Highlights

Dynamic single additive manufacturing machine scheduling considering postponement decisions: A lookahead approximate dynamic programming

Hao Wu, Chunlong Yu

- Investigate strategic postponement in dynamic additive manufacturing machine scheduling.
- Model the problem as a finite-horizon Markov decision process with proactive postponement choices.
- Propose a lookahead approximate dynamic programming approach to overcome curse of dimensionality.
- Demonstrate performance superior to heuristics and competitive with offline optimum.

Dynamic single additive manufacturing machine scheduling considering postponement decisions: A lookahead approximate dynamic programming

Hao Wu[a], Chunlong Yu[a,*]

[a]*School of Mechanical Engineering, Tongji University, Caoan Road 4800, 201804, Shanghai, China*

Abstract

Additive Manufacturing (AM), facing challenges of high costs, is driving service providers toward social manufacturing by pooling distributed orders to amortize fixed expenses. However, for a single-machine operator in such environments, real-time scheduling under stochastic order arrivals presents a critical challenge: whether to process available jobs immediately to reduce tardiness or to strategically postpone production in anticipation of future arrivals for cost-efficient batch consolidation. We model this problem as a finite-horizon Markov decision process, where postponement is explicitly treated as a proactive choice, and the objective is to maximize long-term expected profit. To overcome the curse of dimensionality, we propose a lookahead approximate dynamic programming approach that evaluates the state values by simulating future trajectories. The proposed approach integrates a Monte Carlo tree search framework for efficient lookahead tree evaluation, a one-step lookahead surrogate policy to guide rollout simulations, and a Pareto-based pruning strategy to focus on high-quality feasible decisions. Extensive numerical experiments demonstrate that the proposed policy outperforms conventional reactive rule and heuristics with static postponement thresholds, achieving an average empirical competitive ratio of 1.27 relative to the offline optimum.

Keywords: Additive manufacturing, Dynamic scheduling, Postponement, Markov decision process, Approximate dynamic programming, Direct lookahead approximation, Monte Carlo tree search

1. Introduction

Additive Manufacturing (AM) is revolutionizing the contemporary manufacturing landscape by enabling the production of highly customized parts with minimal material waste and significant design freedom. However, the widespread industrial adoption of AM technologies, particularly Powder Bed Fusion (PBF) variants like Selective Laser Melting (SLM), remains hindered by substantial capital investments and high per-batch operational expenses (Thomas & Gilbert, 2014).

A characteristic of the SLM process is its subadditive cost structure, which creates a compelling economic imperative for batch processing. Rather than scaling linearly with the number of parts, production costs are largely driven

*Corresponding author.
Email addresses: hao_wu@tongji.edu.cn (Hao Wu), chunlong_yu@tongji.edu.cn (Chunlong Yu)

by fixed setup overheads and height-dependent recoating cycles that are incurred regardless of batch density (Lv et al., 2021). This distinct structure implies that processing parts individually is economically inefficient; conversely, consolidating multiple parts into a single build allows these fixed and semi-fixed costs to be amortized over a larger volume. As demonstrated by Ding et al. (2021), unit costs in SLM drop significantly as capacity utilization improves. Consequently, operational strategies that achieve high nesting density and minimize height variance are critical for the economic viability of AM systems (Altekin, F. T., & Bukchin , 2022).

The economic advantages of batching have driven the emergence of the social manufacturing paradigm, wherein small and medium-sized AM service providers share their limited production capacity and pool geographically dispersed orders through cloud platforms, such as Xometry (Xometry, 2025) and JLC-3DP (JLC-3DP, 2025). By forming cost-effective batches, the per-part cost can be substantially reduced, thereby achieving the economies of scale necessary to make AM competitive for a broader range of applications (Gopsill, 2024).

However, this paradigm introduces significant operational challenges. A single-machine operator, representing a typical entity within this ecosystem, must navigate a dynamic and stochastic environment. Unlike mass production environments with predictable demand, the order stream in social manufacturing is characterized by high stochasticity and heterogeneity. Customized orders arrive randomly with diverse geometries and distinct due dates. The operator is thus tasked with a complex real-time decision-making problem: dynamically balancing the conflicting objectives of minimizing production costs via batch consolidation and maintaining high service levels via timely delivery.

A straightforward approach to managing such a system is *reactive scheduling*, where production decisions are triggered solely by discrete events such as a new order arrival or the machine becoming idle (Sabuncuoglu & Bayız, 2000). While simple to implement, a purely reactive policy that processes known orders as soon as the machine is free can lead to significant opportunity losses. By initiating a batch immediately with the currently available parts, the operator irrevocably consumes machine capacity for a significant duration (often tens of hours). This effectively locks out any high-priority or geometrically compatible orders that might arrive shortly after the batch starts. This myopic behavior prevents the exploitation of future consolidation opportunities, potentially resulting in a degradation of overall operational performance.

Postponement, as a potent strategy in dynamic environments, refers to the proactive delay of a decision until more information is available or a more opportune moment arises. Its fundamental advantage, when compared to reactive scheduling, lies in the ability to leverage the passage of time to enhance decision quality. This is achieved by enabling more efficient resource utilization through the consolidation of tasks or orders (Van Mieghem & Dada, 1998; Yang & Burns, 2003; Xie, Ma, & Xin, 2025). In the context of this particular problem, strategic postponement of batch processing initiation offers a chance to accumulate a larger pool of orders, which in turn, facilitates the possibility of consolidating them into more cost-effective batches (Bhimaraju, Etesami, & Varshney, 2025). Therefore, incorporating the option of postponement in dynamic AM machine scheduling holds significant potential for enhancing operational performance (Prataviera, Jazairy, & Abushaikha, 2024).

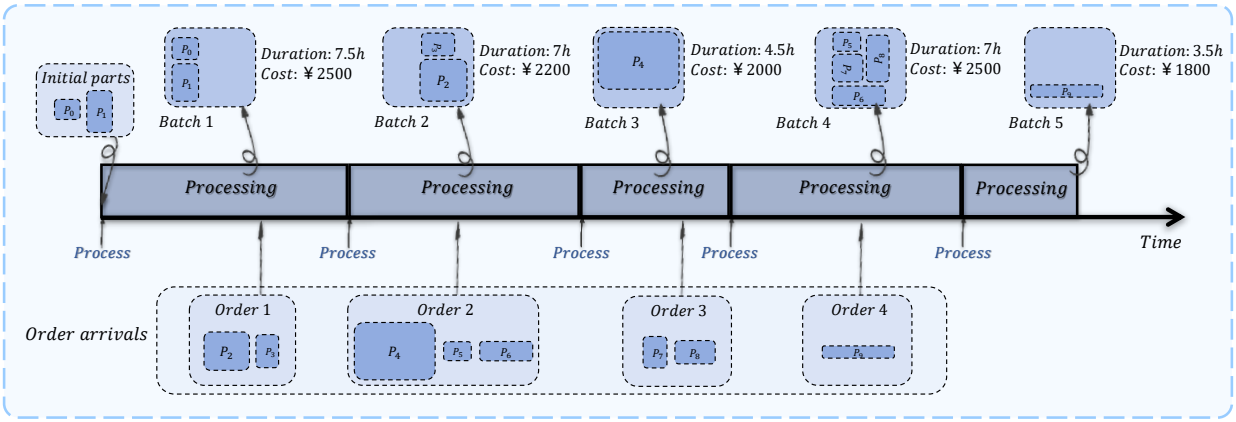
The strategic value of postponement in dynamic single AM machine scheduling is multifaceted, as illustrated in

Figure 1. Fundamentally, this strategy leverages the flexibility to delay commitment, enabling the consolidation of geometrically compatible orders into cost-efficient batches. Consider the scenario where orders P_0 and P_1 are initially available. A reactive policy (Figure 1a) would process them immediately in Batch 1, leaving subsequent arrivals P_2 and P_3 into a separate Batch 2. This fragmentation incurs high total costs due to the duplication of pre-processing, post-processing, and recoating times. In contrast, a policy with strategic postponement (Figure 1b) employs "proactive waiting." By consolidating early and late arrivals ($P_0 - P_3$) into a single, high-density batch, the system amortizes fixed costs. Furthermore, strategic postponement could enhance the utilization of build platform. As shown in Figure 1b, by postponing the immediate processing of any subset of Order 2, whose parts cannot be feasibly packed within a batch, several better-nested batches (Batch 2, 3) can be formed once parts with complementary geometries arrive subsequently. Finally, while postponement inherently risks delaying current orders, this apparent paradox can be resolved through efficiency gains. The reduction in total processing times achieved via consolidation effectively increases the system's throughput. This allows the machine to "catch up," potentially resulting in earlier completion of subsequent parts (e.g., $P_5 - P_8$) compared to reactive scheduling, where machine capacity is often occupied by inefficient builds.

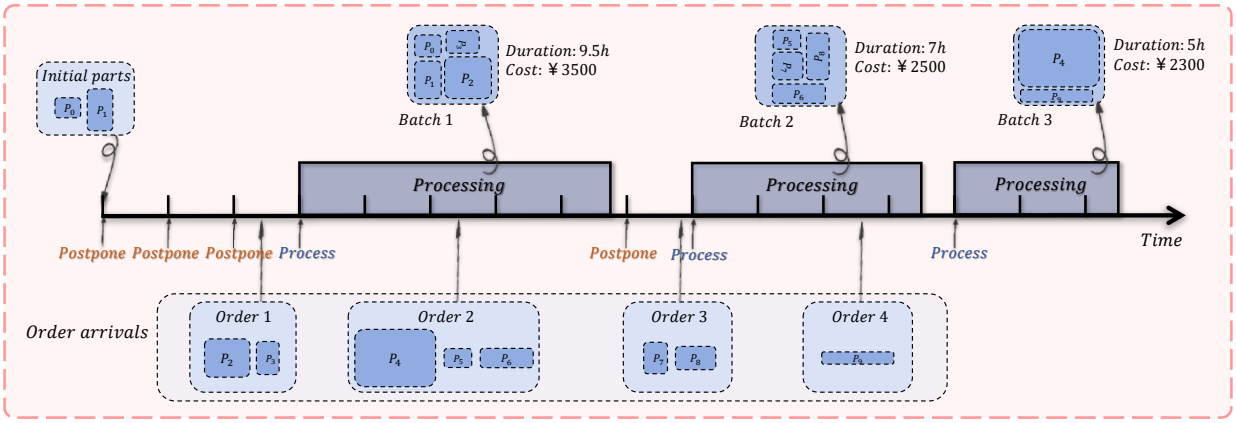
Motivated by these multifaceted benefits, this study explicitly integrates strategic postponement into the dynamic scheduling framework for a single AM machine. Consequently, the operational problem for a single-machine AM operator transforms into a critical sequential decision-making challenge: at any discrete point in time, should the system *process* a subset of available orders immediately, or *postpone* to await better opportunities? This decision encapsulates a fundamental trade-off. On one hand, immediate processing can reduce tardiness risk in many cases and upholds service levels, but it may incur higher per-unit costs. On the other hand, postponement presents the potential for significant cost reductions and efficiency gains through improved consolidation, yet it inherently carries the risk of delaying urgent orders. Navigating this trade-off motivates the core research question in this paper: In a dynamic environment characterized by stochastic AM order arrivals, how can one optimally decide when to postpone and which parts to include in a batch to maximize long-term profitability?

Existing research in dynamic AM scheduling has primarily focused on reactive strategies for batching and sequencing known orders (Li et al., 2019; Wu et al., 2022; Sun et al., 2025). While valuable, they often overlook the temporal dimension of decision-making, specifically, the option to *not* process.The absence of a formal methodology for making such proactive postponement decisions limits the economic potential of on-demand AM systems. To address this gap, this paper develops a dynamic scheduling framework for a single AM machine where postponement is an explicit, proactive strategic choice. The main contributions of this paper are summarized as follows:

- We formulate the dynamic single-machine AM scheduling problem as a finite-horizon Markov Decision Process (MDP), which explicitly incorporates postponement as a decision variable. This formulation shifts the scheduling paradigm from reactive to proactive, enhancing overall profitability through strategic postponement. To provide a theoretical offline bound for benchmarking online policies, we formulate a Mixed-Integer Linear



(a) Dynamic single AM machine scheduling without postponement considerations.



(b) Dynamic single AM machine scheduling with strategic postponement.

Figure 1: A comparison of dynamic single AM machine scheduling *without* (a) and *with* (b) strategic postponement. In scenario (a), the operator processes orders immediately, resulting in five batches with a total processing cost of 11,000 CNY and a total processing duration of 29.5 hours. In scenario (b), by strategically postponing production with intervals of 2 hours, the operator consolidates P_0, P_1 with the arriving P_2, P_3 into a single, efficient Batch 1. Similar consolidation occurs for subsequent orders. This strategy reduces the total number of batches to three, lowering the total processing cost to 8,300 CNY and the total processing duration to 21.5 hours.

Program (MILP) that assumes perfect a priori information of order arrivals.

- To overcome the curse of dimensionality inherent in the exact resolution of MDP, we develop a Direct Lookahead Approximation (DLA) policy. This policy enables online decision-making by constructing a lookahead tree, where state values are estimated through simulations of future trajectories. We utilize a tailored Monte Carlo Tree Search (MCTS) framework, which effectively navigates the immense state space to approximate optimal decision. Given the computational intractability of deriving the optimal policy governing simulations, a one-step lookahead heuristic is employed as a surrogate. To mitigate the combinatorial growth of decision space, we introduce a Pareto-based pruning strategy that directs computational resources towards high-potential decisions.

The remainder of this paper is organized as follows. Section 2 provides a review of the relevant literature. Section 3.1 formally defines the dynamic decision-making problem, presents the MDP formulation, and introduces the mathematical model for offline bound. Section 4 details the proposed lookahead-based Approximate Dynamic Programming (ADP) approach for online decision-making. Section 5 presents the numerical experiments and discusses the results. Section 6 concludes the paper and suggests directions for future research.

2. Related Works

In this section, we firstly review the literature on dynamic AM machine scheduling. Next, we examine research that incorporates the postponement strategy in dynamic decision-making contexts. Finally, we discuss studies that apply the MCTS framework as a direct lookahead policy for online decision-making.

2.1. Dynamic Additive Manufacturing Machine Scheduling

With the emergence of social AM paradigms, certain dynamic scheduling strategies are needed to enhance system performances under a diverse and fluctuating stream of orders.

The literature has begun to address these dynamic challenges. Li et al. (2019) address the dynamic order acceptance and scheduling problem for an on-demand PBF system with stochastic order arrivals. The objective is to maximize the average profit per unit-time by making real-time decisions on order acceptance and batch formation. They propose a metaheuristic, where nesting feasibility is managed by a 2D bin packing heuristic.

In the context of cloud-based manufacturing, Wu et al. (2022) address the online scheduling of orders across geographically distributed AM machines, with each machine operating under its own independent time window. To minimize the average production cost per unit volume, they develop a non-preemptive heuristic that incorporates a skyline-based heuristic for nesting parts within batches. A MILP model assuming complete future information is also formulated to provide a competitive bound for the dynamic problem.

Advanced learning-based methods have also been explored. Sun et al. (2025) focus on minimizing total tardiness for a parallel AM machine system with dynamic order arrivals, where the nesting subproblem is simplified to an area-based constraint. They employ a deep reinforcement learning framework, specifically a dueling deep Q-network, to learn an effective scheduling policy.

Despite these contributions in managing order acceptance, batch formation, and sequencing under stochastic arrivals, a gap persists in the literature: the consideration of postponement as a decision. While existing works react to dynamic events, they do not typically consider the proactive postponement of a batch to potentially consolidate future orders. The chance to enhance overall performance of dynamic AM scheduling through postponement decisions remains an underexplored area of research.

2.2. Postponement Strategy in Dynamic Decision-making

In many dynamic operational environments, decisions must be made sequentially under uncertainty regarding future events, such as order arrivals, processing times, or resource availability. A potent approach in such settings is the *postponement* strategy. Although it appears in the literature under various terms, such as “holding," “delay," or “consolidation", these concepts share a common theoretical foundation: the deliberate, temporary deferral of a decision until more information is revealed or a more opportune moment for execution arises (Van Mieghem & Dada, 1998; Yang & Burns, 2003; Xie, Ma, & Xin, 2025). This often involves strategically retaining items, orders, or tasks rather than processing them immediately upon readiness.

The theoretical underpinnings of why such postponement can be beneficial in dynamic settings are explored by Xie, Ma, & Xin (2025). They study the fundamental trade-off in delaying real-time decisions to gather more information, proving that the performance gap between an online “delayed prophet" algorithm and the offline optimal policy decays exponentially with the length of the delay. Their work provides the insight that “a little delay is all we need" to significantly improve decision quality, a principle that resonates across various applications of postponement.

In dispatching and logistics, postponement typically manifests as strategic waiting to form more efficient batches. Van Heeswijk, Mes, & Schutten (2017) employ ADP to manage vehicle dispatching from urban consolidation centers. Their mechanism of delaying vehicle departures allows for the accumulation of more parcels, with the intended benefit of improving vehicle utilization and reducing the number of trips. Similarly, Ulmer et al. (2021) study dynamic meal delivery, where drivers may postpone departure to manage random food ready times or to facilitate order bundling. This operational delay is a mechanism to absorb upstream uncertainties and improve downstream delivery efficiency while respecting deadlines. In the context of the “subadditive dispatching problem," Erazo & Toriello (2025) show that waiting for more orders can lead to batches with lower per-item dispatch times due to subadditive cost functions, ultimately minimizing the makespan.

Postponement strategies have also been studied in e-commerce and online retailing. Aminian, Ma, & Xin (2023) investigate real-time personalized order holding, where the intention is to consolidate an order with potential future orders from the same customer. They leverage predictive analytics to inform index-based holding policies, with the

goal of reducing total fulfillment costs. Chen, Yan, & Lim (2024) address a similar problem, focusing on the trade-off between holding costs and the benefits of shipment consolidation. For multi-location settings, Wei, Kapuscinski, & Jasin (2021) examine shipping consolidation across two warehouses, where delaying shipments is a crucial mechanism for achieving cost-effective consolidation while managing delivery deadlines. In the context of "no-rush delivery" options, Chen et al. (2024) highlight the strategic benefits of retailer-initiated postponement to mitigate workload imbalances and improve sustainability, showcasing how the strategy can serve broader goals beyond immediate cost savings.

While the postponement strategy has been effectively applied to a wide range of dynamic decision-making problems, yielding benefits such as improved resource utilization, cost reduction, and better risk management, its application to dynamic batch-processing-machine scheduling remains unexplored.

2.3. Monte Carlo Tree Search as Direct Lookahead Policies

While Dynamic Programming provides a foundational framework for optimal sequential decision-making, its practical application is often precluded by the "curse of dimensionality." ADP offers a suite of methodologies to seek near-optimal policies. Among these, DLA, focusing computational effort on evaluating decisions from the current state by simulating their consequences over a finite future horizon, is a compelling strategy (Powell, 2011, 2019). In particular, MCTS has emerged as an efficient framework of DLA implementation. It incrementally constructs a search tree and executes simulations initiated from potential future states, where the Upper Confidence Bound for Trees (UCT) criterion facilitates navigation through vast search spaces (Kocsis & Szepesvári, 2006; Browne et al., 2012).

The efficacy of MCTS as a direct lookahead policy for dynamic decision-making has been demonstrated across a diverse range of domains, most prominently in Artificial Intelligence (AI) for games. The AlphaGo, AlphaZero, and MuZero programs achieve superhuman performance by synergistically integrating MCTS with deep neural networks. In this synergy, MCTS navigates the search of lookahead state-action spaces, while deep neural networks serve as learned approximations of value functions, policy functions, and environment dynamics (Silver et al., 2016, 2018; Schrittwieser et al., 2020). From an operations research perspective, these advancements represent a sophisticated form of lookahead-based policy improvement for finite-horizon MDPs (Fu, 2016).

Beyond game AI, MCTS has proven useful for dynamic resource management. Bertsimas et al. (2017) provide a comparative analysis of MCTS and rolling horizon optimization for large-scale dynamic resource allocation, highlighting the strength of the lookahead-based nature of MCTS in exploring stochastic future scenarios. Fu et al. (2021) propose a policy-network-assisted MCTS to guide online deployment decisions within service function chain network, effectively evaluating the long-term impact of decisions by the lookahead of its consequences.

In contexts of logistics and routing problems, MCTS offers effective solutions. For dynamic vehicle routing, Mańdziuk & Okulewicz (2015) propose an MCTS-based method to make adaptive dispatching and routing decisions in response to real-time customer requests. For dynamic police patrolling, Tschernutter & Feuerriegel (2024) utilize

MCTS to optimize routes to maximize long-term crime prevention, where consequent outcomes of patrol routing are simulated in lookahead model to anticipate spatial-temporal crime distributions. For stochastic orienteering with chance constraints, Carpin (2025) apply MCTS for solving, where its lookahead simulations effectively manage the trade-off between the expected rewards and the probability of violation for different paths.

Despite its demonstrated success in these areas, the application of MCTS as a DLA policy for dynamic machine scheduling, remains relatively unexplored. The inherent characteristics of MCTS, its online planning capability, its simulation-based approach for handling stochastic events like random order arrivals, and its flexibility for integration with learned heuristics, make it a promising candidate for this problem class.

3. Problem Formulation

3.1. Problem Description

We consider a single AM machine that utilizes the SLM technique to process batches of customized parts. The build chamber of the machine is characterized by a cuboidal volume with dimensions $\mathcal{L} \times \mathcal{W} \times \mathcal{H}$ (length, width, height). The machine's operational performance is defined by the scanning speeds per unit volume for the part body (V^{body}) and support structures (V^{supp}), as well as a recoating time per unit of height (t^{rec}). Each ordered part p is defined by its geometric attributes: volume (v_p), support structure volume (s_p), height (h_p), and the dimensions of its minimum bounding box projected onto the build platform ($l_p \times w_p$).

The batching process is governed by a set of operational rules. First, once a batch is initiated, it is uninterruptible; no parts can be added or removed, and all parts within the batch share a common completion time. Second, any part exceeding the build chamber's height $\mathcal{H}$ is considered infeasible and is rejected a priori. Third, parts within a batch must satisfy a set of nesting constraints (Yu et al., 2022). Specifically, their bounding boxes must be placed within the platform boundaries ($\mathcal{L} \times \mathcal{W}$) without overlapping, and they cannot be stacked vertically. Additionally, the parts must be aligned parallel to the platform's axes but can be rotated by 90° around the z-axis to optimize the fit. Figure 2 provides an illustrative example of this nesting problem within a batch.

The total processing time of a batch comprises three components: pre-build time, build time, and post-build time. The pre-build time, t^{pre}, encompasses preparatory tasks such as file loading, parameter setting, powder tank refilling, and substrate pre-heating. The post-build time, t^{post}, includes substrate cooling, part disassembly, cleaning, and potential recoater maintenance. Due to their reliance on manual operations, the durations of these pre- and post-build phases are difficult to model analytically and are therefore estimated as fixed values based on established production practices. We denote the sum of auxiliary times as $t^{\text{aux}} = t^{\text{pre}} + t^{\text{post}}$. In contrast, the build time, t^{bld}, can be estimated with reasonable accuracy using an analytical formula derived from the batch geometry and machine parameters, $t^{\text{bld}} = \sum_{p \in \mathbb{P}^{\text{bat}}} \left(\frac{v_p}{V^{\text{body}}} + \frac{s_p}{V^{\text{supp}}} \right) + t^{\text{rec}} \cdot \max_{p \in \mathbb{P}^{\text{bat}}} h_p$, where $\mathbb{P}^{\text{bat}}$ is the set of parts in the batch (Lv et al., 2021; Wu, Yu, & Yu, 2025). This formula captures the dependency of build time on total volume and maximum batch height.

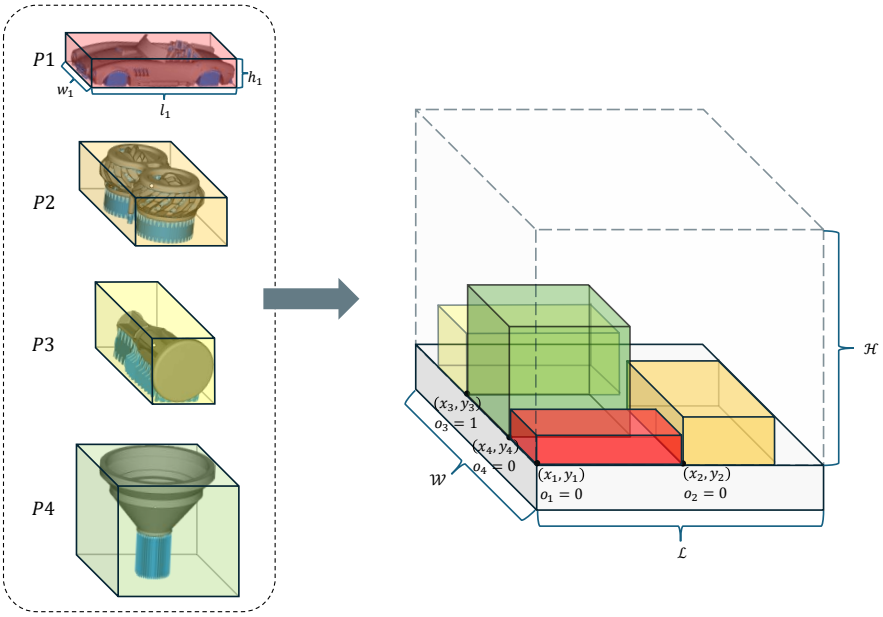


Figure 2: Illustration of the 2D nesting problem. Parts must be packed within the platform boundaries ($\mathcal{L}, \mathcal{W}$) without overlapping, allowing 90° rotations.

The scheduling environment is inherently dynamic and stochastic. Orders for parts arrive randomly over a finite time horizon according to a stochastic process, $\mathcal{F}_D$, typically derived from historical data or domain expertise. Each part p is associated with an arrival time r_p and a requested due date d_p. All information regarding a part is revealed only upon its arrival, and a part can only be scheduled for processing after this point. The time horizon is discretized into K equal-length time slots of duration u, with decisions made at discrete time points $k \cdot u$, where $k \in \{0, 1, \ldots, K\}$. To avoid operational inefficiencies, the time slot length u is constrained to be shorter than t^{aux}. At each decision epoch k, the operator must decide (1) whether to *initiate* processing immediately or to *postpone* the decision to the next epoch, and if choosing to process, (2) which subset of available parts to *select* for the batch.

The objective is to devise a scheduling policy that maximizes the total profit over the entire time horizon. Total profit is defined as the total revenue from completed parts, minus the total production costs and any penalties for tardiness. Each part p carries a price ϱ_p, reflecting its geometry and delivery urgency. The cost of processing a batch is composed of three elements (Rickenbacher, Spierings, & Wegener, 2013). First, the operator cost, which covers manual tasks such as data preparation, machine setup, and part removal, is represented as a fixed value (C^{opr}) per batch. Second, the material cost includes powder consumption (proportional to the total volume of the build) and inert gas usage (proportional to the build time). Third, the energy cost is estimated to be proportional to the build time, as this phase dominates the machine's power consumption (Lin & Yu, 2025). In accordance with common industrial practice, a tardiness penalty, proportional to the length of the delay T_p, is incurred for any part p not completed by its due date d_p.

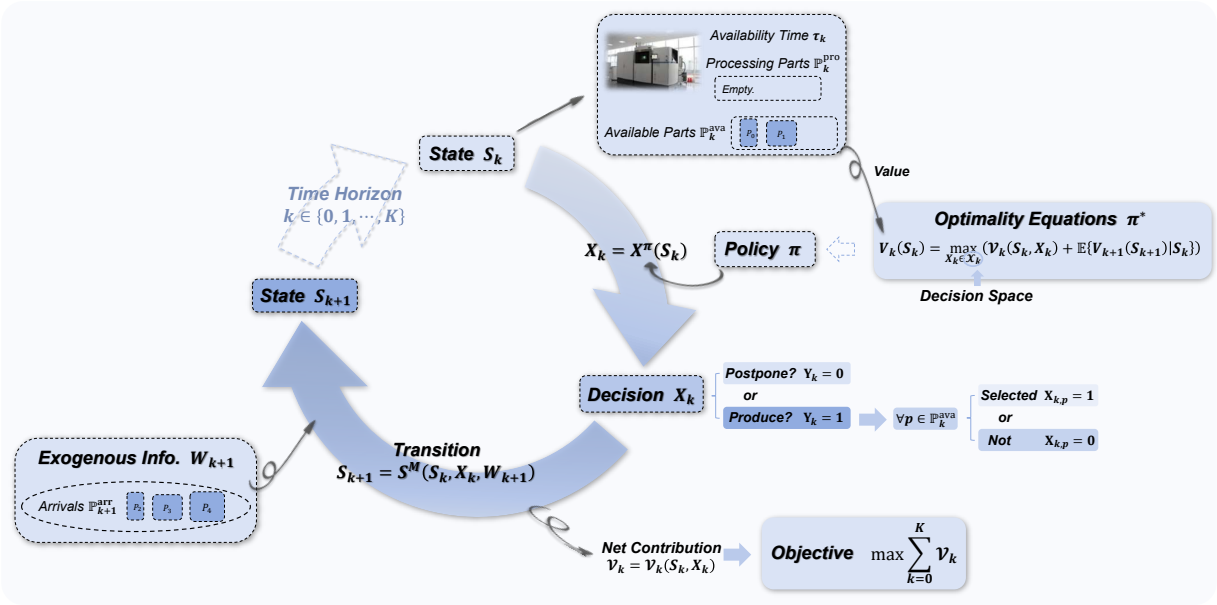


Figure 3: Schematic of the state transition dynamics in the Markov Decision Process framework.

3.2. Markov Decision Process Formulation

The dynamic scheduling problem can be formally modeled as a finite-horizon MDP. The core components of this formulation, i.e., the state, decision, exogenous information, transition dynamics, objective and optimality criteria, are detailed below. The evolution of the system from one state to the next, governed by these components, is illustrated schematically in Figure 3.

State Variable. At decision epoch k, the system state S_k is defined by the tuple:

$$S_k = (\mathbb{P}_k^{\text{ava}}, \mathbb{P}_k^{\text{pro}}, \tau_k, \mathbf{A}_k) \tag{1}$$

where, the components are defined as:

- $\mathbb{P}_k^{\text{ava}}$: The set of available parts in the queue waiting to be processed. This includes newly arrived parts from the last interval $((k-1)u, ku]$ and any parts that arrived prior to $(k-1)u$ but have not yet been processed. The initial set of available parts is denoted by $\mathbb{P}_0^{\text{ava}}$.
- $\mathbb{P}_k^{\text{pro}}$: The set of parts currently being processed on the machine. This is empty if the machine is available at current epoch k.
- τ_k: The machine availability time (the time at which the machine will be available for processing). Naturally, $\tau_k \geq ku$. If $\tau_k = k \cdot u$, the machine is currently idle.
- $\mathbf{A}_k$: The attribute set containing information $\{v_p, s_p, h_p, l_p, w_p, d_p, r_p, \varrho_p\}$ for all part $p \in \mathbb{P}_k^{\text{ava}} \cup \mathbb{P}_k^{\text{pro}}$.

Decision Variables. The decision taken at epoch k, denoted by $\mathbf{X}_k$, is a vector comprising two types of binary variables: a primary decision on whether to process or postpone (Y_k), and a conditional decision on which available parts to include in a batch ($\{X_{k,p}\}_{p\in\mathbb{P}_k^{\text{ava}}}$):

$$\mathbf{X}_k = (Y_k, \{X_{k,p}\}_{p\in\mathbb{P}_k^{\text{ava}}}) \tag{2}$$

where $Y_k = 1$ implies initiating a new batch, and $Y_k = 0$ implies postponement.

Decision Space. The feasible decision space, denoted $\mathcal{X}_k(S_k)$, is constrained by machine status and geometric feasibility, which can be defined by the following constraints (3-15):

- *Machine Availability:* A batch can only be initiated if the machine is free or becomes free within the current interval (i.e., $\tau_k < (k+1)\cdot u$):

$$\tau_k - (k+1)u \le M(1 - Y_k), \tag{3}$$

  where M is a sufficiently large constant. If the machine is busy beyond $(k+1)u$, only postponement ($Y_k = 0$) is feasible.

- *Batch Validity:* If a batch is initiated ($Y_k = 1$), it must be non-empty:

$$Y_k \le \sum_{p\in\mathbb{P}_k^{\text{ava}}} X_{k,p}, \tag{4}$$

- *Height Feasibility:* The height of the batch must not exceed the height of the build chamber:

$$X_{k,p} h_p \le \mathcal{H}, \forall p \in \mathbb{P}_k^{\text{ava}}, \tag{5}$$

- *Nesting Constraints:* Any selected subset of parts must satisfy the 2D nesting constraints. Let (x_p, y_p) be the lower-left coordinates, and $o_p \in \{0, 1\}$ indicate the orientation of part p (where $o_p = 1$ corresponds to a 90° rotation around z-axis). For all $p, p' \in \mathbb{P}_k^{\text{ava}}$, the following must hold:

$$x_p + l_p(1 - o_p) + w_p o_p \le \mathcal{L} + M(1 - X_{k,p}), \forall p \in \mathbb{P}_k^{\text{ava}}, \tag{6}$$

$$y_p + w_p(1 - o_p) + l_p o_p \le \mathcal{W} + M(1 - X_{k,p}), \forall p \in \mathbb{P}_k^{\text{ava}}, \tag{7}$$

$$x_p + l_p(1 - o_p) + w_p o_p \le x_{p'} + M(3 - X_{k,p} - X_{k,p'} - PL_{pp'}), \forall p, p' \in \mathbb{P}_k^{\text{ava}}, p \ne p', \tag{8}$$

$$y_p + w_p(1 - o_p) + l_p o_p \le x_{p'} + M(3 - X_{k,p} - X_{k,p'} - PB_{pp'}), \forall p, p' \in \mathbb{P}_k^{\text{ava}}, p \ne p', \tag{9}$$

$$PL_{pp'} + PB_{pp'} + PL_{p'p} + PB_{p'p} \ge X_{k,p} + X_{k,p'} - 1, \forall p, p' \in \mathbb{P}_k^{\text{ava}}, p < p'. \tag{10}$$

  Constraints (6-7) ensure that parts within the same batch do not exceed the platform boundaries. Constraints (8-9) prevent overlap of parts assigned to the same batch. Here, the binary auxiliary decision variables, $PL_{pp'}$ and $PB_{pp'}$, represent the relative positioning of the parts. Specifically, $PL_{pp'} = 1$ if the upper-right point of part p is to the left of the lower-left point of part p', and $PB_{pp'} = 1$ if the upper-right point of part p is below the lower-left point of part p'. Both of these variables take the value 0 otherwise. Constraints (10) guarantee

that at least one valid positioning relationship is imposed between each pair of parts p and p' within the same batch. Specifically, for any two parts p and p', one of the following positioning relationships must be true: the upper-right point of part p is (1) $PL_{pp'} = 1$: to the left of the lower-left point of part p', (2) $PB_{pp'} = 1$: below the left of the lower-left point of part p', or the upper-right point of part p' is (3) $PL_{p'p} = 1$: to the left of the lower-left point of part p, (4) $PB_{p'p} = 1$: below the left of the lower-left point of part p. As illustrated in Figure 2, the placement must ensure that all parts lie within the platform boundaries and that no two parts overlap. This combinatorial feasibility problem is NP-hard in general.

- The decision variables are defined as follows:

$$X_{k,p} \in \{0,1\}, \forall p \in \mathbb{P}_k^{\text{ava}}, \tag{11}$$

$$Y_k \in \{0,1\}, \tag{12}$$

$$o_p \in \{0,1\}, \forall p \in \mathbb{P}_k^{\text{ava}}, \tag{13}$$

$$PL_{pp'}, PB_{pp'} \in \{0,1\}, \forall p, p' \in \mathbb{P}_k^{\text{ava}}, \tag{14}$$

$$x_p, y_p \geq 0, \forall p \in \mathbb{P}_k^{\text{ava}}. \tag{15}$$

Exogenous Information. The exogenous information realized between decision epochs k and $k+1$ is denoted by W_{k+1}. It incorporates the parts arrived within this time slot $(ku, (k+1)u]$, $\mathbb{P}_{k+1}^{\text{arv}}$, and their corresponding attributes $\mathbf{A}_{k+1}^{\text{arv}} = \{v_p, s_p, h_p, l_p, w_p, d_p, r_p, \varrho_p \mid p \in \mathbb{P}_{k+1}^{\text{arv}}\}$. Notably, $W_0 = (\mathbb{P}_0^{\text{arv}}, \mathbf{A}_0^{\text{arv}})$ is the initially arrived parts ($r_p = 0$) and their information. The arrival process follows the distribution $\mathcal{F}_D$. A sample path ω defines a complete sequence of exogenous information $\mathbf{W}(\omega) = (W_0(\omega), \ldots, W_K(\omega))$ over the time horizon.

Transition Function. The system transitions from state S_k to S_{k+1} according to the transition function $S_{k+1} = S^M(S_k, \mathbf{X}_k, W_{k+1})$. This function updates the state variables as follows:

$$\mathbb{P}_{k+1}^{\text{pro}} = \begin{cases} \mathbb{P}_k^{\text{pro}}, & \text{if } \tau_k \geq (k+1) \cdot u \\ \{p \in \mathbb{P}_k^{\text{ava}} \mid Y_k X_{k,p} = 1\}, & \text{if } \tau_k < (k+1) \cdot u \end{cases}, \tag{16}$$

$$\mathbb{P}_{k+1}^{\text{ava}} = \mathbb{P}_k^{\text{ava}} \cup \mathbb{P}_{k+1}^{\text{arv}} \setminus \mathbb{P}_{k+1}^{\text{pro}}, \tag{17}$$

$$t_k^{\text{bld}} = \sum_{p \in \mathbb{P}_{k+1}^{\text{pro}}} \left(\frac{v_p}{V^{\text{body}}} + \frac{s_p}{V^{\text{supp}}}\right) + t^{\text{rec}} \max_{p \in \mathbb{P}_{k+1}^{\text{pro}}} h_p, \tag{18}$$

$$\tau_{k+1} = \begin{cases} \tau_k + t^{\text{aux}} + t_k^{\text{bld}}, & \text{if } Y_k = 1 \\ \max((k+1)u, \tau_k), & \text{if } Y_k = 0 \end{cases}, \tag{19}$$

$$\mathbf{A}_{k+1} = \{v_p, s_p, h_p, l_p, w_p, d_p, r_p, \varrho_p \mid p \in \mathbb{P}_{k+1}^{\text{ava}} \cup \mathbb{P}_{k+1}^{\text{pro}}\}. \tag{20}$$

Eq. (16) identifies the parts to be in processing at next decision epoch, $k+1$. Specifically, if the machine will not be available before the next decision epoch, the parts already in processing will remain. Otherwise, the set of parts to be

processed will be determined based on the decision variable $\mathbf{X}_k$. Eq. (17) updates the available parts at next decision epoch by removing the parts that have been processed and adding the new arrivals. Eq. (18) calculates the build time t_k^{bld} for the processed batch, based on the decision made at epoch k. Eq. (19) updates the machine availability time. This update depends on whether a batch is initiated within current time slot (i.e., $Y_k = 1$) or not (i.e., $Y_k = 0$). Eq. (20) updates the attribute set $\mathbf{A}_{k+1}$ accordingly.

Objective Function. The objective is to maximize the total expected profit over the time horizon. The profit realized between decision points k and $k+1$, denoted by the contribution function $\mathcal{V}_k(S_k, \mathbf{X}_k)$, is the net result of revenue, production costs, and tardiness penalties:

$$\mathcal{V}_k(S_k, \mathbf{X}_k) = \mathcal{R}_k - C_k^{\text{pro}} - C_k^{\text{tad}}, \tag{21}$$

$$\mathcal{R}_k = Y_k \sum_{p \in \mathbb{P}_{k+1}^{\text{pro}}} \varrho_p, \tag{22}$$

$$C_k^{\text{pro}} = Y_k \left(C^{\text{opr}} + K^{\text{bld}} t_k^{\text{bld}} + K^{\text{pow}} \sum_{p \in \mathbb{P}_{k+1}^{\text{pro}}} (v_p + s_p) \right), \tag{23}$$

$$C_k^{\text{tad}} = K^{\text{fine}} \sum_{p \in \mathbb{P}_k} T_{k,p}, \tag{24}$$

$$T_{k,p} = \begin{cases} \max\left(\tau_k - \max(d_p, ku),\ 0\right) & \text{if } \tau_k < (k+1)u, \\ \max\left((k+1)u - \max(d_p, ku),\ 0\right) & \text{if } \tau_k \geq (k+1)u \end{cases} \quad \forall p \in \mathbb{P}_k^{\text{pro}}, \tag{25}$$

$$T_{k,p} = \max\left((k+1)u - \max(d_p, ku),\ 0\right) \quad \forall p \in \mathbb{P}_k^{\text{ava}}. \tag{26}$$

Eq. (22) calculates the revenue of the batch if it is initiated. Eq. (23) calculates the production cost incurred if a batch is initiated, where $K^{\text{bld}} = K^{\text{eng}} + K^{\text{gas}}$, and K^{eng}, K^{gas}, K^{pow} are cost coefficients of energy, inert gas, and metal powder, respectively. Eq. (24) calculates the tardiness penalty incurred within current time slot $(ku, (k+1)u]$, where $T_{k,p}$ is the tardiness accrued by part p within current time slot, and K^{fine} is the penalty per unit of tardiness for each part. For each part currently in processing at current decision epoch ($p \in \mathbb{P}_k^{\text{pro}}$), it accrues a tardiness of certain length within current time slot according to its completion time τ_k and due-date d_p (Eq. (25)). For each part that is available at current decision epoch ($p \in \mathbb{P}_k^{\text{ava}}$), it accrues a tardiness of certain length within current time slot according to it due-date d_p (Eq. (26)).

Optimality Equations. The problem is to find an optimal policy π^*, which is a mapping from states to decisions, that maximizes the objective function. A policy $\pi \in \Pi$ is a sequence of decision functions, i.e., $\mathbf{X}_k = \mathbf{X}^\pi(S_k)$, where Π denotes the set of all possible policies. Consequently, the optimization problem can be expressed as:

$$\max_{\pi \in \Pi} \mathbb{E}\left\{ \sum_{k=0}^{K-1} \mathcal{V}_k(S_k, \mathbf{X}^\pi(S_k)) \mid S_0 \right\}. \tag{27}$$

An optimal policy π^* must satisfy Bellman's equations, which describe the principle of optimality for this problem

(Bellman, 1954):

$$V_k(S_k) = \max_{\mathbf{X}_k \in \mathcal{X}_k(S_k)} \left(\mathcal{V}_k(S_k, \mathbf{X}_k) + \mathbb{E}\left\{ V_{k+1}(S_{k+1}) \mid S_k, \mathbf{X}_k \right\} \right), \tag{28}$$

for all S_k and for all $k = 0, \ldots, K-1$, with $V_K(S_K) = 0$. Here, $V_k(S_k)$ is the value function, representing the maximum expected future profit from state S_k onwards. The expectation is taken with respect to the distribution of the exogenous information W_{k+1}.

3.3. Mathematical Model for Competitive Bound

To establish a competitive benchmark for the performance of any policy $\pi \in \Pi$, we formulate a deterministic MILP that assumes all information about order arrivals within time horizon ($\mathbf{W} = (W_0, \ldots, W_K)$) is known a priori. The complete formulation of this MILP and introduced linearization are detailed in Appendix A. The optimal solution to this MILP provides an offline optimal upper bound on the performance achievable by any online policy.

4. Approximate Dynamic Programming Approach

The exact resolution of the MDP model formulated in Section 3.2 via Bellman's optimality equation is computationally intractable for any non-trivial problem instance. This intractability stems from the three well-known curses of dimensionality: the exponential growth of the state space (S_k), the decision space ($\mathbf{X}_k$), and the outcome space associated with the exogenous information (W_{k+1}) (Bellman, 1957).

Approximate Dynamic Programming (ADP) provides a robust methodological paradigm to surmount these challenges. ADP encompasses a range of algorithmic strategies that seek near-optimal policies by approximating the value function, $V_k(S_k)$. A key step in many ADP methods is to manage the expectation operator in Bellman's equation by introducing the concept of a *post-decision state* (Powell, 2011).

Definition 1. *The post-decision state, denoted as S_k^x, represents the state of the system at epoch k immediately after the decision $\mathbf{X}_k$ has been executed but before the realization of the random exogenous information W_{k+1} for the subsequent interval.*

In the context of our scheduling problem, given a pre-decision state $S_k = (\mathbb{P}_k^{\text{ava}}, \mathbb{P}_k^{\text{pro}}, \tau_k, \mathbf{A}_k)$, a decision $\mathbf{X}_k$ deterministically transitions the system to a post-decision state $S_k^x = (\mathbb{P}_k^{\text{ava}} \setminus \mathbb{P}_{k+1}^{\text{pro}}, \mathbb{P}_{k+1}^{\text{pro}}, \tau_{k+1}, \mathbf{A}_k^x)$. Here, $\mathbb{P}_{k+1}^{\text{pro}}$ denotes the set of parts committed to processing, and τ_{k+1} reflects the updated machine availability. Subsequently, the stochastic arrival of new parts, $W_{k+1} = (\mathbb{P}_{k+1}^{\text{arv}}, \mathbf{A}_{k+1}^{\text{arv}})$, transitions the system from S_k^x to the next pre-decision state, S_{k+1}. This decomposition allows Bellman's equation to be computed in two recursive steps:

$$V_k(S_k) = \max_{\mathbf{X}_k \in \mathcal{X}_k(S_k)} \left(\mathcal{V}_k(S_k, \mathbf{X}_k) + V_k^x(S_k^x) \right), \tag{29}$$

$$V_k^x(S_k^x) = \mathbb{E}_{W_{k+1}} \left\{ V_{k+1}(S_{k+1}) \mid S_k^x \right\}. \tag{30}$$

The central challenge of ADP thus shifts to finding a computationally tractable approximation for the post-decision value function $V_k^x(S_k^x)$.

Given the complexity of our problem's high-dimensional state variables (combinatorial part geometries) and infinite outcome spaces (continuous arrival times), standard value function approximation methods (e.g., basis functions) are difficult to design effectively. Consequently, the Direct Lookahead Approximation (DLA), which has demonstrated success in complex stochastic domains (Truong, 2014; Yan et al., 2020; Deng & Santos, 2022), emerges as a particularly suitable strategy. A DLA avoids constructing a global approximation of the value function over the entire state space. Instead, it approximates the value of being in a specific state by explicitly simulating the consequences of decisions over a finite horizon (Powell, 2019). This approach is powerful for problems where the number of immediate decisions is manageable but the space of future random outcomes is vast, as is the case here.

Formally, our DLA policy selects a decision at state S_k by solving the following lookahead model:

$$\mathbf{X}^{\text{DLA}}(S_k) = \arg\max_{\mathbf{X}_k \in \mathcal{X}_k(S_k)} \left(\mathcal{V}_k(S_k, \mathbf{X}_k) + \mathbb{E}_{W_{k+1}} \left\{ \max_{\pi} V_{k+1}^{\pi}(S_{k+1}) \mid S_k^x \right\} \right), \tag{31}$$

where $V_{k+1}^{\pi}(S_{k+1}) = \mathbb{E}\left[\sum_{k'=k+1}^{K-1} \mathcal{V}_{k'}(S_{k'}, \mathbf{X}^{\pi}(S_{k'})) \mid S_{k+1}\right]$ represents the estimated value of state S_{k+1} under a predefined lookahead policy π. As conceptually illustrated in Figure 4, this involves constructing a state-decision *tree* rooted at the current state S_k. Each branch represents a potential decision $\mathbf{X}_k$ leading to a post-decision state S_k^x, from which various future scenarios (driven by outcomes $W_{k+1}, W_{k+2}, \ldots$) unfold under the guidance of the lookahead policy.

However, implementing the DLA for this specific problem presents three significant computational challenges:

1. **Lookahead Tree Evaluation**: The nested expectations in Eq. ((31)) imply an exploration of a vast state-decision tree that grows exponentially with the lookahead horizon. Given the infinite outcome space for new part arrivals, an exhaustive evaluation is impossible, necessitating an efficient method for sampling the tree (addressed in Section 4.1).
2. **Lookahead Policy Design**: The performance of the DLA is critically dependent on the quality of the lookahead policy π. Devising an optimal lookahead policy is as complex as solving the original problem itself, thus requiring a well-performing and computationally efficient heuristic policy $\tilde{\pi}$ (addressed in Section 4.2).
3. **High-quality Feasible Decision Space Generation**: For each state $S_{k'}$ encountered in the tree, the set of feasible decisions $\mathcal{X}_{k'}(S_{k'})$ is constrained by the NP-hard 2D nesting problem and thus grows superlinearly with the parts number. Adding the full set of feasible decisions as branches is computationally prohibitive, requiring a fast heuristic to generate a representative subset of high-quality candidate decisions (addressed in Section 4.3).

4.1. Search Framework of Lookahead Tree

To address the challenge of evaluating the exponentially growing lookahead tree, we employ Monte Carlo Tree Search (MCTS) as the core mechanism for our DLA policy. MCTS is an online, simulation-based optimization

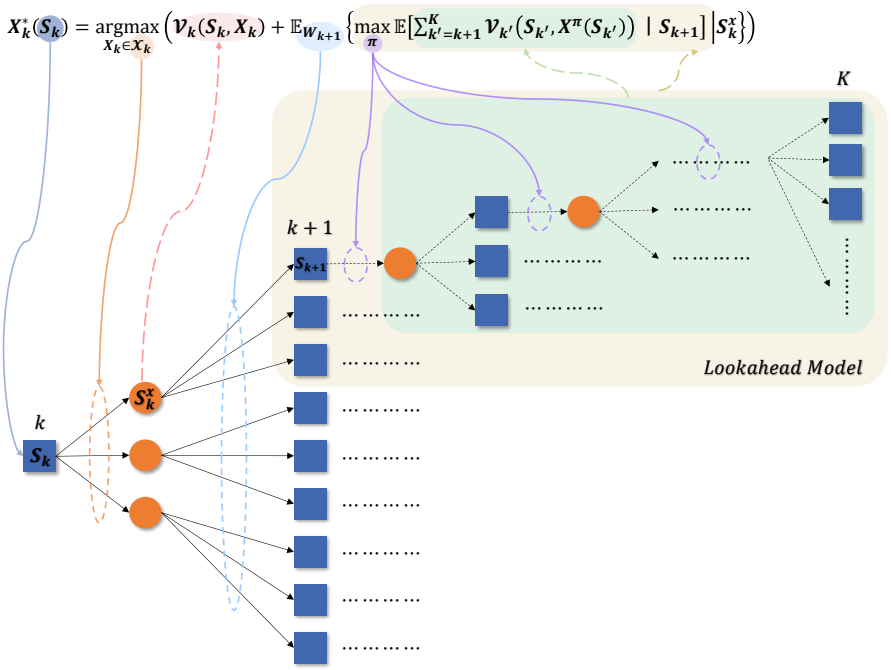


Figure 4: Conceptual illustration of the Direct Lookahead Approximation. Square nodes represent pre-decision states $S_{k'}$, and circular nodes represent post-decision states $S_{k'}^x$. A arrow from a square to a circle indicates a deterministic transition triggered by a decision, i.e., $S_{k'} \xrightarrow{\mathbf{X}_{k'}} S_{k'}^x$. A arrow from a circle to a square indicates a stochastic transition driven by exogenous information, i.e., $S_{k'}^x \xrightarrow{W_{k'+1}} S_{k'+1}$. The value of a decision is estimated by aggregating the outcomes of sample paths simulated under a lookahead policy π.

technique that builds an asymmetric search tree to intelligently sample the vast state-decision space. By focusing computational resources on high-potential branches, it provides robust estimates for the value of feasible decisions $\mathbf{X}_k \in \mathcal{X}_k(S_k)$ at the current state S_k (Kocsis & Szepesvári, 2006; Browne et al., 2012).

Our scheduling problem is formally characterized as a MDP with exogenous inputs. In this framework, the system evolution is driven by two distinct forces: deterministic endogenous decisions and stochastic exogenous information. To explicitly model this structure, our MCTS framework constructs a bipartite search tree that strictly alternates between decision-making and stochastic transitions (Powell, 2022):

1. Pre-decision State Nodes ($\tilde{S}$): These nodes represent a system state $S_{k'}$ where a decision must be made. Each node stores a visit count $N(\tilde{S})$ and the accumulated value $\tilde{V}(\tilde{S})$ from passing simulations. Naturally, the *root* node corresponds to the current actual state S_k.
2. Post-decision State Nodes ($\tilde{S}^x$): These nodes represent the deterministic state $S_{k'}^x$ reached immediately after applying a decision $\tilde{\mathbf{X}}$ to $\tilde{S}$ (i.e., $\tilde{S} \xrightarrow{\tilde{\mathbf{X}}} \tilde{S}^x$), capturing the immediate contribution $\mathcal{V}(\tilde{S}, \tilde{\mathbf{X}})$. Each node stores a visit count $N(\tilde{S}^x)$, the accumulated value $\tilde{V}^x(\tilde{S}^x)$, and sampling count $W_{\text{count}}(\tilde{S}^x)$. They serve as the branching points for stochastic transition driven by exogenous information.

As outlined in Algorithm 1, the MCTS procedure executes for a fixed computational budget of B iterations. Each

iteration involves three phases: *Tree-Traversal*, *Rollout*, and *Backpropagation*.

Tree Traversal. Starting from the *root*, the algorithm traverses the existing tree to select a *leaf* node. This traversal involves two distinct transition types corresponding to the bipartite structure of the tree (Algorithm 2):

- Decision Node Selection ($\tilde{S} \rightarrow \tilde{S}^x$): At a pre-decision node $\tilde{S}$, the algorithm selects a decision $\tilde{\mathbf{X}}$ to transition to a post-decision state $\tilde{S}^x$. If the node $\tilde{S}$ has no child node, the *Expansion* procedure is invoked. This procedure generates the pruned set of feasible decisions $\mathcal{X}'(\tilde{S})$ using the heuristic strategy detailed in Section 4.3.2, and subsequently adds a corresponding child node $\tilde{S}^x$ for each $\tilde{\mathbf{X}} \in \mathcal{X}'(\tilde{S})$. Conversely, if $\tilde{S}$ has already been expanded, the algorithm selects one of the child nodes. If any of child nodes has not been explored ($N(\tilde{S}^x) = 0$), one is selected randomly. Otherwise, the child node corresponding to the decision $\tilde{\mathbf{X}}^*$, which maximizes the UCT criterion, is selected to balance exploitation and exploration (Kocsis & Szepesvári, 2006):

$$\tilde{\mathbf{X}}^* = \arg\max_{\tilde{\mathbf{X}} \in \mathcal{X}'(\tilde{S})} \left(\bar{Q}(\tilde{S}, \tilde{\mathbf{X}}) + \theta \sqrt{\frac{2 \ln N(\tilde{S})}{N(\tilde{S}^x)}} \right) \tag{32}$$

  where θ is the exploration constant. The exploitation term, $\bar{Q}(\tilde{S}, \tilde{\mathbf{X}})$, represents the normalized estimated value of taking $\tilde{\mathbf{X}}$. This value aggregates the immediate reward and the future expected value: $Q(\tilde{S}, \tilde{\mathbf{X}}) = \mathcal{V}(\tilde{S}, \tilde{\mathbf{X}}) + \tilde{V}^x(\tilde{S}^x)/N(\tilde{S}^x)$. To handle the varying scales of rewards, we apply dynamic min-max normalization as suggested by Schrittwieser et al. (2020):

$$\bar{Q}(\tilde{S}, \tilde{\mathbf{X}}) = \frac{Q(\tilde{S}, \tilde{\mathbf{X}}) - \min_{\tilde{\mathbf{X}}' \in \mathcal{X}'(\tilde{S})} Q(\tilde{S}, \tilde{\mathbf{X}}')}{\max_{\tilde{\mathbf{X}}' \in \mathcal{X}'(\tilde{S})} Q(\tilde{S}, \tilde{\mathbf{X}}') - \min_{\tilde{\mathbf{X}}' \in \mathcal{X}'(\tilde{S})} Q(\tilde{S}, \tilde{\mathbf{X}}') + \epsilon} \tag{33}$$

  where ϵ is a small constant to prevent division by zero.

- Stochastic Outcome Sampling ($\tilde{S}^x \rightarrow \tilde{S}'$): At a post-decision node $\tilde{S}^x$, the transition to the next pre-decision state $\tilde{S}'$ is driven by the random exogenous information W. Due to the effectively infinite space of potential order arrivals, it is impossible to enumerate all possible next states. To address the continuous nature of the outcome space, we employ a progressive widening strategy governed by a threshold parameter e^{thr} (Couetoux & Doghmen, 2011). If the sampling count of the post-decision node $W_{\text{count}}(\tilde{S}^x)$, which denotes the number of times the algorithm samples a next state from this node, is lower than e^{thr}, the algorithm performs *Outcome Exploration* by sampling a new realization of W to generate the next pre-decision state $\tilde{S}'$. If this state is new and distinct, it is added to the tree. Once $W_{\text{count}}(\tilde{S}^x) \geq e^{thr}$, the algorithm transitions to *Outcome Exploitation*, selecting an existing child node $\tilde{S}'$ based on the empirical frequency of previously observed outcomes.

This process repeats until a *leaf* pre-decision state node is selected for rollout.

Rollout. From the selected leaf node, a simulation is executed to the end of the time horizon (Algorithm 3). This rollout employs a computationally efficient base policy $\tilde{\pi}$ (described in Section 4.2) to make decisions along the trajectory. The cumulative profit of this trajectory, q_{roll}, serves as an estimate of the value function.

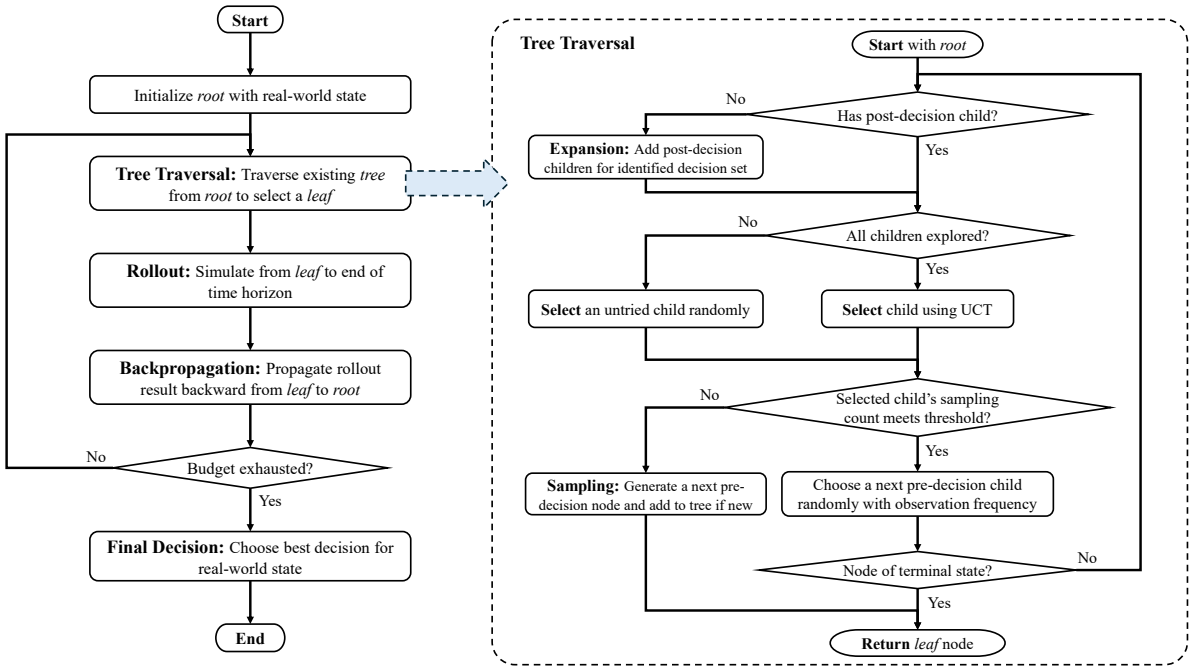


Figure 5: Flow chart of the Monte Carlo Tree Search framework with progressive widening strategy applied for stochastic transition.

Backpropagation. The simulation result is propagated backward from the leaf to the root (Algorithm 4). In a finite-horizon MDP with rewards allocated on edges, the value associated with any node reflects the cumulative rewards accrued from that point onward. To facilitate this, we maintain an accumulated profit variable q_{acc}, which is initialized with the result obtained from the rollout q_{roll}. As we ascend the tree, the following process occurs: For a pre-decision node $\tilde{S}$, its total value $\tilde{V}(\tilde{S})$ is updated with the current value of q_{acc} and its visit count $N(\tilde{S})$ is incremented. The transition from a pre-decision node to its parent post-decision node $\tilde{S}^{x}_{\text{parent}}$ involves a net contribution. This contribution is added to q_{acc} to reflect the total path value for the parent node.

Final Decision. After the budget B is exhausted, the algorithm selects the optimal decision $\mathbf{X}^{*}_{k}$ for the current real-world state S_k, by choosing the decision with the highest average value.

4.2. Design of Lookahead Policy

The performance of the DLA approach depends on the accuracy of the post-decision value estimates. The lookahead policy $\tilde{\pi}$, which governs the decision-making process within *Rollout* simulations, serves as a surrogate for the optimal policy. Since the true optimal policy is computationally intractable to identify, and the simulation is performed thousands of times within a single MCTS, the design of $\tilde{\pi}$ necessitates a careful trade-off between solution quality and computational efficiency (Bertsekas & Castanon, 1999; Powell, 2022).

We propose a composite heuristic policy that addresses the decision-making process at two levels: (1) a temporal decision on whether to initiate production, and (2) a combinatorial decision on which parts to select if production is

Algorithm 1 Monte Carlo Tree Search

Input: Initial state S_k, simulation budget B, time horizon K, stochastic sample threshold e^{thr}

Output: Optimal decision $\mathbf{X}_k^*$

Tree Structure:

$\tilde{S}$: Pre-decision state node. Stores $N(\tilde{S})$, $\tilde{V}(\tilde{S})$, children $\rightarrow \{\tilde{\mathbf{X}} \mapsto \tilde{S}^x\}$, sampling weight w_{sample}.

$\tilde{S}^x$: Post-decision state node. Stores $N(\tilde{S}^x)$, $\tilde{V}^x(\tilde{S}^x)$, children $\rightarrow \Omega(\tilde{S}^x) = \{\tilde{S}'_1, \tilde{S}'_2, ...\}$, sampling count W_{count}

procedure MCTS(S_k)
  $\tilde{S}_{\text{root}} \leftarrow S_k$
  for $i \leftarrow 1$ to B **do**
    $\tilde{S}_L \leftarrow \textsc{TreeTraversal}(\tilde{S}_{\text{root}})$ ▷ Traverse the existing tree to select a *leaf* node
    $q_{\text{roll}} \leftarrow \textsc{Rollout}(\tilde{S}_L)$ ▷ Simulate from the *leaf* to the end of time horizon
    $\textsc{Backpropagation}(\tilde{S}_L, q_{\text{roll}})$ ▷ Propagate simulation result backward from the *leaf* to the *root*
  end for
  $\mathbf{X}_k^* \leftarrow \arg\max_{\tilde{\mathbf{X}} \in \tilde{S}_{\text{root}}.\text{children}} \left(\mathcal{V}_k + \frac{\tilde{V}^x(\tilde{S}_k^x)}{N(\tilde{S}_k^x)} \right)$ ▷ Select best decision based on average value
  return $\mathbf{X}_k^*$
end procedure

initiated.

Temporal Decision: One-Step Lookahead Greedy Policy. At any decision epoch k' within a rollout, certain conditions may force postponement (i.e., if $\tau_{k'} \geq (k'+1)u$ or $\mathbb{P}_{k'}^{\text{ava}} = \emptyset$). In other cases, we employ a one-step lookahead greedy policy to determine the decision $Y_{k'}$. Instead of relying on static rules, this policy explicitly compares the immediate value of production against the potential value of a one-step postponement. This mechanism dynamically quantifies the opportunity cost of postponing, thereby aligning myopic decisions with longer-term objectives (Wang et al., 2014). Formally, the policy evaluates two quantities:

1. Estimated Reward of Immediate Production ($\hat{R}_{k'}^{\text{prod}}$): This estimates the maximum net profit achievable by initiating a batch at the current epoch k', which theoretically necessitates the resolution of an optimization problem to identify the optimal part combination that maximizes net profit. However, due to the computational intensity associated with repeatedly solving this within simulation frameworks, a fast part selection heuristic `HeuSelect` (Algorithm B.1) is employed:

$$\hat{R}_{k'}^{\text{prod}} = \mathcal{V}_{k'}\left(S_{k'}, (Y_{k'} = 1, \texttt{HeuSelect}(\mathbb{P}_{k'}^{\text{ava}}))\right). \tag{34}$$

2. Estimated Reward of One-Step Postponement then Production ($\hat{R}_{k'}^{\text{post}}$): This estimates the value of postponing the production decision to the next epoch $k'+1$. It is composed of the immediate cost of postponing and the expected profit of the best batch that could be formed in the next epoch. To estimate the future profit, we

Algorithm 2 Tree Traversal

procedure TREE TRAVERSAL($\tilde{S}$)
  while $\tilde{S}$.epoch $< K$ **do** ▷ While state is non-terminal
    if $\tilde{S}$.children is empty **then** ▷ Expansion Phase
      EXPANSION($\tilde{S}$)
    end if
    untried $\leftarrow \{\tilde{S}^x \in \tilde{S}.\text{children} \mid N(\tilde{S}^x) = 0\}$
    if untried is not empty **then** ▷ Selection Phase
      $\tilde{S}^x \leftarrow$ RandomChoice(untried)
    else
      $\tilde{S}^x \leftarrow \arg\max_{\tilde{S}^x \in \tilde{S}.\text{children}} \left(\bar{Q} + \theta \sqrt{\frac{2 \ln N(\tilde{S})}{N(\tilde{S}^x)}}\right)$ ▷ Select using UCT
    end if
    if $\tilde{S}^x.W_{\text{count}} < e^{thr}$ **then** ▷ Outcome Exploration
      $\tilde{S}^x.W_{\text{count}} \leftarrow \tilde{S}^x.W_{\text{count}} + 1$
      $\tilde{S}_{\text{leaf}} \leftarrow$ SAMPLING($\tilde{S}^x$) ▷ Sample outcome
      return $\tilde{S}_{\text{leaf}}$ ▷ Return *leaf* node
    else ▷ Outcome Exploitation
      $\tilde{S} \leftarrow$ WeightedRandomChoice($\Omega(\tilde{S}^x)$, [$\tilde{S}'.w_{\text{sample}}$ for $\tilde{S}' \in \Omega$])
    end if
  end while
  return $\tilde{S}$ ▷ Return terminal state
end procedure

procedure EXPANSION($\tilde{S}$)
  $\mathcal{X}' \leftarrow$ HeuristicDecisionPruning($\tilde{S}$) ▷ See Section 4.3.2
  for $\tilde{\mathbf{X}} \in \mathcal{X}'$ **do**
    $\tilde{S}^x \leftarrow S^{M,x}(\tilde{S}, \tilde{\mathbf{X}})$ ▷ Apply decision $\tilde{\mathbf{X}}$ to $\tilde{S}$
    $\tilde{S}.\text{children}[\tilde{\mathbf{X}}] \leftarrow \tilde{S}^x$
  end for
end procedure

procedure SAMPLING($\tilde{S}^x$)
  $\tilde{S}' \leftarrow S^M(\tilde{S}^x, \tilde{W}_{k+1})$ ▷ Sample exogenous information
  if $\tilde{S}'$ already exists in $\Omega(\tilde{S}^x)$ **then**
    $\tilde{S}'.w_{\text{sample}} \leftarrow \tilde{S}'.w_{\text{sample}} + 1$
  else
    $\Omega(\tilde{S}^x) \leftarrow \Omega(\tilde{S}^x) \cup \{\tilde{S}'\}$
    $\tilde{S}'.w_{\text{sample}} \leftarrow 1$
  end if
  return $\tilde{S}'$
end procedure

Algorithm 3 Rollout

procedure ROLLOUT($\tilde{S}_{\text{roll}}$)
  $q_{\text{roll}} \leftarrow 0$
  while $\tilde{S}_{\text{roll}}$.epoch $< K$ **do**
    $\mathbf{X} \leftarrow \tilde{\mathbf{X}}^{\tilde{\pi}}(\tilde{S}_{\text{roll}})$ ▷ Decision by lookahead policy $\tilde{\pi}$
    $q_{\text{roll}} \leftarrow q_{\text{roll}} + \mathcal{V}(\tilde{S}_{\text{roll}}, \mathbf{X})$ ▷ Accumulate net profit
    $\tilde{S}_{\text{roll}} \leftarrow S^M(\tilde{S}_{\text{roll}}, \mathbf{X}, \tilde{W}_{\text{roll}})$ ▷ State transition
  end while
  return q_{roll} ▷ Accumulated value
end procedure

simulate a single step forward by sampling a set of random new arrivals, $\tilde{\mathbb{P}}^{\text{arv}}_{k'+1}$, merging them with the currently available parts to form a hypothetical future pool $\tilde{\mathbb{P}}^{\text{ava}}_{k'+1}$. Similarly, the optimal part selection is approximated using `HeuSelect` procedure:

$$\hat{R}^{\text{post}}_{k'} = \mathcal{V}_{k'}\left(S_{k'}, (Y_{k'} = 0, \{X_{k',p} = 0\}_{p\in\mathbb{P}^{\text{ava}}_{k'}})\right) + \left[\mathcal{R}_{k'+1} - C^{\text{pro}}_{k'+1}\right]_{\left(Y_{k'+1}=1, \texttt{HeuSelect}(\tilde{\mathbb{P}}^{\text{ava}}_{k'+1})\right)}, \tag{35}$$

Note that for a fair comparison of these two quantities, tardiness costs in the hypothetical future state are excluded from $\hat{R}^{\text{post}}_{k'}$.

The policy $\tilde{\pi}$ chooses to postpone if $\hat{R}^{\text{post}}_{k'} \geq \hat{R}^{\text{prod}}_{k'}$; otherwise, it makes the decision of producing with the selected batch.

Combinatorial Decision: Generate-then-Repair Heuristic. Within the rollout simulations, efficiently determining the optimal batch configuration is critical. Since exact resolution is computationally prohibitive, we employ a fast generate-then-repair heuristic, denoted as `HeuSelect`. This heuristic operates on a greedy principle, prioritizing parts with high "profit density" (revenue per unit of projected area) to construct a greedy candidate batch as the area constraint is obeyed. If this candidate violates the nesting constraints, a repair mechanism iteratively removes the part with the lowest profit density until feasibility is restored. This approach strikes a balance between batch quality and computational speed. The detailed algorithmic procedure are provided in Appendix B.

4.3. Approximation and Pruning of Decision Space

A core computational challenge in our ADP approach is the generation and management of the feasible decision space, $\mathcal{X}_{k'}(S_{k'})$, at each state $S_{k'}$ encountered. While trivial states (machine busy or queue empty) lead to a "postpone" decision, obtaining the decision space of other states requires identifying the set of all combinatorially feasible "produce" batches, i.e., subsets $\{X_{k',p}\}_{p\in\mathbb{P}^{\text{ava}}_{k'}}$ that satisfy the complex 2D nesting constraints.

Algorithm 4 Backpropagation

procedure BACKPROPAGATION($\tilde{S}_{\text{leaf}}, q_{\text{roll}}$)
  $\tilde{S}_{\text{curr}} \leftarrow \tilde{S}_{\text{leaf}}$
  $q_{\text{acc}} \leftarrow q_{\text{roll}}$ ▷ Initialize accumulated profit with rollout value
  while $\tilde{S}_{\text{curr}} \neq$ null **do**
    $\tilde{S}_{\text{curr}}.N \leftarrow \tilde{S}_{\text{curr}}.N + 1$ ▷ Update current pre-decision node statistics
    $\tilde{S}_{\text{curr}}.\tilde{V} \leftarrow \tilde{S}_{\text{curr}}.\tilde{V} + q_{\text{acc}}$
    if $\tilde{S}_{\text{curr}} == \tilde{S}_{\text{root}}$ **then**
      break
    end if
    $\tilde{S}^{x}_{\text{parent}} \leftarrow \tilde{S}_{\text{curr}}.\text{parent}$ ▷ Move up to parent post-decision node
    $\tilde{S}^{x}_{\text{parent}}.N \leftarrow \tilde{S}^{x}_{\text{parent}}.N + 1$ ▷ Update post-decision node statistics
    $\tilde{S}^{x}_{\text{parent}}.\tilde{V} \leftarrow \tilde{S}^{x}_{\text{parent}}.\tilde{V} + q_{\text{acc}}$
    $q_{\text{acc}} \leftarrow q_{\text{acc}} + \tilde{S}^{x}_{\text{parent}}.\mathcal{V}$ ▷ Add net contribution of the decision leading to this post-decision node
    $\tilde{S}_{\text{curr}} \leftarrow \tilde{S}^{x}_{\text{parent}}.\text{parent}$ ▷ Move up to grand-parent pre-decision node
  end while
end procedure

4.3.1. Approximating Feasible Decision Space

The exact identification of all feasible "produce" decisions necessitates verifying the nesting feasibility for every non-empty subset of the available parts, $\mathbb{P}^{\text{ava}}_{k'}$. Mapping parts to their minimal bounding boxes transforms this verification into a variant of the 2D bin packing problem, which is known to be NP-hard (Gilmore & Gomory, 1965). Formally, this problem can be modeled as an MILP that seeks a valid placement. However, solving this MILP repeatedly for $2^{|\mathbb{P}^{\text{ava}}_{k'}|} - 1$ subsets to construct the exact decision space is computationally prohibitive.

To overcome this bottleneck, particularly within the thousands of states evaluated in MCTS, we propose a heuristic framework to obtain a high-quality approximation of the feasible decision space. The framework is composed of two main components: a fast best-fit based heuristic for feasibility check of a single combination, and a top-down search strategy to traverse all combinations.

Fast Feasibility Check Heuristic. The nesting feasibility of a part combination is checked using a *skyline*-based best-fit heuristic (Algorithm 5), a standard and efficient method for 2D packing (Burke, Kendall, & Whitwell, 2004). This algorithm maintains the *skylines*, $\mathcal{S}$, a set of horizontal segments describing the current upper contour of the packed parts. Each skyline $s \in \mathcal{S}$ is denoted by a tuple (x, y, w), where (x, y) are the coordinates of its left-end point and w is its width. With the platform formulated as an empty bin ($\mathcal{L} \times \mathcal{W}$), where its length makes an initial skyline $(0, 0, \mathcal{L})$, the heuristic iteratively attempts to place the best-fitting part onto the lowest skyline. The best fitting part

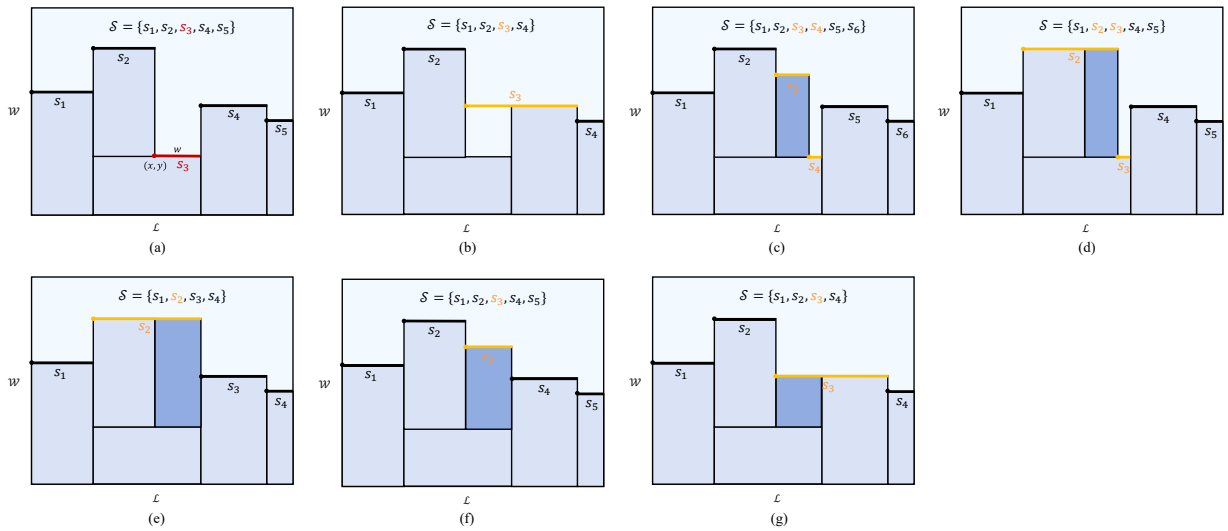


Figure 6: Schematic illustration of the skyline-based best-fit heuristic operations. (a) The initial skyline profile $\mathcal{S}$, identifying s_3 as the lowest skyline. (b) The *Lift* strategy: triggered when the lowest segment cannot accommodate any available part, its height is raised to match the lower neighbor, merging them to eliminate the unusable void. (c)–(d) Part placement where the part dimension is less than the skyline width ($w_p < w_{s_3}$), resulting in (c) a fragmented skyline update or (d) a left-merge if the part's top edge aligns vertically with the left neighbor s_2. (e)–(g) Part placement where the part dimension equals the skyline width ($w_p = w_{s_3}$), resulting in (e) a left-merge given height alignment with the left neighbor, (f) a simple update, or (g) a right-merge given height alignment with the right neighbor.

is determined by minimizing the wasted horizontal space. If a part is placed, the skylines are updated; if no part fits, the lowest skyline is lifted and merged with its neighbor. Figure 6 illustrates these operations. This process continues until all parts in the combination are packed. Eventually, the combination is checked feasible if the highest skyline's y coordinate does not exceed the platform's width, W.

Top-Down Search Strategy. The feasibility checking procedure is organized as a top-down search exploiting the downward-closure property of feasible part sets (Algorithm 6). The search begins with the largest candidate batch and progressively considers smaller subsets. Whenever a batch is identified as feasible, all of its non-empty subsets are immediately classified as feasible and excluded from further checks, resulting in substantial pruning of the search space. This mechanism is illustrated in Figure 7, where identifying a feasible node at a higher level automatically validates all of its descendant nodes.

4.3.2. Pruning Feasible Decision Space

While the heuristic method described in Section 4.3.1 efficiently identifies a set of feasible "produce" decisions, simply adding all these options into the search tree presents a computational challenge. As illustrated in Figure C.1, the number of feasible batches grows superlinearly with the number of available parts. Adding a child post-decision state node for every feasible "produce" decision to the search tree would result in an excessively large branching

Algorithm 5 Best-Fit Feasibility Check

procedure FEASIBILITYCHECK($\mathbb{P}_{\text{sub}}, \mathcal{L}, \mathcal{W}$)
  $\mathcal{S} \leftarrow \{(0, 0, \mathcal{L})\}$ ▷ Initialize skyline list: (x, y, width)
  $\mathbb{P}_{\text{unpacked}} \leftarrow \mathbb{P}_{\text{sub}}$
  while $\mathbb{P}_{\text{unpacked}} \neq \emptyset$ **do**
    Find skyline $s \in \mathcal{S}$ with minimum y-coordinate
    Find part $p^* \in \mathbb{P}_{\text{unpacked}}$ and orientation o^* that fits on s best
    if no part fits s **then** ▷ Lift strategy
      Identify neighbors s_l (left) and s_r (right) of s
      if $s_l.y \leq s_r.y$ **then** ▷ Merge with lower neighbor
        $s_l.w \leftarrow s_l.w + s.w$
      else
        $s_r.w \leftarrow s_r.w + s.w$
        $s_r.x \leftarrow s.x$
      end if
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{s\}$
    else ▷ Place part
      Place p^* at $(s.x, s.y)$ with orientation o^*
      Update $\mathcal{S}$ to reflect new upper contour
      $\mathbb{P}_{\text{unpacked}} \leftarrow \mathbb{P}_{\text{unpacked}} \setminus \{p^*\}$
    end if
    if any skyline in $\mathcal{S}$ has $y > \mathcal{W}$ **then return** FALSE
    end if
  end while
  return TRUE
end procedure

factor. This phenomenon, often termed the "curse of width," would dilute the fixed simulation budget across too many shallow branches, preventing the tree search from reaching sufficient depth to accurately evaluate the long-term consequences of decisions (Chaslot et al., 2008).

Therefore, a strategic pruning mechanism is essential. We need to filter the exhaustive set of feasible decisions down to a concise, high-potential subset that is likely to contain the optimal decision. To achieve this without sacrificing solution quality, we adopt a multi-objective perspective. We posit that a high-quality batching decision should strike a balance between maximizing immediate economic return and minimizing resource consumption.

Specifically, we evaluate each feasible "produce" decision $\mathbf{X}_k$ based on two competing but commensurable objectives:

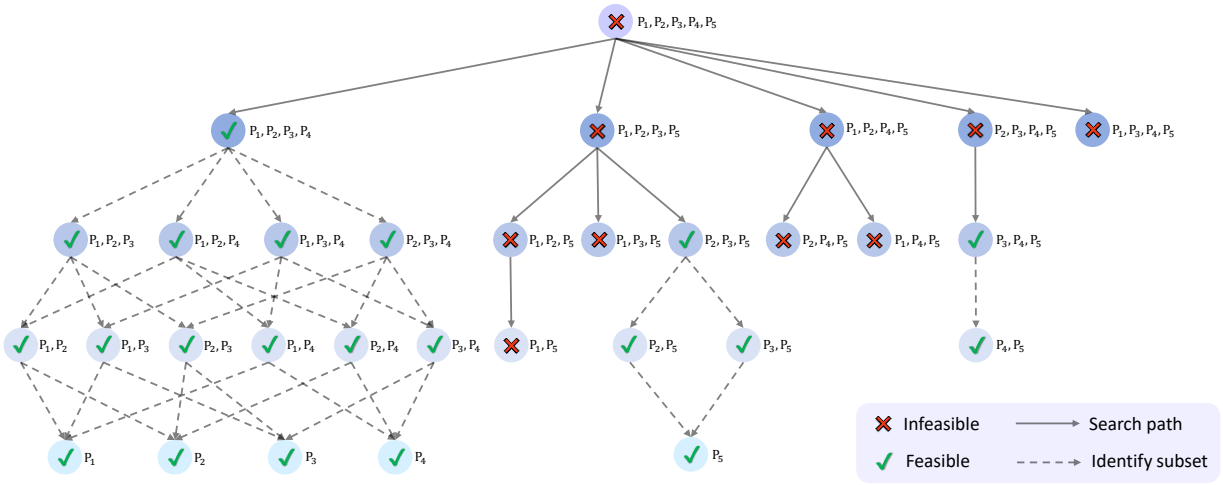


Figure 7: Illustration of the Top-Down Search strategy. The search proceeds from larger to smaller batch sizes. Validating a parent node (e.g., $\{P_1, P_2, P_3, P_4\}$) implicitly validates all its subsets (dashed lines), allowing the algorithm to skip redundant feasibility checks.

1. Immediate Net Profit (Maximize):

$$f_1(\mathbf{X}_k) = \mathcal{V}_k(S_k, \mathbf{X}_k) \tag{36}$$

This metric, consistent with the MDP's cotribution function, captures the direct economic contribution of the batch. It accounts for the revenue of processed parts minus the production costs and any accrued tardiness penalties within the time slot. Maximizing this objective prioritizes myopically profitable decisions.

2. Average Production Cost per Part (Minimize):

$$f_2(\mathbf{X}_k) = \frac{C_k^{\text{pro}}}{|\{p \mid X_{k,p} = 1\}|} \tag{37}$$

This metric serves as a proxy for production efficiency and nesting density. A lower average cost implies that the fixed operator and setup costs are better amortized across the batch. Crucially, high efficiency often correlates with better resource utilization, which implicitly preserves capability for future arrivals. Minimizing this objective prioritizes strategically efficient decisions.

Instead of aggregating these objectives into a single scalar using arbitrary weights, we employ a Pareto-based filtering strategy. A decision $\mathbf{X}$ is said to dominate another decision $\mathbf{X}'$ if it yields a higher or equal immediate profit ($f_1(\mathbf{X}) \geq f_1(\mathbf{X}')$) and a lower or equal average cost ($f_2(\mathbf{X}) \leq f_2(\mathbf{X}')$), with at least one inequality being strict. We identify the set of non-dominated decisions (the Pareto frontier) from the pool of all feasible "produce" decisions. The final pruned decision space $\mathcal{X}'(S_k)$ consists of: (1) the single "Postpone" decision, (2) the set of non-dominated "Produce" decisions. Figure 8 visualizes an example of this pruning strategy. The scatter plot shows feasible batches mapped into the objective space. The Pareto frontier (highlighted points) captures the best trade-offs between high-profitable batches with cost-efficient batches. By discarding the dominated solutions, we significantly reduce the

Algorithm 6 Top-Down Search for Feasible Decision Space

procedure FINDFEASIBLECOMBINATIONS($\mathbb{P}^{ava}, \mathcal{L}, \mathcal{W}$)
  $C_{feasible} \leftarrow \emptyset$ ▷ Set of feasible part combinations
  for $n \leftarrow |\mathbb{P}^{ava}|$ down to 1 **do**
    for each combination $\mathbb{P}_{sub}$ of size n from $\mathbb{P}^{ava}$ **do**
      if any superset of $\mathbb{P}_{sub}$ is in $C_{feasible}$ **then**
        continue ▷ Already covered by a larger feasible batch
      end if
      if FEASIBILITYCHECK($\mathbb{P}_{sub}, \mathcal{L}, \mathcal{W}$) **then**
        Add $\mathbb{P}_{sub}$ and all its non-empty subsets to $C_{feasible}$
      end if
    end for
  end for
  return $C_{feasible}$
end procedure

search space size while retaining the most promising candidates for both short-term profit and long-term efficiency.

5. Numerical Results

This section presents a comprehensive evaluation of the proposed ADP-based dynamic scheduling approach. The computational experiments are systematically designed to achieve the following objectives:

1. Quantify the economic value of proactive postponement by comparing the proposed policy with a reactive baseline (Section 5.2).
2. Demonstrate the superiority of the lookahead mechanism over parameter-dependent static rule-based policies (Section 5.3).
3. Validate the effectiveness of the heuristic decision space pruning in reducing computational complexity without compromising solution quality (Section 5.4).
4. Benchmark the performance of our proposed online policy against the offline optimum obtained by solving the deterministic MILP model (Section 5.5).

All algorithms were implemented in Python 3.9 and executed on a workstation equipped with an Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz and 128GB of RAM. The MILP models, serving as the competitive bound, were solved using Gurobi Optimizer 12.0.3. The source code, tested instances, comprehensive experimental results, and an interactive simulator associated with this study are openly accessible at the following website: https://haowutongji.github.io/Dynamic_AM_Postpone_web.

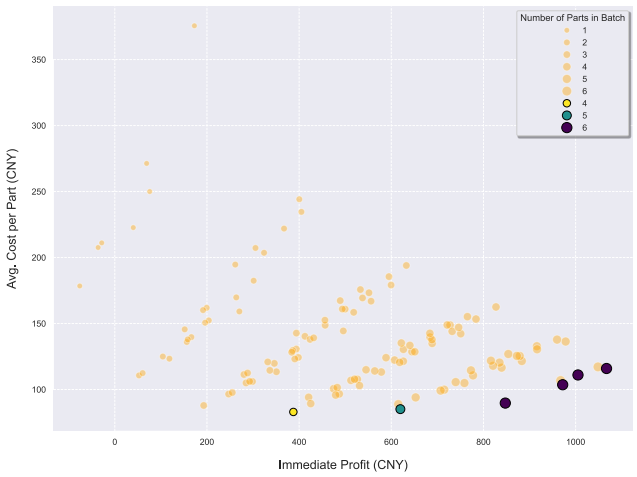


Figure 8: An example of Pareto frontier analysis for the "Produce" decision space with seven available parts. Each point represents a feasible batch. The highlighted points on the lower-right boundary constitute the Pareto frontier, representing the most efficient trade-offs between maximizing immediate profit and minimizing average cost.

5.1. Experimental Setup

The performance of the scheduling policies is evaluated by simulating their operation over a finite time horizon T_{horizon}. The primary performance metric is the total net profit accrued, aligning with the MDP objective function.

Simulation Environment. For each experimental test, a sample path of exogenous information $\mathbf{W}(\omega)$, is required for its simulation environment. The stochastic arrival of parts is modeled as a homogeneous Poisson process. To capture the geometric heterogeneity inherent in AM parts, they are classified into six types based on two height categories ($i \in \{\text{Low}, \text{High}\}$) and three projected bounding box categories ($j \in \{\text{Small}, \text{Long}, \text{Large}\}$). The arrival process for each type (i, j) is independent with a constant rate λ_{ij}, specified by the matrix $\mathbf{\Lambda}$:

$$\mathbf{\Lambda} = \begin{pmatrix} \lambda_{\text{Low,Small}} & \lambda_{\text{Low,Long}} & \lambda_{\text{Low,Large}} \\ \lambda_{\text{High,Small}} & \lambda_{\text{High,Long}} & \lambda_{\text{High,Large}} \end{pmatrix}. \tag{38}$$

To avoid cold-start bias, the simulation is initialized with the machine availability time $\tau_0 = 0$ and a non-empty queue $\mathbb{P}_0$. Upon arrival, each part is assigned specific geometric dimensions and a due date. The price of each order is determined by a model considering production cost and risk for tardiness penalties. Detailed parameter ranges and probability distributions for part attributes, due date assignment and pricing logic are provided in Appendix D.

Testbed Instances. Testbed instances are defined by a combination of internal system parameters, which reflect the machine and cost structure, and external environmental parameters, which describe the operational context. As shown in Table E.1, internal system parameters are assigned fixed values based on common production realities. Specifically, the machine parameters are informed by the specifications of the Sisma EVEMET 200 SLM system, and the cost

coefficients are derived from prevailing industrial rates in China. The external parameters are varied to construct a diverse set of test environments. Therefore, the testbed is constructed using a full factorial design based on:

- Time Horizon: 3 levels (36, 72, and 144 hours), representing short, medium, and long-term time horizons.
- Arrival Pattern: 3 scenarios (*Uniform*, *Small-dominant*, *Large-dominant*), representing balanced demand and demand skewed towards specific part bounding boxes.

This results in 9 unique configurations. For each configuration, 3 instances are generated with different sample path ω of exogenous information. The performance of the proposed approach is evaluated against benchmark policies and an offline optimal bound across this testbed of 27 instances. Each instance is identified by a code, e.g., '*H72-Large-S3*', which corresponds to a 72-hour horizon, Large-dominant arrival pattern, and the third sample path of exogenous information.

Parameter Settings. The proposed approach relies on several critical algorithmic parameters that can be classified into two categories: trade-off parameters and tunable parameters. The trade-off parameters are designed to strike a balance between solution quality and computational efficiency, while the tunable parameters necessitate experimental calibration to optimize performance. For trade-off parameters, based on preliminary tests, we set the simulation budget per decision B to 3000 and the decision epoch length u to 1 hour to balance solution quality with computational feasibility. For tunable parameters, we conduct an extensive calibration study utilizing six representative instances. This study finds out that, an exploration constant θ of 10 in the UCT equation, combined with an sampling threshold of stochastic outcomes e^{thr} of 5, provides more robust performance. This calibration of parameters is detailed in Appendix F.

5.2. Value of Strategic Postponement

A core contribution of this research is the integration of strategic postponement into the dynamic AM scheduling framework. To quantify the value of incorporating postponement as an active decision, we compare our approach with the Process while Available (PwA) policy. PwA represents a reactive, myopic strategy, processing orders immediately whenever feasible. As detailed in Algorithm G.1, at each decision epoch, this policy initiates a new batch as soon as two conditions are met: the machine is available, and there is at least one part in the queue. The decision of which parts to include is made myopically by generating all feasible part combinations (Algorithm 6) and selecting the one that yields the highest immediate net profit.

The comparative results, summarized in Table 1 and visualized in Figure 9, provide compelling evidence for the economic value of strategic postponement. Here, to evaluate the performance of a policy across different instances, the Relative Deviation Index (RDI) is applied to normalize the total accrued net profit $\text{Obj} = \sum_{k=0}^{K-1} \mathcal{V}_k$ of a policy against the best (Obj_{best}) and worst ($\text{Obj}_{\text{worst}}$) performance observed for that instance:

$$\text{RDI} = \frac{\text{Obj} - \text{Obj}_{\text{worst}}}{\text{Obj}_{\text{best}} - \text{Obj}_{\text{worst}}} \times 100\%. \tag{39}$$

Table 1: Summary of main experimental results comparing the proposed ADP approach with baseline policies. Columns 'avg.', 'max', and 'min' for 'Our Approach' report statistics calculated over 5 independent stochastic simulation runs. The baselines include: Process while Available (PwA), Process with Waiting Buffer (PWB) with varying buffer times T_{buffer}, and Process with Capacity Rule (PCR) with varying area utilization thresholds η.

T_{horizon}	Arrival patt.	Ins.	**Our Approach**			PwA	PWB				PCR			
			avg.	max	min		T_{buffer}=3	T_{buffer}=6	T_{buffer}=9	T_{buffer}=12	η=20%	η=40%	η=60%	η=80%
36	Uniform	#1	**981.24**	1198.19	836.60	-137.82	-137.82	-137.82	302.89	-15.62	664.48	-87.89	445.81	-1123.32
		#2	**1318.95**	1662.19	1090.13	365.63	736.42	1058.13	1198.64	271.43	1083.89	1165.39	492.93	487.10
		#3	**2164.09**	2263.21	2139.31	1251.19	1164.96	1786.38	939.30	932.20	1616.62	1530.40	2147.78	1354.87
	Small-dom.	#1	507.99	765.39	403.79	-334.52	-248.36	121.90	121.90	109.86	**765.39**	-182.17	-1123.32	-1123.32
		#2	**1052.97**	1182.13	859.21	-440.71	493.52	252.74	846.86	329.49	859.21	493.28	480.54	-775.26
		#3	707.51	874.88	665.67	-223.80	312.16	633.68	719.03	271.43	**763.63**	375.97	157.43	-1230.41
	Large-dom.	#1	**990.77**	1213.47	847.93	477.07	534.79	847.93	448.13	358.13	610.36	852.21	130.81	443.95
		#2	**1821.08**	1966.60	1602.80	942.38	942.38	1287.70	479.03	209.03	942.38	962.77	1025.31	233.89
		#3	**1312.42**	1316.57	1309.65	572.98	944.22	941.15	485.65	883.71	938.41	1309.65	899.27	918.07
72	Uniform	#1	1964.71	2106.36	1782.61	1018.35	1278.19	1771.92	847.41	821.34	1836.50	**2065.82**	1056.08	-922.20
		#2	**2666.52**	3001.02	2413.35	1812.81	2346.75	2620.04	2192.21	2461.98	2264.43	2305.61	2480.39	-1886.12
		#3	**2706.70**	2926.30	2565.54	1478.40	2202.65	1925.47	1910.38	1360.29	2045.65	2509.81	708.97	-1064.83
	Small-dom.	#1	**1613.88**	1883.77	1068.20	966.67	498.41	1458.37	847.46	1370.93	860.89	885.82	-2700.94	-4959.47
		#2	2444.28	3018.59	1941.28	889.04	1214.04	1824.19	1134.39	941.17	2843.48	**2893.47**	-406.51	157.22
		#3	**2651.13**	2794.08	2516.17	1084.25	2060.05	1858.81	2408.71	1418.12	2557.43	2204.76	1862.58	-455.44
	Large-dom.	#1	3859.66	4079.54	3713.64	3157.60	3116.46	3145.54	1939.29	3089.08	3531.75	3994.73	**4114.86**	3047.16
		#2	**2960.75**	3124.67	2725.29	2058.76	2510.34	2829.24	2601.75	2503.79	2505.42	2591.01	2409.44	-227.88
		#3	**2512.88**	2850.32	2179.90	703.42	2006.03	1695.34	1119.19	1793.10	1791.04	1922.66	-30.99	-417.77
144	Uniform	#1	**6717.85**	6975.03	6550.04	5546.59	6276.12	5615.07	4475.73	-334.78	6099.66	5852.83	5763.85	1185.76
		#2	**6753.35**	6913.55	6629.53	5938.63	6240.90	4845.26	3679.85	2987.86	6283.02	5909.04	5025.98	3120.89
		#3	**4682.94**	5055.90	4534.04	3454.02	3778.71	3622.25	3597.49	2129.13	4346.61	3521.62	1066.90	-2851.09
	Small-dom.	#1	**4248.13**	4368.25	4082.72	2635.30	3646.24	3299.82	3336.28	1881.13	3914.53	1383.61	343.74	-1539.53
		#2	**5671.94**	5976.12	5427.38	4453.40	5074.35	4139.89	3493.36	3759.81	4197.83	4653.13	1081.92	-8433.62
		#3	**3406.09**	3712.73	3287.87	1189.65	2571.24	3121.10	2551.32	2197.40	2995.89	3375.27	-1846.08	-5143.31
	Large-dom.	#1	**5507.14**	5692.58	5283.38	3860.78	5363.68	4926.08	2384.79	1325.90	5340.78	5208.57	2997.47	629.27
		#2	**8286.35**	8548.77	8057.49	6974.21	7930.77	3963.07	1820.02	-5784.13	7648.54	7397.60	7439.49	3974.42
		#3	**5071.76**	5610.88	4726.62	3404.26	4515.25	4771.70	1691.47	1969.35	5044.92	4727.73	1832.32	-1701.43
	Avg. Val.		**3132.71**			1978.65	2521.67	2378.70	1761.95	1083.00	2753.81	2586.03	1402.07	-677.87
	Avg. RDI		**92.56%**			55.44%	69.94%	72.76%	56.74%	46.70%	80.31%	77.32%	51.94%	10.71%

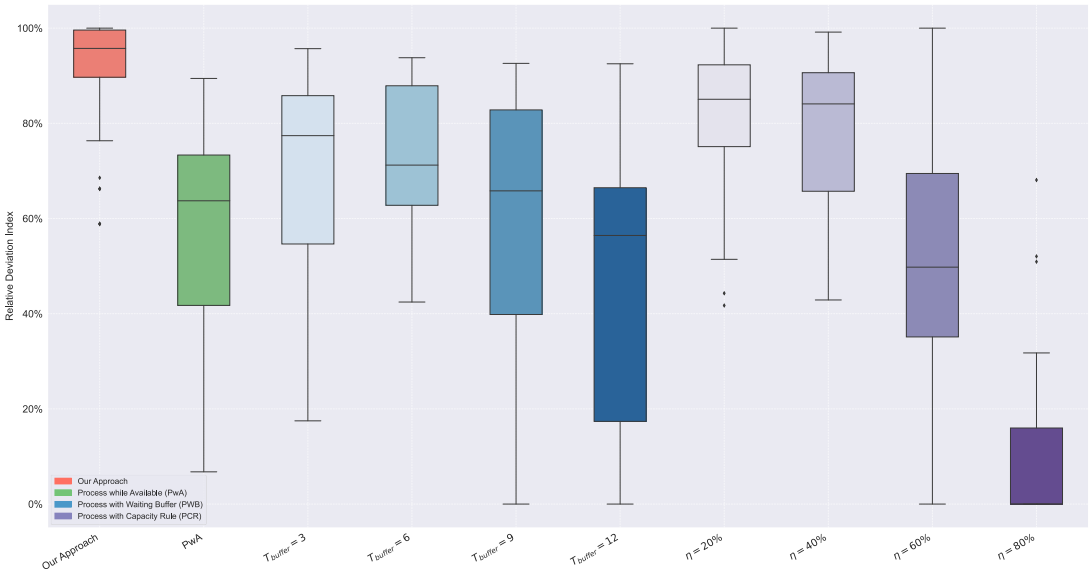


Figure 9: Boxplot of Relative Deviation Index for all compared policies across the testbed.

As shown, the proposed ADP-based approach consistently dominates the PwA policy across all test instances. Our approach achieves an average RDI of 92.56%, maintaining near-optimal performance regardless of the horizon length or arrival pattern. In contrast, the PwA policy exhibits significantly inferior performance (average RDI 55.44%) and, more importantly, high volatility (ranging from 6.77% to 89.43%). Moreover, Figure 10(a) visualizes the profit trajectories comparison for a representative instance 'H72-Large-S3' between our approach with PwA. The PwA policy (blue line) processes batches with low-profit frequently. In contrast, our approach (orange line) utilizes periods of active waiting to enable the sharp vertical jumps of the trajectory that represent the initiation of highly consolidated, profitable batches.

Accordingly, we provide some analysis. The instability of the PwA policy reveals a fundamental backward in reactive scheduling for AM: it tightly couples production decisions with the stochasticity of the arrival process. Because PwA commits to a batch immediately upon machine availability, its performance is entirely dependent on the current queue composition. If the available parts make an insufficient utilization of building platform, PwA may force an inefficient batch, incurring high fixed operator and recoating costs per part. This explains the underlying reason of particularly poor performance in Small-dominant scenarios, where the machine typically requires intentionally waiting to accumulate parts for efficient nesting. However, the proposed policy utilizes strategic postponement as a buffer to decouple production timing from order arrival times. By selectively delaying production, the algorithm accumulates a pool of orders, effectively filtering out the short-term noise of the arrival process. This allows for the consistent formation of high-density, high-value batches that effectively amortize the substantial fixed costs and height-dependent costs.

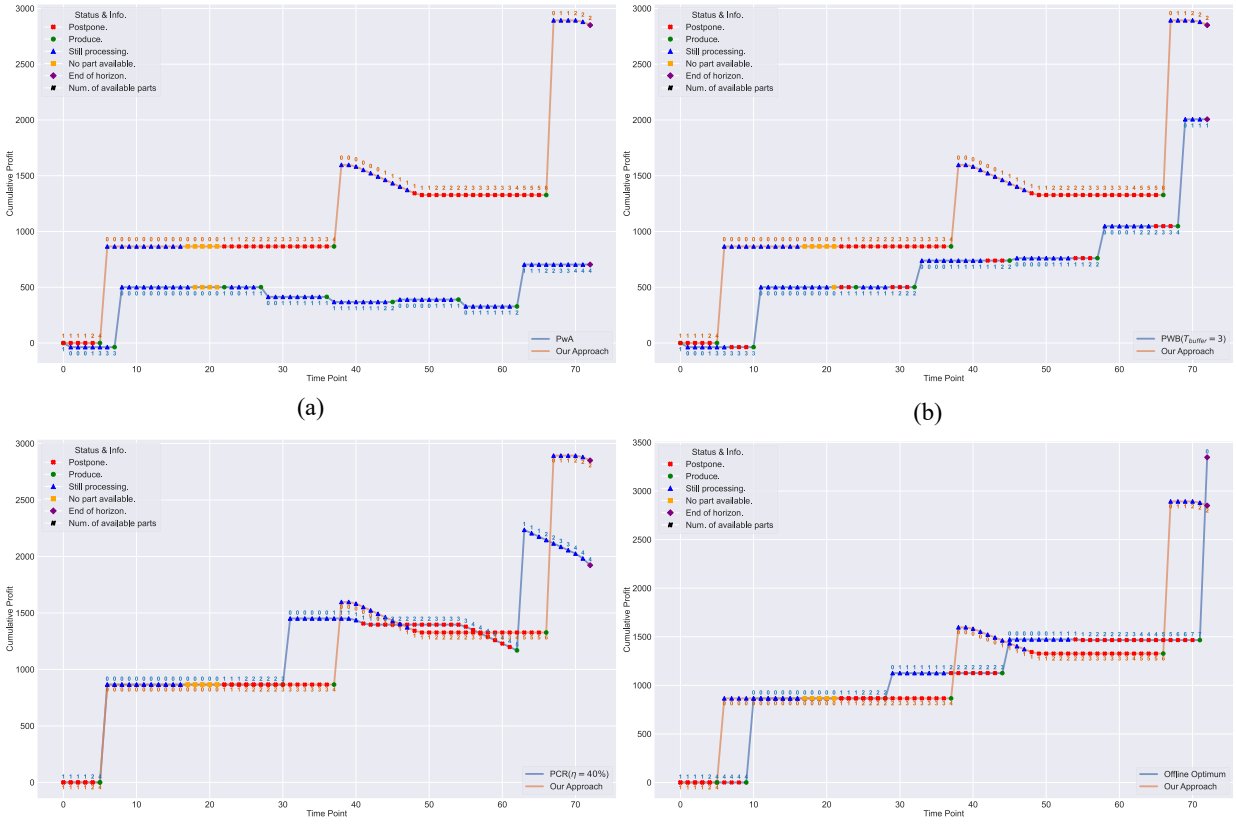


Figure 10: Cumulative profit trajectory comparison of Instance *H72-Large-S3*: Our Approach vs. (a) Process while Available. (b) Process with Waiting Buffer ($T_{\text{buffer}} = 3$). (c) Process with Capacity Rule ($\eta = 40\%$). (d) Offline Optimum. Numbers on the plot indicate the number of available parts at each decision epoch.

5.3. Value of Lookahead Mechanism

To isolate and quantify the value derived specifically from the simulation-based lookahead capability of our ADP approach, we compare it against two parameter-dependent heuristic policies. Like our approach, these baselines incorporate postponement, but they lack the forward-looking mechanism, relying instead on static, state-independent rules to trigger production.

The Process with Waiting Buffer (PWB) policy employs a static, time-based postponement rule. Whenever the machine becomes free, this policy enforces a mandatory waiting period of fixed duration, T_{buffer}. This systematic delay aims to accumulate incoming orders regardless of the current queue state. Once the buffer timer expires, if parts are available, the policy terminates the postponement and initiates the batch that yields the highest immediate net profit from the accumulated pool. On the other hand, the Process with Capacity Rule (PCR) policy postpones processing as long as all currently available parts can be feasibly nested into a single batch that utilizes less than the proportion, η, of the build platform area. The pseudocode of both algorithms are given in Algorithm G.1.

Table 1 provides a comparison between our approach with PWB & PCR baselines across different parameter settings on the testbed. As illustrated, the proposed lookahead-based approach consistently outperforms both PWB and PCR across the majority of instances. Analysis of these results are given as follows. Firstly, a critical drawback of the baseline heuristics is their sensitivity to parameter settings. As shown, the performance of PWB and PCR fluctuates significantly with variations in T_{buffer} and η. In a real-world production environment, determining the optimal parameter requires extensive historical data analysis and continuous re-tuning as demand patterns shift. Furthermore, even when configured with the "best" parameters identified, these static policies exhibit high performance variance across different instances, indicating a lack of structural robustness. In contrast, our lookahead-based ADP approach demonstrates superior robustness. It achieves consistently high RDI scores with low variance across diverse arrival patterns without relying on instance-specific parameter tuning. This validates that the lookahead mechanism generalizes to varying operational conditions.

Figures 10(b) and (c) illustrate the decision trajectories of the best-performing configurations of PWB ($T_{\text{buffer}} = 3$) and PCR ($\eta = 40\%$) against our proposed approach, for Instance 'H72-Large-S3'. As demonstrated in Figure 10(b), the PWB policy suffers from temporal rigidity. Due to its reliance on a fixed time buffer, it may be forced to initiate an unprofitable batch as the time buffer expires just before a critical order arrives. For example, it initiates a single-part batch at 24h, thereby missing a high-value arrival at 25h. In contrast, our proposed approach proactively mitigates such inefficiencies. Facing the same situation at 24h, our policy weighs the opportunity cost of postponing against immediate producing by simulating future arrivals within lookahead. Consequently, it postpones production until 37h, resulting in a batch characterized by both high density and high profitability. Figure 10(c) reveals that the PCR policy is indifferent to the economic attributes and delivery urgency of orders. By focusing solely on area utilization, this policy may keep postponing a high-value but low-volume batch if the threshold is not met. For example, it postpones production from 55h to 61h, incurring severe tardiness penalties. On the contrary, our approach dynamically adjusts the postponement duration based on a comprehensive system state rather than adhering to a rigid area-based proxy. This adaptive mechanism allows our policy to process urgent parts as early as 37h, thereby ensuring greater state flexibility at 55h compared to the PCR policy.

5.4. Effectiveness of Decision Space Pruning

This experiment quantifies the computational and operational benefits of the heuristic decision space pruning strategy proposed in Section 4.3.2. We compare our approach against a baseline of its variant with entire decision space. This baseline is identical to our proposed approach in all algorithmic aspects with one exception: during the *Expansion* step of the MCTS, it considers all feasible batches generated by the heuristic (Algorithm 6) as child nodes.

Table 2 presents the comparative results of the two approaches across the testbed. The values represent the average total profit obtained from 5 independent simulation runs for each instance. As shown, the results demonstrate the effectiveness of the decision space pruning. In every test instance, the "Pareto-Front" strategy significantly outperforms the baseline, which means that the application of pruning strategy to our approach consistently enables the discov-

Table 2: Comparison of average total profits between Our Approach (using Pareto-Front decision pruning) and the Baseline (exploring the entire feasible decision space). 'Gap' columns denote performance improvement by the pruning strategy proportional to the baseline value.

Arrival patt.	Ins.	T_{horizon}=36			T_{horizon}=72			T_{horizon}=144		
		Pareto-Front (Ours)	Entire Space (Baseline)	Gap	**Pareto-Front (Ours)**	Entire Space (Baseline)	Gap	**Pareto-Front (Ours)**	Entire Space (Baseline)	Gap
Uniform	#1	981.24	302.89	+223.96%	1964.71	1149.81	+70.87%	6717.85	5669.30	+18.50%
	#2	1318.95	591.90	+122.84%	2666.52	2222.59	+19.97%	6753.35	5315.55	+27.05%
	#3	2164.09	1391.69	+55.50%	2706.70	1266.00	+113.80%	4682.94	3179.22	+47.30%
Small-dom.	#1	507.99	86.55	+486.92%	1613.88	1038.23	+55.44%	4248.13	3409.74	+24.59%
	#2	1052.97	390.30	+169.78%	2444.28	1633.28	+49.65%	5671.94	4535.55	+25.06%
	#3	707.51	227.62	+210.82%	2651.13	1328.05	+99.63%	3406.09	2163.03	+57.47%
Large-dom.	#1	990.77	531.34	+86.47%	3859.66	3244.16	+18.97%	5507.14	3514.14	+56.71%
	#2	1821.08	944.77	+92.75%	2960.75	2225.53	+33.04%	8286.35	6636.65	+24.86%
	#3	1312.42	788.52	+66.44%	2512.88	1396.60	+79.93%	5071.76	3316.11	+52.94%

ery of higher-value policies. The performance gap is particularly pronounced in Small-dominant instances. In these scenarios, the number of small parts leads to a combinatorial explosion of feasible batches, making the full decision space unmanageable. Our pruning heuristic effectively navigates this complexity, identifying efficient consolidation opportunities that the baseline fails to find amid the noise.

5.5. Benchmarking against Offline Optimum

To evaluate the quality of the solutions generated by our ADP approach, we benchmark its performance against the offline optimum. The offline optimum represents the theoretical upper bound of profit achievable under the assumption of perfect information, where the complete sequence of future order arrivals is known a priori at the beginning of the planning horizon. This bound is obtained by solving the deterministic MILP model formulated in Section 3.3.

By comparing the online performance (operating under strict uncertainty) with this offline bound, we can assess how effectively our policy approximates the true optimal decisions. We define the Empirical Competitive Ratio (ECR) for each instance as:

$$\text{ECR} = \frac{\text{Obj}_{\text{MILP}}}{\text{Obj}_{\text{ADP}}}, \tag{40}$$

where Obj_{MILP} is the objective value obtained by the MILP solver, and Obj_{ADP} is the profit obtained by our ADP approach, which is approximated by the average of five simulation runs. An ECR closer to 1 indicates performance closer to the theoretical optimum.

Table 3 summarizes the comparison results. The ECR values across all solvable instances range from 1 to 1.70, with an average of approximately 1.27. For instances with short horizon ($T_{\text{horizon}} = 36$), the average ECR is approximately 1.26, with several instances achieving ECR = 1. This indicates that in shorter timeframes, where uncertainty is less cumulative, our ADP approach is possible to identify solutions that are identical to the global optimum. Furthermore, it is crucial to note that for instances with long horizon ($T_{\text{horizon}} = 144$), the MILP solver fails to prove optimality within the 3600-second time limit, leaving significant optimality gaps. In contrast, our ADP approach

Table 3: Comparison of Our Approach (Online) vs. MILP (Offline) across the testbed. **Our Approach (online)**: 'avg.' and 'max' denote the average and maximum profit over 5 runs. Bold values indicate that the online policy achieved the offline optimal value. 't_{ADP}(s)' is the average run time of each execution of our proposed ADP policy within the simulation experiments. **MILP (offline)**: Solved using Gurobi 12.0.3 with a 3600s time limit. 'val.' is the objective value; '*opt./feas.*' indicates if the solution is proven optimal (*opt.*) or the best feasible solution found within the time limit (*feas.*); 'MIPGap' reports the optimality gap for feasible solutions; 't_{MILP}(s)' is the computation time.

T_{horizon}	Arrival Patt.	Ins.	**Our Approach (online)**			MILP (offline)				ECR
			avg.	max	t_{ADP}(s)	val.	*opt./feas.*	MIPGap	t_{MILP}(s)	
36	Small-dom.	#1	981.24	**1198.19**	5.20	1198.19	*opt.*	0.00%	0.61	1.22
		#2	1318.95	**1662.19**	6.47	1662.19	*opt.*	0.00%	4.66	1.26
		#3	2164.09	2263.21	5.81	2482.14	*opt.*	0.00%	96.21	1.15
	Large-dom.	#1	507.99	**765.39**	4.57	765.39	*opt.*	0.00%	0.80	1.51
		#2	1052.97	1182.13	5.79	1220.05	*opt.*	0.00%	1.13	1.16
		#3	707.51	874.88	5.46	1205.59	*opt.*	0.00%	2.40	1.70
	Uniform	#1	990.77	**1213.47**	8.10	1213.47	*opt.*	0.00%	1.88	1.22
		#2	1821.08	**1966.60**	11.75	1966.60	*opt.*	0.00%	1.02	1.08
		#3	1312.42	**1316.57**	10.62	1316.57	*opt.*	0.00%	1.27	1
72	Small-dom.	#1	1964.71	2106.36	8.89	2535.18	*opt.*	0.00%	544.37	1.29
		#2	2666.52	3001.02	12.90	3586.46	*opt.*	0.00%	336.34	1.34
		#3	2706.70	2926.30	10.23	3231.46	*opt.*	0.00%	2172.88	1.19
	Large-dom.	#1	1613.88	1883.77	13.13	2721.11	*feas.*	13.31%	3601.87	-
		#2	2444.28	3018.59	11.80	3643.28	*opt.*	0.00%	62.17	1.49
		#3	2651.13	2794.08	11.74	3008.22	*opt.*	0.00%	2811.38	1.13
	Uniform	#1	3859.66	4079.54	12.66	4852.88	*opt.*	0.00%	396.15	1.26
		#2	2960.75	3124.67	10.74	3718.22	*feas.*	8.74%	3602.66	-
		#3	2512.88	2850.32	11.28	3347.33	*opt.*	0.00%	2298.40	1.33
144	Small-dom.	#1	6717.85	6975.03	20.68	7399.36	*feas.*	27.40%	3602.16	-
		#2	6753.35	6913.55	23.49	7753.62	*feas.*	20.38%	3603.40	-
		#3	4682.94	5055.90	21.17	6357.04	*feas.*	25.49%	3613.72	-
	Large-dom.	#1	4248.13	4368.25	20.45	5627.55	*feas.*	35.07%	3602.17	-
		#2	5671.94	5976.12	21.19	6989.95	*feas.*	25.99%	3601.99	-
		#3	3406.09	3712.73	21.38	4497.08	*feas.*	30.97%	3608.29	-
	Uniform	#1	5507.14	5692.58	22.85	6912.11	*feas.*	29.48%	3619.28	-
		#2	8286.35	8548.77	19.38	7781.44	*feas.*	49.19%	3601.61	-
		#3	5071.76	5610.88	21.63	6111.66	*feas.*	26.34%	3601.76	-

consistently delivers high-quality solutions within a much shorter and controllable computation time. This highlights the practical scalability of our method for real-world industrial deployment, where rapid decision-making is essential.

6. Conclusion

This study addresses the dynamic scheduling problem for a single Additive Manufacturing (AM) machine subject to stochastic order arrivals. Motivated by the distinct subadditive cost structure of Powder Bed Fusion (PBF) technologies, where high setup and recoating costs create a compelling economic imperative for batch consolidation, we investigate the strategic value of *postponement*. Unlike traditional reactive scheduling paradigms that prioritize machine utilization, we formulated the problem as a finite-horizon Markov Decision Process (MDP). In this formulation, the decision to postpone production is modeled as a proactive and strategic choice, balancing the risks of tardiness against potential gains through improved future batch consolidation. Moreover, an offline mixed-integer linear program assuming perfect a priori information is established to provide a theoretical bound to online policies.

To overcome the curse of dimensionality inherent in exact resolution of MDP formulation, we developed a Direct Lookahead Approximation (DLA) policy. This approach is implemented via a tailored Monte Carlo Tree Search (MCTS) to approximate optimal decision by efficiently navigating the immense lookahead tree. A computationally-efficient one-step lookahead heuristic is employed as a surrogate policy governing rollout simulations. A Pareto-based pruning strategy was introduced to filter the combinatorially explosive space of feasible decisions, allowing the search budget to focus on high-potential ones.

Extensive numerical experiments demonstrate the superior performance and robustness of the proposed approach. First, the comparison with the reactive baseline confirms that strategic postponement serves as a critical economic lever, effectively amortizing costs by decoupling production initiation from stochastic order arrivals. Second, benchmarking against parameter-dependent heuristics highlights the advantage of the lookahead mechanism, which adaptively navigates the trade-off between order urgency and batch efficiency without requiring instance-specific tuning. Third, the evaluation of the decision space pruning strategy validates its necessity for mitigating combinatorial explosion, ensuring computational resources are focused on high-potential branches. Finally, the proposed policy achieves an average empirical competitive ratio of 1.27 against the offline theoretical optimum while maintaining computational tractability.

Our analysis yields critical managerial insights for AM service providers. First, strategic postponement can be a vital source of value rather than waste. Operators should actively decouple production initiation from stochastic arrivals to amortize fixed costs through reasonable batching. Second, static scheduling rules prove inadequate in fluctuating social AM markets due to their reliance on fixed parameters and their inflexibility in handling stochastic arrivals. The simulation-based nature of lookahead mechanism enables the DLA policy to make state-dependent decisions. This superior robustness underscores the necessity for operators to transition towards foresight-centric decision-making.

While this work provides a comprehensive framework for single-machine dynamic scheduling with postponement, several avenues for future research remain. First, the current model considers a single machine system, while parallel AM machine environments requires an even scalable framework to make dynamic postponement and scheduling decisions. Second, although the 2D packing serves as an approximation for nesting in PBF technologies, sophistic 3D packing could be considered to extend the applicability of this work to other AM technologies (e.g., Selective Laser Sintering, Multi Jet Fusion). Finally, our assumption of a homogeneous Poisson arrival process could be relaxed. Investigating the impact of non-stationary or time-dependent arrival rates (e.g., rush hours or seasonal demand) would provide further insights into robust postponement and scheduling under fluctuating demand patterns.

CRediT authorship contribution statement

Hao Wu: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing - Original Draft, Visualization. **Chunlong Yu:** Conceptualization, Methodology, Formal analysis, Resources, Writing - Review & Editing, Supervision, Funding acquisition.

Declaration of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant [72301196]; Tongji University 'Fundamental Research Funds for the Central Universities' under Grant [22120240607].

References

Altekin, F. T., & Bukchin, Y. (2022). A multi-objective optimization approach for exploring the cost and makespan trade-off in additive manufacturing. *European Journal of Operational Research*, *301*(1), 235-253.

Aminian, M. R., Ma, W., & Xin, L. (2023). Real-time personalized order holding. Available at SSRN 4644495.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society, 60*(6), 503-515.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

Bertsekas, D. P., & Castanon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, *5*(1), 89-108.

Bertsimas, D., Griffith, J. D., Gupta, V., Kochenderfer, M. J., & Mišić, V. V. (2017). A comparison of Monte Carlo tree search and rolling horizon optimization for large-scale dynamic resource allocation problems. *European Journal of Operational Research, 263*(2), 664-678.

Bhimaraju, A., Etesami, S. R., & Varshney, L. R. (2025). Dynamic batching of online arrivals to leverage economies of scale. *European Journal of Operational Research*.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., et al. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games, 4*(1), 1-43.

Burke, E., Kendall, G., & Whitwell, G. 2004. A New Placement Heuristic for the Orthogonal Stock-cutting Problem. *Operations Research*, *52*(4): 655-671.

Carpin, S. (2025). Solving stochastic orienteering problems with chance constraints using Monte Carlo tree search. *IEEE Transactions on Automation Science and Engineering, 22*, 7855.

Chaslot, G., Winands, M., Herik, H., Uiterwijk, J., & Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, *4*(3), 343-357.

Chen, S., Yan, Z., & Lim, Y. F. (2024). Managing the Personalized Order-Holding Problem in Online Retailing. *Manufacturing & Service Operations Management, 26*(1), 47-65.

Chen, T., Zhang, X. L., Chu, F., & Zhang, J. (2024). Unleashing the Power of No-Rush Delivery: Postponement Policies for Sustainable Order Fulfillment. Available at SSRN 4889746.

Couetoux, A., & Doghmen, H. (2011, September). Adding double progressive widening to upper confidence trees to cope with uncertainty in planning problems. In The 9th European Workshop on Reinforcement Learning (EWRL-9).

Deng, Q., & Santos, B. F. (2022). Lookahead approximate dynamic programming for stochastic aircraft maintenance check scheduling optimization. *European Journal of Operational Research, 299*(3), 814-833.

Ding, J., Baumers, M., Clark, E. A., & Wildman, R. D. (2021). The economics of additive manufacturing: Towards a general cost model including process failure. *International Journal of Production Economics*, *237*, 108087.

Erazo, I., & Toriello, A. (2025). Optimizing the trade-off between batching and waiting: Subadditive dispatching. *Operations Research*.

Fu, M. C. (2016). AlphaGo and Monte Carlo tree search: The simulation optimization perspective. In *Proceedings of the 2016 Winter Simulation Conference*, 657-670.

Fu, Z., Fan, Q., Zhang, X., Li, X., Wang, S., & Wang, Y. (2021). Policy network assisted Monte Carlo tree search for intelligent service function chain deployment. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 1082-1089.

Gilmore, P. C., & Gomory, R. E. (1965). Multistage cutting stock problems of two and more dimensions. *Operations Research*, *13*(1), 94-120.

Gopsill, J. (2024). Distributed Additive Manufacturing: A Social Change in Manufacturing. In Engineering For Social Change (pp. 115-124). IOS Press.

JLC-3DP. (2025). JLC 3D Printing Service. Available at: https://www.jlc-3dp.cn (Accessed: 25 December 2025).

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *Proceedings of the 17th European Conference on Machine Learning*, 282-293.

Li, Q., Zhang, D., Wang, S., & Kucukkoc, I. (2019). A dynamic order acceptance and scheduling approach for additive manufacturing on-demand production. *The International Journal of Advanced Manufacturing Technology*, *105*, 3711-3729.

Lin, J., & Yu, C. (2025). The energy-aware nesting and scheduling problem on a single selective laser melting machine: mathematical model and solution approach. *International Journal of Production Research*, 1-30.

Lv, J., Peng, T., Zhang, Y., & Wang, Y. (2021). A novel method to forecast energy consumption of selective laser melting processes. *International journal of production research*, *59*(8), 2375-2391.

Mańdziuk, J., & Okulewicz, M. (2015). Dynamic vehicle routing problem: A Monte Carlo approach. In *the ITRIA 2015 Conference*. IPI PAN.

Powell, W. B. (2011). *Approximate dynamic programming: Solving the curses of dimensionality* (2nd ed.). Wiley.

Powell, W. B. (2019). A unified framework for stochastic optimization. *European Journal of Operational Research, 275*(3), 795-821.

Powell, W. B. (2022). *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. Hardback.

Prataviera, L. B., Jazairy, A., & Abushaikha, I. (2024). Navigating the intersection between postponement strategies and additive manufacturing: insights and research agenda. *International Journal of Production Research*, 1-23.

Rickenbacher, L., Spierings, A., & Wegener, K. (2013). An integrated cost-model for selective laser melting (SLM). *Rapid Prototyping Journal*, *19*(3), 208-214.

Sabuncuoglu, I., & Bayız, M. (2000). Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research*, *126*(3), 567-586.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., ... & Silver, D. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature, 588*(7839), 604-609.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*(7587), 484-489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science, 362*(6419), 1140-1144.

Sun, M., Ding, J., Zhao, Z., Chen, J., Huang, G. Q., & Wang, L. (2025). Out-of-order execution enabled deep reinforcement learning for dynamic additive manufacturing scheduling. *Robotics and Computer-Integrated Manufacturing, 91*, 102841.

Thomas, D. S., & Gilbert, S. W. (2014). Costs and cost effectiveness of additive manufacturing. NIST special publication, 1176, 12.

Truong, V. A. (2014). Approximation algorithm for the stochastic multi-period inventory problem via a look-ahead optimization approach. *Mathematics of Operations Research, 39*(4), 1039-1056.

Tschernutter, D., & Feuerriegel, S. (2024). Data-driven dynamic police patrolling: An efficient Monte Carlo tree search. *European Journal of Operational Research, 321*(1), 177-191.

Ulmer, M. W., Thomas, B. W., Woyak, N., & Campbell, A. M. (2021). The Restaurant Meal Delivery Problem: Dynamic Pick-Up and Delivery with Deadlines and Random Ready Times. *Transportation Science, 55*(5), 1000-1020.

Van Heeswijk, W. J. A., Mes, M. R. K., & Schutten, J. M. J. (2017). The Delivery Dispatching Problem with Time Windows for Urban Consolidation Centers. *Transportation Science, 53*(1), 203-223.

Van Mieghem, J. A., & Dada, M. (1998). Price versus production postponement: capacity and competition. *Management Science, 45*(12), 1631-1649.

Wang, K., Chen, L., Liu, Q., Wang, W., & Li, F. (2014). One step beyond myopic probing policy: A heuristic lookahead policy for multi-channel opportunistic access. *IEEE Transactions on Wireless Communications*, *14*(2), 759-769.

Wei, L., Kapuscinski, R., & Jasin, S. (2021). Shipping Consolidation Across Two Warehouses with Delivery Deadline and Expedited Options for E-commerce and Omni-channel Retailers. *Manufacturing & Service Operations Management, 23*(6), 1634-1650.

Wu, H., Yu, C., & Yu, S. (2025). Nesting and scheduling for parallel additive manufacturing machines with uncertain processing times: a simulation-optimisation approach. *International Journal of Production Research*, 1–30.

Wu, Q., Xie, N., Zheng, S., & Bernard, A. (2022). Online order scheduling of multi 3D printing tasks based on the additive manufacturing cloud platform. *Journal of Manufacturing Systems, 63*, 23-34.

Xie, Y., Ma, W., & Xin, L. (2025). The Benefits of Delay to Online Decision Making. *Management Science*.

Xometry Inc. (2025). Xometry: Custom Manufacturing on Demand. Available at: https://www.xometry.com (Accessed: 25 December 2025).

Yan, J., Hu, B., Xie, K., Niu, T., Li, C., & Tai, H. M. (2020). Dynamic repair scheduling for transmission systems based on look-ahead strategy approximation. *IEEE Transactions on Power Systems, 36*(4), 2918-2933.

Yang, B., & Burns, N. D. (2003). Implications of postponement for the supply chain. *International Journal of Production Research, 41*(9), 2075-2090.

Yu, C., Matta, A., Semeraro, Q., & Lin, J. (2022). Mathematical models for minimizing total tardiness on parallel additive manufacturing machines. *IFAC PapersOnLine, 55*(10), 1521-1526.

Appendix A. MILP Formulation for Offline Version of Problem

In this section, we presents the MILP formulation of the offline version of our online decision-making problem, under the assumption is that all information regarding order arrivals within time horizon is known a priori. We adopt the parameters, decision variables and notation defined in Section 3.2. Let $\mathbb{P}$ denote the set of all parts ordered over the entire time horizon, and $\mathbb{K} = \{0, 1, \ldots, K-1\}$ be the set of discrete decision epochs. To formulate the problem linearly, we introduce the following auxiliary variables:

- $H_k \in \mathbb{R}^+$: Height of the batch initiated at epoch k.
- $c_p, T_p \in \mathbb{R}^+$: Completion time and tardiness of part p, respectively.
- $b_p \in \{0, 1\}$: Binary indicator for linearizing the tardiness calculation of part p.

The objective is to maximize the total profit, defined as the total revenue from processed parts minus the total production costs and total tardiness penalties accrued within time horizon:

$$\max \quad \underbrace{\sum_{k\in\mathbb{K}}\sum_{p\in\mathbb{P}} \varrho_p X_{k,p}}_{\text{Total Revenue}} - \underbrace{\sum_{k\in\mathbb{K}} Y_k \left(C^{\text{opr}} + K^{\text{bld}} t_k^{\text{bld}} + K^{\text{pow}} \sum_{p\in\mathbb{P}} (v_p + s_p) X_{k,p} \right)}_{\text{Total Production Costs}} - \underbrace{\sum_{p\in\mathbb{P}} K^{\text{fine}} T_p}_{\text{Total Tardiness Penalties}} . \tag{A.1}$$

This formulation is subject to the following constraints (A.2-A.22):

$$\sum_{k\in\mathbb{K}} X_{k,p} \le 1, \qquad \forall p \in \mathbb{P} \tag{A.2}$$

$$k \cdot u \ge r_p - M(1 - X_{k,p}), \qquad \forall p \in \mathbb{P}, \forall k \in \mathbb{K} \tag{A.3}$$

Constraints (A.2) ensure that each part is processed at most once. Constraint (A.3) enforces that a part can only be assigned to a batch at or after its arrival time.

$$X_{k,p} \le Y_k, \qquad \forall p \in \mathbb{P}, \forall k \in \mathbb{K} \tag{A.4}$$

$$Y_k \le \sum_{p\in\mathbb{P}} X_{k,p}, \qquad \forall k \in \mathbb{K} \tag{A.5}$$

$$\tau_k \le (k+1)u + M(1 - Y_k), \qquad \forall k \in \mathbb{K} \tag{A.6}$$

Constraints (A.4) state that if the decision is postponed, no part will be selected at this decision epoch. Constraints (A.5) ensure that a processed batch contains at least one part. Constraints (A.6) enforce that a batch can be initiated for processing only if the machine is available before the next decision epoch.

$$\tau_{k+1} \ge (k+1)u - MY_k, \qquad \forall k \in \mathbb{K} \tag{A.7}$$

$$\tau_{k+1} \ge \tau_k - MY_k, \qquad \forall k \in \mathbb{K} \tag{A.8}$$

$$\tau_{k+1} \ge (\tau_k + t^{\text{aux}} + t_k^{\text{bld}}) - M(1 - Y_k), \qquad \forall k \in \mathbb{K} \tag{A.9}$$

Constraints (A.7)-(A.9) model the state transition for the machine's availability time.

$$H_k \geq h_p X_{k,p}, \qquad \forall p \in \mathbb{P}, \forall k \in \mathbb{K} \tag{A.10}$$

$$H_k \leq \mathcal{H} Y_k, \qquad \forall k \in \mathbb{K} \tag{A.11}$$

$$t_k^{\text{bld}} \geq \sum_{p \in \mathbb{P}} \left(\frac{v_p}{V^{\text{body}}} + \frac{s_p}{V^{\text{supp}}} \right) X_{k,p} + t^{\text{rec}} H_k - M(1 - Y_k), \qquad \forall k \in \mathbb{K} \tag{A.12}$$

$$t_k^{\text{bld}} \leq M Y_k, \qquad \forall k \in \mathbb{K} \tag{A.13}$$

Constraints (A.10) set the batch height H_k to the maximum height of any part within it. Constraints (A.11) enforce the machine's height limit and ensures $H_k = 0$ if no batch is processed at decision epoch k. Constraints (A.12)-(A.13) calculate the batch's build time if $Y_k = 1$ and force it to zero otherwise.

$$c_p \geq (\tau_k + t^{\text{aux}} + t_k^{\text{bld}}) - M(1 - X_{k,p}), \qquad \forall p \in \mathbb{P}, \forall k \in \mathbb{K} \tag{A.14}$$

$$T_p \geq (K \cdot u - d_p) - M \sum_{k \in \mathbb{K}} X_{k,p}, \qquad \forall p \in \mathbb{P} \tag{A.15}$$

$$T_p \geq (c_p - d_p) - M(1 - b_p) - M\left(1 - \sum_{k \in \mathbb{K}} X_{k,p}\right), \qquad \forall p \in \mathbb{P} \tag{A.16}$$

$$T_p \geq (K \cdot u - d_p) - M b_p - M\left(1 - \sum_{k \in \mathbb{K}} X_{k,p}\right), \qquad \forall p \in \mathbb{P} \tag{A.17}$$

Constraints (A.14) set the completion time c_p for any processed part. Constraints (A.15)-(A.17) model the tardiness $T_p = \max(0, \min(c_p, K \cdot u) - d_p)$ for processed parts and $T_p = \max(0, K \cdot u - d_p)$ for unprocessed parts. Specifically, if a part is unprocessed, (A.15) applies the penalty relative to the horizon end. If processed, (A.16) and (A.17) utilize auxiliary binary variable b_p to linearize the minimum function $\min(c_p, K \cdot u)$.

$$x_{k,p} + l_p(1 - o_{k,p}) + w_p o_{k,p} \leq \mathcal{L} + M(1 - X_{k,p}), \qquad \forall p \in \mathbb{P}, \forall k \in \mathbb{K} \tag{A.18}$$

$$y_{k,p} + w_p(1 - o_{k,p}) + l_p o_{k,p} \leq \mathcal{W} + M(1 - X_{k,p}), \qquad \forall p \in \mathbb{P}, \forall k \in \mathbb{K} \tag{A.19}$$

$$x_{k,p} + l_p(1 - o_{k,p}) + w_p o_{k,p} \leq x_{k,p'} + M(3 - X_{k,p} - X_{k,p'} - PL_{k,pp'}), \qquad \forall p, p' \in \mathbb{P}, p \neq p', \forall k \in \mathbb{K} \tag{A.20}$$

$$y_{k,p} + w_p(1 - o_{k,p}) + l_p o_{k,p} \leq y_{k,p'} + M(3 - X_{k,p} - X_{k,p'} - PB_{k,pp'}), \qquad \forall p, p' \in \mathbb{P}, p \neq p', \forall k \in \mathbb{K} \tag{A.21}$$

$$PL_{k,pp'} + PL_{k,p'p} + PB_{k,pp'} + PB_{k,p'p} \geq X_{k,p} + X_{k,p'} - 1, \qquad \forall p, p' \in \mathbb{P} \text{ with } p < p', \forall k \in \mathbb{K} \tag{A.22}$$

Constraints (A.18)-(A.19) ensure that all selected parts are placed within the platform boundaries. Constraints (A.20)-(A.22) are the non-overlapping constraints, which are activated only when a pair of parts are included in the same batch, enforcing a valid spatial relationship between them.

Appendix B. Generate-then-Repair Heuristic for Fast Part Selection

The pseudocode of generate-then-repair heuristic is given in Algorithm B.1. The core idea is to prioritize parts based on their "profit density" ($\frac{\varrho_p}{l_p \cdot w_p}$), which serves as a proxy for the economic value of consuming the limited build platform surface. The heuristic proceeds in two phases:

1. Generate Phase: All available parts are sorted in descending order of their profit density. A candidate batch is greedily constructed by adding parts from the top of this list until the cumulative area of their bounding boxes exceeds the platform area ($\mathcal{L} \times \mathcal{W}$). This step typically yields a set that is high in potential value but likely infeasible due to geometry.
2. Repair Phase: The candidate batch is subjected to a rigorous nesting feasibility check (Algorithm 5). If the batch is infeasible, a greedy repair mechanism is invoked. This procedure iteratively removes the part with the lowest profit density from the current candidate set. The feasibility check is repeated after each removal until a valid subset is found.

This strategy avoids the high overhead of constructive nesting algorithms by performing expensive feasibility checks only on promising subsets, ensuring the efficiency required for real-time decision-making.

Algorithm B.1 Generate-then-Repair Heuristic for Part Selection

Input: Set of available parts $\mathbb{P}$

Output: A feasible subset of parts for production $\mathbb{P}^*$

procedure HEUSELECT($\mathbb{P}$)
  $\mathbb{P}_{\text{sorted}} \leftarrow$ Sort $\mathbb{P}$ by descending order of $\frac{g_p}{l_p \cdot w_p}$
  $\mathbb{P}_{\text{candidate}} \leftarrow \emptyset$, Area $\leftarrow 0$
  for $p \in \mathbb{P}_{\text{sorted}}$ **do**
    if Area $+ l_p \cdot w_p \leq \alpha \cdot \mathcal{L} \cdot \mathcal{W}$ **then** ▷ $\alpha \geq 1$ is an overfilling factor
      $\mathbb{P}_{\text{candidate}} \leftarrow \mathbb{P}_{\text{candidate}} \cup \{p\}$
      Area $\leftarrow$ Area $+ l_p \cdot w_p$
    else
      break
    end if
  end for
  while not FEASIBILITYCHECK($\mathbb{P}_{\text{candidate}}$) **do** ▷ Repair loop
    if $|\mathbb{P}_{\text{candidate}}| \leq 1$ **then**
      return $\emptyset$ ▷ No feasible batch found
    end if
    Remove part $p_{\min}$ with lowest $\frac{g_p}{l_p \cdot w_p}$ from $\mathbb{P}_{\text{candidate}}$
  end while
  return $\mathbb{P}_{\text{candidate}}$
end procedure

Appendix C. Example of Feasible “Produce" Decision Space Growth

To illustrate the combinatorial complexity encountered in the `Expansion` phase of the MCTS, this appendix analyzes the growth of the decision space for a representative example. We consider a set of 10 parts with varying dimensions (P_1: 48×37, P_2: 69×65, P_3: 62×51, P_4: 47×45, P_5: 88×38, P_6:37×45, P_7: 61×63, P_8: 98×37, P_9: 59×87, P_{10}: 62×91) to be packed on a standard 200×200 platform. Figure C.1 depicts the relationship between the number of available parts and the number of subsets that are feasibly packable. As shown in Figure C.1, the number of combinatorially feasible batches grows superlinearly. Even for a small instance of 10 parts, the feasible decision space becomes substantial. This rapid expansion empirically justifies the necessity of the pruning mechanism employed in our approach to maintain computational tractability.

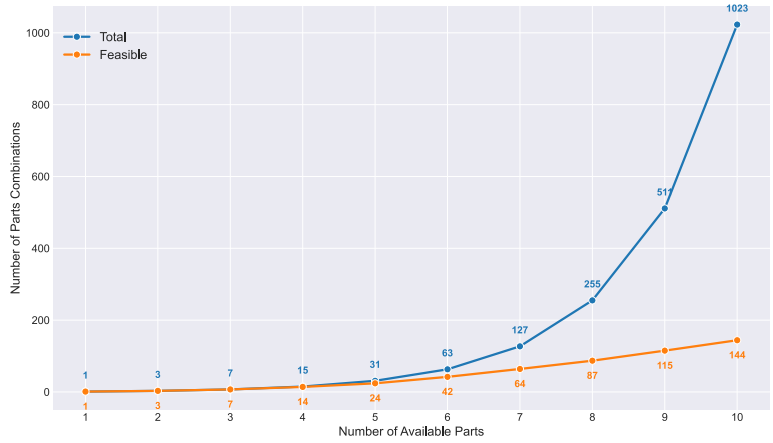


Figure C.1: Growth of feasible part combinations for an example set of 10 parts on a 200×200 platform.

Appendix D. Simulation of Part Attributes

This appendix details the specific parameters and logic used to simulate the attributes of each arrived part p for the realization of exogenous information $\mathbf{W}(\omega)$:

- *Dimensions*: The length (l_p), width (w_p), and height (h_p) are sampled from uniform distributions corresponding to the ranges of their part type classification. This classification is based on the build chamber dimensions ($\mathcal{L}, \mathcal{W}, \mathcal{H}$). The specific ranges for the categories are defined as follows:
  - Height categories: (1) *Low*: $h_p \in [\frac{1}{3}\mathcal{H}, \frac{1}{2}\mathcal{H}]$; (2) *High*: $h_p \in (\frac{1}{2}\mathcal{H}, \frac{2}{3}\mathcal{H}]$.
  - Projected bounding box categories: (1) *Small*: length $\in [\frac{1}{6}\mathcal{L}, \frac{1}{3}\mathcal{L}]$, width $\in [\frac{1}{3}\mathcal{W}, \frac{1}{2}\mathcal{W}]$; (2) *Long*: one dimension $\in (\frac{1}{3}\mathcal{L}, \frac{1}{2}\mathcal{L}]$, the other $\in [\frac{1}{6}\mathcal{W}, \frac{1}{3}\mathcal{W}]$; (3) *Large*: length $\in (\frac{1}{3}\mathcal{L}, \frac{1}{2}\mathcal{L}]$, width $\in (\frac{1}{3}\mathcal{W}, \frac{1}{2}\mathcal{W}]$.

- *Volume*: The volume of the part v_p is generated by sampling from the expression $l_p \cdot w_p \cdot h_p \cdot U(0.05, 0.15)$, where $U(a, b)$ denotes a uniform distribution defined by the lower bound a and the upper bound b. The volume of the support structure s_p is sampled from the distribution $v_p \cdot U(0, 0.3)$.

- *Due Date*: The due date d_p is assigned based on the standalone processing time t_p^{alo} and a "looseness factor" $\xi \geq 1$: $d_p = r_p + \xi \cdot t_p^{\text{alo}}$, where $t_p^{\text{alo}} = t^{\text{aux}} + (v_p/V^{\text{body}} + s_p/V^{\text{supp}} + h_p \cdot t^{\text{rec}})$.

- *Price*: The price of the part ϱ_p is governed by a pricing model that incorporates a base estimation of production cost and risk premiums for urgency and potential penalties:

$$\varrho_p = C_p^{\text{est}} \times \underbrace{\left(1 + \frac{1}{\xi}\right)}_{\text{Urgency Premium}} \times \underbrace{\left(1 + \frac{K^{\text{fine}}}{K_{\text{norm}}^{\text{fine}}}\right)}_{\text{Penalty Risk Premium}}, \tag{D.1}$$

  where $K_{\text{norm}}^{\text{fine}}$ is a normalizing constant. This pricing structure implies that orders with tighter deadlines and stricter penalties generate higher revenue. The estimated base cost C_p^{est} is the sum of direct volume-related costs and amortized fixed and height-related costs:

$$C_p^{\text{est}} = \underbrace{\frac{\max(l_p, w_p)}{\min(\mathcal{L}, \mathcal{W})}(C^{\text{opr}} + K^{\text{bld}} h_p t^{\text{rec}})}_{\text{Amortized Costs}} + \underbrace{K^{\text{pow}}(v_p + s_p) + K^{\text{bld}}\left(\frac{v_p}{V^{\text{body}}} + \frac{s_p}{V^{\text{supp}}}\right)}_{\text{Direct Costs}} \tag{D.2}$$

  Here, the operator cost and the height-dependent recoating costs are amortized based on the part's bound box dimension relative to the build platform size.

Appendix E. Parameters Values for Testbed Instance

Table E.1 summarizes the parameters utilized and their corresponding values adopted for the generation of testbed instances.

Table E.1: Parameters and their adopted values for the generation of instance testbed.

Parameter	Symbol	Value(s)
Internal System Parameters		
Machine Build Speeds	$V^{\text{body}}, V^{\text{supp}}$	15, 30 mm^3/s
Recoating Time per Unit Height	t^{rec}	200 s/mm
Platform Dimensions	$\mathcal{L}, \mathcal{W}, \mathcal{H}$	200, 200, 200 mm
Pre/Post-Build Times	$t^{\text{pre}}, t^{\text{post}}$	0.5, 0.5 h
Energy/Gas Cost Coeffs.	$K^{\text{eng}}, K^{\text{gas}}$	8, 3.6 CNY/h
Powder Cost Coeffs.	K^{pow}	0.001 CNY/mm^3
Operator Cost per Batch	C^{opr}	300 CNY
Tardiness Penalty Coefficient & Normalizer	$K^{\text{fine}}, K_{\text{norm}}^{\text{fine}}$	30,100 CNY/h

Parameter	Symbol	Value(s)	
External Environment Parameters			
Time Horizon	T_{horizon}	{36, 72, 144} h	
Due Date Looseness Factor	ξ	3	
Arrival Rate Matrix	$\mathbf{\Lambda}$	Uniform	$\begin{pmatrix} 0.04 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0.04 \end{pmatrix}$
		Small-dominant	$\begin{pmatrix} 0.06 & 0.04 & 0.02 \\ 0.06 & 0.04 & 0.02 \end{pmatrix}$
		Large-dominant	$\begin{pmatrix} 0.02 & 0.04 & 0.06 \\ 0.02 & 0.04 & 0.06 \end{pmatrix}$

Appendix F. Details of Parameter Calibration

The tunable parameters and their tested levels are: the exploration constant in the UCT equation (Eq. 32) $\theta \in \{5, 7.5, 10, 50\}$ and the sampling threshold which governs the transition from exploration to exploitation of stochastic outcomes $e^{thr} \in \{1, 5, 15\}$. To find the optimal settings for the tunable parameters, we conduct a calibration study on a representative subset of 6 instances, selected to cover a range of time horizons, arrival rates, and random seeds. The performance of different parameter configurations is evaluated using the RDI:

$$\text{RDI}(\mathcal{I}, C) = \frac{\text{Obj}(\mathcal{I}, C) - \text{Obj}(\mathcal{I}, C_{\text{worst}})}{\text{Obj}(\mathcal{I}, C_{\text{best}}) - \text{Obj}(\mathcal{I}, C_{\text{worst}})} \times 100\% \tag{F.1}$$

where $\text{Obj}(\mathcal{I}, C)$ is the objective value for instance $\mathcal{I}$ using parameter configuration C. This metric normalizes performance on a scale from 0% (worst) to 100% (best) for each instance. The tuning results are visualized as line plots showing the average RDI for each parameter level. As shown in Figure F.1, setting $\theta = 5$ leads to insufficient exploration, causing the search to converge prematurely to suboptimal branches. Conversely, $\theta = 50$ enforces excessive exploration, wasting the simulation budget on unpromising nodes. $\theta = 10$ provides better balance. Meanwhile, $e^{thr} = 5$ yields the better performance. A lower threshold ($e^{thr} = 1$) fails to capture the variance of future arrivals, leading to biased value estimation. A higher threshold ($e^{thr} = 15$) consumes too much budget on sampling the outcome space, reducing the depth of the search tree. Consequently, we fix $\theta = 10$ and $e^{thr} = 5$ for all subsequent experiments.

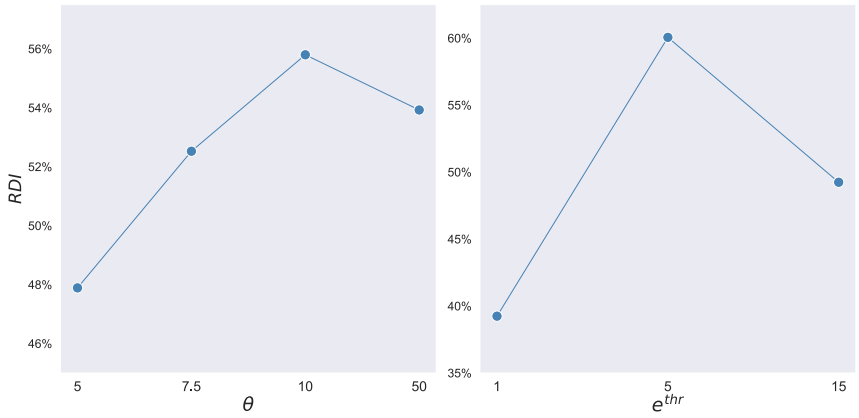


Figure F.1: Relative deviation index for different levels of θ and e^{thr}.

Appendix G. Pseudocode of Baseline Policies

The pseudocode for the baseline policies, which include the Process While Available policy, the Process with Waiting Buffer policy, and the Process with Capacity Rule policy, is presented in Algorithm G.1.

Algorithm G.1 Baseline Policies

Input: State S_k

Output: *Decision*

procedure PROCESS WHILE AVAILABLE(S_k)
  if $\tau_k \geq (k+1)u$ **or** $\mathbb{P}_k^{\text{ava}} = \emptyset$ **then**
    return *Postpone*
  else
    $C_{\text{feasible}} \leftarrow$ FINDFEASIBLECOMBINATIONS($\mathbb{P}_k^{\text{ava}}$)
    $\mathbb{P}_{\text{best}} \leftarrow$ the batch with the highest net profit $\mathcal{V}$ in C_{feasible}
    return *Produce* with $\mathbb{P}_{\text{best}}$
  end if
end procedure

procedure PROCESS WITH WAITING BUFFER(S_k, t_{wait}) ▷ Input waiting time counter t_{wait}
  if $\tau_k \geq (k+1)u$ **then**
    return *Postpone*, t_{wait}
  end if
  if $t_{\text{wait}} < T_{\text{buffer}}$ **or** $\mathbb{P}_k^{\text{ava}} = \emptyset$ **then**
    return *Postpone*, $t_{\text{wait}} + u$
  else
    $C_{\text{feasible}} \leftarrow$ FINDFEASIBLECOMBINATIONS($\mathbb{P}_k^{\text{ava}}$)
    $\mathbb{P}_{\text{best}} \leftarrow$ the batch with the highest net profit $\mathcal{V}$ in C_{feasible}
    return *Produce* with $\mathbb{P}_{\text{best}}$, 0 ▷ Reset wait timer
  end if
end procedure

procedure PROCESS WITH CAPACITY RULE(S_k)
  if $\tau_k \geq (k+1)u$ **or** $\mathbb{P}_k^{\text{ava}} = \emptyset$ **then**
    return *Postpone*
  end if
  if FEASIBILITYCHECK($\mathbb{P}_k^{\text{ava}}$) **and** $\frac{\sum_{p \in \mathbb{P}_k^{\text{ava}}} l_p w_p}{\mathcal{LW}} < \eta$ **then**
    return *Postpone*
  else
    $C_{\text{feasible}} \leftarrow$ FINDFEASIBLECOMBINATIONS($\mathbb{P}_k^{\text{ava}}$)
    $\mathbb{P}_{\text{best}} \leftarrow$ the batch with the highest net profit $\mathcal{V}$ in C_{feasible}
    return *Produce* with $\mathbb{P}_{\text{best}}$
  end if
end procedure