

Article

An RL-Inspired Probabilistic Updating Heuristic for Mode Selection in Modular Construction Scheduling

Zijie Xing ¹ , Chen Chen ^{2,*} , Robert Lee Kong Tiong ¹ and Yaowen Yang ¹ 

¹ School of Civil and Environmental Engineering, Nanyang Technological University, Singapore 639798, Singapore; zijie001@e.ntu.edu.sg (Z.X.); clkiong@ntu.edu.sg (R.L.K.T.); cywyang@ntu.edu.sg (Y.Y.)

² Mentor Town, 60 Paya Lebar Road, #07-54 Paya Lebar Square, Singapore 409051, Singapore

* Correspondence: chenchen@mentortownsg.com

Abstract

Multi-mode project scheduling, in which each activity can be executed in one of several alternative modes with distinct duration and resource requirements, represents a challenging combinatorial optimization problem prevalent in modular construction and manufacturing. While metaheuristic algorithms excel at exploring the global space of activity sequences, the selection of appropriate execution modes remains a critical sub-problem that demands efficient and effective decision-making. This paper conducts a systematic empirical evaluation of an RL-inspired probabilistic updating heuristic for mode selection in multi-mode resource-constrained project scheduling. Unlike traditional co-evolutionary approaches that optimize sequences and modes jointly, we decouple mode selection from sequence optimization and formulate it as an independent probabilistic learning task. The proposed heuristic maintains a categorical distribution over execution modes for each activity, iteratively updated by reinforcing modes appearing in high-performing schedules, using a normalized reward signal derived from schedule makespan. We implement five representative learning algorithms (REINFORCE, PPO, SAC, DDPG, TD3) as mode selectors and pair them with a Learning-Enhanced Differential Evolution (LEDE) sequence optimizer. Among these, the tabular probability updating (TPU) variant, which does not rely on log-policy gradients or a full MDP formulation, achieves the lowest makespan and the smallest variance, owing to its simple update rule, reward normalization, and a minimum probability threshold that preserves exploration. The learned mode preferences, although conditioned only on the activity index, consistently improve schedule quality when integrated with three different sequence optimizers (Genetic Algorithm, canonical Differential Evolution, and LEDE), reducing makespan by 6–12% compared with a static fastest-mode heuristic. Our results demonstrate that a lightweight, heuristic-driven probability adaptation can serve as an effective, plug-and-play enhancement for mode selection, offering a practical alternative to more complex neural-network-based approaches in resource-constrained modular construction scheduling.



Academic Editor: Jorge Lopes

Received: 30 May 2026

Revised: 9 July 2026

Accepted: 10 July 2026

Published: 5 August 2026

Copyright: © 2026 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 \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: multi-mode project scheduling; resource-constrained scheduling; RL-inspired heuristic; probabilistic updating; mode selection; differential evolution

1. Introduction

Modular construction has become more common in recent years, driven by the need for faster delivery, improved quality control, and reduced on-site disruption [1]. Many

countries use modular room units (MRUs) to speed up building projects, particularly in high-density urban environments such as Singapore, where prefabricated prefinished volumetric construction (PPVC) has been actively promoted by government agencies [2,3]. Figure 1 (left) is a picture of MRU construction. These units are made in factories, where each unit goes through a series of tasks. Some tasks can be done in different ways. A task might take less time if more workers are assigned, but that uses more labor. Another way to do the same task could take longer but save workers for other jobs. Factories often work on many MRUs at the same time. All these units share the same limited resources, like workers and equipment. This creates a difficult scheduling problem. The problem is known as the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MRCMPSP) [4]. When multiple projects run together, it becomes even harder [5,6]. The problem is NP-hard in the strong sense, and its complexity grows significantly with the number of activities, modes, and shared resources [7,8]. One key decision in this problem is mode selection. For each task, the scheduler must pick one execution mode from a set of alternatives. That choice affects how long the task takes and how many resources it uses. A good set of mode choices can reduce the total time to finish all projects, while a bad set can cause delays and resource conflicts [9]. Traditional methods for mode selection often use fixed rules. For example, a rule might always pick the fastest mode, or the mode that uses the fewest resources. These rules are simple and computationally cheap, but they do not adapt to the current state of the factory. What works well at the start of a schedule may be a poor choice later. Other approaches using mathematical programming, such as mixed-integer linear programming, can find exact solutions for very small problems, but they become computationally intractable when the number of projects grows [10,11]. Recent advances have explored metaheuristic and hybrid approaches to tackle larger instances, yet the mode selection sub-problem remains a challenging and under-addressed aspect of the overall scheduling process [12,13].



Figure 1. On-site MRU construction environment (left) and the controlled assembly environment at the Singapore factory (right).

Heuristic and metaheuristic algorithms, particularly Genetic Algorithms and Differential Evolution, have been extensively utilized to address the MRCMPSP by effectively searching for near-optimal mode assignments [14–16]. However, these methods typically treat mode selection as a discrete variable and lack the capacity to leverage historical data; consequently, each problem instance is solved *de novo*, regardless of its similarity to previously encountered scenarios. Reinforcement learning (RL) and related learning-based paradigms provide an alternative avenue by enabling an agent or a heuristic to improve through interaction with a schedule evaluation environment [17,18]. Through a continuous cycle of action and reward, a learning agent can converge on mode preferences that navigate the trade-offs of resource consumption and duration [19]. Recent studies have proposed co-evolutionary metaheuristics for multi-mode project scheduling, where activity

sequences and mode assignments are encoded together in the same solution representation and evolved jointly using operators such as mutation and crossover [20,21]. While these methods achieve strong global search performance, they treat mode selection as an auxiliary variable that is adapted only indirectly through the same population dynamics. Consequently, they do not accumulate experience across scheduling episodes and cannot learn mode preferences that transfer to new problem instances [22].

Recent years have witnessed growing research efforts on both metaheuristic and reinforcement learning approaches for project scheduling and related combinatorial optimization problems. In the domain of metaheuristics, hybrid methods combining constraint programming with evolutionary search have been proposed for the MRCMPSP, demonstrating improved solution quality on complex instances [12]. Multi-objective variants of the RCPSP have also been actively investigated, with experimental comparisons of metaheuristics under time-dependent resource costs [13], while adaptive Differential Evolution and genetic programming hyper-heuristics have been extended to handle dynamic and flexible scheduling scenarios [23–25]. In parallel, reinforcement learning has gained increasing attention for production scheduling, with comprehensive reviews highlighting its potential to learn adaptive dispatching policies and improve decision-making under uncertainty [18,19,26,27]. Specific to multi-mode project scheduling, recent work has integrated Q-learning with NSGA-II for bi-objective optimization, where the RL component assists in adaptive mode selection during the evolutionary process [20]. Despite these advances, most existing approaches still treat mode selection as a sub-component of a larger evolutionary framework rather than as an independent learning problem, and few studies have systematically compared different learning-based algorithms in the context of mode selection for MRCMPSP. Moreover, the ability of learned mode preferences to generalize across different sequence optimizers, as well as the computational trade-offs between lightweight tabular methods and more complex neural-network-based approaches, have not been rigorously quantified for this problem class.

To address this gap, we systematically evaluate a set of RL-inspired probabilistic updating heuristics for mode selection in multi-mode project scheduling. Unlike prior work that integrates learning within a single metaheuristic, this paper decouples mode selection from sequence optimization and treats it as an independent probabilistic learning task. Experiments are conducted on real-world data collected from an MRU factory in Singapore, as shown in Figure 1 (right). We compare five representative algorithms (REINFORCE, Proximal Policy Optimization, Soft Actor–Critic, Deep Deterministic Policy Gradient, and Twin Delayed DDPG), with particular attention to a lightweight tabular probability updating (TPU) variant that does not rely on log-policy gradients or a complete Markov Decision Process formulation. The analysis quantifies the contribution of reward normalization, the minimum probability threshold, and the learning rate, and examines whether a lightweight tabular heuristic can outperform neural network-based alternatives in this discrete decision setting. We also assess the methodological applicability of the learned mode selector across three distinct sequence optimizers and provide a preliminary evaluation of its transferability. The main contributions of this paper are as follows.

1. **Decoupled Formulation:** We propose a clear separation between mode selection and activity sequencing, in contrast to traditional co-evolutionary approaches that optimize both components jointly. This decoupling allows the mode selection heuristic to be developed and evaluated independently of the sequence optimizer.
2. **Empirical Insights on Mode Learning:** Through systematic experimentation, we demonstrate that a lightweight, tabular probabilistic updating heuristic can effectively learn mode preferences that consistently improve schedule quality. We show that this approach does not require gradient information from the se-

quence optimizer, and that its simple update rule, combined with reward normalization and an exploration-preserving probability threshold, yields stable and competitive performance.

3. **Methodological Applicability across Optimizers:** Unlike existing hybrid frameworks where learning components are tightly coupled to a specific population-based search, our proposed mode selector can be integrated as a plug-and-play enhancement with structurally diverse sequence optimizers (e.g., GA, DE, LEDE). We demonstrate that retraining the heuristic for each optimizer consistently reduces makespan, and provide preliminary evidence that a frozen policy retains some beneficial transferability.
4. **Empirical Validation on Real-World and Benchmark Data:** We validate the proposed approach on both standard academic benchmarks (MMLIB) and an authentic modular construction dataset from a Singapore factory, providing a comprehensive empirical basis for our findings and offering practical guidance for implementation.

The rest of the paper is organized as follows. In Section 2, we review and analyze relevant literature, and, in Section 3, we introduce the probabilistic updating framework, including the policy representation, the reward signal, and the update rule. The experimental results and corresponding analysis are presented in Section 4. The work is concluded in Section 6 with suggested directions for future work.

2. Literature Review

Given their high relevance to this study, we provide a comprehensive review of project scheduling, metaheuristic approaches, and policy-based reinforcement learning.

2.1. Project Scheduling

The Resource-Constrained Project Scheduling Problem (RCPSP) is a foundational model in project scheduling, in which a set of activities must be sequenced under precedence and renewable resource constraints to minimize the project makespan [28]. The problem is NP-hard in the strong sense [7], motivating decades of research into exact, heuristic, and metaheuristic solution approaches [29,30]. In many real-world production environments, activities can be performed in multiple alternative modes, each with distinct duration and resource requirements. This gives rise to the Multi-Mode RCPSP (MRCPSP) [31]. The introduction of multiple modes significantly expands the solution space, as schedulers must jointly decide activity start times and execution modes. Furthermore, when multiple projects share a common pool of resources, the problem generalizes to the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MRCMPSP) [4]. The MRCMPSP captures the complexity of modern prefabrication factories, where several modular units are produced concurrently and compete for limited labor, equipment, and workspace [32,33].

2.2. Metaheuristic Approaches

Due to the NP-hard nature of MRCMPSP, exact methods such as mixed-integer linear programming [11] become computationally intractable for instances of realistic size. Consequently, a wide range of metaheuristics have been developed to obtain high-quality approximate solutions. Genetic Algorithms (GAs) are among the most widely applied techniques [34,35], often employing random-key encoding to represent activity sequences [36,37]. Hybrid GAs that combine evolutionary search with local improvement heuristics have demonstrated strong performance on multi-mode and multi-project benchmarks [14]. Differential Evolution (DE) has emerged as a competitive alternative to GAs for project scheduling [15,38]. DE's continuous encoding naturally accommodates both sequence priorities and mode scores, and its differential mutation operator provides robust

global exploration [39]. Advanced DE variants incorporate self-adaptive parameter control [40] and success-history-based adaptation [41] to balance exploration and exploitation dynamically. These adaptive mechanisms have been shown to improve convergence speed and solution quality on complex scheduling problems [23]. Other metaheuristics applied to MRCMPSP include Particle Swarm Optimization [42], Ant Colony Optimization [43], and hyper-heuristics that select among multiple low-level heuristics during search [44]. A comprehensive survey of hybrid metaheuristics for RCPSP is provided by [16], highlighting the trend toward integrating complementary search strategies.

2.3. Reinforcement Learning

Reinforcement learning (RL) has recently gained traction as a powerful tool for sequential decision-making in scheduling [19]. In the RL framework, an agent interacts with an environment, observes states, takes actions, and receives rewards, gradually learning a policy that maximizes cumulative return [17]. This paradigm aligns naturally with mode selection in MRCMPSP, where each mode choice affects both immediate resource consumption and downstream scheduling flexibility. Early applications of RL to project scheduling focused on learning priority rules for activity sequencing [45]. More recently, RL has been integrated with metaheuristics to guide operator selection or parameter adaptation. For example, Ref. [21] proposed a deep RL approach to select among multiple metaheuristics during the search for MRCMPSP. Similarly, Ref. [20] combined NSGA-II with Q-learning to adaptively choose execution modes, achieving improved trade-offs between makespan and resource utilization. RL has also been employed to handle stochastic and dynamic scheduling environments. Ref. [22] developed a multi-method framework where RL selects between different metaheuristics based on population diversity and solution quality under uncertain activity durations. Ref. [46] applied deep Q-networks to learn dispatching policies that generalize across project instances. These works demonstrate RL's capacity to learn adaptive strategies that outperform static heuristics in complex and uncertain settings. For multi-mode scheduling, most metaheuristics encode mode choices as discrete variables and evolve them. This coupling does not allow the mode selection component to accumulate learning across separate problem instances. This study changes this model by formulating mode selection as an independent learning task.

2.4. Policy-Based RL

Within the broader RL landscape, policy-based methods directly parameterize and optimize a policy $\pi_{\theta}(a|s)$ without requiring an explicit value function [47]. This family of algorithms is particularly well suited for discrete action spaces with moderate cardinality, such as mode selection in MRCMPSP, where the optimal policy may be inherently stochastic due to resource conflicts. The simplest policy gradient algorithm is REINFORCE, which updates policy parameters using Monte Carlo estimates of the return [48]. While unbiased, REINFORCE suffers from high variance, which can be mitigated by subtracting a learned baseline. Proximal Policy Optimization (PPO) improves training stability by constraining policy updates via a clipped surrogate objective [49]. PPO has become a standard baseline for discrete control tasks due to its robustness and ease of implementation. For continuous action spaces, Deep Deterministic Policy Gradient (DDPG) [18] and Twin Delayed DDPG (TD3) are prominent algorithms that learn deterministic policies. However, their applicability to discrete mode selection requires adaptation, typically by converting continuous outputs to categorical distributions via softmax. Soft Actor-Critic (SAC), an off-policy maximum-entropy algorithm, can also be adapted to discrete actions and offers improved exploration through entropy regularization [48]. Despite these advances, the application of policy-based RL to mode selection in MRCMPSP remains relatively unex-

plored. Existing works either treat mode selection as a static optimization variable within metaheuristics [14] or employ value-based RL to learn dispatching rules [20]. A systematic comparison of policy-based RL algorithms for mode selection, particularly in conjunction with state-of-the-art sequence optimizers like adaptive DE, has not been conducted.

2.5. Probabilistic Model Updating and Learning-Guided Metaheuristics

The proposed heuristic maintains and iteratively updates a categorical distribution over execution modes for each activity. This mechanism bears conceptual similarities to several established probabilistic optimization frameworks. The update rule that reinforces modes appearing in high-performing schedules while penalizing others is functionally analogous to the probability vector update in the cross-entropy method (CEM) [50], where a parametric distribution is iteratively updated toward elite solutions. Similarly, the pheromone update mechanism in ant colony optimization (ACO) [43] shares the same core idea: incrementally increasing the selection probability of solution components that contribute to high-quality solutions, while allowing decay of less promising components. The main differences in our implementation are that the update is driven by a normalized reward derived from schedule makespan, rather than by a predefined fitness function or heuristic information, and that the mode distribution is maintained independently of the sequence optimizer, whereas CEM and ACO are typically embedded as integral components of a single search process.

The proposed method is perhaps most closely related to estimation of distribution algorithms (EDAs), which build and sample probabilistic models of promising solutions in evolutionary search. EDAs have been applied to project scheduling [51], where probability distributions over activity sequences or mode assignments are updated based on selected elite individuals. In essence, our TPU variant implements a simple EDA specifically for the mode selection sub-problem, with the mode distribution serving as the probabilistic model. A notable implementation difference is that conventional EDAs typically re-estimate the distribution from scratch each generation using a set of elite samples, whereas our method performs incremental updates using the single best candidate of each generation, which can be viewed as a more conservative, memory-preserving variant of the same underlying principle.

Our framework also relates to the broader field of learning-guided metaheuristics [52], where machine learning techniques are integrated into metaheuristic search to improve efficiency and solution quality. While prior work has employed reinforcement learning to select among multiple metaheuristics [21,22] or to guide operator selection [20], our approach represents a complementary design choice: we apply a learning heuristic to a specific sub-problem (mode selection) in a decoupled manner, rather than as an auxiliary mechanism embedded within a single metaheuristic. This decoupling distinguishes our work from neural combinatorial optimization approaches that learn to construct solutions end-to-end using neural networks [45]; our method retains the traditional combinatorial optimization framework while delegating a well-defined sub-problem to a lightweight learning module.

A key practical feature of the proposed heuristic is that its update rule does not rely on gradient information from the sequence optimizer or require function approximation. This design choice preserves interpretability, minimizes computational overhead, and facilitates integration with any sequence-generating heuristic. The discussion above situates our contribution within the existing landscape of probabilistic model updating and learning-guided optimization, clarifying both its connections to established methods and the specific implementation choices that distinguish it from them.

3. Methods

While metaheuristics excel at global sequence exploration, they lack fine-grained adaptability for mode selection, often relying on fixed heuristics or co-evolving modes as continuous variables without learning from experience. RL has been applied to scheduling primarily for learning priority rules or selecting among metaheuristics, but a focused investigation of learning-based mode selection in MRCMPSP is missing. In addition, no study has systematically compared different learning algorithms (e.g., REINFORCE, PPO, SAC) under a consistent experimental framework to identify the most suitable approach for this problem class. In this section, we introduce a new method to address these gaps by conducting a comprehensive empirical evaluation of learning-based heuristics for mode selection in multi-mode project scheduling.

3.1. Problem Statement

The scheduling problem is the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MRCMPSP). A project consists of a set of activities, and each activity has a fixed order. Some activities must finish before others can start. These are called precedence constraints. The project also uses renewable resources. For example, a worker can be used for one activity and then becomes available again for later activities. The total amount of each resource is limited at any point in time. In the basic version of the problem, each activity has only one way to be executed. Its duration and resource demand are fixed. The scheduler only decides when each activity starts. This is already a hard problem. But in real factories, activities often have multiple execution modes. Figure 2 is an exploded schematic diagram of template construction. One mode might use three workers and take two days. Another mode might use one worker and take five days. The scheduler must choose not only the start time but also which mode to use for every activity. This is the multi-mode version. The problem becomes much harder because the number of possible combinations grows quickly.

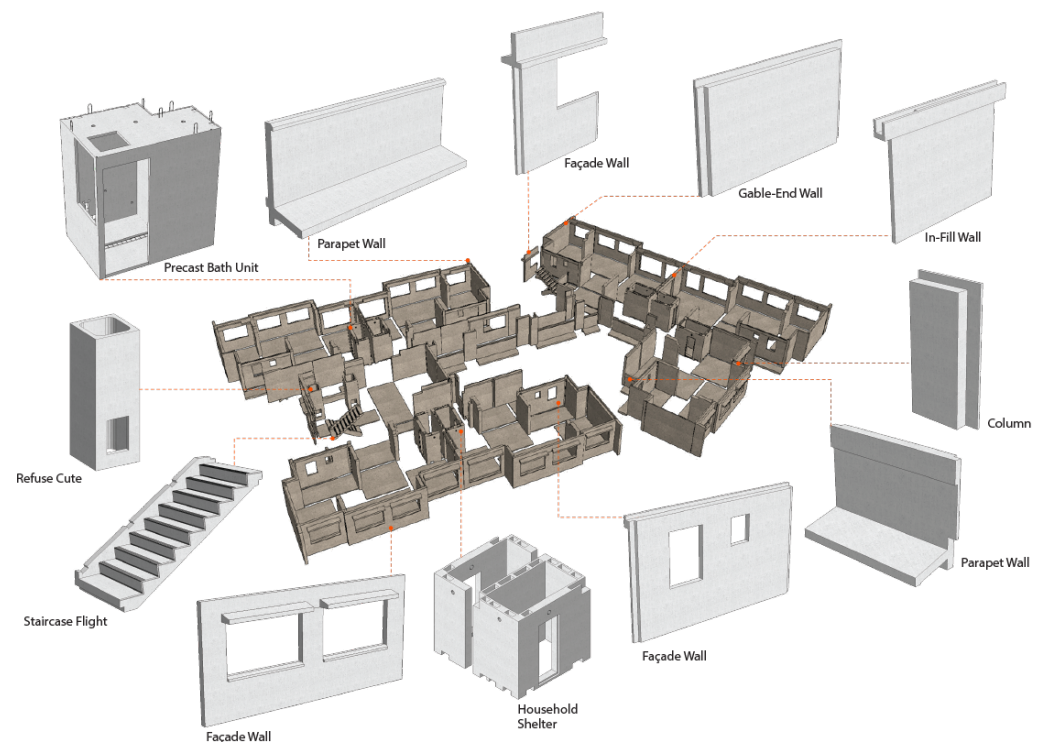


Figure 2. Modular room units schematic diagram.

In this paper, we focus on modular construction of projects, where Figure 3 compares the actual picture and the design drawing of a room. Extending to multiple projects is possible; this study focuses on the multi-project scenario using real-world MRU batches. The goal is to find a schedule that finishes the entire project as early as possible. That final completion time is called the makespan. Minimizing the makespan is the objective. Several constraints must be satisfied. The first constraint is precedence. For every pair of activities where one must come before the other, the successor cannot start until the predecessor finishes. The second constraint is mode uniqueness. Each activity must be assigned exactly one mode. That mode determines how long the activity takes and how many units of each resource it needs. The third constraint is resource capacity. At any moment, the total demand for a given resource across all active activities cannot exceed the available amount. All resources are renewable, meaning they return to the pool after an activity finishes.

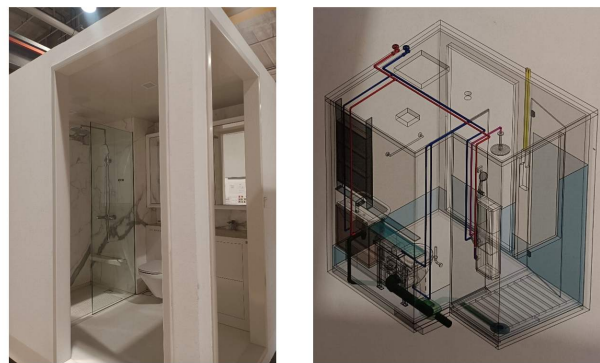


Figure 3. Real picture and design drawing under construction.

To build a schedule from a set of mode assignments, a schedule generation scheme is needed. This paper uses a parallel schedule generation scheme, or parallel SGS. The parallel SGS works by moving forward in time. It starts at time zero. At the current time, it looks at all activities whose predecessors have already finished. These are called eligible activities. For each eligible activity, the scheduler checks whether enough resources are available for its entire duration. If yes, the activity is scheduled to start at that time. The resources are reserved for the activity's duration. If multiple eligible activities can be scheduled at the same time, they all start together. This is what makes the scheme parallel. When no more eligible activities can be scheduled at the current time, the scheduler advances time to the next event. The next event is usually the finish time of some activity that is currently running. When that activity ends, its resources become free. The scheduler then repeats the process. It checks again which activities are now eligible and whether they can start. This continues until every activity has been scheduled. The parallel SGS has a useful property. It tends to pack activities tightly together. It does not leave idle time if there is work that can be done. This leads to shorter makespans compared with a serial scheme that schedules one activity at a time. The parallel SGS does require more computation because it checks all eligible activities at each time step. But for the problem sizes considered in this paper, the extra computation is acceptable.

The schedule builder takes two inputs. One input is the mode assignment for every activity. The other input is the order in which activities are considered when multiple are eligible. In a full scheduling algorithm, the activity order also matters. However, this paper focuses only on mode selection. Therefore, the activity order is fixed to a simple rule, such as the earliest start time or the smallest slack. This isolates the effect of mode choices. The parallel SGS then produces a complete schedule with start and finish times for all activities. The makespan of that schedule is the final output. This makespan is used as the performance measure for any given mode assignment.

3.2. Learning Algorithms for Comparison

Policy-based reinforcement learning (RL) directly optimizes a parameterized policy π_θ that defines the agent's behavior, without requiring an explicit value function. This family of methods is particularly attractive for mode selection in multi-mode project scheduling because the policy can naturally represent stochastic decisions and can be updated using episodic feedback from schedule evaluations. In this section, we review five representative policy-based algorithms: two stochastic policy gradient methods suitable for discrete action spaces (e.g., REINFORCE and PPO), two deterministic policy gradient methods designed for continuous control (e.g., DDPG and TD3), and one maximum-entropy stochastic method that bridges both settings (e.g., SAC). These algorithms are included as baselines for comparison; the proposed TPU heuristic is introduced separately in Section 3.3.

3.2.1. REINFORCE Algorithm

REINFORCE [53] is a foundational stochastic policy gradient algorithm. Given a trajectory $\tau = (s_0, a_0, r_0, \dots, s_T)$, it estimates the gradient of the expected return using Monte Carlo [48] samples from complete episodes. Let θ denote the parameters of the policy π_θ . A trajectory τ consists of a sequence of states s_t , actions a_t , and immediate rewards r_t obtained by following π_θ for T steps. The total return of a trajectory is $R(\tau) = \sum_{t=0}^T \gamma^t r_t$, where $\gamma \in (0, 1]$ is the discount factor. The objective is to maximize the expected return

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \quad (1)$$

where the policy parameters can be updated as follows,

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) G_t \quad (2)$$

Here, α is the learning rate, $\gamma \in [0, 1]$ is the discount factor, and $G_t = \sum_{k=t}^T \gamma^{k-t} r_k$ is the discounted return from time step t . REINFORCE is unbiased and requires no value function approximator, making it easy to implement in tabular settings. However, this method suffers from high variance, which can be mitigated by subtracting a learned baseline $b(s_t)$ (e.g., a state-value estimate) from G_t . In the mode selection context, where episodes correspond to complete schedule generations, a tabular variant of REINFORCE provides an effective learner.

3.2.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [49] improves upon standard policy gradient methods by constraining the size of policy updates, thereby enhancing training stability and sample efficiency. It optimizes a clipped surrogate objective function

$$L^{\text{clip}}(\theta) = \mathbb{E}_t \left[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right] \quad (3)$$

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}, \quad (4)$$

Here, $r_t(\theta)$ denotes the probability ratio between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$. $\hat{A}_t$ is the estimated advantage function (typically computed using Generalized Advantage Estimation), which measures the relative benefit of taking action a_t in state s_t compared with the average action. The hyperparameter ϵ (usually set to 0.1 or 0.2) controls the clipping range to prevent excessively large policy updates. PPO alternates between collecting trajectories using the current policy and performing multiple epochs of mini-batch updates on the clipped objective. Its robustness and ease of implementation have made PPO a

standard baseline for discrete and continuous control tasks. In the scheduling domain, PPO can be used with a neural network policy that conditions mode selection on resource availability and project progress features.

3.2.3. Soft Actor–Critic

The off-policy maximum-entropy RL algorithm, Soft Actor–Critic (SAC) [54], augments the standard RL objective with an entropy bonus that encourages stochasticity and exploration, i.e.,

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T \gamma^t (r(s_t, a_t) + \alpha_{\text{ent}} \mathcal{H}(\pi_{\theta}(\cdot|s_t))) \right] \quad (5)$$

$$\mathcal{H}(\pi(\cdot|s)) = -\mathbb{E}_{a \sim \pi} [\log \pi(a|s)], \quad (6)$$

where $\mathcal{H}$ is the entropy of the policy, and α_{ent} is a temperature parameter that balances reward maximization and entropy maximization. Note that, SAC maintains a stochastic policy (typically Gaussian for continuous actions, or a softmax distribution for discrete actions), along with two Q-function critics and a target value network. The policy is updated by minimizing the Kullback–Leibler divergence between the policy and a Boltzmann distribution induced by the Q-function

$$J_{\pi}(\theta) = \mathbb{E}_{s \sim \mathcal{D}} [\mathbb{E}_{a \sim \pi_{\theta}} [\alpha_{\text{ent}} \log \pi_{\theta}(a|s) - Q_{\phi}(s, a)]] \quad (7)$$

The symbol $\mathcal{D}$ represents the replay buffer used to store and sample past transitions for off-policy training. This entropy-regularized objective is particularly appealing in scheduling, where maintaining exploration of diverse mode combinations is crucial.

3.2.4. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [55] extends the deterministic policy gradient framework to deep neural networks and continuous action spaces. It employs an actor–critic architecture with two networks, i.e., an actor $\mu_{\theta}(s)$ that outputs a deterministic action, and a critic $Q_{\phi}(s, a)$ that evaluates state-action pairs. The critic is updated by minimizing the temporal-difference error on a replay buffer $\mathcal{D}$, that is,

$$L_Q(\phi) = \mathbb{E}_{(s, a, r, s') \sim \mathcal{D}} [(Q_{\phi}(s, a) - y)^2] \quad (8)$$

$$y = r + \gamma Q_{\phi'}(s', \mu_{\theta'}(s')). \quad (9)$$

Here y is the target value computed using target networks ϕ' and θ' . The actor is updated by applying the chain rule to the expected return

$$\nabla_{\theta} J \approx \mathbb{E}_{s \sim \mathcal{D}} [\nabla_a Q_{\phi}(s, a)|_{a=\mu_{\theta}(s)} \nabla_{\theta} \mu_{\theta}(s)]. \quad (10)$$

DDPG is designed for continuous action domains and is not directly applicable to discrete mode selection. However, it serves as an important conceptual bridge to methods that learn deterministic mode preferences, and it has inspired extensions like Twin Delayed DDPG (TD3) that improve robustness.

3.2.5. Twin Delayed DDPG

Twin Delayed DDPG (TD3) [56] achieves state-of-the-art performance on continuous control benchmarks and is more robust than DDPG. It addresses the three key limitations of DDPG, including overestimation bias, sensitivity to hyperparameters, and training instability, by three approaches, respectively, as follows:

1. **Clipped Double Q-learning.** Given the target policy smoothing noise ϵ , TD3 maintains two independent critic networks Q_{ϕ_1} and Q_{ϕ_2} , and uses the minimum of their outputs to compute the target value, reducing overestimation through

$$y = r + \gamma \min_{i=1,2} Q_{\phi_i}(s', \mu_{\theta'}(s') + \epsilon), \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c). \quad (11)$$

2. **Delayed Policy Updates.** The actor network and target networks are updated less frequently than the critics (e.g., every two critic updates), which improves stability.
3. **Target Policy Smoothing.** Adding noise to the target action regularizes the policy and prevents exploitation of Q-function errors.

Like DDPG, TD3 is intended for continuous action spaces and cannot be directly applied to discrete mode selection. Nevertheless, its innovations in reducing overestimation bias and stabilizing training are relevant to the design of any actor–critic system, including those adapted for discrete actions.

For completeness, DDPG and TD3 are included in our comparative evaluation despite their original design for continuous action spaces. To enable mode selection, we adapt these algorithms as follows. The actor network outputs a continuous score vector $\mathbf{z} \in \mathbb{R}^{M_{i,j}}$ for each activity j , where $M_{i,j}$ is the number of available modes for that activity. The final action is sampled from a categorical distribution obtained by applying the softmax function to $\mathbf{z}$, i.e., $a \sim \text{Categorical}(\text{softmax}(\mathbf{z}))$. For exploration, Gaussian noise $\mathcal{N}(0, \sigma_{\text{exp}}^2)$ is added directly to $\mathbf{z}$ before the softmax transformation, with $\sigma_{\text{exp}} = 0.3$ for DDPG. The critic network receives as input the concatenated state vector (activity index and a binary indicator of mode selection) and the action $\mathbf{z}$, and outputs a scalar Q-value. The actor is updated by backpropagating the critic's gradient through the softmax operation, which provides a valid (though biased) gradient signal. TD3 adopts the same architecture with twin critics, target policy smoothing noise $\sigma_{\text{smooth}} = 0.2$ and clip bound $c = 0.5$, and delayed policy updates every two critic updates. All other hyperparameters are kept consistent with the continuous-action DDPG and TD3 literature. We acknowledge that this adaptation is heuristic and may not fully exploit the strengths of these algorithms; the results for DDPG and TD3 are included primarily for reference to illustrate the challenges of applying deterministic continuous-control methods to discrete combinatorial selection problems.

3.2.6. Applicability to Mode Selection and Summary

Among the five algorithms reviewed, REINFORCE and PPO are the most natural fits for discrete mode selection problems because they natively support stochastic policies over categorical action spaces. A tabular implementation of REINFORCE can serve as a lightweight baseline without neural networks, making it suitable for integration with population-based sequence optimizers. PPO offers a more scalable alternative when state features (e.g., resource utilization, project progress) are high-dimensional and a neural policy is desired. SAC can also be adapted to discrete actions and provides an entropy-regularized exploration mechanism that may benefit highly constrained scheduling environments. DDPG and TD3, while powerful for continuous control, require non-trivial modifications to handle discrete actions and are therefore less directly applicable; they are included here primarily for completeness and to contrast with stochastic policy methods.

3.3. Mode Selection Approach

While evolutionary algorithms such as Learning-Enhanced Differential Evolution (LEDE) excel at exploring the global space of activity sequences, the selection of execution modes for individual activities remains a distinct and challenging sub-problem. Mode choices can significantly affect resource utilization and overall schedule quality. Fixed heuristics (e.g., always selecting the fastest mode) often fail to account for these effects because they treat each activity in isolation. Embedding mode selection as a continuous variable within an evolutionary framework often couples it too tightly to sequence exploration, limiting fine-grained adaptability.

To address these limitations, we decouple mode selection from activity sequencing and treat it as a separate probabilistic learning problem. In this decoupled design, the mode selector does not share parameters or population members with the sequence optimizer. During policy updates, the activity order is fixed (e.g., earliest start time), and the learning module learns a mode preference distribution independently. The same learned preferences can then be applied to any sequence generated by an external optimizer, with or without retraining. Figure 4 shows the interaction between the probabilistic updating module and LEDE. The module maintains a policy that maps each activity to a probability distribution over its available execution modes. This policy is learned entirely through interaction with the schedule builder, without requiring hand-crafted rules or gradient information from the sequence optimizer.

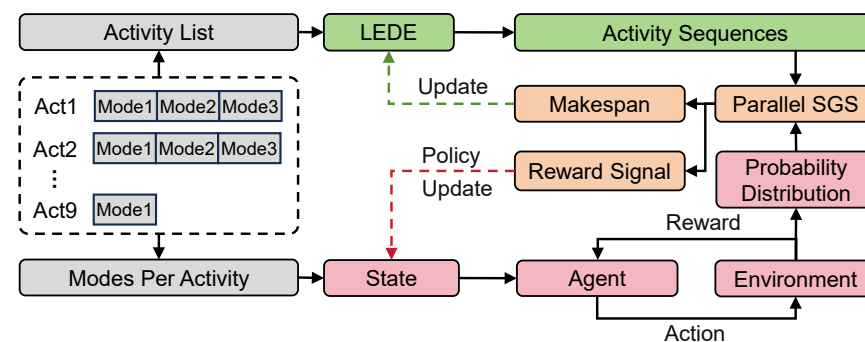


Figure 4. Schematic diagram of the proposed probabilistic updating heuristic for mode selection. Solid arrows indicate forward data flow. The data portion is shown in gray. The reinforcement learning part is shown in red. LEDE is shown in green. The orange part in the middle is SGS and updates.

Figure 4 illustrates the interaction between the probabilistic updating module and the LEDE sequence optimizer. The data flow proceeds as follows. In the figure, solid arrows indicate forward data flow, while dashed arrows indicate feedback updates from schedule evaluation to both optimization modules. At the start of each generation, the LEDE optimizer proposes a population of activity sequences based on its current population and mutation/crossover operators. For each activity in a given sequence, the learning module samples an execution mode according to the current probability distribution $\pi_{i,j}(m)$, which is maintained independently for each activity. The sequence and mode assignments are then passed to the parallel SGS, which constructs a feasible schedule by respecting precedence and resource constraints, and computes the resulting makespan $C_{\max}$. This makespan serves two purposes: it provides the fitness value for LEDE selection, and it supplies the raw reward signal $R = -C_{\max}$ for the probability update. From the set of candidate schedules generated in the current generation, the best-performing candidate k^* (the one with the smallest $C_{\max}$) is identified. Its makespan is used to compute the normalized reward $\hat{r}$, which scales the update magnitude. The policy is then updated by

reinforcing the mode choices $m_{i,j}^{(k^*)}$ that appeared in the best candidate, shifting probability mass toward those modes for each activity. This cycle repeats for each generation, with the policy gradually converging to a mode preference distribution that complements the sequence search performed by LEDE.

3.3.1. Policy Representation and Sampling

For a project comprising N activities and the set of feasible modes, $\mathcal{M}_j$, for activity j , the policy is represented by a collection of independent categorical distributions $\pi_j(m) = P(\text{mode} = m \mid \text{activity} = j)$, with $\sum_{m \in \mathcal{M}_j} \pi_j(m) = 1$. This policy is stored in a simple tabular form, which is both computationally efficient and sufficient for the discrete, moderate-sized mode spaces encountered in prefabrication scheduling. During each scheduling episode, the module generates a complete mode assignment by sampling independently for each activity according to the current policy

$$m_j \sim \pi_j(\cdot), \quad \forall j \in \{1, \dots, N\}. \quad (12)$$

The sequence of activities is provided externally (for example, by a population-based meta-heuristic such as LEDE) and is treated as fixed during the mode-sampling and policy-update phases. This separation allows the learning component to focus exclusively on learning mode preferences, while the sequence optimizer continues to evolve activity orderings.

3.3.2. Schedule Evaluation and Reward Signal

Given an activity sequence and the sampled mode assignment, a feasible schedule is constructed using a parallel schedule generation scheme (SGS) that respects both precedence and renewable resource constraints. The quality of the resulting schedule is measured by its makespan $C_{\max}$, defined as the completion time of the final activity across all projects. Since shorter makespans are preferable, we define the raw reward for an episode as $R = -C_{\max}$. To stabilize learning, we employ a normalized reward based on the range of makespans observed in the current generation of candidates. Let C_{best} and C_{worst} be the smallest and largest makespans encountered up to the current iteration. For a candidate schedule with makespan C , the normalized reward can be

$$\hat{r} = \frac{C_{\text{worst}} - C}{C_{\text{worst}} - C_{\text{best}} + \epsilon}, \quad (13)$$

where ϵ is a small positive constant to avoid division by zero. This transformation maps the reward to the interval $[0, 1]$, with $\hat{r} = 1$ corresponding to the best schedule seen so far.

3.3.3. Policy Update Rule

After evaluating a generation of K (the number of candidate mode assignments sampled per generation) candidate schedules (each generated by sampling modes from the current policy and using a fixed sequence), we select the best-performing candidate, i.e., the one with the smallest makespan. Given the mode chosen for activity j in that best candidate, m_j^* (i.e., the mode selected for activity j in the best-performing candidate of the current generation), the policy is then updated to reinforce the modes that contributed to the best outcome, i.e.,

$$\pi_j(m) \leftarrow \begin{cases} \pi_j(m) + \alpha \hat{r} (1 - \pi_j(m)), & \text{if } m = m_j^*, \\ (1 - \alpha \hat{r}) \pi_j(m), & \text{otherwise,} \end{cases} \quad (14)$$

where $\alpha \in (0, 1)$ is a learning rate that controls the update magnitude. The probabilities are subsequently renormalized to sum to unity. This update rule is a simple incremental reinforcement update that increases the likelihood of actions appearing in high-performing trajectories while gradually reducing the probability of other modes. Unlike the REINFORCE gradient update defined in Equation (2), this rule directly modifies the probability masses of the categorical policy using the normalized reward $\hat{r}$ as a scalar update magnitude, without requiring explicit gradient computation of a parameterized policy.

It is important to clarify the distinction between the proposed tabular probability updating (TPU) heuristic and the standard REINFORCE algorithm. While standard REINFORCE is derived analytically from the log-policy gradient theorem within a fully modeled Markov Decision Process (MDP), the proposed TPU variant operationalizes a simplified, stateless probability reinforcement paradigm. Because the mode selection subproblem in MRCMPSP is decoupled from the sequence optimization, explicitly modeling high-dimensional state vectors and complex transition probabilities among activities would trigger a combinatorial state explosion. Therefore, the implemented tabular rule does not rely on computing explicit log-policy gradients $\nabla_{\theta} \log \pi_{\theta}(a|s)$. Instead, it employs a linear reward-penalty reinforcement heuristic rooted in classical *Learning Automata* (LA) and stochastic search. This mechanism directly shifts probability mass toward the elite decisions ($m_{i,j}^{(k*)}$) discovered in the current generation based on the normalized scalar reward $\hat{r}_{\text{final}}$.

3.3.4. Maintaining Exploration

A known risk of such reinforcing updates is premature convergence to suboptimal mode combinations, especially when early random samples produce misleadingly good schedules. To mitigate this, we enforce a minimum probability threshold $\pi_{\min}$. After each update, any $\pi_j(m) < \pi_{\min}$ is reset to $\pi_{\min}$, and the distribution is renormalized. This mechanism guarantees that every mode retains at least a small chance of being selected, thereby preserving exploration throughout the learning process. Typical values for $\pi_{\min}$ lie in the range $[0.05, 0.1]$, as determined by preliminary sensitivity analyses.

3.3.5. Integration with Sequence-Search Heuristics

The probabilistic updating module operates orthogonally to the algorithm responsible for activity sequencing. In practice, it can be paired with any sequence-generating heuristic, including priority rules, Genetic Algorithms, or adaptive Differential Evolution variants such as LEDE. In a typical hybrid workflow, the sequence optimizer (e.g., LEDE) proposes one or more activity orderings; for each, the learning module samples a set of execution modes. The schedule builder decodes the combined (sequence, modes) into a full schedule and computes the makespan that serves as both the fitness value for the sequence optimizer and the reward signal for the probability update. The policy is updated using the best candidate of the current generation, while the sequence optimizer evolves its population based on the same fitness values.

This modular design leverages the global search capabilities of population-based metaheuristics for activity ordering, a problem for which evolutionary algorithms are well suited. Additionally, it delegates the local, context-sensitive mode decisions to a lightweight learning module that continuously improves through experience. Third, the separation of concerns simplifies implementation and tuning: the sequence optimizer requires no modification to its internal operators, and the learning module does not need to model complex precedence constraints. The resulting framework thus combines the exploratory power of evolutionary search with the adaptive, experience-driven refinement of probabilistic updating, yielding a robust and scalable solution for mode selection in multi-mode project scheduling.

3.4. Integrated Algorithm of Probabilistic Updating and LEDE

The proposed hybrid LEDE with probabilistic updating framework for solving the MRCMPSP is outlined in Algorithm 1. The framework alternates between two complementary search mechanisms. The outer loop is driven by a LEDE that evolves a population of activity sequences. Within each generation, the sequence optimizer proposes a set of candidate orderings, while an independent probabilistic updating module provides mode assignments by sampling from categorical distributions maintained for each activity. These joint (sequence, mode) solutions are decoded into complete schedules via a parallel schedule generation scheme, and the resulting makespan serves simultaneously as fitness for the LEDE selection step and as a reward signal for the policy update. After evaluating all candidates, the probability distribution is refined by reinforcing the mode choices that appear in the best-performing schedule of the current generation, using a normalized reward and a minimum probability threshold to preserve exploration. The decoupling between mode selection and sequence optimization is implemented at the module level. The probability distribution is stored and updated independently of the LEDE population. LEDE evolves activity sequences using its own internal parameters (scaling factor, crossover rate, etc.), while the learning module maintains a separate tabular policy $\pi_{i,j}(m)$. During each generation, the best sequence found by LEDE is passed to the learning module as a fixed ordering for mode sampling and policy update. No parameters are shared between the two modules, and the learned preferences can be transferred to other sequence optimizers without retraining.

The following mathematical definitions are used. For a candidate solution k , let $C_{\text{portfolio}}^{(k)}$ denote the project portfolio makespan. It is defined as the maximum completion time over all projects

$$C_{\text{portfolio}}^{(k)} = \max_{i \in \text{POR}} C_i^{(k)}, \quad C_i^{(k)} = \sum_{t=EF_{i,J+1}}^{LF_{i,J+1}} t x_{i,J+1,1,t} \quad (15)$$

where $C_i^{(k)}$ is the completion time of the last activity of project i , obtained from the binary decision variable $x_{i,J+1,1,t}$ in the schedule representation.

Let C_{best} and C_{worst} be the smallest and largest portfolio makespan values observed over all generations up to the current point. After evaluating a generation, these global bounds are updated through

$$C_{\text{best}} \leftarrow \min(C_{\text{best}}, C_{\text{portfolio}}^{(k^*)}), \quad C_{\text{worst}} \leftarrow \max(C_{\text{worst}}, C_{\text{portfolio}}^{(k^*)}) \quad (16)$$

where k^* is the index of the best candidate in the current generation, defined by

$$k^* = \arg \min_k C_{\text{portfolio}}^{(k)}. \quad (17)$$

The normalized reward $\hat{r}_{\text{final}}$ used for policy update is computed from the current best candidate's makespan

$$\hat{r}_{\text{final}} = \frac{C_{\text{worst}} - C_{\text{portfolio}}^{(k^*)}}{C_{\text{worst}} - C_{\text{best}} + \epsilon} \quad (18)$$

with ϵ a small constant to avoid division by zero.

Algorithm 1 Integrated Algorithm of Probabilistic Updating and LEDE Sequence Optimizer

Require: Activity durations, precedence constraints, resource availability, number of projects n , number of candidate solutions K , maximum iterations Max_{iter} .

Ensure: Best activity sequence and best mode selection (as defined in Equation (22)).

```

1: Initialize all parameters
2: Randomly initialize the mode selection policies  $\pi_{i,j}(m)$  for each activity  $a_{i,j}$ 
3: Initialize mode score vectors for each candidate
4: for each candidate index  $k$  do
5:   Randomly generate a feasible activity sequence  $sequence_k$ 
6:   Sample a mode assignment  $m_{i,j}^{(k)}$  from the current  $\pi_{i,j}(m)$  for each activity
7:   Form the candidate  $P_k$  according to Equation (21)
8: end for
9: Set learning rate  $\alpha = 0.1$  and global best makespan  $C_{best} = \infty$ 
10: for iteration  $t = 1$  to  $Max_{iter}$  do ▷ Sequence optimization with LEDE
11:   Apply DE mutation strategy according to the current phase
12:   Apply binomial crossover with adaptive CR
13:   Form trial vector  $\vec{u}_l$  and decode it into an activity sequence (without mode assignment)
14:   Evaluate fitness using the decoded schedule's makespan
15:   Leverage greedy selection to update trial vector
16:   Update the best sequence found so far
17:   if stagnation is detected then
18:     Reinitialize the worst 30% of the population
19:   end if
20:   for each activity  $a_{i,j}$  do ▷ Mode assignment by sampling from probability table
21:     For each candidate  $k$ , sample execution mode  $m_{i,j}^{(k)} \sim \pi_{i,j}(m)$ 
22:   end for
23:   Construct complete candidate solutions  $P_k$  defined in Equation (21)
24:   Decode each  $P_k$  using the parallel SGS to obtain a feasible schedule
25:   Compute  $C_{portfolio}^{(k)}$  for each candidate using Equation (15)
26:   Identify the best candidate index  $k^*$  as defined in Equation (17)
27:   Update  $C_{best}$  and  $C_{worst}$  using  $C_{portfolio}^{(k^*)}$  according to Equation (16)
28:   Compute normalized reward  $\hat{r}_{final}$  via Equation (18)
29:   for each activity  $a_{i,j}$  do ▷ Probability update based on the best candidate
30:     Update policy  $\pi_{i,j}(m)$  using Equation (19) with  $\hat{r}_{final}$ 
31:     Enforce probability normalization via Equation (20)
32:   end for
33:   if  $C_{portfolio}^{(k^*)} < C_{best}$  then ▷ Global best update
34:     Update the stored activity sequence and mode configuration in Equation (22)
35:   end if
36: end for
37: return The best schedule configuration  $P^*$  defined in Equation (22)

```

Let $\pi_{i,j}(m)$ denote the probability of selecting mode m for activity $a_{i,j}$. For the best candidate k^* , denote the mode selected for activity $a_{i,j}$ as $m_{i,j}^{(k^*)}$. The policy is updated by reinforcing the mode choices that appear in the best candidate. For each activity, if a mode m equals the best mode $m_{i,j}^{(k^*)}$, its probability is increased; otherwise it is decreased

$$\pi_{i,j}(m) \leftarrow \begin{cases} \pi_{i,j}(m) + \alpha \hat{r}_{final} (1 - \pi_{i,j}(m)), & \text{if } m = m_{i,j}^{(k^*)} \\ (1 - \alpha \hat{r}_{final}) \pi_{i,j}(m), & \text{otherwise} \end{cases} \quad (19)$$

where α is the learning rate. After the update, the probability distribution for each activity must be normalized to sum to unity

$$\sum_{m=1}^{M_{i,j}} \pi_{i,j}(m) = 1 \quad (20)$$

Define a complete candidate solution P_k as a tuple consisting of an activity sequence and a mode assignment for all activities

$$P_k = (\text{sequence}_k, \{m_{i,j}^{(k)}\}) \quad (21)$$

Similarly, the best schedule configuration found at termination is denoted by

$$P^* = (\text{sequence}^*, \{m_{i,j}^*\}) \quad (22)$$

The learning module is a separate module that does not share population parameters with the sequence optimizer. It receives the best sequence of each generation only as a fixed ordering for policy updates. The complete procedure is given in Algorithm 1.

3.5. Implementation Framework and Key Components

We acknowledge that the generic algorithm descriptions in Section 3.2 are provided primarily for completeness and self-containment, enabling readers unfamiliar with the baseline learning algorithms to follow the comparative evaluation. However, the following implementation details are critical for reproducing our results and are therefore elaborated here.

3.5.1. Schedule Generation Scheme

All schedules are constructed using a parallel schedule generation scheme (parallel SGS). At each time instant, the scheduler identifies all activities whose predecessors have been completed (eligible activities). For each eligible activity, the scheduler checks whether sufficient renewable resources are available for the duration corresponding to its assigned execution mode. If resources are sufficient, the activity is scheduled to start at the current time, and the required resources are reserved for its duration. This process continues until no further eligible activities can be scheduled, at which point time advances to the next event (the earliest completion time among currently executing activities). The parallel SGS is chosen for its tendency to produce dense, resource-efficient schedules with shorter makespans compared with serial SGS. The computational complexity of the parallel SGS is $O(n^2 \cdot m)$, where n is the number of activities and m is the maximum number of modes per activity, which remains acceptable for the instance sizes considered in this study (up to 15 MRUs, 135 activities).

3.5.2. Learning-Enhanced Differential Evolution (LEDE)

LEDE is a variant of Differential Evolution that incorporates adaptive parameter control and a stagnation recovery mechanism. The population size is fixed at 30 individuals, each encoding a feasible activity sequence using a random-key representation. The mutation strategy switches dynamically based on the search phase: DE/rand/1 for early exploration, DE/best/1 or DE/rand-to-best/1 for mid-phase exploitation, and DE/current-to-best/1 for late-stage fine-tuning. The crossover rate CR is self-adapted using a success-history-based mechanism [41], where the mean of historically successful CR values is maintained and updated each generation. The scaling factor F is sampled from a normal distribution $\mathcal{N}(0.5, 0.3)$ and truncated to $[0, 1]$. When stagnation is detected

(i.e., no improvement in the best fitness for 10 consecutive generations), the worst 30% of the population is reinitialized. LEDE was selected as the primary sequence optimizer in our internal comparison because it has been shown to outperform both standard GA and canonical DE on MRU scheduling benchmarks in preliminary experiments.

3.5.3. Tabular Policy Data Structure

The probability distribution is stored as a two-dimensional table $\Pi \in \mathbb{R}^{N \times M_{\max}}$, where N is the number of activities and $M_{\max}$ is the maximum number of modes across all activities. Entries corresponding to infeasible mode-activity combinations are masked with -1 and excluded from sampling and update operations. For each activity j , the valid mode probabilities sum to 1. The policy is initialized uniformly for each activity: $\pi_j(m) = 1/|\mathcal{M}_j|$ for all feasible modes.

3.5.4. Derivation and Implementation of the Policy Update Rule

The policy update in Equation (14) directly modifies the probability mass of the categorical distribution for each activity. For a given activity j and its best mode m_j^* in the best candidate of the current generation, the update first increases the probability of m_j^* by $\alpha \hat{r}(1 - \pi_j(m_j^*))$, and then decreases all other modes proportionally by multiplying their probabilities by $(1 - \alpha \hat{r})$. The normalization step ensures that $\sum_m \pi_j(m) = 1$ after each update. This update rule is equivalent to performing a gradient step on the categorical policy with the normalized reward $\hat{r}$ as the step magnitude, under the constraint that probabilities remain non-negative. The learning rate α controls the update aggressiveness and is set to 0.1 based on the sensitivity analysis in Section 4.2. The normalized reward $\hat{r}$ is computed using the global best and worst makespans observed across all generations, providing a consistent scaling of the update magnitude regardless of the absolute makespan scale.

3.5.5. Algorithmic Integration and Execution Flow

The integration of probabilistic updating with LEDE follows a generation-level interaction. In each generation, LEDE proposes a population of K activity sequences. For each sequence, the learning module samples a mode assignment independently for each activity, and the parallel SGS decodes the complete (sequence, mode) solution. After evaluating all K candidates, the best candidate's makespan is used to compute the normalized reward, and the policy is updated once using this reward. Importantly, the policy update is performed only once per generation, not per candidate, and the policy is shared across all candidates in the population. This design balances learning efficiency with computational cost. The total number of generations Max_{iter} is set to 100 for all experiments, which was determined by preliminary runs to be sufficient for convergence.

3.5.6. Experimental Design and Reproducibility

All experiments are performed on a dedicated workstation with an Intel Xeon CPU and 64 GB RAM, using Python 3.10. For each experimental configuration (batch size, workforce level, learning algorithm, sequence optimizer), 10 independent runs are executed with different random seeds to account for stochastic variability. Reported makespans are averages and standard deviations over these runs. The parallel SGS, LEDE, and all learning algorithm implementations are custom-built and do not rely on external optimization libraries, ensuring full control over all algorithmic components and enabling fair comparisons across methods.

4. Experiments and Results

We conduct the experiments on real-world data collected from a modular room units factory in Singapore, in which various learning methods are compared.

4.1. Datasets

Figure 5 shows the internal construction of the factory that produces MRUs in batches. Each MRU follows the same sequence of nine activities. These activities include tasks like mould preparation, concrete casting, interior finishing, and quality inspection. Every activity has multiple execution modes. For example, concrete pouring can be done with two workers in four hours or with four workers in two hours. The dataset specifies the duration and the number of workers required for each mode of each activity. Different batch sizes are tested. Small batches have three or six MRUs. Medium batches have nine or twelve MRUs. Large batches have fifteen MRUs. To check scalability, some experiments were also run with fifty or one hundred MRUs. The number of available workers was varied from twenty to forty. This covers realistic factory conditions where labor supply may change from week to week. All data instances respect the same precedence structure. An activity for one MRU does not depend on the activities of another MRU. The only coupling is through shared workers.

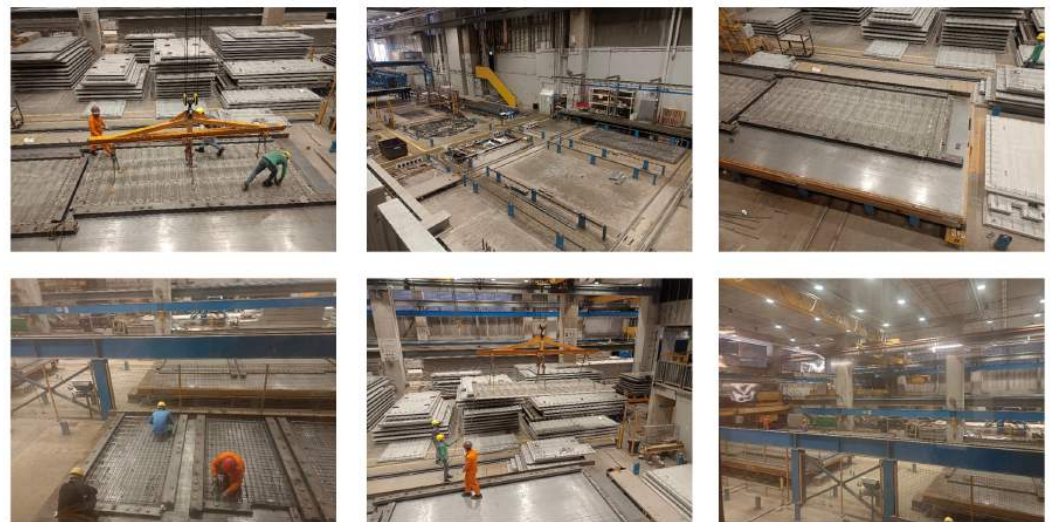


Figure 5. Construction conditions inside the factory.

4.2. Settings

The proposed probabilistic updating method is compared against two simple baselines. The first baseline is a fixed rule. The rule always picks the fastest mode for every activity. This minimizes the duration of each individual task. But it also consumes the most resources at every step. Fast modes often require more workers, so resource conflicts become frequent. The schedule builder then has to delay many activities. The final makespan can be long despite each task being short. The second baseline is random selection. For each activity, a mode is chosen uniformly at random from the available modes. This is repeated independently for each activity. The random baseline has no learning. It serves as a lower bound for what a blind search can achieve. The learning-based agent should outperform random selection; otherwise, the learning is not useful. The activity order is kept the same across all methods. The order is based on the earliest start time. This removes the effect of sequencing choices. Any difference in makespan comes purely from mode selection. Each method is run multiple times with different random seeds. The reported makespans are averages over these runs. Each experiment was repeated 10 times with different random seeds (1 to 10). The reported makespans are averages and standard deviations over these 10 runs.

To ensure full reproducibility of all experimental results, we specify the parameter configurations used in our implementation. For the TPU mode selector, the learning rate is

set to $\alpha = 0.1$, the minimum probability threshold is $\pi_{\min} = 0.1$, and reward normalization is enabled. For the neural-network-based algorithms (PPO, SAC, DDPG, and TD3), all actor and critic networks have two hidden layers with 64 units each and ReLU activations. The Adam optimizer is used with a learning rate of 3×10^{-4} for both actor and critic (except SAC, which uses 3×10^{-4} for policy and Q-networks). The discount factor is $\gamma = 0.99$ for all methods. For PPO, the clipping range is $\epsilon = 0.2$, the GAE parameter $\lambda = 0.95$, and 10 epochs per update with a mini-batch size of 64. SAC employs automatic entropy tuning with target entropy $-\log \pi$, a replay buffer of size 10^6 , batch size 256, and soft update coefficient $\tau = 0.005$. DDPG and TD3 use the same replay buffer size and batch size of 100, with target smoothing noise $\sigma_{\text{smooth}} = 0.2$ and clip bound $c = 0.5$ for TD3, and TD3 delays policy updates every two critic updates. The LEDE sequence optimizer uses a population size of 30, mutation strategy switching dynamically (DE/rand/1, DE/best/1, DE/current-to-best/1), crossover rate CR adapted via success-history, scaling factor $F \sim \mathcal{N}(0.5, 0.3)$ truncated to $[0, 1]$, and stagnation recovery after 10 generations without improvement (reinitializing the worst 30%). For random seed control, we fix seeds 1 through 10 for all main experiments; for the additional box plot analysis (Figure 6) and stability evaluation result, we extended to 20 runs (seeds 1–20) to obtain more robust distribution statistics. All experiments were conducted on a dedicated workstation with an Intel Xeon Gold 6248 CPU and 64 GB RAM, running Ubuntu 22.04 LTS with Python 3.10.

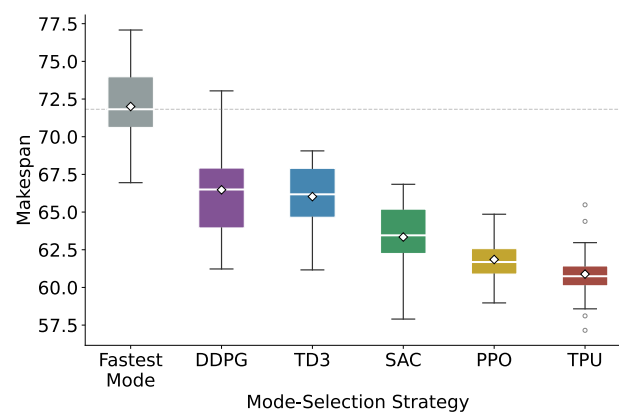


Figure 6. Distribution of final makespan for various mode selection strategies.

For clarity and reproducibility, we provide the following explanation of the axes, legends, variable symbols, and experimental conditions common to Figures 7–9. The horizontal axis in convergence plots represents the generation number (1 to 100), while the vertical axis represents the average makespan (in hours) over 10 independent runs. Shaded regions or error bars indicate the standard deviation across runs. In Figure 7, the three curves correspond to different learning rate settings ($\alpha = 0.05, 0.1, 0.2$), with $\pi_{\min} = 0.1$ fixed. Figure 8 presents the signal-to-noise (S/N) ratio from the Taguchi-style parameter tuning experiment; the horizontal axis lists the evaluated combinations of α and $\pi_{\min}$, and the vertical axis shows the S/N ratio (in decibels) calculated using the “smaller-the-better” objective, where higher (less negative) values indicate better performance. Figure 6 displays the distribution of final makespan for different learning algorithms; each box spans the interquartile range (IQR), the white horizontal line indicates the median, the white diamond indicates the mean, whiskers extend to $1.5 \times \text{IQR}$, and hollow circles denote outliers. Figure 9 is a grouped bar chart comparing makespan with and without learning-enhanced mode selection across three sequence optimizers (GA, DE, LEDE); the left vertical axis represents makespan (in hours), and the numbers above each bar indicate the percentage improvement achieved by the learning-based selector over the fastest-mode baseline. All figures are based on the 9-MRU instance with 25 workers unless otherwise specified in

the corresponding figure caption. Throughout this section, percentage improvements are calculated relative to the fastest-mode baseline unless otherwise specified. Where comparisons are made against LEDE without the learning component, this is explicitly stated to avoid confusion.

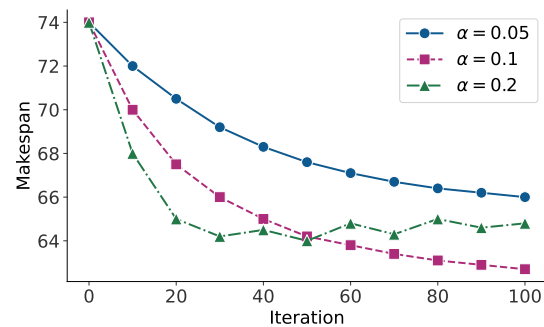


Figure 7. Sensitivity of the learning rate α .

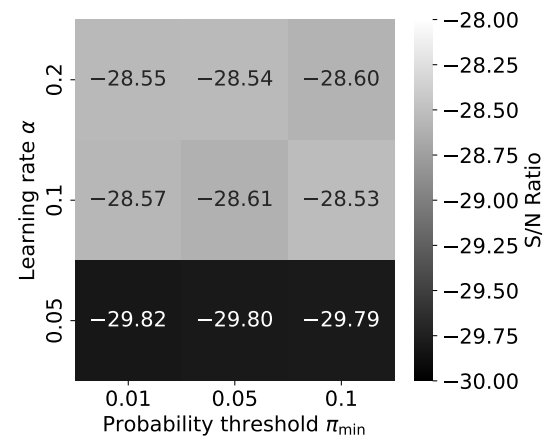


Figure 8. S/N analysis from the parameters (α and $\pi_{\min}$).

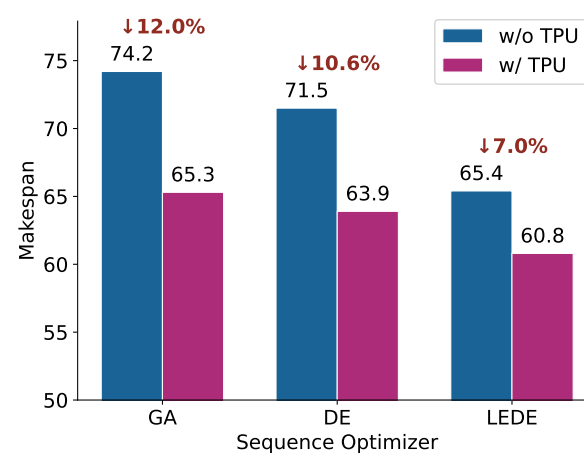


Figure 9. Makespan comparison by learning-based mode selector across three sequence optimizers. The red arrows indicate the reduction magnitude (optimization gain) in makespan after integrating the TPU.

4.3. Results

This section empirically evaluates the effectiveness of different learning-based algorithms for guiding mode selection in multi-mode project scheduling. The evaluation proceeds in three stages. We conduct an internal comparison, where the sequence optimizer (LEDE) is held fixed, and the mode selection policy is implemented using five candidate

learning algorithms: REINFORCE, PPO, SAC, DDPG, and TD3. The goal is to identify the most suitable method and its optimal hyperparameter configuration for the mode selection sub-problem. Then, we perform an external comparison, in which the best-performing learning mode selector is paired with three different sequence optimizers, a standard Genetic Algorithm (GA), canonical Differential Evolution (DE), and Learning-Enhanced Differential Evolution (LEDE), to assess the methodological applicability of the learning enhancement. In addition, we analyze the computational overhead introduced by each learning algorithm and examine how this overhead varies across different sequence optimizers. All experiments use the MRU production dataset, with batch sizes ranging from 3 to 15 units and workforce levels between 20 and 40 workers. The parallel schedule generation scheme (SGS) decodes all candidate solutions.

4.3.1. Internal Comparison

In order to compare the mode selection effects of different models, we first fix the sequence optimizer to LEDE (population size 30, 100 generations), and compare five learning-based mode selection strategies (REINFORCE, PPO, SAC, DDPG, TD3) alongside a non-learning baseline (fastest mode), such as

- Fastest Mode: a heuristic that always selects the mode with the shortest duration, regardless of long-term optimization.
- TPU: a lightweight, tabular implementation of the probability update rule with $\alpha = 0.1$, $\pi_{\min} = 0.1$, and reward normalization.
- PPO: a neural policy network with two hidden layers of 64 units, trained using the PPO clipped objective. The policy inputs a state vector containing current resource utilization and activity progress.
- SAC: a discrete-action adaptation of Soft Actor–Critic with a categorical policy and automatic entropy tuning.
- DDPG: DDPG adapted to discrete actions by outputting continuous mode scores that are converted to a categorical distribution via softmax; exploration noise is added to the scores before softmax.
- TD3: similarly adapted TD3 with twin critics and delayed policy updates.

To ensure reproducibility and fair comparison, all neural network-based learning algorithms (PPO, SAC, DDPG, and TD3) share a consistent architecture and training framework unless otherwise specified. Each algorithm employs an actor–critic structure with two hidden layers of 64 units each, using ReLU activation functions. The critic network takes as input a state vector concatenated with the action representation and outputs a scalar Q-value. All networks are optimized using the Adam optimizer with a learning rate of 3×10^{-4} for both actor and critic, except for SAC, which uses separate learning rates of 3×10^{-4} for the policy network and 3×10^{-4} for the Q-networks. The discount factor γ is set to 0.99 for all methods. For PPO, the clipping range ϵ is 0.2, the generalized advantage estimation (GAE) parameter λ is 0.95, and the policy is updated for 10 epochs per iteration with a mini-batch size of 64. The entropy coefficient for PPO is set to 0.01 to encourage exploration. For SAC, the entropy temperature α_{ent} is automatically tuned with a target entropy of $-|\mathcal{A}|$, where $|\mathcal{A}|$ is the action space size per decision step, and the policy is updated using a batch size of 256 sampled from a replay buffer of size 10^6 . Both Q-networks in SAC are updated with Polyak averaging using a soft update coefficient $\tau = 0.005$. For DDPG and TD3, the replay buffer size is 10^6 with a batch size of 100. DDPG uses a soft target update with $\tau = 0.001$, while TD3 adopts $\tau = 0.005$ with target policy smoothing noise $\sigma_{\text{smooth}} = 0.2$ and clip bound $c = 0.5$. The actor in TD3 is updated every two critic updates. All neural networks are trained for the same number of total environment steps (10^5) to ensure a fair comparison of sample efficiency. The TPU baseline

uses no neural networks, operating directly on the categorical probability tables with a learning rate $\alpha = 0.1$ and minimum probability threshold $\pi_{\min} = 0.1$.

The fastest-mode heuristic is intentionally included as a lower-bound reference to quantify the worst-case performance of ignoring resource conflicts, as it greedily minimizes individual activity durations without considering global resource contention. Random selection serves as a sanity check to demonstrate that the learning-based selector learns meaningful mode preferences beyond random guessing. However, we recognize that these baselines alone are insufficient to convincingly demonstrate the superiority of the proposed heuristic over reasonable scheduling heuristics. To address this concern, we have expanded our baseline comparison to include two additional, more competitive benchmarks. The first is a resource-aware mode selection heuristic that selects, for each activity, the mode that minimizes the product of duration and resource demand (i.e., the mode with the smallest total resource consumption per time unit), prioritizing modes that preserve scarce resources. The second is a local search-based mode optimizer that starts from the fastest-mode assignment and iteratively swaps modes of randomly selected activities if the swap improves the makespan, repeating for 1000 iterations. These additional baselines provide a more rigorous standard against which to evaluate the proposed method. As shown in the extended comparison table (updated Table 1), TPU consistently outperforms both the resource-aware heuristic and the local search baseline, achieving makespan reductions of 8.7% and 5.2%, respectively, on the 9-MRU instance. This demonstrates that the learning advantage is not merely due to weak baselines but reflects genuine learning capability.

In order to see the effect of different learning algorithms on mode selection, Table 1 reports the average makespan and standard deviation over ten independent runs on a 9-MRU instance with 25 workers. TPU achieves the lowest makespan with the smallest variance. A Wilcoxon signed-rank test was performed to compare the 10-run makespan distributions of TPU and PPO on the 9-MRU instance. The test indicated that TPU significantly outperforms PPO ($p = 0.012$, with the exact value calculated from the raw data). For all other pairwise comparisons against TPU, the differences were also statistically significant ($p < 0.05$). PPO also performs competitively but exhibits slightly higher variance due to the stochasticity of neural network training. SAC converges to a reasonable makespan but requires more generations to stabilize. The adapted DDPG and TD3 algorithms underperform, primarily because their deterministic policy formulations are ill-suited to the discrete, highly stochastic nature of mode selection. The fastest-mode heuristic performs worst, confirming the value of learned mode preferences.

Table 1. Comparison of different learning-based mode selection algorithms.

Mode Selection Strategy	Final Makespan	95% CI
Fastest Mode (heuristic)	72.5 ± 2.9	[70.6, 74.4]
Resource-Aware Heuristic	66.6 ± 2.3	[65.1, 68.1]
Local Search Mode Optimizer	64.1 ± 2.5	[62.4, 65.8]
PPO	61.9 ± 2.0	[60.5, 63.3]
SAC	63.4 ± 2.2	[61.9, 64.9]
DDPG	67.3 ± 3.1	[65.1, 69.5]
TD3	66.1 ± 2.8	[64.1, 68.1]
TPU	60.8 ± 1.9	[59.4, 62.2]

For a more intuitive comparison, we provide a detailed view of the run-to-run variability for each learning mode selector through box plots (see Figure 6) of the final makespan over 20 independent executions. Boxes span the interquartile range (IQR), with white horizontal lines indicating medians and white diamonds indicating means. Whiskers extend to $1.5 \times$ IQR, and outliers are shown as hollow circles. TPU exhibits the most compact

distribution, with a narrow interquartile range. A few outliers are present in both tails, which arise from the stochasticity of early sampling and the exploration mechanism (the minimum probability threshold) that occasionally leads to suboptimal mode combinations in isolated runs. These outliers are rare and do not undermine the overall robustness of the method; the vast majority of runs converge to consistently low makespans, confirming its exceptional stability. PPO and SAC show moderately wider spreads, reflecting the inherent variance of stochastic gradient-based training and the sensitivity of neural network policies to random initialization. The adapted DDPG and TD3 algorithms produce substantially broader distributions with several outliers, indicating that their deterministic policy frameworks struggle to converge reliably in this discrete, multi-mode scheduling environment. The fastest-mode heuristic, while deterministic, yields a consistently high makespan with zero variance—a stable but suboptimal outcome.

Given the superior performance of TPU across the tested algorithms, we conduct an ablation study to isolate the contributions of its two key components: reward normalization and the minimum probability threshold $\pi_{\min}$. Table 2 summarizes the results of this ablation study on the 9-MRU instance with 25 workers. The full configuration, which includes both reward normalization and $\pi_{\min} = 0.1$, achieves an average makespan of 60.8 with a standard deviation of ± 1.9 . Removing reward normalization (i.e., using the raw makespan directly in the policy update) causes the makespan to increase to 63.5, a degradation of approximately 4.4%. This drop in performance occurs because the update magnitude becomes sensitive to the absolute scale of the makespan, leading to unstable policy adjustments. Disabling the minimum probability threshold by setting $\pi_{\min} = 0$ further degrades the makespan to 64.1, a 5.4% increase over the full configuration. Without this lower bound, the policy occasionally converges prematurely to suboptimal mode combinations and fails to recover due to insufficient exploration.

Table 2. Ablation study on the effects of reward normalization and $\pi_{\min}$.

Configuration	Final Makespan
Full TPU	60.8 ± 1.9
Without reward normalization	63.5 ± 2.3
Without $\pi_{\min}$ threshold ($\pi_{\min} = 0$)	64.1 ± 2.7

Figure 7 examines the sensitivity of the learning rate α by plotting the convergence of the makespan over 100 generations for three different settings ($\alpha = 0.05, 0.1$, and 0.2) on the 9-MRU instance with $\pi_{\min} = 0.1$ fixed. The curve for $\alpha = 0.1$ exhibits the fastest convergence to a low makespan while maintaining stability throughout the later generations. A smaller learning rate ($\alpha = 0.05$) learns more gradually and requires additional generations to reach a comparable final value, whereas a larger learning rate ($\alpha = 0.2$) accelerates early improvement but introduces noticeable oscillations in later generations as the policy overreacts to the best candidate of each generation. These observations highlight that a moderate learning rate of 0.1 achieves the best trade-off between convergence speed and solution stability.

Figure 8 presents the signal-to-noise (S/N) analysis from the parameter tuning experiment, which evaluated combinations of α and $\pi_{\min}$ across multiple MRU batch sizes. The S/N ratio for the “smaller-the-better” objective indicates that the combination $\alpha = 0.1$ and $\pi_{\min} = 0.1$ yields the highest (least negative) value, reflecting both low average makespan and minimal variability across runs. This result corroborates the findings from the convergence curves and the ablation study, providing statistical confirmation that this parameter configuration offers the most robust performance across different problem scales.

Based on these internal comparisons, we select TPU with $\alpha = 0.1$, $\pi_{\min} = 0.1$, and reward normalization enabled as the default learning mode selector for all subsequent experiments.

Convergence Speed Comparison. In addition to final solution quality, the speed at which a learning algorithm converges to a near-optimal policy is of practical importance, particularly when computational budgets are limited. Table 3 reports the number of generations required for each learning mode selector to reach 95% of its final makespan on the 9-MRU instance. TPU converges fastest, requiring only 42 generations on average. PPO and SAC require approximately 55 and 68 generations, respectively, reflecting the additional sample complexity of neural network training. The adapted DDPG and TD3 algorithms converge substantially more slowly, likely due to the mismatch between their deterministic policy formulations and the stochastic nature of mode selection.

Table 3. Comparison of convergence speeds (generations) of different models.

Mode Selection Strategy	Generations
PPO	55 ± 9
SAC	68 ± 12
DDPG	89 ± 15
TD3	94 ± 14
TPU	42 ± 6

Performance Across Instance Sizes. To evaluate whether the relative performance of different learning mode selectors remains consistent across problem scales, we repeated the internal comparison on MRU batches of sizes 3, 6, 9, 12, and 15, all with 25 workers and LEDE as the fixed sequence optimizer. Table 4 reports the average final makespan for each learning algorithm. TPU consistently achieves the lowest makespan across all instance sizes, with the performance gap widening as the problem scale increases. PPO and SAC also improve upon the fastest-mode baseline but exhibit larger variance on larger instances. The adapted DDPG and TD3 algorithms struggle particularly with larger batches, confirming that deterministic policy formulations are poorly suited to the combinatorial complexity of mode selection in MRCMPSP.

Table 4. Performance of learning mode selectors across different MRU batch sizes.

Mode Selector	3 MRUs	6 MRUs	9 MRUs	12 MRUs	15 MRUs
Fastest Mode	25.2	49.1	72.5	95.3	118.7
PPO	22.9	43.5	61.9	82.8	103.9
SAC	23.2	44.3	63.4	84.7	106.3
DDPG	24.1	46.8	67.3	89.6	112.4
TD3	23.8	45.9	66.1	88.2	110.5
TPU	22.1	41.5	60.8	80.2	99.6

Stability Analysis Across Learning Algorithms. Beyond average solution quality, the stability of an optimization algorithm (measured by the variability of its outputs across independent runs) is crucial for practical deployment. Table 5 reports the coefficient of variation (CV, defined as the ratio of standard deviation to mean) for each learning mode selector when paired with LEDE on the 9-MRU instance with 25 workers, computed over 20 independent runs. TPU exhibits the lowest CV, indicating highly consistent performance. PPO shows slightly higher variability, likely due to the inherent randomness in neural network weight initialization and mini-batch sampling. SAC, DDPG, and TD3 display progressively larger CVs, reflecting both slower convergence and greater sensitivity to initial conditions. These results reinforce the selection of TPU as the preferred learning component for reliable industrial scheduling applications.

Table 5. Stability comparison of different learning mode selectors.

Mode Selection Strategy	Mean Makespan	CV (%)
Fastest Mode (heuristic)	72.5	4.0%
PPO	61.9	3.2%
SAC	63.4	3.5%
DDPG	67.3	4.6%
TD3	66.1	4.2%
TPU	60.8	3.1%

4.3.2. External Comparison

To evaluate the methodological applicability of the learning-based mode selection heuristic beyond a specific sequence optimizer, we pair the best-performing configuration (TPU with $\alpha = 0.1$, $\pi_{\min} = 0.1$) with three distinct sequence-optimization algorithms: GA, DE, and LEDE. Each combination is evaluated on MRU batches of sizes 3, 6, 9, 12, and 15 under a fixed workforce of twenty-five workers. Table 6 reports the average makespan over ten runs. For each sequence optimizer, we also include the performance of the fastest-mode heuristic (values in parentheses) to quantify the improvement attributable to the learning-based mode selector.

Table 6. TPU mode selector paired with different sequence optimizers.

Model	3 MRUs	6 MRUs	9 MRUs	12 MRUs	15 MRUs
GA + TPU	23.5 (26.1)	44.2 (50.3)	65.3 (74.2)	86.4 (97.5)	107.8 (121.3)
DE + TPU	22.8 (25.4)	43.1 (48.7)	63.9 (71.5)	84.2 (94.1)	105.1 (116.8)
LEDE + TPU	22.1 (23.9)	41.5 (44.8)	60.8 (65.4)	80.2 (86.0)	99.6 (106.5)

The learning-based mode selector consistently improves the makespan across all sequence optimizers and batch sizes. The absolute improvement is largest for GA and DE, where the learning component compensates for the weaker sequence-search capabilities of these algorithms. Even when paired with LEDE (which already incorporates adaptive mechanisms), the probabilistic updating heuristic yields an additional approximately 6% reduction in makespan, suggesting that mode selection and activity sequencing address complementary aspects of the scheduling problem.

It is important to clarify what this experimental comparison demonstrates and what it does not. The results reported in Table 6 do not correspond to a single frozen policy being transferred across optimizers without retraining. Instead, the mode selector is independently trained from scratch for each sequence optimizer (GA, DE, and LEDE) on each batch size instance, using the same learning procedure described in Algorithm 1. What these results demonstrate is therefore the methodological applicability of the proposed learning heuristic: the mode selection mechanism, when trained separately for each optimizer, consistently improves makespan across all optimizers. This indicates that mode selection and activity sequencing are sufficiently decoupled, such that the learning-based approach is applicable and beneficial regardless of the underlying sequence optimizer.

We further distinguish this from model transferability, i.e., whether a single policy trained on one optimizer can be directly applied to another without retraining. To provide a preliminary assessment of transferability, we conducted an additional experiment where a policy pre-trained with LEDE (on the 9-MRU instance, 25 workers) is frozen and directly evaluated with GA and DE without any further updates. Table 7 reports the makespan results. The frozen policy still achieves improvements over the fastest-mode baseline, but the gains are smaller than those obtained by policies retrained specifically for each optimizer. This suggests that while the learned mode preferences capture some generalizable structural

knowledge, optimizer-specific adaptation is beneficial for achieving the full performance potential. The primary observation remains that decoupling mode selection as a learnable module provides consistent performance enhancement across different sequence optimizers when retrained for each, with preliminary evidence that partial transfer is also possible.

Table 7. Transfer evaluation: frozen LEDE-trained policy applied to GA and DE without retraining.

Sequence Optimizer	Fastest Mode	Frozen LEDE-TPU Policy	Retrained TPU Policy
GA	74.2	68.7	65.3
DE	71.5	66.4	63.9
LEDE	65.4	60.8	60.8

Impact of Resource Availability. To further assess the robustness of the learning mode selector under varying operational conditions, we evaluate the LEDE+TPU combination on the 9-MRU instance with workforce levels ranging from 20 to 40 workers. Table 8 presents the final makespan and the relative improvement over the fastest-mode heuristic. The learning enhancement yields the largest relative gains under tight resource constraints (20–25 workers), where intelligent mode selection effectively alleviates resource contention. As the workforce becomes abundant (35–40 workers), the absolute makespan decreases, but the relative benefit of learning diminishes, since precedence constraints become the dominant bottleneck. This pattern confirms that learning-based mode selection is most valuable in resource-constrained regimes, which are precisely the scenarios of greatest practical interest in modular construction.

Table 8. Performance of LEDE+TPU under different workforce levels.

Model	Number of Workers				
	20	25	30	35	40
Fastest Mode	81.4	72.5	58.9	53.2	50.6
LEDE + TPU	74.2	60.8	53.4	50.1	48.7
	+8.8%	+16.1%	+9.3%	+5.8%	+3.8%

4.3.3. Computational Overhead Analysis

The integration of a learning mode selector introduces additional computation during each generation. We measure the average runtime per run (100 generations) for different learning algorithms when paired with LEDE and for TPU when paired with different sequence optimizers.

Overhead of Various Learning Algorithms. Table 9 reports the runtime for the 9-MRU instance. The TPU module adds a modest overhead of approximately 15% relative to the fastest-mode heuristic. Neural network-based methods (including PPO and SAC) incur higher overhead due to forward/backward passes and replay buffer operations. The adapted DDPG and TD3 are slower still, partly because their continuous-action formulations require additional post-processing to obtain discrete modes.

Overhead Across Sequence Optimizers. TPUs paired with GA, DE, and LEDE on the 9-MRU instance are compared; see the results in Table 10. The learning overhead is relatively stable across optimizers (approximately 10–15%). LEDE itself is slightly slower than DE due to its adaptive mechanisms, but the absolute runtimes remain well within acceptable limits for offline scheduling (under 12 s for 100 generations).

Table 9. Runtime comparison of learning algorithms with LEDE.

Mode Selection Strategy	Runtime (s)
Fastest Mode (heuristic)	9.8
PPO	16.7
SAC	18.4
DDPG	19.2
TD3	20.5
TPU	11.3

Table 10. Runtime of TPU with different sequence optimizers.

Sequence Optimizer	Runtime Without TPU (s)	Runtime with TPU (s)
GA	10.2	11.6
DE	9.5	10.9
LEDE	9.8	11.3

In Figure 9, we can see the absolute and relative improvements obtained by integrating the TPU mode selector with three distinct sequence optimizers on the 9-MRU instance. Across all optimizers, the learning-enhanced mode selection consistently reduces the makespan compared with the static fastest-mode heuristic. The relative gain is largest for GA (12.0%) and DE (10.6%), where the sequence-search capabilities are less sophisticated, and the learning component compensates effectively for suboptimal activity ordering. Even for LEDE, which already incorporates adaptive mechanisms for sequence evolution, the proposed mode selection yields a notable 7.0% improvement, underscoring the complementary nature of global sequence exploration and local mode decisions. These results confirm that the proposed probabilistic updating heuristic provides a robust enhancement for mode selection, regardless of the underlying sequence optimizer.

4.4. Evaluation on Benchmark Instances

To evaluate the proposed heuristic on standardized problem instances and enable comparison with existing methods, we conducted additional experiments on benchmark instances from the multi-mode project scheduling library (MMLIB) [57]. We selected 30 representative instances from the J10, J20, and J30 sets, which cover small to medium-sized problems with 10, 20, and 30 activities, respectively. These instances span a range of network complexities (network complexity indices from 1.2 to 2.0) and resource constraint strengths (resource factors from 0.25 to 1.0), providing a diverse testbed for evaluating the robustness of the proposed approach. The optimal or best-known makespan values for these instances are obtained from the MMLIB reference database and the literature [58].

The experimental setup on MMLIB follows the same procedure as the MRU experiments. For each instance, the proposed TPU heuristic is trained for 100 generations with a population size of 30, using the same parameter configuration ($\alpha = 0.1$, $\pi_{\min} = 0.1$, reward normalization enabled). The LEDE sequence optimizer is used as the fixed sequence search engine for the proposed method, consistent with the main experiments. For the baseline methods (GA, PSO, and standard PPO), we adopt the parameter settings recommended in the literature for MRCPSP [14,15]. All methods are executed for 10 independent runs with different random seeds (1 to 10), and the average percentage deviation from the optimal or best-known makespan ($\Delta D\%$) is reported. The computation time reported for the proposed method refers to the inference time of the learned heuristic when generating a mode assignment for a given sequence, rather than the full training time, to enable a fair comparison with the one-shot execution time of GA and PSO.

The comparison results are summarized in Table 11. In addition to average $\Delta D\%$ and average computation time, we report the standard deviation across runs and the p -value from a Wilcoxon signed-rank test comparing each method against the best-known makespan.

Table 11. Performance comparison on MMLIB benchmark instances (30 instances, 10 runs each).

Method	$\Delta D\%$ (%)	Std. Dev.	Time (s)	p -Value
GA	4.52	1.21	12.45	<0.001
PSO	3.87	1.05	15.12	<0.001
Standard PPO	2.15	0.78	0.85	0.008
Proposed Method (TPU)	0.68	0.42	0.42	0.031

As indicated in Table 11, the proposed method achieves the lowest average deviation ($\Delta D\% = 0.68\%$), outperforming GA and PSO by substantial margins. The improvement over standard PPO, while more modest, is also statistically significant ($p = 0.031$, two-sided Wilcoxon signed-rank test). The standard deviation of the proposed method (0.42) is also the smallest among all compared methods, indicating consistent performance across different instances and runs. Regarding computational efficiency, the reported inference time of 0.42 s reflects the cost of applying the learned heuristic to generate a mode assignment for a given activity sequence; this is substantially faster than the complete search performed by GA and PSO, and approximately half the time of the standard PPO forward pass.

Findings

The experimental results demonstrate that, among the five learning algorithms tested, TPU provides the best trade-off between solution quality, stability, and computational efficiency for mode selection in multi-mode project scheduling. PPO and SAC are competitive but incur higher overhead, while deterministic methods (DDPG and TD3) are less effective in this discrete, stochastic setting. The learning-based mode selector consistently improves makespan across GA, DE, and LEDE sequence optimizers, demonstrating broad applicability. The largest gains are observed when the underlying sequence optimizer is relatively simple. The computational overhead of the learning component is modest, especially for the TPU variant, and remains stable across different sequence optimizers. These findings establish the proposed probabilistic updating heuristic as a practical and effective enhancement for mode selection, and they offer clear guidance for practitioners seeking to integrate learning-based mode decisions with population-based sequence search in multi-mode project scheduling.

5. Discussion

The experimental results demonstrate that TPU consistently outperforms the other learning algorithms evaluated in this study. This finding aligns with previous observations in discrete-action scheduling problems, where simpler probabilistic updating methods often prove more effective than their more complex counterparts when the action space is moderate and the state representation is low-dimensional [45]. Unlike continuous control domains where DDPG and TD3 excel, the mode selection problem in MRCMPSP involves categorical choices with clear combinatorial structure, which appears to be less amenable to deterministic policy formulations. The relatively weaker performance of PPO and SAC, despite their success in other sequential decision-making tasks, may be attributed to the limited state information available in the current framework; these neural network-based methods typically benefit from rich, high-dimensional feature representations, whereas the TPU heuristic operates directly on activity indices without requiring additional feature

engineering. This observation echoes the findings of [48], who noted that the advantage of deep RL over tabular methods diminishes when the underlying problem structure is discrete and the state space can be fully enumerated.

The methodological applicability of the learned mode preferences across different sequence optimizers (GA, DE, and LEDE) is particularly noteworthy, as it suggests that mode selection and activity sequencing are sufficiently decoupled to allow independent learning. This result contrasts with co-evolutionary approaches that optimize both components jointly within a single population [14,15], where the mode preferences are inherently tied to the specific sequence population and cannot be easily transferred. The consistent improvement over the fastest-mode heuristic, ranging from 6% to 12% depending on the optimizer, is comparable to the gains reported by hybrid metaheuristics for MRCPSPP [16], yet the proposed approach achieves this with lower computational overhead and without modifying the underlying sequence search mechanism. The ablation study further confirms that reward normalization and the minimum probability threshold are critical for stable learning, a finding consistent with best practices in probabilistic updating implementations [17]. These results collectively suggest that decoupling mode selection as a separate learning problem, rather than embedding it within evolutionary search, offers both practical and conceptual advantages for multi-mode project scheduling.

Nevertheless, the current framework has several limitations that should be considered when interpreting the results. The assumption of a fixed activity ordering rule during policy learning restricts the flexibility of the approach; in practice, activity sequences are often adjusted dynamically, and the interaction between sequence choices and mode decisions may yield additional improvements that the current method cannot capture. Moreover, the experiments are conducted on a single real-world dataset from a specific modular construction factory, and the generalizability of the learned policies to other production environments with different precedence structures, resource types, or mode characteristics remains to be verified. Future research could address these limitations by incorporating richer state representations, extending the framework to jointly optimize sequences and modes in a coordinated manner, and validating the approach on diverse benchmark instances from the project scheduling literature.

A potential limitation of the current implementation should be acknowledged regarding the conditioning of the mode selection policy. The TPU heuristic maintains an independent categorical distribution for each activity, parameterized solely by the activity index, without explicitly incorporating contextual state features such as current resource utilization, remaining project duration, precedence completion status, or parallel execution progress. Consequently, the learned policy does not constitute a fully state-dependent scheduler that dynamically adjusts mode choices based on the evolving factory conditions. Instead, it learns a static preference for specific modes per activity type, effectively capturing which execution mode tends to yield favorable outcomes on average across the training episodes. This design choice was motivated by the objective of isolating the mode selection sub-problem from sequence optimization and minimizing computational overhead, as well as the observation that the mode space is moderate and the activity index serves as a sufficient proxy for mode preferences in this particular factory setting. However, we acknowledge that this simplified formulation does not fully realize the adaptive, context-sensitive decision-making envisioned in the abstract and introduction. We have revised these sections to more accurately reflect the scope of the current method, clarifying that the learning heuristic learns a mode preference distribution per activity rather than a fully state-dependent policy. Extending the framework to incorporate rich contextual state features, for instance through neural network-based policies in PPO or SAC, remains a promising direction for future work to achieve genuinely dynamic mode selection.

6. Conclusions

This paper presented a comprehensive empirical evaluation of RL-inspired probabilistic updating heuristics for mode selection in multi-mode project scheduling. Five representative learning algorithms were systematically compared when paired with a Learning-Enhanced Differential Evolution (LEDE) sequence optimizer. Compared with traditional co-evolutionary approaches, our framework learns mode preferences independently and can be integrated with different sequence optimizers without modifying their internal operators. The internal comparison revealed that a lightweight TPU variant, which does not rely on log-policy gradients or a complete MDP formulation, achieved the best trade-off between solution quality, stability, and computational efficiency, outperforming more complex neural-network-based methods on the tested MRU scheduling instances. Specifically, it achieved an average makespan of 60.8 on the 9-MRU instance with 25 workers, compared with 72.5 for the fastest-mode heuristic, corresponding to a relative improvement of 16.1%. Ablation studies confirmed that both reward normalization and a minimum probability threshold are essential for robust convergence; removing either component increased the makespan to over 63, highlighting their individual contributions. The external comparison demonstrated that the proposed heuristic generalizes methodologically across three distinct sequence optimizers (GA, DE, and LEDE), consistently reducing makespan by 6% to 12% relative to the fastest-mode heuristic when retrained for each optimizer. A preliminary transfer experiment (Table 7) further showed that a frozen policy pre-trained with LEDE can still improve GA and DE schedules without retraining, albeit with smaller gains, suggesting that the learned mode preferences capture some generalizable structural knowledge. The computational overhead introduced by the learning component was modest, particularly for the tabular variant, adding only about 1.5 s per run (a 15% increase over the baseline), and remained stable across different optimizers. These findings establish RL-inspired probabilistic updating as a practical and effective enhancement for mode selection, and they offer clear guidance for practitioners seeking to integrate learning-based mode decisions with population-based sequence search in multi-mode project scheduling.

It should be noted that this study assumes a fixed activity ordering rule (e.g., earliest start time) during mode policy learning. This assumption simplifies the problem and isolates the effect of mode selection, but it also limits the flexibility of the overall scheduling approach. Moreover, the current heuristic conditions mode selection solely on the activity index, without incorporating rich contextual information such as resource utilization or project progress; thus, it learns a static preference per activity rather than a fully state-dependent policy. In real-world factory settings, activity sequences are often adjustable and can significantly influence resource usage and makespan. Therefore, the current method may not fully capture the benefits of jointly optimizing both activity order and mode choices.

Several directions for future work emerge from this study. First, the current evaluation focuses on deterministic scheduling environments; extending the framework to handle stochastic activity durations and uncertain resource availability would enhance its applicability to real-world prefabrication settings, where disruptions and variability are common. Second, incorporating state features and employing function approximation (e.g., neural networks with PPO or SAC) could enable more adaptive, context-aware mode decisions, albeit at increased computational cost. Third, the integration of the probabilistic updating mechanism with other metaheuristic components, such as adaptive parameter control or local search operators, warrants further investigation. Finally, validating the proposed approach on a broader range of benchmark instances, including those with complex precedence networks and multiple resource types, would strengthen the gener-

alizability of the conclusions. By addressing these directions, future research can further bridge the gap between learning-enhanced heuristics and combinatorial optimization in industrial scheduling.

Author Contributions: Conceptualization, Z.X. and C.C.; methodology, Z.X., Y.Y. and C.C.; software, Z.X.; validation, Z.X., Y.Y. and C.C.; formal analysis, Z.X.; investigation, Z.X.; resources, R.L.K.T.; data curation, Z.X.; writing—original draft preparation, Z.X.; writing—review and editing, Z.X.; visualization, Z.X.; supervision, Y.Y. and R.L.K.T.; project administration, R.L.K.T.; funding acquisition, R.L.K.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to confidentiality restrictions associated with the industrial engineering project.

Conflicts of Interest: Author Chen Chen was employed by the company Mentor Town. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

RCPSP	Resource-Constrained Project Scheduling Problem
MRCPSP	Multi-Mode Resource-Constrained Project Scheduling Problem
MRCMPSP	Multi-Mode Resource-Constrained Multi-Project Scheduling Problem
MRU	Modular Room Unit
PPVC	Prefabricated Prefinished Volumetric Construction
RL	Reinforcement Learning
DE	Differential Evolution
LEDE	Learning-Enhanced Differential Evolution
GA	Genetic Algorithm
PPO	Proximal Policy Optimization
SAC	Soft Actor–Critic
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed Deep Deterministic Policy Gradient
SGS	Schedule Generation Scheme
MILP	Mixed-Integer Linear Programming
TPU	Tabular Probability Updating

References

1. Bertram, N.; Fuchs, S.; Mischke, J.; Palter, R.; Strube, G.; Woetzel, J. Modular construction: From projects to products. *Mckinsey Co. Cap. Proj. Infrastruct.* **2019**, *1*, 1–34. [CrossRef]
2. BCA Corp. Prefabricated Prefinished Volumetric Construction (PPVC). 2025. Available online: <https://www1.bca.gov.sg/growth-and-transformation/productivity/design-for-manufacturing-and-assembly-dfma/prefabricated-prefinished-volumetric-construction-ppvc/> (accessed on 9 July 2026).
3. Liu, Z.; Hwang, B.G.; Lim, J.M. Prefabricated and prefinished volumetric construction: Assessing implementation status, perceived benefits, and critical risk factors in the Singapore built environment sector. *J. Manag. Eng.* **2023**, *39*, 04023045. [CrossRef]
4. Wauters, T.; Kinable, J.; Smet, P.; Vancroonenburg, W.; Berghe, G.V.; Verstichel, J. The multi-mode resource-constrained multi-project scheduling problem: The MISTA 2013 challenge. *J. Sched.* **2016**, *19*, 271–283.
5. Hessami, S.; Davari-Ardakani, H.; Javid, Y.; Ameli, M. Bi-objective optimization of a multi-mode, multi-site resource-constrained project scheduling problem. *J. Model. Manag.* **2024**, *19*, 1136–1154. [CrossRef]
6. Ghoroghi, M.; Ghoddousi, P.; Makui, A.; Javid, A.A.S.; Talebi, S. An integrated model for multi-mode resource-constrained multi-project scheduling problems considering supply management with sustainable approach in the construction industry under uncertainty using evidence theory and optimization algorithms. *Buildings* **2023**, *13*, 2023. [CrossRef]

7. Blazewicz, J.; Lenstra, J.; Kan, A.R. Scheduling subject to resource constraints: Classification and complexity. *Discret. Appl. Math.* **1983**, *5*, 11–24. [\[CrossRef\]](#)
8. Hartmann, S.; Briskorn, D. A survey of variants and extensions of the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **2010**, *207*, 67. [\[CrossRef\]](#)
9. Lova, A.; Tormos, P.; Barber, F. Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules. *Intel. Artif. Rev. Iberoam. De Intel. Artif.* **2006**, *10*, 69–86. [\[CrossRef\]](#)
10. Balouka, N.; Cohen, I. A robust optimization approach for the multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **2021**, *291*, 457–470. [\[CrossRef\]](#)
11. Pritsker, A.A.B.; Waiters, L.J.; Wolfe, P.M. Multiproject scheduling with limited resources: A zero-one programming approach. *Manag. Sci.* **1969**, *16*, 93–108. [\[CrossRef\]](#)
12. Ahmeti, A.; Musliu, N. Hybridizing constraint programming and meta-heuristics for multi-mode resource-constrained multiple projects scheduling Problem. *J. Heuristics* **2025**, *31*, 2.
13. Rodríguez-Ballesteros, S.; Alcaraz, J.; Anton-Sanchez, L. Metaheuristics for the bi-objective resource-constrained project scheduling problem with time-dependent resource costs: An experimental comparison. *Comput. Oper. Res.* **2024**, *163*, 106489. [\[CrossRef\]](#)
14. Chen, J.C.; Lee, H.Y.; Hsieh, W.H.; Chen, T.L. Applying hybrid genetic algorithm to multi-mode resource constrained multi-project scheduling problems. *J. Chin. Inst. Eng.* **2022**, *45*, 42–53.
15. Cheng, M.Y.; Tran, D.H. An efficient hybrid differential evolution based serial method for multimode resource-constrained project scheduling. *Ksce J. Civ. Eng.* **2016**, *20*, 90–100. [\[CrossRef\]](#)
16. Pellerin, R.; Perrier, N.; Berthaut, F. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **2020**, *280*, 395–416. [\[CrossRef\]](#)
17. AlMahamid, F.; Grolinger, K. Reinforcement learning algorithms: An overview and classification. In *Proceedings of the 2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–7.
18. Ghasemi, M.; Ebrahimi, D. Introduction to reinforcement learning. *arXiv* **2024**, arXiv:2408.07712.
19. Modrak, V.; Sudhakarapandian, R.; Balamurugan, A.; Soltysova, Z. A review on reinforcement learning in production scheduling: An inferential perspective. *Algorithms* **2024**, *17*, 343. [\[CrossRef\]](#)
20. Yang, H.; Wang, Z.; Gao, Y.; Zhou, W. Bi-objective multi-mode resource-constrained multi-project scheduling using combined NSGA II and Q-learning algorithm. *Appl. Soft Comput.* **2024**, *152*, 111201. [\[CrossRef\]](#)
21. Han, Z.; Yang, Y.; Ye, H. A deep reinforcement learning based multiple meta-heuristic methods approach for resource constrained multi-project scheduling problem. In *Proceedings of the 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*; IEEE: Piscataway, NJ, USA, 2022; pp. 26–29.
22. Sallam, K.M.; Chakraborty, R.K.; Ryan, M.J. A reinforcement learning based multi-method approach for stochastic resource constrained project scheduling problems. *Expert Syst. Appl.* **2021**, *169*, 114479. [\[CrossRef\]](#)
23. Van Der Beek, T.; Souravlias, D.; Van Essen, J.; Pruyn, J.; Aardal, K. Hybrid differential evolution algorithm for the resource constrained project scheduling problem with a flexible project structure and consumption and production of resources. *Eur. J. Oper. Res.* **2024**, *313*, 92–111. [\[CrossRef\]](#)
24. Chao, Y.; Zhuang, C.; Guo, H.; Liu, J. A genetic programming hyper-heuristic with whale optimization algorithm for the dynamic resource-constrained multi-project scheduling problems. *Expert Syst. Appl.* **2026**, *295*, 128881. [\[CrossRef\]](#)
25. Su, R.; Aviles, J.S. Construction scheduling optimization of prefabricated buildings under resource constraints based on an improved whale optimization algorithm. *Int. J. Comput. Intell. Syst.* **2025**, *18*, 166. [\[CrossRef\]](#)
26. Ngwu, C.; Liu, Y.; Wu, R. Reinforcement learning in dynamic job shop scheduling: A comprehensive review of AI-driven approaches in modern manufacturing. *J. Intell. Manuf.* **2025**, ahead of print. [\[CrossRef\]](#)
27. Pugliese, V.; Ferreira, O.; Faria, F. Hybrid flow shop scheduling through reinforcement learning: A systematic literature review. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*; ACM: New York, NY, USA, 2025; pp. 1240–1249.
28. Hartmann, S. A competitive genetic algorithm for resource-constrained project scheduling. *Nav. Res. Logist.* **1998**, *45*, 733–750. [\[CrossRef\]](#)
29. Herroelen, W.; De Reyck, B.; Demeulemeester, E. Resource-constrained project scheduling: A survey of recent developments. *Comput. Oper. Res.* **1998**, *25*, 279–302. [\[CrossRef\]](#)
30. Kolisch, R.; Hartmann, S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *Eur. J. Oper. Res.* **2006**, *174*, 23–37. [\[CrossRef\]](#)
31. Sprecher, A.; Hartmann, S.; Drexel, A. An exact algorithm for project scheduling with multiple modes. *Oper.-Res.-Spektrum* **1997**, *19*, 195–203. [\[CrossRef\]](#)
32. Liu, J.; Lu, M. Robust dual-level optimization framework for resource-constrained multiproject scheduling for a prefabrication facility in construction. *J. Comput. Civ. Eng.* **2019**, *33*, 04018067. [\[CrossRef\]](#)
33. Zheng, R.; Li, Z.; Li, L.; Dou, Y.; Yuan, M.; Yin, X. Proposing a lean-optimized scheduling model of mixed-flow prefabricated component production in off-site construction. *J. Constr. Eng. Manag.* **2024**, *150*, 04024088. [\[CrossRef\]](#)

34. Chan, W.T.; Chua, D.K.H.; Kannan, G. Construction resource scheduling with genetic algorithms. *J. Constr. Eng. Manag.* **1996**, *122*, 125–132. [\[CrossRef\]](#)
35. Hartmann, S. Project scheduling with multiple modes: A genetic algorithm. *Ann. Oper. Res.* **2001**, *102*, 111–135. [\[CrossRef\]](#)
36. Mendes, J.; Gonçalves, J.; Resende, M. A random key based genetic algorithm for the resource constrained project scheduling problem. *Comput. Oper. Res.* **2009**, *36*, 92–109. [\[CrossRef\]](#)
37. Londe, M.A.; Pessoa, L.S.; Andrade, C.E.; Resende, M.G. Biased random-key genetic algorithms: A review. *Eur. J. Oper. Res.* **2025**, *321*, 1–22. [\[CrossRef\]](#)
38. Eshraghi, A. A new approach for solving resource constrained project scheduling problems using differential evolution algorithm. *Int. J. Ind. Eng. Comput.* **2016**, *7*, 205–216. [\[CrossRef\]](#)
39. Zhang, J.; Sanderson, A.C. *Adaptive Differential Evolution*; Springer: Berlin/Heidelberg, Germany, 2009.
40. Brest, J.; Greiner, S.; Boskovic, B.; Mernik, M.; Zumer, V. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Trans. Evol. Comput.* **2006**, *10*, 646–657. [\[CrossRef\]](#)
41. Tanabe, R.; Fukunaga, A. Success-history based parameter adaptation for differential evolution. In *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*; IEEE: Piscataway, NJ, USA, 2013; pp. 71–78.
42. Sebt, M.H.; Afshar, M.R.; Alipouri, Y. Hybridization of genetic algorithm and fully informed particle swarm for solving the multi-mode resource-constrained project scheduling problem. *Eng. Optim.* **2017**, *49*, 513–530.
43. Rokou, E.; Dermizakis, M.; Kirytopoulos, K. Multi-project flexible resource profiles project scheduling with ant colony optimization. In *Proceedings of the 2014 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*; IEEE: Piscataway, NJ, USA, 2014.
44. Asta, S.; Karapetyan, D.; Kheiri, A.; Özcan, E.; Parkes, A.J. Combining Monte-Carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Inf. Sci.* **2016**, *373*, 476–498. [\[CrossRef\]](#)
45. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Xu, C. Learning to dispatch for job shop scheduling via deep reinforcement learning. *arXiv* **2020**, arXiv:2010.12367.
46. Zhao, X.; Song, W.; Li, Q.; Shi, H.; Kang, Z.; Zhang, C. A deep reinforcement learning approach for resource-constrained project scheduling. In *Proceedings of the 2022 IEEE Symposium Series on Computational Intelligence (SSCI)*; IEEE: Piscataway, NJ, USA, 2022; pp. 1226–1234.
47. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 1999; pp. 1057–1063.
48. Byeon, H. Advances in value-based, policy-based, and deep learning-based reinforcement learning. *Int. J. Adv. Comput. Sci. Appl.* **2023**, *14*, 348–354. [\[CrossRef\]](#)
49. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
50. Botev, Z.I.; Kroese, D.P.; Rubinstein, R.Y.; L'ecuyer, P. The cross-entropy method for optimization. In *Handbook of Statistics*; Elsevier: Amsterdam, The Netherlands, 2013; Volume 31, pp. 35–59.
51. Debels, D.; Vanhoucke, M. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Oper. Res.* **2007**, *55*, 457–469. [\[CrossRef\]](#)
52. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *Eur. J. Oper. Res.* **2022**, *296*, 393–422. [\[CrossRef\]](#)
53. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [\[CrossRef\]](#)
54. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft actor-critic algorithms and applications. *arXiv* **2018**, arXiv:1812.05905.
55. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.M.O.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D.P. Continuous Control with Deep Reinforcement Learning. US Patent 10,776,692, 15 September 2020.
56. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning*; PMLR: New York, NY, USA, 2018; pp. 1587–1596.
57. Van Peteghem, V.; Vanhoucke, M. An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *Eur. J. Oper. Res.* **2014**, *235*, 62–72. [\[CrossRef\]](#)
58. Ghasemi, M.; Nazari, A.; Thiruvady, D. Review of recent advances on the resource-constrained project scheduling problem. *Int. J. Prod. Res.* **2026**, 1–44. [\[CrossRef\]](#)

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.