MOSA-based Q-Learning for the Unrelated Parallel Machine Scheduling Problem with Maintenance Planning

Meriem TOUAT[1] and Karima BENATCHBA[2] and Lyna-Razane MEGUELLATI[2]

Abstract— **We address the Unrelated Parallel Machine Scheduling Problem (UPMSP), where machines are subject to unavailability periods due to maintenance interventions. Two conflicting objectives are considered: the first focuses on production and maintenance tardiness, while the second aims to minimize energy consumption. This problem was previously studied in [15], where it was formulated as a Mixed-Integer Linear Programming (MILP) model and solved using an enhanced Multi-Objective Simulated Annealing algorithm based on a population of solutions (POP-MOSA).**

In this work, we aim to improve the performance of POP-MOSA by integrating reinforcement learning techniques, specifically the Q-learning algorithm. The resulting approach, referred to as Q-MOSA, leverages Q-learning to adaptively select the most appropriate local search strategy at each iteration.

Q-MOSA introduces several novel contributions. First, at the end of the learning phase, the Q-table is generated using a compact design that reduces its size while preserving decision quality. Second, given the conflicting nature of minimizing both tardiness and energy consumption, a more flexible learning mechanism is required. This motivates the adoption of a multi-reward reinforcement learning strategy, which offers a more effective alternative to traditional single-reward schemes.

Q-MOSA is evaluated using small-scale benchmark instances, and its performance is compared against the exact Pareto front obtained in [15].

I. Introduction

In this paper, we address a realistic problem involving two key business management functions: production scheduling and maintenance planning. Specifically, we focus on the UPMSP, which is one of the three main types of parallel machine shop environments: identical parallel machines [3], uniform parallel machines [13], and unrelated parallel machines [8]. Furthermore, maintenance planning can vary depending on the type of maintenance considered. It may be flexible [14] or fixed [22], depending on predetermined start and end times, and in some cases, it may also be periodic ([4]).

The problem of production scheduling with maintenance planning is usually referenced as the scheduling problem under unavailabilities and several state of arts were proposed ([6]).

To better reflect the realities of production workshops and align with sustainability and environmental concerns, a multi-criteria approach is necessary. Several studies have explored this within the context of parallel machine shop environments [12].

The methods for solving scheduling problems are primarily exact methods and approximate methods, which are mainly realized through metaheuristics [11]. Recently, these methods have been enhanced by techniques from artificial intelligence to improve their performance. We are particularly interested here in reinforcement learning (RL) technique.

[16] provides a review of the applications of reinforcement learning (RL) techniques to various scheduling problems, with particular emphasis on their integration with metaheuristics. In relation to the Q-learning algorithm, five papers are presented, each detailing how Q-learning is applied within the scheduling context. [7] presents a review of 29 publications on the topic of ”RL methods for manufacturing scheduling problems.” Among them, four papers specifically use Q-learning, with some incorporating it into metaheuristic frameworks. [10] presents a state-of-the-art review of 80 papers published between 1995 and 2020, organized according to workshop design. The reviewed learning algorithms include Q-learning. 17% of the studies deal with parallel machine scheduling problems, and 64% use the Q-learning algorithm. In the context of scheduling, RL algorithms are often employed to select dispatching rules. However, only one study considers the UPMSP from a multi-objective perspective.

Since these three state-of-the-art reviews, several recent studies have emerged. [2] tackles the UPMSP with the goal of optimizing three objective functions: total tardiness, makespan, and energy consumption. The Q-learning algorithm is combined with NSGA-II. [21] proposes a deep reinforcement learning approach to solve the Job-Shop Scheduling Problem (JSSP), modeled as a Markov Decision Process. [20] addresses the Distributed Flexible Job-Shop Scheduling Problem with integrated maintenance decisions. The objective is to minimize makespan, total maintenance cost, and total energy consumption. The authors propose a multi-objective evolutionary algorithm in which Q-learning is used to guide the selection of local search operators. [9] focuses on the Hybrid Flow-Shop Scheduling Problem with the aim of minimizing the maximum completion time (makespan) at the final stage. Reinforcement learning is applied to compute high-quality solutions.

In this paper, we deal with the unrelated parallel machine scheduling problem presented in [15]. The unavailability due to maintenance interventions and energy constraints are taken into account, allowing us to minimize two objective functions related to production and maintenance earliness/tardiness, as well as energy consumption. Each machine is subject to a

[1] Computer Science and Digital Society (LIST3N). Université de Technologie de Troyes. 12 rue Marie Curie, CS 42060. Troyes (10300). France meriem.touat@utt.fr

[2] Laboratoire des Méthodes de Conception des Systèmes (LMCS). Ecole nationale Supérieure d'Informatique (ESI). Alger, Oued Smar, 16309, Algérie k_benatchba@esi.dz and kl_meguellati@esi.dz

flexible maintenance activity to be performed within a tolerance interval (TI), repeated at regular intervals. Additionally, energy consumption depends on the characteristics of the machines and the duration of production jobs. We propose a MOSA algorithm enhanced with the Q-learning reinforcement learning technique to improve the results obtained by the classical POP-MOSA presented in [15]. We consider the same benchmarks and use standard performance metrics to evaluate the results.

The originality of this work is directly linked to the solution method in the resolution method, highlighted through three main aspects. First, the use of Q-learning algorithm to select the local search procedure in MOSA algorithm from one iteration to the next. Indeed, to the best of our knowledge, there is no paper that applies the Q-learning paradigm to improve the MOSA metaheuristic, particularly in the context of scheduling problems. Second, the use of a multi-reward approach instead of a single-reward which has not been applied in various contributions in the literature. Third, designing the state representation is essential for modeling the state space. We adopt a grid-based discretization approach, where each objective function is mapped within a square using a specific mapping function.

The remainder of the paper is structured as follows: In Section II, we present the problem under study. Sections III and IV detail the two proposed versions of Q-MOSA, respectively. In Section V, we present and discuss the obtained results. Finally, in Section VI, we conclude the paper and provide some perspectives for future work.

II. Problem definition

We consider the scheduling problem addressed in [15]. Specifically, we focus on the unrelated parallel machine scheduling problem with periodic and flexible maintenance activities. The objective is to schedule N production jobs on M unrelated (non-identical) machines. Each production job J_i, $i = 1..N$ has a processing time p_{ik} that depends on the machine M_k, $k = 1..M$, on which it will be scheduled, and a due date d_i that should not be exceeded. J_i is considered late (i.e., has a tardiness T_i) if its completion time c_i exceeds d_i. The tardiness is computed as follows (Eq. 1):

$$T_i = max(0, c_i - d_i) \qquad i = 1..N \tag{1}$$

Each machine M_k, requires the planning of one maintenance periodic and flexible maintenance activity. We denote each maintenance occurrence as h_{jk}, $j = 1..H_k$. These maintenance activities must be planned periodically, with a fixed interval P_k, and within a time window called the tolerance interval, denoted as $TI_{jk} = [Tm, Tx]_{jk}$. Thus, each maintenance occurrence h_{jk} is associated with its corresponding tolerance interval TI_{jk}, and its timing is determined based on the previous occurrence h_{jk-1}. Moreover, the processing time of each maintenance activity h_{jk} is denoted $p^{'}_{k}$, which is given and depends solely on the machine. Therefore, this duration varies from one machine to another.

Since the maintenance activities are flexible, each h_{jk} can be either advanced or delayed within its tolerance interval TI_{jk}. A maintenance occurrence is considered early if it starts before Tm_{jk} (i.e., its starting time $t^{'}_{jk} < Tm_{jk}$) , and late if it finishes after Tx_{jk} (i.e., its completion time $c^{'}_{jk} > Tx_{jk}$). These earliness and tardiness values are denoted E_{jk} and T_{jk} respectively, and are computed as shown in Eq. 2:

$$\begin{cases} E^{'}_{jk} = max(0, Tm_{jk} - t^{'}_{jk}) & j = 1..H_k \qquad k = 1..M \\ T^{'}_{jk} = max(0, c^{'}_{jk} - Tx_{jk}) & j = 1..H_k \qquad k = 1..M \end{cases} \tag{2}$$

Where H_k represents the maximum number of maintenance interventions on machine M_k.

Each machine consumes energy based on its operational state (active or idle), its specific characteristics, and the jobs being processed. In fact, energy consumption varies depending on whether the machine is active, idle, or undergoing maintenance. We denote by e^{0}_{ij}, the energy consumption per unit of time when job i is processed on machine j. Additionally, e^{1}_{j} represents the energy consumption per unit of time for machine j when it is idle. We formulate this as a bi-objective optimization problem. The first objective function, f_T, concerns time optimization in both production and maintenance contexts. It aims to minimize the total tardiness of production jobs (Eq. 3) as well as the combined earliness and tardiness of maintenance activities (Eq. 3). The second objective focuses on energy efficiency, targeting the minimization of total energy consumption (Eq. 4). A Mixed-Integer Linear Programming (MILP) formulation for this problem was proposed in [15].

$$\begin{cases} f_p = \sum_{i=1}^{N} T_i & i = 1..N \\ f_m = \sum_{j=1}^{M} \sum_{k=1}^{H} (E^{'}_{jk} + T^{'}_{jk}) & j = 1..H_k \qquad k = 1..M \end{cases} \tag{3}$$

$$\begin{gathered} f_E = \sum_{j \in \mathcal{M}} \sum_{i \in \mathcal{J}} \sum_{l \in \mathcal{P}} (X_{ijl} * e^{0}_{i-1j-1} * p_{j-1i-1} + \\ \sum_{j \in \mathcal{M}} ((Cmax_j - A_j) * e^{1}_{j-1}) \end{gathered} \tag{4}$$

III. Q-MOSA

Q-learning is a fundamental reinforcement learning (RL) algorithm [18] that enables an agent to learn how to make optimal decisions by interacting with its environment.

It is a model-free, value-based learning method, meaning that the agent does not require prior knowledge of the environment's dynamics; instead, it learns through trial and error [1]. In the learning phase, the agent explores different actions in a given state and updates its estimates of the optimal action-value function based on the rewards received. Over time, it constructs a Q-table, which maps state-action pairs to expected long-term rewards. This table allows the agent to identify the best action to take in any given state to maximize cumulative rewards.

In the execution phase, the agent uses the learned Q-table to select actions based on the values it has computed.

The proposed Q-MOSA represents an improvement over the one presented in [15]. In addition to the specific features of POP-MOSA, the integration of the Q-learning algorithm is implemented as follows:

- The Q-learning algorithm is used to choose the local search;

- To compute the Q-table in the learning stage, a specific design is proposed allowing us to reduce the Q-table volume and consequently the search time;
- In addition to the single-reward MOSA, a multi-reward version is proposed.

A. Element of the learning algorithm

We start by presenting the Q-learning elements:

1) State Environment: The solution space is finite but extremely large, making it impractical to store or evaluate each solution individually in a tabular format. Therefore, state space discretization is necessary. However, the design of the state representation must balance the trade-off between information loss and the size of the state space. In this approach, the state representation captures relevant information from the scheduling problem [17], while also modeling the solution independently of the number of machines and tasks. This is achieved by basing the representation solely on the objectives of the solution.

To model the state of a given solution within a set of solutions S, we consider the values of its two objective functions: f_T and f_E. A mapping function takes these two values as input along with the maximum and minimum values across the solution set S, which represent the extreme cases and maps them to a discrete state within a fixed-size grid. This is done by normalizing each objective value within its respective range, followed by scaling it to a small number of discrete levels. The values are then clipped to ensure they remain within predefined limits.

This approach allows us to categorize potentially infinite schedules into a finite set of zones, thereby simplifying the state space and making it more manageable for the Q-learning algorithm to process and optimize scheduling decisions.

$$\begin{aligned} x &= [\tfrac{(f_T - T_m)\times\gamma}{T_m - T_x}] \\ y &= [\tfrac{(f_E - E_m)\times\gamma}{E_m - E_x}] \end{aligned} \tag{5}$$

- (x, y) present the index of a zone (x, y) in the discretized grid.
- T_m, T_x, E_m and E_x present the minimum and maximum values of the tardiness and energy objective functions, respectively.
- T and E present the indexes of the cell associated to the the current solution.
- γ defines the number of states, corresponding to the number of cells in the grid, and is determined by the discretization factor γ. The grid represents a discretized solution space, where each cell (or zone) corresponds to a distinct state.
- The main grid has $\gamma \times \gamma$ cells, and additional zones are included beyond the main grid to handle extreme cases, resulting in a $\gamma + 1 \times \gamma + 1$ total grid size.

2) Actions space: To model the action space, the six defined types of neighborhoods are considered as actions. Each action consists of selecting one neighborhood type and performing a random perturbation within it.

3) Reward function definition: The reward function (R) evaluates the improvement achieved by an action by considering the differences in energy consumption (Δf_E) and total delay (Δf_T) before and after the action, as well as the time required to perform the action (CPU), thereby favoring faster perturbations. The reward function is calculated as follows:

$$R = \frac{max(0,\alpha.(\Delta f_T) + max(0,\beta.(\Delta f_E)}{CPU} \tag{6}$$

A perturbation performed within a neighborhood may result in a solution that is worse than the current one. To avoid accounting for negative improvements, any negative difference will be set to a minimum of zero.

4) Q-learning table: The Q-table is defined as a matrix Q[S, A], where :

- S represents the space of states. Each state corresponds to the index of a zone (x, y) in the discretized grid, which represents different scheduling solutions.
- A represents the space of actions, which are perturbations applied to scheduling solutions.

The rows representing the states correspond to the zones calculated previously, while the columns represent the six possible actions. For example, if γ = 9, then the state space consists of $(9 + 1)^2 = 100$ possible states. Each state is indexed as a Cartesian coordinate (x,y) in the grid (zone). The Q-table is thus a matrix of size $S \times A = 100 \times 6 = 600$ values to learn. Each cell in this matrix, Q[s, a], represents the estimated value of taking action aa while in state S.

B. Q-MOSA - Learning

The learning phase in Q-learning is where the agent explores various scheduling solutions, applies perturbations (actions), receives rewards, and updates the Q-table based on the outcomes. The goal is for the agent to learn an optimal strategy for selecting the most effective perturbations to improve scheduling performance over time. We execute the learning phase on a benchmark instance with the highest number of jobs and machines, as this instance allows for the exact computation of the Pareto front. Algorithm 1 illustrates this phase of the Q-MOSA framework.

For the current state$s$, the agent selects an action a based on the $\epsilon - greedy$ policy: With probability p, the agent selects a random action (exploration). Otherwise, it selects the action with the highest Q-value (exploitation): $a = argmax_a Q(s, a)$.

In this case, we use a value of$\epsilon = 1$ in the $\epsilon - greedy$ policy, which means that all actions are chosen randomly, regardless of the state of the solution. This approach ensures that all the cells of the Q-table are visited. During the initial iterations, exploitation (choosing the best action) has no real meaning, as the Q-values are not yet informative. Therefore, the agent is intentionally forced to explore in the early stages.

C. Q-MOSA - Exploitation

After the learning phase, the agent uses the Q-table to choose the optimal action at each step, but through tests many limitations were observed with direct exploitation, and

Algorithm 1 Q-MOSA - Learning.

```
Input: Benchmark, Env_Size, Action_Size;
Output: Q[s, a];
1: $S \longleftarrow get\_env(Env\_Size)$;
2: $A \longleftarrow get\_actions(Action\_Size)$;
3: $Q[s,a] \longleftarrow 0$ for all pairs (s, a);    ▷ Initialize Q-table
4: for (Each Instance in Benchmark) do
5:     $Sol \longleftarrow Generate\_Random\_Solutions(POP)$;    ▷ Generate Pop random solutions
6:     repeat    ▷ Start of the episode
7:         for Solution in Solutions do
8:             $s \longleftarrow get\_state(Solution)$;    ▷ Neighborhood selection for perturbation
9:             $a \longleftarrow \epsilon - greedy(s)$;
10:            $Q[s,a] \longleftarrow Q[s,a] + \alpha \times (R + \gamma \times max_a Q[s^{'}, a^{'}] - Q[s,a])$;    ▷ $\alpha \in [0,1]$ is the learning rate, $\gamma \in [0,1]$ is the discount factor
11:        end for
12:    until Stopping Criterion
13: end for
14: Return $Q - table$;
```

the obtained results were clearly inferior to those of a standard MOSA.

Modified $\epsilon - greedy$: Directly selecting the action with the highest Q-value reduces the diversity of explored actions and can lead the agent to become trapped in local optima, limiting convergence to the global optimum. The Q-table approach also discretizes the state space, and if some states or actions are underexplored, their Q-values may be poorly estimated. Tests showed that a strictly exploitative policy biases the search toward specific regions of the Pareto front. Therefore, maintaining a small degree of exploration helps preserve solution diversity and improves overall performance.

To solve this problem, a variability in the choice of neighborhood types used for perturbations. A modified $\epsilon - greedy$ policy, which takes parameters ϵ_1 and ϵ_2, such that $\epsilon_1 > \epsilon_2$ and $\epsilon_1 + \epsilon_2 < 1$:

- In $\epsilon_1\%$ of the cases, we consider the best neighborhood, i.e., the highest value in the Q-table for the given state.
- In $\epsilon_2\%$ of the cases, we select the second-best neighborhood, i.e., the second-highest value in the Q-table.
- In the remaining cases, we choose a random neighborhood among the four remaining ones and perform the perturbation.

IV. Multi-reward Q-MOSA approach

In traditional RL approaches applied to scheduling problems, it is common to combine multiple objectives into a single reward using fixed manual weights, as in the current implementation with fixed coefficients α and β. However, this single-reward strategy has major limitations. It assumes a static trade-off between objectives throughout learning, which is unrealistic in dynamic and uncertain environments. Moreover, it obscures the individual contribution of each objective, reducing the model's ability to adapt to shifting priorities. In contrast, using multiple distinct reward signals allows the agent to capture different aspects of the desired behavior and combine them adaptively during training. This improves robustness, as deficiencies in one reward signal can be compensated by others, making the system less sensitive to poorly designed reward functions [5].

This observation motivates the use of multi-reward reinforcement learning as a more effective alternative to traditional single-reward techniques. We integrate a multi-reward Q-learning approach into the scheduling system, where the agent maintains three independent Q-tables instead of collapsing multiple objectives into a single scalar reward.

At each learning step, the Q-tables for delay and energy are updated separately using standard Q-learning updates. A combined Q-table is then computed dynamically, with weights for delay and energy adapted based on the relative magnitudes of the learned Q-values. This dynamic weighting enables the agent to adjust its prioritization between objectives in real time, allowing for more effective action selection at each step while the rest of the process follows the same structure as the single-reward approach.

By tracking each objective independently and dynamically adapting the trade-offs, the agent is able to generate more Pareto-efficient scheduling policies, eliminate the need for manual weight tuning, and better adapt to changing system requirements compared to fixed-weight single-reward formulations.

A. Learning Phase

Unlike classical RL approaches that maintain a single Q-table, here the agent maintains **three separate Q-tables** throughout the learning process:

- A **Q-table for tardiness** Q_{T} tracking the expected improvement in delay associated with each action.
- A **Q-table for energy** Q_{E} tracking the expected improvement in energy consumption for each action.
- A **combined Q-table** Q_{combined} which dynamically aggregates the two previous tables to guide action selection.

The learning updates are applied only on the delay and energy Q-tables, following the classical Q-learning update rule. These updates are performed independently for each objective using the corresponding reward signal. The combined Q-table is not updated through a Q-learning rule, instead, it is recalculated dynamically after each learning step based on the current values of Q_{T} and Q_{E} and the according weights.

Once both Q_{T} and Q_{E} have been updated, the combined Q-value is computed using dynamic weights based on the relative magnitudes of the two Q-values, they allow the agent to calculate how big the Q-values are for each objective and uses that to decide which one to prioritize at each step ([23]) :

$$w_{\mathrm{T}} = \frac{|Q_{\mathrm{T}}(s,a)|}{|Q_{\mathrm{T}}(s,a)| + |Q_{\mathrm{E}}(s,a)| + \epsilon}$$

$$\text{and} \quad w_{\mathrm{E}} = \frac{|Q_{\mathrm{E}}(s,a)|}{|Q_{\mathrm{T}}(s,a)| + |Q_{\mathrm{E}}(s,a)| + \epsilon}$$

Then, the combined Q-value is computed as:

$$Q_{\text{combined}}(s,a) = w_{\mathrm{T}} \times Q_{\mathrm{T}}(s,a) + w_{\mathrm{E}} \times Q_{\mathrm{E}}(s,a)$$

By the end of the learning phase, only the combined Q-table Q_{combined} is retained for exploitation, as it represents the agent's learned dynamic trade-off between minimizing delay and minimizing energy consumption. the process is described in the following algorithm:

B. *Exploitation phase*

At each learning step, the agent updates the Q-tables for delay and energy separately using classical Q-learning updates. The combined Q-table is then computed dynamically, where the weights for delay and energy are adapted based on the relative magnitudes of the learned Q-values. This dynamic weighting allows the agent to adjust its prioritization between delay minimization and energy saving according to the evolving learning context.

V. Experimental results

As in [15], we implement the Q-MOSA and Q-MOSA-multi-reward in Python3. Moreover, we use the same benchmarks. We consider 5 instances of the three benchmarks: 10_2, 20_2 and 30_2.

To validate the proposed Q-MOSA, three metrics are used to measure the efficiency of each method. The first one is the Ziztler metric (Z) [25], which determines the proportion of dominated solutions compared to the Pareto front.

The second metric is the average distance (D) [19].

The third metric is the hypervolume metric denoted (H) [24], and based on the eliminated area proportion related to the total area. To compare two approximated fronts B and C relative to a reference front A, we proceed as follows:

- Compute the hypervolumes associated with fronts A, B and C denoted H_A, H_B and H_C, respectively.
- Calculate the ratios $\frac{H_B}{H_A}$ and $\frac{H_C}{H_A}$.
- A ratio greater than 1 indicates that front B is better than A, while a ratio less than 1 indicates the opposite. Additionally, we consider the front with the largest hypervolume to be superior, as it covers the largest area relative to the exact Pareto front.

The execution time, representing the computational efficiency of the algorithm.

These methods provide an objective and direct way to assess the relative quality of Pareto fronts based on their coverage of the objective space.

In Tables I and I, we present the obtained results. According to the metrics, we remark:

- In terms of execution time, Q-MOSA with a multi-reward structure achieves better results in approximately 80% of the cases, with a significant advantage on large-scale benchmark instances. This improvement can be attributed to faster convergence resulting from the use of separate Q-tables.
- In terms of the Zitzler metric, no clear trend is observed, as all three methods exhibit similar performance. In 20% of the cases, the performance is identical across methods. However, in another 20%, Q-MOSA with multi-rewards performs better, while in 40%, the classical MOSA achieves the highest number of exact solutions. Nevertheless, the Zitzler metric alone is insufficient to fully evaluate the effectiveness of the approach, since exact solutions are rarely found. Instead, a significant number of near-optimal solutions are often identified, which justifies the inclusion of additional performance metrics.
- Regarding the distance metric, Q-MOSA with multi-rewards outperforms classical MOSA in approximately 50% of the cases, while in 20%, both methods yield identical distance values.
- The hypervolume metric is particularly important, as it reflects how closely the approximated Pareto front matches the true Pareto front. Based on this metric, Q-MOSA with multi-rewards consistently delivers the best performance in nearly all test cases.

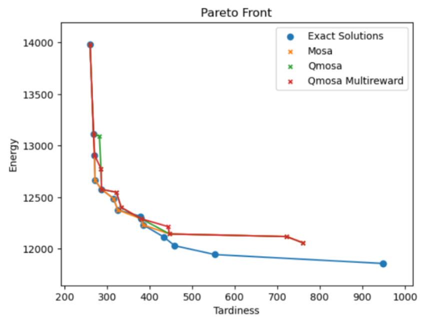

Fig. 1: Benchmark 20_2_2_1.

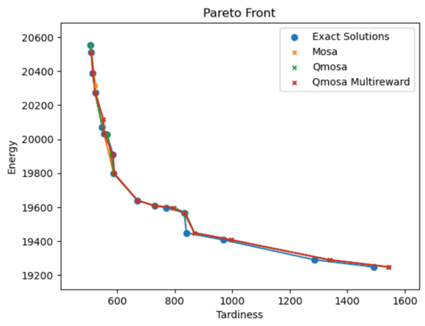

Fig. 2: Benchmark 20_2_2_3.

We confirm the previous observations through the graphs in Figures 1 and 2. In Figure 1, which represents the benchmark instance 20_2_2_1, we observe that although MOSA achieves the smallest distance to the exact front, the hypervolume is relatively large. This indicates that the method covers a wide area of the Pareto front, particularly by exploring extreme solutions located at the edges of the front.

In Figure 2, the Q-MOSA with multi-rewards shows a graph that is closest to the exact front and covers the largest area, followed by Q-MOSA, and finally the classical MOSA. These visual results are consistent with the numerical results presented in Tables I and II.

VI. Conclusion and future works

We address the Unrelated Parallel Machine Scheduling Problem (UPMSP) with flexible and periodic maintenance interventions. Two objective functions are considered: the first aims to minimize earliness and tardiness, while the second focuses on minimizing energy consumption. This problem was previously studied in [15], where a Mixed-Integer Linear

TABLE I: Computational results (Part 1: Solutions, Time, Zitzler).

Bench	Instance	Solutions			Time			Zitzler		
		S1	Q	QM	S1	Q	QM	S1	Q	QM
10_2	1	3	3	3	10.586	12.029	21.167	0.25	0.25	0.25
	2	1	1	1	10.028	9.698	8.121	0.75	0.75	0.971
	3	0	0	0	2.996	1.043	0.538	1.00	1.00	1.00
	4	1	1	1	12.260	9.174	6.131	0.75	0.666	0.5
	5	0	0	0	19.744	33.417	17.709	0.0	1.0	1.0
20_2	1	9	4	5	553.909	166.725	119.727	0.307	0.666	0.583
	2	3	2	2	42.234	41.157	30.191	0.5	0.714	0.75
	3	2	1	8	251.755	404.712	290.947	0.846	0.909	0.466
	4	10	8	10	425.681	367.566	282.612	0.389	0.421	0.477
	5	6	6	6	34.711	45.114	23.839	0.333	0.333	0.143
30_2	1	13	7	8	242.575	697.646	183.477	0.187	0.562	0.5
	2	1	4	3	178.412	304.200	147.046	0.941	0.666	0.8
	3	3	1	2	115.941	127.957	75.209	0.75	0.909	0.818
	4	5	11	5	355.722	649.875	471.974	0.782	0.607	0.821
	5	3	7	4	1568.908	994.135	550.143	0.893	0.741	0.857

TABLE II: Computational results (Part 2: Average distance, Hypervolume).

Benchmark	Instance	Average distance			Hypervolume		
		S1	Q	QM	S1	Q	QM
10_2	1	0.248	0.248	0.248	0.859	0.859	0.859
	2	0.971	0.971	0.971	0.479	0.285	0.284
	3	1.090	1.090	1.090	0.0	0.0	0.0
	4	0.310	0.302	0.301	0.788	0.776	0.761
	5	0.0	2.392	2.392	0.0	0.618	0.718
20_2	1	0.0568	0.073	0.068	0.932	0.931	0.934
	2	0.097	0.0804	0.073	0.475	0.685	0.859
	3	0.0441	0.036	0.015	0.790	0.858	0.987
	4	0.041	0.035	0.035	0.791	0.898	0.913
	5	0.003	0.057	0.071	0.836	0.860	0.984
30_2	1	0.0152	0.021	0.0216	0.980	0.955	0.995
	2	0.066	0.0468	0.028	0.860	0.888	0.986
	3	0.107	0.094	0.065	0.744	0.803	0.977
	4	0.037	0.036	0.016	0.897	0.808	0.984
	5	0.027	0.013	0.036	0.836	0.935	0.964

Programming (MILP) model and a population-based Multi-Objective Simulated Annealing (POP-MOSA) algorithm were proposed.

In the present work, we enhance the MOSA algorithm by integrating Q-learning reinforcement learning techniques. Two variants are developed based on the reward formulation: a single-reward version and a multi-reward version. Additionally, we introduce a specific Q-table design that significantly reduces the table size. This is achieved through a mapping function that projects Pareto-optimal solutions into a reduced representation space—constituting a novel contribution of this work.

Experimental results show a notable improvement in execution time, while improvements in other performance metrics are more moderate.

As future work, we plan to extend the proposed method to larger benchmark instances and to develop a fully reinforcement learning-based approach, moving beyond the current metaheuristic RL hybrid framework.

References

[1] Bilal H Abed-alguni and Mohammad Ashraf Ottom. Double delayed q-learning. *International Journal of Artificial Intelligence*, 16(2):41–59, 2018.

[2] Christian Perez Bernal, Miguel A Salido, and Carlos March Moya. Optimizing energy efficiency in unrelated parallel machine scheduling problem through reinforcement learning. *Information Sciences*, 693:121674, 2025.

[3] I-A. Chaudhry and I-A. Elbadawi. Minimisation of total tardiness for identical parallel machine scheduling using genetic algorithm. *Sādhanā*, 42:11–21, 2017.

[4] A. Costa, F-A. Cappadonna, and S. Fichera. Total tardiness minimization in a parallel machine system with flexible periodic maintenance. *Journal of Industrial and Production Engineering*, https://doi.org/10.1080/21681015.2015.1136898, 2015.

[5] Christoph Dann, Yishay Mansour, and Mehryar Mohri. Reinforcement learning can be more efficient with multiple rewards. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 7147–7175, Honolulu, Hawaii, USA, 2023. PMLR.

[6] M. Durasevic and D. Jakobovic. Heuristic and metaheuristic methods for the unrelated machines scheduling problem: A survey. *arXiv preprint arXiv*, 13106, 2021.

[7] Yaping Fu, Yifeng Wang, Kaizhou Gao, and Min Huang. Review on ensemble meta-heuristics and reinforcement learning for manufacturing scheduling problems. *Computers and Electrical Engineering*, 120:109780, 2024.

[8] W. Haibo and B. Alidaee. Effective heuristic for large-scale unrelated parallel machines scheduling problems. *Omega*, 83:261–274, 2019.

[9] Wei Han, Fang Guo, and Xichao Su. A reinforcement learning method for a hybrid flow-shop scheduling problem. *Algorithms*, 12(11):222, 2019.

[10] Behice Meltem Kayhan and Gokalp Yildiz. Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. *Journal of Intelligent Manufacturing*, 34(3):905–929, 2023.

[11] W-C. Lee, J-Y. Wang, and L-Y. Lee. A hybrid genetic algorithm for an identical parallel machine problem with maintenance activity. *Journal of the Operational Research Society*, page doi:10.1057/jors.2015.19, 2015.

[12] Deming Lei and Hai Yang. Scheduling unrelated parallel machines with preventive maintenance and setup time: Multi-sub-colony artificial bee colony. *Applied Soft Computing*, 125:109154, 2022.

[13] Kai Li, Jianfu Chen, Hong Fu, Zhaohong Jia, and Weizhong Fu. Uniform parallel machine scheduling with fuzzy processing times under resource consumption constraint. *Applied Soft Computing*, 82:105585, 2019.

[14] Ling-Huey Su and Hsiao-Ling Tsai. Flexible preventive maintenance planning for two parallel machines problem to minimize makespan. *Journal of Quality in Maintenance Engineering*, 16(3):288–302, 2010.

[15] Meriem Touat, Karima Benatchba, Annis-Sahnoune Haned, and Mohammad-Mohsen Aghelinejad. Bi-objective unrelated parallel machine scheduling problem under availability and energy constraints. In *2024 10th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1005–1010. IEEE, 2024.

[16] Ling Wang, Zixiao Pan, and Jingjing Wang. A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex System Modeling and Simulation*, 1(4):257–270, 2021.

[17] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(4):5064–5078, 2022.

[18] Christopher John Cornish Hellaby Watkins et al. Learning from delayed rewards. 1989.

[19] Xueqi Wu and Ada Che. A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. *Omega*, 82:155–165, 2019.

[20] Qi Yan, Hongfeng Wang, and Shengxiang Yang. A learning-assisted bi-population evolutionary algorithm for distributed flexible job-shop scheduling with maintenance decisions. *IEEE Transactions on Evolutionary Computation*, 2024.

[21] Erdong Yuan, Shuli Cheng, Liejun Wang, Shiji Song, and Fang Wu. Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143:110436, 2023.

[22] Chuanli Zhao, Min Ji, and Hengyong Tang. Parallel-machine scheduling with an availability constraint. *Computers & Industrial Engineering*, 61(3):778–781, 2011.

[23] Fuqing Zhao, Zewu Geng, Jianlin Zhang, Tianpeng Xu, and Jonrinaldi Jonrinaldi. A q-learning-based multi-objective hyper-heuristic algorithm with dynamic weight adjustment. *Expert Systems with Applications*, XX.

[24] Eckart Zitzler and Simon Künzli. Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*, pages 832–842. Springer, 2004.

[25] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.