Toward Just-In-Time Production Scheduling Using Masked Actor-Critic Reinforcement Learning and Colored Timed Petri Nets

1st Sofiene Lassoued, 2nd Diyar Altinses, 3rd Stefan Lier, 4th Andreas Schwung
Automation Technology and Learning Systems
Institute for Sustainable Technologies and Integrated Production Systems
South Westphalia University of Applied Sciences
Soest, Germany
lassoued.sofiene, altinses.diyar, lier.stefan, schwung.andreas@fh-swf.de

***Abstract*—This paper presents a reinforcement learning approach to optimising Just-In-Time production in job shop environments. We model the production system using Colored Timed Petri Nets, capturing complex constraints including resource sharing, operation sequencing, and timing. An Actor-Critic RL algorithm is deployed to learn optimal scheduling policies. We first focus on minimizing the makespan and then target buffer-size reduction, which is essential for JIT adherence. A multi-objective optimisation is introduced via reward scalarization to generate a Pareto front of trade-offs between makespan and buffer usage. Experimental results demonstrate the effectiveness of our approach in achieving JIT goals through interpretable, adaptive scheduling strategies.**

***Index Terms*—Reinforcement Learning, Just-In-Time Production, Job Shop Scheduling, Colored Timed Petri Nets, Actor-Critic, Multi-objective Optimization, Reward Scalarization, Pareto Front.**

I. Introduction

In today's evolving industrial landscape, production scheduling is essential for operational efficiency and competitiveness. Modern manufacturing systems increasingly emphasise waste reduction, targeting idle time, excess inventory, and inefficient resource utilisation. These pressures, driven by competition and rising customer expectations, require scheduling strategies that minimise non-value-adding activities [1].

In response to these challenges, production philosophies such as Just-In-Time (JIT) have gained significant prominence. Originating from the Toyota Production System, JIT advocates producing only what is needed, when it is needed, and in the required quantity. Central to this philosophy is the elimination of waste, particularly overproduction, waiting, unnecessary transportation, over-processing, excess inventory, motion, and defects [2], [3]. Achieving these objectives requires tightly coordinated operations, minimal buffer capacities, and highly synchronised resource allocation.

Achieving the objectives of JIT, particularly minimal work-in-progress and synchronised machine utilisation, fundamentally depends on effective scheduling and resource allocation. In complex manufacturing environments, these decision-making challenges are commonly formulated as a Job Shop Scheduling Problem (JSSP).

The JSSP is known to be NP-hard [4], making the derivation of optimal schedules computationally prohibitive for large-scale instances. Approaches to solving it have ranged from exact methods to heuristic and metaheuristic methods [5], and, more recently, to machine learning methods, primarily RL [6]. Reinforcement Learning offers a promising alternative by enabling agents to learn scheduling policies through interaction with the production environment. Unlike static heuristics, RL-based methods can adapt to changing conditions and optimise multiple performance criteria simultaneously [7]. Recent advances in deep reinforcement learning (DRL) further extend this capability to high-dimensional and complex scheduling problems [8].

Most JSSP optimisation approaches focus on minimising makespan or tardiness, often overlooking inventory-related objectives central to JIT. To address this, we propose a framework that integrates Colored Timed Petri Nets (CTPN) for system modelling and Actor-Critic Reinforcement Learning (AC-RL) for scheduling optimisation. The CTPN captures system constraints, including timing, operation precedence, and resource contention, while the RL agent learns to make dispatching decisions. We first train the agent to minimise makespan, then introduce buffer penalties to guide it toward JIT-compatible schedules. A reward scalarization technique enables multi-objective learning and the generation of a Pareto front, offering decision-makers a spectrum of trade-offs.

Our key contributions can be summarised as follows:

(i) We propose a novel JIT production scheduling framework that integrates production-environment models using CTPNs and AC-RL to enable efficient decision-making.
(ii) We first train an agent to minimise makespan, and then train a second agent to minimise machine buffers, thereby better aligning with Just-In-Time (JIT) principles.
(iii) We propose a reward scalarization technique that enables multi-objective learning and the generation of a Pareto

front, allowing decision-makers to choose among a spectrum of trade-offs.

(iv) We validate the approach by testing it on various JSSP instances and comparing it with the Evolutionary one.

This paper is organised as follows: We review related work in Sec. II. The system model is based on CTPN, followed by a multi-objective optimisation framework with scalarized rewards, as introduced in Sec. III. The results are presented in Sec . IV, and the conclusion is given in Sec. V.

II. Related Work

This section reviews related work on four critical areas central to this paper: Job Shop Scheduling, Reinforcement Learning in Manufacturing, and Multi-Objective Optimisation.

A. Job Shop Scheduling optimisation

The Job Shop Scheduling Problem (JSSP) is a classical combinatorial optimisation problem with practical relevance, but its NP-hard nature makes it computationally challenging. One of the earliest approaches to solving JSSP was the exact methods. For example, [9] evaluated four MIP formulations using commercial solvers (CPLEX, GUROBI, SCIP) and compared MILP to constraint programming. In addition, the branch-and-bound algorithm introduced in [10] remains highly influential in recent works. Nevertheless, due to the NP-hardness of JSSP, exact methods quickly become infeasible for large-scale or real-world instances. Metaheuristic strategies, often inspired by nature, provide a compromise by finding near-optimal solutions within a reasonable computational time. For example, [11] combined ant colony optimisation with tabu search for JSSP, using a decomposition method to further improve solution quality.

However, metaheuristics have inherent limitations: each run may yield a different "best-of-run" solution, which affects repeatability, and they often require manual parameter tuning to achieve competitive results [12].

B. Reinforcement Learning in Manufacturing

Reinforcement learning has proven highly effective in solving the JSSP Problem. In their survey, [13] reported that 85% of RL implementations achieve significant improvements in scheduling performance. Building on this, [14] proposed GraSP-RL, a GNN-based RL framework where the agent is trained on node features extracted from the graph and operates in a decentralized, asynchronous manner across production units. GraSP-RL outperforms classical heuristics and metaheuristics, such as tabu search and genetic algorithms, in minimizing makespan. Similarly, [15] introduced a sequence-to-sequence framework combining attention mechanisms with disjunctive graph embeddings; node2vec extracts graph representations that are processed by an enhanced transformer with multi-head attention to generate scheduling solutions. While GNNs and Transformers effectively encode the JSSP state, RL algorithms, whether value-based, policy-based, or actor-critic, determine how the agent learns to act on these representations. For instance, [16] employs deep actor-critic RL with parallel DDPG in a multi-agent setting, achieving competitive performance on OR Library benchmarks in dynamic environments.

Most reviewed approaches focus on single objectives, such as makespan or due-date-related criteria, while overlooking equally important aspects of real manufacturing systems. In particular, minimising in-process inventory and supporting Just-In-Time production are rarely considered, despite their direct impact on efficiency, costs, and system responsiveness.

C. Multi-Objective Optimization in scheduling

In complex production systems, scheduling decisions must balance multiple conflicting objectives, such as makespan, tardiness, and work-in-progress. Multi-objective optimization methods are generally grouped into four families: a priori methods, which incorporate preferences before optimization; a posteriori methods, which generate the Pareto front for post-selection; interactive methods, which iteratively incorporate decision-maker feedback; and hybrid methods, which combine these approaches [17].

Within this context, evolutionary methods are the most widely used for approximating the Pareto front [18]. Among them, the Non-dominated Sorting Genetic Algorithm (NSGA) [19] is especially prominent, although its performance is highly sensitive to parameters such as fitness sharing. To enhance efficiency and robustness, it was subsequently extended to NSGA-II [20].

Another prominent approach within evolutionary algorithms is decomposition-based methods, such as MOEA/D [21]. These algorithms split a multi-objective problem into scalar subproblems, each associated with a weight vector, and optimize them simultaneously. This ensures broad coverage of the Pareto front and can improve convergence for high-dimensional objectives. However, despite these advantages, decomposition-based methods are offline and inflexible: each set of weights requires a separate run, and the resulting solutions cannot easily adapt to changing conditions or real-time disturbances.

Evolutionary algorithms dominate multi-objective scheduling but require extensive tuning, consume substantial computational resources, and offer limited knowledge retention. Reinforcement learning, by encoding policies or value functions, enables adaptive, reusable decision-making and thus can offer a promising alternative.

Overall, prior research has advanced job shop scheduling through exact optimization, metaheuristics, and reinforcement learning. However, most RL-based approaches emphasize single-objective formulations, primarily makespan or tardiness, while key manufacturing concerns such as buffer management and work-in-progress reduction remain insufficiently addressed. Multi-objective scheduling methods are dominated by evolutionary approaches, though effective, they largely require repeated tuning or re-optimization. To overcome these limitations, this work formulates scheduling as a multi-objective reinforcement learning problem that explicitly incorporates buffer-related objectives.

III. PetriRL for Just in Time Scheduling

A. Problem definition

The Job Shop Scheduling Problem concerns the allocation of a set of n jobs $\mathcal{J} = \{1, \ldots, n\}$ to a set of m machines $\mathcal{M} = \{1, \ldots, m\}$. Each job $j \in \mathcal{J}$ consists of an ordered sequence of operations $O_{j1}, \ldots, O_{j\ell_j}$, where ℓ_j is the number of operations in job j. Each operation O_{jk} must be processed on machine $M_{jk} \in \mathcal{M}$ for a processing time $p_{jk} > 0$.

The objective is to minimize the makespan $C_{\max}$ and the total discrete buffer B_{total},to adhere to just-in-time goals:

$$\min \quad \mathbf{f} = \left(C_{\max},\ B_{\text{total}}\right) \tag{1}$$

$$C_{\max} = \max_{\substack{j \in \mathcal{J} \\ k \le \ell_j}} (S_{jk} + p_{jk}), \quad B_{\text{total}} = \sum_{m \in \mathcal{M}} \sum_{t=0}^{C_{\max}} B_m(t) \tag{2}$$

where $B_m(t)$ is the number of operations waiting in the buffer. subject to JSSP constraints:

$$S_{j,k+1} \ge S_{jk} + p_{jk}, \quad \forall\, j \in \mathcal{J},\ k < \ell_j \tag{a}$$

$$S_{jk} \ge S_{j'k'} + p_{j'k'} \ \text{ or } \ S_{j'k'} \ge S_{jk} + p_{jk}, \quad \forall\, (j,k) \ne (j',k'),\ M_{jk} = M_{j'k'} \tag{b}$$

$$C_{\max} \ge S_{jk} + p_{jk}, \quad \forall\, j \in \mathcal{J},\ k \le \ell_j \tag{c}$$

$$\tag{3}$$

Constraint (a) enforces the operation order within each job. Constraint (b) ensures no two operations overlap on the same machine. Constraint (c) defines $C_{\max}$ as the maximum completion time.

B. Reinforcement learning framework

An RL problem is formalized as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, $\mathcal{P}(s'|s,a)$ the transition dynamics, $\mathcal{R}(s,a,s')$ the reward function, and $\gamma \in [0,1)$ the discount factor. The agent learns a policy $\pi(a|s)$ that maximizes the expected discounted return :

$$\pi^* = \arg\max_{\pi} \mathbb{E}\Big[G_t \,|\, \pi\Big], \quad G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{4}$$

The reinforcement learning framework relies on the continual interaction between the agent and the environment. At each step, the environment provides the agent with an observation of its current state, and the agent selects an action in response. The environment then updates its state and returns feedback as a reward along with the next state observation, repeating this cycle iteratively.

The framework includes a model that simulates the JSSP with AGVs dynamics (see III-C) and a reward function detailed in III-E. and the agent is an actor-critic algorithm with invalid actions masking detailed in III-D.

C. CTPN as JSSP dynamics engine

As detailed in [22] and further extended to include automated guided vehicles (AGVs) in [23], we leverage the modeling capabilities of timed colored Petri nets to simulate the dynamics of the Job Shop Scheduling Problem.

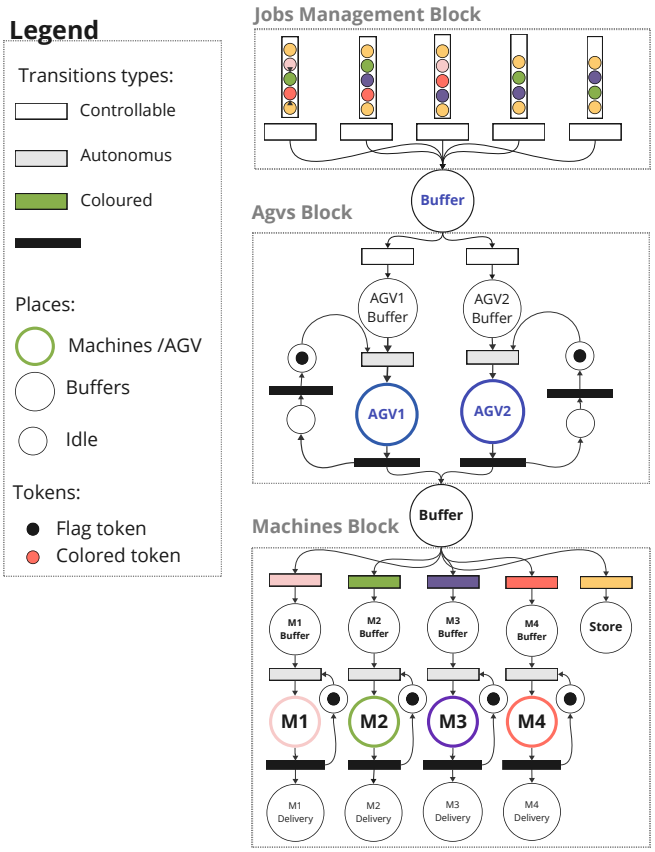

Fig. 1. JSSP environment modeled as Timed Colored Petrinet

In Fig. 1, we depict 5 jobsmachines JSSP model using a CTPN, divided into three blocks: job management, where each job is a place containing an ordered list of colored tokens representing its operations; the handling system, composed of AGVs that transport pieces between machines; and the machine block, where all jobs are processed.

The RL agent acts directly on the Petri net by firing controllable transitions, deciding which jobs to allocate and which AGVs to use. In the example, seven actions are controllable, representing the agent's action space. The remaining process is automatically managed by the Petri net: tokens are routed via colored transitions, and transport and processing times are handled by timed places.

The RL agent observes the state primarily through the token distribution in the places, known as the marking, augmented with token features such as color and remaining processing time. This distribution, together with guard functions, determines whether a transition is enabled. It is then transformed into a Boolean mask that prevents the agent from selecting invalid actions, enforcing system constraints.

D. Masked actor-critic agent

The agent is based on a masked actor-critic algorithm, namely Proximal Policy Optimization (PPO) [24], where the dynamic mask of invalid action is provided by the guard function of the colored timed Petri net. The agent aims to maximize the expected discounted return:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\Big], \tag{5}$$

where τ denotes a trajectory generated by the policy π_θ. The policy gradient is estimated using the advantage-weighted log-probabilities:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s_t,a_t\sim\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\,\hat{A}_t\Big], \tag{6}$$

Invalid actions are filtered using a guard function derived from the Petri net. The guard evaluates the current state and, based on the token distribution, produces a Boolean mask over the action space:

$$\mathbf{g}(s_t) \in \{0,1\}^{|\mathcal{A}|}, \tag{7}$$

where $g_i(s_t) = 1$ if action a_i corresponds to an enabled transition, and $g_i(s_t) = 0$ otherwise. Policy logits $\mathbf{z}_\theta(s_t)$ are masked as

$$\tilde{z}_i(s_t) = \begin{cases} z_i(s_t), & g_i(s_t) = 1, \\ -\infty, & g_i(s_t) = 0, \end{cases} \tag{8}$$

To stabilize learning, PPO introduces a clipped surrogate objective based on the probability ratio between the new and old policies:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{9}$$

The clipped objective is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\hat{A}_t\Big)\Big], \tag{10}$$

which discourages excessively large policy updates. The full PPO loss includes a value function loss and an entropy bonus:

$$L(\theta,\phi) = \mathbb{E}_t\Big[L^{\text{CLIP}}(\theta) - c_1\big(V_\phi(s_t) - R_t\big)^2 + c_2 S[\pi_\theta](s_t)\Big], \tag{11}$$

where R_t denotes the bootstrapped return used for value function training. The parameters (θ,ϕ) are optimized jointly using stochastic gradient descent.

E. Multi-objective reward functions

A multi-objective optimization problem involves the simultaneous optimization of multiple objective functions, it can be expressed as:

$$\min_{x\in\mathcal{X}}\ \mathbf{f}(x) = \min_{x\in\mathcal{X}}\left[f_1(x), f_2(x), \ldots, f_m(x)\right], \tag{12}$$

where x is the decision vector in the feasible set $\mathcal{X}$, and $\mathbf{f}(x)$ denotes the vector of m objective functions.

A solution $x^* \in \mathcal{X}$ is Pareto optimal if there exists no $x \in \mathcal{X}$ such that

$$f_i(x) \le f_i(x^*), \quad \forall i \in \{1,\ldots,m\}, \tag{13}$$

$$f_j(x) < f_j(x^*), \quad \text{for at least one } j. \tag{14}$$

This paper addresses two objectives: minimising the makespan and reducing inventory in front of machines. At each time step, the agent promotes machine utilisation while simultaneously limiting work-in-process by reducing tokens in the machine buffer. Accordingly, two reward functions are defined to capture these objectives.

Let P_m denote the set of places modeling machines, P_{idle} the set of idle places, and P_{buffer} the set of buffer places. Let $n_p(t)$ be the number of tokens in place p at time t. The normalised utilisation reward is defined as

$$R_{\text{utilization}}(t) = -\frac{1}{|P_m|}\sum_{p\in P_{\text{idle}}}\mathbf{1}_{\{n_p(t)>0\}}, \tag{15}$$

where P_{buffer} denotes the set of buffer places in front of machines, and $n_b(t)$ the number of tokens in buffer place $b \in P_{\text{buffer}}$ at time t. The normalized JIT reward is defined as

$$R_{\text{JIT}}(t) = -\frac{1}{|P_m|}\sum_{b\in P_{\text{buffer}}} n_b(t). \tag{16}$$

Various approaches exist in the literature to address the multi-objective nature of optimization problems. In this work, an *a priori* strategy is adopted, specifically a reward scalarization technique. A scalar objective is obtained by forming a weighted sum:

$$R(t) = w_1\ R_{\text{JIT}}(t) + w_2\ R_{\text{utilization}}(t), \tag{17}$$

with $w_1, w_2 \ge 0$ and $w_1 + w_2 = 1$.

By varying w_1 and w_2, different trade-offs between inventory reduction and machine utilization can be explored, enabling the approximation of the Pareto front.

IV. Experiments and Results

A. Experimental Setup

The experiments were conducted on a machine equipped with an NVIDIA Quadro RTX 5000. All models were implemented using the PyTorch deep learning framework and run on Windows 11. We employed a modified version of Maskable PPO from Stable Baselines with a learning rate of 3×10^{-4}, a clip range of 0.2, an entropy coefficient of 0.1, 2048 steps per update, and a fixed seed of 101 for reproducibility. Agents for each instance-size group were trained for 3×10^5 steps. For comparison, we used the NSGA-II algorithm from the Pymoo package with a default population of 10 over 10 generations. Our evaluation was performed on the benchmark instances proposed by [25], which include problem sizes from 5 jobs $\times$ 4 machines up to 8 jobs $\times$ 8 machines. Each set contains four distinct shop floor layouts, influencing the travel distances between machines for the automated guided vehicles in the material handling system.

B. Training the agent

In Fig. 2, we show the training performance on a representative instance from the Bilge benchmark. In the top subfigure (a), we plot the evolution of the episodic mean collected reward. The reward increases monotonically, indicating that the agent is learning to maximize it. In the bottom subfigure (b), we plot the evolution of the total agent loss, which combines the policy loss, the value function loss, and the entropy loss. The initial spikes at the beginning of training reflect the exploration phase, after which the agent transitions to the exploitation phase. The total loss depends on the ability of the critic to predict state values and on the policy network to map states to actions while maximizing the reward.

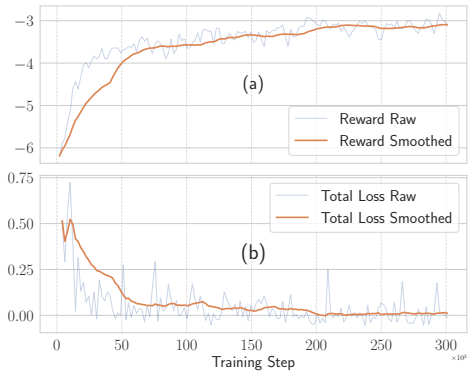

Fig. 2. Training performance of the MPPO agent. (a) Raw and smoothed episode reward. (b) Raw and smoothed total loss.

C. Results

In the results section, we first present the performance obtained for single-objective optimization. The evaluation criteria include makespan, which measures overall productivity, and the average number of tokens in all buffer places at each step, which serves as a metric for inventory reduction. We then identify the non-dominated Pareto-optimal points in a 10-objective mixed-optimization scenario.

TABLE I
COMPARISON OF RL AND EA ACROSS EXTREME OBJECTIVE SETTINGS ON RAJ BENCHMARK INSTANCES

	Obj: Makespan (1.0, 0.0)				**Obj: Buffer (0.0, 1.0)**			
	Makespan		**Buffer**		**Makespan**		**Buffer**	
Inst.	EA	RL	EA	RL	EA	RL	EA	RL
ra01	118	118	0.077	0.210	136	130	0.072	0.050
ra02	126	132	0.169	0.630	137	152	0.135	0.078
ra03	138	123	0.120	0.460	164	233	0.111	0.000
ra04	161	146	0.040	0.030	191	210	0.029	0.000
ra05	109	102	0.050	0.180	119	135	0.041	0.000
ra06	161	141	0.210	0.570	173	183	0.155	0.071
ra07	140	138	0.100	0.420	193	200	0.084	0.045
ra08	192	167	0.260	0.740	221	245	0.226	0.012
ra09	157	141	0.130	0.740	193	218	0.106	0.000
ra10	196	177	0.220	0.780	232	269	0.176	0.000
Avg	149.8	**138.5**	0.138	0.476	182.2	197.5	0.114	**0.026**

Table I compares the results obtained using reinforcement learning (RL) with the Maskable PPO algorithm and the evolutionary algorithm (EA), specifically the genetic-based NSGA-II. Results are reported for optimizing both machine utilization (makespan) and inventory reduction (buffer).

First, we note that the two objectives are conflicting and non-degenerate: optimizing one objective typically comes at the expense of the other. A higher buffer ensures that machines always have pieces ready to process, improving machine utilization. Conversely, if the buffer is too low, machines must wait for the handling system to deliver pieces, increasing idle time and reducing overall productivity.

Second, we evaluated the two extreme objective settings. In the first case, the goal is to minimize makespan. For the RL agent, the reward scalarization mix is $w_1 = 1$ and $w_2 = 0$, representing 100% emphasis on makespan minimization. This is compared with NSGA-II, which also optimizes the single objective of makespan. In the second scenario, the goal is to minimize the buffer, with the reward mix reversed. In both cases, we record the resulting makespan and buffer values.

The results indicate that RL is more capable of adhering to the set scenarios. In the makespan-minimization scenario, RL achieves an average makespan of 138.5 steps with a buffer of 0.476, whereas EA achieves a better buffer of 0.138 but a worse makespan of 149.8 steps. Since the primary objective is makespan minimization, RL outperforms EA in this scenario. Conversely, in the buffer-minimization scenario, RL achieves an average buffer of 0.026 compared to 0.114 for EA, demonstrating that RL better aligns with the target objectives in both extreme cases.

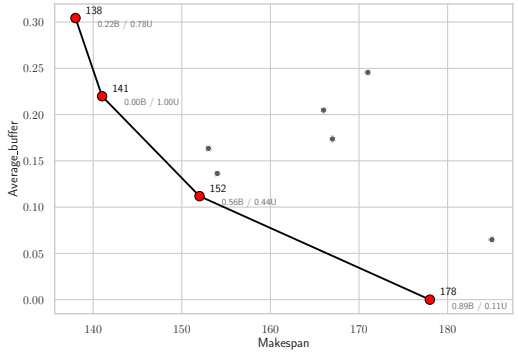

Fig. 3. Pareto front obtained for the EX72 instance (8 jobs, 4 machines). The figure depicts the compromise between minimizing the average buffer level and minimizing the makespan. Each point represents a reward parameterization (w_1, w_2), while the red points identify the Pareto-efficient (non-dominated) solutions.

After optimizing for a single objective, we transition to joint multi-objective optimization. Since we have two objectives, we divide the range $[0, 1]$ between them to create 10 reward mixtures of the form $[w_1, w_2]$. We then train 10 agents using reward scalarization with these weight combinations. After training, each agent is evaluated in our environment to obtain the average buffer and the makespan. These points are plotted

in Fig. 3, where we highlight 4 non-dominated agents out of the 10 mixtures.

The Pareto front illustrates the trade-offs between objectives, with non-dominated points representing the best achievable compromises. While our approach uses predefined weights, it still enables partial a posteriori decision-making by offering multiple agent options.

V. Conclusion

In this paper, we propose a framework that integrates colored timed Petri nets for formal modeling with actor-critic reinforcement learning for multi-objective decision-making. The framework balances productivity through makespan minimization and waste reduction via work-in-progress, aligning with Just-In-Time principles. The Petri net provides interpretable system states and masks invalid actions, while the actor-critic agent retains knowledge through a trained policy for decision-making.

Experimental results demonstrate that RL more effectively adheres to target objectives under extreme scenarios: in makespan-minimization scenarios, RL achieves lower makespan, and in buffer-minimization scenarios, it achieves lower average buffer while maintaining competitive makespan. To generate the Pareto front, we employed reward scalarization to train 10 expert agents with different objective preferences, highlighting non-dominated configurations. While this approach provides the decision maker with pretrained Pareto-optimal points a priori, it also lays the groundwork for a full a posteriori method capable of producing high-quality Pareto front approximations.

Future work includes exploring Mixture-of-Experts approaches, where expert agents vote at inference rather than combining objectives at the reward level. This reduces retraining for new objective combinations and enables near-instant, fine-grained adaptation, useful in dynamic scenarios such as balancing energy use and productivity in real time. Challenges remain in ensuring the policy mixture safely maps decisions to a feasible solution space.

Funding

The research project was funded by "The Ministry of the Environment, Nature Conservation, and Transport of the State of North Rhine-Westphalia in Germany, and co-financed by the European Union for the research project "AI4DRONE", with a grant number IN-NX-1-033b.

References

[1] R. Shah and P. T. Ward, "Lean manufacturing: context, practice bundles, and performance," *Journal of Operations Management*, vol. 21, no. 2, pp. 129–149, 2003.

[2] K. K. Humphreys, *Toyota Production System*. Productivity Press, 2011.

[3] T. Ōno and S. Mito, *Just-in-time for today and tomorrow*. Cambridge Mass.: Productivity Press, 1988.

[4] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, ser. A Series of books in the mathematical sciences. San Francisco: W. H. Freeman, 1979.

[5] M. S. Umam, M. Mustafid, and S. Suryono, "A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 9, pp. 7459–7467, 2022.

[6] X. Zhang and G.-Y. Zhu, "A literature review of reinforcement learning methods applied to job-shop scheduling problems," *Computers & Operations Research*, vol. 175, p. 106929, 2025.

[7] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, p. 1054, 1998.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/pdf/1312.05602

[9] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[10] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete Applied Mathematics*, vol. 49, no. 1-3, pp. 107–127, 1994.

[11] K.-L. Huang and C.-J. Liao, "Ant colony optimization combined with taboo search for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1030–1046, 2008.

[12] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational Intelligence*, ser. Intelligent Systems Reference Library, J. Kacprzyk, L. C. Jain, C. L. Mumford, and L. C. Jain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 1, pp. 177–201.

[13] M. Panzer and B. Bender, "Deep reinforcement learning in production systems: a systematic literature review," *International Journal of Production Research*, vol. 60, no. 13, pp. 4316–4341, 2022.

[14] M. S. A. Hameed and A. Schwung, "Graph neural networks-based scheduler for production planning problems using reinforcement learning," *Journal of Manufacturing Systems*, vol. 69, pp. 91–102, 2023.

[15] R. Chen, W. Li, and H. Yang, "A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1322–1331, 2023.

[16] C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71 752–71 762, 2020.

[17] R. Ojstersek, M. Brezocnik, and B. Buchmeister, "Multi-objective optimization of production scheduling with evolutionary computation: A review," *International Journal of Industrial Engineering Computations*, pp. 359–376, 2020.

[18] J. Para, J. Del Ser, and A. J. Nebro, "Energy-aware multi-objective job shop scheduling optimization with metaheuristics in manufacturing industries: A critical survey, results, and perspectives," *Applied Sciences*, vol. 12, no. 3, p. 1491, 2022.

[19] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[20] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Parallel Problem Solving from Nature PPSN VI*, ser. Lecture Notes in Computer Science, G. Goos, J. Hartmanis, J. van Leeuwen, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, vol. 1917, pp. 849–858.

[21] Y. Qi, X. Li, J. Yu, and Q. Miao, "User-preference based decomposition in moea/d without using an ideal point," *Swarm and Evolutionary Computation*, vol. 44, pp. 597–611, 2019.

[22] S. Lassoued and A. Schwung, "Introducing petrirl: An innovative framework for jssp resolution integrating petri nets and event-based reinforcement learning," *Journal of Manufacturing Systems*, vol. 74, pp. 690–702, 2024.

[23] S. Lassoued, L. S. Baheti, N. Weiß-Borkowski, S. Lier, and A. Schwung, "Flexible manufacturing systems intralogistics: Dynamic optimization of agvs and tool sharing using colored-timed petri nets and actor–critic rl with actions masking," *Journal of Manufacturing Systems*, vol. 82, pp. 405–419, 2025.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[25] Ü. Bilge and G. Ulusoy, "A time window approach to simultaneous scheduling of machines and material handling system in an fms," *Operations Research*, vol. 43, no. 6, pp. 1058–1070, 1995.