POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

Reinforcement Learning for Dynamic Scheduling

Supervisors

Prof. Edoardo FADDA

Prof. Leonardo Kanashiro FELIZARDO

Candidate

Lucca GAMBALLI

September 2025

Summary

This thesis investigates Reinforcement Learning (RL) for the Dynamic Job Shop Scheduling Problem (DJSSP), where agents make sequencing decisions under random job arrivals and tardiness is realized only upon job completion. This work argues that asynchronous per-machine decisions mitigate the credit-assignment challenge and assist training stability, motivating designs that explicitly align rewards with the causality of shop-floor events. The scheduler adopts a Centralized-Training and Decentralized-Execution (CTDE) scheme with parameter sharing and an event-driven policy that acts only at irregular decision epochs. This preserves local detail while remaining size-agnostic as queues fluctuate. State is constructed leveraging a "Minimal Repetition" encoder that packs the top job candidates of each machine into fixed slots with job-specific features, enabling direct job selection without fixing problem size. The delayed reward is handled via a chronological joint-action pipeline: Transitions are buffered without reward and completed only when a job finishes, allocating a joint signal to the responsible agents in proportion to the queueing they induced. Finally, this thesis proposes a hierarchical learning extension to the multi agent scheduler. This introduces a High-Level Agent that selects operating modes for Low-Level (per-machine) agents, enabling the system to adapt to the shop-flor current state. Simulation results indicate that the hierarchical RL framework proposed in this thesis is able to reduce the general shop-flor tardiness when compared to standard or learning based sequencing rules.

Acknowledgements

I am deeply grateful to both my supervisors, Edoardo Fadda and Leonardo Kanashiro Felizardo, for the guidance, high standards, and steady encouragement that shaped this work from the first sketches of the problem to its final form. Your ability to ask the right questions, pushing me to connect methodological choices with practical impact, and to insist on clear thinking has been invaluable.

Finally, I owe the biggest thanks to my family and friends. To my parents, José and Carolina, and brother Rafael, for unconditional support, patience through deadlines, and for reminding me to keep perspective. Also, to friends in Brazil and Italy who celebrated small wins along the way. This work is as much yours as it is mine.

Table of Contents

List of Tables

List of Figures

Acronyms

ATC

Apparent Tardiness Cost

ATCS

Apparent Tardiness Cost with Setups

AVPRO

Average Processing Time per Operation

CI

Confidence Interval

COVERT

Cost Over Time

CR

Critical Ratio

CRSPT

Composite dispatching rule combining CR and SPT

CTDE

Centralized Training with Decentralized Execution

CJ-Shared

Chronological Joint reward with shared operation mode

CJ-Ind

Chronological Joint reward with individual operation mode

DDQN

Double Deep Q-Network

DJSS

Dynamic Job Shop Scheduling

DJSSP

Dynamic Job Shop Scheduling Problem

DQN

Deep Q-Network

DRL

Deep Reinforcement Learning

EDD

Earliest Due Date

FIFO

First-In-Frist-Out

GP

Genetic Programming

HL

Hierarchical Learning

HLA

High-Level Agent

HRL

Hierarchical Reinforcement Learning

HH

Hyper-Heuristics

HL-MARL

Hierarchical Multi-Agent Reinforcement Learning

HL-RS

Hierarchical Learning Rule Sequencing

JSSP

Job Shop Scheduling Problem

LWKR

Least Work Remaining

LLA

Low-Level Agent

MARL

Multi-Agent Reinforcement Learning

MS

Minimum Slack

MDD

Modified Due Date

MDP

Markov Decision Process

MLP

Multi-Layer Perceptron

MOD

Modified Operational Due Date

MR

Minimal Repetition (per-machine slot encoder)

NAT

Normalized Average (Cumulative) Tardiness

NCT

Normalized Cumulative Tardiness

NPT

Next-Operation Processing Time

PDR

Priority Dispatching Rule

PT

Processing Time

PTWINQS

Composite dispatching rule combining *PT*, *WINQ*, and *Slack*

RL

Reinforcement Learning

SMDP

Semi-Markov Decision Process

SPT

Shortest Processing Time

Tard-Shared

Tardiness based reward shaping with shared operation mode

Tard-Ind

Tardiness based reward shaping with individual operation mode

TD

Temporal Difference

WINQ

Work In Next Queue

WIP

Work In Progress

WR

Win Rate (fraction of runs with the best result)

Chapter 1

Introduction

Efficient scheduling of production activities has been a central concern in manufacturing and operations management for decades. Among the various scheduling environments studied in the literature, the Job Shop Scheduling Problem (JSSP) stands out as one of the most representative and challenging. The JSSP models a manufacturing system in which a finite set of jobs, each consisting of a predefined sequence of operations, must be processed on a set of machines or resources. Each operation is assigned to a specific machine, and the order of operations within a job must be respected. The central task of the scheduler is to determine a feasible allocation of jobs to machines over time, with the ultimate goal of optimizing certain performance criteria, such as minimizing the total completion time (makespan), average flow time, the number of tardy jobs or a combination of these criteria [1].

The importance of the JSSP stems from its direct applicability to real-world production systems, particularly in industries where products are highly customized, and production volumes are relatively low. Examples include tool manufacturing, metal processing, semiconductor production, and other high-mix, low-volume environments. In such settings, scheduling plays a critical role in determining the efficiency of resource utilization, throughput levels, and the ability of firms to meet delivery commitments. Poor scheduling can lead to increased work-in-progress (WIP), excessive lead times, higher operating costs, and reduced customer satisfaction [2].

From a computational perspective, the JSSP is also of significant interest due to its complexity. It belongs to the class of NP-hard combinatorial optimization problems, meaning that the solution space grows exponentially with the number of jobs and machines involved [3]. This complexity makes it impractical to rely solely on exact optimization methods for realistically sized problems, as computation times can become prohibitively large. As a result, the JSSP has served as a fertile testbed for the development and benchmarking of heuristic, metaheuristic, and hybrid approaches, such as genetic algorithms, tabu search, simulated annealing,

and more recently, RL methods.

Finally, the study of job shop scheduling is not only limited to academic interest but also aligns closely with the increasing demands of modern manufacturing. Global competition, mass customization, and the emergence of Industry 4.0 paradigms have intensified the need for highly efficient and responsive scheduling solutions [4]. Its enduring relevance arises from the dual nature of the problem: it is computationally difficult and, at the same time, directly connected to the operational efficiency of modern industrial environments.

1.1 Static vs. Dynamic Scheduling

The classical formulation of JSSP assumes a static environment, where all relevant information is known in advance and remains fixed throughout the planning horizon. In this setting, the complete set of jobs, their operations, processing times, and machine requirements are fully specified at the outset. Under such assumptions, the scheduler can construct a global and comprehensive plan before execution begins, with the expectation that the schedule will remain valid until all jobs are completed. This static perspective, while mathematically convenient, is largely a fragile abstraction from reality [5].

In practice, manufacturing and production systems operate in dynamic environments, where unexpected events continuously alter the conditions under which scheduling decisions are made. The Dynamic Job Shop Scheduling Problem (DJSSP) captures these realities by explicitly considering changes and uncertainties that occur during the execution of the schedule. These changes may originate from multiple sources. In many industries, customer orders arrive unpredictably, often with urgent due dates that were not accounted for in the initial plan. For example, in a metalworking shop, a high-priority repair order may arrive and need to be inserted immediately into an already crowded production schedule. Moreover, equipment may fail unexpectedly, rendering a machine unavailable for a certain period. This disruption not only delays the jobs assigned to that machine but also affects downstream operations, creating cascading effects across the schedule. A further source of change is the fact that in real systems, processing durations are rarely constant. Factors such as operator performance, material inconsistencies, or technical adjustments can lead to significant deviations from planned times. Even small variations may accumulate and compromise the feasibility of the original schedule. Finally, as a last source of change, customers may cancel orders or request changes in product specifications. In such cases, the initial schedule may no longer be valid, and resources allocated to canceled tasks must be reassigned promptly. These examples illustrate the core distinction between the static JSSP and the DJSSP [5].

The shift from static to dynamic settings also fundamentally alters the criteria for evaluating schedules. While static scheduling often focuses solely on efficiency-oriented objectives, such as minimizing makespan or total flow time, dynamic scheduling must additionally consider attributes such as robustness, the ability of a schedule to absorb disturbances without significant degradation, stability when minimizing unnecessary changes to previously assigned operations, and responsiveness which is the ability to quickly adapt to new information [6]. Balancing these sometimes conflicting objectives is a core challenge of the DJSSP.

Moreover, this distinction has important methodological implications. Algorithms that perform well in static environments, such as exact methods, mathematical programming are often inadequate in dynamic settings due to their computational cost and lack of adaptability. Instead, dynamic environments require approaches that can generate good solutions rapidly and update them in real time. Examples include dispatching rules, which prioritize jobs according to simple heuristics, reactive rescheduling methods, which revise the schedule when disruptions occur, rolling horizon strategies, which periodically re-optimize a subset of the problem, and learning-based techniques, which leverage historical data or online feedback to improve scheduling performance under uncertainty.

1.2 Relevance & Motivation

The study of scheduling problems, and in particular the DJSSP, holds great significance for both academic research and industrial practice. From a practical standpoint, scheduling directly influences critical aspects of production systems, including lead times, resource utilization, throughput, and the ability to meet delivery deadlines. In an increasingly competitive and customer-driven environment, organizations are required not only to operate efficiently but also to demonstrate high levels of flexibility and responsiveness. This makes effective scheduling strategies an indispensable component of operational success.

The relevance of the DJSSP has grown even more pronounced with the advent of Industry 4.0 and the digital transformation of manufacturing systems. Modern production environments are characterized by a high degree of connectivity, automation, and data availability, enabled by technologies such as the Internet of Things (IoT), cyber-physical systems, and real-time analytics. These advances have amplified both the opportunities and the challenges associated with scheduling. On the one hand, richer data streams and computational power create possibilities for more sophisticated and adaptive decision-making. On the other hand, the volatility and unpredictability of global markets, combined with shorter product life cycles, customized production demands, and complex supply chain interactions, place unprecedented pressure on scheduling systems to remain robust and adaptable

under dynamic conditions.

Traditional optimization approaches often struggle to cope with the scale and dynamism of modern production systems, creating a need for novel solution strategies that can reconcile efficiency with adaptability. This has motivated growing interest in heuristic and metaheuristic methods, multi-agent systems, and machine learning based techniques (including RL) that can exploit feedback from the environment to improve scheduling performance [7].

It is in this context that the present work is inserted. The objective is to contribute to the ongoing search for methods that are both computationally efficient and operationally realistic, bridging the gap between theoretical scheduling models and the practical demands of modern industry. By addressing this challenge, the project aims to extend the work developed in [8], leveraging a Hierarchical Learning approach on top of the developed Multi Agent Reinforcement Learning framework. In doing so, it becomes possible to better understand how machine learning approaches figure in comparison to more classical approaches and also how they can be further extended or adapted to achieve better results. In this way, it was possible to:

- Study and understand how and where state of the art machine learning approaches are situated in comparison to classical approaches.
- Design a Hierarchical Learning approach that enhances current machine learning approaches, improving how much the overall tardiness can be minimized.
- Introduce a reward boosting mechanism that softly biases per-decision rewards towards the dominant classical sequencing rules.
- Evaluate how the proposed hierarchical approach behaves under different circumstances and how well it can adapt to changes in the current shop-floor state.

Chapter 2

Literature Review

The literature on DJSS spans more than four decades and sits at the intersection of queueing, combinatorial optimization, and online control. Unlike the static JSSP, where complete job sets and routes are known a priori, DJSS unfolds under stochastic arrivals, partial observability, and event-driven decision epochs. Machines become available asynchronously, buffers evolve with upstream congestion, and performance hinges on long-horizon interactions between local sequencing choices and global WIP. Any method intended for deployment on the shop floor must therefore reconcile three demands: low-latency decisions aligned with machine events, interpretability and auditability for operational acceptance, and finally, robustness across regimes of utilization, due-date tightness, variability, and routing complexity.

This chapter reviews the main families of dispatching and learning based approaches through that lens, each section covering a specific family. Taken together, this review distills a set of design principles for DJSS controllers. These principles can be summarized into keeping the decision boundary local and event-driven, while normalizing inputs to comparable scales and masking invalid or absent actions so they are never selectable. Additionally, recent works exploit parameter sharing across exchangeable machines to improve consistency. When congestion and due-date pressure are decisive factors in the environment, studies often introduce RL with hierarchy levels. This enables that a manager style signal guides local dispatch over a specific intent on top of auditable shop-flor parameters.

This chapter is organized as follows: Section 2.1 surveys classical priority dispatching rules and their hybrids. Section 2.2 reviews learning-based dispatching. While the next section is dedicated to outline specifically RL for job-shop scheduling. Section 2.4 synthesizes multi-agent RL design choices, such as state representations, action abstractions, CTDE, shaping, and evaluation. Finally, Section 2.5 examines hierarchical learning and its temporal/spatial abstraction and training regimes. The chapter closes with a synthesis of design principles, their implications for the

baselines used later, and where this work is positioned in the literature.

2.1 Classical priority dispatching rules

Priority dispatching rules (PDRs) map a local shop floor state encompassing queue conditions, due-date pressure, and operation processing times, into a short horizon job selection at each machine. They are favored for their computational efficiency, interpretability, and robustness to model and parameter uncertainty, which explains their persistent adoption and the substantial body of simulation evidence accumulated since the 1980s [9, 10, 11]. A consistent conclusion from this literature is that no single PDR is universally dominant. Performance depends on shop load, due-date tightness, variability, routing structure, and the target objective, such as mean tardiness, tardy-job rate, or schedule stability [11, 12].

Representative single factor rules, i. e. rules that consider only one primitive, include Shortest Processing Time (SPT) that selects the job with the smallest current operation time to reduce mean flow time and congestion [13, 14]. While Earliest Due Date (EDD) dispatches the job with the earliest due date, which minimizes maximum lateness on a single machine [15, 14]. Moreover, Next-operation Processing Time (NPT) favors jobs whose next operation is short to reduce downstream blocking [12]. Least/Most Work Remaining (LWKR/MWKR), in turn, order jobs by the sum of processing times over the job's remaining route as a proxy for downstream load [14, 9]. Finally, Work in Next Queue (WINQ) prefers jobs whose next machine faces lower queued work to anticipate congestion [16, 12]. Each of these rules embodies a distinct risk posture when trying to schedule jobs across the shop-floor.

Since single factor rules can be brittle, hybrid dispatching strategies introduce limited look-ahead and explicit multi-criteria trade-offs. Notable families include Critical Ratio (CR), which prioritizes the remaining time until due date over total work remaining, balancing urgency against size [14, 9]. Minimum Slack (MS) chooses the smallest slack and emphasizes due-date adherence [14, 9]. Apparent Tardiness Cost (ATC), in turn, scales a processing-time priority by an exponential term in time-to-due-date, and ATC with Setups (ATCS) augments ATC with a setup-time penalty for sequence-dependent setups [16, 17]. Cost OVER Time (COVERT) prioritizes a cost over expected time index to balance tardiness cost against processing/queueing exposure [18, 19]. Modified Due Date (MDD) selects jobs by the larger sum of the current time and remaining processing time. while Modified Operational Due Date (MOD) select jobs by the larger due date, providing a due-date–aware look-ahead at completion time [20]. These hybrids generally enhance robustness as utilization and routing complexity increase, though benefits remain regime-dependent.

In dynamic job-shop settings, the rescheduling cadence and the triggering policy (periodic vs. event-driven) are consequential. PDRs are naturally event-driven and scale with negligible latency. However, the same reactivity can induce schedule nervousness, which is the frequent resequencing of operations caused by the frequent top job update due to small state changes. As a result, gains in tardiness or flow time often come at the cost of lower stability. Composite rules incorporating weak look-ahead terms can mitigate excessive resequencing by smoothing local choices in anticipation of downstream congestion [12].

PDRs remain strong baselines for DJSS because they are inexpensive, interpretable, and unexpectedly competitive when aligned with operating regimes. However, their performance tends to degrade outside their sweet spots and they do not manage the responsiveness–stability trade-off explicitly. This duality both motivates and challenges learning-based approaches, which must deliver consistent improvements over these baselines across regimes rather than only on average.

2.2 Learning-based dispatching rules

Learning-based dispatching denotes methods that learn a reusable mapping from state to decision and then apply it online with negligible latency. Training can be either offline or online, but the artifact you deploy is a policy/rule that you evaluate quickly when a machine becomes idle. This contrasts with metaheuristics, which perform per-instance search over complete or partial schedules and return a schedule rather than a policy. When the shop state changes, they require re-optimization or repair and thus are ill-suited to event-driven dispatch in DJSS. Learning-based approaches aim to surpass handcrafted priority dispatching rules (PDRs) while preserving shop floor acceptability in terms of auditability and ease of deployment. Two predominant methodological families dominate this line of work: hyper-heuristics (HH) and program synthesis via genetic programming (GP).

Selection based HHs learn a state conditioned policy that chooses among a portfolio of established rules, while generation based HHs construct new rules by composing features and operators. The literature covers both online and offline variants, choice function formulations, and mechanisms for maintaining portfolio diversity [21, 22]. Selection HHs are comparatively straightforward to implement, since they reuse verified rules, but their performance is bounded by the expressiveness of the portfolio. By contrast, generation HHs can uncover novel functional forms, although with heightened risks of representation bloat and overfitting [21, 22].

GP evolves priority functions over terminal features and arithmetic/logical operators, frequently incorporating feature selection or surrogate modeling to enhance generalization [23]. Advantages include compact, human-readable rules,

cross-regime generalization when appropriately regularized, and negligible online evaluation latency. Limitations include computationally intensive offline evolution, sensitivity to distributional shift, such as novel routing variability, and the need for careful terminal operator set design to preclude pathological expressions [23, 24].

A complementary strategy trains classifiers or regressors, as decision trees, SVMs, artificial neural networks, to imitate "good" dispatch choices extracted from optimized schedules. These models evaluate quickly and can encode interactions beyond linear composites. However, training is typically decoupled from deployment and optimizes a one step decision proxy rather than long-run performance, unless the learner is embedded within a closed loop simulation for feedback [25]. Since these methods act essentially over a short horizon and are optimized offline, they often underperform when long horizon queueing dynamics under non-stationary arrivals are the primary performance drivers. This setting motivates reinforcement learning (RL), which explicitly targets sequential, long run objectives [25, 23].

2.3 RL for job shop scheduling

Deep Reinforcement Learning (DRL) combines RL with deep neural networks to learn policies and/or value functions from raw or engineered observations. The agent interacts with a stochastic environment, receiving observations o_t, taking actions a_t, and obtaining rewards r_t. The objective is to maximize expected discounted return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ by adjusting the policy parameters θ considering the discount factor γ [26]. DRL acts on scheduling as sequential decision making under uncertainty. In static JSSP, agents typically select complete job sequences. In dynamic DJSS, by contrast, agents act online under partial observability with evolving WIP. This shift induces event-driven decision epochs, sparse and delayed rewards, and nonstationary dynamics [25].

A central challenge is obtaining size and order agnostic state representations. Recent approaches employ permutation invariant set encoders and graph neural networks over job–machine graphs to generalize across unseen problem sizes and routings [25, 27]. An alternative is state shaping that restricts attention to a small, informative candidate set per decision (e.g., SPT/LWKR/MS/WINQ picks), thereby yielding a fixed size input while preserving job specific features. This "minimal-repetition" compression is effective in DJSS multi-agent RL because most dispatching decisions involve few competing jobs and the construction preserves the one to one job–action mapping. For an explicit instantiation of such MR compression and a feature set including relevant job information and inter machine availability, see [8].

Given the state representation choices, the next design decision is how to expose a scheduling move as an RL action. In DJSS, two abstractions dominate. Direct

job selection considers that, at each idle event, the agent picks a job from the local queue. The action set varies with queue length, so legal-action masking and compact candidate-set compression keep decision size-agnostic while preserving the one-to-one job–action mapping [25, 8]. A second approach is more common in flexible shops and consists of route allocation. The agent selects a machine (route) and, optionally, a sequence. This is expressive but combinatorially larger and is often factorized, route-then-dispatch.

Objectives range from cumulative tardiness, tardy-job ratio, stability penalties, and changeover or waiting costs. Common training stabilizers include curricula over utilization levels, prioritized replay, and reward shaping. Credit assignment is the principal difficulty and refers to the impact of a dispatching decision being manifested several operations later. Designs that align experience tuples with the job's completion event help to narrow this assignment gap [28, 29, 8].

2.4 Multi-agent RL design choices

In a Multi-Agent Reinforcement Learning (MARL) paradigm, each agent is assigned to an individual machine or work center, aligning the learner's control boundary with physical actuation. As machines face the same environment, they are often exchangeable from a learning perspective, which motivates parameter sharing. A single policy is shared across agents, optionally conditioned on a role/ID embedding, for instance machine family, setup group, or a learned positional code. Parameter sharing reduces sample complexity, encourages consistent behavior across symmetric entities, and simplifies deployment to new shop sizes. Practical refinements include role masking to restrict actions incompatible with a machine's capabilities. Further refinements includes domain-normalized inputs by scaling time features scaled by mean processing time, due dates expressed as slack or critical ratio, to stabilize learning across instances. Finally, event time features are also included as refinements, as time since last idle or predicted next arrival time, that capture local cadence. Alternatives to per-machine agents include per-job agents, which increase action churn, credit-assignment difficulty, and hierarchical designs with a meta-controller that considers local dispatchers. Empirically, the per-machine decomposition with parameter sharing remains a strong baseline [30, 31].

Dispatching is inherently asynchronous. Decisions occur when a machine becomes idle, and the number of admissible actions equals the queue length. Two implementation details are critical. First, the environment-learner interface should be event-synchronous: a training "step" corresponds to a set of machine idle events at a common simulation timestamp. Agents without an event take a non-operation and are masked in the loss to avoid spurious gradients. Second, variable action

sets require legal-action masking: Q-values or logits for illegal actions are set to $-\infty$ before the softmax or argmax, and losses ignore masked entries. For actor critic methods, this masking effectively prevents the agent from selecting illegal actions during training and inference. Ensuring the agent only explores valid actions, improving exploration efficiency and preventing illegal moves. For value-based methods, adopting Double/Dueling Q with masked targets tends to improve stability [32, 33].

Under the required conditions to deploy decentralized execution, each agent experiences non-stationary opponents (other learners) and a drifting arrival process. Centralized training with decentralized execution (CTDE) addresses this by providing a critic $Q_\phi(s, a_{1:N})$ or a centralized value baseline with access to joint context, global WIP, in-transit jobs, utilization regime indicators, while actors consume only local observations at test time [34, 35]. Value-factorization, VDN or QMIX can be adapted to event-driven settings by factorizing over the subset of active agents at a simulation timestamp and carrying forward the previous values for inactive agents. However, strictly synchronous factorization is often mismatched to DJSS, where idle events are sparse and uneven. In practice, CTDE with parameter sharing and event-synchronous updates yields stronger convergence and lower variance than purely independent learners [8].

Even with CTDE and parameter sharing, a dispatching decision often affects tardiness only after several downstream operations, making temporal credit assignment the central challenge [25]. To mitigate this issue, different reward designs are considered in the literature. Potential-based shaping augments the environment reward with a dense term $r_t = \Delta\Phi(x_t)$ for a potential Φ such as negative total slack or negative total flow time. This preserves optimal policies while providing informative intermediate signals in event-driven settings with irregular inter event times [29, 36]. Then, a completion-aligned approach defers credit until the job is completed, which reduces reward delay and leakage across overlapping jobs in congested regimes [8]. Finally, multi-objective penalties are considered for schedule stability, setups, or changeovers, combined via scalarization or constrained RL. Curricula over utilization and due-date tightness, n-step returns, prioritized replay keyed by tardiness deltas, and conservative reward scales (to avoid gradient explosion at heavy load) are effective stabilizers [29, 36, 28].

A machine's local observations typically include queue descriptors, e.g., per-candidate features such as $t_{i,k}$, WR_i, S_i, WINQ/NPT proxies, machine status referring to its setup state, remaining setup time, blocking, and short-horizon predictions such as expected arrival time of the next upstream job. Rather than learning free-form communication protocols to share each local observation, lightweight structured sharing is often sufficient. Exposing downstream machine availability, aggregate next-queue WIP, or the earliest due date among in-transit jobs enables the anticipation of congestion with minimal complexity [8]. When coupling is

strong, message passing encoders, as graph neural networks over the machine buffer topology, or limited bandwidth attentional messages can be added, but with explicit rate limits to match shop floor latency constraints.

Among the most common action encoders, there are three most frequent designs. Direct job selection, rule selection, and routing/allocation. Direct selection maximizes expressiveness but faces variable action sets when pairing it with candidate-set compression, as an example, taking the union of SPT/LWKR/MS/WINQ picks and top-k shortest processing times. This approach yields a small action space while preserving near optimal choices in most states [25, 37]. Rule selection is size agnostic and interpretable but introduces an additional mapping from rule to job, which can blur credit assignment [38, 37]. Routing/allocation is necessary in flexible shops, here, a factored action (route, then sequence) or hierarchical policy (router $\rightarrow$ dispatcher) reduces combinatorial blow-up [39, 40].

To generalize across shop sizes and permutations, architectures should enforce permutation invariance/equivariance where appropriate. Set encoders (Deep Sets) or attention over per-job candidate features for the local queue, maintaining shared encoders for all machines (parameter sharing) with optional role embeddings, and normalization by instance-level statistics (mean/variance of processing times, due-date span). Recurrent policies, as Long Short-Term Memory, can partially compensate for partial observability, while remaining light enough for sub-millisecond inference [41].

Exploration in dynamic shop-floor control is particularly challenging under tight due dates, where suboptimal actions incur large penalties. To mitigate this, it is considered per-agent ϵ-greedy exploration with annealing schedules tied to utilization curricula, thereby tapering randomness as congestion rises and learning stabilizes. Further incorporation of entropy regularization while capping its contribution to prevent thrashing among near-equivalent choices. In addition, safe action priors are employed that bias sampling toward the top candidates induced by a baseline portfolio of PDRs, providing a pragmatic anchor when the value function remains uncertain. For deployment, guardrails are introduced to ensure operational acceptability. An ε-mixtures that interlocks the learned policy with a verified PDR, explicit negates unsafe actions, forbidding setup starts that violate stability thresholds, and short "freeze windows" that limit resequencing rates and reduce disruption on the shop floor [25].

Since events arrive irregularly, experience is time-stamped and discounted using Δt-aware horizons, with returns scaled as $\gamma^{\Delta t/\tau}$ to preserve temporal consistency across variable inter-event intervals. To respect simultaneity among distributed decision makers, transitions are batched by simulation timestamp so that concurrently acting agents contribute coherent updates. Replay is prioritized either by the absolute change in cumulative tardiness or by temporal-difference (TD) error, focusing learning on consequential events such as bottleneck releases. To control

non-stationarity in this multiagent setting, slowly updated target networks are maintained, and also it is applied a clipped importance sampling for off-policy corrections under CTDE when replay is used [28].

Given the event-driven setting with masked actions, parameter sharing, and CTDE previously described, evaluation protocols in the literature are designed to stress both performance and robustness under the same constraints. Baseline portfolios span single-factor rules, SPT, EDD, CR/MS, LWKR/MWKR, and WINQ, and composite heuristics including ATC/ATCS, COVERT, each tuned to the operating regime. Regime sweeps cover utilization levels, due-date tightness, processing-time variability, and routing structures. Metrics include normalized cumulative tardiness, tardy-job ratio, flow time, WIP, and stability indicators such as resequencing rate and deferred starts, alongside wall-clock inference time to assess deployability. Generalization is validated via zero-shot transfer to unseen sizes and routes, temporal drift in arrival rates, and heavy-tailed processing times. Statistical analysis goes beyond means, reporting win rates against the best baseline per instance, interquartile ranges, and paired bootstrap tests with effect sizes. Finally, ablation studies isolate the contributions of parameter sharing, CTDE, candidate-set compression, reward shaping, and structured inter-agent communication, following recent guidance for learning-based dispatching in job-shop environments [12, 16, 42, 27, 8].

2.5 Hierarchical learning

Hierarchical Learning (HL) introduces temporal and functional abstraction to address long-horizon credit assignment and combinatorial action spaces in DJSS. A two-level design is natural and frequent: a manager (global or area-level) sets short-horizon intents or limits, WIP caps, due-date bands, setup budgets, routing choices, while per-machine agents execute fine grained dispatching decisions that track these intents. This separation lowers the manager's decision rate, regularizes local behavior via top-down guidance, and curbs schedule nervousness through commitment windows that limit resequencing [43].

Typically, studies also consider spatial abstraction to complement this picture. Useful partitions include route then sequence, where the manager selects a routing or machine family for flexible operations and workers sequence locally under the chosen route. A second option is release then dispatch, where the manager throttles WIP via order release and workers dispatch among available jobs. Finally, setup aware batching, in which the manager proposes batch/sequence templates to reduce changeovers and workers fill templates while honoring due-date risk bounds. Interfaces are often built from soft goals, such as targets for local slack or bounds on next-queue WIP, and hard constraints, as action masks that forbid setups above

a threshold or explicit WIP caps. It is also common to adopt a from of goal conditioning at the worker level [29].

Across hierarchy levels, CTDE is frequently adopted, equipping critics with joint context (global WIP, bottleneck status, arrival-rate regime) while actors depend on local observations and the manager's goal signal. This is done to mitigate inter-level non-stationarity. Representative HL algorithms include options and call-and-return [43], option-critic for end-to-end learning of policies and terminations [44], and also goal relabeling for consistent off-policy updates [45]. Several works report pretraining workers by distillation from strong PDRs to anchor behavior in interpretable skills before fine-tuning with HL.

Attributing returns to the appropriate level is a central concern. Reported mechanisms include advantage decomposition with separate critics (often sharing encoders). They also include completion-aligned targets that defer bootstrapping to job completion to reduce leakage across overlapping options in congested regimes. Commitment-consistent bootstrapping updating high-level targets only at option termination or commit-window expiration is used to align learning with operational commitment policies [36].

To respect shop-floor latencies, observability and communication are typically lightweight. Managers consume coarse, slowly varying summaries, while workers observe candidate-level features plus the goal embedding. Upward messages indicating localized lateness risk are rate-limited, while downward directives (goal updates) are likewise updated no more than once per commitment window to avoid oscillations. Staged exploration is common: managers sample among conservative macro-intents (moderate WIP caps, routing preferences), while workers explore within legal-action masks biased by PDR priors. Safety mechanisms include ε-mixtures with certified PDR fallbacks, vetoes on hard-constraint violations, and interruption policies that allow early option termination under exogenous events (machine breakdowns, rush orders), with penalties accounted for during training.

Reported implementations keep the manager's decisions infrequent and moderately sized, whereas workers use compact encoders for a quick dispatch. Event batching and shared encoders amortize cost, and goal embeddings avoid recomputing global features at every dispatch. Evaluation typically augments the metrics in Section 2.4 with HRL-specific indicators: commitment adherence (fraction of actions within commit windows), interruption rate and penalties, manager overhead (updates per hour), and skill-utilization diversity. Baselines include flat MARL, rule-selection hierarchies (in which the manager chooses a PDR and workers apply it), and release-control heuristics paired with tuned local PDRs. Ablation studies vary option duration, interface type (goals vs. constraints), and the presence of distillation or relabeling. Representative methods adapted to event-driven DJSS include options [43], option-critic [44], hierarchical actor–critic with goal relabeling [45], and feudal networks [46].

When comparing HL to classical dispatching rules, it is noted that dispatching rules are fast and robust, but static, they adapt poorly to distributional shift and offer limited cross machine coordination [47, 9, 48]. Metaheuristics can optimize full schedules but must be re-solved frequently, incurring latency on decisions in highly dynamic shops [5, 49, 21, 22]. HL, however, learns reusable skills offline and applies them online with low latency. The manager selects a context appropriate mode, and per-machine agents are responsible for coherent local decisions consistent with that mode, preserving the agility of rules while enabling context driven adaptation and system level coordination.

Supervised approaches learn to imitate "good" decisions but require labeled targets and struggle when problem size and context vary over time. They also do not address delayed consequences. HL is reinforcement-driven and optimizes long horizon objectives under changing conditions. Learned skills can transfer across utilization regimes and product mixes without re-labeling [50].

Flat RL learns end-to-end mappings from raw states to primitive actions at every step, facing sparse and delayed reward, large combinatorial actions (job selection per machine), and multi-agent non-stationarity. In MARL, CTDE style approaches mitigate some of these issues but still leave exploration and credit assignment challenging at fine temporal scales [34, 51, 35]. HL narrows the low-level search through subgoals, shapes exploration, and improves credit assignment via aggregated feedback at the high level. Intrinsic or shaped rewards can further align local behavior with system objectives [29].

2.6 Research gaps and positioning

The literature shows a progression from basic single-factor PDRs and their hybrids, through learning-based dispatching (hyper-heuristics, GP, imitation), to flat MARL and, more recently, hierarchical designs. Event-driven DJSS imposes asynchronous decision epochs, partial observability, delayed rewards, and regime dependence (utilization, due-date tightness, routing variability). Methods that succeed operationally tend to be robust to long horizons, adaptable over stochasticity, and consistent across regimes rather than only on average.

Despite notable progress, several recurring themes emerge across the literature. First, credit assignment under asynchrony remains unsettled: completion-aligned returns and shaping techniques show promise but lack standardization, and their effects appear to vary across operating regimes. Second, action abstraction warrants clearer isolation of its benefits: while direct selection from compact candidate sets performs well, few studies disentangle its contribution relative to rule selection or routing/allocation within identical evaluation protocols. Third, the field's emphasis on tardiness and flow often eclipses other dimensions of responsiveness and stability.

Schedule nervousness and the consequences of setups and changeovers are treated less systematically. Fourth, interfaces for hierarchical reinforcement learning are frequently task-specific, more attention is needed to design anchored in auditable shop-floor levers—such as WIP caps, setup budgets, or due-date bands. Finally, generalization claims are common, yet comprehensive examinations under regime drift, in arrival rates, for instance, and in the presence of heavy-tailed phenomena, remain comparatively rare.

Motivated by these gaps, this thesis proposes a strong Hierarchical MARL baseline to develop a scheduling strategy for the DJSSP. The proposed Hierarchical MARL baseline considers per-machine agents with parameter sharing leveraging a CTDE paradigm. Additionally, it is evaluated against tuned PDR portfolios across regimes, including stability metrics. This is done to achieve the main goal of minimizing the overall tardiness in a dynamic job shop environment.

Chapter 3

Job Shop Scheduling

The JSSP is a classical combinatorial optimization challenge in which a set of jobs must be processed by specific machines. Each job is composed of a sequence of operations. Furthermore, every operation has its own fixed processing time and requires exclusive use of the designated machine. Operations must be processed respecting a predefined execution order. Finally, only after all operations that compose a job are completed, the job can then be considered processed.

When formulated in this way, the JSSP considers static conditions on the behavior of the shop floor. Even though the static approach is widely studied [1, 5], it rarely reflects how real world manufacturing environments behave. Modern production systems face dynamic conditions, such as sudden job arrival, abrupt machine failure or even inconsistent processing time. Under this scenario, a static job shop model quickly becomes invalid and the need to model the shop floor dynamically arises.

It is in this context that the DJSSP is inserted. It is an extension of the JSSP that considers the dynamicity of real world manufacturing environments. In the DJSSP, the set of jobs in the system changes over time as new jobs arrive and existing jobs are completed, making the problem size non-constant and the scheduling horizon potentially unbounded.

This Chapter discusses the DJSSP and how it differs from the JSSP in Section 3.1. Then it presents the DJSSP model specifications and motivate the choices made in Section 3.2. Finally, a discussion of how the model behaves is presented in Section 3.3, both in terms of how it functions and how it is affected by its parameters.

3.1 Dynamic Job Shop Scheduling

The distinction between the static and dynamic variants of the job shop scheduling problem lies primarily in how they handle the evolution of the shop floor state

over time. In the static JSSP, all jobs, their processing routes, and associated parameters are known in advance, enabling the construction of a complete schedule before execution begins. In contrast, the DJSSP operates under continuous change, new jobs may arrive unexpectedly, processing times can vary, and machine availability may be disrupted. These dynamic events require scheduling decisions to be revised, often without full knowledge of future events. Figure 3.1 illustrates the conceptual difference between static and dynamic job shop scheduling, highlighting the arrival of a new job and the need for continuous adaptation required in the latter.

On the left-hand side of Figure 3.1, it can be seen that all jobs that will arrive at the shop floor are known in advance, which allows for a complete schedule to be built before execution. However, on the other side, only a fraction of the jobs that can arrive at the shop floor are known. In this example, a new job arrives during execution and then it is needed to reschedule in order to resume job processing. The arrival of the new job illustrates the impact of dynamic events on the shop floor.

In real world manufacturing, three recurring situations illustrate the need to model the shop floor dynamically. First, make-to-order or built-to-order environments experience stochastic order arrivals with custom attributes that are not homogeneous across time, so the set of jobs is not fully known at release time and schedules must consider both non-peak and peak periods [5, 52]. Second, equipment disruptions, both unplanned breakdowns and planned maintenance windows, invalidate precomputed sequences and trigger rescheduling to restore production feasibility and performance [49, 53]. Third, stochastic processing times and delay effects that may occur due to product variability, rework, or operator quality fluctuations, undermine deterministic assumptions, requiring policies that adapt as actual processing times deviate from estimates [54, 55].

Under this dynamic environment, the main objective of the DJSSP then becomes to minimize performance metrics such as makespan or tardiness. The former being the total time to complete all jobs, while the latter refers to the amount of time elapsed after a job's due date. Minimizing metrics like these means enhancing the utilization of available resources of the shop floor, and also increasing responsiveness of the productive system.

3.2 Dynamic Model Specifications

Since this project is inserted in the context of extending the work of [8], the shop floor model is maintained the same. The model built by [8] considers generic scenarios with random job arrivals, and it also considers extended simulation horizons for both training and testing. Random arrivals inject stochasticity and a

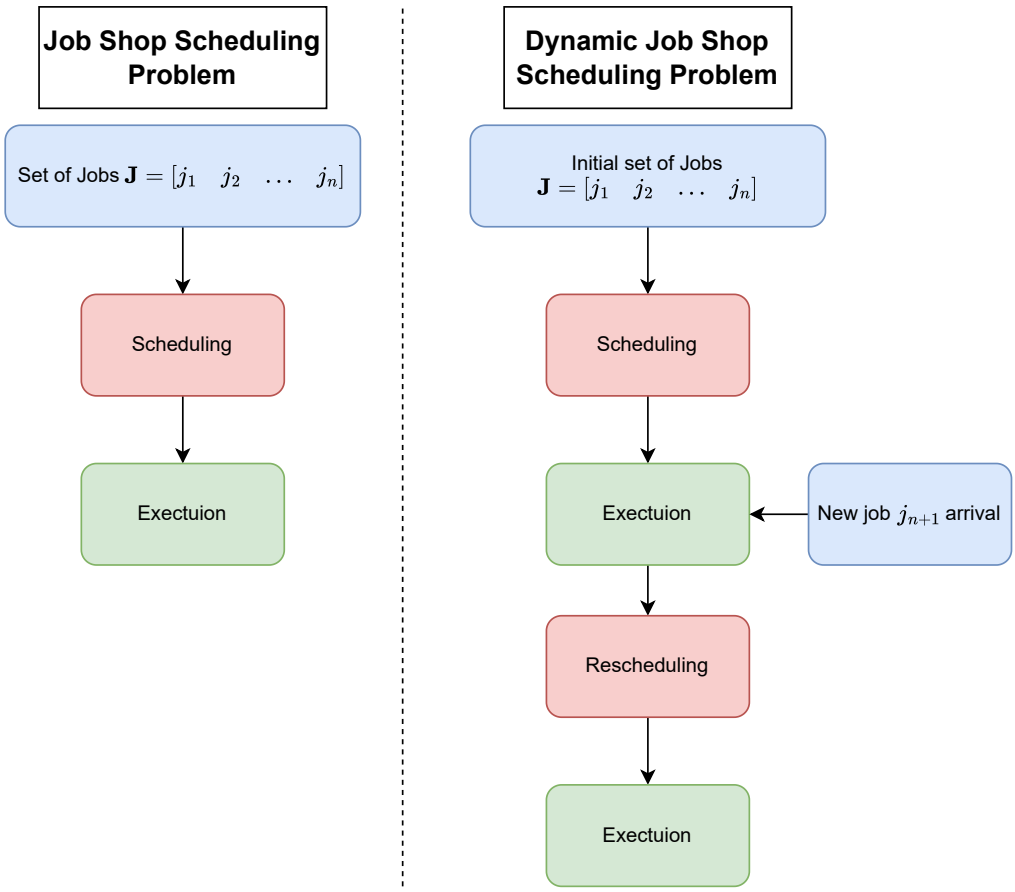


Figure 3.1: Conceptual contrast between a static Job Shop Scheduling Problem (left) and its dynamic variant (right). In the static case, the full set of jobs and routes is known a priori, enabling an off-line schedule executed without further change. In the dynamic case, stochastic job arrivals (illustrated by the new job arrival j_{n+1}) and other disturbances require on-line rescheduling during execution, so only a partial schedule is valid at any time.

diverse mix of jobs, while long-run simulations expose periods of uneven utilization of the system's capacity. Moreover, each job must be processed by each machine exactly once. Scheduling is carried out in a discrete event simulator by decentralized agents acting at the machine level. A scheduling operation occurs whenever a machine becomes idle and at least two jobs are waiting to be processed, then the local agent selects the next operation with the objective of minimizing job tardiness. The selected operation must be among those that compose one of the jobs waiting

to be processed.

The built environment obeys the following modeling assumptions and constraints:

1. Operations are non-preemptive and, once started, run to completion.
2. Rework and re-entrant job flows are not considered.
3. Machine setup times and job transfer/movement times are neglected.
4. Jobs are independent and each machine can process only one job at any given time.
5. Jobs queue buffers are considered to have unlimited capacity and are not explicitly modeled.

In this model, the goal of the scheduling strategy is to minimize the cumulative tardiness CT. It is considered as the objective to be minimized because it captures service level performance as experienced both by customers and supply chain partners, additionally, it ties scheduling decisions to customer satisfaction. In a competitive market setting where loyalty is critical, tardiness-based indicators are key to evaluating schedule quality [1, 2].

Formally, over the evaluation horizon, cumulative tardiness is derived as

$$CT = \sum_{i=1}^{N} CT_i, \tag{3.1}$$

where CT_i represents the tardiness of J_i, N represents the total number of jobs processed and can be obtained as

$$CT_i = \max(C_i - D_i, 0), \tag{3.2}$$

where C_i denotes the completion time of job J_i, D_i denotes the due date of job J_i.

In a dynamic job shop with stochastic arrivals and long horizons, service level is fundamentally about meeting due dates and not merely finishing a finite batch quickly. The cumulative tardiness CT directly measures due-date adherence across all decisions over time and is therefore the natural objective for this setting. By contrast, makespan is well-posed in static, one-shot instances but becomes ill-defined in ongoing dynamic environments. Optimizing makespan can also ignore due-date urgency, encouraging schedules that clear the final job early while allowing many intermediate jobs to miss deadlines, which is undesirable in markets where customer satisfaction and partner SLAs depend on on-time delivery. The DJSS literature emphasizes that dynamic disruptions continually change problem specifications and trigger rescheduling, thus, performance should reflect rolling service quality, not one terminal completion time.

Other common metrics have their own pitfalls here. Mean flow time promotes low WIP and short cycles but does not penalize lateness. Rules that excel on flow-time objectives often behave differently under tardiness objectives, so optimizing one does not guarantee performance on the other. Comparative studies explicitly show priority-rule rankings shifting between flow-time and tardiness criteria, underscoring the need to choose the metric that matches the operational goal (on-time delivery).

Number of tardy jobs counts delays but is insensitive to how late jobs are. A single severely late order is treated the same as a marginal miss. In contrast, CT captures both incidence and magnitude of lateness and is therefore a more discriminative and optimizable training signal for RL. Finally, from an RL design standpoint, CT aggregates the long-run effects of local event-driven decisions and admits principled credit assignment, enabling decentralized agents to learn policies that consistently improve due-date performance in dynamic contexts.

Since the DJSSP is a stochastic problem, where safe, repeatable experimentation on a real shop floor is impractical, both the development and validation of scheduling strategies are simulation based. The simulation environment must model the stochastic dynamics of job arrivals while enabling large-scale, safe, and reproducible experimentation that is necessary for training and comparing RL-based policies [5, 25]. Alongside with its goals and objectives, a proper simulation environment can be created and defined by three key factors, which are the job shop utilization rate, the job processing time and the due date tightness.

A job shop model utilization rate is defined as the proportion of the available processing capacity of the shop that is actually being used to process jobs. In the context of a DJSSP in which every job visits every machine exactly once, the utilization rate can be obtained by the ratio between the expected processing time of operations on each individual machine and the expected time interval between job arrivals as [8]

$$\mathbb{E}(\text{utilization rate}) = \frac{\mathbb{E}(t)}{\mathbb{E}(\Delta t_j)}, \tag{3.3}$$

where $\mathbb{E}(t)$ denotes the expected processing time of operations per machine, $\mathbb{E}(\Delta t_j)$ represents the expected time interval between job arrivals and Δt_j represents the time interval between job arrivals. The utilization rate of a job shop is the core value to create training and test scenarios. A higher utilization rate value means that more jobs arrive at the job shop at random and a more robust scheduling strategy is needed. Whereas, with a lower utilization rate value fewer jobs will arrive, which opens space for a less demanding system. By fixing the utilization rate exogenously, it is possible to shape the congestion regime under which policies are trained and evaluated. This treats utilization as a design parameter defining the environment, while the optimization objective for the policy remains cumulative tardiness CT. In this work, the job shop model was simulated under utilization values of 70%, 80% and 90%, as for higher values, a robust strategy is needed the

most.

Processing times are defined by the duration required to execute job J_i on machine M_k, and it is denoted by t_{i,M_k}. There are different ways to model the processing time, some studies [27] consider specific distributions obtained empirically, while others [12, 25] consider an uniform distribution to derive t_{i,M_k}. The uniform distribution is most commonly found in the literature since it opens space for an easier simulation of different scenarios, while also being easily comparable and transferable across studies and applications. Following the common practice, the processing time is drawn from a uniform law,

$$t_{i,M_k} \sim U[a, b], \tag{3.4}$$

which induces an expected processing time $\mathbb{E}(t) = (a + b)/2$. The choice of the interval $[a, b]$ directly influences the shop's congestion. For a fixed job arrival dynamics, i. e. a constant $\mathbb{E}$(interval) value, a larger $\mathbb{E}(t)$ increases the expected utilization rate, as seen in Eq. (3.3). It also raises queue lengths and amplifies how often a machine actually has to perform a sequencing decision. However, a smaller $\mathbb{E}(t)$ reduces the pressure on the system, diminishing jobs waiting time and alleviating tardiness pressure. In this work, to balance realism and comparability across scenarios, processing times are drawn as done in [8]

$$t_{i,M_k} \sim U[1{,}50], \tag{3.5}$$

which is a moderate range that yields diverse machine loads and nontrivial scheduling dynamics [8].

Due-date tightness controls the urgency of jobs by regulating the initial slack at arrival. Considering job J_i, its due date is defined by

$$D_i = \text{NOW} + TTD_i^{\text{initial}} \tag{3.6}$$

where D_i represents the due date of job J_i, NOW represents the current time at when the job J_i arrives at the system and TTD_i^{initial} represents the initial time till due of job J_i. In this simulation based context, the time till due of a job is obtained by multiplying the sum of each of its operations processing times by a tightness factor. The tightness factor α_i represents the amount of calendar time job J_i is given relative to its own total processing time and is drawn from a uniform distribution whenever a job arrives at the shop floor. The initial time till due of job J_i is set as

$$TTD_i^{\text{initial}} = \alpha_i \sum_{j=1}^{O_i} t_{i,j}, \tag{3.7}$$

$$\alpha_i \sim U[1, \text{upper limit}], \tag{3.8}$$

where TTD_i^{initial} represents the time till due of job J_i, O_i represents the total number of operations of job J_i, and $t_{i,j}$ represents the processing time of the j-th operation of job J_i. Finally, the slack time, which is used to regulate the due date tightness, is a measurement of the amount of extra time a job has available beyond the time it still needs for processing. Using its definition, the initial slack time of job J_i can be obtained as

$$S_i^{initial} = TTD_i^{initial} - WR_i^{initial} \tag{3.9}$$

$$= \alpha_i \sum_{j=1}^{O_i} t_{i,j} - \sum_{j=x}^{O_i} t_{i,j} \qquad \text{on initial condition: } x = 1 \tag{3.10}$$

$$= (\alpha_i - 1) \sum_{j=1}^{O_i} t_{i,j}, \tag{3.11}$$

where $S_i^{initial}$ represents the initial slack time, $WR_i^{initial}$ represents the amount of work remaining time of job J_i, x represents the index of the next operation of job J_i. From (3.11), it is possible to control the due date tightness simply by adjusting the tightness factor α_i. A lower value of α_i means that jobs have tighter deadlines which increases the likelihood of tardiness and makes urgency-aware rules more impactful. Additionally, a very tight α_i can cause all rules to under perform, while looser α_i values can make the problem trivial to solve, since there are many on time jobs. Choosing the range for α_i balances how hard the problem is going to be. In this work α_i is drawn from a uniform distribution U[1,3], since on average it produces a slack time equal to the job's total processing time, as follows

$$\mathbb{E}(S_i^{initial}) = (\mathbb{E}(\alpha_i) - 1) \sum_{j=1}^{O_i} t_{i,j} = (2 - 1) \sum_{j=1}^{O_i} t_{i,j} = \sum_{j=1}^{O_i} t_{i,j}, \tag{3.12}$$

where $\mathbb{E}(S_i^{initial})$ represents the expected value of the initial slack time of job J_i and $\mathbb{E}(\alpha_i)$ represents the expected value of the tightness factor of job J_i. It is worth mentioning that any uniform distribution that has the difference between its limits equals to two would produce the same effect.

An additional factor that characterizes the dynamic model is its size, represented by the number of machines M, and the induced complexity of scheduling decisions. Prior studies indicate that the relative performance of dispatching rules is not overly sensitive to system size, and that a job shop with six or more machines can already be considered complex [12]. Complementarily, [56] relates problem difficulty as the ratio of jobs to machines, suggesting a complex regime when the ratio exceeds 2.5, and a hard regime for instances with $N \geq 15$, $M \geq 10$, and a total number of operations above 200.

In line with [8], we adopt a shop with $M = 10$ machines. This choice satisfies widely cited complexity thresholds while remaining computationally tractable

for large scale simulation. Moreover, the extended simulation horizons used in both training and testing produce job sets that comfortably exceed the hardness thresholds in terms of total jobs and jobs to machines ratio, ensuring that learned policies are exercised under demanding conditions.

Table 3.1 summarizes the dynamic model specifications. This parameterization balances realism and experimental control: (i) utilization is tuned through the inter arrival process to stress different congestion regimes; (ii) service urgency is governed by α_i, shaping initial slack and tardiness pressure; and (iii) system size and simulation horizon jointly ensure a challenging yet reproducible benchmark for training and evaluating RL-based schedulers.

Aspect	Possible configurations settings
Number of machines	$m = 10$
Routing	Each job J_i has $O_i = m$ operations (one per machine)
Processing times	$t_{i,k} \sim U[1,50]$
Due date tightness	$\alpha_i \sim U[1,3]$
Utilization rates	$\{0.70, 0.80, 0.90\}$
Dispatch trigger	When a machine becomes idle and its queue length ≥ 2
Objective	Minimize cumulative tardiness T_{sum}
Operating regime	Discrete-event, random job arrivals & long-run horizons

Table 3.1: Dynamic job shop parameters and configurations used throughout. Jobs visit each machine exactly once; processing is non-preemptive; no setups/transfer times, re-entrance, or rework are modeled. Target utilization rates (70%,80%,90%) are achieved by tuning the inter-arrival process; processing times are sampled as $t_{i,k} \sim U[1,50]$; due-date tightness draws $\alpha_i \sim U[1,3]$ imply initial slack $(\alpha_i - 1) \sum_j t_i^j$. The objective is to minimize cumulative tardiness CT.

3.3 Dynamic Model Behavior

This section describes how the simulated shop evolves over time under random job arrivals and machine-level dispatching. It also formalizes the event flow and decision epochs, summarizes the emergent regime-level patterns observed in long-run simulations, and discusses their implications for learning-based schedulers.

The environment is driven by a discrete event simulation that considers two types of events, job arrivals and operation completions. Let $J_{k,i}$, $k \in [1, 2 \ldots, K]$ and $i \in [1,2,\ldots,N]$, represent the job processed by machine M_k. Let $\mathbf{J}^k = [J_{k,1}, J_{k,2}, \ldots, J_{k,N_k}]$ denote the set of all jobs that are queued to be processed by M_k, with N_k representing the number of jobs in M_k queue and $\mathbf{J} = [J_1, J_2, \ldots, J_N]$

represent the set of all jobs. A sequencing decision at machine M_k is required when the machine becomes idle and $|\mathbf{J}^k| \geq 2$, where $|\cdot|$ denotes the cardinality operator. These decisions are called active. However, if $|\mathbf{J}^k| = 1$, then the decision is called passive. With this approach, each machine makes a decision, i. e. selects the next operation, only when there is an actual choice to be made. After the decision is made, the machine can then process the operation. Then, at operation conclusion, it becomes idle once again and the decision process is repeated. The event loop for a machine M_k is illustrated in Figure 3.2, it can be seen that the event loop considers and inserts arriving jobs on the set of all jobs $\mathbf{J}$. Additionally, for each machine M_k, the event loop consists of checking the number of jobs in its queue, making a scheduling decision and processing the selected operation. Then, the machine M_k propagates the concluded operation and the job moves to the next machine or exits the system. Finally, the machine can check on its queue for waiting jobs once again to restart the loop and process a new operation.

At each event time t, the system evolves as follows:

1. If a new job J_i arrives, it is instantiated with route and operation times. Its due date is set via the tightness mechanism in Section 3.2, and it enters the first machine's queue.

2. Machine M_k finishes its current operation. If $|\mathbf{J}^k| \geq 2$, an active decision selects the next operation to process. If $|\mathbf{J}^k| = 1$, a passive decision is made and the only candidate starts immediately. If $|\mathbf{J}^k| = 0$, the machine idles.

3. The job with the just completed operation moves to the next machine in its route (or exits if finished), potentially triggering new active decisions at other machines.

It is only when a job is completed that tardiness, the objective to be minimized, is realized. Since machines decide locally and asynchronously, the eventual T_i reflects a chronological chain of earlier decisions by different machines along the job's route. A single dispatching choice at machine M_k can affect downstream arrival times at following machines and induce congestion that only manifests several operations later, often after other machines have made additional interacting choices. This delayed and distributed effect underlies the credit assignment challenge in decentralized learning when developing a scheduling strategy. Credit assignment refers to the problem of mapping a sparse, delayed, team-level outcome, in this context, the realized tardiness T_i at a job's completion, back to the specific, earlier local decisions that produced it, so each machine can update its policy in proportion to its responsibility.

Moreover, random arrivals and finite processing capacity create alternating behaviors of congestion and slack across machines. The fraction of active decisions increases with utilization rate since having to choose among waiting jobs becomes

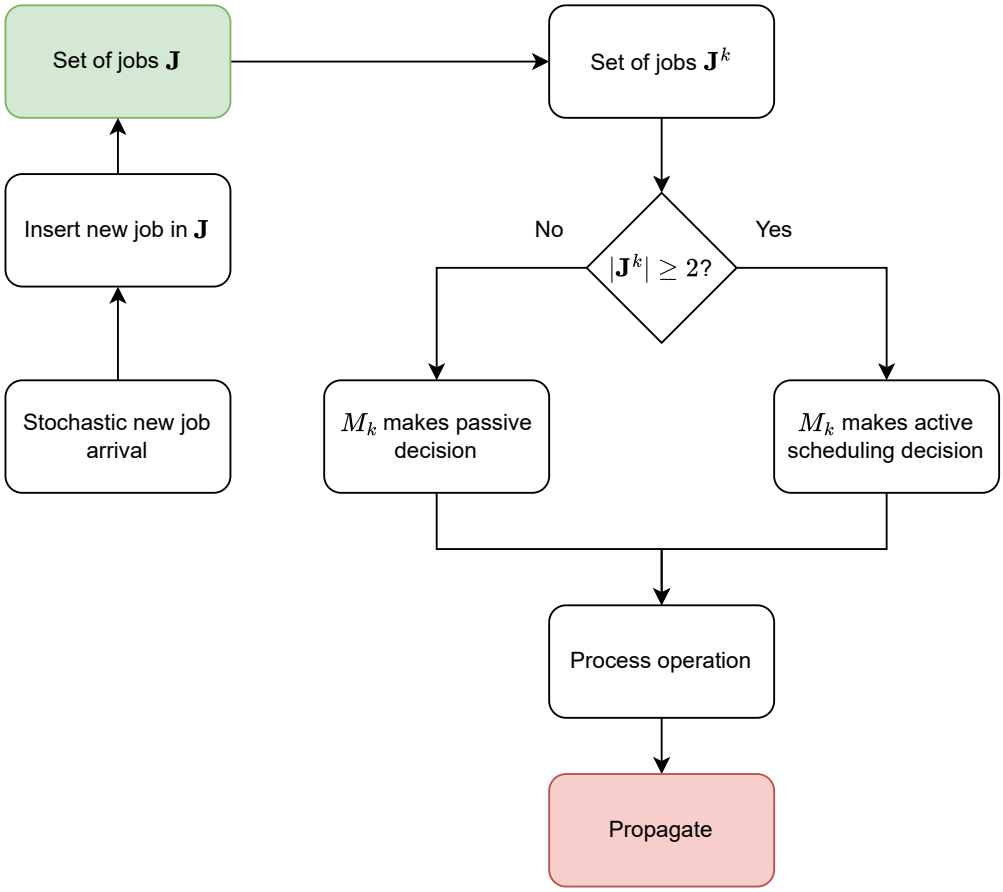


Figure 3.2: Event-driven evolution of the DJSSP simulation. Exogenous job arrivals are inserted into the global job set and the appropriate machine queues. When a machine becomes idle, it makes an active sequencing decision if its queue has more than one job, a passive decision if exactly one job is waiting, or idles otherwise. The selected operation runs non-preemptively and upon completion the job moves to its next machine (or exits), potentially triggering new decisions elsewhere.

more frequent. However, it tends to decrease as the number of machines grows due to the work being spread more evenly across stations. Both the tardy rate (number of tardy jobs over total number of jobs) and the average tardiness increase with the utilization rate due to a collateral increase in congestion. Although when comparing with the number of machines, a non-monotonic behavior is observed. More machines reduce per-station congestion, but also create more places to queue

and more overhead. The balance of these effects makes both the tardy rate and average tardiness non-monotone in the number of machines. Finally, even at moderate loads, non-negligible tardiness appears when arrivals meet busy machines since slack time is consumed while a job waits for service, even if scheduling choices were reasonable.

This dynamic behavior imposes structural requirements on any scheduler, regardless of how the underlying control problem is solved. First, decisions occur at irregular, event-driven epochs—precisely when a machine becomes idle with $|\mathbf{J}^k| \geq 2$, so the problem is naturally a semi-Markov decision process (SMDP), not a fixed-time-step MDP. Therefore, Policies should be defined on events rather than clock ticks. Second, the action set is variable and instance dependent: at each machine, it is the current queue, which changes in size and composition over time. Algorithms must handle masking of invalid or absent actions and compare candidates on a common scale. Third, effective policies exploit locality and symmetries, sequencing depends on features of the few top candidates in a queue, and invariance to job permutation should be respected. This argues for representations that are size-agnostic and permutation-aware with parameter sharing across machines. Fourth, objectives are sparse and delayed, a job's tardiness CT_i is realized only at completion after many interacting decisions along its route, so any method needs a principled way to attribute outcome responsibility along a job's chronological path. Finally, since decisions are triggered by machine-idle events, the scheduler must deliver low-latency choices even under high utilization, which favors policies that evaluate quickly at run time.

Within this landscape, reinforcement learning is one practical way, though not the only way, to solve the induced SMDP. Exact dynamic programming is intractable due to the enormous state space (asynchronous queues across machines, due-date states, etc.) and variable action sets. Rolling-horizon Mixed Integer Linear Programming approaches can encode lookahead and side-constraints, but they often incur nontrivial solve times at each event and require careful re-optimization under stochastic arrivals and disruptions. Dispatching rules are extremely fast and interpretable but either rely on fixed templates or require manual tuning to changing regimes. RL offers an amortized alternative: heavy computation is shifted to simulation-based training, yielding a policy that maps state to action rapidly at deployment. Moreover, the event-driven SMDP formulation aligns well with RL design. Decentralized agents with parameter sharing exploit machine exchangeability while keeping inference local. Invalid or absent actions are masked so only feasible jobs are ranked. Experience is stored and updated along each job's chronological path so that sparse, completion-time tardiness can be shaped into learnable signals. Hierarchical extensions can place slower, global decisions above fast local sequencing to balance agility and schedule stability. RL is adopted here because it meets the latency and adaptability requirements of the dynamic shop

while leveraging simulation to learn policies that generalize across tightness and routing variability. Nonetheless, it complements rather than excludes alternatives and can be hybridized with rule-based priors or rolling-horizon safeguards when beneficial.

Chapter 4

Methodology

Considering the dynamic job shop model defined in Section 3, a MARL scheduler was built by [8] to solve the scheduling problem. In this reinforcement learning approach, each machine acts as an autonomous agent trained to minimize cumulative tardiness under random job arrivals. A CTDE scheme with parameter sharing was adopted to mitigate non-stationarity during learning. Experience was structured using a chronological joint action view with knowledge based reward shaping so that realized tardiness can be attributed to the actions that caused it, mitigating the credit assignment issue.

This chapter also extends the MARL framework to a hierarchical setting. Empirically, the analysis with classic dispatching rules suggests that composite rules such as PTWINQS and CRSPT achieve strong win rates under moderate and high utilization levels. Motivated by these findings, this chapter proposes a Hierarchical Multi-Agent Reinforcement Learning (H-MARL) approach that leverages a high level meta policy to select or blend such modes into the scheduling strategy. At the same time, low level decentralized agents perform job sequencing decisions guided by the meta policy, preserving the chronological credit assignment developed by the reward shaping mechanism introduced in Section 4.3.

DJSS features long decision horizons, asynchronous events, sparse and delayed rewards (tardiness realized only at completion), and decentralized interactions among machines. Considering such settings, traditional learning, whether single-agent or multi-agent, often struggles with exploration, credit assignment, and non-stationarity. In this context, Hierarchical Learning (HL) is especially useful, since it addresses these issues by introducing abstraction levels. Higher levels of abstraction are responsible for selecting modes, goals, or extended options, while lower levels of abstraction execute the induced sub tasks over multiple steps [43, 57, 50]. For DJSS, this allows a better understanding of the current job shop state and enables more informed sequencing decisions, improving stability and responsiveness under volatility.

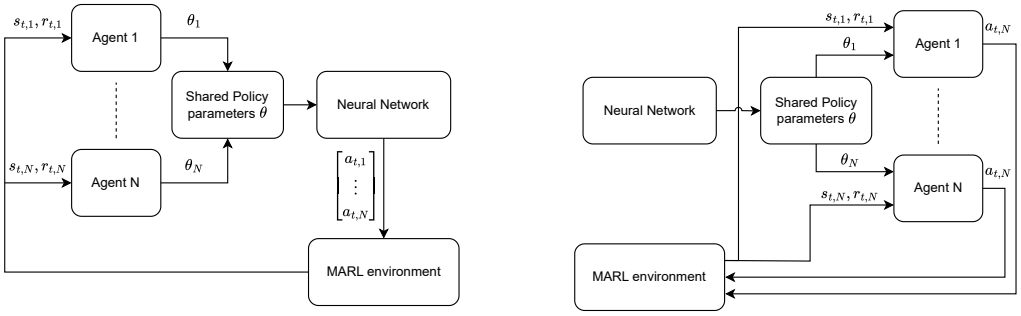
The remainder of this chapter first specifies the RL framework in Section 4.1, then it discusses the Minimal Repetition state representation in Section 4.2. It proceeds to explain how each agent's reward is constructed in Section 4.3. Then, Section 4.4 transitions to the HL approach and states how it differs from more conventional learning techniques. Section 4.5 discussess how HL is inserted in the context of the DJSSP. Section 4.6 contains a detailed explanation of how the high level state and low level states are defined, while Section 4.7 details the reward mechanism for all hierarchy levels.

4.1 MARL Framework

In the context of this work, the task is fully cooperative. Which means that there is a single team objective among all agents, minimize cumulative tardiness. This goal is achieved in a decentralized machine environment, since the shop is modeled as a multi-agent system where each machine contains an agent that decides the next operation locally. Each agent primarily sees its local queue features and few limited information is shared only to "augment observability". Moreover, agents are not required to learn a communication protocol, so every agent has only a partial observation of the complete state. Finally, state transitions depend on each agent's decisions, introducing joint-dynamics and multi-agent non-stationarity. This scenario, a fully cooperative multi-agent task, alongside with all these features, can be described as a Decentralized Partial Observable Markov Decision Process [58].

Considering that each machine acts with partial information while the environment's next state and reward depend on all machines' choices, i. e. depend on the joint dynamics. Further considering that, from the perspective of any single agent, policies of other agents keep changing during learning, making the world appear non-stationary. A CTDE paradigm with shared network parameters is adopted to solve this situation. This is achieved by giving learning algorithms access to richer information at training time while respecting the deployment constraint that decisions must be taken locally and asynchronously with only local observations. This is achieved by leveraging two steps: training and execution. At the training step, a centralized critic evaluates the global shop state and on teammates' observations and actions, while at the execution step, no centralized controller is required since actors use only their local inputs and act asynchronously. Figure 4.1 illustrates how the CTDE paradigm works.

Figure 4.1 depicts CTDE. On Figure 4.1a, during training, all agents interact with the MARL environment and generate trajectories of local state observations $s_{t,i}$, actions $a_{t,i}$ and rewards r_t^i, typically, in a fully cooperative team setting $r_{t,i} = r_{t,j}, \forall i, j$. A policy network with shared parameters $\theta = [\theta_1, \ldots, \theta_N]$ maps each agent's local observation to an action. The Neural Network block represents



(a) CTDE Training Step.

(b) CTDE Execution Step.

Figure 4.1: Centralized Training with Decentralized Execution (CTDE). Training (left): all agents interact with the MARL environment and a shared policy is updated using a centralized critic that has access to global state/action information, mitigating non-stationarity under partial observability. Execution (right): the learned parameters are deployed read-only; each machine/agent selects actions using only its local observation, asynchronously and without inter-agent communication or gradients.

a batched forward pass of $\pi_\theta(\mathbf{s}_t)$ that produces the joint action vector $\mathbf{a}_t$. The environment applies joint dynamics and returns the next observations and reward signals. Learning signals derived from each agent's experience (e.g., policy-gradient terms, advantages, TD errors) flow back as contributions that are aggregated to update the single shared parameter set θ. In Figure 4.1b, at execution time, the learned parameters are respectively transferred to every agent and used read-only. Each agent now acts purely on its local input $s_{t,i}$ to produce an action $a_{t,i}$. There is no centralized controller, no gradient flow, and no requirement for inter agent messaging. The environment collects the joint actions, advances one step, and returns the next local observations, closing the decision loop.

This separation is exactly what CTDE promises, a centralized learning critic that stabilizes learning under partial observability by lowering target variance, leveraging a centralized value function over the global shop state. It also mitigates non-stationarity by conditioning the centralized evaluator on every agent's observations and actions so that targets shift less as policies evolve. Credit assignment is improved because centralized critics can attribute value to joint behavior even with a shared global reward. Finally, it remains deployable in practice since the learned actors at runtime consume only their respective local inputs available on the shop floor.

4.2 Minimal Repetition State

Using neural networks, deep policies and value functions, all of which require inputs and outputs of fixed dimensionality, as the basis of the development of a scheduling strategy raises a contradiction. In a dynamic shop floor, each machine faces a variable number of candidate jobs, therefore, a variable input space size, making the contradiction explicit. A common workaround is to compress the queue into aggregated statistical descriptors (e.g., means or histograms), but such abstractions blur extreme cases, such as very urgent or very long jobs, and break the one-to-one correspondence between input features and selectable jobs. Conversely, restricting experiments to fixed-size instances defeats the purpose of modeling an open, event-driven system with arbitrary arrivals. Under this scenario, a desired state space representation must:

1. Preserve job-specific information for candidates under consideration.
2. Maintain a fixed input/output size for the neural networks.
3. Provide a one-to-one mapping from input slots to executable actions.
4. Avoid unnecessary duplication and keep a consistent slot schema.

In order to make it possible to consider variable queue lengths with fixed-size neural networks, while respecting all four features, a minimal-repetition (MR) state representation was built by [8].

In order to define the amount of jobs needed to be considered when building the state and defining the proper input size, a behavioral simulation of the job shop model was conducted. It considers FIFO as sequencing rule and Table 4.1 shows the ratio between the number of decisions when there are 1, 2, 3, 4, 5 and above jobs in a machine's queue and the total number of decisions. Results in Table 4.1 consider a simulation of the dynamic job shop considering 100 simulation runs, with each run lasting for 2000 time steps, under all three levels of utilization rates (70%, 80%, 90%), all machines using First-In-First-Out sequencing. It can be seen that even at the highest load of 90%, only 6% of decisions happen with 5 and above jobs. Therefore, the MR state representation builds a state space that covers four jobs.

MR state representation leverages four well known and established sequencing rules to select optimal candidate jobs to extract information and build the state space. The rules are SPT, LWKR, MS, WINQ, and each one of them focuses on a different aspect of a job to guarantee that a candidate job has at least one feature that makes it more suitable or of a higher priority than other jobs. Beyond the candidate job information, agents share a small set of data to improve what each one can observe. To support information sharing in the system, three helper

		Number of jobs in queue				
		1	2	3	4	5 and above
Utilization rate	70%	0,64	0,17	0,10	0,05	0,04
	80%	0,61	0,18	0,11	0,05	0,05
	90%	0,57	0,19	0,11	0,07	0,06

Table 4.1: Share of machine-level sequencing decisions by queue length and utilization level. Values are ratios relative to the total number of decisions in each scenario; active decisions occur when the queue has two or more jobs. Aggregated over 100 runs of 2,000 time steps with FIFO dispatching at all machines. Even at 90% utilization, only 6% of decisions occur with five or more jobs waiting, motivating an MR state with four candidate slots.

variables are introduced and to build the state five features of each candidate job are extracted. Table 4.2 depicts each of these variables and features.

Name	**Definition**	**Condition**
SAM_i	availability time of the machine that will handle its next operation	helper & feature
CQ_i	amount of time J_i has already waited in its current queue	helper & feature
NA_k	for machine M_k represents the remaining time until the next job arrives	helper
$t_{i,k}$	job's own processing-time cost, which also represents the extra delay it would impose on all other jobs if selected	feature
WR_i	minimum remaining time to completion, indicating how effectively the current system congestion might be alleviated	feature
S_i	slack time, used to measure how urgent the job is	feature

Table 4.2: Helper variables and per-job features used to build the Minimal-Repetition (MR) state. SAM_i (next-machine availability) and CQ_i (time already waited in the current queue) are both shared as helpers and included as features; NA_k (time until the next job arrives at M_k) is a helper only. For each candidate job, the feature vector $\left[t_{i,k},\, WR_i,\, S_i,\, SAM_i,\, CQ_i\right]$ is used; these rows, plus an arriving-job row, compose the 5×5 MR state tensor consumed by the policy network.

Each variable in Table 4.2 captures a complementary facet of local or downstream

congestion in time units. For a job J_i currently waiting at machine M_k, t_{i,M_k} denotes its processing time on M_k and acts as the immediate service-time cost as well as the opportunity cost imposed on all other jobs if J_i is selected (the basis of SPT). The remaining-work term WR_i aggregates the processing times of all yet-to-be-executed operations for J_i, smaller WR_i indicates that scheduling J_i now is more likely to release it from the system sooner, thereby easing global WIP (the intuition of LWKR). Job urgency is measured by the slack $S_i = (D_i - \text{NOW}) - WR_i$, i.e., time-to-due-date minus remaining work, low or negative S_i flags imminent or inevitable tardiness and motivates prioritization (MS). Two quantities are included, both as features and as lightweight shared "helpers." The first, SAM_i, is the estimated availability time of the successor machine that will process J_i's next operation, smaller SAM_i suggests a receptive downstream station and encourages choices that avoid pushing work into future bottlenecks (WINQ-style look-ahead). The second, CQ_i, is the elapsed time J_i has already spent waiting in the current queue at M_k, providing the policy with recency/fairness context that correlates with tardiness risk under congestion. Finally, NA_k is a helper, not a selectable feature row, giving the remaining time until the next job is expected to arrive to M_k, it informs the arriving-job context row so the agent can anticipate load, trading off immediate service against impending work to reduce idleness and resequencing.

Algorithm 1 is invoked online at every machine-level sequencing epoch both during training rollouts and evaluation. It is triggered whenever M_k completes an operation or a new arrival changes the local feasible set. Given the local queue $\mathbf{J}^k$, the current machine-availability A_{M_k}, the helper NA_k, and a fixed candidate budget K=4, the procedure assembles the observation consumed by the policy at that instant. It initializes an empty state tensor s_t (to be reshaped to 5×5 at the end) and, for bookkeeping, maintains an action mask m and a slot→job map g (useful for logging and credit assignment). The first stage fills the candidate slots with distinct jobs when available: for slot $c \in \{1, \ldots, K\}$, the rule $r = \mathcal{R}[c]$ (SPT, LWKR, MS, WINQ, in order) selects the best job among the yet-unassigned set U, with ties broken by earliest due date and then FIFO arrival. For each selected job, the corresponding feature row $[t_{j,k}, WR_j, S_j, SAM_j, CQ_j]$ is appended to s_t, and the slot is marked actionable if $|\mathbf{J}^k| \geq 2$.

If fewer than K rows were obtained (i.e., $|\mathbf{J}^k| < 4$), a second stage replicates jobs to keep input dimensionality fixed. The algorithm cycles through the rule list $\mathcal{R}$ and, this time, selects from the full queue $\mathbf{J}^k$, allowing the same job to appear in multiple slots when it satisfies several criteria. Finally, a non-actionable arriving-job context row is appended, if some upstream job will next visit M_k, the row $[t_{q,k}, WR_q, S_q, A_{M_k}, NA_k]$ for that job q is inserted. Otherwise a dummy row [0,0,0, A_{M_k},0] is used. The result is a fixed 5×5 tensor in which the top four rows correspond one-to-one with selectable slots (possibly with duplicate entries when the queue is short), and the fifth row provides look-ahead context to discourage

myopic dispatching into imminent congestion.

Having built this state representation, the action space acts over it. Each action targets one of the four candidate slots in the top rows of the 5×5 matrix. Since jobs can be replicated when the queue is short, multiple slots may point to the same job. The agent processes the slot with the highest state–action value, and over time it learns a direct mapping from job-specific features to job selection.

4.3 Reward Shaping Mechanism

Learning signals for event-driven DJSS are fundamentally different from those in short-horizon, episodic tasks. The objective of reducing cumulative tardiness emerges from long chains of interdependent, asynchronous decisions. Feedback is not an immediate win/loss or dense per-step score. Instead, the relevant signal, tardiness, materializes only when a job completes, long after the local decision that set parts of that outcome in motion. This makes temporal credit assignment the central challenge.

Following the slack-based view introduced by [8], each job's initial slack is treated as a finite time budget that is gradually consumed while waiting in queues. If, by completion, the accumulated consumption exceeds the initial budget, the excess equals the observed tardiness. Shaping the reward around this overspend aligns the agent's incentives with the objective. Decisions that inflate queueing delays for urgent jobs receive stronger penalties, while inessential waiting under low criticality is down weighted. To make these penalties causally meaningful, the learning pipeline must also align the timing of reward assignment with when consequences are revealed.

Unlike standard MARL benchmarks where agents advance in synchronized rounds, each agent acts at step t, receives immediate feedback at $t{+}1$, and stores a complete transition tuple. Scheduling on a shop floor unfolds in a fundamentally different rhythm. Here, every decision is tied to a particular job's trajectory, and the true impact of that decision becomes visible only when the job exits the system. This event-driven coupling between decisions and their eventual outcomes can be described as a chronological joint action perspective.

This view has implications for both the state-transition bookkeeping and the data that enter the replay buffer. During rollouts, the system must retain incomplete transitions, states and actions without rewards, keyed to the corresponding job until completion reveals the appropriate feedback. Practically, this calls for a parallel, job-indexed pipeline that buffers partial records and later finalizes them by backfilling the shaped penalties. Figures 4.2 and 4.3 illustrate the contrast: panels 4.2a and 4.3a depict synchronous MARL with immediate rewards, whereas panels 4.2b and 4.3b show the asynchronous, chronological process with deferred

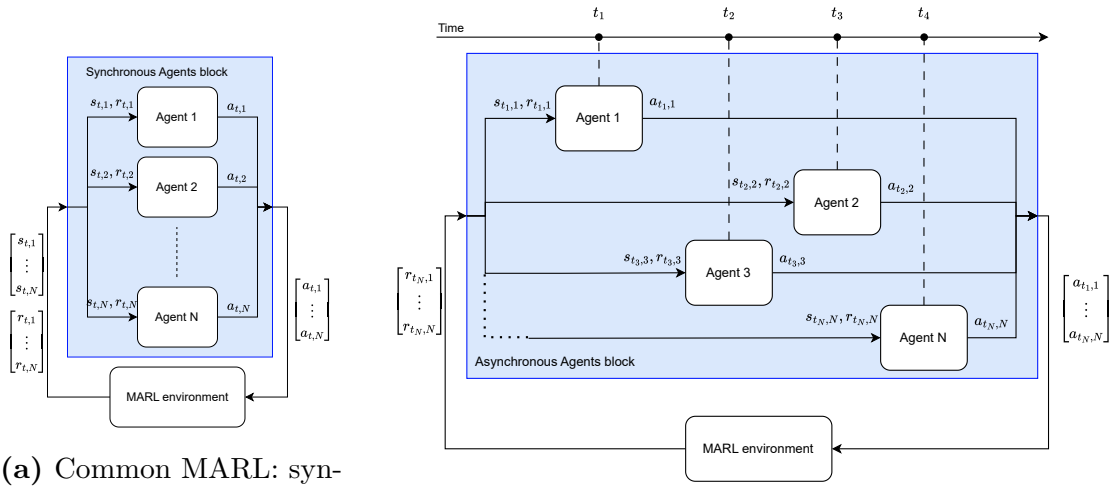
Algorithm 1 Construct Minimal-Repetition (MR) state for machine M_k

Require: Local queue $\mathbf{J}^k$; machine availability A_{M_k}; helper NA_k; candidate budget K=4; rule list $\mathcal{R} = [\text{SPT}, \text{LWKR}, \text{MS}, \text{WINQ}]$

Ensure: State tensor $s_t \in \mathbb{R}^{5\times5}$; action mask $m \in \{0,1\}^4$; slot→job map $g \in (\mathbf{J}^k \cup \{\perp\})^4$

$s_t \leftarrow [\,]$ ▷ rows will be appended; final shape is 5×5
$m \leftarrow [0,0,0,0]$ ▷ mask
$g \leftarrow [\perp, \perp, \perp, \perp]$ ▷ slot→job mapping
$U \leftarrow \mathbf{J}^k$ ▷ U is the set of unassigned jobs
$c \leftarrow 1$ ▷ c indexes candidate slots
Stage 1: fill with distinct jobs when available (no replication).
while $U \neq \emptyset$ **and** $c \leq K$ **do**
  $r \leftarrow \mathcal{R}[c]$ ▷ use rule aligned with the slot index
  $j^\star \leftarrow \arg\min_{j\in U} Key(j, r)$ ▷ ties: earliest due date, then FIFO arrival
  Append $\left[t_{j^\star,k},\ WR_{j^\star},\ S_{j^\star},\ SAM_{j^\star},\ CQ_{j^\star}\right]$ to s_t
  $g[c] \leftarrow j^\star$
  $m[c] \leftarrow \mathbf{1}[\,|\mathbf{J}^k| \geq 2\,]$ ▷ only actionable if at least two jobs exist
  $U \leftarrow U \setminus \{j^\star\}$
  $c \leftarrow c{+}1$
end while
Stage 2: if fewer than K rows, replicate by cycling rules.
while $c \leq K$ **do**
  $r \leftarrow \mathcal{R}[\,((c{-}1) \bmod 4) + 1\,]$ ▷ cycle SPT→LWKR→MS→WINQ
  $j^\star \leftarrow \arg\min_{j\in\mathbf{J}^k} Key(j, r)$ ▷ replication allowed
  Append $\left[t_{j^\star,k},\ WR_{j^\star},\ S_{j^\star},\ SAM_{j^\star},\ CQ_{j^\star}\right]$ to s_t
  $g[c] \leftarrow j^\star$
  $m[c] \leftarrow \mathbf{1}[\,|\mathbf{J}^k| \geq 2\,]$
  $c \leftarrow c{+}1$
end while
Arriving-job context (non-actionable fifth row).
if HasUpstreamJobNextTo(M_k) **then**
  Let q be that job
  Append $\left[t_{q,k},\ WR_q,\ S_q,\ A_{M_k},\ NA_k\right]$ to s_t
else
  Append $\left[0,0,0,\ A_{M_k},0\right]$ to s_t
end if
return s_t

attribution used in scheduling.



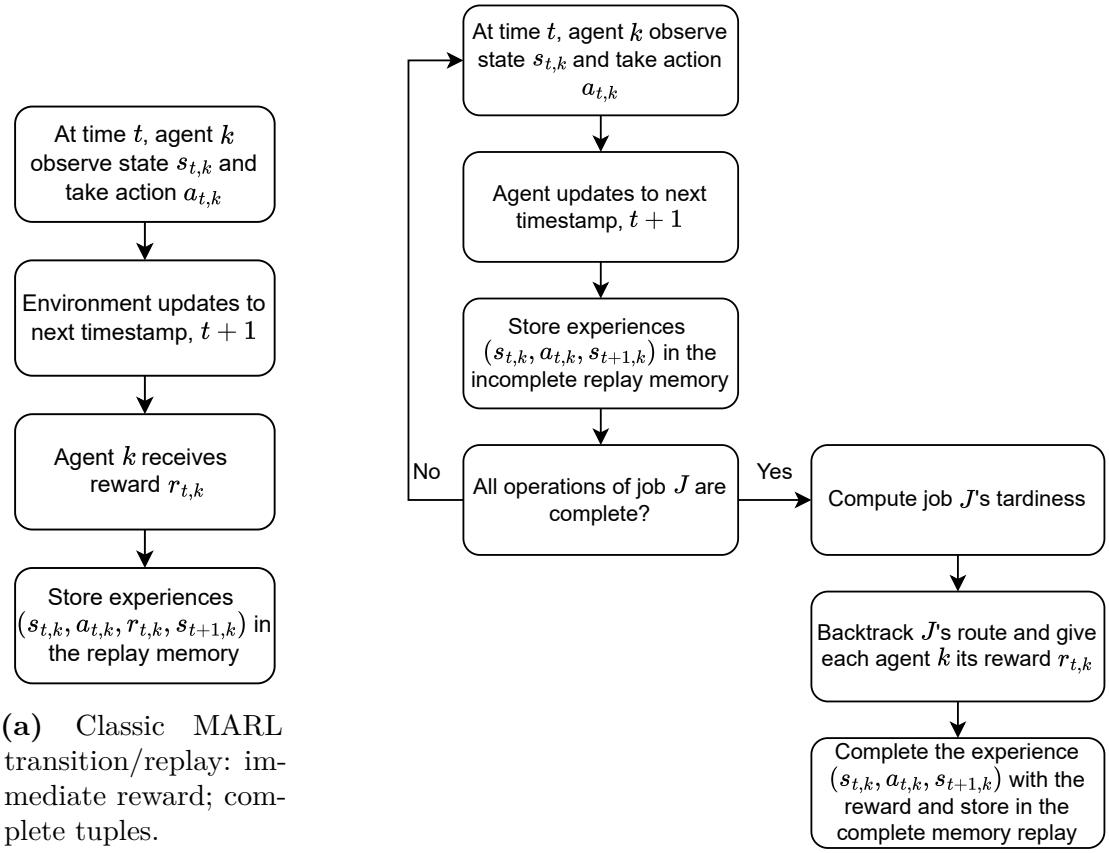
(a) Common MARL: synchronous block updates.

(b) Scheduling MARL: asynchronous, chronological joint action.

Figure 4.2: Agent–environment interaction patterns. (a) Common MARL with synchronous joint steps. (b) Scheduling with asynchronous events; the effective "joint action" unfolds chronologically as machines become eligible.

In Figures 4.2 and 4.3, the contrast between the MARL and the dynamics of a job shop is illustrated, and it shows why a different replay pipeline is required. In conventional MARL, Figure 4.2a, all agents act synchronously at step t, yielding a joint action $A_t = \left[a_{t,1} \dots a_{t,N}\right]$ and an immediate joint reward $R_t = \left[r_{t,1} \dots r_{t,N}\right]$. In scheduling problems, Figure 4.2b, machines (agents) make decisions asynchronously as their queues become eligible, thus the joint decision is realized as a chronological sequence $a_{t_1 \sim t_m}$, and the true outcome that occurs along a job's route and is only revealed upon completion. Consequently, the classic construction, Figure 4.3a, stores complete transitions $(s_{t,k}, a_{t,k}, s_{t+1,k}, r_{t,k})$ with immediate rewards. The proposed scheme, Figure 4.3b, first buffers incomplete transitions $(s_{t,k}, a_{t,k}, s_{t+1,k})$ without reward, then, when a job finishes, backtracks along its operation sequence to compute each agent's reward contribution, fills in $r_{t,k}$, and finally commits complete experiences to replay. This chronological joint action, alongside with a delayed reward attribution, better matches the causal structure of the shop floor and mitigates the credit assignment issue.

To respect the chronological-joint-action view, a queue-time–based shaping rule is adopted. This shaping issues rewards only at job completion, when the full causal chain is known. At that moment, the job's production record is traversed backward and the overall outcome is redistributed across the machines that handled it in proportion to the queueing delays their decisions induced.



(a) Classic MARL transition/replay: immediate reward; complete tuples.

(b) Scheduling replay: buffer incomplete tuples; backtrack at job completion to assign shaped rewards.

Figure 4.3: Transition and replay pipelines. (a) Classic immediate-reward tuples. (b) Buffering/backtracking to align rewards with the job that reveals the consequence.

The shaping respects five design commitments. First, urgency modulates penalties: the slack observed on arrival is smoothly mapped to a criticality weight $\beta \in (0,2)$ via a saturating (sigmoid-like) transform, with a sensitivity parameter δ tuning how sharply β grows as slack shrinks, so that detaining low-slack jobs is penalized more strongly. Second, to avoid punishing delays outside the agent's control, queue-time accounting begins only when a job becomes selectable at a machine. Only waiting attributable to the actionable decision is charged. Third, since a local sequencing choice also propagates downstream, a fraction of the next station's waiting is "shifted back" to the current decision. A coefficient $\alpha = 0.2$ is used to internalize part of the congestion the decision creates at the successor and thereby discourage pushing work into already loaded queues. Fourth, only

late completions are penalized: every station that processed a tardy job receives a non-positive contribution, whereas early or on-time jobs yield zero reward, avoiding the common pitfall of incentivizing gratuitous earliness that can reduce tardy incidence while worsening aggregate tardiness. Finally, for numerical stability and emphasis on harmful tails, the penalty scales quadratically with the reconstructed waiting time and is normalized by a factor ϕ so that most shaped rewards fall in $[-1,0]$, providing well-behaved gradients while preserving the intended preference ordering.

The complete procedure is triggered only when a job J_i finishes, because that is the first moment its true consequence, tardiness, is known. If the tardiness is equal to zero, no penalties are issued and a zero vector of length equal to the job's number of operations O_i is returned. If the job is late, the algorithm walks through each operation $j = 1, \ldots, O_i$ and reconstructs the portion of delay attributable to the sequencing decision taken at that machine by operating over each operation's queue time Q_i, j. The reconstructed waiting time $RQ_{i,j} = (1-\alpha)\, Q_{i,j} + \alpha\, Q_{i,j+1}$ transforms the queue. Each decision is penalized with its own queue time and a small shift-back fraction α of the next station's queue time, attributing part of downstream congestion to the upstream decision that set that arrival time. This redistribution internalizes part of the downstream congestion created by choosing J_i now rather than an alternative, discouraging short term pushes into already-loaded buffers.

Each reconstructed wait is then modulated by the job's urgency at the time it arrived to that station. The mapping $\beta = 1 - \left(S_i^j / (|S_i^j| + \delta)\right)$ smoothly compresses a wide slack range into the interval $(0,2)$. Large positive slack yields $\beta \approx 0$ concurring in mild or no penalty for delaying a comfortable job. Slack near zero sits around $\beta \approx 1$ providing neutral scaling, and negative slack drives β toward 2 driving a strong penalty for detaining already endangered work. The shaped contribution is a negative quadratic in the scaled wait, $R_{i,j} = -\left(\beta\, RQ_{i,j}/\phi\right)^2$, which emphasizes harmful tails and de-emphasizes small, routine waits. Finally, values are clipped to $[-1,0]$ for numerical stability. The net effect is a sparse, delayed signal that penalizes only late jobs, attributes blame to the specific machines/agents that caused the waiting, weight the penalty factor by how critical the job was at the time, and lightly "looks ahead" via α so local choices account for the congestion they induce downstream.

4.4 Hierarchical Learning

HL decomposes the problem it is trying to solve into multiple levels of abstraction. A common two-level scheme consists of a High-Level agent (HLA) acting in an abstract space, and one or more Low-Level Agents (LLA) at every decision point,

conditioned on the current high-level agent directive. Formally, when the HLA chooses an macro-action o at state s_t that persists for τ steps. A macro or extended action is defined by an intra-option policy $\pi_o(a \mid s)$ and a termination rule $\beta_o(s) \in [0,1]$. The process is a semi-Markovian Decision Process (SMDP) [43, 59] and the corresponding backup is

$$Q(s_t, o) = \mathbb{E}\left[\sum_{k=0}^{\tau-1} \gamma^k r_{t+k} + \gamma^\tau \max_{o'} Q(s_{t+\tau}, o')\right], \tag{4.1}$$

where $Q(s_t, o)$ represents the option value function, i. e. the expected discounted return when starting in s_t and choosing the macro-action o, and following the specified high-level policy. s_t represents the state at high-level decision at time t. The number of low-level steps the option o executes before terminating is represented by τ, which is governed by the environment dynamics and β_o. The the reward at low-level step $t + k$ is denoted by r_{t+k}, it is the resulting reward k steps after the choosing o, while γ represents the discount factor. Finally, $s_{t+\tau}$ represents the state when option o ends and the next high-level choice is made. The SMPD backup given by (4.1) compresses long range consequences into fewer decisions [43].

HL operationalizes temporal abstraction by letting the HLA pick options that commit LLAs to consistent local behavior until a termination condition is met. In the options framework, each option o is defined by an initiation set $\mathcal{I}_o \subseteq \mathcal{S}$, an intra-option policy $\pi_o(a \mid s)$, and a stochastic termination rule $\beta_o(s) \in [0,1]$ [43]. When the HLA samples $o \sim \mu(\cdot \mid s_t)$ with $s_t \in \mathcal{I}_o$, the environment evolves for τ low-level steps under π_o, and the high level updates only at option boundaries via the SMDP backup in (4.1). This abstract time scale structure enables to compress long range consequences into fewer high-level decisions while preserving reactivity at the low level.

Beyond temporal abstraction, HL enables structural abstraction through state and action factoring. The HLA reasons over compact, slowly varying context, such as system congestion, due-date tightness, bottleneck load, while LLAs observe rich, local features needed to make feasible and responsive choices. Learning can proceed off-policy with SMDP Q-learning at the high level and standard actor–critic or value-based updates at the low level. End-to-end variants such as Option-Critic differentiate through π_o and β_o to discover options automatically [44], while goal-conditioned designs, such as [45, 46, 60] pass subgoals or embeddings that LLAs optimize with shaped or intrinsic rewards [46, 45, 60].

For a scheduling context, the abstraction is natural: the HLA selects a mode, for instance expedite tardy jobs, balance WIP across bottlenecks, minimize setups, and LLAs at individual machines convert that mode into concrete dispatches subject to local constraints. Termination can be event driven, for instance, when lateness backlog drops below a threshold, or time-based, reevaluating every H minutes. This separation yields low decision latency, once LLAs evaluate simple mode conditioned

scores. Additionally, HL promotes coherent cross machine behavior, since all LLAs are conditioned on the same intent. Finally, it also provides robustness, due to the HLA being able to switch modes as context drifts.

Approach	Decision Latency	Drift Adaptivity	System-level Coordination	Labels Needed	Horizon Handling	Re-solve Frequency
Dispatching Rules	Very low	Low	Limited	None	Short-horizon	None
Metaheuristics	High	Medium	High	None	Long-horizon	Frequent
Supervised Learning	Low	Medium	Limited	Required	Short-horizon	Retrain on shift
Flat RL MARL	Medium–High	Medium	Medium	None	Long-horizon	Continuous training
Hierarchical RL (HL)	Low	High	High	None	Long-horizon	Continuous training

Table 4.3: Qualitative comparison of shop-floor control approaches. Columns summarize decision latency, adaptation to distributional shift (drift adaptivity), system-level coordination, need for labels, ability to handle long horizons, and how often re-optimization/re-training is required. Higher is better for "Drift Adaptivity" and "System-level Coordination", while lower is better for "Decision Latency" and "Re-solve Frequency".

Under a scheduling context Table 4.3 summarizes how HL compares to other approaches. Dispatching rules are unmatched for speed but provide weak cross-machine coupling [47, 9, 48]. Metaheuristics, on the other hand, can coordinate globally but incur re-optimization delays that are ill-suited to highly dynamic environments. Supervised policies replicate historical "good" choices yet struggle when problem size, routing, or product mix evolves and do not address delayed consequences. Flat RL/MARL improves horizon reasoning but pays a heavy exploration and credit assignment tax at primitive time scales [34, 51, 35].

HL occupies a sweet spot, it amortizes long horizon reasoning and communicates intents, while LLAs keep per-decision latency comparable to rules. Since feedback at the high level aggregates across many low-level steps, exploration and credit assignment are easier, and shaped or intrinsic rewards align local actions with system objectives [29]. These properties make HL a particularly good fit for dynamic job shop control, where fast, local feasibility must coexist with global, long-horizon performance.

The DJSSP exhibits three properties that make it natural to target with HL. First, it contains asynchronous decisions over a long horizon coupling. This means that machines act at different times, yet tardiness depends on a chain of decisions

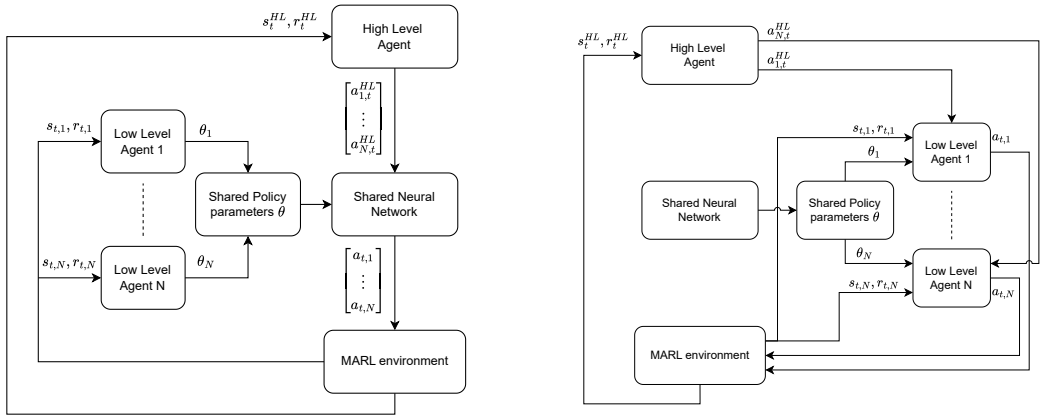
across machines. A HLA that sets intents promotes consistency across these asynchronous local choices. Second, feedback is sparse and delayed, tardiness materializes only at completion. Nevertheless, a LLA can receive intrinsic or shaped rewards aligned with the current mode, while the HLA optimizes longer horizon metrics [29]. Third, there is context drift and heterogeneity. Utilization rate and product mix change over time, so skills specialized for "high congestion" vs. "low congestion" or "tight" vs. "loose" due dates can be reused and recombined, improving sample efficiency and robustness [50, 60, 45].

4.5 Hierarchical Framework

A HL framework that considers two levels of abstraction was designed to develop sequencing strategies. It was built considering different levels of abstraction ideas in hierarchical RL [43, 57, 50, 44, 46, 45, 60], and leveraging the best performing Heuristics (PTWINQS and CRSPT). A HLA is responsible for analyzing the current state of the job shop modeled as in Section 3 and set the mode of operation of the LLAs. Then, each LLA is responsible for making sequencing decisions under the assigned mode of operation. When assigning an operation mode to a LLA, the HLA chooses one among three possible options: (i) PTWINQS, (ii) CRSPT or (iii) MARL. In this framework, there is only one HLA for the entire shop floor, while there is one LLA per machine.

Agents in all hierarchy levels contain neural networks that are trained leveraging RL techniques. After a brief warm-up period that fills the replay buffers with exploratory experience, the HLA and the LLAs learn simultaneously and influence each other's behavior. During the warm-up, both levels of hierarchy explore the environment gathering experience of early environment interaction data. Later, this experience is randomly sampled, which is done to reduce the bias introduced by considering sequential experiences. Concretely, the HLA observes a compact representation of the job shop state and at each decision point, it selects a mode of operation for the LLAs among PTWINQS, CRSPT and MARL. Which is done via an ϵ-greedy policy and stores the transition $(s_t^{HL}, a_t^{HL}, s_{t+1}^{HL}, r_t^{HL})$ in a high level replay buffer. At the low level, each machine hosts a LLA that makes sequencing choices under the mode prescribed by the HLA. LLAs follow the CTDE scheme with parameter sharing as defined in Section 4. All machines write their experiences to a single replay memory and are served by one shared value network whose outputs score up to four candidate jobs extracted from the local queue, while actions are chosen ϵ-greedily. Figure 4.4 illustrates how the CTDE paradigm is extended to accommodate both hierarchy levels.

Figure 4.4 represents the extended CTDE scheme to two hierarchy levels for DJSS. In the Training Step, Figure 4.4a, the HLA observes an abstract shop



(a) CTDE HL extended Training Step.

(b) CTDE HL extended Execution Step.

Figure 4.4: CTDE extended to a two-level hierarchy for DJSS. Left (training): the High-Level Agent (HLA) observes an abstract shop state s_t^{HL}, selects a mode vector $[a_{1,t}^{\mathrm{HL}}, \ldots, a_{N,t}^{\mathrm{HL}}]$, and conditions the shared Low-Level network used by per-machine LLAs, each LLA uses a local state $s_{t,i}$ and shaped reward $r_{t,i}$. Right (execution): learning is disabled and the same policies act in inference. Shared parameters θ enable fast, decentralized dispatching coordinated by a common global intent.

state s_t^{HL} and its aggregated reward r_t^{HL}, selects a mode vector $[a_{1,t}^{HL}, \ldots, a_{N,t}^{HL}]$ and broadcasts it to condition the shared low-level network. Each LLA, one per machine, supplies its local state $s_{t,i}$, receives shaped reward $r_{t,i}$, and—via the shared value network with parameters θ chooses a dispatching action $a_{t,i}$. The MARL environment executes the actions and returns next states and rewards. In the Execution Step, Figure 4.4b, learning is disabled and the same policies are used in inference. The HLA continues to issue the mode leveragin $[a_{1,t}^{HL}, \ldots, a_{N,t}^{HL}]$, LLAs act with the frozen shared parameters θ, and the environment advances. This architecture yields fast, decentralized dispatching conditioned on a global intent, while learning is centralized and sample efficient via shared representations.

4.6 Hierarchical State Representations

At each decision point t, the HLA observes a compact job shop state representation and selects a sequencing mode for the LLAs from one of the possible options ((i) PTWINQS, (ii) CRSPT, (iii) MARL). The chosen mode can be applied either globally to all machines or individually to each, depending on how the HLA is set to handle the DJSSP. In this way, there are two possible representations for

the high level state. It is worth mentioning that once training starts, it is not possible to change between setting the HLA to apply the operation mode globally or individually. When set to apply the operation mode globally, the state vector g_t contains only descriptive information that summarizes capacity usage, workload balance, due-date pressure, downstream congestion, and realized lateness on the current state of the job shop floor. However, when set to apply the operation mode individually to each machine, g_t is appended by the machine's i vector of local descriptive information $d_{t,i}$ to produce the state vector $s_{t,i} = \begin{bmatrix} g_t & d_{t,i} \end{bmatrix}$. The latter situation compensates increased state space size by adding extra information, which is most often meaningful to solve the problem.

The compact g_t aggregates shop wide signals. Using only g_t yields a single mode applied consistently across all machines, which enforces coherent behavior. In homogeneous or lightly unbalanced regimes, a coherent global rule is often near optimal [47, 9, 12, 48] and trains faster [57, 50]. The augmented $s_{t,i} = \begin{bmatrix} g_t & d_{t,i} \end{bmatrix}$ adds local summaries, allowing the HL policy to tailor the mode to each machine while still considering the global context. This reduces state aliasing at bottlenecks and improves responsiveness under heterogeneous routing and variable processing times.

Let $\mathcal{M}$ be the set of machines. The HL global vector is

$$g_t = \left[\bar{u}_t, \quad \mu_t^{(q)}, \quad \sigma_t^{2(q)}, \quad \mu_t^{(s)}, \quad \rho_t^{(\text{urgent})}, \quad \mu_t^{(\text{winq})}, \quad \sigma_t^{2(\text{winq})}, \quad r_t^{(\text{tardy})}\right], \tag{4.2}$$

where $\bar{u}_t$ represents the the average utilization up to time t, $\mu^{(q)}$ represents the average queue length at time t, $\sigma^{2(q)}$ represents the system wide queue variance at time t, $\mu_t^{(s)}$ represents the mean slack time of jobs in queue at time t, $\rho_t^{(\text{urgent})}$ represents the ratio of urgent jobs in queue at time t, $\mu_t^{(\text{winq})}$ and $\sigma_t^{2(\text{winq})}$ represents the system wide average work in queue and variance at time t, respectively, finally $r_t^{(\text{tardy})}$ represents the tardy rate at time t. The reasons for considering these variables to build the high level state are as follows:

- $\bar{u}_t$ (mean utilization) is a measurement of the shop's capacity usage. At high $\bar{u}_t$, due-date pressure typically rises and options that prioritize urgency become preferable, while at low $\bar{u}_t$, efficiency-oriented rules are often adequate.
- $\mu_t^{(q)}$ and $\sigma_t^{2(q)}$ (mean and variance of queue lengths) separate global load from imbalance. A high variance is a flag for emerging bottlenecks when a balancing option tends to reduce downstream blocking, whereas a uniformly long queue landscape calls for system wide expediting.
- $\mu_t^{(s)}$ (mean slack) and $\rho_t^{(\text{urgent})}$ (ratio urgent jobs) quantify how tight the due-date landscape is. When these indicate strong pressure, CR style or slack aware modes are favored, when pressure is mild, processing time oriented choices are less risky.

- $\mu_t^{(\text{winq})}$ and $\sigma_t^{2(\text{winq})}$ summarize expected work in the next queues across the shop, i.e., a one-step look-ahead of congestion propagation. These signals directly align with PTWINQS reasoning, processing time efficiency tempered by predicted downstream load, allowing the HLA to switch into a "congestion-aware" mode before blocking materializes.

- $r_t^{(\text{tardy})}$ (tardy rate) closes the loop between high level decisions and realized performance, giving the SMDP critic a dense, aggregated proxy of long-horizon cost while low level shaped rewards handle credit assignment within options.

Using only g_t yields a single mode broadcast to all machines, enforcing coherent shop-wide behavior, useful in homogeneous regimes and faster to learn. When heterogeneity is pronounced, augmenting with $d_{t,i}$ to form $s_t^i = [g_t, d_{t,i}]$ reduces state aliasing for the HLA by exposing local summaries at option selection time, enabling machine specific modes without losing global context.

When selecting modes individually per machine, the global vector g_t is augmented with machine local statistics. Machine i produces the sate vector

$$s_{t,i} = \begin{bmatrix} g_t & d_{t,i} \end{bmatrix}, \tag{4.3}$$

$$d_{t,i} = \begin{bmatrix} |q_t^{(i)}|, & \mu_t^{(q,i)}, & \sigma_t^{2(q,i)}, & \mu_t^{(s,i)}, & \rho_t^{(\text{urgent},i)}, & \mu_t^{(\text{winq},i)}, & \sigma_t^{2(\text{winq},i)} \end{bmatrix}, \tag{4.4}$$

where $|q_t^{(i)}|$ is the local queue length, $\mu_t^{(s,i)}$ and $\rho_t^{(\text{urgent},i)}$ are the local slack mean and urgent-job ratio, and $\mu_t^{(\text{winq},i)}, \sigma_t^{2(\text{winq},i)}$ summarize downstream workload from i. This feature selection was motivated by the same reason as in the construction of g_t, however, from a local perspective. The HL input is then either $s_t = g_t$ (global mode) or $s_{t,i} = [g_t,\, d_{t,i}]$ (per-machine). Finally, a summary of the high level state is represented in Table 4.4.

When considering either g_t or each $s_{t,i}$, a HLA action means to select a mode of operation. Which results in the high level action space $\mathcal{A}$ having cardinality equal to 3, containing each possible decision mode and being represented by

$$\mathcal{A} = \begin{Bmatrix} \text{PTWINQS} & \text{CRSPT} & \text{DDQN} \end{Bmatrix}. \tag{4.5}$$

Finally, the low level state representation leverages the same MR construction described in Section 4. Using this strategy, the low-level state s_t stacks four queued jobs plus a fifth "arriving-job" option as rows, each described by five features. Constructing a 5×5 local queue snapshot in the typical case. When multiple jobs wait, the four queued rows are populated by selecting distinct candidates via column wise minimization over the state features to diversify options. If fewer than four exist, the tensor is padded as done in Section 4.2. The fifth row represents the best imminent arrival on this machine, enabling the policy to consider intentional

Symbol	Scope	Description
$\lvert q_t^{(i)} \rvert$	Local	Queue length at i
$\mu_t^{(q,i)}$		Mean local work content
$\sigma_t^{2(q,i)}$		Variance of local work
$\mu_t^{(s,i)}$		Mean slack at i
$\rho_t^{(\text{urgent},i)}$		Urgent job ratio
$\mu_t^{(\text{winq},i)}$		Avg. downstream load
$\sigma_t^{2(\text{winq},i)}$		Var. of downstream load
$\bar{u}_t$	Global	Mean utilization
$\mu_t^{(q)},\ \sigma_t^{2(q)}$		Load & imbalance
$\mu_t^{(s)}$		System slack
$\rho_t^{(\text{urgent})}$		System urgency
$\mu_t^{(\text{winq})},\ \sigma_t^{2(\text{winq})}$		Downstream congestion
$r_t^{(\text{tardy})}$		Tardy jobs ratio

Table 4.4: Local and global features used in the high-level state. Local terms (indexed by machine i) describe the queue and downstream workload seen by a single machine, while global terms summarize shop-wide congestion, urgency, and performance. The HLA consumes either the global vector g_t (shared mode) or the concatenated vector $s_t^{(i)}$ (per-machine mode).

idleness. An action then maps a selected row to a concrete queue position. It is worth mentioning that classic dispatching rules operate only on the queued jobs of s_t, so the arriving-job row influences only the learned policy. Alongside with the state space, the action space also stays the same as in Section 4.2.

4.7 Hierarchical Reward Design

In a hierarchical scheduler, the high level and low level agents solve different control problems on different clocks with different levers and visibility, so they require different rewards to learn efficiently and avoid interference. The HLA acts by selecting a system mode that shapes macro dynamics. Its feedback must therefore reflect global outcomes aggregated over a window, such as tardiness, lateness mix, WIP stability, or a summary of recent LLA outcomes, so that the value of a mode is identifiable. The LLA acts at every dispatch, directly choosing which job a machine processes next, if it were trained only on sparse global objectives realized at completion, without proper treatment, credit assignment would traverse long chains of decisions with high variance and stall learning. Instead, LLA needs dense, local

shaping tied to immediate consequences based terms, which reduce variance while preserving the global optimum. Using a single shared reward either floods HL with short-horizon noise (if made dense for LLA) or starves LLA of signal (if kept sparse for HLA), and it blurs responsibilities so that modes become indistinguishable or oscillatory. Distinct rewards thus decouple roles: HLA optimizes strategic mode selection against system level metrics, LLA optimizes fine-grain dispatch under that mode. Therefore, for the HLA it was considered two possible reward construction mechanisms, while the LLA counts with a single option to obtain its reward.

In order to minimize the total tardiness observed by the system a naive reward mechanism was built. This first approach allows the HLA to select, at every low level decision, the mode by which machines should sequence. At each high level decision point the naive reward is constructed by aggregating the tardiness produced system wide. Formally, it can be obtained as

$$r_t^{\text{Tard}} \;=\; -\sum_{i=0}^{N} \text{Tard}_\text{i}, \tag{4.6}$$

where Tard_i is the realized tardiness of each job i and r_t^{Tard} is HLA reward at time t. The leading minus sign converts tardiness, a cost we wish to minimize, into a reward to be maximized by the RL algorithm. This naive approach directly optimizes the thesis objective by minimizing system wide tardiness.

A second approach to build the HLA reward was still considered. The second approach tries to leverage the chronological joint reward mechanism described in Section 4. Instead of attempting to directly increase system wide performance by minimizing tardiness as the naive one, it chooses an indirect path. By considering the accumulated latest shaped reward in each machine, this indirect approach can be defined as

$$r_t^{\text{HL}} \;=\; \sum_{k=1}^{K} R_{k,\text{last}}^{\text{LL}}, \tag{4.7}$$

where, r_t^{HL} is the HLA reward and $R_{k,\text{last}}^{\text{LL}}$ represents the most recent reward obtained by machine k, last specifies the recent most reward obtained. In this way, r_t^{HL} does not directly minimize system wide tardiness. However, by passing the most recent low level signals up to the high level, it makes the high level's target sensitive to the machines' current congestion and urgency. Therefore, it optimizes for congestion and urgency, indirectly affecting overall tardiness.

When considering the low level agents, the reward shaping mechanism is preserved the same as the one discussed in Section 4. Rewards are assigned only after a job J_i completes. Its route is backtracked and each of its operations receives a penalty proportional to a reconstructed queue time that internalizes downstream congestion,

$$RQ_{i,j} \;=\; (1-\alpha)\, Q_{i,j} \;+\; \alpha\, Q_{i,j+1}, \qquad \alpha \in [0,1], \tag{4.8}$$

where $RQ_{i,j}$ attributes waiting to the decision taken for operation j of job i. The shift-back coefficient α internalizes a small fraction of downstream delay created by the current choice. $Q_{i,j}$ is the queueing time observed before operation j and $Q_{i,j+1}$ is the queueing time observed before the next operation on the job's route.

Urgency $\beta_{i,j}$ is captured by the job's slack $S_{i,j}$ upon arrival, mapped to a criticality factor

$$\beta_{i,j} \;=\; 1 \;-\; \frac{S_{i,j}}{|S_{i,j}| + \delta}, \qquad \delta > 0, \tag{4.9}$$

where $S_{i,j}$ is the slack observed when job i arrives to the station executing operation j (time-to-due-date minus remaining work). The mapping compresses a wide slack range into $\beta_{i,j} \in (0,2)$. Lower slack values implies in larger $\beta_{i,j}$, more urgent jobs, while comfortable slack produces smaller $\beta_{i,j}$. The parameter δ controls sensitivity.

The per operation reward is then

$$r_{i,j}^{\mathrm{LL}} \;=\; -\left(\frac{\beta_{i,j}\, RQ_{i,j}}{\phi}\right)^2, \tag{4.10}$$

where $r_{i,j}^{\mathrm{LL}}$ is the shaped reward assigned to the agent that sequenced operation j of job i, and $\phi > 0$ scales magnitudes so typical values lie in $[-1,0]$. Values are clipped to $[-1,0]$ for numerical stability. The negative quadratic emphasizes harmful long waits while de-emphasizing small, routine delays.

Queue time counting begins only when the job becomes selectable on a machine, and on time or early completions are not positively rewarded to avoid incentivizing earliness. This shaping yields dense, low-variance feedback that drives agents to relieve congestion and protect urgent jobs, while remaining aligned with the global objective of reducing tardiness.

Chapter 5

Simulation Results

5.1 Experiment Specifications

Each machine is associated with a RL agent, and all agents share the same neural network architecture and hyperparameters under a parameter-sharing scheme. The input features primarily consist of raw time-based quantities, such as operation processing times $t_{i,k}$ and job slack S_i. Since these features naturally extend beyond the effective input range of common activation functions, they can aggravate vanishing-gradient problems during training. Additionally, the state vector itself lacks temporal ordering or graph connectivity, being simply a collection of heterogeneous time measurements. To address these issues, the input features are first normalized using Layer Normalization, after which the agents learn over the stabilized representation through a multi-layer perceptron (MLP).

The MLP adopts a lightweight yet expressive design. It consists of six hidden layers with progressively decreasing widths, structured as $64 \times 48 \times 48 \times 36 \times 24 \times 12$. Nonlinear transformations are performed using the hyperbolic tangent activation function, and the network is optimized with stochastic gradient descent enhanced by a momentum term of 0.9. The optimization objective is defined through the Huber loss, chosen for its robustness to outliers and stability across varying error magnitudes. Training begins with a learning rate of 5×10^{-3}, which gradually decays to 10^{-3} to encourage convergence toward more stable minima.

From the reinforcement learning perspective, the agents employ a discount factor of $\gamma = 0.95$, balancing immediate and long-term scheduling outcomes. Exploration is guided by an ϵ-greedy policy, with the exploration rate decaying from 40% to 10% as training progresses. Experience replay is used to stabilize learning: the replay buffer holds up to 1,024 transitions, from which minibatches of size 64 are sampled to perform updates. The network layout is depicted in Figure 5.1.

A training run simulates 100,000 units of time, while a validation run simulates

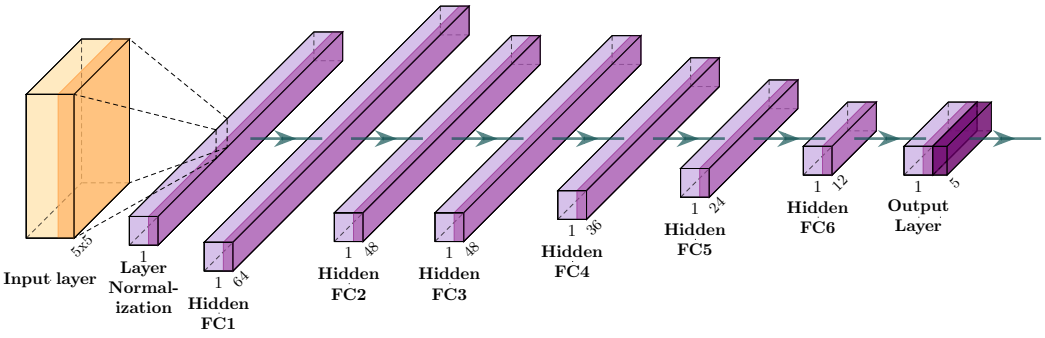


Figure 5.1: Policy network used by all agents (parameter sharing). The input is the 5×5 Minimal-Repetition (MR) state flattened and layer-normalized, followed by an MLP with hidden sizes 64−48−48−36−24−12 (tanh), producing 5 outputs that score the four candidate queue slots plus the arriving-job slot. A higher Q-value represents a higher selection preference.

2,000 units of time. Under the three utilization levels, the shop floor receives 2,800, 3,200, and 3,600 job arrivals, respectively, during training and 56, 64, 72 jobs arrivals during validation. In order to better understand the behavior and responsiveness of the proposed model it was submitted to 30 different training and validation runs. The one that provided the best validation results was selected.

Running a large number of independent training and validation runs is needed because both the DJSSP environment and the learning algorithm are inherently stochastic, which makes single run results unreliable. In the formulated job shop instance, inter arrival times are sampled from an exponential distribution, processing times from a uniform distribution, and due-date tightness from a uniform range. Which when combined with three utilization rate scenarios produce different realizations instances across runs, and thus different learning signals for each agent. Beyond environment noise, RL itself adds variability through random initialization and exploration, yielding performance distributions that are often skewed, consequently, many seeds are needed to estimate means and variability faithfully rather than over or under estimating performance from a “lucky” seed. Empirical RL guidance further shows that confidence intervals become reliable with roughly 30 runs, achieving a practical balance between statistical stability and compute cost [61].

Figure 5.3 reports the evolution of the training loss across all training steps for three utilization rates, when considering the MARL approach. In each panel,

the highlighted curve is the moving-average mean over 30 independent runs (window=50), and the shaded area represents a 95% confidence interval (CI). At 70% utilization the loss quickly drops and stabilizes at a low level with a narrow CI, indicating fast, stable learning in a relatively easier, lighter loaded shop. At 80% utilization the loss exhibits a pronounced transient rising to a peak around 5000 steps before decaying, while the CI widens during this phase, reflecting variability as the agent adapts to a busier, more non-stationary queueing process. At 90% utilization the loss stays higher and oscillatory throughout, and the CI remains broad, revealing persistent instability and greater sensitivity to stochastic realizations under heavy congestion. Overall, increasing utilization makes the problem harder, slows convergence, and increases variability.

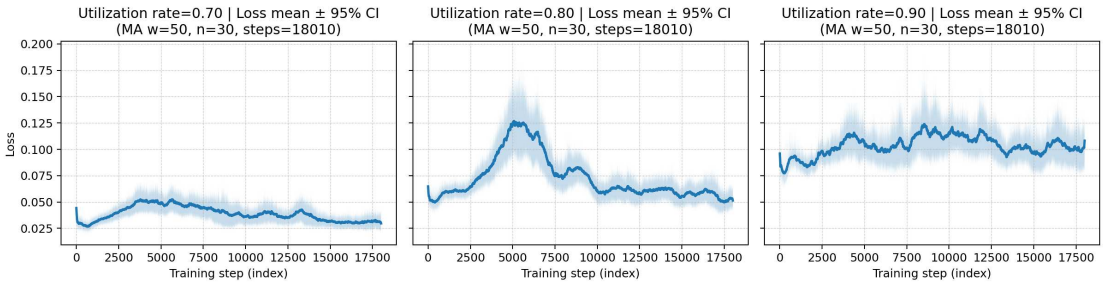


Figure 5.2: Training loss vs. step under three utilization regimes, considering MARL approach. Curves show the moving-average mean (window = 50) across n=30 independent runs; shaded bands are 95% CIs (steps = 18,010). Learning is fast and stable at 70%, shows a transient and wider variability at 80%, and remains higher/oscillatory at 90%, indicating increased difficulty and sensitivity under heavy congestion.

Figure 5.3 reports the same diagnostic for the HL approach that considers individual operation application mode and tardiness as high level reward. This result was computed as in Figure 5.3 (moving-average mean with window = 50 across n=30 runs; shaded 95% CIs). At 70% utilization the curve closely matches the non-hierarchical MARL baseline, showing a rapid drop and a tight CI, indicating that hierarchy adds little overhead in an already easy regime. At 80% utilization the transient is clearly attenuated: the peak loss is lower, the decay begins earlier, and the CI narrows sooner, consistent with the HLA stabilizing WIP and shaping LLAs' exploration through the mode vector. At 90% utilization the loss remains higher than at lighter loads but is less oscillatory and accompanied by a noticeably tighter CI, suggesting improved robustness to stochastic congestion. Overall, injecting a global intent via the HLA reduces non-stationarity seen by the shared low-level value network, yielding faster and more stable convergence in medium-to-heavy traffic while preserving performance in light traffic.

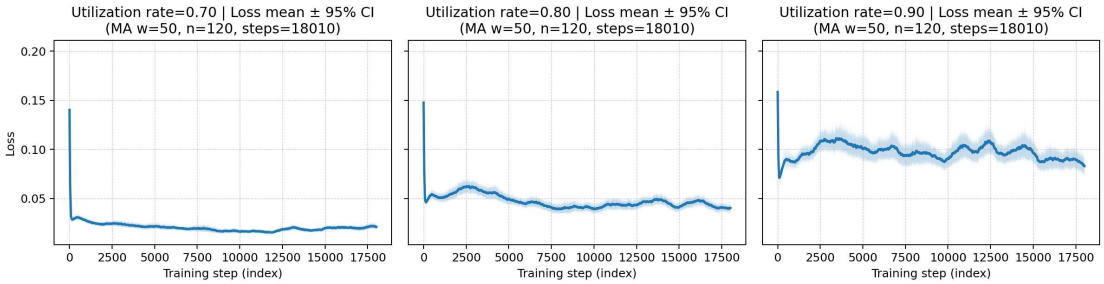


Figure 5.3: Training loss vs. step for the H-MARL approach under three utilization regimes. Curves show moving-average means (window = 50) across n=30 runs; shaded bands are 95% CIs (steps = 18,010). Relative to the non-hierarchical MARL baseline (Fig. 5.3), hierarchy (i) matches fast, stable learning at 70%, (ii) attenuates the transient at 80% (lower peak, earlier decay, tighter CI), and (iii) reduces oscillations and variability at 90%, indicating improved robustness under heavy congestion.

Building directly on the flat CTDE MARL scheduler for DJSSP proposed by [8], it was developed an algorithmic extension and a methodological upgrade. Algorithmically, it is introduced a two-level hierarchy in which a HLA issues a per-machine mode vector that conditions the shared Low-Level value network used by all per-machine agents. This preserves decentralized, fast dispatching while mitigating the non-stationarity faced by LLAs. Methodologically, we adopt a statistically grounded evaluation: for each utilization regime (70/80/90%), we run 30 independent seeds and report moving-average means with 95% confidence intervals. The resulting diagnostics show that, relative to the baseline MARL, hierarchy matches the rapid, stable convergence at low load, attenuates the transient and narrows variability at medium load, and reduces oscillations and dispersion under heavy congestion, evidencing that the HLA's global intent improves training stability and robustness without sacrificing the original CTDE design.

5.2 Validation & Performance Metrics

After training, each sequencing method is *frozen* (no learning, no exploration) and evaluated on the same validation set composed of R=100 independent runs at three utilization regimes (70%, 80%, 90%). Each run instantiates a fresh DJSSP trajectory from the stochastic generators used in this study (exponential inter-arrivals, uniform processing times, uniformly sampled due-date tightness), producing distinct event streams and queue dynamics. For fairness, all competing methods in a given regime are exposed to the same pool of validation instances.

For comparative context, both learning-based schedulers against are compared

against a set of classical dispatching rules widely used in the DJSS literature. First considering singular-factor rules that compute a priority score per job at each decision epoch. ATC balances short processing times and due-date urgency via an exponential look-ahead term, favoring jobs with small slack. Average Processing Time per Operation (AVPRO) prioritizes jobs with smaller average remaining operation times, implicitly smoothing variability along the route. COVERT ranks jobs by an urgency-per-time ratio, seeking the largest marginal tardiness reduction per unit of processing time. CR uses the ratio between time-to-due and remaining processing time; values below one indicate imminent risk of lateness. EDD sequences strictly by due dates, minimizing the number of late jobs when processing times are similar. LWKR selects the job with the smallest sum of remaining processing times, a horizon-aware alternative to myopic rules. MDD replaces the due date by the minimum between the true due date and the projected completion under current load, mitigating unrealistic earliness/latency signals. MOD extends MDD at the operation level by attributing intermediate due dates along the route. MS schedules the job with the least slack (time-to-due minus remaining processing time), emphasizing immediate urgency. NPT is a purely local variant that prioritizes the shortest next operation. SPT chooses the minimum immediate processing time and is known to reduce average flow time in light traffic. Finally, WINQ anticipates downstream congestion by preferring jobs whose next machine has the smallest queued workload. Unless a rule is natively "maximize," all indices are oriented so that lower scores are preferred to ensure comparability at selection time.

Beyond single-factor, it is also considered composite linear combinations that blend complementary signals. CR+SPT merges due-date urgency with short processing times. LWKR+SPT tempers myopia by adding horizon awareness to SPT; and LWKR+MOD couples remaining workload with operation-level due-date control. Congestion-aware blends include PT+WINQ, which trades off immediate processing time against the next-station buffer, and PT+WINQ+S, which augments that trade-off with slack to better protect due-date feasibility. Two higher-weight composites 2PT+LWKR+S and 2PT+WINQ+NPT up-weight immediate processing time while retaining either horizon- or congestion-awareness (and, in the latter case, local next-operation time). Coefficients are fixed and indices are normalized to comparable scales before summation to avoid dominance by any single term. As a simple reference, First-In–First-Out (FIFO) is included, which sequences by arrival order without regard to processing times or due-dates.

Each method i is evaluated using four complementary metrics computed over the validation runs:

$$NAT_i = \frac{\overline{CT}_i}{\min(\overline{\mathbf{CT}})}, \tag{5.1}$$

$$NCT_i = \frac{CT_i - CT_{\text{baseline}}}{CT_{\text{baseline}}}, \tag{5.2}$$

$$\text{PLJ}_i = \frac{1}{N}\sum_{j=1}^{N}\mathbb{J}_j, \tag{5.3}$$

$$\text{WR}_i = \frac{1}{100}\sum_{j=1}^{100}\mathbb{T}_i \times 100. \tag{5.4}$$

where NAT_i represents the normalized average tardiness of sequencing rule i, $\overline{CT}_i$ represents the average cumulative tardiness of sequencing rule i, $\overline{\mathbf{CT}} = [\overline{CT}_1, \ldots, \overline{CT}_I]$ represents the array of average cumulative tardiness and $\min(\cdot)$ represents the min operator. Moreover, NCT_i represents the normalized cumulative tardiness of sequencing rule i, while CT_i represents the cumulative tardiness of rule sequencing i and CT_{baseline} represents the baseline cumulative tardiness. Additionally, PLJ_i represents the proportion of late jobs when considering sequencing rule i, N represents the total number of jobs processed and $\mathbb{J}_j$ is equal to one if job j is late and zero if it is not. WR_i represents the win rate obtained by sequencing rule i and $\mathbb{T}_i$ is equal to one if the cumulative tardiness obtained by sequencing rule i is the lowest among all sequencing rules. Formally. $\mathbb{J}_j$ and $\mathbb{T}_i$ can be expressed by

$$\mathbb{J}_j = \begin{cases} 1, & \text{if job j is late,} \\ 0, & \text{otherwise,} \end{cases} \tag{5.5}$$

$$\mathbb{T}_i = \begin{cases} 1, & \text{if } CT_i < CT_j \ \forall i \neq j, \\ 0, & \text{otherwise.} \end{cases} \tag{5.6}$$

5.3 Hierarchical Sequencing Designs

Across Sections 4.5 to 4.7, 4 different variants of a Hierarchical Learning Rule Sequencing (HL-RS) approaches were proposed. They differ in the high level reward mechanism and how the high level action is applied over each LLA. The HLA reward can be constructed in a way to minimize the system cumulative tardiness or to leverage the chronological joint reward mechanism. While the action application mode can be set either globally or on a per machine basis. If set to global mode, the HLA action is always applied to every machine of the shop floor, however, if set to a machine individual application, the HLA affects exclusive a single machine. Additionally, when in global mode the HLA state is constructed using g_t, whereas when applying individually to each machine the HLA state is built by $s_{t,i}$. In such way, the HL approach can be referred to by one of the 4 following approaches:

1. Chronological Joint Reward on shared operation modes (CJ-Shared),
2. Chronological Joint Reward on individual operation modes (CJ-Ind),

3. Cumulative Tardiness on shared operation modes (Tard-Shared),

4. Cumulative Tardiness on individual operation modes (Tard-Ind).

Each of these four approaches were submitted to the event driven based simulation environment described in Chapter 3. In this way, it is possible to compare them with the MARL framework described in Chapter 4 by leveraging the set up detailed in Section 5.2.

In order to foster leverage the most performing dispatching rules, an adaptation of the low level reward mechanism was developed. This adaptation considers a small boost that increases low level agents' reward when the selection aligns with the either one of PTWINQS or CRSPT heuristics. In other words, when the low level agent selects an operation that is the same operation that either PTWINQS or CRSPT heuristics would select, a small boost is added to its reward. Considering machine k, its reward can be modeled as

$$R_{k,t}^{\text{boost}} \;=\; R_{k,t}^{\text{LL}} \;+\; \lambda\, \mathbb{I}_{k,t}, \tag{5.7}$$

where $R_{k,t}^{\text{boost}}$ is the reward obtained by machine k after the boost λ is applied, $R_{k,t}^{\text{LL}}$ is the original reward that follows the cronological joint reward echanism, λ represents the reward boosting factor, $\text{PTWINQS}(\text{s}_{\text{k,t}})$ represents the action chosen by PTWINQS sequencing rule given the state $(s_{k,t})$, $\text{CRSPT}(\text{s}_{\text{k,t}})$ represents the action chosen by CRSPT sequencing rule given the state $s_{k,t}$. Finally, $\mathbb{I}_{k,t}(a_{k,t}, \text{PTWINQS}(\text{s}_{\text{k,t}}), \text{CRSPT}(\text{s}_{\text{k,t}}))$ represents an indicator to wether or not the selected $a_{k,t}$ matches the action chosen by PTWINQS or CRSPT. It can be defined as

$$\mathbb{I}_{k,t}(a_{k,t}, \text{PTWINQS}(\text{s}_{\text{k,t}}), \text{CRSPT}(\text{s}_{\text{k,t}})) \;=\; \begin{cases} 1, & \text{if } a_{k,t} = \text{PTWINQS}(\text{s}_{\text{k,t}}), \\ 1, & \text{if } a_{k,t} = \text{CRSPT}(\text{s}_{\text{k,t}}), \\ 0, & \text{otherwise.} \end{cases} \tag{5.8}$$

Equation (5.7) preserves the shaped signal's objective orientation (tardiness reduction via queue time penalties) while giving a small, consistent increment toward the LLA's guidance. In effect, the boost plays the role of a behavioral prior: when the LLA selects an action that agrees with known well performing rules, LLA agents are slightly rewarded for agreement, otherwise they remain governed by the main shaped signal. Because λ is small, boosting does not overpower the base reward, it simply accelerates the training toward policy regions favored by well performing sequencing rules.

Moreover, all four HL-RS approaches can consider the reward boosting mechanism. The reward boosting factor λ acts, in this way, as a tunable hyperparameter. Therefore, to proper define a value for λ a hyperparameter tuning was considered

over the array $\begin{bmatrix}0.01 & 0.02 & 0.03 & 0.04 & 0.05 & 0.06 & 0.07 & 0.08 & 0.09 & 0.1\end{bmatrix}$. Due to computational complexity, when considering the hyperparameter tuning scenario only 5 independent training and validation runs were considered. After obtaining the best λ value, the 8 resulting HL approaches (4 that considers reward boosting and 4 that do not) were submited to 30 independent training and validation runs were considered and once again selected the models that achieved the best validation performance.

Both Tables 5.1 and 5.2 report the average WR over the 5 runs of the hyperparameter tuning over the boost factor λ. Table 5.1(a) reports the results considering the chronological joint reward with shared operation modes, while Table 5.1(b) reports the results considering the chronological joint reward with individual operation modes. Table 5.2(a) reports the results considering the cumulative tardiness with shared operation modes, while Table 5.2(b) reports the results considering the cumulative tardiness with individual operation modes. All tables consider all utilization rates values. The patterns are in a way consistent, however some statistical variance can be explained due to the few number of runs. Smaller boosts excel at light and moderate load, as in CJ–Shared at 80% with $\lambda = 0.01$ at 33 WR, but mid boosts become reliably strong as load increases, as seen at CT–Shared at 90% with $\lambda = 0.04$ at 24 WR and CT–Ind at 80–90% with $\lambda = 0.04$ at 33 and 16. When we aggregate across tables and loads, $\lambda = 0.04$ is the most robust single choice: it achieves the highest overall wins while being consistent enough over utilization rates. Intuitively, $\lambda = 0.04$ amplifies useful temporal credit without over-boosting noisy local signals, yielding stable improvements across reward definitions, coordination modes, hence $\lambda = 0.04$ is chosen as best value, referred as HL-Boost.

5.4 Simulation Results

Having defined the best λ value as 0.04, it becomes possible to better compare the HL approaches that consider the reward boosting mechanism with all other considered sequencing rules. Then, all 8 HL approaches are tested and compared against both the MARL-RS and the sequencing rule benchmark. Table 5.3 represents the results when the HL approaches consider CJ-Shared, Table 5.4 represents the results when the HL approaches consider CJ-Ind, while Tables 5.5 and 5.6 represents the results when the HL approaches consider Tard-Shared and Tard-Ind, respectively. Tables 5.3–5.6 show the performance of several sequencing rules over 100 validation runs at three congestion levels using three metrics: NAT, PLJ, and WR. NAT is the normalized average cumulative tardiness, obtained by (5.1), PLJ is the percentage of late jobs, obtained by (5.3) and WR is the fraction of runs in which a rule attains the lowest cumulative tardiness, obtained by (5.4).

HL-RS when set as CJ-Shared, produces the results depicted in Table 5.3. It

(a) Win rates percentages — Chronological Joint Reward - Shared operation

	Utilization rate		
Boost Factor	70%	80%	90%
0.01	10	**33**	15
0.02	13	16	8
0.03	7	26	17
0.04	16	28	12
0.05	9	13	9
0.06	12	26	9
0.07	13	21	12
0.08	13	12	18
0.09	**17**	19	13
0.10	14	15	12

(b) Win rates percentages — Chronological Joint Reward - Individual operation

	Utilization rate		
Boost Factor	70%	80%	90%
0.01	20	22	8
0.02	**19**	17	15
0.03	16	19	19
0.04	15	17	**21**
0.05	11	15	19
0.06	11	**28**	6
0.07	10	22	11
0.08	12	18	18
0.09	17	15	**21**
0.10	8	17	11

Table 5.1: Hyperparameter sweep of reward-boost factor λ under the chronological joint high-level reward. Cells report average win rate (WR, %) over 5 independent training/validation runs at three utilization levels (70%, 80%, 90%). (a) Shared: a single mode broadcast to all machines; (b) Individual: mode set per machine. Boldface marks the highest WR within each utilization column.

shows that HL-RS is competitive to dominant as congestion rises. It attains the best NAT at 80% and 90% (both 100%) and the top WR at 80% (23%) and 90% (21%). At 70%, PTWINQS leads on NAT (100%) and WR (26%), indicating that in light traffic a queue aware heuristic can edge out learned policies. PLJ shows a different picture: SPT delivers the lowest late-job rate at 70% (44.87%), while AVPRO minimizes PLJ at 80% and 90% (54.73% and 61.06%), yet both have poor NAT and WR values. MARL-RS often exhibits lower PLJ than HL-RS, such as 49.45% vs. 51.48% at 70%, but loses in NAT and WR, suggesting that although MARL-RS keeps more jobs on time, the lateness when it occurs tends to be larger, increasing cumulative tardiness.

Analysing Table 5.4, it can be seen that once again performance shifts with congestion. At 70%, PTWINQS is best on NAT (100%) and WR (26%). At 80%, HL-Boost reaches the best NAT (100%) while HL-RS and HL-Boost tie on WR (both 19%), signaling that the boosting mechanism can match HL-RS in moderate load. At 90%, HL-RS takes the lead with the best NAT (100%) and highest WR (23%). As before, the lowest PLJ is achieved by rules like SPT and AVPRO rather than by the NAT or WR winners, reinforcing that minimizing the proportion of late jobs does not necessarily minimize cumulative tardiness.

(a) Win rates percentages — Cumulative Tardiness - Individual operation

	Utilization rate		
Boost Factor	70%	80%	90%
0.01	16	16	14
0.02	14	21	5
0.03	18	16	10
0.04	20	**33**	**16**
0.05	**24**	7	9
0.06	11	20	15
0.07	13	10	15
0.08	19	10	6
0.09	11	14	7
0.10	12	17	8

(b) Win rates percentages — Cumulative Tardiness - Shared operation

	Utilization rate		
Boost Factor	70%	80%	90%
0.01	**28**	24	13
0.02	18	16	4
0.03	10	25	8
0.04	26	21	**24**
0.05	9	17	16
0.06	15	20	12
0.07	16	11	10
0.08	18	**26**	4
0.09	18	18	21
0.10	20	13	16

Table 5.2: Hyperparameter sweep of reward-boost factor λ under the cumulative tardiness (Tard) high-level reward. Entries are average win rate (WR, %) over 5 runs at 70%, 80%, and 90% utilization. (a) Individual: mode set per machine; (b) Shared: single mode broadcast. Boldface denotes the highest WR per utilization.

Moving on to the case when HL is set as Tard-Shared, represented in Table 5.5. In this scenario the leadership alternates, PTWINQS tops NAT at 70% and 90% (100% in both) and has the highest WR at 90% (17%). HL-RS dominates the 80% block, with best NAT (100%) and the strongest WR margin (31%). HL-Boost is consistently close but generally trails HL-RS on NAT and WR. The PLJ minima are again delivered by SPT and AVPRO, not by the NAT or WR leaders.

Finally, the last possible way HL can be set is Tard-Ind, its results are shown in Table 5.6. This is the clearest win for the hierarchical approach, HL-RS sweeps NAT across all utilization rates (exactly 100% at 70%, 80%, and 90%) and is the top WR in each block (32%, 26%, and 20%). HL-Boost and PTWINQS are competitive but consistently behind HL-RS on NAT and WR. As in other tables, SPT and AVPRO minimize PLJ, but at the cost of substantially worse NAT and WR.

From Tables 5.3- 5.6, several observations emerge. Hierarchy helps under stress, as utilization rises, HL-RS gains strength. Frequently taking both best NAT and highest WR, at 80–90% in CJ-Shared and CJ-Ind, and it outright dominates under Tard-Ind. HL-Boost occasionally matches or ties HL-RS (for example, best NAT at 80% in CJ-Ind and a tied WR there), but it rarely surpasses HL-RS and is less robust at high congestion, with NAT exceeding 100% at 90% in CJ-Shared

	Utilization Rate = 70%			Utilization = 80%			Utilization Rate = 90%		
Method	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)
HL-RS	105.01%	51.48%	15.00%	**100.00%**	63.23%	**23.00%**	**100.00%**	74.47%	**21.00%**
HL-Boost	109.86%	53.18%	11.00%	100.30%	63.88%	22.00%	106.75%	72.33%	10.00%
MARL-RS	110.16%	49.45%	11.00%	106.04%	57.52%	15.00%	106.60%	65.94%	10.00%
PTWINQS	**100.00%**	58.77%	**26.00%**	104.32%	74.23%	13.00%	104.47%	85.78%	15.00%
ATC	115.76%	58.04%	2.00%	114.48%	73.72%	1.00%	113.32%	84.28%	2.00%
AVPRO	159.13%	45.55%	0.00%	134.31%	**54.73%**	0.00%	123.72%	**61.06%**	0.00%
COVERT	149.78%	64.66%	0.00%	132.69%	78.53%	0.00%	124.22%	88.58%	0.00%
CR	112.82%	66.36%	9.00%	116.11%	83.14%	4.00%	113.17%	92.35%	0.00%
CRSPT	117.94%	45.39%	2.00%	108.98%	58.00%	1.00%	103.93%	66.86%	10.00%
DPTLWKRS	110.31%	56.29%	10.00%	113.80%	72.78%	2.00%	111.51%	82.78%	2.00%
DPTWINQNPT	134.79%	45.77%	0.00%	117.80%	56.94%	1.00%	112.25%	66.46%	3.00%
EDD	116.46%	57.32%	1.00%	116.27%	73.02%	1.00%	115.99%	82.97%	0.00%
FIFO	199.80%	58.93%	0.00%	178.93%	72.61%	0.00%	174.13%	82.74%	0.00%
LWKR	159.22%	48.39%	0.00%	135.61%	57.70%	0.00%	123.71%	64.78%	0.00%
LWKRMOD	117.20%	53.46%	1.00%	110.36%	67.73%	2.00%	108.22%	78.18%	6.00%
LWKRSPT	147.14%	47.25%	0.00%	125.00%	56.41%	3.00%	113.98%	63.06%	7.00%
MDD	114.51%	53.73%	3.00%	110.79%	67.80%	5.00%	109.14%	78.74%	7.00%
MOD	112.95%	56.79%	2.00%	109.52%	72.77%	4.00%	109.57%	83.35%	3.00%
MS	115.62%	62.39%	6.00%	120.53%	79.17%	2.00%	118.80%	89.33%	0.00%
NPT	183.01%	52.34%	0.00%	154.55%	61.30%	0.00%	137.66%	68.88%	0.00%
PTWINQ	133.06%	46.34%	0.00%	119.51%	57.83%	1.00%	112.89%	66.85%	1.00%
SPT	130.58%	**44.87%**	1.00%	118.36%	55.27%	0.00%	112.85%	63.58%	3.00%
WINQ	161.85%	51.52%	0.00%	139.93%	62.08%	0.00%	130.97%	71.07%	0.00%

Table 5.3: Full benchmark with CJ high-level reward and shared mode selection. Metrics over 30 validation runs at 70%, 80%, and 90% utilization: NAT (normalized average cumulative tardiness; lower is better, the best method in each block equals 100%, PLJ (late-job rate; lower is better), and WR (win rate; higher is better). HL-Boost uses $\lambda = 0.04$. Boldface highlights the best value per column (lowest NAT and PLJ, highest WR).

and Tard-Ind. PTWINQS excels in light traffic and sometimes even at very high load. Under Tard-Shared, it repeatedly wins the 70% blocks and captures the 90% NAT and WR, yet it tends to degrade in the 80–90% range in the other setups. There is a NAT–PLJ trade-off, rules that minimize the fraction of late jobs (SPT, AVPRO) often perform poorly on cumulative tardiness (NAT) and rarely win runs (WR). Conversely, MARL-RS frequently attains lower PLJ than HL-RS but still loses on NAT and WR, implying larger lateness magnitudes when tardy events occur. Practically, policy choice should hinge on whether the objective is to minimize the number of late jobs or the total lateness. Moreover, some baselines are consistently weak. FIFO and NPT remain clearly inferior across metrics and utilizations, and COVERT and WINQ also underperform on NAT and WR despite middling PLJ. Finally, if the primary goal is minimizing cumulative tardiness, HL-RS set as Tard-Ind is the most reliable choice, since it achieves best in class NAT and the highest WR in all utilization rates (Table 5.6). However, if the goal is to prioritize few late jobs over total lateness, rules like SPT or AVPRO minimize

	Utilization Rate = 70%			Utilization = 80%			Utilization Rate = 90%		
Method	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)
HL-RS	103.48%	55.04%	20.00%	102.97%	60.98%	**19.00%**	**100.00%**	72.93%	**23.00%**
HL-Boost	106.81%	55.43%	12.00%	**100.00%**	63.95%	**19.00%**	101.86%	69.71%	13.00%
MARL-RS	107.84%	50.32%	10.00%	104.83%	57.52%	14.00%	104.69%	65.94%	12.00%
PTWINQS	**100.00%**	58.95%	**26.00%**	103.13%	74.23%	16.00%	102.59%	85.78%	10.00%
ATC	119.04%	60.89%	4.00%	113.18%	73.72%	0.00%	111.28%	84.28%	2.00%
AVPRO	151.25%	47.38%	0.00%	132.78%	**54.73%**	0.00%	121.50%	**61.06%**	0.00%
COVERT	148.29%	66.09%	0.00%	131.18%	78.53%	0.00%	121.99%	88.58%	0.00%
CR	112.42%	68.09%	6.00%	114.79%	83.14%	5.00%	111.13%	92.35%	1.00%
CRSPT	118.83%	47.89%	1.00%	107.74%	58.00%	3.00%	102.06%	66.86%	10.00%
DPTLWKRS	110.69%	57.48%	3.00%	112.50%	72.78%	2.00%	109.50%	82.78%	3.00%
DPTWINQNPT	129.21%	47.59%	1.00%	116.46%	56.94%	2.00%	110.24%	66.46%	4.00%
EDD	115.73%	58.54%	3.00%	114.94%	73.02%	1.00%	113.91%	82.97%	0.00%
FIFO	193.60%	60.04%	0.00%	176.89%	72.61%	0.00%	171.00%	82.74%	0.00%
LWKR	156.60%	49.71%	0.00%	134.07%	57.70%	0.00%	121.49%	64.78%	0.00%
LWKRMOD	116.78%	54.89%	2.00%	109.11%	67.73%	5.00%	106.28%	78.18%	5.00%
LWKRSPT	147.37%	49.21%	0.00%	123.57%	56.41%	3.00%	111.93%	63.06%	6.00%
MDD	114.94%	54.62%	1.00%	109.53%	67.80%	4.00%	107.18%	78.74%	5.00%
MOD	112.26%	58.30%	5.00%	108.28%	72.77%	2.00%	107.60%	83.35%	2.00%
MS	114.85%	62.66%	6.00%	119.16%	79.17%	3.00%	116.66%	89.33%	0.00%
NPT	175.52%	52.54%	0.00%	152.79%	61.30%	0.00%	135.19%	68.88%	0.00%
PTWINQ	132.11%	48.05%	0.00%	118.15%	57.83%	2.00%	110.87%	66.85%	2.00%
SPT	127.69%	**46.09%**	0.00%	117.01%	55.27%	0.00%	110.82%	63.58%	2.00%
WINQ	159.00%	53.23%	0.00%	138.33%	62.08%	0.00%	128.62%	71.07%	0.00%

Table 5.4: Full benchmark with CJ high-level reward and individual mode selection. Metrics over 30 validation runs at 70%, 80%, and 90% utilization: NAT (normalized average cumulative tardiness; lower is better, the best method in each block equals 100%, PLJ (late-job rate; lower is better), and WR (win rate; higher is better). HL-Boost uses $\lambda = 0.04$. Boldface highlights the best value per column (lowest NAT and PLJ, highest WR).

PLJ, but expect worse overall tardiness (higher NAT) and fewer wins (lower WR).

As done in Section 5.1 an analysis considering only the best performing sequencing rules was conducted. Figures 5.4 and 5.5 display the performance distributions across all test instances, leveraging violin plots. The vertical axis is the Normalized Cumulative Tardiness (NCT, %), obtained by (5.2). Each violin encodes the empirical distribution (its width is the density), the center marker denotes the median, and whiskers indicate the observed range. Each row in Figures 5.4 and 5.5 are grouped by shop utilization rates (70%, 80%, 90%), whereas columns are how the HLA is set (Shared or Individual). Red violins correspond to the proposed hierarchical methods, HL-RS and HL-Boost, while blue violins are benchmark dispatching rules. The dashed horizontal line represents the mean obtained by HL-RS, chosen as reference since HL-RS Tard-Ind was the best performing over the complete benchmark. Figure 5.6 reports the win rate (WR, %), i. e. the fraction of instances (within each utilization) in which a method achieves the best NCT, obtained by (5.4). Bars are grouped by how the high level is set CJ-Shared, CJ-Ind,

	Utilization Rate = 70%			Utilization = 80%			Utilization Rate = 90%		
Method	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)
HL-RS	100.48%	49.82%	**24.00%**	**100.00%**	60.38%	**31.00%**	100.52%	64.15%	13.00%
HL-Boost	105.99%	51.09%	13.00%	109.72%	64.47%	10.00%	100.86%	66.47%	10.00%
MARL-RS	109.76%	49.48%	8.00%	111.25%	56.44%	8.00%	103.84%	64.75%	10.00%
PTWINQS	**100.00%**	57.86%	**24.00%**	107.18%	72.75%	13.00%	**100.00%**	83.14%	**17.00%**
ATC	118.47%	58.61%	0.00%	120.42%	72.16%	1.00%	108.38%	81.89%	1.00%
AVPRO	154.48%	46.29%	0.00%	142.84%	**52.47%**	1.00%	124.39%	**59.93%**	0.00%
COVERT	148.81%	65.55%	0.00%	140.28%	78.05%	0.00%	121.96%	86.21%	0.00%
CR	111.30%	67.32%	7.00%	118.90%	82.03%	5.00%	110.92%	90.47%	1.00%
CRSPT	118.27%	46.16%	2.00%	112.77%	56.06%	5.00%	100.38%	65.79%	15.00%
DPTLWKRS	111.80%	56.61%	5.00%	115.22%	69.91%	3.00%	107.45%	80.03%	2.00%
DPTWINQNPT	134.88%	46.75%	0.00%	126.41%	55.42%	0.00%	110.48%	64.43%	1.00%
EDD	116.33%	56.98%	0.00%	120.10%	71.16%	1.00%	109.87%	80.86%	0.00%
FIFO	197.29%	59.02%	0.00%	189.80%	69.95%	0.00%	170.08%	80.13%	0.00%
LWKR	152.11%	49.05%	1.00%	140.22%	56.28%	0.00%	122.59%	63.57%	0.00%
LWKRMOD	116.36%	53.37%	2.00%	112.26%	65.36%	4.00%	104.09%	75.54%	10.00%
LWKRSPT	143.92%	47.46%	0.00%	129.09%	54.69%	3.00%	111.07%	61.56%	3.00%
MDD	114.22%	53.23%	4.00%	115.26%	66.42%	6.00%	103.68%	76.67%	8.00%
MOD	111.75%	56.64%	2.00%	114.47%	70.17%	2.00%	104.17%	80.92%	3.00%
MS	116.06%	62.50%	8.00%	121.27%	77.38%	5.00%	115.73%	86.75%	4.00%
NPT	176.30%	51.20%	0.00%	157.26%	60.16%	0.00%	134.87%	67.03%	0.00%
PTWINQ	136.19%	47.37%	0.00%	126.16%	55.84%	1.00%	111.09%	64.32%	1.00%
SPT	131.11%	**45.57%**	0.00%	122.98%	53.95%	1.00%	111.09%	62.68%	1.00%
WINQ	163.29%	51.50%	0.00%	150.16%	60.88%	0.00%	129.99%	69.79%	0.00%

Table 5.5: Full benchmark with Tard high-level reward and shared mode selection. Metrics over 30 validation runs at 70%, 80%, and 90% utilization: NAT (normalized average cumulative tardiness; lower is better, the best method in each block equals 100%, PLJ (late-job rate; lower is better), and WR (win rate; higher is better). HL-Boost uses $\lambda = 0.04$. Boldface highlights the best value per column (lowest NAT and PLJ, highest WR).

Tard-Shared, Tard-Ind.

In Figure 5.4, both hierarchical strategies stay above the mojority of other sequencing rules and show similar spreads to most benchmarks, with the advantage decreasing as utilization rises. At 70% (light congestion), most rules are viable, but HL-RS and HL-Boost achieve higher means and smaller spreads than the majority of baselines, falling behind only to PTWINQS. At 80%, the differences is diminished, the HL methods overcome PTWONQS at CJ-Shared and the distance at CJ-Ind is smaller. Additionally, they remain on top of other rules with shorter upper tails, indicating better and more stable performance, while several baselines develop heavy upper tails, signaling reduced reliability. Near saturation at 90%, the HL methods overcome all other rules, the benefit of hierarchical coordination is most evident here. Under the chronological joint reward, individual credit yields slightly higher means at 70–80%; at 90% the gap narrows, but HL remains clearly superior to all baselines in both the Shared and Individual variants.

	Utilization Rate = 70%			Utilization = 80%			Utilization Rate = 90%		
Method	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)	*NAT* (%)	PLJ (%)	WR (%)
HL-RS	**100.00%**	51.07%	**32.00%**	**100.00%**	58.80%	**26.00%**	100.00%	69.82%	**20.00%**
HL-Boost	110.77%	48.91%	8.00%	100.49%	57.67%	21.00%	106.21%	65.54%	11.00%
MARL-RS	111.02%	49.70%	8.00%	111.61%	53.14%	2.00%	106.71%	64.31%	11.00%
PTWINQS	107.27%	59.36%	13.00%	103.06%	67.84%	19.00%	102.24%	83.50%	14.00%
ATC	119.67%	58.98%	2.00%	117.51%	68.14%	0.00%	111.13%	82.57%	1.00%
AVPRO	154.32%	46.11%	0.00%	144.59%	**51.23%**	0.00%	123.13%	**59.85%**	2.00%
COVERT	154.33%	66.29%	0.00%	141.49%	73.80%	0.00%	123.18%	86.68%	0.00%
CR	117.07%	67.41%	8.00%	116.12%	76.70%	6.00%	111.12%	90.08%	2.00%
CRSPT	118.69%	45.84%	2.00%	111.11%	51.75%	5.00%	103.20%	65.93%	10.00%
DPTLWKRS	113.61%	57.25%	7.00%	112.58%	66.42%	5.00%	109.72%	81.14%	4.00%
DPTWINQNPT	136.18%	47.27%	0.00%	124.10%	52.09%	0.00%	111.61%	64.68%	0.00%
EDD	118.58%	58.27%	2.00%	116.96%	68.09%	1.00%	113.92%	81.92%	0.00%
FIFO	204.07%	60.23%	0.00%	190.34%	67.09%	0.00%	171.45%	80.19%	0.00%
LWKR	157.16%	48.96%	0.00%	146.57%	54.23%	0.00%	122.82%	63.31%	0.00%
LWKRMOD	119.44%	54.48%	3.00%	116.87%	62.34%	0.00%	106.23%	76.04%	6.00%
LWKRSPT	146.26%	47.52%	0.00%	136.93%	52.97%	2.00%	114.43%	62.08%	3.00%
MDD	116.19%	54.87%	5.00%	114.34%	63.38%	5.00%	109.33%	75.92%	9.00%
MOD	115.36%	57.89%	3.00%	112.82%	67.30%	2.00%	107.79%	81.72%	4.00%
MS	117.74%	63.43%	7.00%	117.52%	72.47%	6.00%	114.79%	86.51%	1.00%
NPT	181.70%	52.50%	0.00%	164.86%	57.14%	0.00%	137.20%	67.32%	0.00%
PTWINQ	135.52%	47.62%	0.00%	123.98%	52.98%	0.00%	113.71%	64.44%	1.00%
SPT	133.30%	**45.09%**	0.00%	123.60%	49.98%	0.00%	113.81%	62.82%	1.00%
WINQ	163.61%	52.14%	0.00%	148.77%	57.44%	0.00%	132.30%	69.35%	0.00%

Table 5.6: Full benchmark with Tard high-level reward and individual mode selection. Metrics over 30 validation runs at 70%, 80%, and 90% utilization: NAT (normalized average cumulative tardiness; lower is better, the best method in each block equals 100%, PLJ (late-job rate; lower is better), and WR (win rate; higher is better). HL-Boost uses $\lambda = 0.04$. Boldface highlights the best value per column (lowest NAT and PLJ, highest WR).

In Figure 5.5, aligning the high-level signal with the objective (tardiness) preserves the advantages of hierarchy and often strengthens them, especially at 70–80% utilization. At 70%, HL-RS is only smallet than PTWINQS over the Shared mode, it remains above all other rules. HL-Boost remain competitive yet typically lag and exhibit fat tails on Individual mode. At 80%, HL-RS attains a very high NCT, the greatest among all rules, with HL-Boost close behind on Individual and a bit further on Shared modes. Benchmark performance degrades with long upper and lower tails. Under heavy load at 90%, HL remains robust, maintaining high means and shorter tails as many heuristics become volatile. With the tardiness signal, setting machine modes individually presents itself more beneficial than allowing machines to share operation modes. Specially, considering that, HL-RS always comes up with the highest mean NCT across all utilization rates.

In Figure 5.6, which consolidates the frequency of best results on terms of WR, at 70% utilization HL-RS on Tard-Ind achieves the highest win rate of 40%, while HL-Boost falls behind at 9%. Benchmarks sit mostly in the single digits to low

teens, with PTWINQS being an exception that is able to compete with HL-RS. At 80% utilization, HL-RS on Tard-Shared leads at 35% WR, with HL-Boost close behind (mid–20s), and the best benchmark trailing by a clear margin (low 20s) being once again PTWINQS. Near saturation at 90%, HL-RS (CJ-Ind) wins most often at 26% win rate, while in this scenario PTWINQS falls short and the WR becomes more evenly spread across benchmark rules. Overall, Figure 5.6 clearly illustrates that hierarchical methods dominate win rate.

HL-RS and HL-Boost consistently reduce NCT relative to strong dispatching baselines and do so more reliably, especially as utilization increases. Second, setting the HLA properly matters: objective aligned (tardiness) signal is more effective under low congestion, while CJ is more valuable at moderate congestion. Among the compared rules, PTWINQS and MARL-RS are the most competitive, while due date centric or simple ratio rules develop wider high tardiness tails as utilization rises. Overall, hierarchy improves not only average performance but also robustness by avoiding catastrophic tails, crucial for such environments.

Finally, since HL-RS set under Tard-Ind consistently produced the best results, the experiment was repeated to understand how it would behave when not running against its reward boosted counterpart. Table 5.7 and Fig. 5.7 confirm that HL-RS remains the best method across all utilization rates. By construction NAT is normalized to the best method, and HL-RS attains 100% at 70/80/90% utilization, whereas the strongest competitors stay above it. At 70% the closest rule is PTWINQS (107.27%), at 80% PTWINQS narrows the gap (103.06%), and at 90% it is again the nearest challenger (102.24%). The corresponding win rates (WR) in the full method set of Table 5.7 are 35%, 30%, and 22% for 70/80/90% utilization, respectively. Which are the highest among all rules. When we plot only the reduced competitors set in Figure 5.7, the absolute number of HL-RS wins is unchanged but, because fewer methods are considered, the WR percentage appears slightly higher (43%, 35%, 25% for 70/80/90%). The difference arises from the denominator, Table 5.7 computes WR over all listed rules, whereas Figure 5.7 shows a subset. Indicating that it consistently wins over other rules.

The left column of Figure 5.7 (violin plots) shows that HL-RS has a higher mean than the best heuristics in each utilization regime, since all other rules lies below the dashed horizontal line that represents HL-RS mean. Competing rules exhibit heavier upper tails, indicating occasional large tardiness spikes. This distributional robustness is reflected on the right column (WR bars), where HL-RS wins at all loads, while PTWINQS is the only heuristic that becomes competitive as congestion increases (30% WR at 80% and 22% at 90% in the reduced set).

Although HL-RS minimizes NAT, it does not always yield the smallest fraction of late jobs. For instance, at 70% utilization the lowest PLJ values are achieved by due date or service time oriented heuristics such as SPT (45.09%) and CRSPT (45.84%), and AVPRO reaches 51.23% at 80% and 59.85% at 90%. However,

these same rules incur substantially higher NAT revealing a classic trade-off: they reduce the count of late jobs by letting a smaller subset become very late, inflating cumulative tardiness. HL-RS, in contrast, reduces the magnitude of lateness, curbing catastrophic tails and improving overall weighted tardiness.

Removing the reward–boosted competitor does not alter the central conclusion, HL-RS (Tard-Ind) remains the top performing among the tested rules, offering the best average performance and the most reliable distribution across utilizations. At moderate and high congestion (80–90%), PTWINQS is the most credible baseline, but it still trails HL-RS both in NAT and in win frequency.

	Utilization Rate = 70%			Utilization = 80%			Utilization Rate = 90%		
Method	NAT (%)	PLJ (%)	WR (%)	NAT (%)	PLJ (%)	WR (%)	NAT (%)	PLJ (%)	WR (%)
HL-RS	**100.00%**	51.07%	**35.00%**	**100.00%**	58.80%	**30.00%**	**100.00%**	69.82%	**22.00%**
MARL-RS	111.02%	49.70%	9.00%	111.61%	53.14%	3.00%	106.71%	64.31%	11.00%
PTWINQS	107.27%	59.36%	14.00%	103.06%	67.84%	22.00%	102.24%	83.50%	18.00%
ATC	119.67%	58.98%	2.00%	117.51%	68.14%	1.00%	111.13%	82.57%	1.00%
AVPRO	154.32%	46.11%	0.00%	144.59%	**51.23%**	0.00%	123.13%	**59.85%**	2.00%
COVERT	154.33%	66.29%	0.00%	141.49%	73.80%	0.00%	123.18%	86.68%	0.00%
CR	117.07%	67.41%	8.00%	116.12%	76.70%	9.00%	111.12%	90.08%	2.00%
CRSPT	118.69%	45.84%	2.00%	111.11%	51.75%	10.00%	103.20%	65.93%	12.00%
DPTLWKRS	113.61%	57.25%	8.00%	112.58%	66.42%	5.00%	109.72%	81.14%	4.00%
DPTWINQNPT	136.18%	47.27%	0.00%	124.10%	52.09%	0.00%	111.61%	64.68%	1.00%
EDD	118.58%	58.27%	2.00%	116.96%	68.09%	1.00%	113.92%	81.92%	0.00%
FIFO	204.07%	60.23%	0.00%	190.34%	67.09%	0.00%	171.45%	80.19%	0.00%
LWKR	157.16%	48.96%	0.00%	146.57%	54.23%	0.00%	122.82%	63.31%	0.00%
LWKRMOD	119.44%	54.48%	3.00%	116.87%	62.34%	0.00%	106.23%	76.04%	7.00%
LWKRSPT	146.26%	47.52%	0.00%	136.93%	52.97%	2.00%	114.43%	62.08%	4.00%
MDD	116.19%	54.87%	6.00%	114.34%	63.38%	6.00%	109.33%	75.92%	9.00%
MOD	115.36%	57.89%	3.00%	112.82%	67.30%	4.00%	107.79%	81.72%	4.00%
MS	117.74%	63.43%	7.00%	117.52%	72.47%	6.00%	114.79%	86.51%	1.00%
NPT	181.70%	52.50%	0.00%	164.86%	57.14%	0.00%	137.20%	67.32%	0.00%
PTWINQ	135.52%	47.62%	0.00%	123.98%	52.98%	0.00%	113.71%	64.44%	1.00%
SPT	133.30%	**45.09%**	1.00%	123.60%	49.98%	1.00%	113.81%	62.82%	1.00%
WINQ	163.61%	52.14%	0.00%	148.77%	57.44%	0.00%	132.30%	69.35%	0.00%

Table 5.7: Full benchmark with Tard high-level reward and individual mode selection excluding the rewarding boosting variant method. This removal allows to better understand how the most consistent approach behaves. Metrics over 30 validation runs at 70%, 80%, and 90% utilization: NAT (normalized average cumulative tardiness; lower is better, the best method in each block equals 100%, PLJ (late-job rate; lower is better), and WR (win rate; higher is better). Boldface highlights the best value per column (lowest NAT and PLJ, highest WR).

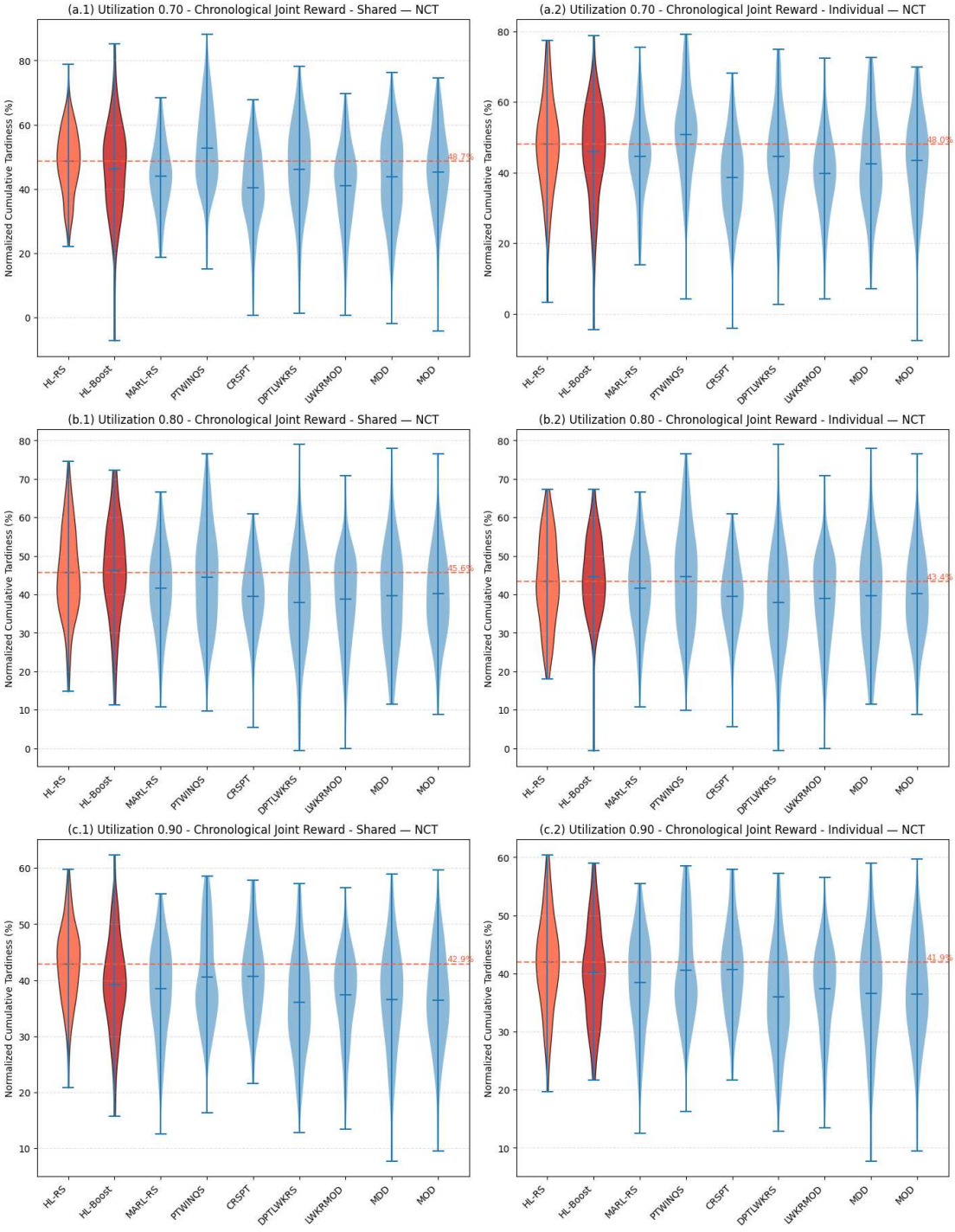


Figure 5.4: Distribution of normalized cumulative tardiness under the chronological joint high-level reward across utilization levels and mode setting regimes. Red violins are hierarchical methods; blue violins are benchmark dispatching rules.

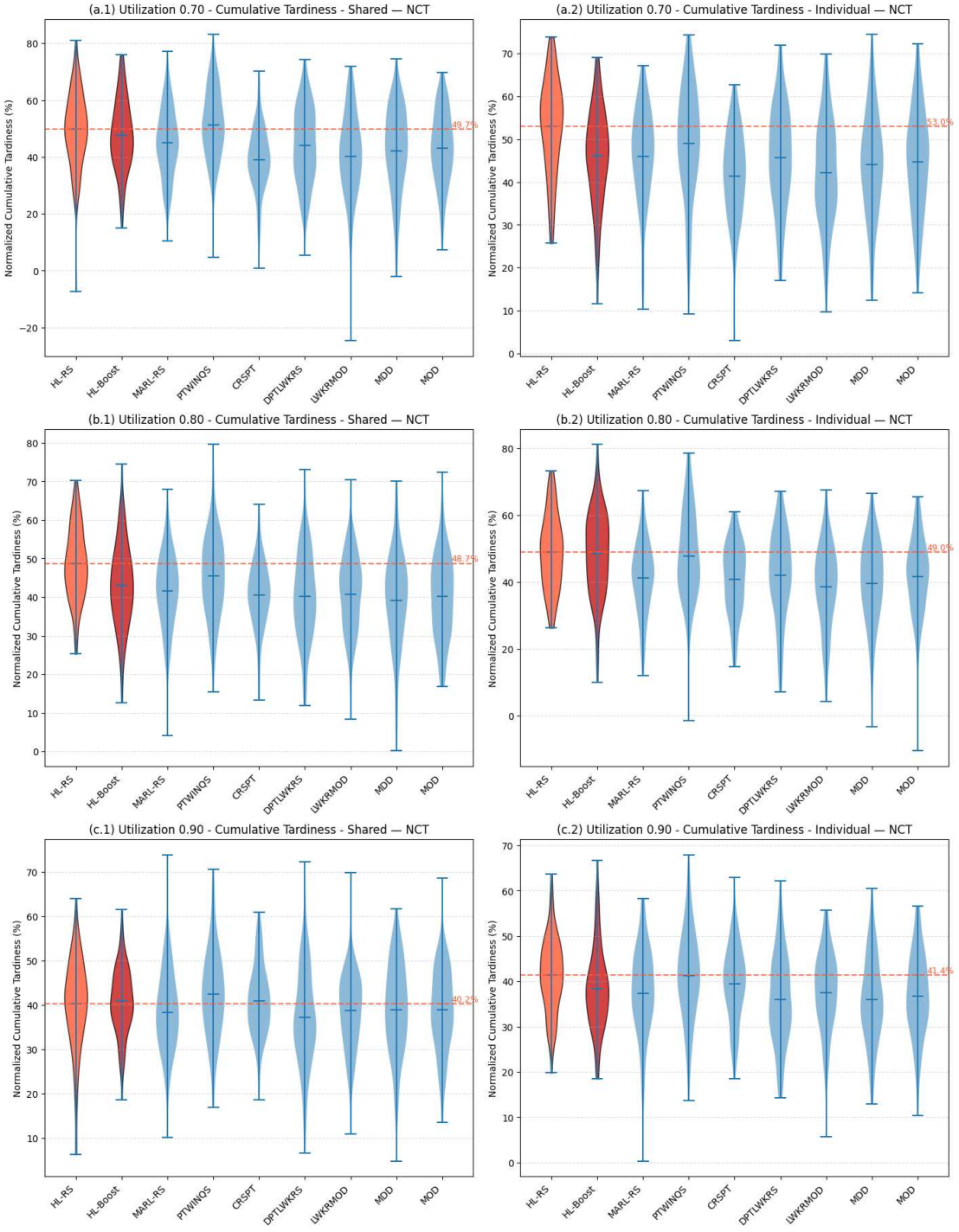


Figure 5.5: Distribution of normalized cumulative tardiness under the tardiness high-level reward across utilization levels and mode setting regimes. Red violins are hierarchical methods; blue violins are benchmark dispatching rules.

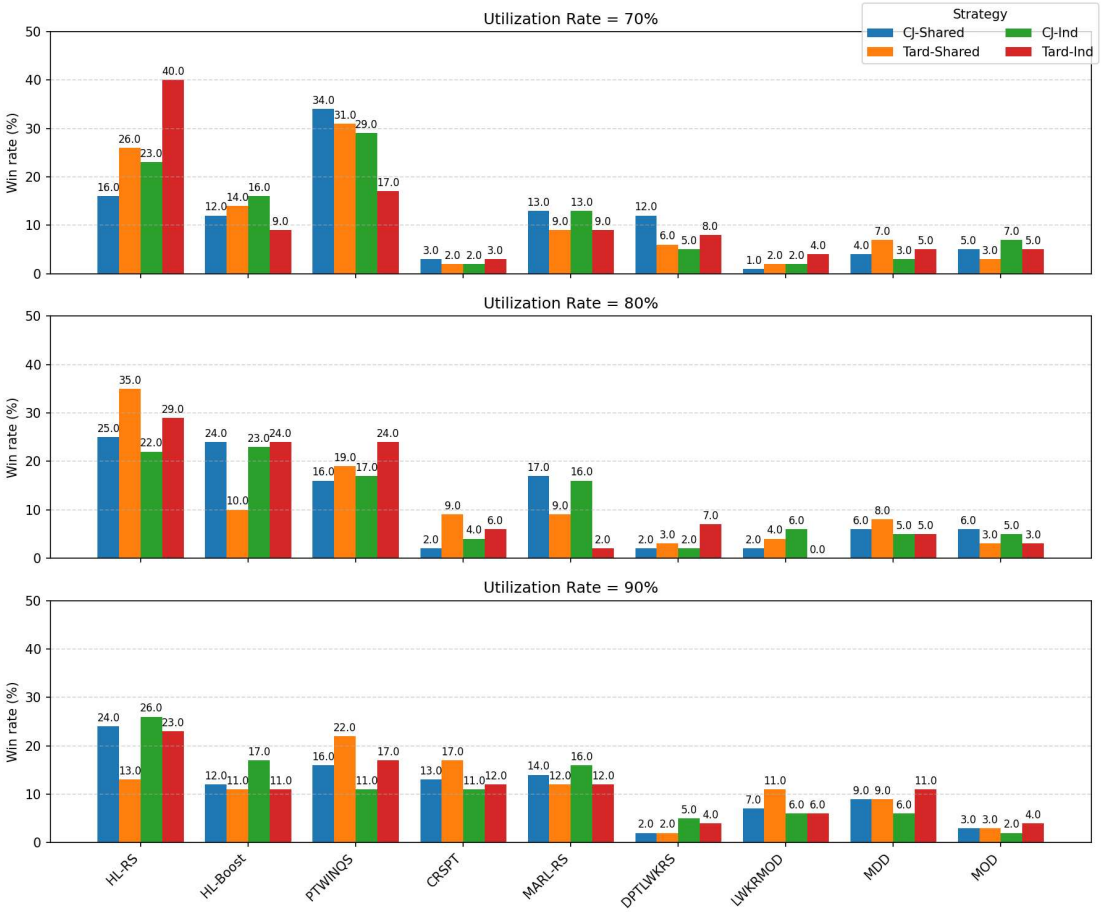


Figure 5.6: Win rate (WR, %) grouped by high-level setup (CJ–Shared, CJ–Individual, Tard–Shared, Tard–Individual) and by utilization level (70%, 80%, 90%). Bars compare hierarchical methods against strong dispatching baselines over 100 validation runs per utilization. Higher is better.

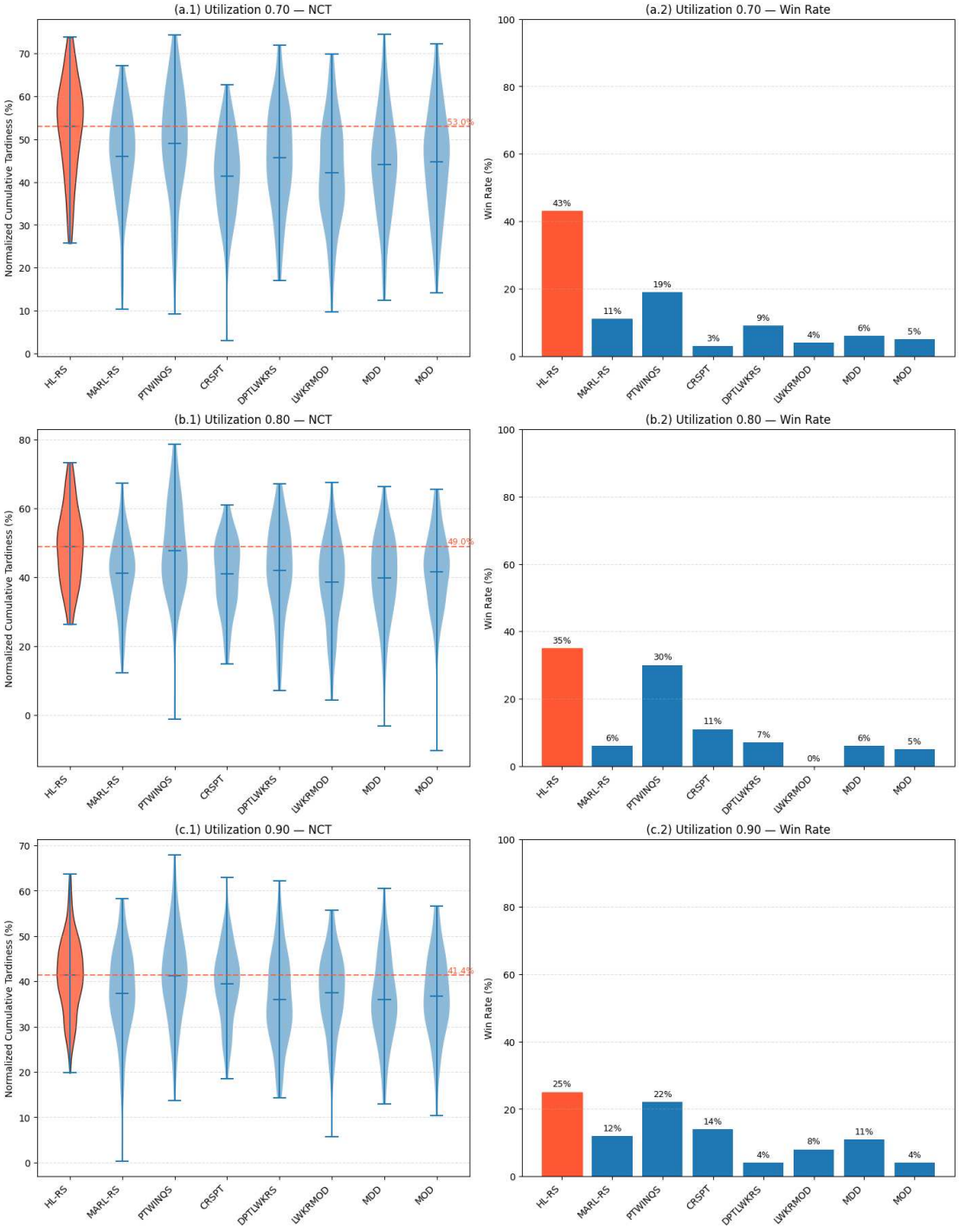


Figure 5.7: Reduced competitor benchmark under Tard–Individual. Left: violin plots of NCT (%) comparing HL-RS against top baselines; the dashed line is HL-RS mean. Right: WR (%) over the same set. Higher NCT and WR are better.

Chapter 6

Conclusions & Future Work

6.1 Conclusion

This thesis investigated reinforcement learning for the DJSSP in an event-driven, asynchronous setting where machines act locally and tardiness is realized only at job completion. It was first revisited the MARL baseline under a CTDE scheme with parameter sharing, adopting two architectural elements that proved decisive in dynamic contexts: a fixed-size MR state to preserve job-specific detail while remaining size-agnostic to fluctuating queue lengths and a chronological joint-action view with knowledge-based reward shaping to align sparse, delayed tardiness with the actions that caused it, thereby mitigating the distributed credit-assignment challenge.

Building on this foundation, it was proposed a HL extension in which a high-level manager sets machine operation modes over short commitment windows, while low-level (per-machine) agents perform job selection. Two families of high-level signals were studied, chronological-joint and tardiness aligned, each under Shared and Individual mode settings. It was further introduced a lightweight reward-boost mechanism for the low level agents and calibrated its strength via a simple hyperparameter search, across utilization levels, $\lambda = 0.04$ emerged as the most robust setting, amplifying informative temporal credit without destabilizing learning.

Comprehensive simulation results across 70%, 80%, and 90% utilization show that hierarchy systematically improves robustness and top-line performance against strong dispatching-rule baselines and the MARL baseline. In particular, HL-RS with Tard-Ind consistently delivered the best normalized average cumulative tardiness (NAT) and the highest win rates, while maintaining shorter tails under heavy load. The PTWINQS composite rule remained the most competitive heuristic, yet was typically overtaken by hierarchical methods as congestion increased. These findings

support three main contributions.

Firstly, it was introduced a hierarchical MARL scheduler in which a high-level manager selects machine operation modes over short commitment windows while low-level agents execute job selection. This design addresses the need to tame schedule nervousness under stochastic arrivals and to concentrate learning on leverageful decision epochs, yielding consistent NAT gains and shorter tails at 70–90% utilization.

Secondly, it was designed and compared two signal families (Chronological-Joint vs. Tardiness-aligned) under Shared and Individual scopes. Additionally, a lightweight reward-boost that scales temporally reconstructed credit for low-level agents. A simple, hyperparameter calibration identifies λ=0.04 as a robust setting that amplifies informative temporal credit without destabilizing learning.

Thirdly, experiments results show that aligning the HLA reward signal with realized tardiness at the machine level (Tard-Ind) most effectively resolves distributed credit assignment in asynchronous settings. This contribution operationalizes the thesis's main goal of extending the work of [8]. Improving on tardiness optimization by allowing agents to adapt to the current job shop state, raising stability and performance as congestion increases.

At a broader level, this work reinforces that size-agnostic representations (MR state), asynchronous credit attribution (chronological joint action and reward shaping), and hierarchical intent are complementary ingredients for DJSSP. They jointly narrow the gap to well-tuned heuristics and, under high congestion, surpass them with better reliability.

6.2 Future Work

Several directions appear promising. A first direction is to develop a progressive training curriculum that systematically sweeps utilization levels, due-date tightness, and routing complexity from easier to harder regimes. Concretely, this would start with lightly loaded systems and loose due dates to stabilize policy learning, then incrementally introduce higher arrival rates, tighter slack, longer routing chains, and more heterogeneous processing times. Evaluation should track learning curves, sample efficiency, and generalization across curriculum stages, with ablation studies isolating the contribution of each curriculum dimension.

A second direction is to replace hand-crafted MR slots with permutation-invariant encoders that better exploit cross-queue structure while retaining size-agnostic inputs and outputs. Set encoders or graph transformers can natively handle variable numbers of jobs and machines, capture pairwise and higher-order interactions (e.g., contention for downstream resources), and maintain equivariance to permutations of job order. Practical design choices include learned positional surrogates for

event chronology, attention masks keyed to machine availability, and hierarchical pooling to summarize queues without losing critical detail. This line should be benchmarked against MR slots on representational fidelity, policy performance, and compute overhead to confirm that expressivity gains justify added complexity.

A third line of work is to investigate counterfactual baselines and value decomposition variants explicitly adapted to asynchronous decision epochs, while preserving the chronological linkage of rewards. For example, counterfactual credit assignment can be re-formulated on event time, with baselines conditioned on the realized local configuration and pending dispatch options at each machine tick. Similarly, value decomposition (e.g., QMIX, VDN, or monotonic relaxations) could be extended with time-stamped mixing networks that aggregate per-agent utilities only when actions are co-temporal or causally coupled.

A fourth opportunity is to expand the HLA's action space and systematically study commitment windows to make explicit the trade-off between agility and schedule stability. Beyond mode selection, high-level controls could include dynamic WIP caps, due-date banding (prioritization tiers), and setup/changeover budgets that constrain reconfiguration frequency. Commitment windows can be tuned to smooth oscillations and reduce schedule nervousness while retaining responsiveness to changes. Adaptive schemes might lengthen windows under steady state and shorten them during disruptions.

A fifth path is to incorporate operational realism, setups, transport times, re-entrant flows, and machine calendars. Setups introduce sequence dependent changeovers that challenge both dispatch and mode decisions. Transport and blocking effects create spatial coupling, while re-entrancy generates cyclical resource contention. Finally, calendars impose non-stationary availability.

A final contribution to the community would be to release seeded dynamic problem suites and containerized stacks to standardize comparisons and reduce environment drift across studies. Problem suites should cover a matrix of utilization, variability, and routing patterns, with fixed random seeds and auditable generators to ensure replicability. The software stack containing containerized simulators, data loggers, and training scripts, should pin dependency versions, expose clean APIs for state and action hooks, and bundle evaluation assessment with pre-registered metrics and reporting templates. Such artifacts would enable robust baselines, facilitate ablations, and make incremental progress measurable and comparable across research groups.

Appendix A

Original Work Divergences

This section lists why the MARL results reported in Chapter 4 did not reach the original performance of the MARL baseline found in [8]. The points below isolate where small but consequential differences may arise for dynamic job shop scheduling (DJSS), and how those differences propagate into the objective of interest.

1. **Codebase version:** Authors of [8] themselves now flag the repository [62] that contains the DJSSP codebase as "vulnerable to job-overstay", causing training instability and failure, and "no longer maintained". Additionally, the environment stack (SimPy, PyTorch/NumPy/Matplotlib) is fragile to version drift. Even small library changes shift event timing and stochastic streams, which breaks reproducibility unless versions, seeds, and determinism are locked down.

2. **RL weigth's initialization:** It determines the agent's very first action preferences and, therefore, the entire trajectory of data it will learn from. In value-based RL with bootstrapping, tiny differences in initial Q-values alter early ϵ-greedy choices, which change what enters replay, which changes gradients, quickly pushing runs with different seeds into different solutions. In DJSS/MARL, this effect is amplified. A slightly different early dispatch at one machine perturbs downstream queues and due-date pressure seen by others, cascading into distinct coordination patterns and final policies.

3. **Environment mismatch:** DJSS is highly sensitive to the exact data-generation protocol. Even modest deviations in due-date generation (tightness/offsets), job-arrival process (distribution and warm-up), routing (route-length distribution), processing-time distribution, and queue tie-breaks or preemption rules can reshape congestion and alter tardiness by wide margins. If the simulators are not able to produce the same distribution on any of these

axes, the attainable absolute numbers shift and gaps appear, even when the setups look "equivalent" on paper.

4. **Decentralized learning amplifies credit-assignment difficulty:** Per-machine agents act asynchronously while tardiness is realized only at job completion, $T_i = \max(\, C_i - D_i, 0)$. Thus, each T_i embeds a long causal chain spanning multiple machines and earlier decisions. This delayed, distributed credit makes value estimation noisier and typically demands stronger critics, explicit counterfactual baselines, or longer training to match the quality achieved under settings that reduce this burden.

5. **Training stability:** Unstable learning magnifies seed-level quirks into large performance swings. If TD targets oscillate, gradients spike, or exploration is erratic, two runs that start almost identically will diverge and yield different NAT, Tardy, and WR statistics. Stabilizers, such as Double Q (and dueling), Huber loss with TD-error clipping, soft target updates, and gradient-norm clipping, make the learning dynamics smoother and less sensitive to initial conditions. However, does not entirely solve it.

6. **Statistical protocol:** Differences in the number of validation seeds/runs, tie-handling for win rate (WR) can shift reported metrics. The proposed protocol emphasizes robustness (e.g., many validation runs across multiple loads), which tends to produce results that are statistically more meaningful.

Bibliography

[1] Michael Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 5th ed. Springer, 2016 (cit. on pp. 1, 16, 19).

[2] Wallace J. Hopp and Mark L. Spearman. *Factory Physics*. 3rd ed. McGraw-Hill, 2011 (cit. on pp. 1, 19).

[3] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. «Complexity of Machine Scheduling Problems». In: *Annals of Discrete Mathematics* 1 (1977), pp. 343–362. DOI: 10.1016/S0167-5060(08)70743-X (cit. on p. 1).

[4] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. «Industry 4.0». In: *Business & Information Systems Engineering* 6.4 (2014), pp. 239–242. DOI: 10.1007/s12599-014-0334-4 (cit. on p. 2).

[5] Djamila Ouelhadj and Sanja Petrovic. «A Survey of Dynamic Scheduling in Manufacturing Systems». In: *Journal of Scheduling* 12.4 (2009), pp. 417–431. DOI: 10.1007/s10951-008-0090-8 (cit. on pp. 2, 14, 16, 17, 20).

[6] G. E. Vieira, J. W. Herrmann, and E. Lin. «Rescheduling: A Framework for Research and Practice». In: *International Journal of Production Research* 41.23 (2003), pp. 5113–5133. DOI: 10.1080/0020754031000163802 3 (cit. on p. 3).

[7] B. Waschneck, A. Reichstaller, L. Belzner, et al. «Optimization of Global Production Scheduling with Deep Reinforcement Learning». In: *Procedia CIRP*. Vol. 72. 2018, pp. 1264–1269. DOI: 10.1016/j.procir.2018.03.212 (cit. on p. 4).

[8] Renke Liu, Rajesh Piplani, and Carlos Toro. «A Deep Multi-Agent Reinforcement Learning Approach to Solve Dynamic Job Shop Scheduling Problem». In: *Computers & Operations Research* 159 (2023), p. 106294. DOI: 10.1016/j.cor.2023.106294 (cit. on pp. 4, 8–10, 12, 17, 20–22, 28, 31, 34, 51, 69, 71).

[9] John H. Blackstone, John A. Phillips, and Glen L. Hogg. «A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations». In: *International Journal of Production Research* 20.1 (1982), pp. 27–45. DOI: 10.1080/00207548208947774 (cit. on pp. 6, 14, 40, 43).

[10] R. Haupt. «A Survey of Priority Rule-Based Scheduling». In: *OR Spektrum* 11.1 (1989), pp. 3–16. DOI: 10.1007/BF01721070 (cit. on p. 6).

[11] Ramaswamy Ramasesh. «Dynamic Job Shop Scheduling: A Survey of Simulation Research». In: *Omega* 18.1 (1990), pp. 43–55. DOI: 10.1016/0305-0483(90)90055-O (cit. on p. 6).

[12] Oliver Holthaus and Chelliah Rajendran. «Efficient Dispatching Rules for Scheduling in a Job Shop». In: *International Journal of Production Economics* 48.1 (1997), pp. 87–105. DOI: 10.1016/S0925-5273(96)00077-8 (cit. on pp. 6, 7, 12, 21, 22, 43).

[13] W. E. Smith. «Various optimizers for single-stage production». In: *Naval Research Logistics Quarterly* 3 (1956), pp. 59–66 (cit. on p. 6).

[14] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling*. Addison-Wesley, 1967 (cit. on p. 6).

[15] J. R. Jackson. «Scheduling a production line to minimize maximum lateness». In: *Management Science* 2 (1955), pp. 45–50 (cit. on p. 6).

[16] Ari P. J. Vepsäläinen and Thomas E. Morton. «Priority Rules for Job Shops with Weighted Tardiness Costs». In: *Management Science* 33.8 (1987), pp. 1035–1047. DOI: 10.1287/mnsc.33.8.1035 (cit. on pp. 6, 12).

[17] Young Hoon Lee, Kumar Bhaskaran, and Michael Pinedo. «A heuristic to minimize the total weighted tardiness with sequence-dependent setups». In: *IIE Transactions* 29.1 (1997), pp. 45–52. DOI: 10.1080/07408179708966311 (cit. on p. 6).

[18] Daniel C. Carroll. «Heuristic Sequencing of Single and Multiple Component Jobs». Ph.D. dissertation. PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 1965. URL: https://dspace.mit.edu/handle/1721.1/112600 (cit. on p. 6).

[19] Christos Koulamas. «The Total Tardiness Problem: Review and Extensions». In: *Operations Research* 42.6 (1994), pp. 1025–1041. DOI: 10.1287/opre.42.6.1025 (cit. on p. 6).

[20] Kenneth R. Baker and John J. Kanet. «Job shop scheduling with modified due dates». In: *Journal of Operations Management* 4.1 (1983), pp. 11–22. DOI: 10.1016/0272-6963(83)90022-0 (cit. on p. 6).

[21] Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and Rong Qu. «Hyper-heuristics: A Survey of the State of the Art». In: *Journal of the Operational Research Society* 64.12 (2013), pp. 1695–1724. DOI: 10.1057/jors.2013.71 (cit. on pp. 7, 14).

[22] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. «Recent Advances in Selection Hyper-heuristics». In: *European Journal of Operational Research* 285.2 (2020), pp. 405–428. DOI: 10.1016/j.ejor.2019.07.073 (cit. on pp. 7, 14).

[23] Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. «Effective and Interpretable Dispatching Rules for Dynamic Job Shops via Guided Empirical Learning». In: *Omega* 111 (2022), p. 102646. DOI: 10.1016/j.omega.2022.102646 (cit. on pp. 7, 8).

[24] Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. «Survey on Genetic Programming and Machine Learning Techniques for Heuristic Design in Job Shop Scheduling». In: *IEEE Transactions on Evolutionary Computation* 28.1 (2024), pp. 147–167. DOI: 10.1109/TEVC.2023.3255246 (cit. on p. 8).

[25] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. «Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020, pp. 1621–1632 (cit. on pp. 8–11, 20, 21).

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. Cambridge, MA: MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html (cit. on p. 8).

[27] Igor G. Smit, Jianan Zhou, Robbert Reijnen, Yaoxin Wu, Jian Chen, Cong Zhang, Zaharah Bukhsh, Yingqian Zhang, and Wim Nuijten. «Graph Neural Networks for Job Shop Scheduling Problems: A Survey». In: *arXiv preprint arXiv:2406.14096* (2024) (cit. on pp. 8, 12, 21).

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. «Prioritized Experience Replay». In: *International Conference on Learning Representations (ICLR)*. 2016 (cit. on pp. 9, 10, 12).

[29] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. «Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping». In: *Proceedings of the 16th International Conference on Machine Learning (ICML)*. 1999, pp. 278–287 (cit. on pp. 9, 10, 13, 14, 40, 41).

[30] Justin K. Terry, Nathaniel Grammel, Sanghyun Son, Benjamin Black, and Aakriti Agrawal. «Revisiting Parameter Sharing in Multi-Agent Deep Reinforcement Learning». In: *arXiv preprint arXiv:2005.13625* (2020) (cit. on p. 9).

[31] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. «Cooperative Multi-Agent Control Using Deep Reinforcement Learning». In: *International Conference on Autonomous Agents and Multiagent Systems (AAMAS) Workshops, Best Papers, Revised Selected Papers*. Springer, 2017, pp. 66–83 (cit. on p. 9).

[32] Hado van Hasselt, Arthur Guez, and David Silver. «Deep Reinforcement Learning with Double Q-Learning». In: *AAAI Conference on Artificial Intelligence*. 2016 (cit. on p. 10).

[33] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. «Dueling Network Architectures for Deep Reinforcement Learning». In: *International Conference on Machine Learning (ICML)*. 2016, pp. 1995–2003 (cit. on p. 10).

[34] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. «Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017 (cit. on pp. 10, 14, 40).

[35] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. «Counterfactual Multi-Agent Policy Gradients». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 2018 (cit. on pp. 10, 14, 40).

[36] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. «Potential-Based Difference Rewards for Multiagent Reinforcement Learning». In: *AAMAS*. 2014, pp. 165–172 (cit. on pp. 10, 13).

[37] Pierre Tassel, Martin Gebser, and Konstantin Schekotihin. *A Reinforcement Learning Environment for Job-Shop Scheduling*. 2021. arXiv: 2104.03760 [cs.LG]. URL: https://arxiv.org/abs/2104.03760 (cit. on p. 11).

[38] Kosmas Alexopoulos, Panagiotis Mavrothalassitis, Emmanouil Bakopoulos, Nikolaos Nikolakis, and Dimitris Mourtzis. «Deep Reinforcement Learning for Selection of Dispatch Rules for Scheduling of Production Systems». In: *Applied Sciences* 15.1 (2025), p. 232. DOI: 10.3390/app15010232. URL: https://www.mdpi.com/2076-3417/15/1/232 (cit. on p. 11).

[39] Runqing Wang, Gang Wang, Jian Sun, Fang Deng, and Jie Chen. «Flexible Job Shop Scheduling via Dual Attention Network Based Reinforcement Learning». In: (2023). arXiv: 2305.05119 [cs.LG]. URL: https://arxiv.org/abs/2305.05119 (cit. on p. 11).

[40] Lanjun Wan, Long Fu, Changyun Li, and Keqin Li. «Flexible Job Shop Scheduling via Deep Reinforcement Learning with Meta-Path-Based Heterogeneous Graph Neural Network». In: *Knowledge-Based Systems* 296 (2024), p. 111940. DOI: 10.1016/j.knosys.2024.111940 (cit. on p. 11).

[41] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. «Deep Sets». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017 (cit. on p. 11).

[42] Young Hoon Lee, Kumar Bhaskaran, and Michael Pinedo. «A Heuristic to Minimize the Total Weighted Tardiness with Sequence-Dependent Setups». In: *IIE Transactions* 29.1 (1997), pp. 45–52. DOI: 10.1080/07408179708966311 (cit. on p. 12).

[43] Richard S. Sutton, Doina Precup, and Satinder Singh. «Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning». In: *Artificial Intelligence* 112.1–2 (1999), pp. 181–211. DOI: 10.1016/S0004-3702(99)00052-1 (cit. on pp. 12, 13, 28, 39, 41).

[44] Pierre-Luc Bacon, Jean Harb, and Doina Precup. «The Option-Critic Architecture». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 2017 (cit. on pp. 13, 39, 41).

[45] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. «Data-Efficient Hierarchical Reinforcement Learning». In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. 2018 (cit. on pp. 13, 39, 41).

[46] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. «FeUdal Networks for Hierarchical Reinforcement Learning». In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 3540–3549 (cit. on pp. 13, 39, 41).

[47] S. S. Panwalkar and Wafik Iskander. «A Survey of Scheduling Rules». In: *Operations Research* 25.1 (1977), pp. 45–61. DOI: 10.1287/opre.25.1.45 (cit. on pp. 14, 40, 43).

[48] Chandrasekharan Rajendran and Oliver Holthaus. «A Comparative Study of Dispatching Rules in Dynamic Flowshops and Jobshops». In: *European Journal of Operational Research* 116.1 (1999), pp. 156–170. DOI: 10.1016/S0377-2217(98)00122-0 (cit. on pp. 14, 40, 43).

[49] Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. «Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods». In: *Journal of Scheduling* 6.1 (2003), pp. 39–62. DOI: 10.1023/A:1022235519958 (cit. on pp. 14, 17).

[50] Andrew G. Barto and Sridhar Mahadevan. «Recent Advances in Hierarchical Reinforcement Learning». In: *Discrete Event Dynamic Systems* 13.4 (2003), pp. 341–379. DOI: 10.1023/A:1025696116075 (cit. on pp. 14, 28, 41, 43).

[51] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. «QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning». In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018 (cit. on pp. 14, 40).

[52] Shu Luo, Xuan Li, and Liang Gao. «Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning». In: *Applied Soft Computing* 91 (2020), p. 106208. DOI: 10.1016/j.asoc.2020.106208 (cit. on p. 17).

[53] Ming Zhang, Yang Lu, Lin Chen, Yifei Chen, and Wenxin Shao. «Dynamic Scheduling Method for Job-Shop Manufacturing Systems by Deep Reinforcement Learning with Proximal Policy Optimization». In: *Sustainability* 14.9 (2022), p. 5177. DOI: 10.3390/su14095177 (cit. on p. 17).

[54] Albert T. Jones and Luis C. Rabelo. «Survey of Job Shop Scheduling Techniques». In: *Encyclopedia of Electrical and Electronics Engineering*. New York: Wiley, 1998 (cit. on p. 17).

[55] Alexander Aschauer, Florian Roetzer, Andreas Steinboeck, and Andreas Kugi. «Efficient Scheduling of a Stochastic No-Wait Job Shop with Controllable Processing Times». In: *Expert Systems with Applications* 162 (2020), p. 113879. DOI: 10.1016/j.eswa.2020.113879 (cit. on p. 17).

[56] A. S. Jain and S. Meeran. «Deterministic Job-Shop Scheduling: Past, Present and Future». In: *European Journal of Operational Research* 113.2 (Mar. 1999), pp. 390–434. DOI: 10.1016/S0377-2217(98)00113-1 (cit. on p. 22).

[57] Thomas G. Dietterich. «Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition». In: *Journal of Artificial Intelligence Research* 13 (2000), pp. 227–303 (cit. on pp. 28, 41, 43).

[58] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Cham: Springer, 2016. ISBN: 978-3-319-28928-1. DOI: 10.1007/978-3-319-28929-8 (cit. on p. 29).

[59] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons, 1994. ISBN: 0-471-57997-X (cit. on p. 39).

[60] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. «Learning Multi-Level Hierarchies with Hindsight». In: *arXiv preprint arXiv:1712.00948* (2019). DOI: 10.48550/arXiv.1712.00948 (cit. on pp. 39, 41).

[61] Andrew Patterson, Samuel Neumann, Martha White, and Adam White. «Empirical Design in Reinforcement Learning». In: *Journal of Machine Learning Research* 25.318 (2024), pp. 1–63. URL: http://jmlr.org/papers/v25/23-0183.html (cit. on p. 49).

[62] Renke Liu. *Deep-MARL-for-Dynamic-JSP*. 2023. URL: https://github.com/RK0731/Deep-MARL-for-Dynamic-JSP (cit. on p. 71).