An Improvement of Reinforcement Learning Approach for Permutation of Flow-Shop Scheduling Problems

Article *in* RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao · July 2019

CITATIONS
0

READS
293

5 authors, including:

Yunior César Fonseca-Reyna
National Center for Biotechnology (CNB)
21 PUBLICATIONS **38** CITATIONS
SEE PROFILE

Amilkar Puris
Universidad Técnica Estatal de Quevedo
52 PUBLICATIONS **287** CITATIONS
SEE PROFILE

Yailen Martinez
Universidad Central "Marta Abreu" de las Villas
30 PUBLICATIONS **98** CITATIONS
SEE PROFILE

risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Iberian Journal of Information Systems and Technologies

Recebido/Submission: 19/11/2018
Aceitação/Acceptance: 23/12/2018

An Improvement of Reinforcement Learning Approach for Permutation of Flow-Shop Scheduling Problems

Yunior César Fonseca Reyna[1], Amilkar Puris Cáceres[2], Yailen Martínez Jiménez[3], Yaima Trujillo Reyes[2]

cfonseca@cnb.csic.es, apuris@uteq.edu.ec, yailenm@uclv.edu.cu, ytrujillo@uteq.edu.ec

[1] Laboratorio de Biocomputacíon, Centro Nacional de Biotecnología de Madrid, Calle Darwin, 3, 28049, Madrid, España.

[2] Universidad Técnica Estatal de Quevedo, 120508, km 1.5 vía Santo Domingo, Quevedo, Ecuador.

[2] Universidad Central Marta Abreu de las Villas, 50100, km 5.5 vía Camajuaní, Santa Clara, Cuba.

Pages: 257–270

Abstract: The tasks scheduling problem on linear production systems, Flow Shop Scheduling Problem, has been of a great importance in Operations Research. This paper addresses the problem of minimizing the completion time of all jobs, known in literature as makespan or C_{max}. This study presents an adaptation of a Reinforcement Learning approach known as Q-Learning which is combined with a local search to improve the solutions. The approach employs a NEH heuristic to give a potential starting solution in the search space and finally we applied the improvement through the different local search methods. An evaluation of different parameters combinations of the algorithm is also given. To validate the quality of the solutions, we test our approach on several standard Taillard's benchmark problems and the algorithm effectiveness is compared against eight other advanced techniques. The results show that the proposed algorithm reaches high-quality solutions in short computational times.

***Keywords*:** Flow-shop; makespan; optimization; q-learning; scheduling

1. Introduction

Scheduling is one of the most important issues in Operations Research. It is a form of decision-making that plays a crucial role in manufacturing and service industries and a very active field with a high practical relevance (Wu et al. 2005). A scheduling problem is to find sequences of jobs on a given set of machines with the objective of minimizing some function of the job completion times (Fonseca, M. Martínez, et al. 2015). Depending on the problem being solved, jobs and machines can take many forms, and the objectives can also vary. The problems can be classified according to different characteristics, for example, the number of machines (one machine, parallel machines), the job characteristics (preemption allowed or not, equal processing times) and so on.

When each job has a fixed number of operations requiring different machines, we are dealing with a shop problem, and depending on the constraints it presents, it can be classified as Open Shop, Job Shop, Flow Shop, etc.

In this research we focus on manufacturing scheduling where all jobs share the same route, specifically the Flow Shop Scheduling Problem (FSSP), which has been extensively studied due to its application in industry. This problem is typical of combinatorial optimization and can be found in manufacturing environments. Due to the NP-hard (Fonseca and M. Martínez 2017) nature of the problem, most of the solution procedures employ heuristic approaches to obtain near-optimal sequences in reasonable time. There are many methods for an approximation of the optimal solution by searching only a part of the space of feasible solutions (represented here by all the permutations).

The scheduling literature is abundant with many solution procedures for the FSSP that aim to minimize the makespan or another criterial. Ruiz and Moroto (Ruiz and Moroto 2005), and Mehmet and Betul (Mehmet and Betul 2014) have presented extensive reviews and evaluations of many exact methods, approximation methods, heuristics and meta-heuristics for the FSSP with the makespan as the criterium to optimise. Recently, (Seido-Nagano and HissashiMiyata 2016) published a review about constructive heuristics.

Metaheuristic approaches such as Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms (GA) are very desirable to solve combinatorial optimization problems regarding to their computational performance (Zhang et al. 2009; Chaudhry and Munem khan 2012). Ant Colony Optimization (ACO) approach has been used to solve the FSSP too. Several ACO algorithms by Betul and Memet (Betul and Mehmet-Mutlu 2008), Tavares-Neto (Tavares-Neto and Godinho-Filho 2011) and Puris (Puris et al. 2007) applied this technique to minimize the makespan or another objectives and tested with well-known problems in literature. Even, many authors have investigated the Particle Swarm Optimization algorithm (PSO) to solve the FSSP. Such is the case of Chandresakaran et al. in (Chandrasekaran et al. 2007) where the makespan, total flowtime and completion time variance are considering simultaneously. In (Tasgetiren et al. 2007) the autor, proposed two variants of PSO called PSOvns and HCPSO respectively, which found many best solutions for the first 90 Taillard benchmark instances Taillard 1993. As it was mentioned before, many interesting approaches have been proposed to solve the FSSP. In the next section, the FSSP is described in detail. After this, we summarize the main results of the application of the Q-learning algorithm in the solution of the problem. We start by introducing a basic approach, which is modeled for the FSSP, and then we explain how to adapt this model in order to deal with the constraints that appear in the scheduling scenario. Finally, we present the results obtained by experimenting and comparing with other approaches reported in literature.

2. Material and Method

In many manufacturing and assembly facilities each job has to undergo a series of operations. Often, these operations have to be done on all jobs in the same order implying that the jobs have to follow the same route. The machines are then assumed to be set up in series and the environment is referred to as a FSSP (Pinedo 2008). There is

a set of m different machines $\{M_1,..., M_n\}$ and a set of n independent jobs $\{J_1,..., J_n\}$. Each job comprises a set of m operations which must be executed on the different machines. O_{ij} denotes the operation on machine i of job j. The objective is to find an ordering of the jobs on the machines or sequence that optimizes some given criterion. This paper adresses the problem of minimizing the completion time of all jobs, known in literature as makespan or C_{max}. This problem is usually denoted by $n|m|p|C_{max}$ Yamada 2003 and we take into account the following assumptions: Each job i can only be processed on one machine at any time; each machine j can process only one job i at any time; no preemption is allowed, i.e. the processing of a job i on a machine j cannot be interrupted; all jobs are independent and are available for processing at time zero; the set-up times of the jobs on the machines are sequence independent and are included in the processing times; the machines are continuously available; transportation times are negligible.

As mentioned, the objective is to find a permutation of jobs to be sequentially processed on a number of machines under the restriction that the processing of each job has to be continuous with respect to the objective of minimizing the C_{max}. Therefore: If we have $p(i,j)$ as the processing time of job i on machine j and a job permutation $J1,J2,...,Jn$, then we calculate the completion times $C(J_i,j)$ as follows :

$$C(J_1,1) = p(J_1,1) \tag{1}$$

$$C(J_i,1) = C(J_{i-1},1) + p(J_i,1) \quad for\ i = 2,...,n \tag{2}$$

$$C(J_1, j) = C(J_1, j-1) + p(J_1, j) \quad for\ j = 2,...,m \tag{3}$$

$$C(J_i, j) = max\{C(J_{i-1}, j), C(J_i, j-1) + p(J_i, j)\} \ for\ i = 2,...,n;\ for\ j = 2,...,m \tag{4}$$

$$C_{max} = C(J_n,m) \tag{5}$$

In other words, C_{max} is the end time of the last operation in the last machine (Fonseca, Y. Martínez, et al. 2017).

Reinforcement Learning (RL) is a learning approach which can be applied to a wide variety of complex problems. The ideas were originally developed by Sutton and Barto (Sutton and Barto 2016). RL is learning what to do (how to map situations to actions) so as to maximize a numerical reward signal. In the standard RL model, an agent is connected to its environment via perception and action, as depicted in Figure (1).

In each interaction step, the agent perceives the current state S of its environment, and then selects an action a to change this state. This transition generates a reinforcement signal r, which is received by the agent. The task of the agent is to learn a policy for choosing actions in each state to receive the maximal long-run cumulative rewards. RL methods explore the environment over time to come up with a desired policy (Y. Martínez 2012).

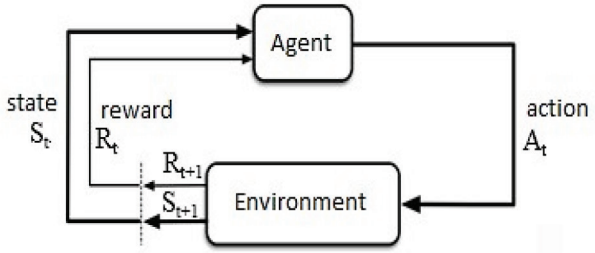

Figure 1 – The standard reinforcement learning model

Formally, the basic reinforcement learning model consists of: a set of environment states S; a set of actions A; a set of scalar ”rewards” in R and a transition function T.

At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives from the environment the new state s_{t+1} and a reward r_{t+1}, this means that the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(s,a)$ is the probability that $a_t = a$ if $s_t = s$, in words, is the probability of selecting action a in state sat time t.

The reward function defines the goal in a RL problem. A RL agent's sole objective is to maximize the total reward it receives in the long run. Consequently, an agent in the RL paradigm must actively explore its environment to observe the effects of its actions. In summary, RL provides a flexible approach to the design of intelligent agents in situations for which, for example, planning and supervised learning are impractical. RL can be applied to problems for which significant domain knowledge is either unavailable or costly to obtain.

After presenting the basic idea of RL, we present the main concepts that have to be taken into account when solving a scheduling problem using RL. Figure 2 shows an initial idea of the learning environment. Following the standard model presented in Figure 1, this is how we map agents and actions in the environment when solving a scheduling problem. The scheduling environment defines the number of agents in the system and the relations among them. Agents may not know the global state of the system. To achieve better performance of agents or the system, agents communicate with each other to determine their actions based on the limited information.

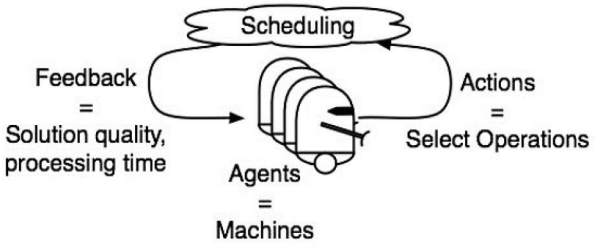

Figure 2 – Agents in a scheduling environment

FSSP are well suited to be modeled using the idea shown in the Figure 2 because the following information is always available at the beginning of the scheduling process:

A well-known RL algorithm is Q-Learning (QL) (Fonseca, A. Martínez Y. V., et al. 2019), which works by learning an action-value function that expresses the expected utility (i.e. cumulative reward) of taking a certain action in a given state. 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*, 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^o. The update rule for the state action pair (*s,a*) is the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \, max_a o Q(s^o,a^o) - Q(s,a)] \quad (6)$$

In this expression $\alpha \in [0,1]$ is the learning rate and *r* the reward or penalty resulting from taking action *a* in state *s*. The learning rate α determines 'the degree' by which the old value is updated. QL has the advantage that is proven to converge to the optimal policy in Markov Decision Processes under some reasonable assumptions (Y. Martínez 2012).

Algorithm 1 shows the pseudo code of QL. This algorithm is used by the agents to learn from experience or training. Each episode is equivalent to one training session. In each training session, the agent explores the environment and gets the rewards until it reaches to goal state. The purpose of the training is to enhance the knowledge of the agent represented by the Q-values. More training will give better values that can be used by the agent to move in more optimal way.

The agent needs to find a balance between exploration and exploitation. The ε–*greedy* action selection method instructs the agent to select the action with the highest Q-value most of the time, but sometimes, to choose an action at **1** *Initialize Q*(*s,a*) *arbitrarily*

```
1 Initialize Q(s, a) arbitrarily
2 for each episodes do
3 |   Initialize s
4 |   for each episodes step do
5 |   |   Choose a from s using policy derived from Q(e.g., e − greedy)
6 |   |   Take action a, observe state s' and r
7 |   |   Q(s,a)← Q(s, a) + α [r + γ max_a' Q(s', a') − Q(s, a)]
8 |   |   s ← s'
```

Algorithm 1 – Q-Learning Algorithm

random. The probability ε determines when to choose a random action; this allows to balance between exploration and exploitation.

Our approach uses QL to solve the FSSP (referred to as QL-FSSP), but incorporates the NEH heuristic idea developed by Nawaz et. al in (Nawaz et al. 1983) and apply four local

search operators to improve the final solution. When RL is applied to solve the FSSP, we use one agent as job permutation for this problem. This agent selects the actions that it must execute.

There are important elements to be decided when choosing the QL algorithm as solution method, which are summarized as follow:

States: For the agent, the definition of state will match with the jobs that have been completed and the order in which they were executed, that is, the sequence that has been built.

Actions: Similar to the NEH algorithm, taking an action by the agent is equivalent to decide into which position of the already constructed sequence that job with the largest total processing time will be entered, it is selected from the set of all the jobs that are still waiting to be processed.

Rewards: As described, the state is represented by the sequence that has been built, an action is to insert a new job into this sequence and taking into account that our ultimate goal is to minimize the makespan, we define $1/makespan$ as the reward; note that the greater is the makespan of the sequence, the lower the reward for the selected action. It should be noted that the largest Q-value determines the best action in any given state.

Q-values: The theoretical assumptions for the convergence of QL state that each state and action should be visited infinitely often. For scheduling problems, except for very small instances, this implies very large memory requirements in practice. So our algorithm stores only those states that are necessary, i.e., those in which the agent was present when actions were selected.

The proposed algorithm mechanism is schematically presented in Figure 3

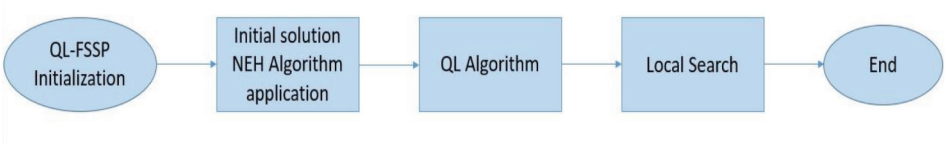

Figure 3 – The proposed algorithm mechanism

Initially, we introduced the well-known constructive heuristic NEH for the FSSP to obtain an initial sequence of jobs. Then, the QL algorithm take this sequence in order to improve the makespan. The Algorithm 2 summarized the QL algorithm applied to FSSP. When QL-FSSP algorithm determine the final solution, the local search phase begin taking into account four operators to explore the neighborhood of the obtained solution by the QL algorithm. The main idea of the local search procedure consist in select randomly one of these operators and modify the obtained sequence by QL-FSSP. If the new sequence is better than the current solution, the current solution is replaced. When the current solution is updated, local search continues for the new current solution in the same manner. This procedure is repeat while no better solution is found. The number of evaluated solutions without the makespan no better, is used as stopping criterion. Specifically, the following actions take place based on the operator value (*op*).

if op = 0, exchange of adjacent two jobs operator (Exchange Adjacent) is executed.

if op = 1, exchange of arbitrary two jobs operator (Exchange Arbitrary Two) is executed.

if op = 2, exchange of arbitrary three jobs operator (Exchange Arbitrary Three) is executed.

if op = 3, insert operation operator (Insert Operation) is executed.

Figures 4, 5, 6 and 7 shown schematically these operators.

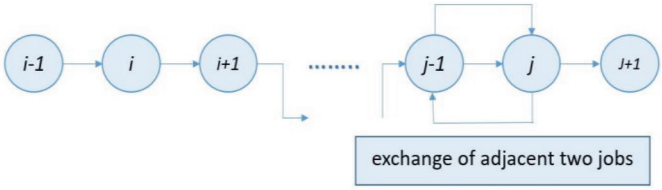

Figure 4 – Exchange of adjacent two jobs operator

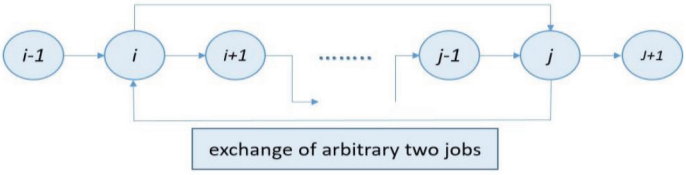

Figure 5 – Exchange of arbitrary two jobs operator

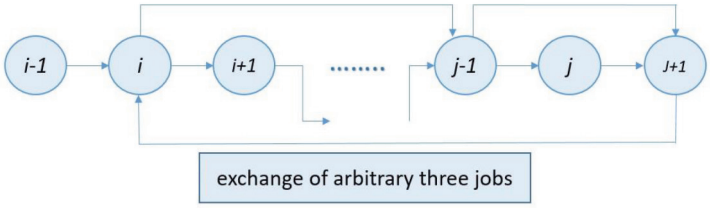

Figure 6 – Exchange of arbitrary three jobs operator

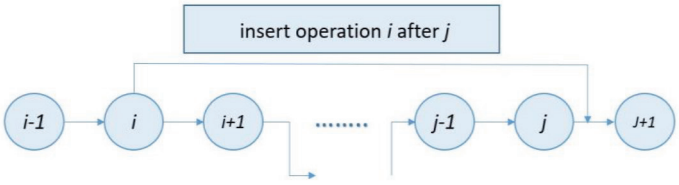

Figure 7 – Insert operator

Input**:** jobs sequence ordered by NEH algorithm: *Sequence*; Number of episodes: *episodes*; learning rate: α; discount factor: γ; value of epsilon; Number of evaluations:

```
1  Initialize:
2  Q(s, a) = []
3  Best = []
4  evaluationsAux ← evaluations
5  while ((episodes > 0) and (time < T_max)) do
6      Initialize:
7       seq = []
8      while (!Sequence.processAllJobs()) do
9          j_m ← selectPendenJobs()
10         actions ← insertPointSet(j_m)
11         if (random() < ε) then
12             a ← random(actions)
13         else
14             a ← bestAction(seq, j_m, actions)
15         Take action a, observe state s' and r = 1/makespan(s')
16         Q(s,a)← Q(s, a) + α [r + γ max_a' Q(s', a') − Q(s, a)]
17         s ← s'
18     if makespan(seq) < makespan(Best) then
19         Best ← seq
20         evaluationsAux ← 0
21     else
22         evaluationsAux ← evaluationsAux + 1
23         if (evaluationsAux > evaluations) then
24             return Best
25     episodes ← episodes − 1
```

Algorithm 2 – Q-Learning Applied to FSSP.

3. Result

In order to test the algorithm, we used the benchmark instances located in the OR-Library (Beasley 1990). The OR-Library is a library of problem instances covering various Operation Research problems. For this, we used different cases to compare the obtained results with the solutions offered in this research. We take the first 90 instances from Taillard's (Taillard 1993) datasets, 10 of each of the sizes 20×5, 50×5, 100×5, 20×10, 50×10, 100×10, 20×20, 50×20, 100×20. Every test was repeated with 10 runs

for each instance. Hence, there were 900 runs in total. The best-known upper bounds for these problems were used for comparison purposes. The Q-Learning algorithm was implemented in Java, running on a Pentium IV PC, with 2.4 GHz CPU and 2 GB RAM.

The performance of the proposed approach was compared with the performances of the ACO methods proposed by Stuetzle known MMAS (Stu¨etzle 1998) and the algorithms M-MMAS and PACO proposed by Rajendran and Ziegler (Rajendran and Ziegler 2004); a multiobjective ACO algorithm known MOACSA proposed by Betul and Mehmet (Betul and Mehmet-Mutlu 2010); the NEH heuristic presented by Nawaz et. al (Nawaz et al. 1983); a PSO metaheuristic proposed by Tasgetiren et. al known PSO_{VNS} (Tasgetiren et al. 2007) and the Genetic Algorithms HGA RMA developed by Ruiz and Moroto (Ruiz, Maroto, et al. 2006) and the $NEGA_{VNS}$ presented in (Zobolas et al. 2009). We selected these approaches to compare mainly because they used the same instances and reported results for all of them. In this research, we do not take the whole group of instances (120) because some authors do not provide results for all of them.

The stopping criterion is fixed to a given maximum elapsed CPU time that follows the expression $T_{max} = n*m/10$ milliseconds where n is the number of jobs and m is the number of machines, which results in practical computational times for all Taillard instances. It should be mentioned that (Ruiz, Maroto, et al. 2006) recoded all the implementations (except MOACSA, PSO_{VNS} and $NEGA_{VNS}$) to compare them more faithfully. For this reason, the reported results in (Ruiz, Maroto, et al. 2006) will be used for the comparative study. The previous expression is used in (Zobolas et al. 2009) and the $NEGA_{VNS}$ performance is compared with the results obtained in (Tasgetiren et al. 2007) and (Ruiz, Maroto, et al. 2006).

Taking into account the Dongarra research concerning the computational power of various computer configurations ranging from home PCs to modern supercomputers in (Dongarra 2014), we developed an analysis to conduct a fairer comparison between all methods. Although computational times are also affected by other parameters such as the programming skills of the developer, the compiler, the programming language, and the precision used during the execution of the runs, a scaling factor is calculated.

Some initial experiments were performed in order to analyze the learning process under the effect of different parameters values. Typical combinations for the QL algorithm are the following (Y. Martínez 2012):

C_1 : episodes = n ∗ m,α = 0.1,γ = 0.8,ε = 0.2

C_2 : episodes = n ∗ m,α = 0.1,γ = 0.9,ε = 0.1

C_3 : episodes = n ∗ m,α = 0.1,γ = 0.8,ε = 0.1

C_4 : episodes = n ∗ m,α = 0.1,γ = 0.9,ε = 0.2

After executing different experiments for each combination, we decided to keep the third combination, as it was able to yield better results:

C_3 : episodes = n ∗ m,α = 0.1,γ = 0.8, ε = 0.1

For this, we randomly take 10 instances from Taillard's and we apply a nonparametric test to determine the best combination using the results from these executions. We executed

10 runs for all the instances. Then, we applied the Friedman test to determine if there are significant differences and the Bergman Hommel's test to do multiple comparisons. The results obtained show that C_3 was the best combination.

Table 1 shows the Q-Learning performance under differents parameters combinations to ta001, ta007, ta011, ta022, ta034, ta052, ta057, ta062, ta078 and ta083 Taillard instances. In these cases, the Mean Relative Error (*MRE*) is calculated. The RE_{avg} is defined as:

$$RE_{arg} = \sum_{i=1}^{10} \left[\frac{MK_i - UB_i}{UB_1} * 100 \right] / 10 \tag{7}$$

where *MK* is the makespan obtained by our approach, *UB* is the upper bound, $C_{max}(avg)$ is the average makespan achieved by QL-FSSP over 10 runs for each instance, and RE_{avg} is the relative error of the average makespan achieved from the best known makespan. The *MRE* takes into account the average of the results for the whole group of instances.

		C1		C1		C1		C1	
Inst.	**UB**	**Cmax (avg)**	**REavg**	**Cmax (avg)**	**REavg**	**Cmax (avg)**	**REavg**	**Cmax (avg)**	**REavg**
ta001	1278	1278.0	0.000%	1278.0	0.000%	1278.0	0.000%	1278.0	0,000%
ta007	1239	1243.2	0.339%	1247.3	0.670%	1239.0	0.000%	1241.3	0.186%
ta011	1582	1584.1	0.133%	1590.1	0.512%	1582.1	0.006%	1586.1	0.259%
ta022	2099	2107.3	0.395%	2103.4	0.210%	2101.1	0.100%	2102.1	0.148%
ta034	2751	2752.1	0.040%	2753.5	0.091%	2751.0	0.000%	2751.0	0.000%
ta052	3704	3761.3	1.547%	3764.2	1.625%	3747.2	1.166%	3747.3	1.169%
ta057	3704	3759.4	1.496%	3758.2	1.463%	3757.5	1.444%	3756.1	1.407%
ta062	5268	5268.4	0.008%	5271.3	0.063%	5268.0	0.000%	5272.2	0.080%
ta078	5617	5662.1	0.803%	5658.1	0.732%	5659.2	0.751%	5661.1	0.785%
ta083	6271	6363.3	1.472%	6362.2	1.454%	6362.7	1.462%	6365.2	1.502%
MRE			0.623%		0.682%		0.493%		0.554%

Table 1 – Q-Learning Performance under four parameters combinations

Table 2 shows that the proposed algorithm is able to obtain good results. For 30 instances, the Q-Learning algorithm matched the best-known upper bound for all runs. The average of RE for all the instances was less than 1.00 percent taking into account the UB except for the 100 × 20 instances.

Instances	**20 × 5**	**20 × 10**	**20 × 20**	**50 × 5**	**50 × 10**	**50 × 20**	**100 × 5**	**100 × 10**	**100 × 20**	**Average**
MRE (%)	0.000	0.258	0.375	0.000	0.382	0.924	0.000	0.473	1.171	0.395

instances equal	10	6	1	10	2	0	10	3	0	42
instances worse	0	4	9	0	8	10	0	7	10	48

Table 2 – Performance solutions of the QL algorithm

Table 3 shows the comparative study between our approach and the other eight algorithms.

n	m	MMAS	M-MMAS	PACO	HGA_RMA	MOACSA	NEH	PSO_{VNS}	$NEGA_{VNS}$	QL-FSSP
20	5	0.41%	0.04%	0.18%	0.04%	1.34%	3.26%	0.03%	0.00%	0.00%
	10	0.59%	0.07%	0.24%	0.02%	0.39%	4.59%	0.02%	0.01%	0.25%
	20	0.41%	0.06%	0.18%	0.05%	2.74%	3.73%	0.05%	0.02%	0.37%
50	5	0.15%	0.02%	0.05%	0.00%	0.55%	0.73%	0.00%	0.00%	0.00%
	10	2.19%	1.08%	0.81%	0.72%	0.00%	4.57%	0.57%	0.82%	0.38%
	20	2.48%	1.93%	1.41%	0.99%	0.05%	6.06%	1.36%	1.08%	0.92%
100	5	0.20%	0.02%	0.02%	0.01%	0.85%	0.65%	0.00%	0.00%	0.00%
	10	0.93%	0.39%	0.29%	0.16%	0.00%	2.18%	0.18%	0.14%	0.47%
	20	2.24%	2.42%	1.93%	1.30%	0.00%	4.28%	1.45%	1.40%	1.17%
		1.07%	0.67 %	0.57%	0.365%	0.657%	3.337%	0.407%	0.385%	0.395%

Table 3 – Comparison of QL-FSSP with other algorithms for the FSSP

As seen in Table 3, the HGA RMA, PSO_{VNS}, $NEGA_{VNS}$ and QL-FSSP perform very well in Taillard instances with mostly minor differences between them. These four algorithms manage to generate much better results than the other ones. The best and second best values of RE_{avg} are colored in the table with dark gray and light gray, respectively.

Specifically, HGA RMA has the lowest average deviation from the bestknown solutions and produces the best results for 50×5 instances. This method reached the second place in 20 × 5, 20 × 10 and 100 × 5 instances. $NEGA_{VNS}$ demonstrates the best results in 20 × 5, 20 × 10, 20 × 20, 50 × 5 and 100 × 5. PSO_{VNS} obtained the best results in 50 × 5 and 100 × 5 instances while QLFSSP has the third lowest average deviation with 0.395% with three first places (20 × 5, 50 × 5, 100 × 5) and two second places (50 × 10, 20 × 5).

In order to determine if there are statistically significant differences between the algorithms under consideration, we applied the Friedman test and the Holm procedure for the post hoc analysis. Figure 8 shows that QL-FSSP ranked again significantly better than the other algorithms, excepting $NEGA_{VNS}$.

It should be mentioned that average computational times for individual instances are not reported in Ruiz, Maroto, et al. 2006. On the other hand, Tasgetiren et al. in Tasgetiren et al. 2007, Zobolas et. al in Zobolas et al. 2009 and Betul and Mehmet in Betul and

Mehmet-Mutlu 2010 report average computational times for groups of instances which are compared to QL-FSSP in Table 4

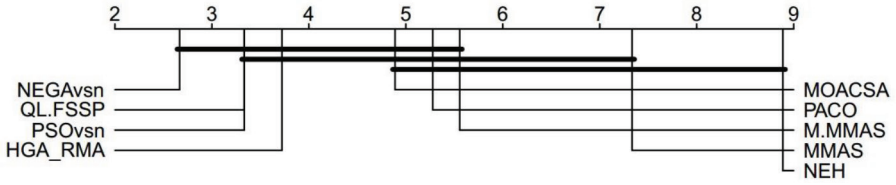

Figure 8 – Comparison of all the algorithms against each other using the Friedman test and the Bergman Hommel's procedure for the post hoc analysis. Groups of algorithms that are not significantly different (at p – value ≥ 0.05) are connected. The ranks were computed according to the Table 3.

	TAVE			
Instances	**QL-FSSP**	**NEGA$_{VNS}$**	**MOACSA**	**PSO*VNS***
ta001 - ta010	2.8	2.2	4.0	13.5
ta011 - ta020	9.1	12.2	6.0	26.3
ta021 - ta030	38.4	29.2	9.0	69.3
ta031 - ta040	21.3	8.2	34.0	2.8
ta041 - ta050	50	32.3	47.0	79.8
ta051 - ta060	100	55.0	71.0	168.1
ta061 - ta070	46.8	30.8	243.0	52.6
ta071 - ta080	100	58.7	332.0	211.0
ta081 - ta090	200	122.7	483.0	310.8
AVERAGE	63.16	39.04	137.0	103.8

Table 4 – Comparison of average running times

4. Conclusions

This paper presented a RL algorithm known as QL to solve the FSSP. The effectiveness of the proposed algorithm is evaluated by considering the benchmark problems and upper bound values for the makespan, given by Taillard. From the results we can conclude that the proposed method constitutes an interesting alternative to solve complex scheduling problems. The numerical experiments demonstrate its potential applicability and also that it is clearly similar or superior to the previously proposed metaheuristics. Finally, we want to highlight that the proposed algorithm is simple and easy to implement. It is important to mention that we are currently studying the main characteristics of the *NEGA*$_{VNS}$ algorithm, which results are slightly better than ours and a new reward function or a variable neighborhood search could be added to our learning algorithm in order to construct alternative solutions.

References

Beasley, J. E. (1990). OR-Library. Retrieved from http: //people.brunel.ac.uk/□mastjjb/jeb/info.html

Betul, Y. & Mehmet-Mutlu, Y. (2008). Ant colony optimization for multi-objective flow shop scheduling problem. *Computers & Industrial Engineering*, *54*, 411–420.

Betul, Y. & Mehmet-Mutlu, Y. (2010). A multi-objective ant colony system algorithm for flow shop scheduling problem. *Expert Systems with Applications*, *37*, 1361–1368.

Chandrasekaran, S., Ponnambalam, S., Suresh, R., & Vijayakumar, N. (2007). Multi-objective particle swarm optimization algorithm for scheduling in flowshops to minimize makespan, total flowtime and completion time variance. In *IEEE Congress on Evolutionary Computation* (pp. 4012–4018).

Chaudhry, I. A. & Munem khan, A. (2012). Minimizing makespan for a no-wait flowshop using genetic algorithm. *Sadhana*, *36*(6), 695–707.

Dongarra, J. (2014). *Performance of various computers using standard linear equations software*. Computer Science Department, University of Tennessee, Knoxville, TN. CS-89-85.

Fonseca, Y., Martínez, A., Y. Verdecia, & Rodríguez, E. (2019). Optimization of Heavily Constrained Hybrid-Flexible Flowshop Problems using a Multi-Agent Reinforcement Learning Approach. *Revista Investigación Operacional*, *40*(1), 100–111.

Fonseca, Y. & Martínez, M. (2017). Adapting a Reinforcement Learning Approach for the Flow Shop Environment with Sequence-Dependent Setup Time. *Revista Cubana de Ciencias Informáticas*, *11*(1), 41–57.

Fonseca, Y., Martínez, M., Bermudez, J., & Méndez, B. (2015). A Reinforcement Learning Approach for Scheduling Problems. *Revista Investigación Operacional*, *36*(3), 225–231.

Fonseca, Y., Martíinez, Y., & Nowé, A. (2017). Q-Learning algorithm for m-machine, n-jobs Permutational Flow Shop Scheduling Problems to minimize makespan. *Revista Investigación Operacional*, *38*(3), 281–290.

Martínez, Y. (2012). *A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems* (PhD Thesis, Vrije Universiteit Brussel, Brussel, Belgium).

Mehmet, Y. & Betul, Y. (2014). Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, *45*, 119–135. retrieved from http://dx.doi.org/ 10.1016/j.omega.2013.07.004

Pinedo, M. (2008). *Scheduling Theory, Algorithms, and Systems* (3th) (U. Englewood Cliffs, Ed.). New Jersey: Prentice Hall Inc.

Puris, A. et al. (2007). “Two-stage ACO to solve the job shop scheduling problem.” In: Lecture and Notes in Computer Science. 4756, 447-456.

Rajendran, C. & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, *115*, 426–438.

Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *Omega—International Journal of Management Science*, *34*, 461–476.

Ruiz, R. & Moroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operation Research*, *64*, 278–275.

Seido-Nagano, M. & Hissashi-Miyata, H. (2016). Review and classification of constructive heuristics mechanisms for no-wait flow shop problem. *International Journal of Advanced Manufacturing Technology*, *86*, 2161–2174.

Stuëtzle, T. (1998). An ant approach for the flow shop problem. In V. Mainz (Ed.), *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT 98)* (Vol. 3, pp. 1560– 1564). Aachen,Germany.

Sutton, R. & Barto, A. (2016). *Reinforcement Learning (An Introduction)* (2th). Cambridge, Massachusetts: The MIT Press.

Taillard, E. (2007). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285.

Tasgetiren, M. F., Liang, Y. C., Sevkli, M., & Gencyilmaz, G. (2011). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, *177*, 1930–1947.

Tavares-Neto, R. & Godinho-Filho, M. (2005). An ant colony optimization approach to a permutational flowshop scheduling problem with outsourcing allowed. *Computers & Operations Research*, *38*, 1286–1293. retrieved from http://doi:10.1016/j.cor.2010.11.010

Wu, T., Ye, N., & Zhang, T. (2003). Comparison of distributed methods for resource allocation. *International Journal of Production Research*, *43*(3), 515–536.

Yamada, T. (2009). *Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems* (Tesis Doctoral, Kyoto University, Kyoto, Japan).

Zhang, Y., Li, X., & Wang, Q. (2009). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, *196*, 869– 876.

Zobolas, G., C.D. Tarantilis, & Ioannou, G. (2009). Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, *36*, 1249–1267.