When All Paths Lead to a Dead End: Deadlock-Depth-guided Monte Carlo Tree Search for Reentrant Blocking Hybrid Flow Shops

Guangqi Zhang[1] **Xuefeng Liu**[1] **Chuyuan Wei**[2] **Shiyu Li**[1]
Shaojie Tang[3] **Jing Yuan**[4] **Jianwei Niu**[1]*

[1]State Key Laboratory of Virtual Reality Technology and Systems,
School of Computer Science and Engineering, Beihang University, Beijing, China
[2]School of Intelligence Science and Technology,
Beijing University of Civil Engineering and Architecture, Beijing, China
[3]Department of Management Science and Systems, University at Buffalo,
Buffalo, New York, USA
[4]Department of Computer Science and Engineering, University of North Texas,
Denton, Texas, USA

Abstract

Monte Carlo Tree Search (MCTS) performance degrades significantly in Reentrant Blocking Hybrid Flow Shop (RBHFS) scheduling, which arises in modern automated scientific laboratories. The interaction of blocking and reentrant constraints causes rollout simulations to consistently end in a "dense deadlock." As a result, infeasible branches become indistinguishable, which undermines the exploitation capability of standard MCTS and effectively reduces it to uniform random exploration. However, we find that deadlocks are not equally uninformative. We observe that the average rollout deadlock depth—the number of steps executed before failure—across different action branches shows a strong positive correlation with true feasibility probability, making it a reliable proxy for the latent likelihood that a branch contains feasible solutions. Guided by this insight, we propose Deadlock-Depth Guided MCTS (D^2-MCTS). By integrating deadlock-depth statistics into the evaluation mechanism, D^2-MCTS is able to distinguish among otherwise indistinguishable infeasible branches. Empirical results demonstrate that D^2-MCTS significantly outperforms state-of-the-art baselines, demonstrating superior robustness and efficiency.

1 Introduction

MCTS is a highly efficient online search algorithm that constructs a search tree incrementally [2]. By executing rollout simulations to evaluate current actions and employing the Upper Confidence Bound for Trees (UCT) for selection, MCTS effectively balances exploration and exploitation during the decision process [24]. This efficient online search capability has led to its successful application in combinatorial optimization domains such as the Flow Shop Scheduling Problem (FSP), where the objective is to sequence jobs on a set of machines to minimize the total completion time [12].

However, the performance of MCTS degrades significantly in the Reentrant Blocking Hybrid Flow Shop (RBHFS) problem. RBHFS is a variant of FSP that arises in modern automated scientific laboratories [1] and semiconductor fabrication facilities [22]. RBHFS is characterized by two critical

*Corresponding author: niujianwei@buaa.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

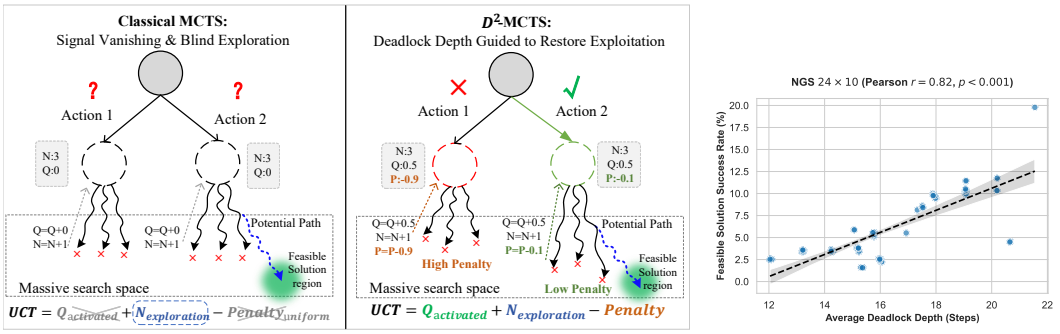


Figure 1: **Left:** Under dense deadlocks, classical MCTS assigns near-identical scores to all branches, degenerating into random exploration; D^2-MCTS restores exploitation by penalizing shallow-deadlock branches and rewarding deeper ones. **Right:** Average deadlock depth strongly correlates with empirical feasibility rate (NGS 24×10), confirming it as a reliable proxy for branch promise.

constraints: (1) reentrant operations, where jobs may revisit the same production stages multiple times [21], and (2) blocking constraint: due to limited or zero buffer capacity between stages, a job that has completed processing on a machine must remain on that machine until the next downstream machine becomes available [20]. For instance, in automated life science laboratories, sample plates repeatedly visit the same incubator or washer across assay cycles and must remain held in place until the next instrument is free; similarly, in high-tech manufacturing environments such as hard disk drive testing facilities, jobs revisit the same processing stages multiple times, and completed test items must remain in their processing slots until the material handling equipment becomes available for unloading [22, 30].

In RBHFS, improper scheduling can lead to circular waiting among machines: a job held on a downstream machine under the blocking constraint may prevent upstream jobs—which must revisit that same machine due to reentrant operations—from ever proceeding, resulting in a system-wide deadlock. Such deadlocks make feasible solutions in RBHFS significantly rarer within the vast search space compared to standard FSP [20], which in turn severely degrades MCTS performance. Since MCTS's finite rollout horizon explores only a small fraction of the space, rollout simulations in RBHFS terminate almost consistently in deadlock states—a phenomenon we term 'dense deadlock.' The prevalence of deadlocks renders infeasible branches indistinguishable from one another, causing standard MCTS to lose its ability to perform informed exploitation and degenerate into near-uniform random search.

However, we argue that not all deadlock states are equally uninformative. Among nodes whose simulations terminate in deadlock, some are more likely than others to contain feasible solutions deeper within their unexplored subtrees. The challenge, then, is to identify these "informative deadlock" nodes. We find that deadlock depth—the number of scheduling steps successfully executed before a deadlock occurs—serves as an effective discriminative signal. As validated by large-scale probe experiments (Figure 1 Right), the average deadlock depth across different action branches shows a strong positive correlation with their true feasibility probability. This indicates that deeper deadlocks reliably signal branches that are more likely to contain feasible solutions.

Leveraging this insight, we propose Deadlock-Depth Guided MCTS (D^2-MCTS), which integrates deadlock depth statistics into MCTS's evaluation and backpropagation mechanisms to provide meaningful differentiation among infeasible branches. By guiding the search toward nodes with greater deadlock depth, D^2-MCTS restores the algorithm's informed exploration capability, steering the search through dense deadlock regions to ultimately reach feasible solutions. Empirical results support our insight, showing that D^2-MCTS achieves a strong balance between solution quality and computational efficiency: it is significantly faster than meta-heuristic algorithms and exhibits much greater stability than reinforcement learning methods. Notably, even for large-scale tasks, the algorithm maintains a search time of approximately 100 ms per step (Figure 3(b)), enabling it to effectively support the dynamic scheduling needs of real-world automated scientific laboratories. We summarize our contributions as follows:

- We show that MCTS fails in RBHFS because it cannot distinguish among search branches whose rollouts consistently terminate in deadlock. Under the standard evaluation scheme,

classical MCTS assigns identical scores to all deadlock states, causing the search to degenerate into unguided random exploration.

- We reveal that deadlock depth serves as an effective indicator of latent feasibility within the search space. By differentiating promising nodes based on their deadlock depth statistics, we enable the restoration of the algorithm's exploitation capability, allowing it to effectively guide the search toward feasible regions.
- Based on this insight, we propose Deadlock-Depth Guided MCTS (D^2-MCTS). Empirical results demonstrate that our method achieves a superior balance between solution quality and computational efficiency compared to state-of-the-art baselines.

2 Related Work

FSP under Reentrant and Blocking Constraints. RBHFS compounds standard Hybrid Flow Shop (HFS) scheduling [13] with two co-occurring structural constraints: *reentrant job flows* and *zero-buffer blocking*. While each has been studied in isolation—blocking FSP [20] and reentrant flows via meta-heuristics [16] and hybrid RL approaches [25, 19]—their *simultaneous* imposition introduces system-wide deadlocks that neither constraint alone produces [14].

Research on this joint setting is remarkably scarce: Sangsawang et al. [22] directly addresses it, via a PSO meta-heuristic with a repair mechanism. However, meta-heuristics are fundamentally ill-suited for dynamic scheduling: in the sparsely feasible search space induced by joint deadlock pressure, stochastic perturbations frequently violate constraints, forcing repeated feasibility checks and solution repairs [6]. This computational overhead renders them too slow for the rapid, step-by-step responses that automated laboratory scheduling demands.

Learning-based Methods for Shop Scheduling. Given the absence of learning-based methods that directly address RBHFS, we review the broader landscape of deep learning-based methods for FSP and Job Shop Scheduling Problem (JSP)—the closest family of approaches from which adaptation might be attempted. Deep learning-based constructive methods—including supervised learning [13], reinforcement learning [27, 9, 8], and self-supervised approaches [5, 4]—have achieved strong performance on standard FSP and JSP by training a policy network to generate solutions step-by-step [18, 11].

However, these methods cannot be directly applied to RBHFS for two reasons. First, these methods lack explicit mechanisms for handling deadlocks. They rely on valid rollout trajectories during training (e.g., via PPO or self-supervised learning); however, in RBHFS, rollouts almost consistently terminate in deadlocks, failing to provide effective reward signals for policy optimization. Second, offline-trained networks cannot guarantee stable and effective behavior on unseen test data. The failure mode is categorically different: in standard JSP/FSP, a generalization error degrades solution quality; in RBHFS, a single erroneous step triggers an *irreversible* deadlock—an empty action set from which no recovery is possible. Such failures are unacceptable in unmanned automated laboratories where no human operator can intervene. This asymmetry necessitates a shift from offline-trained neural policies to simulation-based online search that can *guarantee* feasibility.

MCTS for Scheduling Problems. Monte Carlo Tree Search (MCTS) is an online search technique that balances exploration and exploitation, and it has been widely applied to various scheduling problems[15, 17]. By modeling the problem as a Markov Decision Process (MDP), it effectively handles the dynamic scheduling requirements of standard JSP [12, 23]. However, the application of MCTS to RBHFS remains underexplored, as existing constructive methods struggle to overcome the high risk of deadlocks in RBHFS. Existing literature characterizes deadlocks in scheduling as having a delayed characteristic, where system deadlock is not immediate but the consequence of suboptimal decisions accumulated steps earlier [7]. The rollout simulations in MCTS possess the necessary foresight capability[24], offering the potential to anticipate and avoid these future deadlock scenarios. However, the application of MCTS to deadlock-prone environments like RBHFS remains underexplored. Distinguished from previous works, we leverage this foresight capability to address the deadlock challenge in RBHFS. Besides, we identify the critical issue of rollout failure under dense deadlock conditions and propose a mechanism to successfully resolve it.

3 Preliminaries

3.1 Problem Formulation: RBHFS

Consider a manufacturing system consisting of a set of J jobs $\mathcal{J} = \{1, \ldots, J\}$ and a set of K processing stages $\mathcal{K} = \{1, \ldots, K\}$. Each stage $k \in \mathcal{K}$ contains m_k identical parallel machines. Distinct from the classical HFS, the RBHFS model incorporates reentrant process flows and zero-buffer constraints: **Reentrant Job Flows.** Each job $j \in \mathcal{J}$ requires a sequence of L_j operations, denoted as $O_j = (o_{j,1}, o_{j,2}, \ldots, o_{j,L_j})$. The l-th operation $o_{j,l}$ is processed at a specific stage $S_{j,l} \in \mathcal{K}$ for a deterministic duration $p_{j,l}$. The sequence of stages $(S_{j,1}, \ldots, S_{j,L_j})$ is not required to be acyclic; a job may return to a previously visited stage for subsequent operations [3]. **Blocking Constraints.** The system operates under a strict blocking-after-service mechanism due to the absence of intermediate buffers. Upon completing operation $o_{j,l}$ on a machine M, job j must remain on M until a machine at the next required stage $S_{j,l+1}$ becomes available. During this waiting period, machine M is considered *blocked* and cannot process any other jobs [20]. **Objective.** The objective is to minimize the makespan $C_{\max} = \max_{j \in \mathcal{J}} C_j$, where C_j is the completion time of job j.

3.2 Monte Carlo Tree Search (MCTS)

MCTS [24] constructs a search tree incrementally, where each node represents a specific state s and each edge corresponds to a scheduling action a. The algorithm iterates through four phases: Selection, Expansion, Simulation, and Backpropagation. In the *Selection* phase, the algorithm traverses the tree from the root to a leaf node by recursively choosing the child node that maximizes the Upper Confidence Bound for Trees (UCT) value:$\text{UCT}(s,a) = \frac{Q(s,a)}{N(s,a)} + c\sqrt{\frac{2\ln N(s)}{N(s,a)}}$, where $Q(s,a)$ represents the estimated value (negative average makespan) of taking action a in state s, $N(s)$ denotes the visit count of the current node, $N(s,a)$ denotes the visit count of the specific action edge, and c is a constant governing the exploration rate. Upon reaching a leaf node, a *Simulation* is performed using random rollout until a terminal state is reached. The simulation result is then *Backpropagated* up the tree to update the statistics $Q(s,a)$ and $N(s,a)$. Standard MCTS relies on the premise that random simulations generate informative estimates of solution quality. However, as discussed in Sec. 4.2, this mechanism becomes ineffective in RBHFS because standard simulations predominantly terminate in deadlock states.

4 Methodology

4.1 Markov Decision Process Formulation

The RBHFS scheduling can be modeled as a deterministic MDP tuple $(\mathcal{S}, \mathcal{A}, T, R)$. A state $S_t = (\mathcal{M}_t, \mathcal{J}_t)$ represents the occupation status of all machines and the current processing stage of all jobs. The action space $\mathcal{A}(S_t)$ comprises all valid job-to-machine assignments at step t that satisfy immediate resource constraints, particularly the blocking conditions. The transition function T deterministically updates the system to state S_{t+1} based on the chosen assignment a_t, modifying machine availability and job progress accordingly. A state is terminal if either: (1) all jobs are completed, denoted as a feasible state $s \in \mathcal{S}_{feas}$, or (2) no valid actions exist despite unfinished jobs, denoted as a deadlock state $s \in \mathcal{S}_{dead}$. The objective is to find a trajectory leading to a feasible terminal state that minimizes the makespan $C_{\max}$.

4.2 Analysis: Failure of Standard MCTS under Dense Deadlock Conditions

Standard MCTS employs the UCT criterion to guide the action selection process during the search. In RBHFS, the feasible solution space is sparse; consequently, random rollout simulations almost consistently terminate in deadlock states $\mathcal{S}_{dead}$. In standard scheduling problems, the reward is typically defined as the negative makespan ($-C_{\max}$) to favor shorter schedules. However, deadlocks present a challenge for reward formulation: utilizing the failure time (i.e., the total processing time accumulated before the system becomes blocked) as a reward is inappropriate, because a deadlock that occurs early in the schedule may yield a reward that is numerically higher (less negative) than the reward of a feasible schedule with a larger makespan. To address this issue, a common approach is to

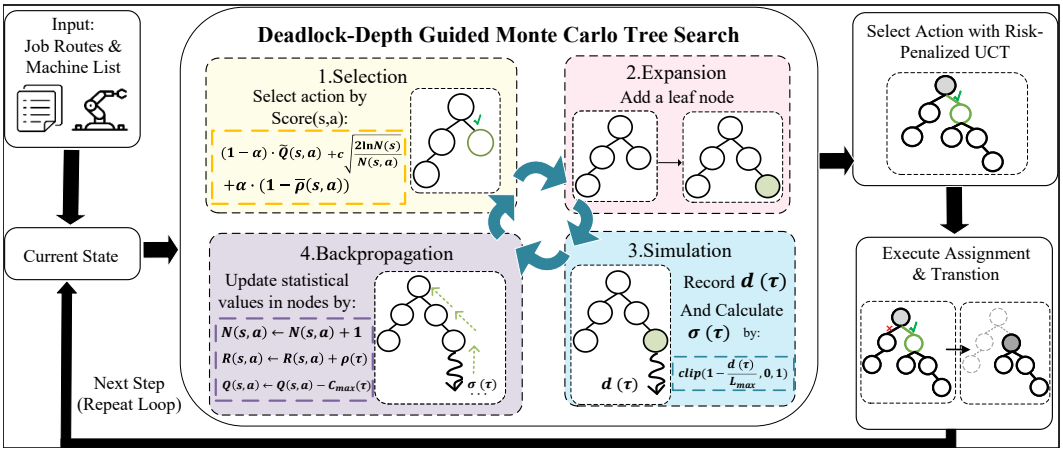


Figure 2: Overview of the D^2-MCTS method.

assign a fixed large penalty P to deadlock states, where $P \ll -C_{\max}$, ensuring that such trajectories are consistently discouraged.

However, due to the high density of deadlocks in RBHFS, the estimated value $Q(s,a)$ for all available actions tends to converge to this uniform penalty: $\lim_{N\to\infty} Q(s,a) = P, \quad \forall a \in \mathcal{A}(s)$. As the exploitation term becomes constant across all actions, the UCT selection rule reduces to a policy that depends solely on the exploration term, prioritizing actions with lower visit counts: $\arg\max_a \text{UCT}(s,a) \approx \arg\min_a N(s,a)$. Consequently, the algorithm loses its capability to exploit promising regions and performs a uniform search, expanding all branches equally. To address this limitation, it is necessary to extract informative feedback directly from the deadlock states to guide the search.

4.3 D^2-MCTS: Restoring Search Guidance

MCTS explores only a limited fraction of the solution space. A key challenge, therefore, is to distinguish which nodes—among those whose simulations terminate in deadlock—are more likely to lead to feasible solutions if explored further. In the absence of explicit feasibility feedback, we introduce a mechanism to differentiate deadlock states based on their depth. The overview of D^2-MCTS is shown in Figure 2.

4.3.1 Key Insight: Depth as a Feasibility Indicator

We observe that not all deadlocks are equally uninformative. We define deadlock depth $d(\tau)$ as the number of successfully executed steps in a simulation trajectory τ before it terminates in a deadlock state s_{dead}. This quantity serves as a useful indicator of latent potential: trajectories that sustain more steps before deadlocking are likely closer to the feasible region, even if no feasible solution has yet been reached. We therefore hypothesize that prioritizing directions that enable deeper rollouts increases the likelihood of guiding the search toward valid solutions. By differentiating deadlocks based on their depth, we obtain a meaningful guidance signal that promotes progress toward feasibility, even in the absence of explicit feasible outcomes. For action a_i with average deadlock depth $\bar{d}_i$ and feasibility probability p_i, we observe the monotone relationship: $\bar{d}_{i_1} > \bar{d}_{i_2} \quad \Rightarrow \quad p_{i_1} > p_{i_2}$, This hypothesis is empirically supported by the results shown in Figure 1 (right), where probe experiments with up to 5×10^5 rollouts reveal a strong positive correlation between the average deadlock depth and the true probability of reaching a feasible solution across multiple instances and decision steps. Additional details and complete probe results are provided in Appendix C.2.

4.3.2 Deadlock Risk Metric

We formalize this insight by introducing a *Deadlock Risk* metric, denoted as ρ. Let $d(\tau)$ represent the number of operations completed in a trajectory τ before termination, and let $L_{\max}$ denote the maximum possible number of operations in a fully feasible trajectory. We define the normalized

Deadlock Risk $\rho(\tau) \in [0, 1]$ as follows:

$$\rho(\tau) = \text{clip}\left(1 - \frac{d(\tau)}{L_{\max}}, 0, 1\right). \tag{1}$$

Based on this definition, shallow deadlocks result in a high risk value ($\rho \approx 1$), while deep deadlocks correspond to a low risk value ($\rho \to 0$).

4.3.3 Risk-Aware Tree Search

We extend the MCTS node structure to integrate this metric. While standard MCTS edges (s, a) store only the visit count N and cumulative reward Q, we introduce a third statistic: the *cumulative risk* R. The key design principle is to integrate feasibility guidance with makespan optimization in the selection process: the depth-based risk metric ρ provides a useful signal for navigating deadlock-dense regions, while Q tracks solution quality.

Augmented Backpropagation: In the backpropagation phase, the node statistics are updated based on the outcome of the simulation trajectory τ. We distinguish between two scenarios:

Scenario 1: Feasible Outcome. If τ successfully reaches a feasible terminal state with a makespan $C_{\max}(\tau)$, we update the cumulative quality Q using the negative makespan as reward, and the cumulative risk R remains unchanged:

$$Q(s, a) \leftarrow Q(s, a) - C_{\max}(\tau), \quad N(s, a) \leftarrow N(s, a) + 1. \tag{2}$$

Scenario 2: Deadlock Outcome. If τ terminates in a deadlock state s_{dead} with risk $\rho(\tau)$, we update the cumulative risk R using $\rho(\tau)$. For $Q(s, a)$, early termination of rollouts due to deadlocks makes the true makespan $C_{\max}(\tau)$ (Sec 4.2) unobservable; consequently, $Q(s, a)$ relies on lower-bound estimates rather than exact outcomes. Specifically, we update $Q(s, a)$ with an optimistic makespan estimate derived from a critical-path lower bound $\text{LB}(s_{\text{dead}})$ to account for makespan optimization in this case:

$$Q(s, a) \leftarrow Q(s, a) - \text{LB}(s_{\text{dead}}), \quad N(s, a) \leftarrow N(s, a) + 1, \quad R(s, a) \leftarrow R(s, a) + \rho(\tau), \tag{3}$$

where $\text{LB}(s) = \max\big(\max_m C_m(s),\ \max_j \big(E_j(s) + \sum_{k=\text{op}_j}^{L_j} p_{j,k}\big)\big)$ is an optimistic estimate of the minimum makespan from the current state, ignoring blocking constraints, where $C_m(s)$ denotes the completion time of machine m at state s; $E_j(s)$ denotes the end time of the last completed operation of job j; $\text{op}_j(s)$ denotes the index of the next operation to schedule for job j; L_j denotes the total number of operations for job j; and $p_{j,k}$ denotes the processing time of operation k of job j.

Risk-Penalized Selection Rule: To guide search through deadlock-dense regions, we propose a risk-penalized variant of the UCT criterion that explicitly prioritizes branches sustaining deeper rollouts before deadlocking. To ensure that the quality signal and the depth guidance signal are comparable across different problem scales, we normalize both to $[0, 1]$ and combine them via a convex weight $\alpha \in [0, 1]$. Let $\tilde{Q}(s, a) = \frac{\bar{Q}(s,a) - Q_{\min}}{Q_{\max} - Q_{\min}}$ denote the min-max normalized Q-value among all children of node s, where $\bar{Q}(s, a) = \frac{Q(s,a)}{N(s,a)}$ denote the average Q-value, $Q_{\max}$ and $Q_{\min}$ denote the maximum and minimum average Q-values among all actions of state s, respectively. When $Q_{\max} - Q_{\min} \leq \epsilon$, we set $\tilde{Q}(s, a) = 0.5$ for all child actions, where $\epsilon = 10^{-12}$. In this case, the quality term is neutral, and action selection is determined by the depth-guidance and exploration terms. The modified selection rule is:

$$\text{Score}(s, a) = (1 - \alpha) \cdot \underbrace{\tilde{Q}(s, a)}_{\text{Quality}} + \alpha \cdot \underbrace{\big(1 - \bar{\rho}(s, a)\big)}_{\text{Depth Guidance}} + c\sqrt{\frac{2 \ln N(s)}{N(s, a)}}, \tag{4}$$

where $\bar{\rho}(s, a) = R(s, a)/N(s, a)$ is the average deadlock risk, α controls the trade-off between depth-based guidance and makespan optimization ($\alpha = 0$: pure makespan optimization; $\alpha = 1$: pure deadlock avoidance). The normalization ensures that α has consistent semantic meaning across problem instances with varying scales and deadlock characteristics. The key insight driving this design is that the depth signal $(1 - \bar{\rho}(s, a))$ provides critical guidance in deadlock-rich phases, where Q-values are dominated by lower-bound estimates and thus lack discrimination, thereby guiding D^2-MCTS to navigate out of the dense deadlock regions in RBHFS. We provide the detailed algorithmic procedures for the D^2-MCTS framework in Algorithm 2.

4.4 Theoretical Properties

We first state the conditional asymptotic optimality of D^2-MCTS. To ensure that the depth-based guidance effectively counteracts the misleading optimistic estimates of deadlocks, the trade-off weight α must be sufficiently large. The rigorous proof via modified reward analysis is detailed in Appendix B.

Assumption 4.1 (Alpha Dominance Condition)**.** Let $\tilde{Q}_{\text{feas}}$ be the normalized Q-value of a feasible path, and $\tilde{Q}_{\text{dead}}$ be the optimistic Q-value of a deadlock state with risk ρ_{dead}. We assume $0 \leq \alpha < 1$ and that α is sufficiently large such that the depth reward outweighs the maximum possible Q-value deception: $\alpha \cdot \rho_{\text{dead}} > (1-\alpha)(\tilde{Q}_{\text{dead}} - \tilde{Q}_{\text{feas}})$.

Proposition 4.1 (Conditional Asymptotic Depth-Guided Search under Alpha Dominance)**.** *Consider a finite, deterministic MDP for RBHFS. Under the Alpha Dominance Condition, as the search budget $N \to \infty$, D^2-MCTS safely isolates the search space. Specifically, for any action a leading exclusively to deadlocks, its visit ratio $\frac{N(a)}{N} \to 0$. For the optimal feasible action $a^* = \arg\max_{a \in \mathcal{A}_{feas}} Q(a)$, the proposed selection rule asymptotically converges to $\lim_{N\to\infty} \frac{N(a^*)}{N} = 1$.*

5 Experiment

5.1 Experimental Setup

Datasets. We evaluate our method on two benchmark suites: (1) **Synthetic RHFS Benchmarks**: We constructed 100 instances modeling two representative laboratory workflows (ELISA, NGS). Distinct from random graphs, these instances are designed to capture the structural hardness of real-world automation: the intersection of cyclic re-entrancy and zero-buffer blocking. Crucially, to prevent problem trivialization at larger scales ($J \in \{6, \ldots, 96\}$), we employ a differential resource scaling policy that maintains high contention on bottleneck stages (details in Appendix C.1). (2) **HDD Dataset**: To test generalization, we include the hard disk device (HDD) manufacturing benchmark HDD[1] [3] for re-entrant flow shops, comprising 15 instances across two scales, and HDD[2] comprising 12 additional reentrant-blocking instances published by Sangsawang et al. [22].

Baselines. We compare D^2-MCTS against four categories of methods: (1) PDRs: Standard priority rules (SPT, LPT, MWKR, SRPT). (2) Constraint Programming (CP): We employ the OR-Tools CP-SAT solver with a 1-hour time limit to provide high-quality bounds or optimal solutions. (3) Metaheuristics: We select HPSO [22] as the representative metaheuristic because it handles concurrent re-entrant and blocking constraints. (4) Learning-based: We employ L2D [26] as the standard GNN-DRL baseline. Additionally, to ensure a fair comparison, we developed $L2D^*$, an enhanced variant that incorporates our deadlock penalty into the reward function to discourage the agent from visiting infeasible states. For HPSO and L2D/L2D*, we strictly adhere to the hyperparameter settings and network architectures described in their original papers.

Metrics, Hyper-parameter and Hardware. We report Makespan ($C_{\max}$) as the primary objective and Computational Time (seconds). For test instances where a constructive solver terminates in a deadlock state, the result is denoted as "D". For search time, we set 50 ms for 6, 12, 24 jobs in ELISA and NGS, 100 ms for 48 jobs, and 200 ms per step for 96 jobs. For α, we set 0.7 for all instances. All experiments were conducted on an Intel i9-13900K CPU and NVIDIA RTX 4090 GPU.

5.2 Main Results Analysis

Table 1 and Table 2 report results across the ELISA, NGS, and HDD benchmarks. Most PDRs deadlock as problem scale increases slightly. SRPT finds feasible solutions for all instances, but its greedy strategy yields suboptimal makespan. NEH and HPSO consistently find feasible solutions; however, their iterative whole-problem solving makes computation time prohibitive at large scales, rendering them unsuitable for dynamic laboratory scheduling. MCTS fails even at the 12×7 scale, directly confirming that rollouts yield no effective signal under dense deadlocks, causing the algorithm to lose exploitation capability entirely. L2D fails in RBHFS due to deadlocks during solution construction; our improved L2D* substantially recovers performance and scales to 48×7, but remains unstable and fails at 96×7. This demonstrates that deadlock depth is not merely an

Table 1: Results across scales on ELISA and NGS. **Obj.** is the average makespan (lower is better). D indicates deadlock/infeasible.

Type	Size	Metric	Baselines									Proposed	CP
			SPT	LPT	MWKR	SRPT	NEH	HPSO	MCTS	L2D	L2D*	D^2-MCTS	
ELISA	6×7	**Obj.**↓	82.9	102.7	65.0	136.2	136.6	135.3	65.8	61.0	117.6	**60.8**	58.8
		Time↓	0.008	0.008	0.009	0.008	0.115	4.50	2.114	0.075	0.078	2.14	
	12×7	**Obj.**↓	137.0	188.0	D	265.6	265.8	263.9	D	D	250.1	**127.5**	107.0
		Time↓	0.026	0.026		0.029	1.228	68.83			0.164	4.32	
	24×7	**Obj.**↓	D	D	D	240.7	221.0	187.1	D	D	269.2	**155.9**	107.5
		Time↓				0.107	18.641	234.81			0.367	8.75	
	48×7	**Obj.**↓	D	D	D	459.6	445.6	394.8	D	D	481.6	**319.4**	202.0
		Time↓				0.360	276.53	599.16			0.975	35.32	
	96×7	**Obj.**↓	D	D	D	896.5	862.4	786.5	D	D	D	**646.7**	397.1
		Time↓				1.058	5913.18	5700.44				137.04	
NGS	6×10	**Obj.**↓	153.2	205.7	110.7	289.8	291.0	288.5	111.6	**108.2**	232.6	**108.2**	105.0
		Time↓	0.018	0.019	0.020	0.017	0.19	20.575	3.051	0.10	0.12	3.06	
	12×10	**Obj.**↓	275.1	372.4	**199.4**	568.0	569.4	566.0	202.4	200.2	455.1	202.4	196.1
		Time↓	0.05	0.05	0.05	0.05	2.45	114.55	6.12	0.22	0.29	6.49	
	24×10	**Obj.**↓	D	D	D	533.7	515.8	469.6	D	D	613.6	**297.7**	196.6
		Time↓				0.19	31.88	349.28			0.70	13.19	
	48×10	**Obj.**↓	D	D	D	1046.8	1004.0	949.5	D	D	1032.3	**644.1**	375.0
		Time↓				0.606	472.55	1243.03			2.102	51.17	
	96×10	**Obj.**↓	D	D	D	2036.9	2023.1	1922.4	D	D	D	**1421.9**	739.0
		Time↓				2.191	10223.63	3032.60				195.81	

Table 2: Results across scales on HDD benchmarks. **Obj.** is the average makespan (lower is better). D indicates deadlock/infeasible.

Type	Size	Metric	Baselines									Proposed	CP
			SPT	LPT	MWKR	SRPT	NEH	HPSO	MCTS	L2D	L2D*	D^2-MCTS	
HDD^1	7×17	**Obj.**↓	D	D	D	488.0	500.6	**486.0**	D	D	D	502.6	486.0
		Time↓				0.098	5.00	123.99				6.48	
	11×17	**Obj.**↓	D	D	D	704.0	716.2	**702.0**	D	D	D	704.0	702.0
		Time↓				0.099	5.00	123.99				10.20	
HDD^2	10×6	**Obj.**↓	D	D	D	215.0	66.7	**60.0**	D	D	215.0	64.0	60.0
		Time↓				0.007	0.02	48.63			0.038	1.5	
	10×12	**Obj.**↓	60.0	60.0	60.0	215.0	66.7	**60.0**	60.9	60.0	215.0	**60.0**	60.0
		Time↓	0.008	0.008	0.008	0.008	0.03	72.311	1.54	0.089	0.092	1.54	
	20×6	**Obj.**↓	D	D	D	436.7	127.7	**120.0**	D	D	436.7	137.8	120.0
		Time↓				0.019	0.16	171.19			0.099	3.08	
	20×12	**Obj.**↓	D	D	D	436.7	127.3	**120.0**	D	D	436.7	125.4	120.0
		Time↓				0.023	0.25	189.03			0.101	3.10	

MCTS-specific trick but a general-purpose indicator that can rescue learning-based methods from the dense deadlock trap as well. Directly coupling deadlock depth into an RL reward function requires careful penalty calibration to balance deadlock avoidance against makespan optimization—a difficulty that limits L2D*'s scalability. MCTS, L2D, and L2D* all fail completely on HDD^1—a structurally harder variant where each RBHFS stage contains multiple machines (versus one in standard RHFS), intensifying resource competition and deadlock frequency. On HDD^1, D^2-MCTS shows no clear makespan advantage over SRPT or HPSO. In contrast, on HDD^2, it reduces the makespan by 70.51% on average relative to SRPT, achieves an average relative gap of 6.50% to HPSO/CP, and uses only 2.16% of HPSO's computation time on average across the four problem groups.

D^2-MCTS succeeds on all instances, consistently finding stable, near-optimal solutions at 50 ms per step. Figure 3 visualizes makespan quality and computational efficiency from Table 1, where the vertical axis shows the CP-normalized gap weighted by success rate, and the horizontal axes represent per-instance total time (a) and per-step time (b). D^2-MCTS is over $10\times$ faster than NEH and HPSO on large-scale instances, and substantially more stable than RL methods whose offline-trained policies fail during online construction. The 50 ms per-step runtime allows scheduling decisions to be interleaved with job execution, making D^2-MCTS suitable

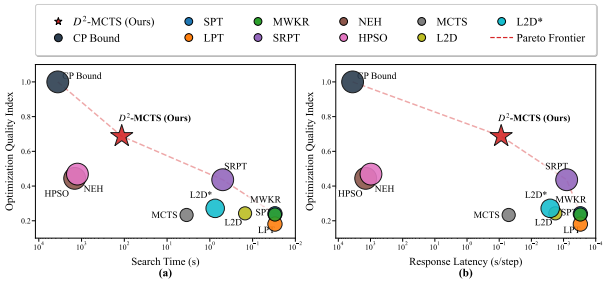


Figure 3: Superior balance between solution quality and computational efficiency. Bubble size indicates success rate. (a) Per-instance total time. (b) Per-step search time.

Table 3: Ablation results. **Obj.** is makespan (lower is better). D indicates deadlock. D_s is avg. deadlock step.

	Size	MCTS		MCTS-C		MCTS-D-R		Ours
		Obj↓	D_s ↑	Obj↓	D_s ↑	Obj↓	D_s ↑	Obj↓
ELISA	6 × 7	65.8	–	65.8	–	63.8	–	**60.8**
	12 × 7	D	20.5	134.6	–	D	25.2	**127.5**
	24 × 7	D	29.8	D	28.9	D	87.3	**155.9**
	48 × 7	D	33.8	D	32.6	D	243.6	**319.4**
	96 × 7	D	37.6	D	37.0	D	571.6	**646.7**
NGS	6 × 10	111.6	–	114.3	–	112.3	–	**108.2**
	12 × 10	202.4	–	**200.4**	–	200.8	–	202.4
	24 × 10	D	39.1	D	40.5	D	117.1	**297.7**
	48 × 10	D	44.7	D	44.7	D	327.8	**644.1**
	96 × 10	D	49.9	D	49.9	D	793.6	**1421.9**
HDD	7 × 17	D	11	523	18	D	14	**502.6**
	11 × 17	D	21.6	D	36	D	85.6	**704.0**

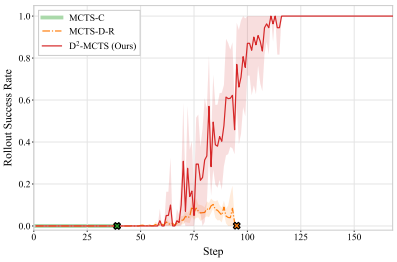


Figure 4: Rollout success rate at each decision step on ELISA 24×7. The y-axis denotes the rollout success rate of reaching a feasible solution at each step.

for the real-time demands of automated laboratory environments. Besides, although L2D* achieves inferior makespan performance compared to D²-MCTS, the progression from the complete failure of L2D to L2D*'s ability to find feasible solutions on most instances demonstrates the effectiveness of our proposed deadlock depth metric, which extends beyond MCTS applications.

5.3 Ablation Study

We perform an ablation study to evaluate our design choices by comparing D^2-MCTS against three variants: (1) classical MCTS; (2) MCTS-C, which replaces deadlock depth with deadlock count as the guidance signal; and (3) MCTS-D-R, which couples the deadlock penalty directly into the Q-value rather than maintaining a separate risk statistic. Table 3 reports makespan and average deadlock step D_s; Figure 4 visualizes the per-step rollout success rate on ELISA 24×7 across 10 test instances. All three baselines exhibit 0% rollout success before step 50, confirming that dense deadlocks cause complete signal vanishing in large-scale RBHFS under a limited search budget. MCTS-C marginally extends the reachable depth ($D_s \approx 30$, marked as green crosses in Figure 4) but still deadlocks on larger instances. MCTS-D-R successfully escapes the total-deadlock zone by leveraging depth information; however, coupling the deadlock penalty into Q makes it difficult to balance deadlock avoidance and makespan optimization, ultimately leading to deadlocks. D^2-MCTS resolves both issues: the risk statistic guides the search through the dense-deadlock region, after which the algorithm transitions to effective makespan optimization and achieves the best results across all scales.

5.4 Mechanism Analysis: Adaptive Concurrency Control

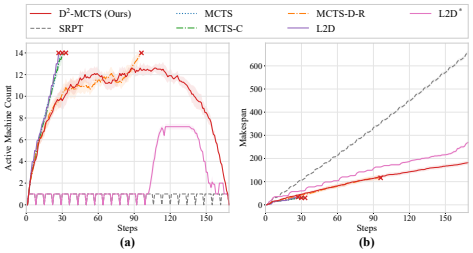


Figure 5: Evolutionary dynamics of system concurrency and makespan on ELISA 24×7. Curves show means over 10 instances, and shaded bands denote 95% confidence intervals.

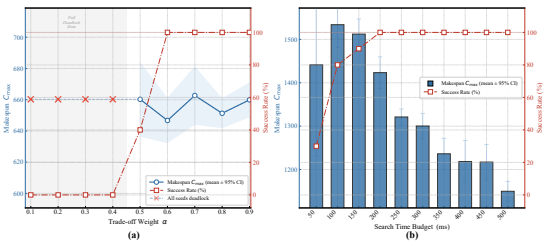


Figure 6: parameter sensitivity (a) α in ELISA 96×7. (b) Search time per step in NGS 96×10. Curves and bars show means over 10 instances; shaded bands and error bars denote 95% confidence intervals.

Furthermore, we analyze the effectiveness of our method from another perspective. To minimize makespan, the algorithm must increase the number of concurrent jobs. However, this increases contention for limited resources, which raises the risk of deadlocks. How to strike an adaptive balance is difficult. Figure 5 visualizes the evolution of system concurrency (active machine count) across methods. MCTS greedily maximizes concurrency until the system crosses the deadlock threshold

and collapses; SRPT and L2D take the opposite approach, maintaining near-serial execution (≈ 1 active machine) that sacrifices throughput for safety—L2D arrives at this conservative policy through deadlock penalties during training rather than by design. In contrast, D^2-MCTS discovers a safe concurrency upper bound: the decoupled depth statistic acts as a soft throttle, pruning actions that spike deadlock risk while sustaining high machine utilization just below the saturation threshold. This adaptive balance between safety and throughput underlies its superior makespan performance.

5.5 Hyperparameter Sensitivity Analysis

In this section, we evaluate two key hyperparameters in D^2-MCTS: (1) the trade-off weight α controlling the balance between deadlock avoidance and makespan optimization, and (2) the per-step search time budget T in D^2-MCTS. As shown in Figure 6(a), when α is too small, aggressive makespan optimization leads to more deadlocks, and success rate improves as α increases. To avoid deadlocks in RBHFS instances during testing, we recommend setting $\alpha > 0.6$ in practice. Figure 6(b) shows that the success rate improves monotonically with budget and reaches saturation around 100–150 ms. Makespan similarly decreases with diminishing returns beyond 200 ms. This rapid convergence demonstrates that lookahead is efficiently exploited, enabling practical real-time deployment. Besides, we provide a more detailed sensitivity analysis result in Appendix C.5. For α, Figure 8 reveals a clear threshold-like behavior: once α reaches a value at which the scheduling success rate remains consistently high, further increases have only a small effect on makespan. This pattern also suggests that an effective value of α can be identified with modest empirical tuning. For example, a small number of trial runs, using binary search when the threshold-like behavior is observed, can identify a threshold that consistently yields high scheduling success rates. Since D^2-MCTS requires no training and each scheduling run is computationally inexpensive, this tuning introduces only a modest additional computational cost.

6 Limitations and Future Work

Although D^2-MCTS provides an effective solution for RBHFS scheduling in automated laboratories and meets the requirements of dynamic scheduling, it requires longer computation time at test time compared to offline learning-based methods. The laboratory-oriented benchmark instances used in our experiments are synthetically generated to represent high-deadlock-pressure scenarios. Although D^2-MCTS performs effectively on both these synthetic instances and third-party benchmarks, the asymptotic analysis characterizes its deadlock-avoidance behavior only as $N \to \infty$. A gap remains between the asymptotic theory and the finite-budget regime used in practice. Besides, we introduce deadlock depth as a proxy indicator to guide the learning-based method L2D*. Preliminary results show that while makespan is suboptimal, L2D* can find feasible solutions on most instances, compared to L2D, which completely fails. In future work, we will further apply the deadlock depth metric to learning-based methods for solving RBHFS to achieve faster solving times, where the key challenge will be ensuring robust generalization.

7 Conclusion

In this paper, we address the failure of standard MCTS in RBHFS scheduling, where the algorithm degenerates into random search due to the prevalence of "dense deadlock". We identify that deadlock depth serves as a robust indicator of latent feasibility, as nodes reaching greater depths show a strong positive correlation with true feasibility probability. Based on this insight, we propose Deadlock-Depth Guided MCTS (D^2-MCTS), which integrates depth statistics into the evaluation mechanism to effectively differentiate promising branches within the infeasible search space. Empirical results demonstrate that D^2-MCTS achieves superior robustness and efficiency on large-scale benchmarks where state-of-the-art constructive baselines fail to locate feasible solutions, providing an effective solution to the deadlock challenges arising in automated laboratory scheduling.

Acknowledgments and Disclosure of Funding

This work was supported by the Civil Space Technology Pre-Research Program under Grant No. D020205 and by the National Natural Science Foundation of China under Grant Nos. 62372028, 62372027 and U23B2025.

References

[1] Yuya Arai, Ko Takahashi, Takaaki Horinouchi, Koichi Takahashi, and Haruka Ozaki. Sagas: Simulated annealing and greedy algorithm scheduler for laboratory automation. *SLAS technology*, 28(4):264–277, 2023.

[2] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[3] Chettha Chamnanlor, Kanchana Sethanan, Chen-Fu Chien, and Mitsuo Gen. Re-entrant flow shop scheduling problem with time windows using hybrid genetic algorithm based on auto-tuning strategy. *International Journal of Production Research*, 52(9):2612–2629, 2014.

[4] Inguk Choi, Woo-Jin Shin, Sanghyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[5] Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. *Advances in Neural Information Processing Systems*, 37:105528–105551, 2024.

[6] Adel Dabah, Ahcene Bendjoudi, Abdelhakim AitZai, and Nadia Nouali Taboudjemat. Efficient parallel tabu search for the blocking job shop scheduling problem. *Soft Computing*, 23(24):13283–13295, 2019.

[7] Hastyn Doshi, Ayush Kumar Tripathi, Keshav Agarwal, Harshad Khadilkar, and Shivaram Kalyanakrishnan. Linear-time optimal deadlock detection for efficient scheduling in multi-track railway networks. In *AI for Critical Infrastructure Workshop@ IJCAI-24*, 2024.

[8] Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:48485–48509, 2023.

[9] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

[10] Stephen R Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. *GSIA, Carnegie Mellon University*, 1984.

[11] Kun Lei, Peng Guo, Yi Wang, Jianyu Xiong, and Wenchao Zhao. An end-to-end hierarchical reinforcement learning framework for large-scale dynamic flexible job-shop scheduling problem. In *2022 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2022.

[12] Kexin Li, Qianwang Deng, Like Zhang, Qing Fan, Guiliang Gong, and Sun Ding. An effective mcts-based algorithm for minimizing makespan in dynamic flexible job shop scheduling problem. *Computers & Industrial Engineering*, 155:107211, 2021.

[13] Longkang Li, Siyuan Liang, Zihao Zhu, Chris Ding, Hongyuan Zha, and Baoyuan Wu. Learning to optimize permutation flow shop scheduling via graph-based imitation learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 20185–20193, 2024.

[14] Xiaoling Li, Keyi Xing, and Qingchang Lu. Hybrid particle swarm optimization algorithm for scheduling flexible assembly systems with blocking and deadlock constraints. *Engineering Applications of Artificial Intelligence*, 105:104411, 2021.

[15] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

[16] Chun-Cheng Lin, Wan-Yu Liu, and Yu-Hsiang Chen. Considering stockers in reentrant hybrid flow shop scheduling with limited buffer capacity. *Computers & Industrial Engineering*, 139:106154, 2020.

[17] Kui Liu, Zhiwei Wu, Qing Wu, and Yuxia Cheng. Smart dag task scheduling with efficient pruning-based mcts method. In *2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 348–355. IEEE, 2019.

[18] Renke Liu, Rajesh Piplani, and Carlos Toro. Deep reinforcement learning for dynamic scheduling of a flexible job shop. *International Journal of Production Research*, 60(13):4049–4069, 2022.

[19] Youshan Liu, Jiaxin Fan, Chunjiang Zhang, and Weiming Shen. A knowledge-driven deep reinforcement learning approach for dynamic scheduling of re-entrant hybrid flow shop with in-line product quality inspection. *Knowledge-Based Systems*, page 114418, 2025.

[20] Hugo Hissashi Miyata and Marcelo Seido Nagano. The blocking flow shop scheduling problem: A comprehensive and conceptual review. *Expert Systems with Applications*, 137:130–156, 2019.

[21] Johanna Mlekusch and Richard F Hartl. The dual-resource-constrained re-entrant flexible flow shop a constraint programming approach and a hybrid genetic algorithm. *International Journal of Production Research*, 63(5):1803–1824, 2025.

[22] Chatnugrob Sangsawang, Kanchana Sethanan, Takahiro Fujimoto, and Mitsuo Gen. Metaheuristics optimization approaches for two-stage reentrant flexible flow shop with blocking constraint. *Expert Syst. Appl.*, 42(5):2395–2410, 2015. doi: 10.1016/J.ESWA.2014.10.043. URL `https://doi.org/10.1016/j.eswa.2014.10.043`.

[23] M. Saqlain, S. Ali, and J. Y. Lee. A monte-carlo tree search algorithm for the flexible job-shop scheduling in manufacturing systems. *Flexible Services and Manufacturing Journal*, page 548–571, Jun 2023. doi: 10.1007/s10696-021-09437-4. URL `http://dx.doi.org/10.1007/s10696-021-09437-4`.

[24] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

[25] Jingwen Yuan, Kaizhou Gao, Adam Slowik, Benxue Lu, and Yanan Jia. Scheduling reentrant flowshops: Reinforcement learning-guided meta-heuristics. *IET Collaborative Intelligent Manufacturing*, 7(1):e70029, 2025.

[26] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632, 2020.

[27] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to search for job shop scheduling via deep reinforcement learning. *CoRR*, abs/2211.10936, 2022. doi: 10.48550/ARXIV.2211.10936. URL `https://doi.org/10.48550/arXiv.2211.10936`.

[28] Jiayuan Zhang, Xuefeng Liu, Jianwei Niu, Shaojie Tang, Haotian Yang, and Xinghao Wu. Causality-inspired federated learning for ood generalization. In *ICML*, 2025.

[29] Jiayuan Zhang, Xuefeng Liu, Jianwei Niu, Haotian Yang, Wanyu Lin, and Xinghao Wu. The aggregated model is a confounder: Enabling deconfounded federated learning for ood generalization. In *IEEE INFOCOM 2026-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2026.

[30] Qianqian Zheng, Yu Zhang, Hongwei Tian, and Lijun He. A cooperative adaptive genetic algorithm for reentrant hybrid flow shop scheduling with sequence-dependent setup time and limited buffers. *Complex & Intelligent Systems*, 10(1):781–809, 2024.

A Related Works

A.1 More Details about the SOTA Learning-based Methods for JSP/FSP and Why They Cannot Handle RBHFS

Learning-based methods for JSP/FSP belong to the constructive method category. These methods model the scheduling problem as an MDP, enabling step-by-step decision-making conditioned on the current scheduling state—a paradigm naturally suited to dynamic scheduling demands. Unlike MCTS, which is an online search method, learning-based approaches train policy networks offline and apply them at test time.

Policy Gradient Methods: L2D [26] is the seminal work, using Graph Neural Networks (GNNs) to encode disjunctive graphs and employing PPO for training. To collect training signals, L2D requires executing rollout simulations during training, computing rewards from complete trajectories. **Self-Supervised Methods:** SLIM [5] adopts a self-supervised paradigm that avoids expensive annotations from exact solvers. It samples multiple trajectories from a generative model, selects the best solution based on objective value, and uses it as a pseudo-label for training. This iterative process gradually improves policy performance without external supervision. **Multi-Policy Approaches:** POMO [9] exploits symmetries in CO solution representations by training toward multiple optimal solutions using a modified REINFORCE algorithm. Multi-policy methods like Poppy [8] and MP-ASIL [4] address mode collapse by training complementary policies on distinct problem subsets, with each policy updated only on instances where it performs best. In standard JSP/FSP, rollout simulations reach terminal states and produce valid reward signals. This enables both policy gradient methods (PPO) to receive informative gradients and self-supervised methods (SLIM, POMO) to identify pseudo-labels from complete, feasible trajectories.

However, in RBHFS under dense deadlock conditions, virtually all rollout simulations terminate in deadlock before reaching any feasible terminal state. For all learning-based methods, this is fatal. **For PPO-based methods (L2D):** The training procedure collects almost no trajectories with valid reward signals. The network receives almost no informative gradient signals and fails to learn any meaningful policy. **For self-supervised methods (SLIM, POMO, MP-ASIL):** Even after sampling hundreds of trajectories, virtually all rollouts terminate in deadlock with no feasible solutions. It is impossible to construct any pseudo-label, making it impossible to train the generative model.

For both constructive learning-based methods, the training procedure collects almost no trajectories with valid signals, making it impossible for any network to learn any meaningful policy. This applies equally to all existing RL-based and self-supervised schedulers, whether designed for unconstrained or other constrained variants, because none incorporates a mechanism to extract useful information from deadlocked trajectories. Empirical verification: when we applied L2D directly to RBHFS, it failed to find any feasible solution, and the network did not converge.

Additionally, the failure mechanism in RBHFS is fundamentally distinct from standard JSP/FSP. A generalization error in a learned policy merely degrades solution quality. Even if a policy selects a suboptimal action, subsequent decisions remain available. The cumulative impact manifests as longer makespan, not failure. The algorithm always generates a valid solution. A single erroneous scheduling decision triggers irreversible deadlock—an empty action set from which no recovery is possible. This is categorically different: the error does not degrade quality; it causes system failure. The system cannot recover. This asymmetry mandates a fundamental paradigm shift. Offline-trained neural policies remain vulnerable to generalization errors when deployment states are out of distribution [28, 29]; in RBHFS, a single such error can trigger an irreversible deadlock, precluding a general feasibility guarantee.

B Detailed Theoretical Analysis and Proofs

In this appendix, we provide a rigorous derivation of the properties of D^2-MCTS. We demonstrate that, under Assumption 4.1 and as $N \to \infty$, the depth-guided penalty successfully restores search gradients in dense deadlock environments and ensures asymptotic convergence to the optimal feasible solutions.

B.1 Definitions and Problem Setup

We formally define the variables and constants used in our analysis in Table 4.

Table 4: Formal Definitions of Variables and Constants

Symbol	Term	Description
s	Current State	The current node in the search tree representing the scheduling state.
a	Action	A valid scheduling decision (e.g., assigning a task to a machine).
$Q(s,a)$	Q-value	The Monte Carlo estimation of the expected return for action a in state s. Each edge is initialized to $Q(s,a) \leftarrow P_{\text{init}}$ before any rollout is observed.
τ	Trajectory	A simulated sequence of actions from the current state to a terminal state.
$d(\tau)$	Trajectory Depth	The total number of operations successfully scheduled in trajectory τ.
$L_{\max}$	Maximum Depth	The total number of operations to be scheduled (total search tree depth).
$\rho(\tau)$	Deadlock Risk	A safety metric for a single trajectory: $\rho(\tau) = \max(0, 1 - d(\tau)/L_{\max})$.
$\bar{\rho}(a)$	Expected Risk	The average deadlock risk over N simulations for action a.
$C_{\max}(\tau)$	Makespan	The total completion time of a feasible trajectory.
$\text{LB}(s)$	Critical-Path LB	Optimistic makespan lower bound from state s: $\max\Big(\max_m C_m,\ \max_j \Big(E_j + \sum_{k=\text{op}_j}^{L_j} p_{j,k}\Big)\Big)$. Used as surrogate Q-value for deadlock rollouts.
$\tilde{Q}(s,a)$	Normalized Q	Min-max normalization among siblings: $\tilde{Q} = (Q - Q_{\min})/(Q_{\max} - Q_{\min}) \in [0,1]$.
α	Trade-off Weight	Convex combination parameter, $\alpha \in [0,1]$. $\alpha = 0$: pure makespan; $\alpha = 1$: pure deadlock avoidance. Proposition 4.1 requires $\alpha < 1$.
$R(s,a)$	Cumulative Risk	The sum of deadlock risks over all simulations passing through edge (s,a).
$r(\tau)$	Trajectory Reward	$-C_{\max}(\tau)$ for feasible; $-\text{LB}(s_{\text{dead}})$ for deadlock.

B.2 Proof of Proposition 4.1

Proof. Let the MDP be finite and deterministic. We partition the action space at the root into two disjoint subsets: $\mathcal{A}_{\text{feas}}$ (actions containing at least one path to a feasible terminal state in their subtrees) and $\mathcal{A}_{\text{dead}}$ (actions where all paths inevitably lead to global deadlocks).

In D^2-MCTS, the child-selection criterion is driven by a modified empirical reward signal $X(v)$:

$$X(v) = (1-\alpha)\tilde{Q}(v) + \alpha(1 - \bar{\rho}(v)) \tag{5}$$

A fundamental challenge in RBHFS is that deadlocks are evaluated using an optimistic critical-path lower bound ($-\text{LB}$). Because $\text{LB} \leq C_{\max}$, an infeasible deadlock path may yield a numerically higher raw Q-value than a feasible path, deceiving standard UCT.

However, under **Assumption 4.1**, we guarantee that for any feasible terminal leaf l_{feas} (where $\rho = 0$) and any deadlock leaf l_{dead} (where $\rho = \rho_{\text{dead}} > 0$), their true combined evaluations satisfy:

$$X(l_{\text{feas}}) - X(l_{\text{dead}}) = \Big[(1-\alpha)\tilde{Q}_{\text{feas}} + \alpha(1)\Big] - \Big[(1-\alpha)\tilde{Q}_{\text{dead}} + \alpha(1-\rho_{\text{dead}})\Big] \tag{6}$$

$$= \alpha \cdot \rho_{\text{dead}} - (1-\alpha)(\tilde{Q}_{\text{dead}} - \tilde{Q}_{\text{feas}}) > 0. \tag{7}$$

This implies that, under the modified reward X, every feasible trajectory is strictly preferred to any deadlock trajectory, despite the latter's optimistic lower-bound estimate.

Since D^2-MCTS applies the standard UCT bounds on this modified bounded reward $X \in [0,1]$, we can invoke the classical regret bounds of UCT (Kocsis & Szepesvári, 2006). For an action $a \in \mathcal{A}_{\text{feas}}$, the algorithm will eventually discover the optimal feasible path. Once discovered, this path becomes the optimal arm for the UCT bandit. By the UCT visit bounds, the optimal arm receives a linear number of visits $\mathcal{O}(N)$, while all suboptimal arms (including all deadlocks in the subtree) receive at most $\mathcal{O}(\ln N)$ visits.

Consequently, the running average of the deadlock risk for $a \in \mathcal{A}_{\text{feas}}$ converges to:

$$\lim_{N\to\infty} \bar{\rho}_N(a) = \lim_{N\to\infty} \frac{\mathcal{O}(N)\cdot 0 + \mathcal{O}(\ln N)\cdot \rho_{\text{dead}}}{N} = 0. \tag{8}$$

Conversely, for an action $a \in \mathcal{A}_{\text{dead}}$, every trajectory deterministically ends in a deadlock. Its average risk converges to the expected regional deadlock risk:

$$\lim_{N\to\infty} \bar{\rho}_N(a) = \bar{\rho}_{\text{dead}} > 0. \tag{9}$$

Substituting these converged limits back into the selection score, for sufficiently large N, the score asymptotically behaves as:

$$\text{Score}_N(a) \approx c\sqrt{\frac{2\ln N}{N(a)}} + \begin{cases} (1-\alpha)\tilde{Q}(a) + \alpha, & \text{if } a \in \mathcal{A}_{\text{feas}} \\ (1-\alpha)\tilde{Q}(a) + \alpha(1-\bar{\rho}_{\text{dead}}), & \text{if } a \in \mathcal{A}_{\text{dead}} \end{cases} \tag{10}$$

Due to Assumption 4.1, the score for any action in $\mathcal{A}_{\text{dead}}$ is strictly upper-bounded by the score of the optimal action in $\mathcal{A}_{\text{feas}}$. In the UCT framework, strictly suboptimal actions are selected with vanishing probability as $N \to \infty$, yielding $\lim_{N\to\infty} \frac{N(a\in\mathcal{A}_{\text{dead}})}{N} = 0$. This effectively isolates the search to the feasible subspace $\mathcal{A}_{\text{feas}}$.

Within $\mathcal{A}_{\text{feas}}$, the depth term $\alpha(1-\bar{\rho})$ converges to the uniform constant α for all actions. Since Assumption 4.1 requires $\alpha < 1$, the coefficient $1-\alpha$ is strictly positive, the relative ordering is determined entirely by $(1-\alpha)\tilde{Q}(a)$. Because affine transformations do not alter the $\arg\max$ operator, the selection criterion asymptotically reduces to identifying the feasible action with the optimal makespan. Therefore, the optimal feasible action a^* receives the majority of visits:

$$\lim_{N\to\infty} \frac{N(a^*)}{N} = 1. \tag{11}$$

This proves the conditional asymptotic result in Proposition 4.1 under Assumption 4.1 as $N \to \infty$. □

B.3 Ruling Out Proxy Hacking: Why Deadlock Depth Reliably Indicates Branch Promise

A natural concern regarding our depth-based guidance is the *proxy hacking* hypothesis: why didn't the search in D^2-MCTS prefer branches that remain longer in infeasible regions but eventually deadlock? We conducted a deeper analysis to explain it.

Local Deadlocks Lead Irreversibly to Global Deadlocks A local deadlock (where a subset of jobs interlock but $\mathcal{A}(S) \neq \emptyset$) inevitably leads to a global deadlock ($\mathcal{A}(S) = \emptyset$). Due to fixed routing and zero-buffer constraints, this circular wait caused by the interlock is physically irreversible.

For instance, assume jobs process sequentially through $M_1 \to M_2 \to M_3$. If A_{bad} causes two jobs to interlock at M_3, MCTS can only schedule other jobs to "delay" the global deadlock. However,

under identical routing constraints in FSP, subsequent jobs will sequentially occupy M_1 and M_2, then be permanently blocked at M_3.

This triggers a rapid backward cascading blockage, exhausting upstream capacity in very few steps.

Therefore, the number of steps to reach a global deadlock has a strictly limited upper bound C_{up} (determined by the number of upstream machines). Thus, A_{bad} cannot "remain indefinitely" in infeasible regions; it can only survive for a few steps.

Expected Depth Dominance Under Sufficient Rollouts We now analyze why D^2-MCTS avoids selecting A_{bad}.

Let $d(\tau)$ be the deadlock depth of rollout trajectory τ. Once A_{bad} enters a local deadlock, its expected distance to a global deadlock is strictly truncated by the physical bound:

$$\mathbb{E}[d(\tau) \mid A_{\text{bad}}] \leq d(S_t) + C_{\text{up}}$$

In contrast, assume A_{good} lies on a path leading to a feasible region. Its expected depth can be expressed as:

$$\mathbb{E}[d(\tau) \mid A_{\text{good}}] = d(S_t) + d_{\text{free}}$$

where d_{free} is the expected survival steps of subsequent rollouts along a feasible path. Crucially, d_{free} scales with the total remaining operations to be scheduled, which is $\mathcal{O}(J \times K)$. In contrast, C_{up} is strictly bounded by the fixed physical capacity of upstream machines, $\mathcal{O}(K)$. For non-trivial scheduling instances where $J \gg 1$, d_{free} is orders of magnitude larger than C_{up}.

Statistically, given sufficient rollouts, the average trajectory depth of A_{good} will always be strictly greater than that of the tightly constrained A_{bad}. Therefore, D^2-MCTS inherently tends to explore A_{good} (which receives a smaller risk penalty and higher selection score) and avoids wasting substantial search effort on A_{bad}.

C More Details about Experiments

C.1 Details for Test Data Set

Rationale and Design Philosophy Real-world life science automation involves intricate constraints, including liquid handling precision, reagent stability, and physical transport limitations. However, the computational hardness of scheduling these systems stems primarily from the intersection of two structural properties: cyclic re-entrancy (jobs revisiting stations) and zero-buffer blocking (no intermediate storage).

To rigorously evaluate solver robustness, we construct a high-fidelity benchmark suite derived from representative assay workflows (ELISA and NGS Library Prep). While abstracting away low-level physical details, these instances preserve the critical resource contention topologies that induce system-wide deadlocks. By systematically varying the scale and resource tightness, we create a stress-test environment that exposes the failure modes of traditional scheduling algorithms.

Workflow Topologies We define benchmark families representing common classes of re-entrant flows. The route definitions and baseline configurations for the two primary families, ELISA and NGS, are detailed in Table 5.

- **ELISA (Sandwich Assay):** A classic cyclic workflow where microplates oscillate between dispensing, incubation, and washing. The *Washer* and *Reader* act as critical shared resources, creating tight re-entrant loops.
- **NGS-PREP (Library Construction):** Represents iterative DNA preparation processes. The workflow involves three repeated cycles of `Pipette-Incubate-Purify`, simulating a long-horizon dependency chain that accumulates blocking risks.

Differential Resource Scaling A key feature of our benchmark is the **differential scaling policy**. Simply scaling machine counts linearly with job counts often trivializes the scheduling problem. To maintain high deadlock pressure at large scales, we separate stages into *Standard* and *Bottleneck* groups.

Table 5: Configuration of Laboratory-inspired Workflows. Processing times are sampled uniformly from the given ranges.

Family	Re-entrant Route (Stage Sequence)	Processing Times ($U[min, max]$)	Bottleneck Stages
ELISA	Disp → Inc → Wash → Disp → Inc → Wash → Read	Disp:[2,4], Inc:[8,12], Wash:[3,5], Read:[2,3]	Wash, Read
NGS	(Pipette → Inc → Purify) ×3 → QC	Pip:[2,4], Inc:[10,15], Purify:[4,6], QC:[3,4]	Purify, QC

Let m_{base} be the baseline machine count for a stage. For a given instance scale $S \in \{\text{Small, Medium, Large, XL, XXL}\}$, the available machines m_S are calculated as:

$$m_S = \max\left(1, \text{round}(m_{base} \times k_S)\right) \tag{12}$$

where k_S is a scale-dependent multiplier. We apply a conservative multiplier $k_{bottleneck}$ for critical resources (e.g., Washers in ELISA, Purifiers in NGS) and a relaxed multiplier k_{std} for others. This asymmetry ensures that as job density increases ($N = 6 \rightarrow 96$), the ratio of jobs to bottleneck resources rises, forcing the solver to navigate an increasingly narrow feasible region. The procedural generation logic is formalized in Algorithm 1.

Algorithm 1 Procedural Benchmark Generation with Differential Scaling

Input: Workflow Family $\mathcal{F}$, Scale Level $\mathcal{S}$, Number of Jobs J
Output: Scheduling Instance $\mathcal{I} = (\text{Jobs}, \text{Machines})$
Let $\mathcal{R}_{\mathcal{F}}$ be the fixed re-entrant route for family $\mathcal{F}$
Let M_{base} be the baseline machine counts for $\mathcal{F}$
Let B be the set of bottleneck stages for $\mathcal{F}$
▷ **Phase 1: Differential Machine Scaling**
for each stage s in M_{base} **do**
    if $s \in B$ **then**
        $k \leftarrow \text{GetBottleneckMultiplier}(\mathcal{S})$ ▷ Conservative growth (e.g., 1.5x)
    else
        $k \leftarrow \text{GetStandardMultiplier}(\mathcal{S})$ ▷ Standard growth (e.g., 2.0x)
    end if
    $m_{\text{count}} \leftarrow \max(1, \text{round}(M_{\text{base}}[s] \cdot k))$
    Create m_{count} machines for stage s
end for
▷ **Phase 2: Job Instantiation**
for $j = 1$ to J **do**
    Create Job j with operation sequence following $\mathcal{R}_{\mathcal{F}}$
    Sample processing times from defined ranges
end for
return Instance $\mathcal{I}$

Blocking Semantics and Dataset Composition **Zero-Buffer Constraint:** We strictly enforce a zero-buffer blocking constraint. A job j completing operation o on machine M holds M until it successfully commences operation $o + 1$ on a downstream machine M'. The release time of M is defined as $T_{release}(M) = \max(T_{finish}(o), T_{start}(o + 1))$. This resource holding mechanism propagates congestion upstream, creating the circular dependencies characteristic of deadlocks.

Dataset Generation: We generate instances at five scales ($J \in \{6, 12, 24, 48, 96\}$). To ensure robust statistical evaluation, we generate 10 random seeds per scale for each workflow family (ELISA and NGS). This results in 50 instances per family, totaling 100 instances in the complete Synthetic RHFS benchmark suite.

C.2 Empirical Validation of the Depth–Feasibility Correlation

To directly validate the core hypothesis of D^2-MCTS—that average rollout deadlock depth serves as a reliable proxy for true feasibility probability—we conduct probe experiments on two instances of distinct types and scales.

Setup. At selected decision steps during a D^2-MCTS run, we pause the search and evaluate each candidate action *independently* using a large rollout budget in each step (500,000 for ELISA 12×7 and NGS 24×10). For each action a, we measure: (1) the *average deadlock depth* $\bar{d}(a)$ over all rollouts, and (2) the *empirical feasibility rate* $\hat{p}(a)$, i.e., the fraction of rollouts reaching a complete valid schedule. At this rollout scale, $\hat{p}(a)$ is a reliable approximation of the true feasibility probability $p(a)$. Crucially, these probe rollouts are *independent* of the D^2-MCTS selection mechanism, eliminating any self-confirming bias.

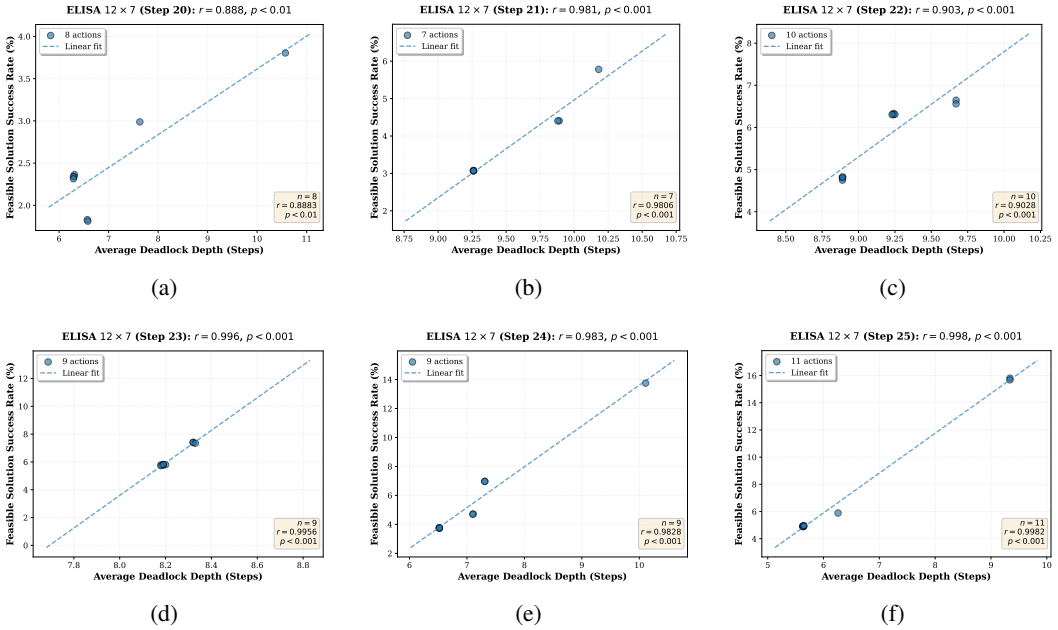


Figure 7: Empirical validation of the depth–feasibility correlation across multiple search steps in ELISA 12×7 instance. Each subplot displays the scatter plot of average rollout deadlock depth (x-axis) versus empirical feasibility rate (y-axis) for candidate actions evaluated at different decision steps. A linear regression line (dashed) is overlaid to visualize the positive monotone relationship. The consistent positive correlation across all steps (Steps 20–25) and varying scheduling phases confirms that deadlock depth is a reliable and robust proxy for action feasibility, validating the core hypothesis underlying D^2-MCTS.

Results. Figure 7 visualizes the scatter plots of average deadlock depth versus rollout success rate across search steps 20–25 on the ELISA 12×7 instance. Table 6 provides detailed quantitative results from representative steps on both ELISA 12×7 and NGS 24×10 instances. Across the examined decision points, actions with higher $\bar{d}(a)$ consistently exhibit higher $\hat{p}(a)$. For ELISA steps 20–25, the step-wise Pearson correlation coefficients range from $r = 0.888$ to $r = 0.998$; the correlation is significant at $p = 0.0032$ for step 20 and at $p < 0.001$ for steps 21–25. For NGS steps 92–95, the coefficients range from $r = 0.918$ to $r = 0.982$, with $p < 0.001$ at every step, confirming that the correlation is statistically robust.

C.3 More experimental results for MCTS with informed rollout baselines.

In this section, we replace the uniform-random rollout policy of fixed-penalty MCTS with six scheduling heuristics: SPT, LPT, MWKR, SRPT, WINQ, and PT+WINQ. The results are shown in Table 7. All variants use the same tree policy, action space, fixed deadlock penalty, instances, seeds, and per-decision search budgets; only the rollout job-priority rule changes. Among these policies, MCTS+SRPT is the strongest informed-rollout baseline. The remaining policies deadlock throughout the reported table. Despite using only a uniform-random rollout policy, D^2-MCTS achieves a lower makespan than MCTS+SRPT in all six ELISA/NGS scale settings, with average reductions of 5.05% on ELISA and 18.09% on NGS.

Table 6: Probe experiment results: average rollout deadlock depth $\bar{d}(a)$ vs. empirical feasibility rate $\hat{p}(a)$ for candidate actions at representative decision steps. Each action is evaluated with $500{,}000$ independent rollouts. Actions are sorted by $\bar{d}(a)$ in ascending order.

Instance	Action	Avg. Depth $\bar{d}(a)$	Feasibility Rate $\hat{p}(a)$
ELISA 12×7	a_1	6.52	3.74%
(Step 24)	a_2	6.52	3.76%
	a_3	6.52	3.76%
	a_4	6.52	3.78%
	a_5	7.11	4.71%
	a_6	7.11	4.72%
	a_7	7.30	7.02%
	a_8	7.31	6.97%
	a_9	10.10	13.82%
NGS 24×10	b_1	12.04	2.40%
(Step 92)	b_2	12.06	2.55%
	b_3	12.07	2.57%
	b_4	12.07	2.57%
	b_5	12.11	2.52%
	b_6	13.19	3.44%
	b_7	13.25	3.38%
	b_8	13.26	3.49%
	b_9	16.93	5.52%
	b_{10}	17.33	8.13%

Table 7: Informed-rollout MCTS baselines. **Obj.** is the average makespan (lower is better). D indicates deadlock/infeasible.

Type	**Size**	**Baselines**						**Proposed**
		SPT	LPT	MWKR	WINQ	PT+WINQ	SRPT	**D^2-MCTS**
ELISA	24×7	D	D	D	D	D	179.8	**155.9**
	48×7	D	D	D	D	D	324.0	**319.4**
	96×7	D	D	D	D	D	652.0	**646.7**
NGS	24×10	D	D	D	D	D	419.3	**297.7**
	48×10	D	D	D	D	D	773.8	**644.1**
	96×10	D	D	D	D	D	1467.7	**1421.9**

C.4 More experiment results for Deadlock value-update ablation

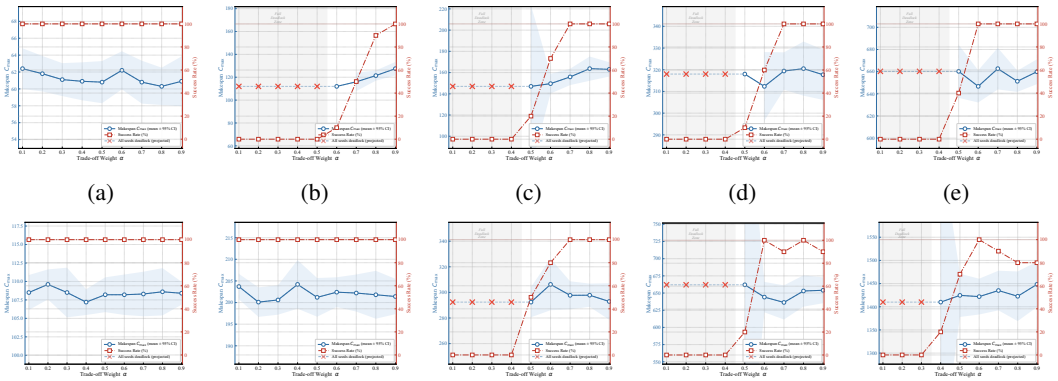
In this section, we hold the depth signal fixed and vary only the value update for deadlock rollouts to test whether the optimistic lower-bound update creates an artifact that the depth penalty merely compensates for. The results are shown in Table 8. The optimistic-LB variant (ours) updates Q with the state-dependent critical-path lower bound; No-Q-update leaves the successful-rollout value estimate unchanged; Fixed penalty assigns the same penalty to every deadlock. All three variants achieve 100% feasibility in the tested settings. Therefore, the optimistic LB is not required to obtain feasible schedules in these experiments. Its contribution is instead to solution quality: it achieves the lowest average makespan in all six settings, with particularly large improvements on ELISA. This ablation also clarifies the distinct information provided by the two components. The no–update alternative discards the quality information from a failed partial schedule, while the fixed penalty treats all such schedules identically. In contrast, the optimistic LB retains a state-dependent estimate of schedule quality when the final makespan is unavailable, providing additional value information that consistently improves solution quality while leaving feasibility unchanged.

Table 8: Deadlock value-update ablation. **Obj.** is the average makespan (lower is better).

Type	**Size**	**Metric**	**Value update for deadlock rollouts**		
			No-Q-update	Fixed penalty	**D^2-MCTS**
ELISA	24×7	**Obj.**↓	190.2	179.8	**155.9**
	48×7	**Obj.**↓	423.6	452.0	**319.4**
	96×7	**Obj.**↓	960.1	972.5	**646.7**
NGS	24×10	**Obj.**↓	301.8	300.2	**297.7**
	48×10	**Obj.**↓	676.2	659.0	**644.1**
	96×10	**Obj.**↓	1482.6	1483.3	**1421.9**

C.5 More experiment results for Hyperparameter Sensitivity, Rollout success rate, and System concurrency

Figure 8 shows the complete alpha sensitivity analysis across all dataset and job scale combinations. The results demonstrate how the makespan and success rate vary with different values of the trade-off weight α in the D^2-MCTS algorithm.

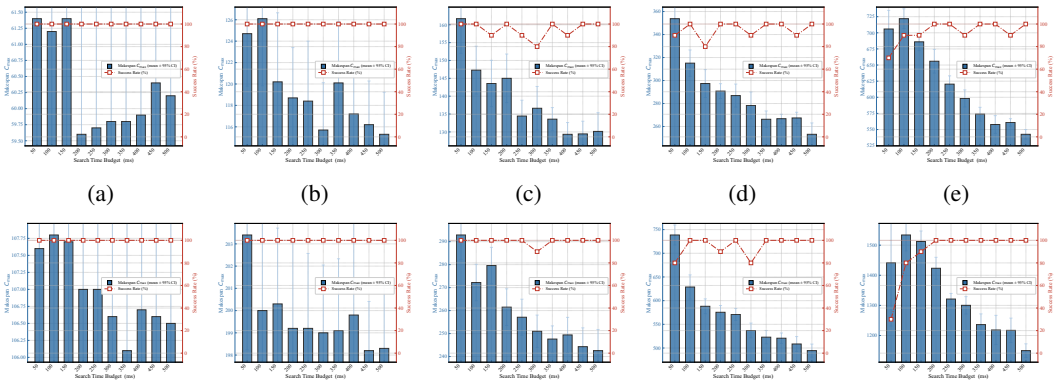


(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 8: Sensitivity analysis of the trade-off weight α parameter across different datasets and job scales. (a) ELISA, $|J| = 6$; (b) ELISA, $|J| = 12$; (c) ELISA, $|J| = 24$; (d) ELISA, $|J| = 48$; (e) ELISA, $|J| = 96$; (f) NGS, $|J| = 6$; (g) NGS, $|J| = 12$; (h) NGS, $|J| = 24$; (i) NGS, $|J| = 48$; (j) NGS, $|J| = 96$. Each plot shows makespan (left axis) and success rate (right axis) as functions of α.

Figure 9 presents the complete search time sensitivity analysis across all dataset and job scale combinations. The results illustrate how the makespan and success rate vary with different search time budgets allocated to each MCTS simulation.

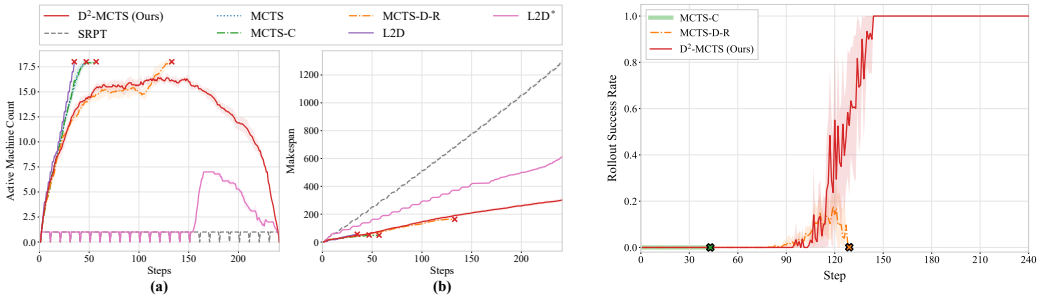


(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 9: Sensitivity analysis of the search time budget parameter across different datasets and job scales. (a) ELISA, $|J| = 6$; (b) ELISA, $|J| = 12$; (c) ELISA, $|J| = 24$; (d) ELISA, $|J| = 48$; (e) ELISA, $|J| = 96$; (f) NGS, $|J| = 6$; (g) NGS, $|J| = 12$; (h) NGS, $|J| = 24$; (i) NGS, $|J| = 48$; (j) NGS, $|J| = 96$. Each plot shows makespan (left axis) and success rate (right axis) as functions of search time budget in milliseconds.

Figure 10 demonstrates the evolutionary dynamics of system concurrency and the rollout success rate at each decision step on the NGS 24×10 instance.



(a) (b)

Figure 10: (a) Evolutionary dynamics of system concurrency and makespan on NGS 24×10. (b) Rollout success rate at each step on NGS 24×10.

C.6 Generalization to Constrained Scheduling: Job Shop with Maximal Time-Lags (JTL-JSP)

The primary focus of this paper is the RBHFS domain, where the co-occurrence of reentrant job flows and zero-buffer blocking creates a structurally sparse feasibility landscape. However, the underlying failure mode of standard MCTS—namely, that *dense deadlock* renders all rollout signals uniformly uninformative—is not unique to RBHFS. Any constrained scheduling problem in which hard constraints cause rollout simulations to fail near-universally will exhibit the same pathology.

To further investigate the generalizability of the deadlock depth indicator discovered by D^2-MCTS across a broader range of universal failure modes, we conduct a transferability experiment on Job Shop Scheduling with Maximal Time-Lags (JTL-JSP). Importantly, D^2-MCTS is applied *without any problem-specific modification*: the depth-guided mechanism transfers directly by treating constraint violations as deadlock states, with no tuning of any hyperparameter.

C.7 Problem Description: JTL-JSP

JTL-JSP extends the classical Job Shop Scheduling Problem (JSP) with *maximal time-lag constraints*: for certain pairs of consecutive operations $(o_{j,l},\ o_{j,l+1})$, the time elapsed between the completion of $o_{j,l}$ and the start of $o_{j,l+1}$ must not exceed a specified maximum lag $\delta_{j,l}$. Violation of this constraint renders the partial schedule infeasible and forces the constructive solver to terminate—the same termination semantics as a deadlock in RBHFS.

When a non-trivial fraction of operation pairs carry tight time-lag constraints, the feasible region of the search space contracts sharply, and random rollouts fail at high rates. We impose hard maximal time-lag constraints on 20% of consecutive operation pairs, with lags sampled uniformly from $[p_{j,l},\ 2 \cdot p_{j,l}]$ (i.e., a window of one processing-time unit above the minimum). This constraint density is sufficient to induce near-universal rollout failure on moderate-scale instances, replicating the dense-deadlock regime of RBHFS.

Experimental Setup We evaluate on the standard Lawrence (LA) benchmark suite [10], which comprises 40 instances across scales from 10×5 to 30×10. For each instance, we independently impose random time-lag constraints and report results averaged over all instances in each group. We compare standard MCTS against D^2-MCTS under identical search budgets ($T = 200$ ms per step). We report: (1) *Feasibility Rate*—the fraction of instances for which a complete feasible schedule is found; (2) *Average Makespan*—computed over feasible instances only (denoted "—" when feasibility rate is 0%).

Results and Analysis. The result as shown in Table 9. Standard MCTS fails to find any feasible solution on all groups with ≥ 15 jobs, exhibiting the same dense-deadlock failure mode documented in the main paper for RBHFS. D^2-MCTS achieves 100% feasibility across all tested scales, including the largest 30×10 instances, with no problem-specific modifications.

Table 9: Generalization to JTL-JSP (LA benchmarks with 20% maximal time-lag constraints). Feasibility rate and average makespan (lower is better) over instances in each scale group. "—" indicates zero feasible solutions found. †LA10×5 results are averaged over 3 structurally feasible instances (see text).

	MCTS		D^2-MCTS (Ours)	
Instance Group	Feasib.%	Avg. Makespan	Feasib.%	Avg. Makespan
LA 10×5†	60%	1040.7	100%	997.8
LA 15×5	0%	—	100%	1448.6
LA 20×5	0%	—	100%	1935.2
LA 10×10	0%	—	100%	1604.3
LA 15×10	0%	—	100%	2116.4
LA 20×10	0%	—	100%	2185.6
LA 15×15	0%	—	100%	2606.0
LA 30×10	0%	—	100%	3756.0

This result confirms that the depth-guided mechanism of D^2-MCTS successfully restores search gradients in JTL-JSP without any domain-specific adaptation, demonstrating that deadlock depth is a domain-agnostic proxy for feasibility proximity whenever constraint violations are structurally correlated with scheduling depth. We emphasize that the primary contribution of this work is the identification of the dense-deadlock failure mode and its resolution in the RBHFS domain, motivated by real-world automated laboratory scheduling. The JTL-JSP experiment is provided as supplementary evidence of transferability. Comprehensive evaluation across the full diversity of constrained scheduling domains is a natural direction for future work.

D More details about Method

This section presents the detailed algorithmic implementation of the proposed D^2-MCTS framework. Algorithm 2 outlines the overall execution flow, including initialization and budget management. Algorithm 3 specifies the core subroutines, detailing the risk-penalized selection, the depth-recording simulation, and the augmented backpropagation mechanisms.

Algorithm 2 Framework of D^2-MCTS

Input: State S_0, Budget T, Max ops $L_{\max}$, Weights c, α
Create root node v_0 with state S_0
while budget T not exhausted **do**
  $v_l \leftarrow$ TREEPOLICY(v_0) ▷ Select & Expand (Alg. 3)
  $r, \rho, q_{\text{est}} \leftarrow$ DEFAULTPOLICY(v_l) ▷ Simulate (Alg. 3)
  BACKUP$(v_l, r, \rho, q_{\text{est}})$ ▷ Update Stats (Alg. 3)
end while
return $\arg\max_{a \in \mathcal{A}(S_0)} \left[(1-\alpha)\widetilde{Q}(S_0, a) + \alpha\,(1 - \bar{\rho}(S_0, a)) \right]$

Algorithm 3 Subroutines for D^2-MCTS

function TREEPOLICY(v)
  while v is not terminal **do**
    if v is not fully expanded **then**
      return EXPAND(v)
    else
      if any child v' of v has $N(v') = 0$ **then**
        return a uniformly random unvisited child of v ▷ Random first visit
      end if
      ▷ Normalized convex-combination score
      Compute $Q_{\min}, Q_{\max}$ over children of v
      $v \leftarrow \arg\max_{v' \in \text{child}(v)} \Big((1-\alpha) \cdot \tilde{Q}(v') + \alpha \cdot \big(1 - \bar{\rho}(v')\big) + c\sqrt{\dfrac{2 \ln N(v)}{N(v')}} \Big)$
      where $\tilde{Q}(v') = \frac{Q(v') - Q_{\min}}{Q_{\max} - Q_{\min}}$, $\bar{\rho}(v') = R(v')/N(v')$
    end if
  end while
  return v
end function

function DEFAULTPOLICY(v)
  $S \leftarrow v$.state; $d_{\text{sim}} \leftarrow 0$
  while S is not terminal **do**
    $a \leftarrow \text{Uniform}(\mathcal{A}(S))$; $S \leftarrow \text{Step}(S, a)$
    $d_{\text{sim}} \leftarrow d_{\text{sim}} + 1$
  end while
  $D \leftarrow \text{depth}(v) + d_{\text{sim}}$ ▷ Global Depth
  if S is feasible **then**
    return $-C_{\max}(S),\ 0,\ \text{Null}$ ▷ Success: risk$= 0$, no LB needed
  else
    ▷ Deadlock: compute risk and LB estimate
    $\rho \leftarrow \max\big(0,\ 1 - D/L_{\max}\big)$
    $q_{\text{est}} \leftarrow -\text{LB}(S)$ ▷ $\text{LB}(S) = \max\big(\max_m C_m,\ \max_j (E_j + \sum_k p_{j,k})\big)$
    return Null, ρ, q_{est}
  end if
end function

function BACKUP(v, r, ρ, q_{est})
  while $v \neq$ Null **do**
    $N(v) \mathrel{+}= 1$; $R(v) \mathrel{+}= \rho$
    if $r \neq$ Null **then**
      $Q(v) \leftarrow \text{Update}(Q(v), r)$ ▷ Feasible: update with true reward
    else
      $Q(v) \leftarrow \text{Update}(Q(v), q_{\text{est}})$ ▷ Deadlock: update with $-$LB (Eq. (3))
    end if
    $v \leftarrow v$.parent
  end while
end function

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state our main contributions and scope.

   Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our work in Section 6

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We provide the proof in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.

- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all implementation details in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide source code and data in the supplemental material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not

including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide all details regarding experiments in Section 5 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Error bars are provided for Figures 4, 5, and 6 in the main paper, and Figures 8, 9, and 10 in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide details on computational resources in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have checked the paper according to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The proposed method has direct applications in automated laboratory scheduling, delivering evident societal benefits through accelerated scientific research and reduced operational costs. Potential negative impacts are limited.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper uses HDD dataset is publicly available under clear terms. The baselines are also standard and open-source.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    The paper introduces a new synthetic RBHFS benchmark with 100 instances across ELISA and NGS workflows spanning 5 problem scales. The dataset generation algorithm (Algorithm 1) and all configuration parameters (Table 5) are fully documented in Appendix C.1. The benchmark is designed to reflect critical real-world laboratory constraints: zero-buffer blocking and reentrant flow.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing experiments and human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing experiments and human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, and non-standard components.

    Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.