[17] KACEM I, HAMMADI S, BORNE P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems[J]. IEEE Trans-

actions on Systems Man and Cybernetics Part C (Applications and Reviews), 2002, 32(1): 1–13.

[18] DU Y, FANG J, MIAO C. Frequency-domain system identification of an unmanned helicopter based on an adaptive genetic algorithm[J]. IEEE Transactions on Industrial Electronics, 2013, 61(2): 870–881.

[19] 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.

[20] YANG X-S, DEB S. Cuckoo search via Lévy flights[C] // 2009 World congress on nature & biologically inspired computing (NaBIC). 2009: 210–214.

[21] LIAN Z. A Local and Global Search Combine Particle Swarm Optimization Algorithm for Job-Shop Scheduling to Minimize Makespan[J]. Discrete Dynamics in Nature and Society,2010,(2010-07-14), 2010, 2010(4): 1038–1045.

[22] YUAN Y, XU H. Multiobjective flexible job shop scheduling using memetic algorithms[J]. IEEE Transactions on Automation Science and Engineering, 2013, 12(1): 336–353.

[23] MIRJALILI S, MIRJALILI S M, LEWIS A. Grey wolf optimizer[J]. Advances in engineering software, 2014, 69: 46–61.

[24] NAJID N M, DAUZERE-PERES S, ZAIDAT A. A modified simulated annealing method for flexible job shop scheduling problem[C] // Systems, Man and Cybernetics, 2002 IEEE International Conference on. 2002.

[25] YANG X-S. Firefly algorithm, stochastic test functions and design optimisation[J]. International journal of bio-inspired computation, 2010, 2(2): 78–84.

[26] HURINK J, JURISCH B, THOLE M. Tabu search for the job-shop scheduling problem with multi-purpose machines[J]. Operations-Research-Spektrum, 1994, 15(4): 205 – 215.

[27] HEIDARI A A, MIRJALILI S, FARIS H, et al. Harris hawks optimization: Algorithm and applications[J]. Future generation computer systems, 2019, 97: 849 – 872.

[28] BUI D T, MOAYEDI H, KALANTAR B, et al. A novel swarm intelligence—Harris hawks optimization for spatial assessment of landslide susceptibility[J]. Sensors, 2019, 19(16): 3590.

[29] MEHTA M S, SINGH M B, GAGANDEEP M. Harris Hawks optimization for solving optimum load dispatch problem in power system[J]. Int J Eng Res Technol, 2019, 8(6): 962 – 968.

[30] MOAYEDI H, OSOULI A, NGUYEN H, et al. A novel Harris hawks'optimization and k-fold cross-validation predicting slope stability[J]. Engineering with Computers, 2019: 1 – 11.

[31] YILDIZ A R, YILDIZ B S, SAIT S M, et al. The Harris hawks, grasshopper and multi-verse optimization algorithms for the selection of optimal machining parameters in manufacturing operations[J]. Materials Testing, 2019, 61(8): 725 – 733.

[32] GOLILARZ N A, GAO H, DEMIREL H. Satellite image de-noising with Harris hawks meta heuristic optimization algorithm and improved adaptive generalized Gaussian distribution threshold function[J]. IEEE Access, 2019, 7: 57459 – 57468.

[33] JIA H, LANG C, OLIVA D, et al. Dynamic harris hawks optimization with mutation mechanism for satellite image segmentation[J]. Remote sensing, 2019, 11(12): 1421.

[34] BRANDIMARTE P. Routing and scheduling in a flexible job shop by tabu search[J]. Annals of Operations research, 1993, 41(3) : 157 – 183.

[35] KACEM I, HAMMADI S, BORNE P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2002, 32(1) : 1 – 13.

[36] XIA W, WU Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems[J]. Computers & industrial engineering, 2005, 48(2) : 409 – 425.

[37] CHEN H, IHLOW J, LEHMANN C. A genetic algorithm for flexible job-shop scheduling[C] // Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C) : Vol 2. 1999 : 1120 – 1125.

[38] ZHANG H, GEN M. Multistage-based genetic algorithm for flexible job-shop scheduling problem[J]. Journal of Complexity International, 2005, 11(2) : 223 – 232.

[39] EMARY E, ZAWBAA H M, GROSAN C. Experienced gray wolf optimization through reinforcement learning and neural networks[J]. IEEE transactions on neural networks and learning systems, 2017, 29(3) : 681 – 694.

[40] SHAHRABI J, ADIBI M A, MAHOOTCHI M. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling[J]. Computers & Industrial Engineering, 2017, 110 : 75 – 82.

[41] 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.

[42] HSIEH Y-Z, SU M-C. A Q-learning-based swarm optimization algorithm for economic dispatch problem[J]. Neural Computing and Applications, 2016.

[43] CHEN F, GAO Y, CHEN Z-Q, et al. SCGA: Controlling genetic algorithms with Sarsa (0)[C] // International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06): Vol 1. 2005: 1177 – 1183.

[44] WANG Y H, LI T H S, LIN C J. Backward Q-learning: The combination of Sarsa algorithm and Q-learning[J]. Engineering Applications of Artificial Intelligence, 2013, 26(9): 2184 – 2193.

[45] YANG X-S. Nature-inspired metaheuristic algorithms[M]. [S.l.]: Luniver press, 2010.

[46] QIN J, LI M, SHI Y, et al. Optimal synchronization control of multiagent systems with input saturation via off-policy reinforcement learning[J]. IEEE transactions on neural networks and learning systems, 2018, 30(1): 85 – 96.

[47] SUTTON R S, BARTO A G. Reinforcement learning: An introduction[M]. [S.l.]: MIT press, 2018.

[48] PINEDO M, HADAVI K. Scheduling: theory, algorithms and systems development[G] // Operations research proceedings 1991. [S.l.]: Springer, 1992: 35 – 42.

[49] BARTHOLOMEW, D. J. Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop[J]. Journal of the Operational Research Society, 1982, 33(9): 862 – 862.

[50] KACEM I, HAMMADI S, BORNE P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic[J]. Mathematics and computers in simulation, 2002, 60(3-5) : 245 – 276.

[51] TANG H, CHEN R, LI Y, et al. Flexible job-shop scheduling with tolerated time interval and limited starting time interval based on hybrid discrete PSO-SA: An application from a casting workshop[J]. Applied Soft Computing, 2019, 78 : 176 – 194.

致　谢

时光荏苒，在华师大中北校区已经度过了三个春夏秋冬，研究生的学习生涯已然到了尾声。回顾这段漫漫求学路，有崎岖坎坷，也有绝处逢生，有穷途末路，也有柳暗花明。

2018 年，我刚毕业不久，生活诸多不顺。抱着逃避困难的想法以及对科研生活的好奇，我决心重返校园读研，顺便重开生活。表姐知道以后特地找我聊天，告诉我人生漫漫几十年，读不读研都是一种生活，有上进心是好事，但万万不可为了逃避遇到的问题而去选择另一种生活。这番话警醒了我，并让我在之后遇到重大困难时，选择直面，尽力解决，而不是继续选择逃避。

一路辛苦备考，差额进入复试，最后幸运的因为扩招而拿到了录取名额。至今还能记得备考的那个冬天的寒冷，和得知录取时的快乐。那段全心全意为了一个目标而奋斗的时光是我人生中的宝贵财富。也许我不是一个长于学习，能力出众的人，但我已经知道了努力和坚持的意义。在此，我也想感谢一下挚友杨文章，我们互相陪伴一起备考，最终都取得了好的结果。罗曼·罗兰说过，世界上只有一种真正的英雄主义，那就是看清生活的真相之后，依然热爱生活。我觉得他就是这样一个敢于直面生活的困苦，努力为了自己的理想拼搏的英雄，这也是我想要学习的地方。接下来他还会在校园里继续深造，希望他可以在科研的道路上继续狂奔，到达他理想的高度。感谢父母对我的支持，对我做出的任何决定都给予充分的肯定，让我可以没有后顾之忧的在学校专心学习。也感谢姐姐、表姐们对我的关心，不仅在生活上对我嘘寒问暖，在人生道路上也给予我宝贵的建议。

进入校园以后，我遇到了一些非常好的老师和同学。在此，我也想对他们表达一下最真挚的感谢。首先，我要由衷的感谢我尊敬的导师卜天明老师。卜天明老师治学严谨，待人和善，不仅在论文课题的选择上充分考虑学生的意见，在论文撰写过程中更是一字一句的帮助学生纠正错误，调整论文框架结构。最终在卜天明老

师的帮助下，获得了一点学术成果。其次要感谢张帆、石康师兄的帮助，不厌其烦的解答我在学习和生活上的问题。也感谢实验室同学：徐宁、王晨超、杨睿和王梦旦，在上海的所有记忆都与你们有关，和你们朝夕相处的日子会是我人生中的难忘经历，感谢你们在枯燥单调的科研学习生活中带给我的欢乐。要感谢的人还有很多，此处就不一一列举了，感谢你们给予我的帮助和陪伴。

最后，我要向参与评审本论文的各位老师、专家表示由衷的感谢，感谢您们在百忙中审阅拙作并提出宝贵的意见！

叶孙飞

二零二二年二月于理科大楼

攻读硕士学位期间科研情况

■ 学术论文

[1] **Sunfei Ye**, Tian-Ming Bu.**A Self-learning Harris Hawks Optimization Algorithm for Flexible Job Shop Scheduling with Setup Times and Resource Constraints**[C]. IEEE International Conference on Systems, Man, and Cybernetics.(SMC2021，CCF C 类，已收录)