WEP Warwick Evans Publishing

Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 13
AMMSAC 2026

Hierarchical Multi-Agent Deep Reinforcement Learning with Guided Search for U-Shaped Seru Production Scheduling

Li Zhang *, Xiuli Wang

School of Economics and Management, Nanjing University of Science and Technology, Nanjing, China

* Corresponding Author Email: z_li1218@139.com

Abstract. This paper investigates an integrated scheduling problem in U-shaped seru production systems under a static and deterministic order environment. The problem jointly determines order-to-seru assignment, intra-seru order sequencing, and multi-skilled worker allocation, with the objective of minimizing the makespan. Compared with conventional parallel-machine scheduling, U-shaped seru scheduling involves stronger decision coupling because workload distribution, learning effects, sequence-dependent setup times, and worker configuration jointly determine seru completion times. To characterize these features, a mixed-integer nonlinear programming model is formulated by incorporating the DeJong learning effect, sequence-dependent setup times, worker allocation constraints, and sequencing feasibility requirements. The scheduling problem is then reformulated as a single-step combinatorial Markov decision process. Based on this formulation, a hierarchical Multi-Agent Deep Deterministic Policy Gradient algorithm with Guided Search, termed MADDPG-GuidedSearch, is proposed. The framework consists of a global coordination agent for order assignment and multiple seru execution agents for intra-seru sequencing and worker allocation. Graph attention-based state encoding, centralized training with decentralized execution, Gumbel-Softmax relaxation, decoding repair, local search, and multi-candidate Guided Search are integrated to improve feasibility, search robustness, and solution quality. Computational experiments show that MADDPG-GuidedSearch-50 achieves a slightly lower average makespan than a strong genetic algorithm baseline while requiring less online solution time after offline training. Sensitivity and ablation analyses further indicate that Guided Search and Repair/Local Search are the main contributors to performance improvement.

Keywords: U-shaped seru; production scheduling; multi-agent deep reinforcement learning; MADDPG; guided search; makespan.

1. Introduction

Seru production is a flexible manufacturing organization paradigm designed to meet high-variety, low-volume, and rapidly changing production requirements. Unlike traditional assembly lines, a seru production system decomposes the production process into several relatively independent seru units. Each unit is typically staffed by multiple multi-skilled workers and is capable of completing the entire processing procedure of assigned orders. Prior studies have shown that seru production originated from Japanese manufacturing practices and can effectively improve productivity, flexibility, and responsiveness under uncertain demand conditions [1]. In recent years, research on seru production has gradually expanded from system configuration and production organization to scheduling and optimization problems involving learning effects, setup times, worker allocation, and production lot assignment [2–6].

In U-shaped seru production systems, scheduling decisions are not limited to assigning orders to production units. They also require the simultaneous determination of the processing sequence within each seru and the allocation of limited multi-skilled workers among serus. These decisions are strongly interdependent: order assignment determines the workload distribution across serus, intra-seru sequencing affects both the learning effect and sequence-dependent setup time, and worker allocation directly changes the processing capacity of each seru. Therefore, the U-shaped seru

Content from this work may be used under the terms of CC BY-NC 4.0 licence (https://creativecommons.org/licenses/by-nc/4.0/).
Published by Warwick Evans Publishing.

scheduling problem possesses a typical combinatorial optimization structure, and its computational difficulty increases rapidly as the number of orders, serus, and worker allocation alternatives grows.

Existing studies have provided important modeling and algorithmic foundations for seru scheduling. Jiang et al. investigated seru scheduling with past-sequence-dependent setup times and the DeJong learning effect and proposed an exact solution method [2]. Zhang et al. developed a logic-based Benders decomposition method for seru scheduling with sequence-dependent setup times and DeJong's learning effect, aiming to minimize the makespan [3]. Li and Wu studied the integrated optimization of worker assignment, batch splitting, and scheduling in a hybrid assembly line–seru production system [4]. Li et al. introduced actor–critic learning with pointer networks to solve dynamic worker allocation problems in seru production systems [5]. These studies indicate that learning effects, setup times, and worker configuration are critical factors in seru production scheduling.

Nevertheless, existing seru scheduling studies still rely mainly on mathematical programming, decomposition algorithms, heuristic rules, and metaheuristics. Mathematical programming methods possess strong modeling capability, but their computational burden becomes substantial as the number of orders, serus, and resource combinations increases. Heuristic rules are computationally efficient but often fail to capture the nonlinear coupling among learning effects, setup times, and worker allocation. Metaheuristics such as genetic algorithms have strong search capability, yet they usually require repeated population evolution for each new instance, which may limit their online scheduling efficiency.

Deep reinforcement learning has recently emerged as a promising approach for combinatorial optimization and scheduling problems [10]. In particular, multi-agent deep reinforcement learning is well suited to problems involving multiple decision entities, distributed resources, and hierarchical decision structures. Pu et al. combined graph neural networks with multi-agent reinforcement learning for dynamic job shop scheduling and represented jobs, machines, and scheduling relationships through graph structures [7]. Wang et al. proposed a multi-agent deep reinforcement learning approach for dynamic flexible assembly job shop scheduling with uncertain processing and transportation times [8]. Jang et al. designed a leader–follower multi-agent reinforcement learning framework for large-scale factory-wide dynamic scheduling, improving the scalability of reinforcement learning-based scheduling methods in complex manufacturing environments [9]. These studies demonstrate the potential of multi-agent reinforcement learning in coordinating multi-entity, multi-resource, and multi-level scheduling decisions.

However, applying multi-agent reinforcement learning to U-shaped seru scheduling remains nontrivial. First, the problem contains hybrid decisions, including discrete order assignment, sequencing priority generation, and integer worker allocation. Second, the scheduling outcome is determined by an entire combinatorial structure rather than by a sequence of independent local actions. Third, the neural policy output must be decoded into a feasible schedule that satisfies assignment, sequencing, and worker constraints. Fourth, a single policy inference may still produce a locally inferior solution, especially in a large discrete search space. These challenges motivate the design of a hierarchical reinforcement learning framework combined with feasibility repair, local improvement, and guided search mechanisms.

Motivated by the above analysis, this paper proposes a hierarchical MADDPG-GuidedSearch method for the U-shaped seru production scheduling problem under a static order environment. The main contributions are summarized as follows.

First, a static U-shaped seru scheduling model is established by considering the DeJong learning effect, sequence-dependent setup times, worker allocation constraints, and sequencing feasibility. The model explicitly describes the coupled relationship among order assignment, intra-seru sequencing, and worker allocation.

Second, the static scheduling problem is transformed into a single-step combinatorial Markov decision process. A hierarchical multi-agent decision framework is constructed, in which the global coordination agent is responsible for order assignment and seru execution agents are responsible for sequencing and worker allocation.

Third, a MADDPG-GuidedSearch algorithm is developed by integrating graph attention encoding, hierarchical actors, a centralized critic, Gumbel-Softmax relaxation, decoding repair, local search, and multi-candidate Guided Search. The proposed framework addresses the hybrid action structure and improves the robustness of policy-based scheduling.

Fourth, the effectiveness of the proposed method is evaluated through comparative experiments, sensitivity analysis on the number of guided-search candidates, and ablation studies. The results show that the proposed method achieves competitive solution quality relative to a strong genetic algorithm baseline while reducing online solution time after offline training.

2. Problem Description and Mathematical Model

2.1. Problem Description and Assumptions

This paper considers a U-shaped seru production scheduling problem with the objective of minimizing the makespan in a static order environment. The production system consists of several seru units. Each seru can independently complete the full processing of assigned orders and is staffed by multiple multi-skilled workers. Each order is indivisible and must be assigned as a whole to exactly one seru unit. Because serus may differ in inherent efficiency, workload status, and worker configuration, and because orders differ in processing time and technological similarity, scheduling decisions must jointly determine order assignment, intra-seru sequencing, and worker allocation.

To focus on the core scheduling mechanism and maintain model tractability, the following assumptions are made.

1. All orders are known before scheduling, and parameters such as base processing times, inter-order technological similarities, and seru efficiencies are deterministic.

2. Orders cannot be split; each order must be assigned entirely to one seru unit.

3. Each order is assigned to exactly one seru unit, and the assignment remains unchanged during the static scheduling horizon.

4. Workers within each seru are assumed to be multi-skilled and capable of completing all operations required by the assigned orders. Detailed operation–worker skill matching is not considered at this stage.

5. The DeJong learning effect is considered. As the processing position of an order becomes later within the same seru, its actual processing time decreases according to a learning curve.

6. Sequence-dependent setup times are considered. When two orders are processed consecutively in the same seru, setup operations such as tool replacement, fixture adjustment, or job preparation may be required due to technological differences.

7. The total number of workers in the system is fixed and can be flexibly allocated among seru units.

8. Dynamic disturbances such as machine breakdowns, order arrivals, and temporary worker absence are not considered.

2.2. Processing Time with the DeJong Learning Effect

Let the set of seru units be $M = \{1, 2, \ldots, M\}$ and the set of orders be $N = \{1, 2, \ldots, N\}$. The total number of workers is denoted by W_{max}, and the minimum number of workers assigned to any seru is denoted by W_{min}. For order $i \in N$, its base processing time is p_i. For seru unit $j \in M$, its inherent

efficiency coefficient is e_j, which reflects the combined effect of equipment condition, spatial layout, and organizational efficiency.

The DeJong learning effect is adopted to describe the influence of processing position on actual processing time. When an order is processed at the k-th position in a seru, the learning function is defined as

$$\ell(k) = Q + (1 - Q)k^{\alpha}, \tag{1}$$

where Q denotes the incompressible proportion of the task and α is the learning index, usually taking a negative value. If order i is processed at the k-th position in seru j, its actual processing time is given by

$$T_{ijk}^{proc} = p_i[Q + (1 - Q)k^s] / (w_j e_j), \tag{2}$$

where w_j denotes the number of workers allocated to seru j. This formulation assumes that additional multi-skilled workers improve effective processing capacity in an approximately proportional manner, which is consistent with highly flexible seru environments. Possible diminishing marginal returns of workers can be considered in future extensions.

2.3. Sequence-Dependent Setup Time

The setup time between two consecutive orders depends on their technological similarity. Let $s_{uv} \in [0,1]$ denote the similarity between order u and order v, and let η be the base setup coefficient. When order u is immediately followed by order v in seru j, the sequence-dependent setup time is defined as

$$T_{uvj}^{setup} = \eta(2 - s_{uv})p_v / e_j. \tag{3}$$

No setup time is incurred before the first order in a seru. This definition captures the effect of technological similarity: a higher similarity value reduces the setup burden, whereas a lower similarity value increases the setup time required between two consecutive orders. The base setup coefficient η controls the overall magnitude of changeover time and may be calibrated using historical shop-floor observations or engineering estimates.

2.4. Decision Variables, Objective Function, and Constraints

The order assignment variable is defined as

$$x_{ij} = 1 \text{ if order i is assigned to seru j, 0 otherwise.} \tag{4}$$

The sequencing variable is defined as

$$y_{ijk} = 1 \text{ if order i is processed at the k-th position of seru j, 0 otherwise.} \tag{5}$$

The integer variable $w_j \in Z^+$ denotes the number of workers assigned to seru j. The objective is to minimize the makespan:

$$\min C_{max}, \tag{6}$$

where $C_{max} = \max_{j \in M} C_j$ and C_j denotes the completion time of seru j.

Each order must be assigned to exactly one seru:

$$\Sigma_{j=1}^{M} x_{ij} = 1, \forall i \in N. \tag{7}$$

The total number of workers is fixed:

$$\Sigma_{j=1}^{M} w_j = W_{max}. \tag{8}$$

Each seru must receive at least the minimum number of workers:

$$w_j \geq W_{min}, \forall j \in M. \tag{9}$$

The assignment and sequencing variables must be consistent:

$$x_{ij} = \Sigma_{k=1}^{N} y_{ijk}, \forall i \in N, \forall j \in M. \tag{10}$$

Each processing position in each seru can accommodate at most one order:

$$\Sigma_{i=1}^{N} y_{ijk} \leq 1, \forall j \in M, k = 1, \ldots, N. \tag{11}$$

Each order appears exactly once across all serus and all processing positions:

$$\Sigma_{j=1}^{M} \Sigma_{k=1}^{N} y_{ijk} = 1, \forall i \in N. \tag{12}$$

To avoid infeasible sequences in which a later position is occupied while an earlier position remains empty, a position-continuity constraint is imposed:

$$\Sigma_{i=1}^{N} y_{ijk} \geq \Sigma_{i=1}^{N} y_{ij(k+1)}, \forall j \in M, k = 1, \ldots, N-1. \tag{13}$$

The completion time within each seru is calculated recursively. For the first processing position in seru j, the completion time is

$$E_j^{(1)} = \Sigma_{i=1}^{N} y_{ij1} T_{ij1}^{proc}. \tag{14}$$

For the k-th processing position with $k \geq 2$, the recursive completion time is

$$E_j^{(k)} = E_j^{(k-1)} + \Sigma_{u=1}^{N} \Sigma_{v=1}^{N} y_{uj(k-1)} y_{vjk} T_{uvj}^{setup} + \Sigma_{v=1}^{N} y_{vjk} T_{vjk}^{proc}. \tag{15}$$

The makespan must be no smaller than the completion time of any seru:

$$C_{max} \geq C_j, \forall j \in M, \tag{16}$$

where $C_j = \max_{k=1,\ldots,N} E_j^{(k)}$. Finally, the variable domains are defined as

$$xij \in \{0,1\}, y_{ijk} \in \{0,1\}, w_j \in Z^{+}. \tag{17}$$

The above model includes binary assignment variables, binary sequencing variables, integer worker allocation variables, and nonlinear relationships introduced by learning effects and sequence-dependent setup times. Therefore, the model can be regarded as a mixed-integer nonlinear programming formulation, and the corresponding scheduling problem is a complex combinatorial optimization problem. As the numbers of orders and serus increase, exact optimization becomes computationally expensive. Consequently, the model serves to characterize the problem structure, while a hierarchical multi-agent deep reinforcement learning-based approximate solution method is developed in the next section.

3. Hierarchical MADDPG-GuidedSearch Algorithm

3.1. Markov Decision Process Formulation

To solve the static seru scheduling problem using deep reinforcement learning, the problem is formulated as a single-step combinatorial Markov decision process. One training episode corresponds to one complete scheduling instance. At the beginning of an episode, the agents observe the instance

state and generate a complete joint action, including order assignment, sequencing priorities, and worker allocation. The environment then decodes the joint action into a schedule, applies feasibility repair and local search, calculates the makespan, and returns a terminal reward.

Different from conventional multi-step production process MDPs, the MDP in this study is not designed to simulate state transitions during production execution. Instead, it generates a complete schedule for a static instance in a single decision step. Therefore, the discount factor is set to $\gamma = 0$ in the experiments, which is consistent with the terminal-feedback structure of the problem.

3.2. Multi-Agent Decision Structure

The term multi-agent in this paper refers to multiple decision entities with distinct scheduling responsibilities in the seru production system. A fully cooperative multi-agent framework is constructed, consisting of one global coordination agent and multiple seru execution agents.

The global coordination agent is responsible for order assignment from a system-wide perspective. It synthesizes global information such as order processing times, seru efficiencies, workloads, and worker configurations to determine which seru each order should be assigned to. The seru execution agents are responsible for local execution decisions within each seru, including intra-seru sequencing and worker resource configuration. Since the seru execution agents face similar local decision tasks, parameter sharing is adopted among them to reduce model complexity and improve training stability.

All agents share the same team objective of minimizing the system makespan. The training process follows the centralized training with decentralized execution paradigm. During training, the centralized critic observes the global state and the full joint action to evaluate the quality of the complete schedule. During inference, the critic is not used, and the trained actor networks directly output scheduling actions.

3.3. State Space and Graph Representation

The state space contains the key information of the scheduling instance, including base processing times, order assignment status, seru workloads, worker configurations, seru efficiency coefficients, and inter-order similarities. To represent the structural relationships among orders, serus, and resources, an order–seru heterogeneous graph is constructed. The graph contains order nodes and seru nodes. The edges include order–seru connection edges and order–order similarity edges. Unassigned orders have candidate connection edges to all serus, whereas assigned orders have definite connection edges to their assigned serus. Weighted order–order edges are constructed based on technological similarity.

This graph representation describes order assignment relationships, workload status, and setup relevance in a unified manner. Inspired by graph neural network applications in scheduling [7], a graph attention network is employed to encode the order–seru heterogeneous graph. For node i, the feature update at layer l+1 is

$$h_i^{(l+1)} = \sigma\left(\Sigma_{j \in N(i)} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l)} \right), \tag{18}$$

where N(i) is the set of neighboring nodes of node i, $W^{(l)}$ is a learnable weight matrix, $\alpha_{ij}^{(l)}$ is the attention weight from node j to node i, and $\sigma(\cdot)$ is a nonlinear activation function. The attention weight is calculated as

$$\alpha_{ij}^{(l)} = \exp(\varphi(h_i^{(l)}, h_j^{(l)})) / \Sigma_{k \in N(i)} \exp(\varphi(h_i^{(l)}, h_k^{(l)})). \tag{19}$$

After several graph attention layers, each node representation integrates neighborhood structure and attribute information. A global graph embedding is then obtained through attentive pooling:

$$h_G = \Sigma_{i=1}^{M+N} \beta_i h_i^{(L)}, \tag{20}$$

where β_i is the pooling weight of node i and h_G denotes the global graph embedding. This embedding serves as an important input to both actor and critic networks.

3.4. Action Space and Reward Function

The action space consists of three components: order assignment, intra-seru sequencing, and worker allocation. To reduce the complexity of the joint action space and reflect the hierarchical nature of seru scheduling decisions, the joint action is defined as

$$a = (a^A, a^S, a^W), \tag{21}$$

where a^A denotes the order assignment action, a^S the sequencing priority action, and a^W the worker allocation action.

The order assignment action is generated by the global coordination agent. For each order, the agent outputs logits corresponding to assignment probabilities over different seru units. During training, Gumbel-Softmax is used to transform the logits into an approximately one-hot differentiable assignment matrix. During inference, argmax is used to obtain deterministic assignment results.

The sequencing priority action is generated by the seru execution agents. For each seru, the execution agent outputs priority scores for all orders. During decoding, only the orders assigned to the corresponding seru are retained, and their processing sequence is determined by sorting the priority scores. The worker allocation action is also generated by the seru execution agents. Its logits are first transformed into allocation proportions by softmax and then converted into an integer worker allocation vector through repair so that the total worker count and minimum worker constraints are satisfied.

The optimization objective is to minimize the makespan C_{max}. Directly using $-C_{max}$ as the reward may cause scale instability across instances. Moreover, since the problem is a single-step combinatorial decision problem, the reward is obtained only after a complete schedule has been generated. To improve training stability and interpretability, a reference makespan C_{ref} is introduced. The reference solution is generated by low-cost rules such as RoundRobin and basic LPT, and the better one is retained. The reward function is defined as

$$r = (C_{ref} - C_{max}) / C_{ref} - \lambda \sigma_{load}, \tag{22}$$

where σ_{load} represents the dispersion of completion times among serus and λ is the penalty coefficient. The first term measures the relative improvement over the reference schedule, while the second term penalizes excessive workload imbalance. Thus, the reward function encourages both makespan reduction and balanced utilization of seru resources.

3.5. Actor–Critic Framework and Discrete Action Processing

The actor network adopts a hierarchical structure corresponding to global assignment decisions and local seru execution decisions. The global coordination actor takes the global graph embedding h_G and the global state vector as inputs and outputs an order assignment logits matrix $L^A \in R^{N\times M}$. The seru execution actor takes the global graph embedding and seru-local state features as inputs and outputs a sequencing priority matrix $L^S \in R^{M\times N}$ and a worker allocation logits vector $L^W \in R^M$. Parameter sharing is applied among seru execution agents to reduce the number of trainable parameters and enhance learning stability.

The centralized critic supports coordinated training under the CTDE paradigm. During training, the critic observes the global graph embedding, global state, and complete joint action to estimate the value of the generated schedule. During inference, the critic is removed. The critic input includes the global graph embedding h_G, the global state s_G, and the flattened joint action:

$$\tilde{a} = [\text{flatten}(A^A); \text{flatten}(A^S); a^W]. \quad (23)$$

The critic outputs a scalar Q-value:

$$Q = f_{critic}(h_G, s_G, \tilde{a}; \theta^Q). \quad (24)$$

This Q-value is used to guide the policy updates of both the global coordination actor and the seru execution actors.

Because order assignment is a discrete action, direct argmax selection during training would block gradient propagation. Gumbel-Softmax is therefore adopted to provide a differentiable approximation. For a logits vector $l = [l_1, \ldots, l_M]$, the Gumbel-Softmax output is

$$y_k = \exp((l_k + g_k) / \tau) / \Sigma_{j=1}^{M} \exp((l_j + g_j) / \tau), k = 1, \ldots, M, \quad (25)$$

where g_k is noise sampled from a standard Gumbel distribution and τ is the temperature parameter. A larger temperature produces a smoother distribution and encourages exploration, whereas a smaller temperature makes the output closer to a one-hot vector and facilitates policy convergence.

3.6. Decoding Repair, Local Search, and Guided Search

The raw outputs of the actor networks must be decoded into a feasible schedule. The decoding process includes three steps: converting assignment logits into deterministic order–seru assignments, determining intra-seru sequences based on priority scores, and transforming worker allocation logits into a feasible integer worker allocation vector. To improve reproducibility, the decoding procedure follows a fixed rule. Assignment logits are first converted into seru labels; assigned orders in each seru are then sorted according to their priority scores; finally, worker allocation proportions are rounded and repaired until the total worker constraint and minimum worker constraints are satisfied.

Since the initial schedule generated by the neural network may contain infeasible or locally suboptimal structures, a repair and local search mechanism is introduced. Repair is mainly used to ensure that each order is assigned exactly once, each seru satisfies the minimum worker requirement, and the total number of allocated workers equals W_{max}. Local search further improves the schedule by exploring the neighborhood of the current solution. The main neighborhood operations include order migration, order exchange, and worker reallocation. Order migration moves an order from the bottleneck seru to another seru; order exchange swaps orders between the bottleneck seru and non-bottleneck serus; and worker reallocation transfers workers from non-bottleneck serus to the bottleneck seru under feasibility constraints. If a neighborhood operation reduces C_{max}, it is accepted and the current solution is updated.

A single policy inference followed by local search is essentially a single-trajectory optimization process and may still depend heavily on the initial solution. To improve robustness, a Guided Search mechanism is proposed. After training, the best saved policy network is loaded, and K candidate schedules are generated for the same scheduling instance. Random perturbations are introduced during inference so that different candidates may have different order assignments, sequences, and worker configurations. Each candidate is decoded, repaired, and improved by local search. The candidate with the smallest C_{max} is selected as the final output. Guided Search can be regarded as a lightweight multi-start search based on learned scheduling priors. Unlike fully random multi-start search, its candidates are generated by the trained policy network and are therefore more likely to be located near promising solution regions.

The overall procedure of the proposed MADDPG-GuidedSearch method is summarized in Algorithm 1.

Algorithm 1. MADDPG-GuidedSearch

Input: Problem instance distribution and algorithm hyperparameters.

Output: Best policy π_{best} and final schedule S*.

1. Initialize the graph encoder, global coordination actor, seru execution actor, centralized critic, target networks, and replay buffer.

2. For each training episode, generate a scheduling instance and construct the corresponding order–seru graph.

3. Encode the graph and obtain the global graph embedding.

4. Generate the joint action (a^A, a^S, a^W) using the global coordination actor and seru execution actor.
5. Decode the joint action into a schedule, apply feasibility repair and local search, and calculate C_{max}.

6. Compute the reward according to Eq. (22) and store the transition in the replay buffer.

7. Update the centralized critic and actor networks using mini-batch samples under the centralized training with decentralized execution paradigm.

8. Preserve the policy with the best validation performance as π_{best}.

9. During inference, load π_{best}, generate K candidate schedules with stochastic perturbations, apply repair and local search to each candidate, and output the candidate with the smallest makespan.

4. Experiments and Results

4.1. Experimental Settings

Simulated seru scheduling instances are generated to evaluate the proposed method. The base parameters are set as follows: the number of seru units is M = 3, the number of orders is N = 12, the total number of workers is W_{max} = 10, and the minimum number of workers per seru is W_{min} = 1. For seru-related parameters, the incompressible proportion in the DeJong learning effect is Q = 0.5, the learning index is $\alpha = -0.152$, and the base setup coefficient is $\eta = 0.1$. Base order processing times are randomly generated from a uniform distribution U(5,15), inter-order similarities from U(0.5,1.0), and seru efficiency coefficients from U(0.8,1.2). All methods are evaluated on the same set of instances to ensure a fair comparison.

The main algorithm parameters are as follows. The number of training episodes is 1200, the discount factor is $\gamma = 0$, the target network soft update coefficient is 0.005, the actor learning rate is 1×10^{-4}, and the critic learning rate is 1×10^{-3}. The replay buffer capacity is 50000, and the batch size is 128. The graph attention network has two layers with a hidden dimension of 128. The Gumbel-Softmax temperature is linearly annealed from 2.0 to 0.5. The elite experience buffer capacity is 300, the number of behavior cloning fine-tuning episodes is 120, the default number of Guided Search candidates is K = 50, and the maximum number of local search iterations is 50. Offline training and online inference time are reported separately because the proposed method follows a train-once-and-infer-fast paradigm.

For the GA-Full baseline, a genetic algorithm is used to jointly optimize order assignment, intra-seru sequencing, and worker allocation. Order assignment is encoded as an integer vector, intra-seru sequences are represented by random keys, and worker allocation is encoded as an integer vector. The main GA parameters are as follows: population size 80, crossover probability 0.8, mutation probability 0.12, and maximum number of generations 150.

All experiments are repeated across 10 independent random seeds. Each seed corresponds to independently generated problem instances, network initialization, and training runs. The reported metrics include mean value, standard deviation, best value, worst value, and computation time. For comparisons between major algorithms, paired-sample t-tests and Wilcoxon signed-rank tests are conducted at the significance level of 0.05.

4.2. Baseline Methods

Several baselines are used to evaluate the effectiveness of the proposed algorithm. Random randomly generates order assignments, intra-seru sequences, and worker configurations, serving as a no-optimization baseline. RoundRobin assigns orders cyclically to serus according to order indices, processes orders within each seru in the original order, and allocates workers as evenly as possible. BasicScheduling-LPT sorts orders in descending order of base processing time and assigns each order to the seru with the smallest current cumulative workload. Seru-LPT-SPT extends the LPT framework by considering seru efficiencies, learning effects, and estimated setup times, and applies a shortest-processing-time rule for intra-seru sequencing. GA-Full uses a genetic algorithm to jointly optimize order assignment, intra-seru sequencing, and worker allocation; its detailed settings are provided in Section 4.1.

Three MADDPG variants are also designed for comparison. MADDPG-FinalGreedy uses the final trained policy network to deterministically generate a schedule without multi-candidate search. MADDPG-BestPolicy-Repaired loads the best policy saved during training and applies repair and local search to the generated schedule. MADDPG-GuidedSearch-50 loads the best policy, generates $K = 50$ candidate schedules, applies repair and local search to each candidate, and selects the candidate with the smallest makespan as the final output.

4.3. Main Experimental Results

Table 1 reports the complete experimental results across all methods for 10 random seeds, including mean, standard deviation, best and worst makespan, and average computation time.

Table 1. Complete experimental results over 10 random seeds

Method	Mean Cmax	Std Cmax	Best Cmax	Worst Cmax	Mean Runtime/s
Random	30.22	3.02	13.75	38.59	0.008
RoundRobin	16.98	1.60	14.80	19.90	<0.001
BasicScheduling-LPT	21.74	3.63	15.83	27.06	<0.001
Seru-LPT-SPT	20.65	8.15	15.36	44.42	<0.001
GA-Full	14.31	1.15	12.83	16.92	1.45
MADDPG-FinalGreedy	15.13	1.17	13.91	18.11	0.12
MADDPG-BestPolicy-Repaired	14.35	1.23	12.83	17.17	0.12
MADDPG-GuidedSearch-50	14.13	1.23	12.36	16.83	0.36

The experimental results clearly show that simple rule-based methods, while computationally cheap, cannot effectively coordinate order assignment, intra-seru sequencing, and worker allocation under the combined influence of learning and setup effects. Random search yields the worst and most variable performance. RoundRobin achieves moderate workload balance but ignores processing time heterogeneity, seru efficiency differences, and sequence-dependent setup times. BasicScheduling-LPT performs reasonably in some instances but exhibits high variance, reflecting the difficulty of directly transferring generic dispatching rules to seru environments. Seru-LPT-SPT integrates seru-specific characteristics, which improves over simpler heuristics in several cases, yet its rule-based structure still limits its adaptivity.

GA-Full achieves competitive performance by jointly searching the three coupled decision spaces through population evolution. Its mean C_{max} is 14.31, demonstrating the strong optimization capability of genetic algorithms for this problem class. The proposed MADDPG-GuidedSearch-50 obtains a mean C_{max} of 14.13, which is approximately 1.2% lower than that of GA-Full. A paired t-test gives $p = 0.0288$ and a Wilcoxon signed-rank test gives $p = 0.0137$, confirming that the improvement is statistically significant. The proposed method thus achieves a modest but statistically significant improvement over the strong metaheuristic baseline in the tested instances, while also providing a clear advantage in online inference time.

The comparison among MADDPG variants further isolates the contribution of each component. MADDPG-FinalGreedy yields a mean C_{max} of 15.13, showing that the raw policy output already captures meaningful scheduling patterns but may still contain locally inferior decisions. MADDPG-BestPolicy-Repaired reduces the mean C_{max} to 14.35, indicating that best-policy preservation combined with repair and local search substantially enhances the solution quality. MADDPG-GuidedSearch-50 further improves to 14.13, demonstrating that multi-candidate policy-guided search can effectively escape local optima and refine the final schedule beyond single-trajectory local optimization.

4.4. Runtime Analysis

Rule-based methods such as RoundRobin, BasicScheduling-LPT, and Seru-LPT-SPT require negligible runtime but produce clearly inferior makespans. GA-Full requires repeated population evolution over 150 generations, resulting in an average solution time of about 1.45 seconds. The runtime of MADDPG-based methods must be interpreted by distinguishing offline training from online inference. Offline training is a one-time computational cost used to obtain the policy model. After training, MADDPG-FinalGreedy requires only 0.12 seconds for inference. Although MADDPG-GuidedSearch-50 generates 50 candidate solutions and applies repair and local search to each, its average online solution time is 0.36 seconds, which is still substantially lower than that of GA-Full.

This result indicates that the proposed method holds promise for online decision support after the one-time offline training investment. However, the current experiments are limited to random instances drawn from the same distribution. While the trained policy is expected to provide fast inference for similarly sized instances, its generalization to larger-scale or out-of-distribution instances requires further investigation. Consequently, the reported runtime advantage should be understood as an online inference advantage rather than a reduction in total model development cost.

4.5. Sensitivity Analysis of Candidate Number K

The number of candidates K is the core parameter of Guided Search, directly affecting the coverage of candidate solutions. If K is too small, the multi-start advantage cannot be fully realized, and performance approaches that of single-trajectory local search. If K is too large, policy inference and local search costs grow approximately linearly, potentially reducing online responsiveness. A sensitivity analysis is therefore conducted with $K \in \{1, 5, 10, 20, 30, 50, 100\}$. The results are shown in Table 2.

Table 2. Sensitivity analysis of candidate number K

K	Mean Cmax	Std Cmax	Best Cmax	Mean Runtime/s	Gap to K=50
1	14.442	1.297	12.778	0.011	+1.75%
5	14.277	1.283	12.641	0.044	+0.59%
10	14.267	1.235	12.768	0.092	+0.52%
20	14.224	1.235	12.692	0.189	+0.22%
30	14.185	1.252	12.645	0.245	−0.06%
50	14.193	1.293	12.551	0.353	Baseline
100	14.119	1.274	12.609	0.788	−0.52%
1	14.442	1.297	12.778	0.011	+1.75%

From the perspective of solution quality, the mean C_{max} generally decreases as K increases, although the improvement is not strictly monotonic. When K = 1, Guided Search degenerates into single-candidate local search and obtains a mean C_{max} of 14.442. As K increases to 20 and 30, the mean C_{max} decreases to 14.224 and 14.185, respectively, indicating that increasing the number of candidates enhances policy-guided exploration. When K = 50, the mean C_{max} is 14.193, which is close to the

result at K = 30, suggesting that the algorithm begins to enter a performance plateau. Further increasing K to 100 reduces the mean C_{max} to 14.119, but the improvement over K = 50 is only 0.52%.

In terms of computation time, the runtime increases approximately linearly with K. The average runtime is 0.011 s when K = 1, 0.353 s when K = 50, and 0.788 s when K = 100. Therefore, a larger candidate number improves the probability of finding better solutions but also increases online computational cost. Considering both solution quality and efficiency, K = 50 is selected as the default setting in the main experiments.

4.6. Ablation Study

To evaluate the contribution of key modules in MADDPG-GuidedSearch, ablation experiments are conducted. Starting from the Full Model, Guided Search, Repair/Local Search, Elite Fine-tuning, and Expert Pretraining are removed separately. The mean makespan, standard deviation, best value, worst value, and online solution time are compared across 10 random seeds. The results are shown in Table 3.

Table 3. Ablation study results

Ablation Model	Mean Cmax	Std Cmax	Best Cmax	Worst Cmax	Mean Runtime/s	Gap to Full
Full Model	14.134	1.292	12.357	16.826	0.438	—
w/o GuidedSearch	14.353	1.295	12.828	17.169	0.183	+1.55%
w/o Repair	14.551	1.399	12.828	17.820	0.069	+2.95%
w/o Elite Fine-tuning	14.141	1.278	12.505	16.857	0.563	+0.05%
w/o Expert Pretraining	14.191	1.204	12.759	16.833	0.712	+0.40%

The Full Model achieves the lowest mean C_{max}, with a value of 14.134. Removing Guided Search increases the mean C_{max} to 14.353, corresponding to a degradation of approximately 1.55%. This indicates that multi-candidate guided search effectively improves the final schedule quality. A single BestPolicy-Repaired solution may still be affected by the quality of the initial policy output, whereas generating multiple candidates and selecting the best one reduces the risk of being trapped in local optima.

Removing Repair/Local Search leads to a larger degradation. The mean C_{max} increases to 14.551, which is approximately 2.95% worse than the Full Model. This result demonstrates that repair and local search are essential for improving final solution quality. Although the neural policy can generate reasonably good initial schedules, local defects may still exist in order assignment, sequencing, or worker allocation. By applying bottleneck-oriented operations such as order migration, cross-seru order exchange, and worker reallocation, local search can effectively improve the schedule structure and reduce the makespan.

By contrast, removing Elite Fine-tuning produces only a slight increase in mean C_{max}, from 14.134 to 14.141. This suggests that, under the current experimental settings, Guided Search and local search can largely compensate for the marginal differences introduced by policy fine-tuning. Similarly, removing Expert Pretraining increases the mean C_{max} to 14.191, corresponding to a gap of approximately 0.40%. This indicates that expert pretraining is not the sole source of high-quality solutions. Rather, it functions mainly as an auxiliary mechanism for accelerating early training and reducing initial exploration difficulty.

Overall, the ablation results show that Guided Search and Repair/Local Search are the two most important modules in the proposed method. Guided Search expands the coverage of high-quality candidate solutions through multi-candidate generation, while Repair/Local Search improves the local structure of each candidate. Elite Fine-tuning and Expert Pretraining play supportive roles in training stabilization and policy initialization, but they are not the primary sources of final

performance improvement. The main advantage of the proposed method therefore comes from the integrated solution paradigm of policy-based candidate generation, decoding repair, local search, and multi-candidate selection.

4.7. Discussion and Limitations

The experimental results support the effectiveness of combining policy-based candidate generation with repair, local improvement, and multi-candidate selection. Nevertheless, the empirical evidence should be interpreted with several limitations in mind. First, the main experiments are conducted on simulated instances with a fixed base scale. Although this setting is sufficient for verifying the feasibility of the proposed framework, larger-scale and cross-distribution instances are still required to fully assess its generalization ability. Second, the current worker productivity formulation assumes approximately proportional capacity improvement with the number of workers. In practical production systems, the marginal productivity of additional workers may decrease due to coordination overhead and space constraints. Third, the proposed method emphasizes online inference efficiency after offline training, but the offline training cost should be considered when evaluating its overall deployment value. Finally, the genetic algorithm baseline was configured with a population size of 80 and 150 generations; further tuning may yield different results, and the comparison should be interpreted within the scope of the chosen hyperparameter settings.

5. Summary

This paper investigates a U-shaped seru production scheduling problem with the objective of minimizing the makespan under a static order environment. A mathematical model is established by incorporating the DeJong learning effect, sequence-dependent setup times, worker allocation constraints, and schedule feasibility requirements. The model characterizes three strongly coupled scheduling decisions: order assignment, intra-seru sequencing, and worker allocation. To solve the problem, the static combinatorial optimization task is formulated as a single-step Markov decision process, and a hierarchical multi-agent reinforcement learning framework is developed.

The proposed MADDPG-GuidedSearch algorithm integrates graph attention network-based state encoding, centralized training with decentralized execution, Gumbel-Softmax relaxation for discrete action processing, best-policy preservation, decoding repair, local search, and multi-candidate Guided Search. Computational experiments show that MADDPG-GuidedSearch-50 achieves a slightly lower average makespan than the GA-Full baseline, and statistical tests confirm the significance of the observed difference under the tested instances. Meanwhile, the online solution time of the proposed method is lower than that of GA-Full, indicating a favorable balance between solution quality and online computational efficiency after offline training.

The sensitivity analysis on the number of candidates shows that increasing K generally improves solution quality but with diminishing marginal returns. Considering both makespan and runtime, K = 50 is a reasonable default setting. The ablation study further confirms that Guided Search and Repair/Local Search are the most critical modules for improving final schedule quality, while Elite Fine-tuning and Expert Pretraining mainly serve as auxiliary mechanisms for training stabilization and policy initialization.

Future research can be extended in three directions. First, due-date constraints, order priorities, and total tardiness objectives can be incorporated to address service-level-oriented scheduling. Second, dynamic disturbances such as order arrivals, machine breakdowns, and temporary worker absences can be considered to enhance practical applicability. Third, the generalization ability of the proposed method should be further evaluated on larger-scale and cross-distribution instances.

Acknowledgements

The authors declare that no specific funding was received for this study.

References

[1] Y. Fujita, K. Izui, S. Nishiwaki, Z. Zhang, Y. Yin, Production planning method for seru production systems under demand uncertainty, Computers & Industrial Engineering, 163 (2022) 107856.

[2] Y. Jiang, Z. Zhang, X. Gong, Y. Yin, An exact solution method for solving seru scheduling problems with past-sequence-dependent setup time and learning effect, Computers & Industrial Engineering, 158 (2021) 107354.

[3] Z. Zhang, X.L. Song, H.J. Huang, X. Zhou, Y. Yin, Logic-based Benders decomposition method for the seru scheduling problem with sequence-dependent setup time and DeJong's learning effect, European Journal of Operational Research, 297 (2022) 866–877.

[4] B. Li, Y. Wu, Integrated optimization of worker assignment, batch splitting and scheduling for a hybrid assembly line-seru production system, Computers & Industrial Engineering, 194 (2024) 110399.

[5] D. Li, H. Jin, Y. Zhang, Dynamic worker allocation in Seru production systems with actor–critic and pointer networks, European Journal of Operational Research, 324 (2025) 62–74.

[6] Z. Zhang, X. Gong, X. Song, Y. Yin, B. Lev, J. Chen, A column generation-based exact solution method for seru scheduling problems, Omega, 108 (2022) 102581.

[7] Y. Pu, F. Li, S. Rahimifard, Multi-Agent Reinforcement Learning for Job Shop Scheduling in Dynamic Environments, Sustainability, 16 (2024) 3234.

[8] H. Wang, W. Lin, T. Peng, Q. Xiao, R. Tang, Multi-agent deep reinforcement learning-based approach for dynamic flexible assembly job shop scheduling with uncertain processing and transport times, Expert Systems with Applications, 270 (2025) 126441, doi: 10.1016/j.eswa.2025.126441.

[9] J. Jang, D. Klabjan, H. Liu, N.S. Patel, X. Li, B. Ananthanarayanan, H. Dauod, T.-H. Juang, Scalable multi-agent reinforcement learning for factory-wide dynamic scheduling in semiconductor manufacturing, Engineering Applications of Artificial Intelligence, 161 (2025) 112168, doi: 10.1016/j.engappai.2025.112168.

[10] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: A methodological tour d'horizon, European Journal of Operational Research, 290 (2021) 405–421.