



# Solving a Job Shop Scheduling Problem Using Q-Learning Algorithm

Manal Abir Belmamoune, Latéfa Ghomri, Zakaria Yahouni

## ► To cite this version:

Manal Abir Belmamoune, Latéfa Ghomri, Zakaria Yahouni. Solving a Job Shop Scheduling Problem Using Q-Learning Algorithm. SOHOMA 2022 12th international workshop on service oriented, holonic and multi-agent manufacturing systems for industry of the future, Sep 2022, BUCHAREST, Romania. pp.196-209, 10.1007/978-3-031-24291-5\_16 . hal-03951519

**HAL Id: hal-03951519**

**<https://hal.science/hal-03951519v1>**

Submitted on 23 Jan 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Solving a Job Shop Scheduling Problem Using Q-Learning Algorithm

Manal Abir BELMAMOUNE<sup>a</sup>, Latéfa GHOMRI<sup>a</sup>, Zakaria YAHOUNI<sup>b</sup>

<sup>a</sup> Manufacturing Engineering Laboratory of Tlemcen (MELT), University of Tlemcen, Algeria

<sup>b</sup> Univ. Grenoble Alpes, CNRS, Grenoble INP G-SCOP, 38000 Grenoble, France

<sup>a</sup>[manalabir.belmamoune, latefa.ghomri]@univ-tlemcen.dz

<sup>b</sup>zakaria.yahouni@grenoble-inp.fr

**Abstract.** Job Shop Scheduling Problem (JSSP) is among the combinatorial optimization and Non-Deterministic Polynomial-time (NP) problems. Researchers have contributed in this area using several methods, among the methods we have machine learning algorithms, more precisely Reinforcement Learning (RL). The reason why the scientists resort to RL is the adequacy of the algorithm for this type of problem. The results of the RL approach tend toward optimal or near-optimal solutions. In this paper, we deal with the JSSP, using the RL algorithm, more specifically a Q-learning algorithm. We propose a new representation of the state of the environment. We introduce two evaluations of the agent using two different methods. The actions selected by the agent are the dispatching rules. Finally, we compared the results obtained by the approach with the literature.

**Keywords:** Job Shop Scheduling Problem, Reinforcement Learning, dispatching rules, Q-learning.

## Introduction

Production planning ensures that the company can develop, manufacture and finalize products efficiently and within predefined deadlines. Scheduling is an important step when planning a production process. By using production scheduling, manufacturing companies can allocate tasks to needed resources efficiently.

Job Shop Scheduling Problems (JSSP) are currently one of the most relevant issues in manufacturing, especially for systems that require a high degree of flexibility to meet customer needs. As manufacturers adapt to this demand, they will make changes to increase system flexibility, which will inevitably increase the complexity of the overall system [1]. JSSP are combinatorial optimization problems and Non-Deterministic Polynomial-time (NP) hard problems. One of the resolution methods is the use of dispatching rules due to their simplicity and ease of use such as Shortest Processing Time (SPT), First in First Out (FIFO), Longest Processing Time (LPT), Last in First Out (LIFO), and Earliest Due Date (EDD). The problem is that dispatching rules may be much more appropriate in one problem and not in another one. Due to the large-scale combinatorial optimization problems, it is complicated to use exact methods, which is the reason why the researchers move to the solutions related to Artificial Intelligence. Generally, they use the meta-heuristics such as Genetic Algorithms (GA) [2], or the Machine Learning

methods such as Supervised Learning [3], Unsupervised Learning [4], and Reinforcement Learning [5]

In this paper, we are interested in JSSP resolution using Machine learning algorithms, in particular, Reinforcement Learning (RL). RL addresses how autonomous agents learn to choose appropriate actions to achieve their goals by interacting with the environment [6]. The reason why we choose this type of machine learning is the dynamic environment that makes the agent in a situation of taking a real-time decision. There are many algorithms in RL, such as SARSA [7], actor critic algorithm [8], and Q-learning algorithm [9]. In our research, we have used the Q-learning algorithm (QL). QL is a model-free RL algorithm in which the agent learns from real-world experience examples from the environment and never uses the generated next-state predictions.

We propose a new state representation, which helps the agent to take decisions depending on a real situation. We have equipped an agent, which selects in each situation the most appropriate dispatching rule according to which it will schedule the jobs. We have evaluated the agent twice, the 1<sup>st</sup> one is during the jobs assignment, and the 2<sup>nd</sup> is after obtaining the makespan (maximum completion time of jobs also called  $C_{max}$ ). The strategy of exploiting and exploring the environment accelerates the convergence of the agent into a good solution and maximizes the rewards of all the decisions driven by this solution. We compared the results we obtained with the results of the dispatching rules (FIFO, LIFO, SPT, and LPT), the meta-heuristics GA [10], Hybrid GA [11], and Deep reinforcement learning (DRL) [12].

This paper is organized as follows: Section 1: works related to JSSP, and the contribution of some authors using the RL algorithms. Section 2: a description of the JSSP. Section 3: background about the RL algorithm we used. Section 4: the implementation of the algorithm on the JSSP. Section 5: the results we obtained as well as a comparison and a discussion. Section 6: a conclusion and perspectives.

## 1 Related works

In recent years, several papers related to JSSP use RL algorithms, with a difference that relies on the chosen RL algorithm, or the objective function as well as the state representation and the reward function. If we take the  $C_{max}$  as the objective function, it was addressed in [13] [14] [15], they used the QL algorithm, and the latter is one of the most prominent algorithms used in the JSSP. Wang et al. [12] proposed a dynamic scheduling method based on deep reinforcement learning (DRL). They adopted the proximal policy optimization (PPO) to find the optimal policy for the scheduling to handle the large scale of the state, to represent the state, they used three matrices, a machine matrix, a processing time matrix for each operation in jobs, which do not change over time and a job processing status matrix. The function of the reward is related to the utilization of the machine and the obtained makespan. Liu et al. [8] proposed deep reinforcement learning to deal effectively with the Job shop scheduling problem. The proposed model comprises an actor network and a critic network. They represent the state of the environment in three matrices, a process time matrix, a boolean matrix of the assigned job to each machine and a boolean matrix of the completed job, and the reward in three different functions, process time of the selected job, remaining process time of the job and the comparison of the smallest makespan.

Jiménez et al. [16] developed a tool for the Job Shop scheduling, which works with the Q-learning algorithm and can be adapted to different scheduling scenarios such as Flexible JSSP or and, Parallel machines problems. Zhou et al. [17] used a DRL, they build two networks, the prediction and target network. Waschneck et al. [18] proposed cooperative DQN agents, which utilize deep neural networks trained with user-defined objectives to optimize scheduling. They applied their study in a small factory simulation of an abstracted frontend-of-line semiconductor production facility. Tassel et al. [19] represented a DRL environment for Job-Shop Scheduling, they make a meaningful state representation using a matrix, each row describes information about the job such as the allocation of the number of operations rests or the competed ones. They designed a dense reward function based on the scheduled area. After each action, they computed the difference between the duration of the allocated operations and the idle time of a machine. Samsonov et al. [20] proposed a new form of reward and designed a new space for action, they represent the state in six characteristics: machine states, the sum of all operations' processing times currently in the queue of each machine, the sum of all operations' processing times for each job, the duration of the next operation for each job, the index of the next required machine for each job, and finally the time already passed in any given moment. For the reward, the agent is evaluated at the end of the episode.

All authors who contribute to this field applied different approaches for defining the states and evaluating the agent's actions. They choose to represent the state by giving all information about the environment to the agent. This representation of the state may not be necessary. The reason is that the agent that selects an order set of jobs on a machine does not need to have information about other jobs. Action selection in most works is based on the selection of jobs to be executed on a machine. This puts the agent in a situation where multiple actions are available on the same machine. We can find this phenomenon in large workshops.

## 2 Problem description

Job shop-type workshops are the most complicated in terms of assigning tasks to machines, where each job has its specific routing and the processing time of tasks in the same machine is not necessarily identical as shown in Fig. 1.

In a Job Shop, we have a set of  $n$  Jobs:  $\{J_1, J_2, J_3, \dots, J_n\}$ , each job has its operations:  $J_i = \{O_{i1}, O_{i2}, O_{i3}, \dots, O_{ij}, \dots, O_{im}\}$  with  $i \in \{1, n\}, j \in \{1, m\}$  and  $m$  is the number of operations executed in  $m$  machines:  $\{M_1, M_2, M_3, \dots, M_m\}$ . All the tasks or the operations  $O_{ij}$  are defined with a start time  $t_{ij}$ , a processing time  $p_{ij}$ , and a completion time  $c_{ij}$ .

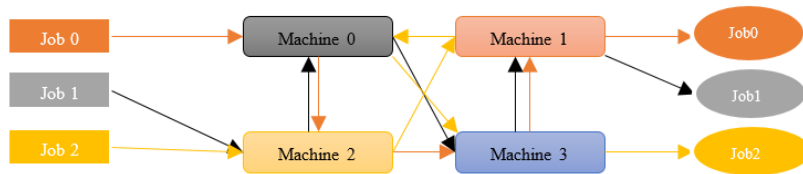
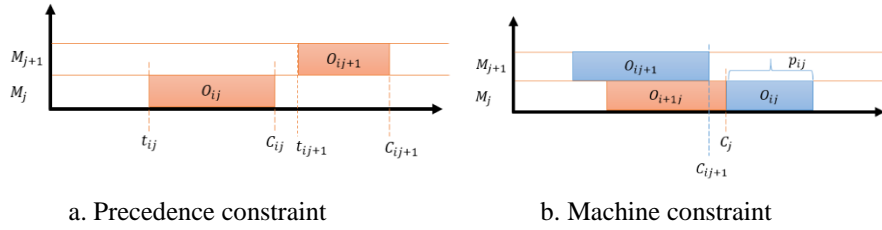


Fig. 1. Job shop example

The completion time of operations on a machine is referred to as  $C_j$ . The maximum load  $maxLoad_j$  of a machine  $M_j$  is the sum of the processing times of all operations to be executed in this machine. The instantaneous load:  $instLoad_j$ , represents the load of the machine after the affectation of an operation (it is the ratio between executed operations and  $maxLoad_j$ ).  $batch_j$  and  $seq_j$  are the current queues of  $M_j$  and the current sequence of tasks of this machine respectively.

Important constraints should be respected to avoid overlapping during the allocation of operations to the machines. The precedence constraint is presented in Fig.2.a, where we can't start a new operation  $O_{ij+1}$  of  $J_i$  without finishing the previous one  $O_{ij}$ .



**Fig. 2.** Constraints presentation.

The other constraint related to the completion time of machine  $C_j$ , we can't start an operation in a machine without finishing the previous one as presented in Fig.2.b. To calculate it, we should see the maximum between the next operation's completion time  $C_{ij+1}$  in a machine  $M_{j+1}$  and the completion time of the machine, which is:

$$C_j = p_{ij} + \max(C_j, C_{ij+1})$$

The objective of our study is to minimize the total completion time of all jobs in all machines

$$C_{max} (makespan) = \min(C_{max}, \text{with } C_{max} = \max(C_j))$$

### 3 Reinforcement learning

RL is a technique that learns what an agent does to interact with a given environment to maximize rewards [21]. A learning agent must be able to sense the state of its environment to some extent and take actions that affect the state. The agent must also have one or more targets related to the state of the environment [22].

There are several RL algorithms, in our paper we used Q-learning. It represents a form of model-free RL algorithm. The core of the algorithm is a simple value iteration update, each state-action pair  $(s, a)$  has a Q-value associated. When action  $a$  is selected by the agent located in state  $s_t$ , the Q-value for that state-action pair is updated based on the reward received when selecting that action, and the best Q-value for the subsequent state  $s_{t+1}$  [23]. The Q-values are saved in a Q-table, making the agent access them rapidly. The function of Bellman which updates the Q-values is as follows:

$$Q^{new}(s_t, a_t) = (1 - \alpha)Q^{old}(s_t, a_t) + \alpha(r_t + \gamma(\max_a(Q(s_{t+1}, a)))) \quad (1)$$

Where  $\alpha$  represents the learning rate, it is the probability of acceptance of the target value.  $(1 - \alpha)$  is the probability of keeping the old Q-value and  $\gamma$  is a discount factor, it is used to balance the immediate and the future reward.  $(\max(Q(s_{t+1}, a)))$  represents the maximum Q-value of the next state, that is the reason why the Q-learning is an off-policy algorithm, the policy used during the evaluation stage can differ from the one used in the improvement stage, which leads to more exploration at the expense of convergence speed [24]. In other words, it is not necessarily that the action  $a_t$  chosen in the state  $s_t$  is the same one as  $a$  in the target  $(r_t + \gamma(\max(Q(s_{t+1}, a))))$ .

The goal is to find the optimal policy in the long term, to reach this objective. The agent needs a process to explore the environment by taking actions randomly and interacting with the environment, and exploits by maximizing its gains when it finds an optimal policy. The  $\epsilon$ -greedy policy is a policy that allows the agent to explore with a probability of  $\epsilon$  and exploit with a probability of  $1 - \epsilon$ .

## 4 Implementation

Now that we have presented the problem and explained the algorithm we used, we will see in this section the presentation of the basic elements of the algorithm in terms of the JSSP, then the learning phase, and the application phase of the algorithm.

### Presentation of the elements of the learning algorithm

#### *State representation.*

Our objective is to minimize the total completion time  $C_{max}$ . We have represented the state as a machine with at least two tasks in the queue. The agent must choose an action that represents the select of one prioritized task in the queue. We have chosen to define the state in this way because the necessary information for the agent is that which must be allocated at this moment. There is no decision to take for the machines that have only one job in their queue. So the state in equation (1) is as follows:

$$s_t = batch_j \quad \text{when } |batch_j| \geq 2$$

#### *Action selection.*

In our study, we tried to choose the action differently, the agent will choose one of the popular dispatching rules for each decision. There are two reasons why we choose this selection. Firstly, it is interesting and easy to understand to schedule using a rule in a state; A different rule may perform better in another state. Secondly, the agent will choose only four actions, which are the dispatching rules SPT, LPT, FIFO and LIFO instead of choosing all the possible Jobs. So, the actions in equation (1) are:

$$a_t = \{SPT, LPT, FIFO, LIFO\}$$

#### *Reward function.*

The reward function represents an important step in the implementation of the algorithm. The reward is most often proportional to the optimization target or a value that highly correlates with it (e.g. makespan and average utilization) [24]. In our study, the reward has a relation to the machine loads ( $instLoad_j$ ) and the completion time of the job. If the machines have been well loaded at the time of the decision means that they are well balanced and the completion time is best. When the agent is in a state  $s_t$  and

it must take a decision on  $M_j$ , it calculates the load of each machine ( $instLoad_j$ ), then it calculates the average loads of all the machines except  $M_j$ . Then, the current load of  $M_j$  ( $instLoad_j$ ) is compared with the average to see if the machine is balanced with the others or not. In this step, we will evaluate the action chosen by the agent, and whether it deserves a reward or a penalty. For instance, if  $instLoad_j = 20\%$  and the average load of the other machines is  $60\%$ , means that the machines are not well balanced. In this case, the correct action is to choose the rule that prioritizes the job with the longest processing time. If the chosen rule does not select first the job with this characteristic, a penalty is given.

The reward function has a relation to the completion time obtained by applying the action chosen by the agent and the completion time obtained for the other actions. This process evaluates the policy chosen by the agent, makes cooperation between the dispatching rules in the same episode and leads to the optimal policy:

$$r_t = \begin{cases} (1 - C_j) / \sum C_j (\text{all actions}), & \text{if chosen action balances the loads} \\ -(1 - C_j) / \sum C_j (\text{all actions}), & \text{otherwise} \end{cases}$$

We have used another method of representing the reward while keeping the same idea of the evaluation manner, as shown below:

$$r_t = \begin{cases} +1, & \text{if chosen action balances the loads} \\ -2, & \text{otherwise} \end{cases}$$

The agent will be evaluated twice, the first one during the learning process in an episode, i.e. when assigning jobs to machines, using loads and completion times as we mentioned before. The second is at the end of the episode where it obtains the  $C_{max}$ . In this case, the evaluation is done using the  $C_{max}$  obtained, if it is the smallest until now, we will maximize the rewards of all the actions taken in this episode. Otherwise, we will add nothing. This policy helps the agent maximize the gain only for the right decisions. The evaluation at the end of the episode is represented as follows:

$$r_t = ((C_{max}(\text{maximal}) - C_{max}) / C_{max}(\text{maximal})) \times \text{episode}, \quad C_{max} \leq C_{max}(\text{maximal})$$

### Learning phase

In this phase, we will use the Q-learning algorithm adapted to the different characteristics of the JSSP. During the learning phase, the agent will try to maximize the rewards of a good solution. We have defined the reward of how it will maximize each time it finds a good solution to its exploration path. But it must also increase the exploitation rate each time when it finds a good solution. We have added a mechanism that will help the agent to exploit much more solutions each time it finds a good solution. This is conducted by increasing the value of epsilon each time until epsilon is greater than 0.9, it will take the value of 0.999. In this way, the agent does not explore too much, which leads to slow learning, and does not converge quickly to the best solution. For the same reason, the agent will not exploit too much so that the actions do not lead to local solutions. Table 1 resumes the learning phase.

**Table 1.** Learning Phase Algorithm

---

1.	initialize the Q – table empty
2.	calculate the $maxLoad_j$ and $instLoad_j$ of all the machines
3.	$\epsilon = 0.2, \alpha = 0.9, \gamma = 0.75$
4.	For several episodes:
	<ul style="list-style-type: none"> <li>While the end of the allocation of all jobs is not done:           <ul style="list-style-type: none"> <li>a. Fill the machines queue</li> <li>b. If <math> batch_j  \geq 2</math>:               <ul style="list-style-type: none"> <li>• Add a row Q – table with 4 columns with null values</li> <li>• Choose an action using an <math>\epsilon</math> – greedy policy</li> <li>• Calculate the reward using the first evaluation</li> <li>• Update the Q – value (<math>Q^{old}(s_t, a_t) \leftarrow Q^{new}(s_t, a_t)</math>) in the Q – table</li> <li>• Add the job to the machine.seq</li> <li>• Update the state of the batch (<math>s_t \leftarrow s_{t+1}</math>)</li> <li>• If there is another machine <math>[m + 1]</math>. batch <math>\geq 2</math> return to (b)</li> </ul> </li> </ul> </li> <li>Compare the <math>C_{max}</math> of this episode with the smallest one obtained and calculate the reward using the second evaluation</li> <li>Increase the <math>\epsilon</math> value</li> <li>If <math>\epsilon &gt; 0.9</math>:           <ul style="list-style-type: none"> <li><math>\epsilon = 0.999</math></li> </ul> </li> </ul>

---

**Final model**

After the end of the learning phase, the Q-table contains all the states experienced by the agent, as well as the Q-values of each action applied in each state. Actions are considered as good decisions with maximum Q-values or bad decisions with minimum Q-values, since the learning strategy leads to maximizing the gain for good decisions. In this phase, the agent's optimal policy  $\pi^*$  is used to choose actions with maximum Q-value. This brings us to the solution approximated by the learning algorithm.

$$\pi^*(s_t) = \max(Q(s_t, a))$$

**5 Results and discussion**

QL01 represents the results by using the reward function related to the completion time of actions, and QL02 where the reward is +1 and -2. We have applied this experience to 46 benchmark instances from the OR Library [25] and compared the results given by QL01 and QL02 with the dispatching rules (FIFO, SPT, LPT), GA [10], HGA [11], DRL [12] and the optimal solution as well. The algorithms are executed under 1000 episodes for small instances (ft06, ft10, la01 to la15, orb01 to orb03 and orb07) and 10000 episodes for mean and large scales (la16 to la40) in a reasonable time. The results are shown in Table. 2. Columns QL01 and QL02 in the second row of this table show if the proposed approach outperformed the compared one (value equal to 1 if it is the case).

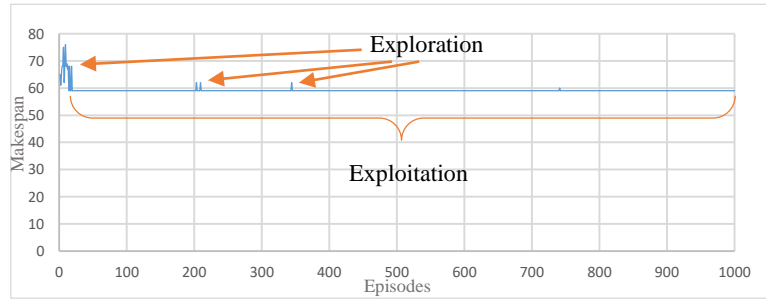
Our approaches gave good results compared with the dispatching rules. In all the instances, the proposed algorithms gave solutions much closer to the optimal solution

**Table 2.** Results of QL01 and QL02, dispatching rules (SPT, LPT, FIFO), meta-heuristics (GA, HGA), and DRL.

Workshop	SPT	LPT	FIFO	OPT	QL 01		QL 02		GA [10]		HGA [11]		DRL [17]							
					C <sub>max</sub>	Error	C <sub>max</sub>	Error	C <sub>max</sub>	Error	QL 01	QL 02	C <sub>max</sub>	Error	QL 01	QL 02	C <sub>max</sub>	Error	QL 01	QL 02
ft06(06x06)	88	77	65	55	57	-2	57	-2	55	0	0	0	55	0	0	0	57	-2	1	1
ft10(10x10)	1074	1295	1184	930	1017	-87	999	-69	994	-64	0	0	938	-8	0	0	1033	-103	1	1
la01(05x10)	751	822	772	666	666	0	666	0	667	-1	1	1	666	0	1	1	666	0	1	1
la02(05x10)	821	990	830	655	685	-30	685	-30	676	-21	0	0	655	0	0	0	715	-60	1	1
la03(05x10)	672	825	755	597	623	-26	619	-22	627	-30	1	1	597	0	0	0	634	-37	1	1
la04(05x10)	711	818	695	590	620	-30	620	-30	608	-18	0	0	590	0	0	0	665	-75	1	1
la05(05x10)	610	693	610	593	593	0	593	0	593	0	1	1	593	0	1	1	593	0	1	1
la06(05x15)	1200	1125	926	926	926	0	926	0	926	0	1	1	926	0	1	1	926	0	1	1
la07(05x15)	1034	1069	1088	890	890	0	967	-77	891	-1	1	0	890	0	1	0	894	-4	1	0
la08(05x15)	942	1035	980	863	863	0	876	-13	863	0	1	0	863	0	1	0	863	0	1	0
la09(05x15)	1045	1183	1018	951	951	0	951	0	951	0	1	1	951	0	1	1	951	0	1	1
la10(05x15)	1049	1132	1006	958	958	0	958	0	958	0	1	1	958	0	1	1	958	0	1	1
la11(05x20)	1473	1467	1272	1222	1222	0	1222	0	1222	0	1	1	1222	0	1	1	1222	0	1	1
la12(05x20)	1203	1240	1039	1039	1039	0	1039	0	1039	0	1	1	1039	0	1	1	1039	0	1	1
la13(05x20)	1275	1230	1199	1150	1150	0	1150	0	1150	0	1	1	1150	0	1	1	1150	0	1	1
la14(05x20)	1427	1434	1292	1292	1292	0	1292	0	1292	0	1	1	1292	0	1	1	1292	0	1	1
la15(05x20)	1339	1612	1587	1207	1207	0	1302	-95	1256	-49	1	0	1207	0	1	0	1212	-5	1	0
la16(10x10)	1156	1229	1180	945	983	-38	995	-50	993	-48	1	0	945	0	0	0	/	/	/	/
la17(10x10)	924	940	943	784	800	-16	800	-16	804	-20	1	1	784	0	0	0	/	/	/	/
la18(10x10)	981	1114	1049	848	873	-25	861	-13	874	-26	1	1	848	0	0	0	/	/	/	/
la19(10x10)	940	1062	983	842	875	-33	875	-33	895	-53	1	1	844	-2	0	0	/	/	/	/
la20(10x10)	1000	1272	1272	902	939	-37	941	-39	942	-40	1	1	911	-9	0	0	/	/	/	/

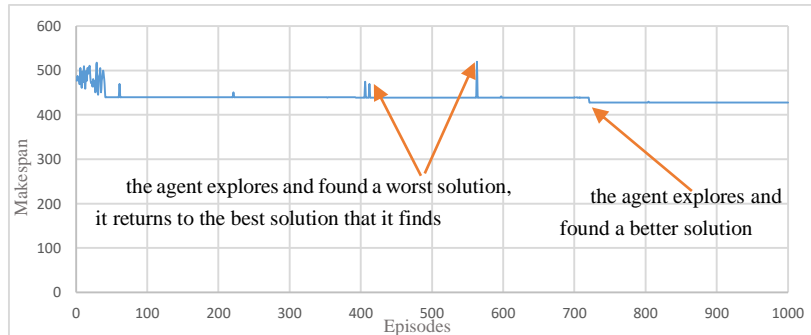
la21(10x15)	1324	1451	1265	1046	1107	-61	1126	-80	1180	-134	1	1	1046	0	0	0	/	/	/	/	
la22(10x15)	1180	1315	1369	927	1022	-95	1026	-99	1103	-176	1	1	935	-8	0	0	/	/	/	/	
la23(10x15)	1162	1302	1354	1032	1038	-6	1053	-21	1100	-68	1	1	1032	0	0	0	/	/	/	/	
la24(10x15)	1203	1245	1141	935	1000	-65	1021	-86	1077	-142	1	1	953	-18	0	0	/	/	/	/	
la25(10x15)	1449	1374	1283	977	1074	-97	1059	-82	1116	-139	1	1	984	-7	0	0	/	/	/	/	
la26(10x20)	1499	1564	1372	1218	1302	-84	1327	-109	1433	-215	1	1	1218	0	0	0	/	/	/	/	
la27(10x20)	1784	1700	1644	1252	1364	-112	1361	-109	1469	-217	1	1	1256	-4	0	0	/	/	/	/	
la28(10x20)	1610	1844	1532	1273	1358	-85	1363	-90	1408	-135	1	1	1225	48	0	0	/	/	/	/	
la29(10x20)	1556	1720	1540	1238	1363	-125	1348	-110	1439	-201	1	1	1196	42	0	0	/	/	/	/	
la30(10x20)	1792	1866	1664	1355	1390	-35	1440	-85	1546	-191	1	1	1355	0	0	0	/	/	/	/	
la31(10x30)	1954	2340	1918	1784	1857	-73	1802	-18	1906	-122	1	1	1784	0	0	0	/	/	/	/	
la32(10x30)	2165	2513	2110	1850	1914	-64	1883	-33	2002	-152	1	1	1850	0	0	0	/	/	/	/	
la33(10x30)	1901	2306	1873	1719	1817	-98	1802	-83	1838	-119	1	1	1719	0	0	0	/	/	/	/	
la34(10x30)	2005	2324	1925	1721	1828	-107	1794	-73	1934	-213	1	1	1721	0	0	0	/	/	/	/	
la35(10x30)	2118	2421	2142	1888	1985	-97	1925	-37	2106	-218	1	1	1888	0	0	0	/	/	/	/	
la36(15x15)	1854	1946	1516	1268	1415	-147	1372	-104	1480	-212	1	1	1287	-19	0	0	/	/	/	/	
la37(15x15)	1632	1944	1873	1397	1533	-136	1561	-164	1606	-209	1	1	1408	-11	0	0	/	/	/	/	
la38(15x15)	1395	1732	1475	1196	1334	-138	1351	-155	1435	-239	1	1	1219	-23	0	0	/	/	/	/	
la39(15x15)	1540	1822	1532	1233	1358	-125	1360	-127	1456	-223	1	1	1245	-12	0	0	/	/	/	/	
la40(15x15)	1493	1822	1604	1222	1311	-89	1315	-93	1492	-270	1	1	1241	-19	0	0	/	/	/	/	
orb01(10x10)	1478	1410	1368	1059	1119	-60	1177	-118	1212	-153	1	1	/	/	/	/	1131	-72	1	0	
orb02(10x10)	1175	1293	1007	888	918	-30	922	-34	960	-72	1	1	/	/	/	/	993	-105	1	1	
orb03(10x10)	1179	1430	1405	1005	1071	-66	1075	-70	1162	-157	1	1	/	/	/	/	1092	-87	1	1	
orb07(10x10)	475	470	504	397	418	-21	423	-26	408	-11	0	0	/	/	/	/	432	-35	1	1	
% of instances with good re- sults												89,13	80,43		27,90		20,93		100		80,95

compared to the minimal solution among the dispatching rules. The reason is that it is better to use a mix of rules in a problem rather than using only one rule. By comparing the difference between the GA and optimal solution and the difference between the QL01 and optimal solution, 89,13 % of the 46 instances give the best result than GA especially the instances with a large scale. For QL02, it was 80,43% of the 46 instances it gives a good solution compared with GA in large scales and some average scales instances except la07, la08, la15, and la16. By comparing with HGA, our algorithm QL01 gives a good result for 27,90% of the compared instances, and 20,93 % in QL02. The reason why our approaches did not give good results compared with the HGA may be because of the limits of the possible actions. The agent in our case chooses four actions instead of all the possible Jobs that can be allocated. For DRL, QL01 gives an interesting solution for all the compared instances (100%), and 80,95% compared with QL02.



**Fig. 3.** Convergence of solution for instance ft06 (06x06) QL01

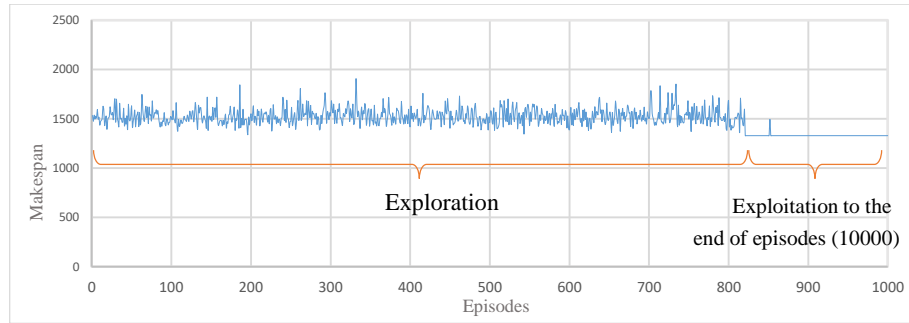
In Fig 3, we present the convergence of the learning algorithm during the learning phase. At the beginning of the learning, the agent explores the environment. Once it finds a better solution, it exploits and keeps a probability of exploring.



**Fig. 4.** Convergence of solution for instance orb07 (10x10) QL02

As shown in the picks in Fig 4, in case it finds a bad solution, it goes back to the good solution founded and maximizes it, in case it finds another good solution it will maximize the rewards for the new one. The large size instances does not limit the agent

to searching the solution space and finding a solution that approaches the optimal solution, as shown in Fig 5. We can notice that the exploration time of the environment increases every time we have a larger instance. This is due to the expansion of the solution space from one instance of small size to another instance of large size.



**Fig. 5.** Convergence of solution for la40 (15 x15) QL02

## 6 Conclusion

In our article, we discussed an important problem of production planning process in a manufacturing system, namely JSSP. We use QL to solve this problem because it is best suited for this type of problem. We use a new representation of the state of the environment that only affects the machines on which the agent will make decisions. Two new reward functions are introduced based on loads of machines. The agent's decision is then evaluated first during the assignment of jobs and second at the end of the schedule.

The application of the algorithm is compared with literature work. The results of the algorithm outperformed the genetic algorithm and dispatching rules. However, due to the number of agent's action, the algorithm was not able to perform better than a deep reinforcement learning. Since this approach yields interesting results, as a perspective, we will apply it to more complex conditions, where the agent will learn in a dynamic environment, when the tasks to be planned over some time are not known in advance and arrive gradually.

## References

- [1] K. Csaba, L. Catherine, G. Viola, et S. Wilfried, « Dynamic scheduling in a job-shop production system with reinforcement learning », 18 May 2020.
- [2] C. GEORGE et S. Velusamy, « Dynamic scheduling of manufacturing job shops using genetic algorithms », *Journal of Intelligence manufacturing*, p. 281-293, June 2001.
- [3] S. Johann et S. Sebastian, « Approaching Scheduling Problems via a Deep Hybrid Greedy Model and Supervised Learning », *Proceedings of the 17th IFAC Symposium on Information Control Problems in Manufacturing Budapest, Hungary*, p. 804-809, 7 June 2021.
- [4] C. Chen-Yang, P. Pourya, Y. Kuo-Ching, et L. Chen-Fang, « Unsupervised Learning-based Artificial Bee Colony for minimizing non-value-adding operations », *journal of Applied Soft Computing*, 17 March 2021.
- [5] L. Sebastian, K. Maximilian, K. Paul, et R. Tobias, « Modeling Production Scheduling Problems as Reinforcement Learning Environments based on Discrete-Event Simulation and OpenAI Gym », *Proceedings of the 17th IFAC Symposium on Information Control Problems in Manufacturing Budapest, Hungary*, p. 792-797, 7 June 2021.

- [6] Y.-C. Wang et Y.-C. M. Usher, « Application of reinforcement learning for agent-based production scheduling », *Engineering Applications of Artificial Intelligence*, p. 73-82, 25 September 2004.
- [7] N. Aissani et D. Trentesaux, « Efficient and effective reactive scheduling of manufacturing system using Sarsa-multi-objective agents », *Proceedings of the 7th international conference MOSIM*, Paris, p. 698-707, 31 April 2008.
- [8] C.-L. Liu, C.-J. Tseng, et C.-C. Chang, « Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems ».
- [9] Y. Wei et M. Zhao, « Composite Rules Selection Using Reinforcement Learning for Dynamic Job-Shop Scheduling », *Proceedings of the IEEE Conference on Robotics, Automation and Mechatronics*, Singapore, p. 1083-1088, 1 December 2004.
- [10] B. M. Ombuki et M. Ventresca, « Local Search Genetic Algorithms for the Job Shop Scheduling Problem », *Applied Intelligence*, p. 99-109, 2004.
- [11] R. Qing-dao-er-ji et Y. Wang, « A new hybrid genetic algorithm for job shop scheduling problem », *Computers & Operations Research*, p. 2291-2299, 2012.
- [12] L. Wang *et al.*, « Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning », 26 February 2021.
- [13] G. Thomas et R. Martin, « On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks », *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, p. 68-75.
- [14] Y. WEI, X. JIANG, P. HAO, et K. GU, « Pattern Driven Dynamic Scheduling Approach using Reinforcement Learning », *Proceedings of the IEEE International Conference on Automation and Logistics*, Shenyang, China, p. 514-519, August 2009.
- [15] W. Yu-Fang, « Adaptive job shop scheduling strategy based on weighted Q-learning algorithm », *Journal of Intelligent Manufacturing (2020) 31:*, p. 417-432, 7 November 2018.
- [16] Y. M. Jiménez, J. C. Palacio, et A. Nowé, « Multi-Agent Reinforcement Learning Tool for Job Shop Scheduling Problems », *Proceedings of the Third International Conference, OLA 2020*, Cádiz, Spain, p. 3-12, 17 February 2020.
- [17] L. Zhou, L. Zhang, et P. H. Berthold K, « Deep reinforcement learning-based dynamic scheduling in smart manufacturing », *53rd CIRP Conference on Manufacturing System*, p. 383-388.
- [18] B. Waschneck *et al.*, « Optimization of global production scheduling with deep reinforcement learning », *Procedia CIRP 72*, p. 1264-1269, 2018.
- [19] P. Tassel, M. Gebser, et K. Schekotihin, « A Reinforcement Learning Environment For Job-Shop Scheduling », *arXiv preprint arXiv:2104.03760*, 2021.
- [20] V. Samsonov *et al.*, « Manufacturing Control in Job Shop Environments with Reinforcement Learning », *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021)*, p. 589-597, 2021.
- [21] Juan. P. Usuga.C, S. Lamouri, B. Grabot, R. Pellerin, et A. Fortin, « Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0 ».
- [22] R. S. Sutton et A. G. Barto, *Reinforcement Learning - An Introduction -*, Second edition. 2020.
- [23] J. C. Palacio, Y. M. Jiménez, L. Schietgat, B. Van Doninck, et A. Nowé, « A Q-Learning algorithm for flexible job shop scheduling in a real-world manufacturing scenario », *Procedia CIRP 106*, p. 227-232, 2022.
- [24] A. Rinciog et A. Meyer, « Towards Standardizing Reinforcement Learning Approaches for Stochastic Production Scheduling », *arXiv preprint arXiv:2104.08196*, 2021.
- [25] « OR - Library ». [En ligne]. Disponible sur: <http://people.brunel.ac.uk/~mastjjb/jeb/or-lib/files/jobshop1.txt>