machines

MDPI

Article

An Optimization Method for Green Permutation Flow Shop Scheduling Based on Deep Reinforcement Learning and MOEA/D

Yongxin Lu, Yiping Yuan *, Adilanmu Sitahong, Yongsheng Chao and Yunxuan Wang

College of Mechanical Engineering, Xinjiang University, Urumqi 830046, China; lyx18280338396@stu.xju.edu.cn (Y.L.); adilanmujx@xju.edu.cn (A.S.); cys21st@xju.edu.cn (Y.C.); w15231980129@163.com (Y.W.)

* Correspondence: yipingyuan@xju.edu.cn

Abstract: This paper addresses the green permutation flow shop scheduling problem (GPFSP) with energy consumption consideration, aiming to minimize the maximum completion time and total energy consumption as optimization objectives, and proposes a new method that integrates end-to-end deep reinforcement learning (DRL) with the multi-objective evolutionary algorithm based on decomposition (MOEA/D), termed GDRL-MOEA/D. To improve the quality of solutions, the study first employs DRL to model the PFSP as a sequence-to-sequence model (DRL-PFSP) to obtain relatively better solutions. Subsequently, the solutions generated by the DRL-PFSP model are used as the initial population for the MOEA/D, and the proposed job postponement energy-saving strategy is incorporated to enhance the solution effectiveness of the MOEA/D. Finally, by comparing the GDRL-MOEA/D with the MOEA/D, NSGA-II, the marine predators algorithm (MPA), the sparrow search algorithm (SSA), the artificial hummingbird algorithm (AHA), and the seagull optimization algorithm (SOA) through experimental tests, the results demonstrate that the GDRL-MOEA/D has a significant advantage in terms of solution quality.

Keywords: deep reinforcement learning; multi-objective optimization; permutation flow shop scheduling; MOEA/D algorithm; energy-saving strategy

check for updates

Citation: Lu, Y.; Yuan, Y.; Sitahong, A.; Chao, Y.; Wang, Y. An Optimization Method for Green Permutation Flow Shop Scheduling Based on Deep Reinforcement Learning and MOEA/D. *Machines* **2024**, *12*, 721. https://doi.org/10.3390/machines12100721

Academic Editor: Kai Cheng

Received: 15 August 2024
Revised: 26 September 2024
Accepted: 9 October 2024
Published: 11 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The permutation flow shop scheduling problem (PFSP), as a classical challenge in the domain of combinatorial optimization problems (COPs), has long been a focal point of intense research and enthusiasm within the international academic community. The key to solving this problem lies in designing algorithms that are efficient, fast, and accurate to fulfill the stringent requirements of contemporary manufacturing for both production efficiency and product quality. In the flow line production environment, this scheduling problem is particularly prominent in industries such as automobile assembly, electronic product manufacturing, food processing, steel, textiles, aerospace, and pharmaceutical processing. Therefore, the successful resolution of this problem not only significantly enhances the efficiency of the production process, but also holds promise for opening up new avenues for innovation and development in the entire manufacturing industry.

Currently, the methods for solving the permutation flow shop scheduling problem are mainly divided into two categories: exact algorithms and approximate algorithms. Exact algorithms aim to find the optimal solution by exhaustively enumerating and evaluating all possible scheduling options, such as branch and bound [1], dynamic programming [2], and integer programming [3]. These methods can theoretically guarantee the optimal solution. However, when faced with large-scale practical problems, due to constraints on computational resources and the "combinatorial explosion" phenomenon, exact algorithms often become impractical. Conversely, approximation algorithms aim to quickly find "good

enough" solutions rather than theoretically optimal ones. They can provide practical solutions within a reasonable timeframe for addressing large-scale and complex problems, which are further categorized into heuristic and metaheuristic methods. Heuristic methods encompass the Campbell Dudek Smith algorithm (DCS) [4], the Gupta algorithm [5], the Nawaz Enscore Ham algorithm (NEH) [6], and a variety of scheduling rules [7], with NEH widely regarded as the most effective approach for resolving PFSP [8–11]. Nevertheless, these heuristic algorithms often operate under simplified assumptions or rules, which may limit their ability to deliver optimal or satisfactory solutions in the face of complex scheduling challenges. Metaheuristic methods, which simulate natural processes or leverage specific heuristic rules, guide the search process to yield satisfactory scheduling solutions and have achieved a series of results in solving PFSP. For example, Zheng et al. [12] developed a hybrid bat optimization algorithm that incorporates variable neighborhood structure and two learning strategies to address the PFSP. Chen et al. [13] proposed a hybrid grey wolf optimization method with a cooperative initialization strategy for solving the PFSP with the objective of minimizing the maximum completion time. Tian et al. [14] proposed a novel cuckoo search algorithm for solving the PFSP. Khurshid et al. [15] proposed a hybrid evolutionary algorithm that integrates an improved evolutionary global search strategy and a simulated annealing local search strategy for solving the PFSP. Razali [16] combined the NEH algorithm with the artificial bee colony algorithm (ABC) to solve the PFSP, which enhances the convergence speed of the ABC. Qin et al. [17] aimed to minimize the completion time and proposed a hybrid cooperative coevolutionary search algorithm (HSOS) based on the cooperative coevolutionary search algorithm combined with local search strategies to solve the PFSP. Rui et al. [18] introduced a multi-objective discrete sine optimization method (MDSOA) to address the mixed PFSP, aiming to minimize the makespan and the maximum tardiness. Yan et al. [19] introduced a novel hybrid crow search algorithm (NHCSA) that enhances the quality of the initial population through an improved version of the NEH method. It employs the smallest-position-value rule for encoding discrete scheduling problems and incorporates a local search mechanism, which is designed to solve the PFSP with the objective of minimizing the maximum completion time. It is evident that metaheuristic algorithms have achieved satisfactory results and performance in the application of PFSP, but their iterative search process is often time-consuming, and they do not fully utilize historical information to optimize and adjust the search strategy. Therefore, there is still significant potential for optimization in solving large-scale problems. For existing specific scheduling problems, delving into the essence of the problem and designing optimization strategies accordingly is crucial. Additionally, rationally utilizing historical information to adjust the search patterns of the algorithms is another key approach to achieve efficient solutions.

In recent years, the rapid advancements in artificial intelligence and machine learning technologies have yielded remarkable achievements across various domains, including speech recognition [20], energy management [21], image processing [22], pattern recognition [23], healthcare [24], and traffic management [25], opening new avenues for solving complex COPs. Consequently, the academic community has started to investigate the application of machine learning techniques to COPs, quickly becoming a research focal point and yielding abundant results. For example, Vinyals et al. [26] introduced Pointer Networks (PN) as an innovative strategy for tackling sequence-to-sequence modeling issues. Ling et al. [27] developed a fully convolutional neural network that learns the optimal solution from the feasible domain to solve the traveling salesman problem (TSP). However, the neural networks employed in the aforementioned approaches are all trained and refined on labeled data, where the quality of the labels directly impacts their performance, a category that is known as supervised learning. Additionally, collecting high-quality data for combinatorial optimization problems (COPs) is a time-consuming and expensive process that requires substantial computational resources and expertise to solve complex problems and generate accurate labels. Therefore, exploring neural network

models that can effectively solve COPs is an important topic that urgently needs to be addressed in current research.

Reinforcement learning (RL), unlike supervised learning methods, is a machine learning approach based on a reward mechanism, where an agent learns the optimal strategy through interactions with the environment. In RL, the agent does not rely on a pre-labeled dataset but instead explores the environment, performs actions, and receives feedback. Through this process, the agent gradually adjusts its behavioral strategies, thereby finding the best solution in a dynamically changing environment. This offers an efficient and adaptable method for solving COPs, yielding substantial advancements and real-world applications in challenging domains, including the TSP, vehicle routing problem (VRP), and workshop scheduling problem. To solve the TSP, Zhang et al. [28] proposed a manager–worker deep reinforcement learning (DRL) network architecture based on a graph isomorphism network (GIN) for solving the multiple-vehicle TSP with time windows and rejections. Luo et al. [29] proposed a DRL approach that incorporates a graph convolutional encoder and a multi-head attention mechanism decoder, aimed at addressing the limitations of existing machine learning methods in solving the TSP, which typically does not fully utilize hierarchical features and can only generate single permutations. Bogyrbayeva et al. [30] proposed a hybrid model that combines an attention-based encoder with a long short-term memory (LSTM) network decoder to address the inefficiency of attention-based encoder and decoder models in solving the TSP involving drones. Gao et al. [31] developed a multi-agent RL approach based on gated transformer feature representation to improve the solution quality of multiple TSPs. To solve the VRP, Wang et al. [32] proposed a method that combines generative adversarial networks with DRL to solve the VRP. Pan et al. [33] presented a method that can monitor and adapt to changes in customer demand in real time, addressing the issue of uncertain customer needs in VRP. Wang et al. [34] proposed a two-stage multi-agent RL method based on Monte Carlo tree search, aiming for efficient and accurate solutions to the VRP. Xu et al. [35] developed an RL model with a multi-attention aggregation module that dynamically perceives and encodes context information, addressing the issue of existing RL methods not fully considering the dynamic network structure between nodes when solving VRPs. Zhao et al. [36] proposed a DRL approach for large-scale VRPs that consists of an attention-based actor, an adaptive critic, and a routing simulator. To solve workshop scheduling problems, Si et al. [37] designed an environment state based on a multi-agent architecture using DRL to solve the job shop scheduling problem (JSSP). Chen et al. [38] proposed a deep reinforcement learning method that integrates attention mechanisms and disjunctive graph embedding to solve the JSSP. Shao et al. [39] redesigned the state space, action space, and reward function of RL to solve the flexible job shop scheduling problem (FJSSP) with the objective of minimizing completion time. Han et al. [40] proposed an end-to-end DRL method based on an encoder and a decoder to solve the FJSSP. Yuan et al. [41] constructed a DRL framework based on a multilayer perceptron for extracting environmental state information to solve the FJSSP, which enhances the computational efficiency and decision-making capabilities of the algorithm. Wan et al. [42] developed a DRL approach based on the actor–critic framework to address the FJSSP with the objective of minimizing the makespan. Peng et al. [43] presented a multi-agent RL method with double Q-value mixing for addressing the extended FJSSP characterized by technological and path flexibility, a variable transportation time, and an uncertain environment. Wu et al. [44] proposed a DRL approach for the dynamic job shop scheduling problem (DJSSP) with an uncertain job processing time that incorporates proximal policy optimization (PPO) enhanced by hybrid prioritized experience replay. Liu et al. [45] introduced a multi-agent DRL framework that can autonomously learn the relationship between production information and scheduling objectives for solving the DJSSP. Gebreyesus et al. [46] presented an end-to-end scheduling model based on DRL for the DJSSP, which utilizes an attention-based Transformer network encoder and a gate mechanism to optimize the quality of solutions. Wu et al. [47] introduced a DRL scheduling model combined with a spatial pyramid pooling network (SPP-Net) to address the DJSSP.

The model employs novel state representation and reward function design and is trained using PPO. Experiments in both static and dynamic scheduling scenarios demonstrated that this method outperforms existing DRL methods and paired priority. Su et al. [48] developed a DRL method based on the graph neural network (GNN) for the DJSSP with machine failures and stochastic processing time, which utilizes GNN to extract state features and employs evolutionary strategies (ES) to find the optimal policies. Liu et al. [49] developed a DRL framework that employs GNN to convert disjunctive graph states into node embeddings and is trained using the PPO algorithm for solving the DJSSP with random job arrivals and random machine failures. Zhu et al. [50] proposed a method based on deep reinforcement learning to solve the DJSSP. Tiacci et al. [51] successfully integrated the DRL agent with a discrete event simulation system to tackle a dynamic flexible job shop scheduling problem (DFJSSP) with new job arrivals and machine failures. Zhang et al. [52] proposed a DRL method integrated with a GNN to address the DFJSSP with uncertain machine processing time, training the agents using the PPO algorithm, and demonstrating through experiments that this method outperforms traditional algorithms and scheduling rules in both static and dynamic environments. Chang et al. [53] put forward a hierarchical deep reinforcement learning method composed of a double deep Q-network (DDQN) and a dueling DDQN to solve the multi-objective DFJSSP. Zhou et al. [54] proposed a DRL method that uses a disjunctive graph to represent the state of the environment for solving the PFSP. Pan et al. [55] designed an end-to-end DRL framework to solve the PFSP with the objective of minimizing the completion time. Wang et al. [56] introduced a DRL method based on a long short-term memory (LSTM) network to address the non-PFSP. Table 1 summarizes the common methods for solving PFSP, as well as the application of RL in dealing with various shop scheduling problems.

Table 1. Existing methods for solving various shop scheduling problems.

References	Type of Problem	Objectives	Approach	Approach Type
[1]	PFSP	Makespan	Branch and bound	Exact algorithms
[2]	PFSP	Makespan	Dynamic programming	
[3]	PFSP	Makespan	Integer programming	
[4]	PFSP	Makespan	Campbell Dudek Smith algorithm	Heuristic methods
[5]	PFSP	Makespan	Gupta algorithm	
[6–11]	PFSP	Makespan	Nawaz Enscore Ham algorithm	
[12]	PFSP	Makespan	Hybrid bat optimization algorithm	Metaheuristic methods
[13]	PFSP	Makespan	Hybrid grey wolf algorithm	
[14]	PFSP	Makespan, total energy consumption	Cuckoo search algorithm	
[15]	PFSP	Makespan	Hybrid Evolution Strategy	
[16]	PFSP	Makespan	Artificial Bee Colony algorithm	
[17]	PFSP	Makespan	Hybrid cooperative coevolutionary search algorithm	
[18]	PFSP	Makespan, maximum tardiness	Discrete sine optimization method	
[19]	PFSP	Makespan	Hybrid crow search algorithm	
[37]	JSSP	Makespan	Soft actor–critic algorithm	Reinforcement learning methods
[38]	JSSP	Makespan	Deep reinforcement learning	
[39]	FJSSP	Makespan	Hybrid deep reinforcement learning	
[40]	FJSSP	Makespan	Deep reinforcement learning	
[41]	FJSSP	Makespan	Deep reinforcement learning	
[42]	FJSSP	Makespan	Deep actor–critic reinforcement learning	
[43]	FJSSP	Makespan	Multi-agent reinforcement learning	
[44]	DJSSP	Makespan	Deep reinforcement learning	
[45]	DJSSP	Makespan	Deep multi-agent reinforcement learning	
[46]	DJSSP	Makespan	Gated-attention model with reinforcement learning	

Table 1. *Cont.*

References	Type of Problem	Objectives	Approach	Approach Type
[47]	DJSSP	Makespan	Deep reinforcement learning	
[48]	DJSSP	Makespan	Graph reinforcement learning	
[49]	DJSSP	Makespan	Graph neural network and deep reinforcement learning	
[50]	DJSSP	Makespan	Deep reinforcement learning	
[51]	DFJSSP	Makespan	Deep reinforcement learning	
[52]	DFJSSP	Makespan	Proximal Policy Optimization algorithm	
[53]	DFJSSP	Makespan	The combination of a double deep Q-network (DDQN) and a dueling DDQN	
[54]	PFSP	Makespan	Deep reinforcement learning	
[55]	PFSP	Makespan	Deep reinforcement learning	
[56]	PFSP	Makespan	Deep reinforcement learning	

In summary, RL can learn how to make optimal decisions through interaction with the environment, offering effective solutions for combinatorial optimization problems such as the TSP, VRP, and workshop scheduling problem. However, there are several shortcomings in the existing research on using RL to solve the workshop scheduling problem:

- Firstly, most researchers have focused their studies primarily on the static JSSP [37,38], static FJSSP [39,43], DJSSP [44–50], and DFJSSP [51–53], with relatively less research on PFSP [54–56].
- Secondly, the optimization goal of most research is to minimize completion time, with little consideration given to other objectives such as energy consumption, machine utilization, and delivery time. However, as an essential aspect of manufacturing, energy consumption has become increasingly significant due to its dual impact on production costs and the environment. Therefore, implementing an effective energy-saving strategy in production scheduling not only helps to reduce production costs and enhance the competitiveness of enterprises, but also reduces carbon emissions. It aligns with the global green environmental trend and promotes the achievement of sustainable development goals.
- Furthermore, in existing research, the state of the environment for RL algorithms is typically constructed as a set of performance indicators, with each indicator usually mapped to a specific feature. However, the intricate correlations among these performance indicators lead to a complex internal structure of the environmental state, which may contain a large amount of redundant information. This complexity not only increases the difficulty of convergence for neural networks, but may also negatively impact the decision-making process of the agent, reducing the accuracy and efficiency of its decisions.
- Finally, in existing research, the action space of the agent is often limited to a series of heuristic rules based on experience. While these rules are easy to understand and implement, they may restrict the exploration of the agent, preventing it from fully uncovering and executing more complex and efficient scheduling strategies.

The multi-objective evolutionary algorithm based on decomposition (MOEA/D), as a classic and effective multi-objective optimization method, is favored by many scholars for its intuitive understandability, high robustness, simple parameter setting, and adaptability to a variety of complex problems. These characteristics make the algorithm particularly prominent in solving job shop scheduling problems and have become a commonly adopted approach by scholars in dealing with such challenges. In recent years, MOEA/D has been applied to solve multi-objective permutation flow shop scheduling problems by breaking down complex multi-objective problems into multiple sub-problems for independent optimization and using Chebyshev aggregation functions to guide the convergence process of the population [57,58]. However, it is worth noting that MOEA/D is highly dependent on the initial solutions, and the quality of the initial solutions will directly affect the overall

performance of the algorithm. This feature requires special attention to the generation strategy of initial solutions when using MOEA/D to ensure that the algorithm can achieve the best results.

Therefore, in response to the aforementioned issues, this paper proposes an innovative algorithmic framework that integrates DRL with the MOEA/D algorithm (GDR-MOEA/D) to address the green permutation flow shop scheduling problem, aiming to minimize the objectives of maximum completion time and total energy consumption. The main contributions of this paper are as follows:

- Firstly, for the existing end-to-end deep reinforcement learning network, which is difficult to apply to different scales of the green permutation flow shop scheduling problem (GPFSP) considering energy consumption, we designed a network model based on DRL (DRL-PFSP) to solve PFSP. This network model does not require any high-quality labeled data and can flexibly handle PFSPs of various sizes, directly outputting the corresponding scheduling solutions, which greatly enhances the practicality and usability of the algorithm.
- Secondly, the DRL-PFSP model is trained using the actor–critic RL method. After the model is trained, it can directly generate scheduling solutions for PFSPs of various sizes in a very short time.
- Furthermore, in order to significantly enhance the quality of solutions produced by the MOEA/D algorithm, this study innovatively employs solutions generated by the DRL-PFSP model as the initial population for MOEA/D. This approach not only provides MOEA/D with a high-quality starting point, but also accelerates the convergence of the algorithm and improves the performance of the final solutions. Additionally, to further optimize the energy consumption target, a strategy of job postponement for energy saving is proposed. This strategy reduces the machine's idle time without increasing the completion time, thereby achieving further optimization of energy consumption.
- Eventually, through comparative analysis of simulation experiments with the unimproved MOEA/D, NSGA-II, MPA, SSA, AHA, and SOA, the GDRL-MOEA/D model algorithm constructed in this study demonstrated superior performance. The experimental results reveal that the solution quality of GDRL-MOEA/D was superior to the other six algorithms in all 24 test cases. In terms of solution speed, GDRL-MOEA/D was not significantly different from the other algorithms, and the difference was within an acceptable range.

The remainder of this paper is organized as follows. The mathematical model of the GPFSP and the objective functions are presented in Section 2. In Section 3, the proposed GDRL-MOEA/D framework is elaborately concluded. Section 4 presents simulation experiments that assess the proposed algorithm against traditional methods, validating its efficiency. Finally, Section 5 offers a comprehensive summary and an outlook on the future of the work.

2. Multi-Objective Optimization Model for the GPFSP

2.1. Problem Description

The GPFSP in this paper can be described as follows: there are n jobs $J = \{J_1, J_2, \ldots, J_n\}$ that need to be processed on m machines $M = \{M_1, M_2, \ldots, M_m\}$ at the initial moment. Each job follows the same production route through all the machines, starting from machine M_1, M_2, M_3, and so on, until all operations are completed on the last machine M_m. The processing time p_{ij} of job i on machine M_j is known, and the processing power and idle power of each machine are also determined. The objective of this paper is to determine an optimal sequence $\pi^* = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}$ for the GPFSP that minimizes the maximum completion time for all jobs and the total energy consumption of all machines. In general, the PFSP is based on the following assumptions:

- All jobs are mutually independent and can be processed at the initial moment;
- Only one job can be processed on each machine at any given time;

- Each job needs to be processed on each machine exactly once;
- All jobs have the same processing sequence on each machine;
- The job cannot be interrupted once it starts processing on a machine;
- The transportation and setup times of jobs between different machines are either disregarded or incorporated into the processing time of the jobs.

Figure 1 exhibits an example of a PFSP scheduling scheme for 6 jobs on 5 machines, with the figure elaborately detailing the processing sequence of each job on every machine. Namely, the sequence of the scheduling plan is $\pi = \{3, 1, 2, 5, 4, 6\}$.

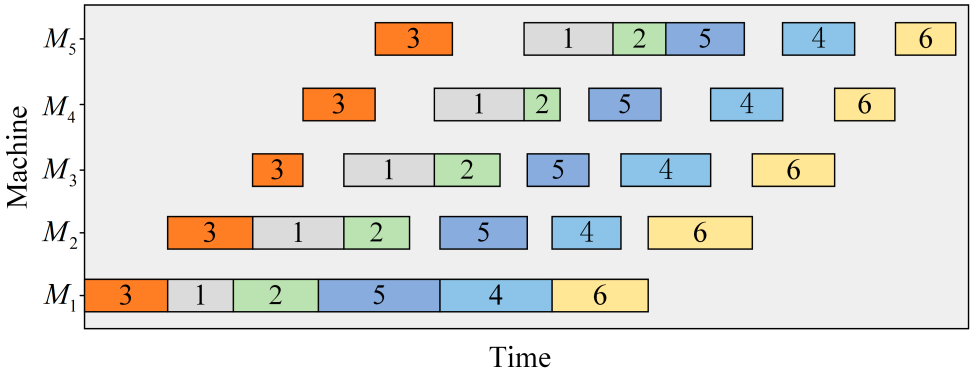


Figure 1. Gantt chart of the instance for PFSP.

2.2. Notations

The notations employed in this article are presented in Table 2.

Table 2. Descriptions of notations.

Notations	Definition
n	The total number of jobs
m	The total number of machines or the total number of job operations
i, i'	i and i' represent the different indices of jobs, $i, i' = 1, 2, \cdots, n$
j	The index of operations, $j = 1, 2, \cdots, m$
k	The index of machines, $k = 1, 2, \cdots, m$
q_i	The number of operations for job i
O_{ij}	The j_{th} operation of job i
p_{ij}	The processing time of job i on machine M_j
C_i	The completion time of job i
S_{ij}	The start time of operation O_{ij}
F_{ij}	The completion time of operation O_{ij}
T_{ij}^k	The processing time of operation O_{ij} on machine M_k, k is the index of machines
a_{ij}^k	The load unit energy consumption of operation O_{ij} on machine M_k, k is the index of machines
b^k	The idle unit energy consumption of machine M_k, k is the index of machines
E	The total energy consumption of all machines
$C_{\max}$	The maximum completion time
X_{ij}^k	$X_{ij}^k = 1$ if the operation O_{ij} is processed on machine M_k; otherwise, $X_{ij}^k = 0$, k is the index of machines
$X_{ii'}$	$X_{ii'} = 1$ if job i is the immediate predecessor operation of job i'; otherwise, $X_{ii'} = 0$
v_a, v_b, W_a, W_b	The learnable parameters of the DRL-PFSP model
e	The context vectors output by the encoder
d_t	The decoding vector from the decoder at time step t
a_t	The "attention" mask for the inputs at time step t
b_t	The context vector at time step t
$P(\rho_{t+1}\|\rho_1, \rho_2, \cdots, \rho_t, X_t)$	The highest probability value of the job, $\rho_1, \rho_2, \cdots, \rho_t$, represents the jobs that have been selected at time step t; X_t represents the jobs available at time step t
θ	The network parameters of the actor network
ϕ	The network parameters of the critic network

Table 2. *Cont.*

Notations	Definition
R^a	The actual reward value yielded by the actor network, for in-stance, a
N	The total number of training instances
$V\left(X_0^a;\phi\right)$	The expected reward of the critic network for each instance
x	The individuals of the population
N'	The population size
$\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_b)$	The weight vectors of the current subproblem, where b is the number of objectives for the problem
z^*	The reference points
H	The subdivision level for each objective coordinate
$g(x\|\boldsymbol{\lambda}, z^*)$	The Chebyshev aggregation function; x represents an individual in the population
RPD_1	The maximum percentage deviation of the maximum completion time
RPD_2	The maximum percentage deviation of the energy consumption
$C^*_{\max}$	The best $C_{\max}$ obtained by all compared algorithms
$C^{a\lg}_{\max}$	The average value of $C_{\max}$ obtained by algorithm $a\lg$
E^*	The best E obtained by all compared algorithms
$E^{a\lg}$	The average value of E obtained by algorithm $a\lg$

2.3. Optimization Objectives

The GPFSP model in this paper consists of optimization objectives and constraint equations, aiming to minimize the maximum completion time of jobs and the total energy consumption of machines. The detailed modeling steps of this model are as follows.

2.3.1. Makespan

The first optimization objective is to minimize the maximum completion time of jobs, which can be represented by Equation (1).

$$\min C_{\max} = \min\left\{\max\left(F_{ij}\right)\right\} = \min\left\{\max\left(S_{ij} + T_{ij}^k\right)\right\} \tag{1}$$

2.3.2. Total Energy Consumption

The second optimization objective is to minimize the total energy consumption E, which is the sum of the processing energy consumption and the idle energy consumption of all machines within the maximum completion time $C_{\max}$, as represented by Equation (2).

$$\min E = \min\left(\sum_{i,i'=1}^{n}\sum_{j=1}^{q_i}\sum_{k=1}^{m}\left[a_{ij}^k X_{ij}^k T_{ij}^k + b^k (S_{i'j} - C_{ij}) X_{ij}^k X_{i'j}^k X_{ii'}\right]\right) \tag{2}$$

To sum up, the model of GPFSP can be expressed as follows:

$$\min f_1 = \min C_{\max} \tag{3}$$

$$\min f_2 = \min E \tag{4}$$

which is subject to:

$$S_{ij} \geq F_{i(j-1)} \forall i, j > 1 \tag{5}$$

$$\sum_{k=1}^{m} X_{ij}^k = 1 \forall i, j \tag{6}$$

$$S_{ij} \geq 0 \forall i, j \tag{7}$$

$$F_{ij} > 0 \forall i, j \tag{8}$$

$$F_{ij} = T_{ij}^k + S_{ij} \forall i, j \tag{9}$$

$$C_{\max} \geq C_{ij} \forall i, j \tag{10}$$

$$X_{ij}^{k} \in \{0,1\} \forall i,j,k \tag{11}$$

$$X_{ii'} \in \{0,1\} \forall i,i' \tag{12}$$

Equations (3) and (4) individually denote the minimization of the maximum completion time and the total energy consumption for the GPFSP. Equation (5) indicates that a job can only commence the processing of the current operation after the completion of the previous operation. Equation (6) states that each machine can process only one job at any given time. Equations (7) and (8), respectively, indicate that the start time and completion time of a job must not be negative. Equation (9) indicates that, once a job starts processing on a machine, the processing cannot be interrupted. Equation (10) represents the constraint of the makespan. Equations (11) and (12) specify the permissible values for the decision variables.

3. The Solution Framework of the GDRL-MOEA/D

In this section, we propose a framework named GDRL-MOEA/D that combines DRL, MOEA/D, and the energy-saving strategy to address the GPFSP. The specific framework of GDRL-MOEA/D is shown in Figure 2, and the brief process is as follows.

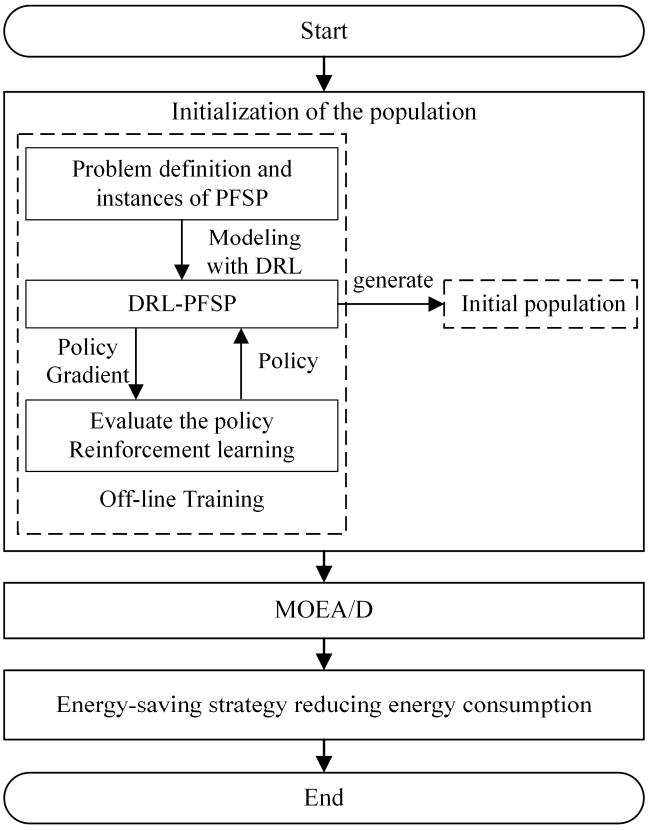


Figure 2. The framework of the GDRL-MOEA/D algorithm.

- First, with the objective of minimizing the maximum completion time, we applied an end-to-end deep reinforcement learning strategy (DRL-PFSP) to model the PFSP problem in Section 3.1 and systematically trained the model using the actor–critic algorithm. Once the DRL-PFSP model is trained, it can efficiently provide high-quality solutions for PFSP instances of varying sizes and complexities.
- Next, in Section 3.2, these solutions are used as the initial population for MOEA/D to further optimize the scheduling results, forming the DRL-MOEA/D approach. This integrated method improves the efficiency and adaptability of the solving process while maintaining the optimization quality of the solutions.

- Finally, in order to further reduce energy consumption without increasing the completion time, an innovative energy-saving strategy is proposed in Section 3.3. This strategy optimizes the energy consumption of the scheduling plans generated by DRL-MOEA/D, aiming to achieve more environmentally friendly and efficient workshop scheduling.

The details of each subsection are as follows.

3.1. The Structure of the DRL-PFSP

This section utilizes the PN neural network approach to model the PFSP and employs the actor–critic algorithm for training the model. Figure 3 depicts the comprehensive structure of this approach, which consists of the input layer, encoding layer, decoding layer, and attention layer. In brief, the encoder processes the input sequence (processing times of jobs in this paper) by converting each element into a hidden state vector, forming the encoded representation of the input. Then, at each step, the decoder compares the current decoding state with the hidden states from the encoder through the attention mechanism. The attention mechanism calculates relevance weights for each input position, and these weights are converted into a probability distribution using the softmax function, indicating which element the decoder should prioritize. Based on these weights, the decoder dynamically selects the output elements, gradually generating the complete output sequence. A detailed elucidation of each component will be provided in the subsequent sections.

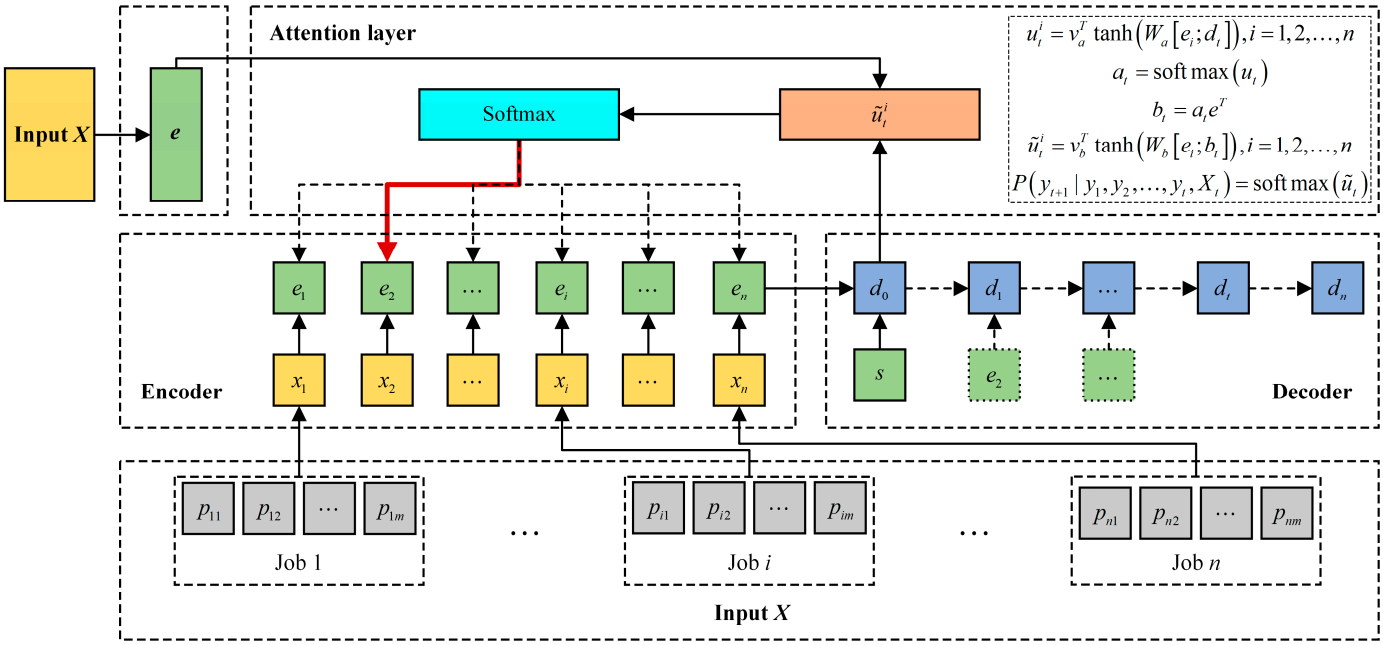


Figure 3. The structure for the DRL-PFSP. The red arrow indicates that the selected job at the current time step is job 2.

3.1.1. Input Layer

The input layer is constructed from a sequence of n fixed-dimensional vectors $X = \{x_i, i = 1, 2, \ldots, n\}$, which is employed as the input for the encoding layer. Here, n represents the number of jobs, and each x_i is composed of a tuple $\{p_i = (p_{i1}, p_{i2}, \ldots, p_{im})\}$. p_{ij} denotes the processing time of job i on machine M_j. Using the PFSP instance with six jobs and five machines as a case study, the structure of the encoder's input is depicted in Figure 4.

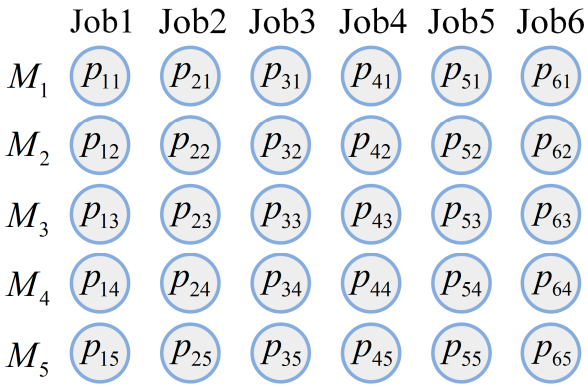


Figure 4. The input structure of the encoder network.

3.1.2. Encoding Layer

The function of the encoder is to recognize the input sequence and extract the features of the jobs, subsequently encoding this information into vectors of a fixed dimension. In conventional PN models, recurrent neural networks (RNNs) typically serve as encoders, tasked with the collection and integration of input data and its sequential order. This is especially important in tasks like machine translation, where the sequence of words is vital to the precision of the translation outcome. However, with regard to the job scheduling problem discussed in this paper, the input jobs are all independent entities without any temporal sequence connection between them. Therefore, the encoder in this study utilizes a straightforward one-dimensional (1-D) convolution embedding layer to replace the complex RNN structure, encoding the input data into a high-dimensional vector. This approach not only significantly reduces the model's complexity, but also effectively decreases the computational cost. And the number of input channels for a 1D convolutional layer corresponds to the dimensionality of the input data. Taking the PFSP with six jobs and five machines in Figure 4 as an example, there are five input channels of the convolutional layer. The encoder ultimately transforms the input data into a vector of dimensions $n \times d_h$, where n denotes the quantity of jobs and d_h represents the number of neurons in the hidden layer. A key point to note is that all jobs share the parameters of the convolutional neural network. This means that, regardless of the number of jobs, each one utilizes the same set of parameters to encode job information into a high-dimensional vector. Consequently, the encoder demonstrates good robustness in handling varying numbers of jobs.

3.1.3. Decoding Layer

The role of decoder is to accurately decode the high-dimensional vectors outputted by the encoder, which encapsulate the rich knowledge information of the input sequence. Its ultimate objective is to produce an output sequence that closely mirrors the original input sequence in both semantics and structure, ensuring no errors are introduced. Unlike the encoder, the decoder contains an RNN that summarizes the information from the previously selected jobs $\rho_1, \rho_2, \ldots, \rho_t$ to determine the next job ρ_{t+1}. The unique advantage of RNN lies in its inherent cyclic structure, which enables it to effectively retain and remember the output information processed previously. This allows it to consider historical context when dealing with sequential data, enhancing the model's ability to capture temporal dynamics. The decoder network structure employed in this paper is based on gated recurrent unit (GRU), a variant of RNN, which has fewer network parameters compared to the long short-term memory (LSTM) used in the original pointer network. In each decoding step t, the GRU decoder synthesizes the hidden state d_t, encapsulating the knowledge from the previous steps $\rho_1, \rho_2, \ldots, \rho_t$ with the input's encoded representation $e_1, e_2, \ldots, e_n$, jointly calculating the conditional probability $P(\rho_{t+1}|\rho_1, \rho_2, \cdots, \rho_t, X_t)$ for the next action selection. This calculation is performed through an attention mechanism, as shown in Figure 3.

3.1.4. Attention Layer

At each stage of decoding, the attention layer is responsible for receiving the context vectors e output by the encoder and the current decoding vector d_t from the decoder. This layer quantifies the association between each job and the potential next job, identifying the job with the highest association as the preferred candidate for the subsequent job. The specific calculation steps are detailed as follows [59]:

$$u_t^i = v_a^T \tanh(W_a[e_i; d_t]), i = 1, 2, \ldots, n \tag{13}$$

$$a_t = \text{softmax}(u_t) \tag{14}$$

$$b_t = a_t e^T \tag{15}$$

$$\widetilde{u}_t^i = v_b^T \tanh(W_b[e_i; b_t]), i = 1, 2, \ldots, n \tag{16}$$

$$P(\rho_{t+1}|\rho_1, \rho_2, \ldots, \rho_t, X_t) = \text{softmax}(\widetilde{u}_t) \tag{17}$$

where ";" denotes the combination of two vectors; v_a, v_b, W_a, and W_b all represent the learnable parameters of the model; and a_t and b_t, respectively, correspond to the "attention" mask for the inputs and the context vector at time step t. The softmax function is utilized to normalize both u_t and $\widetilde{u}_t$, resulting in the probability distribution for selecting each job i during step t. As depicted in Figure 3, job 2 exhibits the highest probability value $P(\rho_{t+1}|\rho_1, \rho_2, \ldots, \rho_t, X_t)$, and is therefore selected as the next job to be visited. During the training process, the model does not select the job with the highest probability in a greedy manner, but instead determines the next job by sampling from the probability distribution.

3.1.5. The Training Method for PFSP

This study utilizes a well-regarded actor–critic policy gradient technique to tackle the training challenges of PFSP. The policy gradient approach facilitates the continuous iterative training of the encoder, decoder, and attention mechanisms by accurately determining the gradients of the expected rewards for all trainable parameters, thereby enhancing the overall performance of the model.

The policy gradient approach in this article integrates two trainable networks—an actor network and a critic network—where the network parameters are designated as θ and ϕ, respectively. The actor network, termed PN, is not only responsible for generating a probability distribution to identify the optimal strategy for subsequent actions, but also adopts the method of randomly sampling actions from this distribution to explore potential solutions. Furthermore, the network evaluates the objective function R^a of the selected actions based on Equation (3) as a measure of reward, which in turn guides the training and optimization of the network. The role of the critic network is employed to evaluate the expected reward $V\left(X_0^a; \phi\right)$ of the solution acquired by the actor network, drawing upon the pertinent details of the specified problem. Moreover, the critic network maintains the same structural design as the encoder of PN, responsible for converting the hidden state of the encoder into the output of the critic network.

The model in this paper undergoes unsupervised training, and the training instances for PFSP should adhere to the distribution Φ_{M} during the training phase, where the parameter M represents the input features of jobs, such as the processing time of jobs. In order to train the model parameters of the actor and critic networks, we randomly select N instances from distribution Φ_{M} to construct the training dataset. For each training instance, the actor network is tasked with devising the ultimate scheduling plan and computing the corresponding reward. Concurrently, the critic network estimates the expected reward for each instance. After these steps are completed, the actor and critic network parameters are updated using the policy gradient method, as specified in Equations (18) and (19), respectively.

$$d\theta = \frac{1}{N} \sum_{a=1}^{N} (R^a - V(X_0^a; \phi)) \nabla_\theta \log P(Y^a | X_0^a) \tag{18}$$

$$d\phi = \frac{1}{N}\sum_{a=1}^{N} \nabla_{\phi}\left(R^a - V\left(X_0^a;\phi\right)\right)^2 \tag{19}$$

where R^a symbolizes the actual reward yielded by the actor network, and $V\left(X_0^a;\phi\right)$ signifies the reward approximation calculated by the critic network for instance N. Moreover, the critic network's input data comprise the processing times of jobs on the machines. Its architecture is composed of three subsequent convolutional layers that follow the encoding network, with the final layer aggregating the output of the preceding convolutional layer to derive the estimated reward value $V\left(X_0^a;\phi\right)$ for each instance. And the training procedure is depicted in Algorithm 1.

Algorithm 1: The Framework of the Actor–Critic Training Algorithm.

Input: θ: the parameters of the actor network; ϕ: the parameters of the critic network
Output: The optimal parameters θ, ϕ

for $iter = 1, 2, 3, \cdots$ **do**
  Generate N instances based on PFSP
  for $a = 1, 2, \cdots, N$ **do**
    $t = 0$
    while the jobs have not been fully accessed **do**
      select the next job ρ_{t+1}^a according to $P\left(\rho_{t+1}^a \middle| \rho_1^a, \cdots, \rho_t^a, X_t^a\right)$
      $t = t + 1$ and update X_t^a
    end while
    compute the reward R^a: $R^a = C_T$
  end for
  Calculate the strategy gradient of actor network and critic network:
  $d\theta = \frac{1}{N}\sum_{a=1}^{N}\left(R^a - V\left(X_0^a;\phi\right)\right)\nabla_{\theta}\log P\left(Y^a \middle| X_0^a\right)$
  $d\phi = \frac{1}{N}\sum_{a=1}^{N}\nabla_{\phi}\left(R^a - V\left(X_0^a;\phi\right)\right)^2$
  Optimize the network parameters of actor network and critic network based on strategy gradient:
  $\theta = \theta + \eta d\theta$
  $\phi = \phi + \eta d\phi$
return θ, ϕ
end for

3.2. The Algorithm of MOEA/D

The optimization strategy of MOEA/D decomposes complex multi-objective optimization problems into multiple single-objective subproblems and optimizes them in parallel. Each subproblem is associated with a specific weight vector, enabling the algorithm to simultaneously optimize solutions in multiple directions. A key advantage of this approach is its ability to leverage neighborhood information, improving solution diversity and convergence efficiency by optimizing neighboring subproblems. This method greatly reduces the scope of ineffective searches, allowing for rapid convergence to the Pareto front within limited computational resources. Furthermore, the decomposition strategy of MOEA/D is highly scalable and can be seamlessly integrated with various optimization techniques, demonstrating exceptional robustness and stability, particularly in high-dimensional problems. Therefore, this paper adopts a strategy that combines MOEA/D with deep reinforcement learning to solve the PFSP.

Upon the completion of the training of the DRL-PFSP model in Section 3.1, it can quickly provide relatively good solutions for PFSP problems of different scales. In this section, we use these solutions as the initial population for the MOEA/D algorithm to further enhance the optimization capabilities of the algorithm, which is the DRL-MOEA/D method proposed in this paper. The fundamental procedure of the algorithm is outlined as follows:

3.2.1. Evaluation of Adaptation Values

In the MOEA/D framework, the Chebyshev aggregation function is typically employed to assess subproblems, as depicted in Equation (20).

$$g(x|\boldsymbol{\lambda}, z^*) = \max\{\lambda_b | f_b(x) - z_b^*\} \tag{20}$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_b)$ represents the weight vectors of the current subproblem, b represents the number of objectives in the multi-objective optimization problem, and $z_b^* = \min\{f_b(x)|x \in X\}$ is the reference point.

3.2.2. Weight Vectors

In the MOEA/D algorithm, the weight vectors $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_b)$ typically adopt uniformly distributed weight vectors, which are generated based on a user-defined integer H, and H represents the subdivision level for each objective coordinate. Simultaneously, the weight vectors must not have duplicate values; hence, the number of weight vectors should meet the requirement of $N'' = C_{H+b-1}^{b-1}$. Furthermore, the weight vectors should satisfy Equations (21) and (22). In this paper, the weight vector $\boldsymbol{\lambda}$ is $((0,1),(0.01,0.99),(0.02,0.98),\ldots,(0.99,0.01),(1,0))$.

$$\lambda_1 + \lambda_2 + \ldots + \lambda_b = 1 \tag{21}$$

$$\lambda_c \in \left\{0, \frac{1}{H}, \frac{2}{H}, \ldots, \frac{H}{H}\right\}, c = 1, 2, \ldots, b \tag{22}$$

3.2.3. Neighborhood

In the MOEA/D algorithm framework, the concept of a neighborhood defines the specific scope for selection, evolution, and update operations within the population. The construction of neighborhoods is initialized by calculating the Euclidean distances between the weight vectors. For a specific weight vector λ_c, the process of determining its neighborhood involves first calculating the Euclidean distances between λ_c and all other weight vectors, and then selecting the T weight vectors with the smallest distances to form the neighborhood of λ_c. The process of population evolution relies on a mutual collaboration and information exchange among subproblems within the neighborhood, thereby promoting the collaborative evolution of the entire population.

3.2.4. The Process of MOEA/D

The flowchart of MOEA/D is shown in Figure 5, and the specific process is as follows:
Step 1: Initialization.

- Refine the initialization of N' uniformly distributed reference weight vectors $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_b)$, and then calculate the Euclidean geometric distance between vector λ_c and each of these weight vectors. Subsequently, filter out the T weight vectors with the smallest Euclidean distances as the neighborhood of λ_c and store the neighborhood information in the neighborhood matrix, i.e., the neighborhood of weight vector λ_c is represented as $B(c) = \{c_1, c_2, \ldots, c_T\}$, $c = 1, 2, \ldots, N'$;
- Initialize population $X = \left(x^1, x^2, \ldots\ldots, x^{N'}\right)$ based on the DRL-PFSP model, and calculate the fitness value $FV^c = F(x^c)$ for each x^c, $c = 1, 2, \ldots, N'$;
- Initialize reference points $z = (z_1, z_2, \ldots, z_b)$, where $z_d = \min\limits_{1 \le c \le N'} f_d(x^c)$, $d = 1, 2, \ldots, b$.

Step 2: Perform evolutionary operations on individual x^c in the population, $c = 1, 2, \ldots, N'$.

- Select two random individuals from the neighborhood $B(c)$ of individual x^c, and generate a descendant individual y using crossover and mutation operations.
- If $f_d(y) < z_d$, then update the reference point $z_d = f_d(y)$, $d = 1, 2, \ldots, b$.
- Update the neighborhood solutions, that is, for $t \in B(c)$, if $g\left(y|\lambda^t, z\right) \le g\left(x^t|\lambda^t, z\right)$, then $x^t = y$, $FV^t = F(y)$.

Step 3: If the termination condition is met, the algorithm stops and outputs the optimal solution; otherwise, continue with step 2.

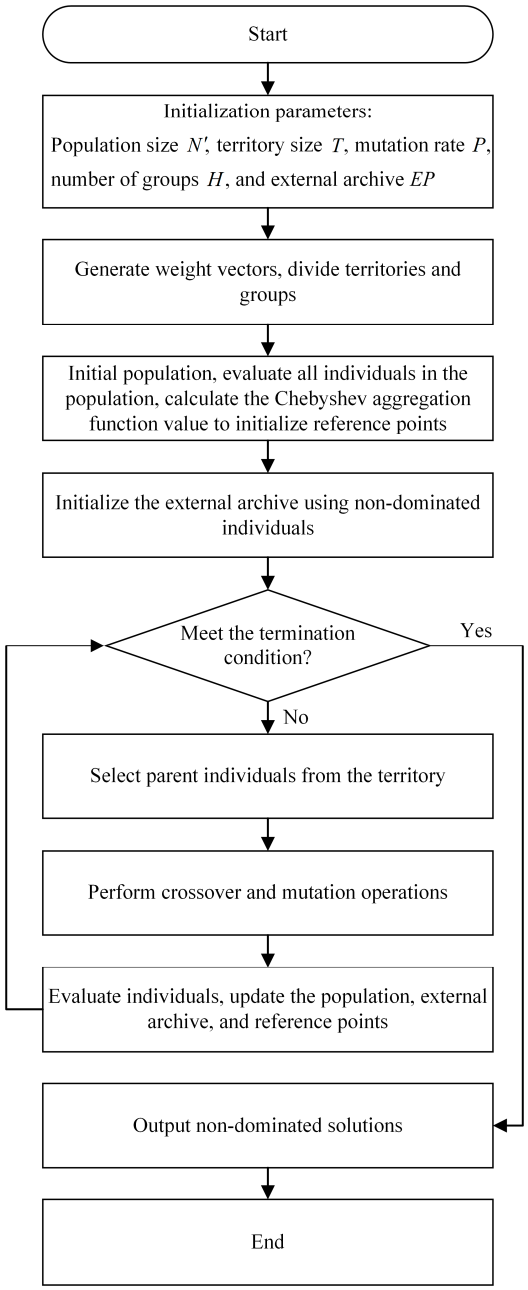


Figure 5. The flowchart of MOEA/D.

3.3. The Energy-Saving Strategy

In Section 3.2, we successfully obtained the relatively optimal solution for the completion time of the PFSP. To further optimize energy consumption, this paper proposes a novel energy-saving strategy aimed at effectively reducing overall energy consumption while keeping the longest completion time unchanged. As indicated by the objective function (4), the energy consumption of a machine is composed of the energy used during processing and the energy consumed while idle. Specifically, the processing energy consumption E^{pro} is the product of the job processing time and the machine's processing power, while the idle energy consumption E^{id} is the product of the machine's idle time and the idle power of the machine. Given the fixed processing time of jobs, the machine's processing power, and the machine's idle power, the key to reducing energy consumption lies in minimizing

the machine's idle time to decrease idle energy consumption, thereby achieving the goal of reducing the overall energy consumption of all machines.

Therefore, this paper proposes a job postponement strategy to reduce the idle time of machines. However, not all jobs are suitable for delayed processing, and those jobs can be postponed, but must meet the following two conditions: (1) The completion time of the job that can be postponed must be earlier than the start time of the job immediately following it, i.e., $C_{ij} < S_{(i+1)j}$; and (2) the completion time of the current operation for the job that can be postponed must be earlier than the start time of the next operation, i.e., $C_{ij} < S_{i(j+1)}$. Specifically, the process begins by traversing all jobs on machine M_m in reverse order based on their start or completion times, performing the postponement operation on jobs that meet condition (1). Subsequently, the same approach is used to traverse all jobs on machine M_{m-1}, applying the postponement operation to jobs that satisfy both condition (1) and condition (2). This process continues in the same manner until the traversal reaches the jobs on machine M_1, at which point the loop ends.

To vividly and concretely understand the energy-saving strategy proposed in this paper, we take the job sequence depicted in Figure 1 as an example to elaborate. First, all jobs on Machine M_5 are traversed in reverse order based on their start or completion times, and jobs 4, 5, 2, 1, and 3 are found to meet condition (1), so the postponement operation is performed on these jobs. Next, using the same method, the jobs on Machine M_4 are traversed, and jobs 4, 5, 2, 1, and 3 are found to meet both condition (1) and condition (2), so the postponement operation is applied to these jobs again. This process continues with the jobs on Machine M_3 and Machine M_2, where each job is checked to see if it meets both condition (1) and condition (2); if they do, the postponement is applied. Finally, the loop ends after traversing the jobs on Machine M_1. The enhanced Gantt chart, as depicted in Figure 6b, is contrasted with Figure 6a to illustrate that the job postponement strategy has greatly reduced the total idle time of the machines, consequently diminishing the overall energy consumption. Consequently, the energy-saving strategy presented in this paper effectively reduces the overall energy consumption.

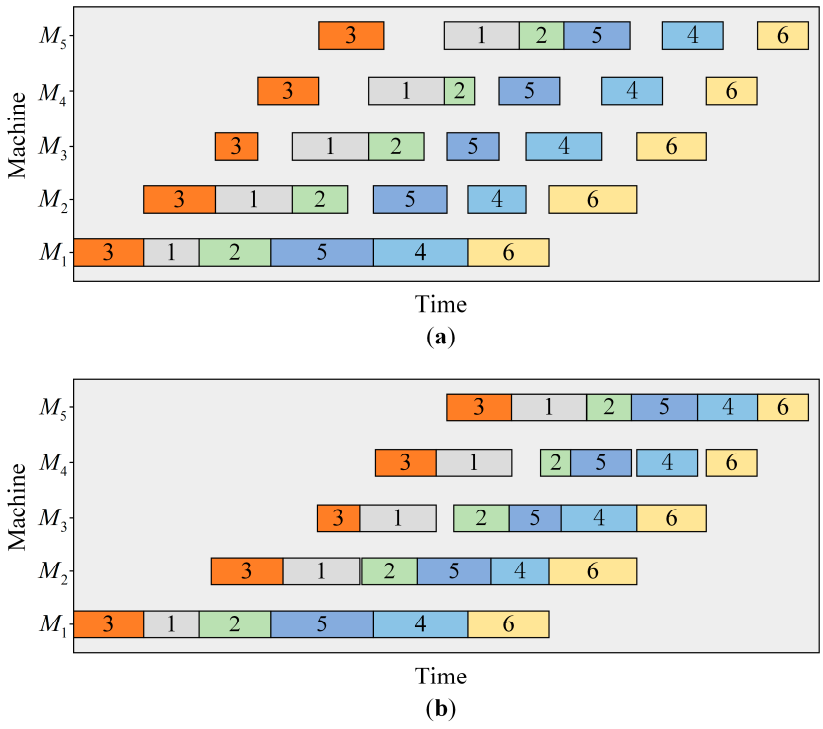


Figure 6. An example of the energy-saving strategy for PFSP. (**a**) represents the Gantt chart without using the energy-saving strategy; (**b**) represents the Gantt chart after using the energy-saving strategy.

4. Numerical Experiments

4.1. Experimental Settings

To validate the effectiveness and practicality of the proposed GDRL-MOEA/D framework and the energy-saving strategy for solving the GPFSP in this paper, a comprehensive series of experiments is conducted. All experiments are developed using Python 3.7 on an Intel Core i7 CPU/2.8 GHz PC and a GTX 2060.

To train the DRL-PFSP model in this study, 100,000 instances are randomly generated in each epoch, with each instance having 50 jobs and the number of machines being an integer in the range of $[5,\ 20]$. The processing times of jobs are generated stochastically from a uniform distribution across the [0, 1] interval. This strategy not only significantly improves the computational efficiency of the model, but also enhances its versatility in accommodating PFSPs of diverse scales, thereby significantly bolstering the model's robustness. The entire training process of the model requires approximately 300 h. To assess the scheduling performance of the proposed GDRL-MOEA/D algorithm, this paper conducts comparative experiments with the MOEA/D, NSGA-II, marine predators algorithm (MPA) [60], sparrow search algorithm (SSA) [61], artificial hummingbird algorithm (AHA) [62], and seagull optimization algorithm (SOA) [63]. All algorithms are tested using instances with the number of jobs n belonging to set $\{50, 100, 150, 200\}$ and the number of machines m belonging to set $\{5, 6, 7, 10, 15, 20\}$. Therefore, each algorithm corresponds to 24 different combinations of (n, m) scale variations. For each scale, the processing times of jobs are randomly generated, adhering to a uniform distribution ranging from 0 to 1. And all comparison algorithms are run independently 15 times. Additionally, to streamline the computation of energy consumption, the machines' processing power is set to 1, while the idle power is assigned a value of 0.2.

To maintain equity in the comparative analysis of algorithms, the parameters for the benchmark algorithms are configured as follows: NSGA-II is assigned a population size of 100, a crossover rate of 0.7, a mutation rate of 0.05, and a total of 200 iterations; MOEA/D is equipped with a population size of 100, a neighborhood size of 15, a mutation rate of 0.05, a weight vector parameter of 99, and a total of 200 iterations; MPA is assigned a population size of 100, an initial fish aggregation device influence value of 0.2, a fast movement probability of 0.5, and a total of 200 iterations; SSA is configured with a population size of 100, an alert threshold of 0.6, a discoverers proportion of 0.7, an aware sparrows proportion of 0.2, and a total of 200 iterations; AHA is equipped with a population size of 100 and a total of 200 iterations; and SOA is assigned a population size of 100 and a total of 200 iterations.

The relative percentage deviation is used as the evaluation metric for all algorithms, as shown in Equations (23) and (24), where RPD_1 and RPD_2 represent the maximum percentage deviation of the maximum completion time and the maximum percentage deviation of the energy consumption, respectively. $C^*_{\max}$ denotes the best $C_{\max}$ obtained by all compared algorithms, $C^{alg}_{\max}$ is the average value of $C_{\max}$ obtained by algorithm $a$lg, E^* is the best E obtained by all compared algorithms, and E^{alg} represents the average value of E obtained by algorithm $a$lg.

$$RPD_1 = \left(C^{alg}_{\max} - C^*_{\max}\right) / C^*_{\max} \times 100 \tag{23}$$

$$RPD_2 = \left(E^{alg} - E^*\right) / E^* \times 100 \tag{24}$$

4.2. Parameter Settings of the DRL-PFSP Model

During the training process, the parameters of the network model are set as shown in Table 3. The variable D_{input} represents the dimensionality of the input data. The decoder employs a single-layer GRU RNN with a hidden layer size of 128 units. Similarly, the critic network is also configured with a hidden layer size of 128 units. Both the actor and critic networks are trained using the Adam optimizer, with a learning rate η of 0.0001 and a batch size of 256 for each training iteration.

Table 3. The parameter settings of the model.

Actor Network (Pointer Network)	Critic Network
Encoder: 1D-Conv(D_{input}, 128, kernel size = 1, stride = 1)	1D-Conv (D_{input}, 128, kernel size = 1, stride = 1) 1D-Conv (128, 20, kernel size = 1, stride = 1)
Decoder: GRU (hidden size = 128, number of layers = 1) Attention (No hyper parameters)	1D-Conv (20, 20, kernel size = 1, stride = 1) 1D-Conv (20, 1, kernel size = 1, stride = 1)

4.3. Experimental Results and Discussions

4.3.1. The Effectiveness of Initializing the Population Based on DRL-PFSP

In Section 3.1, we employ an end-to-end deep reinforcement learning method to model the PFSP and train the model using the actor–critic algorithm. Once training is complete, the model is capable of swiftly generating solutions based on the scale of the PFSP, which are then used as the initial population for the MOEA/D algorithm, thereby forming the DRL-MOEA/D method. To validate the effectiveness of this strategy, this section compares it with the traditional MOEA/D algorithm in terms of the maximum completion time and energy consumption objectives. The averages of the *RPD* value and computation time for each algorithm across different problem scales are presented in Table 4. To more intuitively observe the comparison results, a graphical representation of the outcomes for both methods has been created, as shown in Figures 7 and 8. The numbers 1 to 24 in Figure 8 on the *x*-axis represent different scales from (50, 5) to (200, 20).

Table 4. The average *RPD* value for each algorithm at each scale.

(*n*, *m*)	The Average *RPD* Value for Each Algorithm				The Computation Time of Algorithm/s	
	DRL-MOEA/D		MOEA/D		DRL-MOEA/D	MOEA/D
	RPD_1	RPD_2	RPD_1	RPD_2		
(50, 5)	0.5174	0.6032	0.6480	0.6559	76.2302	78.3117
(50, 6)	1.1057	0.1115	1.7287	0.1775	88.5214	90.2976
(50, 7)	1.9905	0.2166	3.6439	0.2200	104.2893	105.8324
(50, 10)	1.7088	0.1298	2.2966	0.4152	145.2546	148.1882
(50, 15)	2.5123	0.5662	8.4098	0.5736	218.6872	219.9746
(50, 20)	3.1292	0.5978	6.6061	0.6120	291.0637	294.0099
(100, 5)	0.2683	0.1329	1.1107	0.1424	142.1222	154.5248
(100, 6)	0.5515	0.1939	0.9147	0.1727	168.0217	172.2160
(100, 7)	1.4165	0.1718	2.2110	0.1727	205.3689	200.6013
(100, 10)	1.5976	0.1872	4.2494	0.2198	291.2315	293.9304
(100, 15)	1.8348	0.2842	5.0751	0.3575	450.0200	441.8432
(100, 20)	2.2150	0.4492	7.2053	0.4710	592.2356	592.6549
(150, 5)	0.3271	0.1142	0.8654	0.1200	206.2297	216.773
(150, 6)	0.4795	0.1093	1.2698	0.1323	255.2442	254.7261
(150, 7)	1.0168	0.0675	2.0241	0.2564	298.3484	316.9811
(150, 10)	0.4252	0.1607	1.2371	0.1682	439.0293	441.1019
(150, 15)	1.3355	0.1544	4.7322	0.3040	669.9642	670.6341
(150, 20)	1.6796	0.2939	5.1543	0.3693	912.1419	920.1188
(200, 5)	0.5483	0.0517	0.8484	0.0765	276.1188	282.2372
(200, 6)	0.2385	0.0458	1.3282	0.0558	338.6582	342.7295
(200, 7)	0.5096	0.0638	1.6252	0.0943	396.4272	400.0306
(200, 10)	0.4651	0.1128	2.6457	0.2268	587.6343	589.5134
(200, 15)	0.9884	0.2387	4.6916	0.4121	912.2259	914.6555
(200, 20)	2.0133	0.1796	6.2465	0.3529	1271.5979	1254.9765
AVG	1.2031	0.2182	3.1987	0.2816	389.0278	391.5359

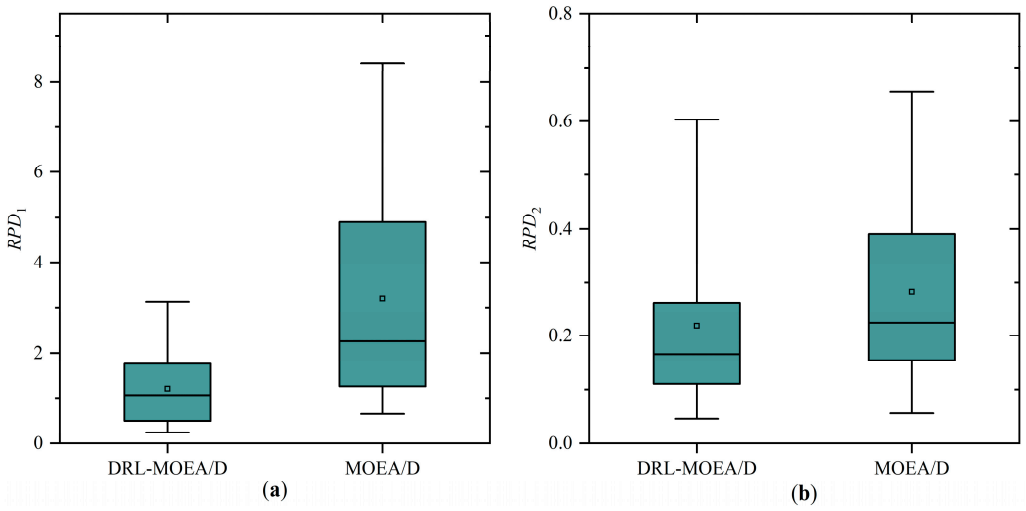


Figure 7. The boxplot of the average *RPD* value. (**a**) represents the boxplot of the average *RPD* value for the maximum completion time; (**b**) represents the boxplot of the average *RPD* value for the energy consumption.

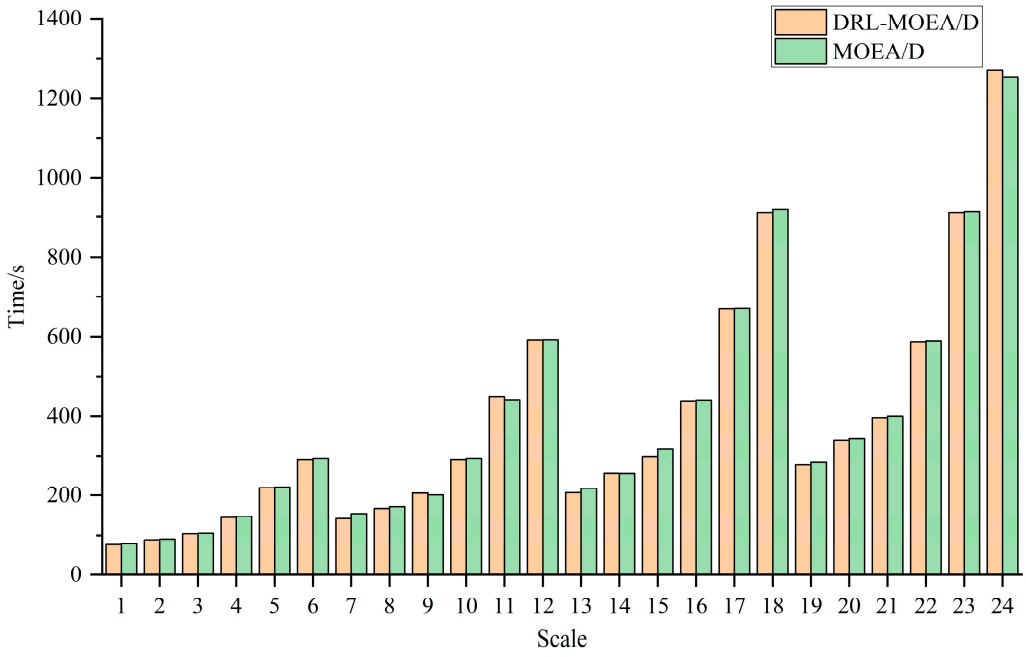


Figure 8. The computation time of two algorithms for each scale.

As indicated in Table 4 and Figure 7, the DRL-MOEA/D algorithm has a lower average *RPD* value for both the maximum completion time and energy consumption across all scales compared to the MOEA/D algorithm, demonstrating that the DRL-MOEA/D algorithm outperforms the MOEA/D algorithm in terms of performance. Furthermore, as observed in Table 4 and Figure 8, the computational times of the two algorithms are relatively close, but the solution time of the DRL-MOEA/D algorithm is generally lower than that of the MOEA/D algorithm across most problem scales. This is because the DRL-PFSP model, once trained, can quickly produce relatively optimized solutions to serve as the initial population for the DRL-MOEA/D algorithm. Therefore, it is evident that using the DRL-PFSP model to enhance the initial population strategy of MOEA/D not only significantly improves the solution quality of the MOEA/D algorithm, but also achieves a slight increase in solution speed.

4.3.2. The Effectiveness of the Energy-Saving Strategy

In Section 3.3, to further optimize the energy consumption objective, we proposed a job postponement strategy for energy saving. The purpose of this section is to validate the effectiveness of this strategy. A comparative experiment is conducted between the GDRL-MOEA/D algorithm with the energy-saving strategy and the DRL-MOEA/D algorithm without the energy-saving strategy, as discussed in the previous section, focusing on the energy consumption objective. The average energy consumption *RPD* value and average computation time for both algorithms are presented in Table 5 and Figure 9. As shown in Table 5 and Figure 9a, the GDRL-MOEA/D algorithm significantly outperforms the DRL-MOEA/D algorithm in terms of average energy consumption *RPD* value across all problem scales. Additionally, Table 5 and Figure 9b indicate that the performance difference in solution time between the two algorithms is minimal, with both being relatively close, although the DRL-MOEA/D algorithm has a slight edge.

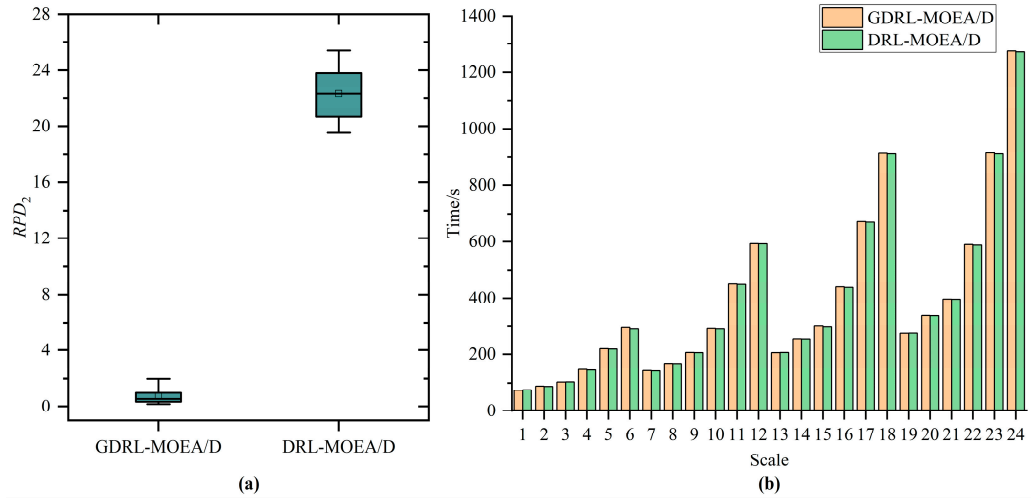


Figure 9. The boxplot of the average *RPD* value and the average computation time of algorithms. (**a**) represents the boxplot of the average *RPD* value; (**b**) represents the computation time of two algorithms for each scale.

Table 5. The average *RPD* value and average computation time for each algorithm at each scale.

(*n*, *m*)	The Average *RPD* Value for Each Algorithm		The Computation Time of Each Algorithm/s	
	GDRL-MOEA/D	DRL-MOEA/D	GDRL-MOEA/D	DRL-MOEA/D
	RPD_2	RPD_2		
(50, 5)	1.6924	20.1261	75.1254	76.2302
(50, 6)	1.6876	24.3180	89.2452	88.5214
(50, 7)	1.9866	24.5283	103.5411	104.2893
(50, 10)	0.9362	19.7679	147.3512	145.2546
(50, 15)	0.8797	24.7655	219.5412	218.6872
(50, 20)	0.4857	20.6822	295.6234	291.0637
(100, 5)	0.4237	19.5342	143.2415	142.1222
(100, 6)	1.4500	23.6355	168.6145	168.0217
(100, 7)	0.3360	22.3653	206.1235	205.3689
(100, 10)	0.3590	20.6560	292.2514	291.2315
(100, 15)	0.6011	25.3751	451.3547	450.0200
(100, 20)	0.3944	21.6173	593.1045	592.2356
(150, 5)	0.9851	20.5960	205.6841	206.2297
(150, 6)	1.2060	21.8462	255.4562	255.2442
(150, 7)	0.7356	23.7286	301.2514	298.3484

Table 5. *Cont.*

(*n*, *m*)	The Average *RPD* Value for Each Algorithm		The Computation Time of Each Algorithm/s	
	GDRL-MOEA/D	DRL-MOEA/D	GDRL-MOEA/D	DRL-MOEA/D
	RPD_2	RPD_2		
(150, 10)	0.3196	22.4336	441.0145	439.0293
(150, 15)	0.3807	22.2747	672.5241	669.9642
(150, 20)	0.1680	23.8685	913.8121	912.1419
(200, 5)	0.9916	19.6915	275.6581	276.1188
(200, 6)	0.4009	21.2566	339.2514	338.6582
(200, 7)	0.7289	22.9484	396.5412	396.4272
(200, 10)	0.2848	21.9042	589.5418	587.6343
(200, 15)	0.2600	25.3070	915.2564	912.2259
(200, 20)	0.1676	22.8183	1274.4514	1271.5979
AVG	0.7442	22.3352	390.2317	389.5359

4.3.3. Comparison with Other Algorithms

To further validate the performance of the proposed GDRL-MOEA/D algorithm, this section conducts comparative experiments with the classic multi-objective optimization algorithms MOEA/D and NSGA-II, as well as the latest metaheuristic algorithms MPA, SSA, AHA, and SOA. These algorithms are briefly summarized in Table 6.

Based on the descriptions of the algorithms compared in Table 6, it is evident that these algorithms exhibit good robustness and global search capabilities in solving complex problems such as production scheduling and multi-objective decision making, effectively finding high-quality solutions. Therefore, selecting these algorithms for comparison with the GDRL-MOEA/D algorithm proposed in this paper can provide a more comprehensive validation of the performance of GDRL-MOEA/D.

Table 6. Brief summary of algorithms.

Algorithms	Descriptions	Characteristic	Application Scenarios
NSGA-II	An advanced genetic algorithm for solving multi-objective optimization problems, which introduces improvements such as fast non-dominated sorting, crowding distance estimation, and elitist strategies based on NSGA, significantly enhancing the algorithm's efficiency and solution quality.	NSGA-II achieves precise sorting of solutions with lower computational complexity and maintains population diversity through crowding distance.	Production scheduling, engineering design, path planning, power system optimization, etc.
MPA	A metaheuristic algorithm designed based on the predatory behavior of marine predators, which simulates the dynamic behavior of predators during the processes of hunting and migration to balance global search and local exploitation.	MPA excels at handling optimization problems with complex search spaces, effectively avoiding local optima and demonstrating strong global optimization capabilities.	Engineering design, logistics optimization, production scheduling, etc.
SSA	A metaheuristic algorithm based on sparrow foraging behavior, aimed at solving complex optimization problems by simulating the collaboration and decision-making mechanisms of sparrows during the foraging process.	SSA is simple in structure and easy to implement, with good optimization ability and convergence speed.	Production scheduling, resource allocation, etc.

Table 6. *Cont.*

Algorithms	Descriptions	Characteristic	Application Scenarios
AHA	A metaheuristic algorithm that simulates behaviors such as guiding foraging, area foraging, and migratory foraging of hummingbirds.	AHA demonstrates high adaptability, allowing it to dynamically adjust search strategies based on the scale and complexity of the problem, exhibiting good flexibility and robustness.	Production scheduling, path planning, multi-objective decision making, etc.
SOA	A metaheuristic algorithm based on the migration and predatory behavior of seagulls, aimed at solving complex global optimization problems.	SOA is characterized by simplicity and ease of implementation, strong global search capabilities, and broad applicability.	Engineering optimization, data mining, production scheduling, image processing, etc.

The results of the comparative experiments for the algorithms are shown in Tables 7 and 8, and Figures 10 and 11. As is shown in Table 7 and the boxplot of Figure 10, the GDRL-MOEAD algorithm achieved the best RPD_1 and RPD_2 indicators across all test sets compared to the other six algorithms, particularly excelling in the RPD_2 indicator, where its performance far exceeded that of the other algorithms, verifying the effectiveness of the energy-saving strategy proposed in this paper. Additionally, Table 7 and Figure 10 demonstrate that the MOEA/D algorithm outperformed the other algorithms in both RPD_1 and RPD_2, further validating the rationale behind combining the DRL and MOEA/D algorithms. Among the remaining five algorithms, MPA and AHA showed relatively close performances in RPD_1, outperforming the other three algorithms, while in RPD_2, the five algorithms exhibited similar performances. A bar chart was created based on the computation time data of the seven algorithms for the different problem sizes shown in Table 8, as illustrated in Figure 11. As visually depicted in Figure 11, the computation times of all seven algorithms increased as the problem size grew, and the solution speeds of the algorithms were relatively close at each scale, with the NSGA2 algorithm performing slightly better. Overall, the computation times of all the algorithms were within an acceptable range.

Table 7. The average *RPD* value for each algorithm at each scale.

(*n*, *m*)	The Average *RPD* Value for Each Algorithm													
	GDRL-MOEA/D		MOEA/D		NSGA-II		MPA		SSA		AHA		SOA	
	RPD_1	RPD_2	RPD_1	RPD_2	RPD_1	RPD_2	RPD_1	RPD_2	RPD_1	RPD_2	RPD_1	RPD_2	RPD_1	RPD_2
(50, 5)	0.5349	1.6924	0.6952	20.1942	4.0498	20.8703	3.1144	20.7063	6.0056	21.2036	4.1135	20.8055	7.2949	21.3956
(50, 7)	0.7269	1.6876	1.7416	24.4001	4.6646	24.8379	2.8409	24.6383	5.2277	24.9639	3.0602	24.7468	6.9934	25.1091
(50, 10)	2.1224	1.9866	3.6366	24.5325	11.5032	25.3896	9.1224	25.2037	13.9339	25.6607	11.0271	25.7350	15.9728	26.1701
(50, 15)	1.7289	0.9362	2.2645	20.1093	10.4516	21.4295	7.5891	21.1843	11.2122	21.5983	9.8977	21.1774	12.8126	21.9546
(50, 20)	2.4884	0.8797	8.3774	24.7746	16.3570	26.5221	14.9996	25.9764	18.1457	26.7899	16.1320	26.4391	20.4967	27.3059
(100, 5)	2.7940	0.4857	6.5506	20.6979	13.8905	22.7676	12.5223	22.4634	14.1222	23.0694	10.8028	22.4707	14.2957	23.8263
(100, 6)	0.4932	0.4237	1.0922	19.5454	4.4411	19.8794	2.8800	19.7333	5.0479	20.1175	2.9957	19.7542	5.5589	20.3161
(100, 7)	0.3513	1.4500	0.8593	23.6094	5.0790	24.2241	3.3404	24.0738	5.1484	24.2105	3.4077	24.2649	4.6836	24.0524
(100, 10)	1.2530	0.3360	2.2087	22.3664	8.0974	23.1223	7.3630	23.0017	10.3049	23.2399	8.6940	23.2307	9.4021	23.9177
(100, 15)	1.6883	0.3590	4.2494	20.6953	12.6673	21.9838	10.6005	21.6971	13.3050	22.2634	9.8033	21.8840	13.4569	22.4374
(100, 20)	1.3746	0.6011	5.0405	25.4667	13.1514	26.8720	11.5269	26.5528	14.1177	27.0574	11.9604	26.9201	14.3532	27.0540
(150, 5)	2.1456	0.3944	7.1890	21.6437	13.6901	23.0835	12.1343	22.9091	14.3731	23.3640	12.8380	23.1266	15.0228	23.3508
(150, 6)	0.5595	0.9851	1.3563	20.6030	3.3165	20.8377	2.5156	20.7410	4.3366	20.8875	2.8546	20.7496	3.1512	20.8535
(150, 7)	0.4932	1.2060	1.2673	21.8742	4.0876	22.2464	2.8358	22.1585	3.7450	22.2795	3.3969	22.2003	4.3025	22.4164
(150, 10)	1.0679	0.7356	2.0229	23.7572	4.9376	24.1118	3.8312	24.0326	4.8591	24.1931	3.4368	24.1881	4.6581	24.3320
(150, 15)	0.6039	0.3196	1.2337	22.4428	6.0427	23.1658	4.8818	22.9335	6.9101	23.2386	5.2370	23.1630	7.4584	23.3205
(150, 20)	1.5000	0.3807	4.7201	22.4572	11.1145	23.4994	9.0835	23.0333	11.9827	23.7174	8.3422	23.0910	12.1749	23.8704
(200, 5)	1.7873	0.1680	5.1543	23.9617	11.6012	25.2615	10.6978	25.4128	12.3295	25.5058	10.1595	25.4325	13.6903	25.5540
(200, 6)	0.4653	0.9916	0.8325	19.7213	1.5293	20.0045	1.2729	19.8921	1.9503	20.0221	1.2842	19.8486	1.8093	20.1448
(200, 7)	0.3032	0.4009	1.3001	21.2688	2.6068	21.5087	1.8491	21.3293	3.3536	21.5723	1.9924	21.4549	3.9930	21.8319
(200, 10)	0.4383	0.7289	1.6349	22.9860	5.5199	23.4555	3.6985	23.1576	6.2797	23.4693	4.5586	23.0760	6.5712	23.5114
(200, 15)	0.5598	0.2848	2.6148	22.0431	8.5497	22.8128	7.0618	22.5205	9.8386	22.9871	7.2791	22.4929	9.6581	22.9751
(200, 20)	1.0207	0.2600	4.6460	25.5237	10.6289	26.6426	8.9969	26.5614	10.9768	26.7788	9.5384	26.6343	11.2898	26.7975
AVG	2.0876	0.1676	6.2365	23.0307	11.6794	24.1396	10.4965	23.8991	12.5157	24.2843	10.1338	23.9889	12.8426	24.2816

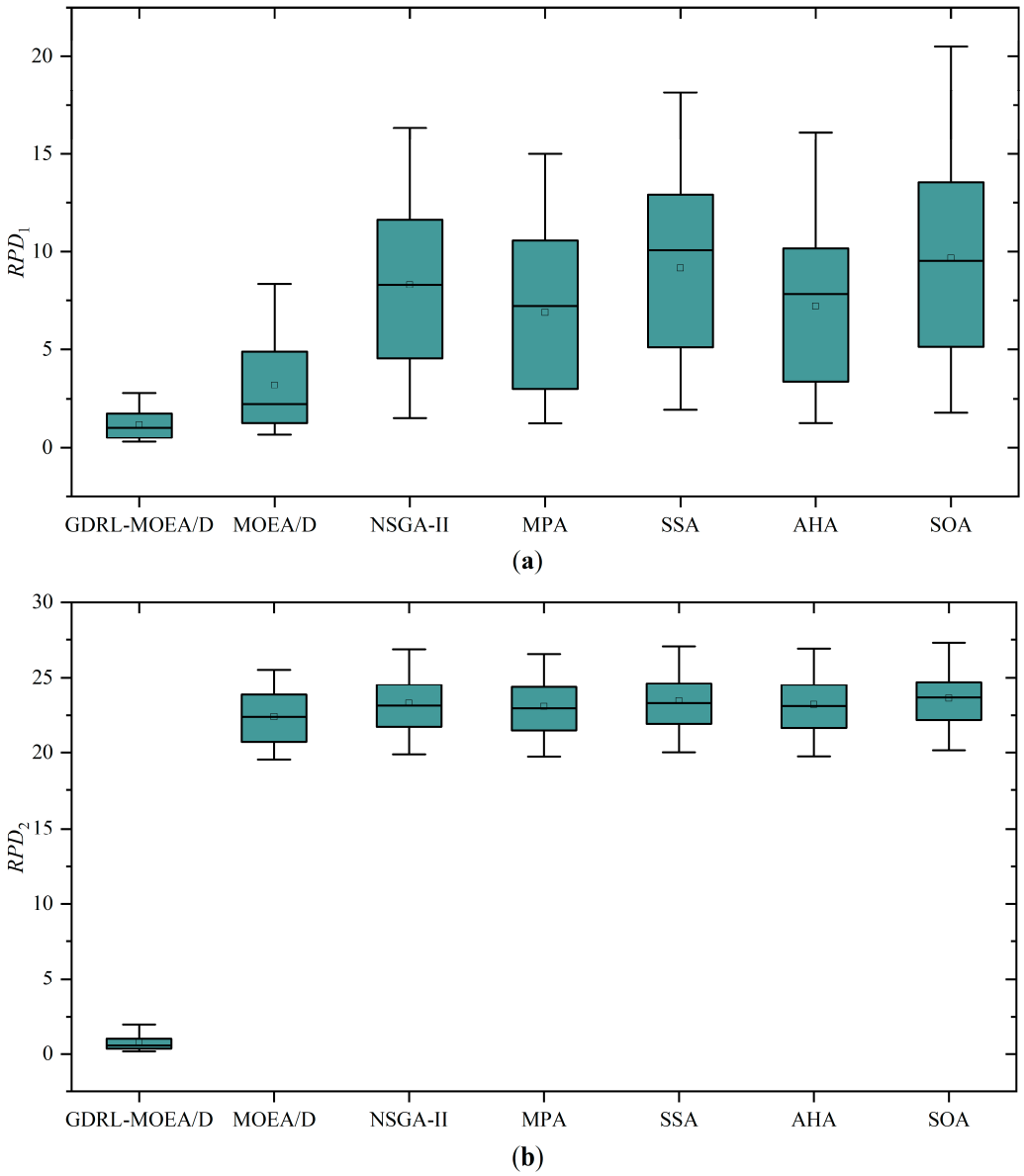


Figure 10. The boxplot of the average *RPD* value. (**a**) represents the boxplot of the average *RPD* value for the maximum completion time; (**b**) represents the boxplot of the average *RPD* value for energy consumption.

Table 8. The average computation times of three algorithms for each scale.

(*n*, *m*)	The Computation Time of Each Algorithm/s						
	GDRL-MOEA/D	**MOEA/D**	**NSGA-II**	**MPA**	**SSA**	**AHA**	**SOA**
(50, 5)	75.1254	78.3117	63.4386	76.2302	80.4125	81.1252	70.5412
(50, 6)	89.2452	90.2976	76.7797	88.5214	90.5412	91.5241	80.4224
(50, 7)	103.5411	105.8324	89.2609	104.2893	102.4125	104.6321	92.4125
(50, 10)	147.3512	148.1881	128.1632	145.2546	150.5412	154.5214	130.5418
(50, 15)	219.5412	219.9746	194.9998	218.6872	230.4512	232.1745	211.2156
(50, 20)	295.6234	294.0099	264.4712	291.0637	300.4124	302.4512	282.9541
(100, 5)	143.2415	154.5248	125.7052	142.1222	150.5416	155.5412	130.1445
(100, 6)	168.6145	172.2160	152.7174	168.0217	180.9841	182.4152	160.7841
(100, 7)	206.1235	200.6013	180.8117	205.3689	219.5412	220.7451	192.5412
(100, 10)	292.2514	293.9304	259.8535	291.2315	300.7451	304.5415	262.3562
(100, 15)	451.3547	441.1938	399.5518	450.0200	460.4152	463.1278	420.7412

Table 8. *Cont.*

(*n*, *m*)	The Computation Time of Each Algorithm/s						
	GDRL-MOEA/D	**MOEA/D**	**NSGA-II**	**MPA**	**SSA**	**AHA**	**SOA**
(100, 20)	593.1045	592.6549	544.2545	592.2356	601.5471	603.4985	563.6482
(150, 5)	205.6841	216.7729	185.8298	206.2297	218.8471	220.7894	202.3541
(150, 6)	255.4562	254.7261	240.0422	255.2442	260.4841	264.3456	251.5624
(150, 7)	301.2514	316.9811	271.1553	298.3484	310.4514	315.4514	291.2481
(150, 10)	441.0145	441.1019	396.4780	439.0293	460.4152	463.3972	420.7892
(150, 15)	672.5241	670.6341	615.5960	669.9642	680.5123	683.7456	642.4874
(150, 20)	913.8121	920.1188	856.0781	912.1419	930.2457	932.1789	902.1872
(200, 5)	275.6581	282.2372	252.1723	276.1188	280.7451	284.4152	270.4671
(200, 6)	339.2514	347.0803	314.1157	338.6582	361.5141	365.8741	331.3416
(200, 7)	396.5412	400.0306	400.3811	396.4272	411.7452	418.8415	421.7412
(200, 10)	589.5418	589.5134	539.3509	587.6343	602.4578	606.7456	561.5715
(200, 15)	915.2564	914.6555	858.1994	912.2259	924.7451	930.6481	894.3251
(200, 20)	1274.4514	1254.9765	1235.4191	1271.5979	1290.4514	1298.3458	1251.8413
AVG	390.2317	391.6902	360.2011	389.0278	400.0483	403.3782	376.6758

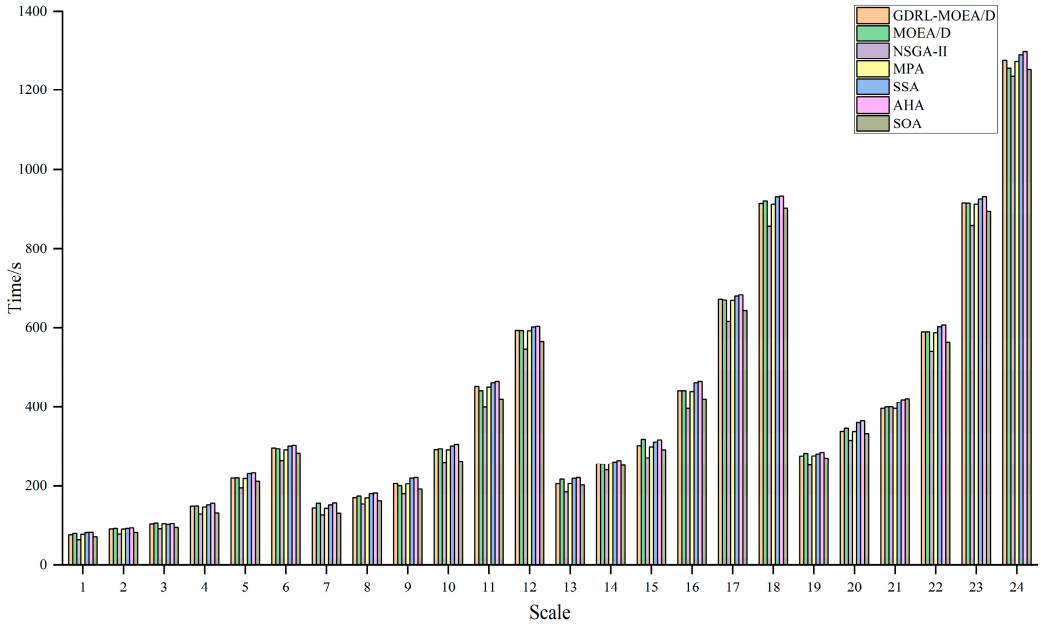


Figure 11. The computation times of seven algorithms for each scale.

In summary, although the GDRL-MOEA/D algorithm proposed in this paper is relatively close to other algorithms in terms of solution speed for different scales of PFSP, it consistently outperforms other algorithms in solution quality. Therefore, by improving the initial population strategy of MOEA/D through deep reinforcement learning and combining it with the proposed energy-saving strategy, this paper not only enhances the solution performance of MOEA/D to a certain extent, but also slightly increases its solution speed.

5. Conclusions and Future Work

This paper focuses on solving the green permutation flow shop scheduling problem with energy consumption consideration (GPFSP), aiming to minimize the maximum completion time and the total energy consumption of machines. A novel solution method combining the end-to-end deep reinforcement learning technique with the MOEA/D algorithm (DRL-MOEA/D) is proposed. Firstly, an end-to-end DRL method is employed to model the PFSP as a sequence-to-sequence model (DRL-PFSP), which is then trained using

the actor–critic algorithm. This model does not rely on high-quality labels and can directly output scheduling solutions for PFSP of various scales once it has been trained. Secondly, considering the advantages of the MOEA/D algorithm in solution quality, robustness, and adaptability to complex problems, the solutions output by the DRL-PFSP model are used as the initial solutions for the MOEA/D algorithm, thereby enhancing the quality of the final solutions produced by the MOEA/D algorithm. Moreover, to more effectively optimize the energy consumption target, a job postponement energy-saving strategy is proposed, which reduces machine idle time without increasing the maximum completion time, thus further optimizing energy consumption. Finally, through a series of simulation experiments, the proposed GDRL-MOEA/D algorithm is compared with the unimproved MOEA/D, NSGA-II, MPA, SSA, AHA, and SOA. The experimental results indicate that the GDRL-MOEA/D algorithm outperforms MOEA/D, NSGA-II, MPA, SSA, AHA, and SOA in solution quality. These algorithms exhibit similar solution speeds across different scales, with the speed differences falling within an acceptable range.

However, the GDRL-MOEA/D method proposed in this paper is designed to solve static GPFSP without considering the dynamic factors that GPFSP may encounter in the actual production process, such as machine failures, sudden order arrivals, and random order arrivals. Therefore, in future work, we will design an DRL algorithm that takes into account the dynamic factors of the workshop and extend the algorithm to solve other types of workshop scheduling problems, such as the parallel machine scheduling problem, the flexible flow shop scheduling problem, and the distributed flow shop scheduling problem.

Author Contributions: Conceptualization, Y.L. and Y.Y.; methodology, Y.L.; software, Y.L.; validation, Y.L.; data curation, Y.L.; writing—original draft preparation, Y.L.; writing—review and editing, Y.Y., A.S., and Y.C.; visualization, Y.L.; supervision, Y.Y., A.S., Y.C., and Y.W.; project administration, Y.Y. and Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (72361032), the Xinjiang Autonomous Region Key R&D Project (2022B01057-2) and the Xinjiang Autonomous Region Natural Science Foundation-Youth Fund (2023D01C177).

Data Availability Statement: The data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. McMahon, G.; Burton, P. Flow-shop scheduling with the branch-and-bound method. *Oper. Res.* **1967**, *15*, 473–481. [CrossRef]
2. Yavuz, M.; Tufekci, S. Dynamic programming solution to the batching problem in just-in-time flow-shops. *Comput. Ind. Eng.* **2006**, *51*, 416–432. [CrossRef]
3. Ronconi, D.P.; Birgin, E.G. Mixed-Integer Programming Models for Flowshop Scheduling Problems Minimizing the Total Earliness and Tardiness. In *Just-in-Time Systems*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 91–105.
4. Campbell, H.G.; Dudek, R.A.; Smith, M.L. A heuristic algorithm for the n job, m machine sequencing problem. *Manag. Sci.* **1970**, *16*, B-630–B-637.
5. Gupta, J.N. A functional heuristic algorithm for the flowshop scheduling problem. *J. Oper. Res. Soc.* **1971**, *22*, 39–47. [CrossRef]
6. Nawaz, M.; Enscore, E.E., Jr.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **1983**, *11*, 91–95. [CrossRef]
7. Johnson, S.M. Optimal two-and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68. [CrossRef]
8. Puka, R.; Duda, J.; Stawowy, A.; Skalna, I. N-NEH+ algorithm for solving permutation flow shop problems. *Comput. Oper. Res.* **2021**, *132*, 105296. [CrossRef]
9. Puka, R.; Skalna, I.; Duda, J.; Stawowy, A. Deterministic constructive vN-NEH+ algorithm to solve permutation flow shop scheduling problem with makespan criterion. *Comput. Oper. Res.* **2024**, *162*, 106473. [CrossRef]
10. Puka, R.; Skalna, I.; Łamasz, B.; Duda, J.; Stawowy, A. Deterministic method for input sequence modification in NEH-based algorithms. *IEEE Access* **2024**, *12*, 68940–68953. [CrossRef]
11. Zhang, J.; Dao, S.D.; Zhang, W.; Goh, M.; Yu, G.; Jin, Y.; Liu, W. A new job priority rule for the NEH-based heuristic to minimize makespan in permutation flowshops. *Eng. Optim.* **2023**, *55*, 1296–1315. [CrossRef]
12. Zheng, J.; Wang, Y. A hybrid bat algorithm for solving the three-stage distributed assembly permutation flowshop scheduling problem. *Appl. Sci.* **2021**, *11*, 10102. [CrossRef]

13. Chen, S.; Zheng, J. Hybrid grey wolf optimizer for solving permutation flow shop scheduling problem. *Concurr. Comput. Pract. Exp.* **2024**, *36*, e7942. [CrossRef]
14. Tian, S.; Li, X.; Wan, J.; Zhang, Y. A novel cuckoo search algorithm for solving permutation flowshop scheduling problems. In Proceedings of the 2021 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE), Tainan, Taiwan, 7–10 November 2021; pp. 1–8.
15. Khurshid, B.; Maqsood, S.; Omair, M.; Sarkar, B.; Ahmad, I.; Muhammad, K. An improved evolution strategy hybridization with simulated annealing for permutation flow shop scheduling problems. *IEEE Access* **2021**, *9*, 94505–94522. [CrossRef]
16. Razali, F.; Nawawi, A. Optimization of Permutation Flowshop Schedulling Problem (PFSP) using First Sequence Artificial Bee Colony (FSABC) Algorithm. *Prog. Eng. Appl. Technol.* **2024**, *5*, 369–377.
17. Qin, X.; Fang, Z.; Zhang, Z. Hybrid symbiotic organisms search algorithm for permutation flow shop scheduling problem. *J. Zhejiang Univer. Eng. Sci.* **2020**, *54*, 712–721.
18. Rui, Z.; Jun, L.; Xingsheng, G. Mixed No-Idle Permutation Flow Shop Scheduling Problem Based on Multi-Objective Discrete Sine Optimization Algorithm. *J. East China Univ. Sci. Technol.* **2022**, *48*, 76–86.
19. Yan, H.; Tang, W.; Yao, B. Permutation flow-shop scheduling problem based on new hybrid crow search algorithm. *Comput. Integr. Manuf. Syst.* **2024**, *30*, 1834.
20. Yang, L. Unsupervised machine learning and image recognition model application in English part-of-speech feature learning under the open platform environment. *Soft Comput.* **2023**, *27*, 10013–10023. [CrossRef]
21. Mohi-Ud-Din, G.; Marnerides, A.K.; Shi, Q.; Dobbins, C.; MacDermott, A. Deep COLA: A deep competitive learning algorithm for future home energy management systems. *IEEE Trans. Emerg. Top. Comput. Intell.* **2020**, *5*, 860–870. [CrossRef]
22. Dudhane, A.; Patil, P.W.; Murala, S. An end-to-end network for image de-hazing and beyond. *IEEE Trans. Emerg. Top. Comput. Intell.* **2020**, *6*, 159–170. [CrossRef]
23. Bai, X.; Wang, X.; Liu, X.; Liu, Q.; Song, J.; Sebe, N.; Kim, B. Explainable deep learning for efficient and robust pattern recognition: A survey of recent developments. *Pattern Recognit.* **2021**, *120*, 108102. [CrossRef]
24. Aurangzeb, K.; Javeed, K.; Alhussein, M.; Rida, I.; Haider, S.I.; Parashar, A. Deep Learning Approach for Hand Gesture Recognition: Applications in Deaf Communication and Healthcare. *Comput. Mater. Contin.* **2024**, *78*, 127–144. [CrossRef]
25. Malik, N.; Altaf, S.; Tariq, M.U.; Ahmed, A.; Babar, M. A Deep Learning Based Sentiment Analytic Model for the Prediction of Traffic Accidents. *Comput. Mater. Contin.* **2023**, *77*, 1599–1615. [CrossRef]
26. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. *arXiv* **2015**, arXiv:1506.03134.
27. Ling, Z.; Tao, X.; Zhang, Y.; Chen, X. Solving optimization problems through fully convolutional networks: An application to the traveling salesman problem. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *51*, 7475–7485. [CrossRef]
28. Zhang, R.; Zhang, C.; Cao, Z.; Song, W.; Tan, P.S.; Zhang, J.; Wen, B.; Dauwels, J. Learning to solve multiple-TSP with time window and rejections via deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 1325–1336. [CrossRef]
29. Luo, J.; Li, C.; Fan, Q.; Liu, Y. A graph convolutional encoder and multi-head attention decoder network for TSP via reinforcement learning. *Eng. Appl. Artif. Intell.* **2022**, *112*, 104848. [CrossRef]
30. Bogyrbayeva, A.; Yoon, T.; Ko, H.; Lim, S.; Yun, H.; Kwon, C. A deep reinforcement learning approach for solving the traveling salesman problem with drone. *Transp. Res. Part C Emerg. Technol.* **2023**, *148*, 103981. [CrossRef]
31. Gao, H.; Zhou, X.; Xu, X.; Lan, Y.; Xiao, Y. AMARL: An attention-based multiagent reinforcement learning approach to the min-max multiple traveling salesmen problem. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *35*, 9758–9772. [CrossRef]
32. Wang, Q.; Hao, Y.; Zhang, J. Generative inverse reinforcement learning for learning 2-opt heuristics without extrinsic rewards in routing problems. *J. King Saud Univ.-Comput. Inf. Sci.* **2023**, *35*, 101787. [CrossRef]
33. Pan, W.; Liu, S.Q. Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Appl. Intell.* **2023**, *53*, 405–422. [CrossRef]
34. Wang, Q.; Hao, Y. Routing optimization with Monte Carlo Tree Search-based multi-agent reinforcement learning. *Appl. Intell.* **2023**, *53*, 25881–25896. [CrossRef]
35. Xu, Y.; Fang, M.; Chen, L.; Xu, G.; Du, Y.; Zhang, C. Reinforcement learning with multiple relational attention for solving vehicle routing problems. *IEEE Trans. Cybern.* **2021**, *52*, 11107–11120. [CrossRef] [PubMed]
36. Zhao, J.; Mao, M.; Zhao, X.; Zou, J. A hybrid of deep reinforcement learning and local search for the vehicle routing problems. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 7208–7218. [CrossRef]
37. Si, J.; Li, X.; Gao, L.; Li, P. An efficient and adaptive design of reinforcement learning environment to solve job shop scheduling problem with soft actor-critic algorithm. *Int. J. Prod. Res.* **2024**, 1–16. [CrossRef]
38. Chen, R.; Li, W.; Yang, H. A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem. *IEEE Trans. Ind. Inform.* **2022**, *19*, 1322–1331. [CrossRef]
39. Shao, C.; Yu, Z.; Tang, J.; Li, Z.; Zhou, B.; Wu, D.; Duan, J. Research on flexible job-shop scheduling problem based on variation-reinforcement learning. *J. Intell. Fuzzy Syst.* **2024**, 1–15. [CrossRef]
40. Han, B.; Yang, J. A deep reinforcement learning based solution for flexible job shop scheduling problem. *Int. J. Simul. Model.* **2021**, *20*, 375–386. [CrossRef]
41. Yuan, E.; Wang, L.; Cheng, S.; Song, S.; Fan, W.; Li, Y. Solving flexible job shop scheduling problems via deep reinforcement learning. *Expert Syst. Appl.* **2024**, *245*, 123019. [CrossRef]

42. Wan, L.; Cui, X.; Zhao, H.; Li, C.; Wang, Z. An effective deep actor-critic reinforcement learning method for solving the flexible job shop scheduling problem. *Neural Comput. Appl.* **2024**, *36*, 11877–11899. [CrossRef]
43. Peng, S.; Xiong, G.; Yang, J.; Shen, Z.; Tamir, T.S.; Tao, Z.; Han, Y.; Wang, F.-Y. Multi-Agent Reinforcement Learning for Extended Flexible Job Shop Scheduling. *Machines* **2023**, *12*, 8. [CrossRef]
44. Wu, X.; Yan, X.; Guan, D.; Wei, M. A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time. *Eng. Appl. Artif. Intell.* **2024**, *131*, 107790. [CrossRef]
45. Liu, R.; Piplani, R.; Toro, C. A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem. *Comput. Oper. Res.* **2023**, *159*, 106294. [CrossRef]
46. Gebreyesus, G.; Fellek, G.; Farid, A.; Fujimura, S.; Yoshie, O. Gated-Attention Model with Reinforcement Learning for Solving Dynamic Job Shop Scheduling Problem. *IEEJ Trans. Electr. Electron. Eng.* **2023**, *18*, 932–944. [CrossRef]
47. Wu, X.; Yan, X. A spatial pyramid pooling-based deep reinforcement learning model for dynamic job-shop scheduling problem. *Comput. Oper. Res.* **2023**, *160*, 106401. [CrossRef]
48. Su, C.; Zhang, C.; Xia, D.; Han, B.; Wang, C.; Chen, G.; Xie, L. Evolution strategies-based optimized graph reinforcement learning for solving dynamic job shop scheduling problem. *Appl. Soft Comput.* **2023**, *145*, 110596. [CrossRef]
49. Liu, C.-L.; Huang, T.-H. Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning. *IEEE Trans. Syst. Man Cybern. Syst.* **2023**, *53*, 6836–6848. [CrossRef]
50. Zhu, H.; Tao, S.; Gui, Y.; Cai, Q. Research on an Adaptive Real-Time Scheduling Method of Dynamic Job-Shop Based on Reinforcement Learning. *Machines* **2022**, *10*, 1078. [CrossRef]
51. Tiacci, L.; Rossi, A. A discrete event simulator to implement deep reinforcement learning for the dynamic flexible job shop scheduling problem. *Simul. Model. Pract. Theory* **2024**, *134*, 102948. [CrossRef]
52. Zhang, L.; Feng, Y.; Xiao, Q.; Xu, Y.; Li, D.; Yang, D.; Yang, Z. Deep reinforcement learning for dynamic flexible job shop scheduling problem considering variable processing times. *J. Manuf. Syst.* **2023**, *71*, 257–273. [CrossRef]
53. Chang, J.; Yu, D.; Zhou, Z.; He, W.; Zhang, L. Hierarchical reinforcement learning for multi-objective real-time flexible scheduling in a smart shop floor. *Machines* **2022**, *10*, 1195. [CrossRef]
54. Zhou, T.; Luo, L.; Ji, S.; He, Y. A Reinforcement Learning Approach to Robust Scheduling of Permutation Flow Shop. *Biomimetics* **2023**, *8*, 478. [CrossRef] [PubMed]
55. Pan, Z.; Wang, L.; Wang, J.; Lu, J. Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. *IEEE Trans. Emerg. Top. Comput. Intell.* **2021**, *7*, 983–994. [CrossRef]
56. Wang, Z.; Cai, B.; Li, J.; Yang, D.; Zhao, Y.; Xie, H. Solving non-permutation flow-shop scheduling problem via a novel deep reinforcement learning approach. *Comput. Oper. Res.* **2023**, *151*, 106095. [CrossRef]
57. Jiang, E.-D.; Wang, L. An improved multi-objective evolutionary algorithm based on decomposition for energy-efficient permutation flow shop scheduling problem with sequence-dependent setup time. *Int. J. Prod. Res.* **2019**, *57*, 1756–1771. [CrossRef]
58. Rossit, D.G.; Nesmachnow, S.; Rossit, D.A. A Multiobjective Evolutionary Algorithm based on Decomposition for a flow shop scheduling problem in the context of Industry 4.0. *Int. J. Math. Eng. Manag. Sci.* **2022**, *7*, 433–454.
59. Nazari, M.; Oroojlooy, A.; Snyder, L.; Takác, M. Reinforcement learning for solving the vehicle routing problem. *arXiv* **2018**, arXiv:1802.04240.
60. Faramarzi, A.; Heidarinejad, M.; Mirjalili, S.; Gandomi, A.H. Marine Predators Algorithm: A nature-inspired metaheuristic. *Expert Syst. Appl.* **2020**, *152*, 113377. [CrossRef]
61. Xue, J.; Shen, B. A novel swarm intelligence optimization approach: Sparrow search algorithm. *Syst. Sci. Control Eng.* **2020**, *8*, 22–34. [CrossRef]
62. Zhao, W.; Wang, L.; Mirjalili, S. Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications. *Comput. Methods Appl. Mech. Eng.* **2022**, *388*, 114194. [CrossRef]
63. Dhiman, G.; Kumar, V. Seagull optimization algorithm: Theory and its applications for large-scale industrial engineering problems. *Knowl.-Based Syst.* **2019**, *165*, 169–196. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.