DON'T LEARN WHEN TO TRIGGER, LEARN WHEN WAITING IS NOT WORTHWHILE: A NEW FRAMEWORK FOR DYNAMIC TASK-RESOURCE ASSIGNMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

For large-scale task–resource assignment problems in dynamic environments with hard constraints, a common solution is a hierarchical RL–heuristic framework, in which an upper-layer reinforcement learning (RL) policy determines which tasks to trigger at each decision epoch, while a lower-layer heuristic assigns available resources to them. However, since most pending tasks remain untriggered at most decision epochs, RL receives extremely sparse reward signals, leading to ambiguous credit assignment and ineffective triggering policies. Our key insight is that, instead of asking RL to repeatedly decide whether to trigger at each decision epoch, we can reformulate the problem as determining how much can still be gained from waiting. A task is triggered when its delay benefit, defined as the expected-return difference between its best triggering opportunity in the remaining life cycle and triggering it now, falls below a learned threshold. Based on this insight, we propose a new triggering framework that decomposes the original triggering problem into two complementary learning problems. First, a delay-benefit estimator is trained with rollout-generated supervision to estimate how much can still be gained by waiting. Second, RL selects a task-specific threshold that determines when the estimated delay benefit is sufficiently small to trigger the task, while accounting for global competition among tasks and resources. This decomposition replaces repeated RL triggering decisions with supervised delay-benefit estimation and one-time RL threshold selection, providing dense supervision for delay-benefit estimation while allowing RL to focus on long-horizon coordination. Experiments on Dynamic Job-Shop Scheduling, Weapon-Target Assignment, and Warehouse Replenishment demonstrate consistent improvements over pure RL and hybrid RL–heuristic approaches.

1 INTRODUCTION

Dynamic task–resource assignment arises in many real-world systems, including job-shop scheduling, weapon–target assignment, and warehouse replenishment (Zeng et al., 2022; Li et al., 2024; Sultana et al., 2020). In these problems, tasks arrive over time, resource states and feasibility constraints evolve dynamically, and decisions must be made before future information is fully revealed. Moreover, once a task is assigned and executed, the decision is difficult or impossible to revise. These characteristics make it essential to balance immediate task processing against the potential benefit of waiting for better future assignment opportunities.

Reinforcement learning (RL) provides a promising approach to such long-horizon decision-making problems. However, directly using RL to construct complete assignments is difficult in large-scale settings with hard constraints, because the policy must simultaneously learn combinatorial resource allocation and maintain solution feasibility. A common alternative is a hierarchical RL–heuristic framework. In this architecture, an upper-layer RL policy determines which pending tasks should be triggered at each decision epoch, while a lower-layer heuristic assigns available resources to the triggered tasks and rapidly constructs a feasible solution (Yang et al., 2024).

Although this decomposition simplifies the assignment problem by delegating feasible task–resource matching to the lower layer, it shifts a different challenge to the upper layer: learning when each

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

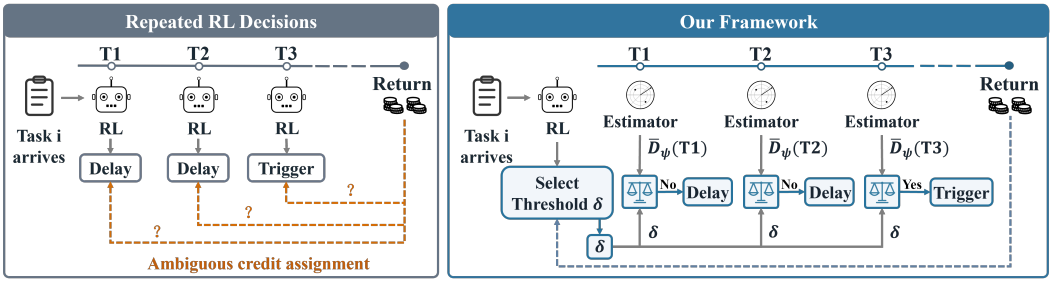

Figure 1: Comparison between repeated RL triggering and our framework. Left: RL repeatedly decides whether to delay or trigger a task, leading to ambiguous credit assignment. Right: RL selects a task-specific threshold once and uses the delay-benefit estimator to determine the triggering time.

task should be triggered. As shown in Fig. 1 (left), RL must repeatedly decide whether to delay or trigger each pending task at decision epochs. Since most tasks remain untriggered for much of their lifecycles, the policy makes many delay decisions but receives little immediate feedback on their quality. Their consequences often become apparent only after a task is eventually triggered and through subsequent system-level rewards. The policy must therefore learn from long sequences of delay decisions with sparse and delayed feedback, making exploration difficult and obscuring how observed rewards should be attributed across decisions, time steps, and tasks.

Our key observation is that task triggering can be guided by the delay benefit, defined as the expected-return gain from waiting for the best future triggering opportunity rather than triggering the task immediately. A task should continue to wait only when further delay is expected to yield sufficient additional return. Based on this observation, we propose a new task-triggering framework that decomposes the direct learning of triggering decisions into two complementary subproblems: estimating the delay benefit and adaptively selecting a triggering threshold. Their interaction is illustrated in Fig. 1 (right). First, a lightweight estimator predicts the delay benefit from the current task–resource state. Training labels are automatically generated from environment rollouts, enabling efficient training via supervised learning. Second, RL learns a task-specific triggering threshold according to the global task–resource state. The estimated delay benefit is compared with this threshold, and the task is triggered once the estimated benefit falls below it.

In summary, our main contributions are as follows. First, we introduce a delay-benefit perspective on task triggering in dynamic task–resource assignment. Rather than directly making repeated binary trigger-or-wait decisions, our approach estimates the benefit of delaying each task and triggers it when this benefit falls below a task-specific threshold. Second, we develop a collaborative framework that combines supervised delay-benefit estimation with RL-based threshold selection, enabling dense task-level supervision while preserving global coordination over competing tasks and resources through long-horizon decision-making under dynamic constraints. Third, we evaluate the proposed framework on Dynamic Job-Shop Scheduling, Weapon–Target Assignment, and Warehouse Replenishment, demonstrating consistent improvements over pure RL and hybrid RL–heuristic approaches.

2 Related Work

2.1 RL Methods for Task–Resource Assignment

RL has been applied to routing and scheduling problems, where policies construct assignments between tasks and limited resources through sequential decisions. Nazari et al. (2018) develop an end-to-end policy-gradient framework for vehicle routing, while Kool et al. (2019) use an attention-based model for several routing problems. Recent work further considers parallel multi-agent construction for vehicle routing and scheduling (Berto et al., 2025), as well as vehicle-routing policies trained across different levels of capacity-constraint tightness (Luo et al., 2025). These methods demonstrate the ability of RL to learn problem-specific assignment policies, but the learned policy remains responsible for directly constructing the solution.

RL has also been used to learn dispatching and improvement policies. Learning to Dispatch (Zhang et al., 2020) represents job-shop states as disjunctive graphs and learns priority dispatching rules with a GNN policy. NeuRewriter (Chen & Tian, 2019) learns actor–critic policies for selecting local regions and rewriting rules, with applications to online job scheduling and vehicle routing. Iterative improvement policies have also been developed for routing (Wu et al., 2022; Guan et al., 2025), job-shop scheduling (Zhang et al., 2024), and real-world production scheduling (Müller & Vollenkemper, 2024). These approaches avoid constructing every solution from scratch, but still require RL to represent problem-specific dispatching or improvement actions.

2.2 Hybrid RL–Heuristic Methods for Task–Resource Assignment

RL has been combined with domain-specific heuristics to solve task–resource assignment problems. Ma et al. (2021) use an upper-level agent to partition dynamic pickup-and-delivery problems and combine operations-research methods with RL at the lower level. Pan et al. (2025) use local search to generate comparison pairs for preference-based fine-tuning in routing and scheduling. More directly related to our work, Delayed Matching studies when an upper-layer policy should invoke a downstream assignment procedure. Yang et al. (2020) jointly optimize the matching interval and matching radius in ride-sourcing markets. Qin et al. (2021) use RL to adapt matching intervals to supply–demand states, while Ke et al. (2022) combine multi-agent RL for request-level delay control with bipartite matching. Tang et al. (2025) design an adaptive delayed matching method using a regime-aware MDP and a self-attention MoE encoder. These approaches generally reconsider timing decisions as the system evolves, resulting in long sequences of policy decisions that complicate exploration and credit assignment. Different from above approaches, our work addresses this issue from a delay-benefit perspective. This method mitigates the effects of sparse feedback and therefore reduces the learning burden on RL and enables more efficient policy optimization.

3 Problem Formulation

We consider dynamic task–resource assignment problems in which tasks arrive over time and compete for limited resources whose states may evolve (Ihde et al., 2022; Liu et al., 2023). At decision epoch t, the state s_t contains the pending task set $\mathcal{P}_t$, resource states, and problem-specific constraints. A decision d_t specifies a set of task–resource assignments satisfying

$$d_t \in \mathcal{F}(s_t), \tag{1}$$

where $\mathcal{F}(s_t)$ denotes the feasible decision set under compatibility, availability, deadline, and other problem-specific constraints (Lei et al., 2026).

Although a task may be feasible to assign now, triggering it too early can occupy resources that may be more valuable for future tasks, while delaying it too long can reduce assignment quality or cause feasible assignment opportunities to be missed. Because executed assignments are generally irreversible and future tasks and resource availability are revealed only over time, the system must make sequential assignment decisions that remain feasible while maximizing the expected cumulative discounted reward:

$$\max \ \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, d_t)\right], \tag{2}$$

where $r(s_t, d_t)$ denotes the problem-specific reward obtained by taking decision d_t in state s_t, and γ is the discount factor.

4 Adaptive Task-Triggering Framework

Repeated delay-or-trigger decisions lead to long decision sequences with delayed feedback, making effective exploration and credit assignment difficult. Our core idea is to replace repeated trigger-or-delay decisions with a persistent triggering criterion for each task, which is automatically evaluated as the system state evolves. A straightforward approach lets RL select a level defined by hand-crafted conditions, but a favorable triggering time may not coincide with any predefined level. A continuous weighted score offers finer gradations, yet its factors and combination rule remain hand-designed and may miss implicit opportunities.

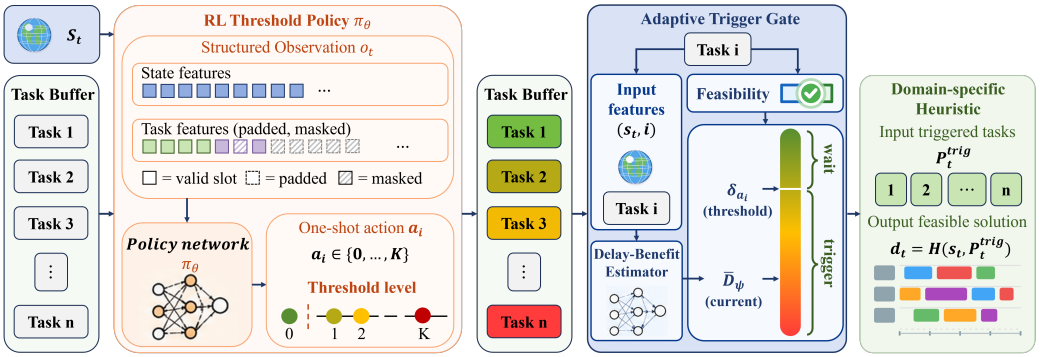

Figure 2: Overview of the proposed adaptive task-triggering framework.

Rather than relying on hand-crafted conditions to decide when a task should trigger, we ask how much benefit could be gained by waiting. We compare the returns of triggering now and at later candidate epochs across multiple environment rollouts, using these comparisons as supervision to train a delay-benefit estimator. RL selects a task-specific threshold from the global task–resource state; as conditions change, the estimator reevaluates the benefit and the task is triggered once its estimate falls below the threshold. This shortens the RL decision sequence and reduces reliance on hand-crafted timing rules, easing exploration and credit assignment while preserving global resource coordination. Figure 2 summarizes the framework.

4.1 MDP Formulation

We formulate triggering control as an MDP in which RL assigns each task a persistent triggering criterion once, rather than repeatedly making delay-or-trigger decisions.

State. The policy receives a global observation o_t derived from s_t, including pending-task features, resource availability and commitments, task–resource competition, and the triggering levels and control status of previously controlled tasks.

Action. When task i enters control, the policy selects

$$a_i \sim \pi_\theta(\cdot \mid o_t, i), \qquad a_i \in \{0, 1, \dots, K\}. \tag{3}$$

Action $a_i = 0$ specifies the earliest feasible triggering epoch, while $a_i = k$, $k = 1, \dots, K$, selects a persistent threshold δ_k. The selected requirement remains associated with the task until triggering.

State transition. At each subsequent epoch, the fixed estimator updates the delay-benefit estimate for task i based on the current state, and the trigger gate checks whether the selected triggering requirement is satisfied. Once the requirement is met, task i is triggered and passed to the domain heuristic for feasible assignment, after which the resulting assignment is incorporated into the next environment state.

Reward. The policy is optimized using the original problem-level reward in Eq. (2), reflecting weighted tardiness in DJSP, interception utility in WTA, and profit in warehouse replenishment. The estimator supplies only the signal used by the trigger gate and introduces no reward shaping.

4.2 Delay-Benefit Estimation

Threshold-based triggering requires a task-level signal that reflects how much long-term benefit can still be gained by waiting. This signal must be forward-looking because triggering quality depends on both the current state and future opportunities. We therefore compare the expected return from triggering the task now with the best expected return attainable by delaying its trigger up to the safe triggering boundary. The resulting gap directly measures the potential benefit forgone by triggering the task at the current epoch.

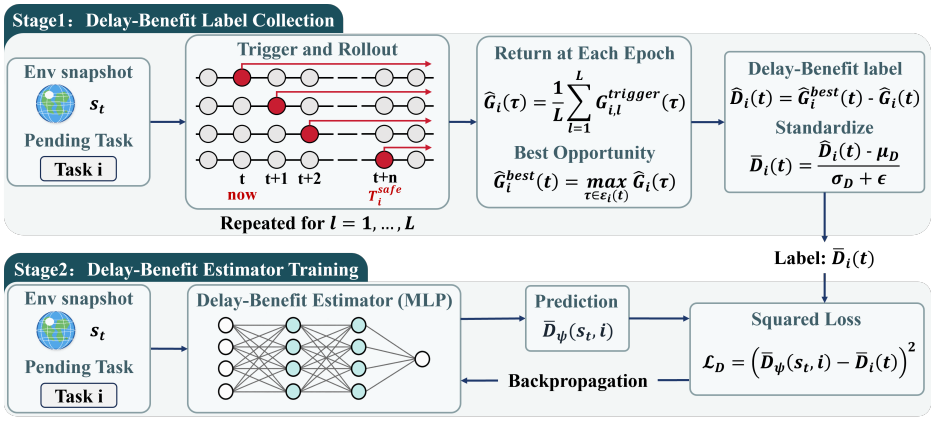

Figure 3: Overview of rollout-based label generation and delay-benefit estimator training.

For each task i, we consider triggering opportunities only within its lifecycle. Let T_i^{safe} denote the latest decision epoch at which task i can remain untriggered without violating its deadline, risk boundary, or other problem-specific safety requirements. For a pending task i at decision epoch t, the remaining candidate trigger epochs are defined as $\mathcal{E}_i(t) = \{\tau \mid t \le \tau \le T_i^{\text{safe}}\}$.

Let $G_i(\tau)$ denote the expected long-term return obtained by keeping task i pending until epoch τ and triggering it at τ. The best remaining triggering opportunity is

$$G_i^{\text{best}}(t) = \max_{\tau \in \mathcal{E}_i(t)} G_i(\tau), \tag{4}$$

and we define the delay benefit as

$$D_i(t) = G_i^{\text{best}}(t) - G_i(t). \tag{5}$$

A large $D_i(t)$ indicates that delaying the task may still provide substantial additional return, whereas a small $D_i(t)$ suggests that triggering now achieves a return close to the best remaining opportunity.

Since $D_i(t)$ is unavailable during online execution, we generate training labels from environment rollouts, as illustrated in Fig. 3. For each sampled environment snapshot s_t and pending task i, we evaluate candidate trigger epochs using L rollouts and estimate $\hat{G}_i(\tau) = \frac{1}{L}\sum_{\ell=1}^{L} G_{i,\ell}^{\text{trigger}}(\tau)$.

The supervision target is then

$$\hat{D}_i(t) = \max_{\tau \in \mathcal{E}_i(t)} \hat{G}_i(\tau) - \hat{G}_i(t). \tag{6}$$

For each candidate epoch, we first estimate its expected return by averaging over multiple rollouts, and then select the epoch with the highest estimated return. This ensures that the resulting label reflects a consistently favorable triggering opportunity rather than an unusually high return from a single stochastic rollout.

We standardize the targets using the training-set statistics as $\bar{D}_i(t) = (\hat{D}_i(t) - \mu_D)/(\sigma_D + \epsilon)$ and train a lightweight MLP $\bar{D}_\psi(s_t, i)$ using the squared loss $\mathcal{L}_D = (\bar{D}_\psi(s_t, i) - \bar{D}_i(t))^2$.

The estimator is fixed during RL training and execution, providing a stable, data-driven delay-benefit signal for threshold construction and triggering control.

4.3 Adaptive Threshold Selection and Triggering

Because the delay benefit captures the potential benefit of waiting for an individual task, it does not by itself account for competition among tasks for shared resources. RL therefore determines how much task-local delay benefit is allowed to remain at triggering, while accounting for system-level resource coordination.

To obtain scale-adaptive candidate tolerances without manually choosing their numerical values, we collect standardized estimator predictions over feasible pending tasks in training snapshots:

$$\mathcal{Z}_D = \left\{\bar{D}_\psi(s_t, i) \mid t \in \mathcal{T}_{\text{ref}},\ i \in \mathcal{P}_t,\ \text{feasible}_i(t) = 1\right\}. \tag{7}$$

The K candidate thresholds are empirical quantiles,

$$\delta_k = Q_{\frac{k}{K+1}}(\mathcal{Z}_D), \qquad k = 1, \ldots, K. \tag{8}$$

Because a smaller delay benefit means less expected return can be gained by waiting, a smaller δ_k imposes a stricter triggering requirement, while a larger threshold tolerates greater opportunity loss. We reserve $a_i = 0$ for triggering at the earliest feasible epoch.

After RL selects a triggering level for task i, the estimator updates its delay-benefit estimate at each subsequent epoch. For $a_i \in \{1, \ldots, K\}$, task i is triggered when

$$\text{trigger}_i(t) = 1 \iff \text{feasible}_i(t) = 1 \wedge \bar{D}_\psi(s_t, i) \leq \delta_{a_i}. \tag{9}$$

The tasks triggered at epoch t form the set $\mathcal{P}_t^{\text{trig}}$, which is then passed to the domain heuristic to construct the assignment decision $d_t = H(s_t, \mathcal{P}_t^{\text{trig}})$.

4.4 Policy optimization.

With the delay-benefit estimator fixed, the policy learns to select each task's triggering tolerance based on the global task–resource state. When a task enters control, π_θ selects a triggering level that determines its tolerance threshold, which remains fixed until the task is triggered.

We optimize π_θ with PPO (Schulman et al., 2017) on trajectories generated by the complete triggering-control process, using the original problem-level reward in Eq. (2). The consequences of each threshold choice are therefore reflected in subsequent rewards through resource consumption, task competition, and future assignment opportunities. The fixed estimator supplies only the task-local signal used by the trigger gate, while RL learns to coordinate task-specific tolerances for long-term system performance.

5 Experiments

5.1 Experiment Setup

We evaluate the proposed framework on three representative dynamic task–resource assignment benchmarks: dynamic job-shop scheduling (DJSP), dynamic weapon–target assignment (WTA), and dynamic warehouse replenishment.

For DJSP, we follow the DRSS-style instance generator (Marques, 2024), configured with 10 machines. Jobs arrive dynamically, and their operations must satisfy standard precedence and operation–machine constraints. Machines may fail randomly and remain unavailable for one or two time steps. The objective is to minimize total weighted tardiness. Difficulty is varied by the number of jobs, the mean inter-arrival time, and due-date tightness (DDT).

For WTA, we follow the WTA settings described in Li et al. (2024), and construct an online environment in which targets must be intercepted under ammunition–target compatibility, engagement-range, and inventory constraints. The objective is to maximize intercepted target value minus engagement costs. Difficulty is controlled by the number of targets, mean inter-arrival time, and ATK Type, which denotes the number of ammunition types capable of attacking each target.

The warehouse replenishment environment is inspired by the multi-product, multi-node inventory setting introduced by Sultana et al. (2020). It contains three product types and four heterogeneous suppliers whose states evolve over time and differ in delivery lead times and supply characteristics. Orders arrive online, and the objective is to maximize total profit. Difficulty is varied by the number of orders, supplier heterogeneity, and fulfillment time-window tightness.

For all benchmarks, the delay-benefit estimator is pretrained using environment rollouts and remains fixed during PPO training. The adaptive triggering thresholds are constructed from empirical quantiles of its standardized predictions at decision epochs in the training set. All methods are evaluated on the same test distributions, and the reported results are averaged over repeated runs. Detailed variance statistics are provided in the appendix.

5.2 BASELINES

The baselines are selected to compare different decision mechanisms and control granularities for dynamic task–resource assignment, including direct solution construction, repeated timing control, and heuristic-guided decision making. **RL** is an end-to-end PPO policy that directly produces scheduling or matching decisions without a lower-level heuristic. **Global Delayed Matching** uses PPO to repeatedly select a global delay-or-trigger action at each decision epoch. The trigger action simultaneously triggers all pending tasks, which are then assigned by the domain heuristic, while the delay action keeps them waiting. **MoE Delayed Matching** is a generic adaptation of RAST-MoE-RL (Tang et al., 2025) to our settings. It introduces an MoE module into the policy network and replaces the original region-level match-or-hold decisions with task-level delay-or-trigger control. The triggered tasks are then assigned by the same domain heuristic. **Hyper-Heuristic**, following the general hyper-heuristic paradigm (Burke et al., 2013), uses PPO to select one of several domain heuristics at each decision epoch, after which the selected heuristic constructs the decision.

For DJSP, we additionally include **Dynamic L2D**, a dynamic adaptation of Learning to Dispatch (Zhang et al., 2020) that retains its disjunctive-graph representation and GNN encoder. We also report an offline **MILP** reference for small instances. Since stochastic events such as machine failures make the original dynamic process unsuitable for direct exact optimization, we generate 10 deterministic failure trajectories using different random seeds, solve the resulting instances to optimality, and report the average optimum. Larger instances are omitted because of excessive solution time. For WTA, **Random**, a commonly used reference method for this problem, first identifies feasible assignments using domain rules and then samples randomly among them. For warehouse replenishment, the classical (s, S) **Heuristic** (Scarf, 1960) places an order whenever inventory falls below the reorder point s and replenishes it to the target level S.

5.3 MAIN RESULTS

The performance of our method on DJSP is shown in Table 1, where *IAT* denotes the expected inter-arrival time between consecutive tasks. In addition, we abbreviate *Global Delayed Matching* and *MoE Delayed Matching* as *Global Delay* and *MoE Delay*.

Table 1: Representative main results on DJSP. Lower is better.

Jobs	IAT	DDT		RL	Global Delay	MoE Delay	Dynamic L2D	Hyper Heuristic	Ours	MILP
50	5	1.5	Obj	280.81	258.31	250.84	243.18	257.56	**219.59**	168.34
			Gap	66.81%	53.45%	49.01%	44.46%	53.00%	**30.44%**	–
		2.0	Obj	127.90	120.72	114.92	110.04	120.07	**97.43**	78.06
			Gap	63.85%	54.65%	47.22%	40.97%	53.82%	**24.81%**	–
	10	1.5	Obj	29.65	30.20	26.95	25.51	26.97	**22.09**	15.12
			Gap	96.10%	99.74%	78.24%	68.72%	78.37%	**46.10%**	–
		2.0	Obj	7.89	8.86	6.78	6.94	6.97	**5.51**	3.59
			Gap	119.78%	146.80%	88.86%	93.31%	94.15%	**53.48%**	–
100	5	1.5	Obj	732.37	698.20	681.57	670.76	679.76	**649.28**	540.05
			Gap	35.61%	29.28%	26.20%	24.20%	25.87%	**20.23%**	–
		2.0	Obj	450.78	449.26	413.78	392.10	414.17	**376.80**	306.56
			Gap	47.04%	46.55%	34.98%	27.90%	35.10%	**22.91%**	–
	10	1.5	Obj	47.84	49.79	45.74	42.09	46.44	**38.59**	30.88
			Gap	54.92%	61.24%	48.12%	36.30%	50.39%	**24.97%**	–
		2.0	Obj	10.94	10.61	8.94	8.56	9.01	**7.20**	4.75
			Gap	130.32%	123.37%	88.21%	80.21%	89.68%	**51.58%**	–

We can see that for DJSP, our method achieves the lowest total weighted tardiness across all settings. End-to-end RL must learn constrained dispatching directly, whereas Global Delayed Matching provides only system-wide timing control. Dynamic L2D and MoE Delayed Matching offer stronger

alternatives through structured representation and adaptive timing control, but still rely on direct dispatching and repeated delay-or-trigger decisions, respectively. Our one-shot threshold selection avoids repeated timing decisions while preserving task-level control.

Table 2: Representative main results on WTA. Higher is better.

Targets	IAT	ATK Type	Random	RL	Global Delay	MoE Delay	Hyper Heuristic	Ours
150	5	2	319.90	372.73	428.83	456.53	460.26	**498.15**
		4	365.53	438.08	506.80	551.54	541.55	**581.43**
	10	2	431.74	444.63	488.35	503.75	488.83	**519.55**
		4	454.29	514.05	573.55	596.86	586.39	**614.11**
200	5	2	398.75	554.82	599.78	635.48	632.08	**665.18**
		4	481.70	647.39	679.27	749.57	745.95	**779.15**
	10	2	569.48	605.23	651.70	674.95	653.94	**696.62**
		4	612.62	676.22	729.98	773.52	767.31	**817.07**

The results of our method on WTA is shown in Table 2. Our method achieves the highest engagement utility across target scales, arrival settings, and compatibility levels. All methods improve as ATK Type increases because each target has more feasible ammunition choices. Our method retains its advantage under both restricted and flexible compatibility, indicating effective coordination of future engagement opportunities.

Table 3: Representative main results on warehouse replenishment. Higher is better.

Orders	Supplier Heterogeneity	Time Window	(s, S) Heuristic	RL	Global Delay	MoE Delay	Hyper Heuristic	Ours
500	Low	Tight	50.63	50.28	52.33	58.91	55.50	**63.01**
		Loose	58.84	60.34	73.84	80.97	77.71	**84.03**
	High	Tight	68.87	62.08	66.52	75.74	66.84	**92.09**
		Loose	94.45	78.59	83.73	100.75	96.80	**116.38**
1000	Low	Tight	93.68	79.72	86.81	95.52	94.38	**132.03**
		Loose	123.14	120.05	136.35	152.78	148.84	**157.64**
	High	Tight	130.78	119.54	127.88	137.90	128.60	**155.58**
		Loose	196.43	167.46	188.43	203.55	201.02	**234.16**

The results of our method on warehouse replenishment is shown in Table 3. Our method achieves the highest profit across all settings, including different order volumes, supplier heterogeneity levels, and fulfillment time-window settings. Unlike the reactive (s, S) heuristic and repeated-delay baselines, task-specific thresholds coordinate immediate fulfillment with future procurement opportunities.

5.4 Ablation Study

We conduct ablations to examine three questions: whether persistent triggering requirements are preferable to repeated binary decisions, whether rollout-based delay benefits provide a more informative triggering signal than hand-crafted alternatives, and whether task-specific threshold selection is necessary. Table 4 reports results on representative settings from the three benchmarks.

Earliest-Feasible Heuristic directly passes every feasible pending task to the same domain heuristic used by our method, removing learned triggering control. **Repeated Delay-or-Trigger** uses PPO to reconsider a binary delay-or-trigger action for each pending task at every decision epoch. **Hand-Crafted Discrete Levels** assigns each task one of the 2^3 subsets of three domain-specific conditions, including the empty subset for earliest-feasible triggering, and triggers the task once all selected conditions are satisfied. The condition thresholds are estimated from the training set. The three

factors are hit quality, cost advantage, and ammunition abundance for WTA; same-product demand aggregation, procurement-cost advantage, and delivery efficiency for warehouse replenishment; and designated-machine availability, inverse queued workload, and remaining-slack urgency for DJSP.

Tuned Continuous Score combines the same three normalized factors as

$$s_i(t) = \sum_{r=1}^{3} w_r z_i^r(t), \qquad w_r \in \{0, 0.1, \ldots, 1\}, \qquad \sum_{r=1}^{3} w_r = 1. \tag{10}$$

We enumerate all 66 weight combinations and select the best one on the validation set. Its K thresholds are constructed from descending training-set quantiles so that smaller action indices consistently represent stricter triggering requirements, and RL selects one persistent level for each task. **Ours (Fixed Global Threshold)** retains the learned delay-benefit estimator but applies one threshold to every task, selected on the validation set from the same K candidates used by the full method. It therefore removes only task-specific threshold selection.

Table 4: Ablation results on representative DJSP (100 jobs, IAT 5, DDT 2.0), WTA (150 targets, IAT 5, ATK Type 2), and warehouse replenishment (500 orders, high supplier heterogeneity, tight time windows) settings. Results are reported as mean and standard deviation over repeated runs.

Ablation Variant	DJSP ↓	WTA ↑	Warehouse ↑
Earliest-Feasible Heuristic	433.38±19.18	423.55±16.85	66.32±2.67
Repeated Delay-or-Trigger	417.15±39.02	446.24±16.16	72.11±3.88
Hand-Crafted Discrete Levels	418.13±22.27	459.15±13.94	81.08±3.57
Tuned Continuous Score	396.71±27.87	477.38±19.14	87.24±2.94
Ours (Fixed Global Threshold)	423.92±23.10	434.61±10.17	73.13±2.45
Ours	**376.80±30.16**	**498.15±15.20**	**92.09±3.64**

Repeated Delay-or-Trigger improves over earliest-feasible triggering by 3.7%, 5.4%, and 8.7% on DJSP, WTA, and warehouse replenishment, respectively, confirming the value of learned timing control. Hand-crafted discrete levels further improve WTA and warehouse replenishment but provide no clear gain on DJSP, indicating that persistent control alone does not ensure an informative triggering signal. Replacing discrete conditions with the tuned continuous score improves the three objectives by 5.1%, 4.0%, and 7.6%, respectively, but its manually selected factors still provide only an indirect description of future opportunities.

Our complete method achieves the best mean performance on all three benchmarks, improving over the tuned continuous score by 5.0%, 4.4%, and 5.6%, respectively. These results support using rollout-estimated delay benefits as the triggering signal rather than hand-crafted state scores. Applying a single global threshold reduces performance by 11.1%, 14.6%, and 25.9% relative to the full method. Although the estimator still produces task- and state-dependent delay benefits, a shared threshold imposes the same tolerance for forgone delay benefit on every task, causing excessive waiting for some tasks and premature triggering for others. The results confirm the complementary roles of supervised task-local opportunity estimation and RL-based global coordination.

Additional analyses. We further evaluate the delay-benefit estimator on offline test snapshots and closed-loop trajectories under the learned policy. It maintains high agreement with rollout-based targets and low prediction error under the policy-induced distribution shift. We also assess sensitivity to the number of triggering levels K. Performance improves substantially up to $K = 10$ and then largely levels off, while larger action spaces require more training without consistent gains. Detailed results are provided in the appendix.

6 Conclusions

We reformulated repeated task-level triggering decisions into supervised delay-benefit estimation and RL-based threshold selection. This decomposition gives RL a compact interface for coordinating competing tasks while leaving feasible assignment construction to domain heuristics. Experiments on three dynamic task–resource assignment benchmarks demonstrate consistent improvements over pure RL, heuristic, and RL–heuristic timing-control baselines.

AI USE STATEMENT

We used generative AI tools to assist with language polishing and improving the clarity and readability of the manuscript, and to support literature retrieval and discovery during the related-work review. All AI-assisted text was reviewed and revised by the authors, and all retrieved references and their relevance to this work were independently verified against the original sources. Generative AI tools were not used to generate experimental data or results. The authors take full responsibility for the final content of the paper, including all statements, interpretations, citations, and conclusions.

ETHICS STATEMENT

This work is a methodological study evaluated exclusively on synthetically generated instances in simulated task–resource assignment environments. It involves no human participants, personal data, or real-world operational systems. The study does not involve operational decision-making, field testing, or deployment; its results characterize algorithmic performance only under the stated simulation settings.

REPRODUCIBILITY STATEMENT

The main text specifies the problem formulation, rollout-based label construction, estimator training, threshold selection, and policy optimization. The appendix documents benchmark generation, evaluation settings, baseline implementations, and hyperparameters. We also submit the code files as supplementary material to support reproduction of the experiments.

REFERENCES

Federico Berto, Chuanbo Hua, Laurin Luttmann, Jiwoo Son, Junyoung Park, Kyuree Ahn, Changhyun Kwon, Lin Xie, and Jinkyoo Park. Parco: Parallel autoregressive models for multi-agent combinatorial optimization. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 167130–167160. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f404c0259def290306159047b0b52584-Paper-Conference.pdf.

Edmund Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724, 07 2013. doi: 10.1057/jors.2013.71.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/131f383b434fdf48079bff1e44e2d9a5-Paper.pdf.

Qingshu Guan, Hui Cao, Lixin Jia, Dapeng Yan, and Badong Chen. Synergetic attention-driven transformer: A deep reinforcement learning approach for vehicle routing problems. *Expert Systems with Applications*, 274:126961, 2025. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2025.126961. URL https://www.sciencedirect.com/science/article/pii/S0957417425005834.

Sven Ihde, Luise Pufahl, Maximilian Völker, Asvin Goel, and Mathias Weske. A framework for modeling and executing task-specific resource allocations in business processes. *Computing*, 104 (11):2405–2429, November 2022. ISSN 0010-485X. doi: 10.1007/s00607-022-01093-2. URL https://doi.org/10.1007/s00607-022-01093-2.

Jintao Ke, Feng Xiao, Hai Yang, and Jieping Ye. Learning to delay in ride-sourcing systems: A multi-agent deep reinforcement learning framework. *IEEE Transactions on Knowledge and Data Engineering*, 34(5):2280–2292, 2022. doi: 10.1109/TKDE.2020.3006084.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Ming Lei, You-Chen Fan, Xi Xiao, Liu Yi, Qi-Chen Wang, You Fu, and Sheng-Liang Fang. Amcats: An adaptive rule-guided framework for dynamic multi-task allocation in heterogeneous satellite systems. *Journal of King Saud University Computer and Information Sciences*, 38, 2026. URL https://api.semanticscholar.org/CorpusID:291169916.

Jinrui Li, Guohua Wu, and Ling Wang. A comprehensive survey of weapon target assignment problem: Model, algorithm, and application. *Engineering Applications of Artificial Intelligence*, 137:109212, 2024. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2024.109212. URL https://www.sciencedirect.com/science/article/pii/S0952197624013708.

Renke Liu, Rajesh Piplani, and Carlos Toro. A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem. *Computers Operations Research*, 159:106294, 2023. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2023.106294. URL https://www.sciencedirect.com/science/article/pii/S0305054823001582.

Fu Luo, Yaoxin Wu, Zhi Zheng, and Zhenkun Wang. Rethinking neural combinatorial optimization for vehicle routing problems with different constraint tightness degrees. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 127283–127317. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/b8fc784104e74f1b427865d30ad22931-Paper-Conference.pdf.

Yi Ma, Xiaotian Hao, Jianye Hao, Jiawen Lu, Xing Liu, Tong Xialiang, Mingxuan Yuan, Zhigang Li, Jie Tang, and Zhaopeng Meng. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 23609–23620. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/c6a01432c8138d46ba39957a8250e027-Paper.pdf.

Nuno Marques. drss: A dynamic rescheduling / dynamic job shop scheduling framework. https://github.com/nuno-a-marques/drss, 2024. GitHub repository, accessed 2026-06-15.

Arthur Müller and Lukas Vollenkemper. Reinforcement learning as an improvement heuristic for real-world production scheduling. In *2024 International Conference on Machine Learning and Applications (ICMLA)*, pp. 296–303, 2024. doi: 10.1109/ICMLA61862.2024.00046.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement learning for solving the vehicle routing problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf.

Mingjun Pan, Guanquan Lin, You-Wei Luo, Bin Zhu, Zhien Dai, Lijun Sun, and Chun Yuan. Preference optimization for combinatorial optimization problems. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47672–47696. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/pan25e.html.

Guoyang Qin, Qi Luo, Yafeng Yin, Jian Sun, and Jieping Ye. Optimizing matching time intervals for ride-hailing services using reinforcement learning. *Transportation Research Part C: Emerging Technologies*, 129:103239, 2021. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2021.103239. URL https://www.sciencedirect.com/science/article/pii/S0968090X21002527.

Herbert E. Scarf. The optimality of (S, s) policies in the dynamic inventory problem. In Kenneth J. Arrow, Samuel Karlin, and Patrick Suppes (eds.), *Mathematical Methods in the Social Sciences, 1959*, pp. 196–202. Stanford University Press, Stanford, CA, 1960.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Nazneen Sultana, Hardik Meisheri, Vinita Baniwal, Somjit Nath, Balaraman Ravindran, and Harshad Khadilkar. Reinforcement learning for multi-product multi-node inventory management in supply chains, 06 2020.

Yuhan Tang, Kangxin Cui, Jung Ho Park, Yibo Zhao, Xuan Jiang, Haoze He, Dingyi Zhuang, Shenhao Wang, Jiangbo Yu, Harilaos N. Koutsopoulos, and Jinhua Zhao. Rast-moe-rl: A regime-aware spatio-temporal moe framework for deep reinforcement learning in ride-hailing. *CoRR*, abs/2512.13727, 2025. doi: 10.48550/ARXIV.2512.13727. URL https://doi.org/10.48550/arXiv.2512.13727.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069, September 2022. ISSN 2162-237X. doi: 10.1109/TNNLS.2021.3068828.

Hai Yang, Xiaoran Qin, Jintao Ke, and Jieping Ye. Optimizing matching time interval and matching radius in on-demand ride-sourcing markets. *Transportation Research Part B: Methodological*, 131:84–105, 2020. ISSN 0191-2615. doi: https://doi.org/10.1016/j.trb.2019.11.005. URL https://www.sciencedirect.com/science/article/pii/S0191261518311731.

Jingfeng Yang, Zhiqin Zhang, and Hoong Chuin Lau. Learning to optimize the dispatch time interval for on-demand food delivery service. In *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1182–1189, 2024. doi: 10.1109/ITSC58415.2024.10919669.

Yunhui Zeng, Zijun Liao, Yuanzhi Dai, Rong Wang, Xiu Li, and Bo Yuan. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism, 2022. URL https://arxiv.org/abs/2201.00548.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1621–1632. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/11958dfee29b6709f48a9ba0387a2431-Paper.pdf.

Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. In *International Conference on Learning Representations*, 05 2024.

A Appendix

A.1 Detailed Experimental Settings

We provide further details of the three dynamic task–resource assignment environments. Within each benchmark, all compared methods use the same instance generation rules, state transitions, feasibility constraints, and objective function. *IAT* denotes the expected time between consecutive task arrivals. In our framework, triggering determines when a pending task is passed to the domain heuristic; the heuristic then constructs feasible task–resource assignments under the current environment state.

A.1.1 Dynamic Job-Shop Scheduling

We construct the dynamic job-shop scheduling (DJSP) benchmark following the DRSS-style instance generation protocol of Marques (2024). The shop contains 10 machines. Jobs arrive online, and each job consists of an ordered sequence of operations. Each operation requires one designated machine and can begin only after its preceding operation is completed and the machine is available. We treat a schedulable operation as a task and its designated machine as the resource.

We use the Poisson-style arrival mode of the instance generator. The number of operations per job is sampled from $[3, 6]$, and operation processing times are sampled from $[1, 20]$. Each machine independently fails with probability 0.01 at each decision epoch; a failure makes it unavailable for one or two epochs. Jobs that have not yet arrived are unknown to the decision maker, and machine failures may change the feasibility of subsequent scheduling decisions.

For job i, the due date is generated as $d_i = r_i + \mathrm{DDT} \sum_{j=1}^{|J_i|} p_{ij} + \epsilon_i$, where r_i is its release time, J_i is its operation sequence, p_{ij} is the processing time of operation j, and ϵ_i is a small random slack term. The objective is to minimize total weighted tardiness, $\sum_i w_i \max(C_i - d_i, 0)$, where C_i and w_i are the job completion time and priority weight. We vary the number of jobs, IAT, and DDT; a smaller DDT produces tighter due dates.

A.1.2 Weapon–Target Assignment

In the weapon–target assignment (WTA) environment, targets appear online and are engaged using limited weapon and ammunition resources. We use 4 weapon types, 6 ammunition types, and 3 target types. A target can be engaged only by a compatible ammunition type, within the applicable engagement range, and when sufficient ammunition inventory remains. We treat each target as a task and a feasible weapon–ammunition option as its resource.

For each target, the generator samples its type, speed, height, approach angle, and compatible ammunition types. The *ATK Type* setting specifies the exact number of ammunition types compatible with that target. Minimum and maximum engagement ranges are generated for compatible options. Target positions evolve as time passes, so the feasible engagement options and their utility can change between decision epochs.

The objective is to maximize total engagement utility, accounting for the value of neutralized targets and ammunition costs. We vary the number of targets, IAT, and ATK Type. Increasing ATK Type gives each target more potential ammunition choices and enlarges the feasible engagement space.

A.1.3 Dynamic Warehouse Replenishment

The warehouse replenishment environment is inspired by the multi-product inventory setting of Sultana et al. (2020). It contains 3 product types and 4 suppliers. Customer orders arrive online; each order specifies a product, demand quantity, order value, and delivery deadline. We treat an order as a task and a supplier capable of meeting its delivery window as a feasible resource.

Orders are generated according to a Poisson arrival process with IAT fixed at 1. Each order's deadline is obtained by adding a sampled delivery-window length to its creation time. The time-window setting controls how restrictive these deadlines are. Suppliers have no supply-capacity limit, but differ in product prices, shipping fees, and delivery lead times, which determine procurement cost and whether an order can be delivered on time.

We consider low and high supplier heterogeneity. In the low-heterogeneity setting, the fast and slow supplier profiles have lead times of 2 and 5 and base price parameters of 12.0 and 11.0, respectively. In the high-heterogeneity setting, their lead times are 1 and 6 and their base price parameters are 14.0 and 9.0. Thus, faster delivery generally comes at a higher procurement price. The objective is to maximize total profit, defined as sales revenue minus procurement costs. We vary order volume, supplier heterogeneity, and delivery time-window tightness.

A.2 Baseline and Ablation Implementation Details

All compared methods are evaluated under the same instance generation rules, dynamic state transitions, feasibility constraints, and objective functions described above. For methods that use a downstream domain heuristic, the same benchmark-specific heuristic is used as in our framework, so that their differences arise from the triggering or control mechanism rather than from assignment construction. Unless otherwise specified, learning-based baselines are trained in the same environment distributions as our method.

A.2.1 Main Baselines

RL. The RL baseline directly learns scheduling or assignment decisions without a downstream heuristic. We use PPO as the policy optimization algorithm. In DJSP, the policy directly selects dispatching decisions among currently feasible operations; in WTA and warehouse replenishment, it directly selects feasible task–resource assignments under the corresponding environment constraints. This baseline evaluates end-to-end learning in the original constrained decision space without decomposing triggering from assignment construction.

Global Delayed Matching. Global Delayed Matching provides system-level timing control. At each decision epoch, PPO selects a binary delay-or-trigger action for the set of pending tasks. If delay is selected, all pending tasks remain waiting until the next decision epoch. If trigger is selected, all currently feasible pending tasks are simultaneously passed to the benchmark-specific domain heuristic, which constructs the corresponding feasible assignments. Thus, the policy controls when the heuristic is invoked but does not distinguish triggering requirements among individual tasks.

MoE Delayed Matching. MoE Delayed Matching is our generic adaptation of RAST-MoE-RL (Tang et al., 2025) to the three task–resource assignment benchmarks. The original method uses a regime-aware spatio-temporal mixture-of-experts representation for delayed matching in ride-hailing. We retain the MoE-enhanced PPO policy while replacing its original region-level match-or-hold decisions with task-level delay-or-trigger decisions. At each decision epoch, the policy independently determines whether each pending task should remain waiting or be triggered. Triggered tasks are then passed to the same benchmark-specific domain heuristic used by our framework, whereas delayed tasks are reconsidered at subsequent decision epochs. Unlike our persistent triggering-level formulation, this baseline repeatedly makes binary timing decisions throughout the lifetime of each task.

Hyper-Heuristic. The Hyper-Heuristic baseline follows the general learning-based hyper-heuristic paradigm (Burke et al., 2013). At each decision epoch, PPO selects one rule from a benchmark-specific set of domain heuristics according to the current environment state, and the selected heuristic constructs the resulting scheduling or assignment decision. The policy therefore adapts which heuristic is applied over time rather than learning task-specific triggering requirements.

Dynamic L2D. For DJSP, we additionally include Dynamic L2D, our adaptation of Learning to Dispatch (L2D) (Zhang et al., 2020) to the dynamic environment. We retain the disjunctive-graph representation and GNN-based policy architecture of L2D and train the policy with PPO for online dispatching. Newly arriving jobs are incorporated into the current scheduling state, while stochastic machine failures dynamically modify machine availability. The policy selects among currently schedulable operations at each decision epoch.

Random. For WTA, Random provides a weak domain-specific reference. At each decision epoch, feasible engagement actions are first identified according to ammunition–target compati-

bility, engagement-range, and remaining-inventory constraints, after which an action is sampled uniformly from the feasible set.

(s, S) Heuristic. For warehouse replenishment, we include the classical (s, S) inventory-control heuristic (Scarf, 1960). For each product, replenishment is triggered when its inventory level falls below the reorder threshold s, and an order is placed to restore inventory toward the target level S. This provides a standard reactive threshold-based inventory-control reference for comparison with learned dynamic triggering policies.

MILP Reference for DJSP. For tractable DJSP instances, we additionally report an offline MILP reference. Since stochastic machine failures alter future machine availability, directly solving the original stochastic online problem to optimality is impractical. We therefore use 10 fixed random seeds to generate 10 deterministic machine-failure trajectories. Each resulting deterministic instance is solved independently to optimality with MILP, and the reported reference value is the average of the 10 trajectory-wise optimal objective values. MILP results are reported only for instance sizes for which exact optimization remains computationally tractable.

A.2.2 Ablation Variants

Earliest-Feasible Heuristic. This variant removes learned triggering control. Every pending task is passed to the corresponding domain heuristic as soon as it becomes feasible, and the heuristic constructs the task–resource assignment under the current state. It therefore represents the $a_i = 0$ behavior without any learned waiting mechanism.

Repeated Delay-or-Trigger. This variant retains task-level learned timing control but removes persistent triggering levels. At every decision epoch, PPO selects a binary delay-or-trigger action for each pending task. A delayed task remains pending and receives another binary decision at the next epoch, whereas a triggered task is passed to the same domain heuristic used by our method. This baseline isolates the effect of replacing repeated timing decisions with a one-shot persistent triggering requirement.

Hand-Crafted Discrete Levels. This variant replaces the learned delay-benefit signal with three manually designed domain-specific triggering factors. For each task, PPO selects one of the 2^3 subsets of these factors when the task enters control, and the selected subset remains fixed throughout its lifetime. The task is triggered once all conditions in the selected subset are satisfied; the empty subset corresponds to earliest-feasible triggering. The condition thresholds are estimated from the training data. The three factors are designated-machine availability, inverse queued workload, and remaining-slack urgency for DJSP; hit quality, cost advantage, and ammunition abundance for WTA; and same-product demand aggregation, procurement-cost advantage, and delivery efficiency for warehouse replenishment.

Tuned Continuous Score. This variant combines the same three domain-specific normalized factors into a scalar triggering score,

$$s_i(t) = \sum_{r=1}^{3} w_r z_i^r(t), \qquad w_r \in \{0, 0.1, \ldots, 1\}, \qquad \sum_{r=1}^{3} w_r = 1. \tag{11}$$

We enumerate all 66 feasible weight combinations and select the best combination on the validation set separately for each benchmark. Candidate triggering levels are then constructed from empirical quantiles of the resulting training-set scores. Because the score increases as the hand-crafted indicators favor triggering, the quantile ordering is reversed relative to the delay-benefit estimator so that action indices retain consistent triggering semantics. PPO selects one persistent level for each task when it enters control.

Ours (Fixed Global Threshold). This variant retains the pretrained delay-benefit estimator and the same trigger gate as the full method but removes task-specific threshold selection by RL. A single threshold is shared by all tasks and is selected on the validation set from the same K candidate thresholds used by our full method. The estimator is still reevaluated at subsequent decision epochs,

and a feasible task is triggered once its estimated delay benefit falls below the shared threshold. This variant isolates the contribution of task-specific threshold selection and global RL coordination.

A.3 Complete Main Results

We report the complete experimental results for all evaluated configurations in Tables 5–7, Table 8, and Table 9. These tables complement the representative results reported in the main text by covering the full range of problem scales and difficulty settings. For all learning-based methods, we additionally report the standard deviation across evaluation runs.

Complete DJSP Results For DJSP, Tables 5–7 present results for Jobs$= \{50, 100, 200\}$ under all combinations of IAT and DDT. Our method achieves the lowest objective value among all online baselines in every evaluated configuration. For Jobs$= 50$ and Jobs$= 100$, we additionally report the offline MILP reference and the corresponding optimality gaps where the MILP objective is nonzero. As discussed in the experimental setup, this reference is computed with the complete deterministic machine-failure trajectory fixed in advance and therefore serves as an offline reference rather than an online competitor. MILP results are not reported for Jobs$= 200$ because exact optimization becomes computationally intractable at this scale.

Complete WTA Results Table 8 reports the complete WTA results over Targets$= \{100, 150, 200\}$, IAT$= \{5, 10\}$, and ATK Type$= \{2, 3, 4\}$. Our method consistently obtains the highest objective value across all evaluated configurations. The improvement is preserved across different target arrival rates, compatibility levels, and problem scales, indicating that the learned triggering mechanism remains effective under substantially different engagement conditions.

Complete Warehouse Results Table 9 gives the complete warehouse replenishment results for Orders$= \{200, 500, 1000\}$, both supplier-heterogeneity settings, and Tight, Medium, and Loose delivery windows. In the table header, "Supplier Hetero." is used as an abbreviation for supplier heterogeneity. Our method again achieves the highest objective value in every configuration. The gains remain consistent as the number of orders increases and across changes in supplier heterogeneity and delivery-window tightness.

Table 5: Complete DJSP results for Jobs$= 50$. Lower is better.

Jobs	IAT	DDT		RL	Global Delay	MoE Delay	Dynamic L2D	Hyper Heuristic	Ours	MILP
50	5	1.5	Obj	280.81	258.31	250.84	243.18	257.56	**219.59**	168.34
			Std	23.51	24.03	19.55	23.03	15.91	10.67	–
			Gap	66.81%	53.45%	49.01%	44.46%	53.00%	**30.44%**	–
		2.0	Obj	127.90	120.72	114.92	110.04	120.07	**97.43**	78.06
			Std	10.99	12.78	13.97	9.48	8.92	6.79	–
			Gap	63.85%	54.65%	47.22%	40.97%	53.82%	**24.81%**	–
		2.5	Obj	67.66	61.42	55.44	50.00	58.15	**47.46**	35.65
			Std	5.42	4.24	5.93	5.29	3.90	4.83	–
			Gap	89.79%	72.29%	55.51%	40.25%	63.11%	**33.13%**	–
	10	1.5	Obj	29.65	30.20	26.95	25.51	26.97	**22.09**	15.12
			Std	4.91	5.59	5.24	4.86	3.45	3.63	–
			Gap	96.10%	99.74%	78.24%	68.72%	78.37%	**46.10%**	–
		2.0	Obj	7.89	8.86	6.78	6.94	6.97	**5.51**	3.59
			Std	1.52	1.16	1.10	1.54	0.23	0.68	–
			Gap	119.78%	146.80%	88.86%	93.31%	94.15%	**53.48%**	–
		2.5	Obj	2.64	2.97	1.48	1.88	1.63	**0.00**	0.00
			Std	0.53	0.17	0.10	0.31	0.07	–	–
			Gap	–	–	–	–	–	–	–

Table 6: Complete DJSP results for Jobs= 100. Lower is better.

Jobs	IAT	DDT		RL	Global Delay	MoE Delay	Dynamic L2D	Hyper Heuristic	Ours	MILP
100	5	1.5	Obj	732.37	698.20	681.57	670.76	679.76	**649.28**	540.05
			Std	64.09	54.82	55.17	50.39	32.37	28.59	–
			Gap	35.61%	29.28%	26.20%	24.20%	25.87%	**20.23%**	–
		2.0	Obj	450.78	449.26	413.78	392.10	414.17	**376.80**	306.56
			Std	41.06	38.98	42.31	33.84	19.26	30.16	–
			Gap	47.04%	46.55%	34.98%	27.90%	35.10%	**22.91%**	–
		2.5	Obj	269.78	242.16	235.34	219.05	233.36	**203.60**	163.35
			Std	30.99	22.18	30.01	24.84	12.74	25.64	–
			Gap	65.15%	48.24%	44.07%	34.10%	42.86%	**24.64%**	–
	10	1.5	Obj	47.84	49.79	45.74	42.09	46.44	**38.59**	30.88
			Std	6.30	6.48	6.92	5.96	5.69	4.01	–
			Gap	54.92%	61.24%	48.12%	36.30%	50.39%	**24.97%**	–
		2.0	Obj	10.94	10.61	8.94	8.56	9.01	**7.20**	4.75
			Std	1.90	2.16	1.94	1.85	0.97	0.97	–
			Gap	130.32%	123.37%	88.21%	80.21%	89.68%	**51.58%**	–
		2.5	Obj	3.97	3.01	2.24	2.06	2.49	**1.08**	0.00
			Std	0.83	0.23	0.27	0.24	0.10	0.14	–
			Gap	–	–	–	–	–	–	–

Table 7: Complete DJSP results for Jobs= 200. Lower is better.

Jobs	IAT	DDT		RL	Global Delay	MoE Delay	Dynamic L2D	Hyper Heuristic	Ours	MILP
200	5	1.5	Obj	2416.97	2451.71	2383.25	2305.60	2316.62	**2261.18**	–
			Std	160.92	98.06	105.14	70.92	63.15	72.65	–
			Gap	–	–	–	–	–	–	–
		2.0	Obj	1796.72	1788.97	1709.81	1685.11	1754.37	**1653.56**	–
			Std	109.44	69.33	75.08	62.92	51.39	64.96	–
			Gap	–	–	–	–	–	–	–
		2.5	Obj	1231.04	1224.01	1169.66	1129.28	1217.15	**1065.39**	–
			Std	98.62	58.13	65.71	57.18	41.41	53.47	–
			Gap	–	–	–	–	–	–	–
	10	1.5	Obj	145.58	143.79	137.40	133.90	141.97	**118.41**	–
			Std	15.08	10.68	13.07	11.98	8.69	8.98	–
			Gap	–	–	–	–	–	–	–
		2.0	Obj	47.16	49.77	40.05	36.21	40.64	**21.54**	–
			Std	10.11	6.54	9.29	8.52	5.81	3.22	–
			Gap	–	–	–	–	–	–	–
		2.5	Obj	19.94	17.25	15.99	12.93	16.63	**9.55**	–
			Std	6.77	3.70	5.82	5.07	2.16	1.23	–
			Gap	–	–	–	–	–	–	–

Table 8: Complete main results on WTA. Higher is better.

Targets	IAT	ATK Type		Random	RL	Global Delay	MoE Delay	Hyper Heuristic	Ours
100	5	2	Obj	222.47	212.48	251.64	279.59	290.08	**321.15**
			Std	36.11	45.97	34.62	46.30	24.15	16.34
		3	Obj	247.46	271.78	308.06	344.79	342.44	**368.10**
			Std	31.51	29.45	27.79	37.38	15.76	11.51
		4	Obj	253.56	295.07	341.72	368.18	368.25	**383.72**
			Std	24.79	24.91	16.22	26.92	13.12	9.07
	10	2	Obj	261.72	288.94	321.17	328.65	320.87	**345.27**
			Std	14.68	17.00	13.77	19.03	11.49	11.06
		3	Obj	299.59	318.15	363.82	371.20	357.48	**390.07**
			Std	15.04	13.98	9.70	13.86	10.02	11.77
		4	Obj	304.98	329.67	370.66	407.59	382.23	**417.22**
			Std	12.19	11.12	5.30	11.09	5.68	7.64
150	5	2	Obj	319.90	372.73	428.83	456.53	460.26	**498.15**
			Std	68.76	56.26	29.61	45.74	12.86	15.20
		3	Obj	346.97	415.87	468.94	489.06	515.60	**558.65**
			Std	64.19	49.86	26.37	40.49	9.15	12.53
		4	Obj	365.53	438.08	506.80	551.54	541.55	**581.43**
			Std	42.31	41.67	21.04	34.93	8.11	11.30
	10	2	Obj	431.74	444.63	488.35	503.75	488.83	**519.55**
			Std	15.72	28.93	18.60	28.71	12.14	11.92
		3	Obj	450.61	485.61	539.07	562.10	561.69	**589.71**
			Std	14.32	15.84	11.19	20.08	6.82	9.92
		4	Obj	454.29	514.05	573.55	596.86	586.39	**614.11**
			Std	11.97	12.51	10.57	17.04	5.70	10.11
200	5	2	Obj	398.75	554.82	599.78	635.48	632.08	**665.18**
			Std	76.20	77.68	45.04	76.18	16.60	15.97
		3	Obj	471.01	629.11	656.25	697.39	697.92	**745.50**
			Std	72.58	60.84	27.74	67.22	13.66	15.05
		4	Obj	481.70	647.39	679.27	749.57	745.95	**779.15**
			Std	52.96	55.75	20.69	56.05	9.44	12.41
	10	2	Obj	569.48	605.23	651.70	674.95	653.94	**696.62**
			Std	27.87	44.12	20.67	52.04	11.07	15.86
		3	Obj	603.67	664.43	707.64	752.86	748.01	**788.11**
			Std	11.16	23.53	15.68	37.21	10.94	15.10
		4	Obj	612.62	676.22	729.98	773.52	767.31	**817.07**
			Std	14.62	17.44	13.55	28.08	9.74	11.98

Table 9: Complete main results on warehouse replenishment. Higher is better.

Orders	Supplier Hetero.	Time Window		(s, S) Heuristic	RL	Global Delay	MoE Delay	Hyper Heuristic	Ours
200	Low	Tight	Obj	24.73	23.27	22.96	27.12	25.89	**33.44**
			Std	1.60	4.23	3.82	4.84	2.19	1.64
		Medium	Obj	28.02	25.03	25.75	30.39	26.66	**36.16**
			Std	1.96	3.72	2.99	3.76	2.03	1.67
		Loose	Obj	28.79	28.32	29.50	34.30	29.04	**38.52**
			Std	1.37	2.34	2.44	3.70	1.86	1.88
	High	Tight	Obj	33.99	30.52	29.72	36.59	34.22	**44.80**
			Std	2.33	3.50	3.23	4.59	2.80	1.70
		Medium	Obj	37.14	31.45	32.60	41.13	38.49	**48.35**
			Std	1.92	3.31	2.83	4.01	2.07	2.05
		Loose	Obj	40.15	37.49	37.25	44.72	41.81	**51.81**
			Std	1.40	2.58	2.27	4.30	1.83	1.86
500	Low	Tight	Obj	50.63	50.28	52.33	58.91	55.50	**63.01**
			Std	3.44	5.53	5.74	7.15	3.95	2.42
		Medium	Obj	58.35	58.30	66.51	71.19	70.05	**73.39**
			Std	3.23	4.90	4.76	6.96	3.38	2.41
		Loose	Obj	58.84	60.34	73.84	80.97	77.71	**84.03**
			Std	2.57	3.93	4.22	5.55	3.97	3.02
	High	Tight	Obj	68.87	62.08	66.52	75.74	66.84	**92.09**
			Std	3.81	5.32	5.98	7.54	3.24	3.64
		Medium	Obj	84.26	74.29	76.17	89.60	83.22	**101.39**
			Std	3.41	5.42	5.95	7.44	3.74	3.61
		Loose	Obj	94.45	78.59	83.73	100.75	96.80	**116.38**
			Std	2.72	4.54	5.16	6.43	3.19	4.14
1000	Low	Tight	Obj	93.68	79.72	86.81	95.52	94.38	**132.03**
			Std	4.45	5.12	6.78	8.29	5.01	3.61
		Medium	Obj	119.15	108.67	120.40	130.03	127.19	**150.98**
			Std	3.60	7.11	6.99	7.31	4.83	3.64
		Loose	Obj	123.14	120.05	136.35	152.78	148.84	**157.64**
			Std	3.01	5.95	5.17	6.72	3.89	4.37
	High	Tight	Obj	130.78	119.54	127.88	137.90	128.60	**155.58**
			Std	4.43	8.43	7.65	9.69	5.47	4.59
		Medium	Obj	172.03	154.69	165.28	184.07	180.32	**219.10**
			Std	4.97	8.11	6.42	9.61	5.67	4.42
		Loose	Obj	196.43	167.46	188.43	203.55	201.02	**234.16**
			Std	4.57	6.36	6.01	9.11	3.74	4.60

Overall, the results support the same conclusion as the representative results in the main text: explicitly learning task-specific triggering requirements improves dynamic task–resource assignment consistently across all three benchmarks and across a broad range of environment conditions.

A.4 Delay-Benefit Estimator Analysis

We evaluate whether the estimator reliably predicts the benefit of delay on held-out snapshots collected before policy optimization. For each pending task, we estimate its empirical delay benefit using Eq. (6) with L_{test} rollouts per candidate epoch. The resulting targets and confidence intervals are standardized using the training-set statistics μ_D and σ_D, matching the space used for threshold construction.

We report three metrics. **CI Agreement** measures the proportion of predicted standardized delay benefits that fall within the corresponding 95% bootstrap confidence intervals of the empirical rollout estimates. **Mean CI Width** measures the average width of these standardized intervals and reflects the uncertainty of the rollout-based targets. **Standardized MAE** is the mean absolute difference between estimator predictions and empirical rollout labels, both standardized using the training-set statistics μ_D and σ_D. Together, CI Agreement and Mean CI Width assess how well predictions agree with rollout estimates in light of their uncertainty, while Standardized MAE directly measures prediction error.

We further perform a closed-loop evaluation after training the triggering policy. Environment snapshots and pending tasks are collected from trajectories induced by the learned RL policy, and their empirical delay benefits are recomputed through environment rollouts. This examines whether the estimator remains reliable under the state distribution encountered by the complete framework.

Table 10: Reliability of the delay-benefit estimator under offline and closed-loop evaluation.

Evaluation	Benchmark	CI Agreement	Mean CI Width	Standardized MAE
Offline	DJSP	0.920	0.148	0.070
	WTA	0.912	0.159	0.064
	Warehouse	0.937	0.132	0.061
Closed-loop	DJSP	0.898	0.156	0.076
	WTA	0.908	0.148	0.069
	Warehouse	0.923	0.134	0.059

As shown in Table 10, the estimator achieves high CI agreement with relatively narrow confidence intervals and low Standardized MAE on all three benchmarks. These results indicate that its predictions closely match empirical rollout-based estimates, both in terms of uncertainty-aware agreement and direct prediction error. Moreover, the closed-loop results remain close to the offline results despite the policy-induced shift in visited states. This supports the use of the estimator as a stable and reliable delay-benefit signal for downstream threshold selection and triggering control.

A.5 Sensitivity to the Number of Triggering Levels

We examine the sensitivity of our method to the number of threshold levels K. For each $K \in \{2, 5, 10, 15, 20\}$, we construct candidate thresholds using the empirical quantiles in Section 4.3, retain $a_i = 0$ as the earliest-feasible action, and train the triggering policy under the same setup.

As shown in Table 11, increasing K from 2 to 10 substantially improves performance on all three benchmarks. With too few candidate thresholds, the policy has limited choices for setting task-specific triggering tolerances; increasing K provides a wider range of requirements from which to choose. Once K reaches 10, however, performance largely levels off: further increases to 15 or 20 produce only small, inconsistent changes across the benchmarks. In our training runs, these larger action spaces also required more iterations and training time to converge, without yielding consistent gains in final performance. We therefore use $K = 10$ as a practical balance between performance and training cost. Overall, the results suggest that performance is relatively insensitive to the precise choice of K once sufficient triggering levels are available. Since changing K under our quantile

Table 11: Performance under different numbers of triggering levels K. Results are reported as mean and standard deviation over repeated runs.

K	DJSP ↓	WTA ↑	Warehouse ↑
2	422.83±20.11	431.53±10.48	70.41±3.98
5	399.69±21.73	459.41±14.84	81.74±3.92
10	376.80±30.16	498.15±15.20	92.09±3.64
15	374.63±29.47	496.98±19.71	90.24±4.31
20	374.91±33.20	502.03±23.44	91.82±5.79

rule affects both threshold spacing and coverage, this conclusion applies to the complete threshold construction rather than to discretization granularity alone.

A.6 Training Details

Table 12: Training and evaluation settings used in the experiments.

Hyperparameter	Value
RL algorithm	PPO
Number of threshold levels K	10
Action space	$\{0, 1, \ldots, K\}$
Learning rate	3×10^{-4}
Batch size	256
PPO optimization epochs	5
Discount factor γ	0.99
GAE parameter λ	0.95
Clip range	0.2
Advantage normalization	Yes
Actor MLP	$[64, 64, 64]$
Critic MLP	$[64, 64, 64]$
Estimator during PPO training	Frozen
Training seeds	13–22
Evaluation seeds	27–36
Evaluation policy	Deterministic
Test repeats per instance	100
Reported result	Mean ± standard deviation

All learning-based triggering policies are optimized with PPO (Schulman et al., 2017). For our method, the delay-benefit estimator is trained first using rollout-generated supervision and is then frozen throughout RL training and evaluation. Its standardized predictions are used to construct the candidate thresholds from the empirical training-set quantiles, as described in Section 4.3. We use $K = 10$ threshold levels in the main experiments.

When a task enters control, the policy observes the current task–resource state and selects a single action $a_i \in \{0, 1, \ldots, K\}$. Action $a_i = 0$ specifies earliest-feasible triggering, whereas $a_i = k$, $k \in \{1, \ldots, K\}$, assigns the persistent triggering threshold δ_k to task i. The selected requirement remains associated with the task until it is triggered and is not reconsidered by RL at subsequent decision epochs. Instead, the fixed delay-benefit estimator reevaluates the task as the environment evolves, and a feasible task with $a_i > 0$ is automatically triggered once its estimated standardized delay benefit falls below the selected threshold. Triggered tasks are then passed to the benchmark-specific domain heuristic for feasible assignment construction.

The triggering policy is optimized using the original problem-level environment reward without additional reward shaping from the estimator. Each threshold choice can therefore influence future returns through resource consumption, competition among pending tasks, and subsequent assignment

opportunities. The Actor and Critic use separate three-layer MLPs with hidden sizes $[64, 64, 64]$. PPO uses a learning rate of 3×10^{-4}, a batch size of 256, five optimization epochs per update, discount factor $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, and clipping range 0.2. Advantages are normalized during policy optimization. The complete settings are summarized in Table 12.

We use training seeds 13–22 and evaluation seeds 27–36. Evaluation is performed with a deterministic policy. Each test instance is repeated 100 times, and we report the mean and standard deviation over the evaluation repetitions.

A.7 Domain Heuristic Details

For all methods that employ a downstream heuristic, triggered tasks are processed by the same benchmark-specific heuristic to construct feasible assignments under the current environment state. The heuristic operates only on tasks that have already been triggered; the triggering mechanism determines when a task becomes available to the heuristic, while the heuristic determines how the currently triggered tasks are assigned to resources. We use the following domain-specific rules throughout the experiments.

DJSP. For each schedulable operation, the heuristic computes a slack-based urgency score using the job due date, the current time, and its remaining processing time. Operations belonging to jobs with smaller remaining slack are given higher priority. Following this priority order, an operation is assigned to its designated machine whenever its precedence constraints are satisfied and the machine is currently available. The procedure continues until no additional triggered operation can be feasibly scheduled under the current state.

WTA. The heuristic repeatedly selects the currently observable triggered target with the highest threat level. For the selected target, it chooses the cheapest feasible weapon–ammunition option satisfying ammunition–target compatibility, engagement-range, and remaining-inventory constraints. The corresponding engagement is then executed, and the procedure continues with the remaining triggered targets until no additional feasible engagement can be constructed.

Warehouse Replenishment. The heuristic assigns each triggered order a priority score based on a weighted combination of urgency, order value, and shortage level. It repeatedly selects the highest-scoring pending order and assigns it to the lowest-cost supplier that provides the required product and can satisfy the corresponding delivery-window requirement. The procedure continues until all currently triggered orders have been processed or no additional feasible assignment can be constructed.