Q-learning algorithm performance for m-machine, n-jobs flow shop scheduling problems to minimize makespan

Q-LEARNING ALGORITHM PERFORMANCE FOR M-MACHINE, N-JOBS FLOW SHOP SCHEDULING PROBLEMS TO MINIMIZE MAKESPAN

Yunior César Fonseca-Reyna[*1], Yailen Martínez-Jiménez[**], Ann Nowé[***]
[*]Universidad de Granma, Bayamo, Granma, Cuba,
[**]Universidad Central de las Villas, Santa Clara, Villa Clara, Cuba
[***]Vrije Universiteit Brussel, Brussel, Belgium
Departamento de Informática, Universidad de Granma, Km 18 ½, Carretera Manzanillo, Bayamo, Granma, Cuba

ABSTRACT
Flow Shop Scheduling Problems circumscribes an important class of sequencing problems in the field of production planning. The problem considered here 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 completion time of all jobs, known in literature as makespan or C_{max}. This problem is as NP-hard, it is typical of combinatorial optimization and can be found in manufacturing environments, where there are conventional machines-tools and different types of pieces which share the same route. The following research presents a Reinforcement Learning algorithm known as Q-Learning to solve problems of the Flow Shop category. This algorithm is based on learning an action-value function that gives the expected utility of taking a given action in a given state where an agent is associated to each of the resources. To validate the quality of the solutions, test cases of the specialized literature are used and the results obtained are compared with the reported optimal results.

KEYWORDS: Flow-shop, makespan; optimization; scheduling, q-learning.

MSC: 68T20, 68T05, 90C59

RESUMEN
El problema de secuenciamiento de tareas es un problema clásico de la programación de trabajos que puede presentarse en diferentes situaciones reales. La solución de este problema consiste en encontrar una secuencia de tareas que emplee un tiempo mínimo de procesamiento (makespan). El mismo está incluido dentro de la gran variedad de problemas de planificación de recursos, el cual como muchos otros en este campo, es de difícil solución y está clasificado técnicamente como de solución en un tiempo no polinomial (NP-hard). Este problema es típico de la optimización combinatoria y se presenta en talleres con tecnología de maquinado donde existen máquinas-herramientas convencionales y se fabrican diferentes tipos de piezas que pueden, en dependencia del escenario, presentar una misma ruta o no. En esta investigación se presenta el algoritmo Q-Learning del Aprendizaje Reforzado para resolver problemas del tipo Flow Shop. Para validar el rendimiento de este algoritmo se utilizan problemas de la literatura especializada que se encuentran disponibles en la librería de investigación de operaciones. Los resultados obtenidos son comparados con los resultados óptimos reportados.

1. INTRODUCTION

Scheduling is a very active field with a high practical relevance. For a long time, manufacturing environment have been known for requiring distributed solution approaches in order to find high-quality solutions, because of their intrinsic complexity and, possibly due to an inherent distribution of the tasks that are involved[45]. This is a decision making process that is used on a regular basis in every situation where a specific set of tasks has to be performed on a specific set of resources. Practical machine scheduling problems are numerous and varied. They arise in diverse areas such as flexible manufacturing systems, production planning, computer design, logistics, comunication, etc. where the schedule construction process plays an important role, as it can have a major impact on the productivity of the company. A scheduling problem is to find sequences of jobs on given machines with the objective of minimising some function of the job completion times[34, 23]. Manufacturing scheduling is defined as an optimization process that allocates limited manufacturing resources over time among parallel and sequential manufacturing activities. This allocation must obey a set of constraints that reflect the temporal relationships between activities and the capacity limitations of a set of shared resources.
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

[1] , e-mail: fonseca@udg.co.cu

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 paper, we focus on manufacturing scheduling where all jobs share the same route, specifically the Flow Shop Scheduling (FSSP) which have been extensively studied due to their application in industry. This problem is typical of combinatorial optimization and can be found in manufacturing environments, where there are conventional machines-tools and different types of pieces which share the same route.

2. LITERATURE REVIEW

The scheduling literature is abundant whit solutions procedures for the general flow shop scheduling problem for developing permutation schedules to minimize the makespan or another criteria. Ruiz and Moroto[31], and Mehmet and Betul[19] have presented an extensive review and evaluation of many exact methods, approximation methods, heuristics and meta-heuristics for the permutation flow shop scheduling problem with the makespan criterion.
Due to the NP-hard [13, 8, 2] nature of the problem, most of the solution procedures employ heuristic approaches to obtain near-optimal sequences in reasonable time. There are many various methods for an approximation of the optimal solution by searching only a part of the space of feasible solutions (represented here by all permutations). For complex combinatorial problems, stochastic heuristic techniques are frequently used.
In 1954 Johnson presented an algorithm that yielded optimum sequencing for an n-job, 2-machine problem [14]. Researchers have tried to extend this notorious result to obtain polynomial time algorithms for more general cases [15, 5, 16, 38]. Other outhors proposed a mathematical models for flow shop scheduling based on a mixed integer programming model [34, 27].
Ancâu[2] proposed two variants of heuristic algorithms to solve the classic FSSP. Both algorithms are simple and very efficient. First algorithm is a constructive heuristic based on α-greedy selection, while the second algorithm is a modified version of the previous, based on iterative stochastic start. The numerical results show the good position of the proposed algorithms within the top known as best heuristic algorithms in the field.
Framinan et al. [11] proposed two heuristics based on the NEH heuristic[22] for the *m*-machine FSSP problem to minimize makespan and flowtime. The proposed heuristics were evaluated and found to be better than existing heuristics.
Branch-and-bound(B&B) technique can find optimal solution but at a very high computational cost and therefore cannot attempt very large problems. This algorithm can be used to find optimal solutions for small size flow shop problems. Some author applied B&B, for example Peter Bruker in your PhD thesis. He presented a method based in branch and bound techniques to solve general scheduling problems, where find a factible solutions to the FSSP [7].
Nagar et al. [21] proposed a B&B procedure for the 2-machine flow shop problem to minimize a weighted sum of flow time and makespan. They also presented a greedy algorithm for the upper bound for the B&B algorithm. The B&B method can be used as a preceding algorithm to a heuristic in order to obtain an initial solution.
Sayin and Karabati [33] presented a B&B algorithm for a 2-machine flow shop with makespan and flowtime objectives. The algorithm obtained all of the efficient solutions to the problem.
In recent years, metaheuristic approaches such as simulated annealing (SA), tabu search (TS), genetic algorithms (GA) are very desirable to solve combina-torial optimization problems regarding to their computational performance. As considering the recent studies for the flow shop scheduling problem, it is obvious that the solution methods based on metaheuristic approach are frequently proposed.
Takeshi Yamada[46] applied AG, SA and TS to the jobshop scheduling problem (and the flowshop scheduling problem as its special case) which is among the hardest combinatorial optimization problems. The author demostrated that the research in this dissertation help advance in the understanding of this significant field.
Ling Wang et. al[17] proposed an hybrid genetic algorithm (HGA) for permutation flow shop scheduling with limited buffers where multiple genetic operators based on evolutionary mechanism are used simultaneously, and a neighborhood structure based on graph model is employed to enhance the local search. The result obtained were compared with SA and TS results and demostrated the effectiviness of HGA.
Varadharajan and Rajendran[42] presented a simulated annealing algorithm for the *m*-machine flow shop problem with the objectives of minimizing makespan and total flowtime. Two variants of the proposed simulated annealing algorithm, with different parameter settings, were shown to out perform four previous multi- objective flow shop scheduling algorithms.
Nagar et. al[21]. combined the B&B procedure with a GA to find approximate solutions to the objective function made of the weighted sum of average flowtime and makespan for the 2-machine problem.

Other researchers apply these metaheuristics where obtained good solutions[28, 17, 32, 1, 47, 9, 10].
The Ant colony optimization (ACO) approach has been used to solve combinatorial optimization problems. Xiangyong Li et. al[16] compared three different mathematical formulations and propose an ACO based metaheuristic to solve this flow shop scheduling problem where demostrated that this metaheuristic is computationally efficient. Other authors applied this technique to minimizing the makespan or another objectives in a permutational flowshop environment and tested with well-known problems in literature [26, 5, 6, 38].
Tasgetiren et. al investigated a Particle Swarm Optimization algorithm(PSO), called PSOvns and HCPSO respectively, which found many best solutions for the first 90 Taillard benchmark instances[36, 37]. On the other hand, Quan-Ke[24] applied a discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem.
Rahimi-Vahed and Mirghorbani[25] studied a PSO approach with the objectives of weighted mean completion time and weighted mean tardiness. The proposed multi-objective particle swarm algorithm was compared with a multi-objective genetic algorithm. The proposed algorithm out-performed the multi-objective genetic algorithm on some specific performance metrics.
Based on idea of adaptative learnig, Anurag Agarwal et. al[3] proposed an improvement-heuristic approach for the general flow-shop problem. This approach employs a one-pass heuristic to give a good starting solution in the search space and uses a weight parameter to perturb the data of the original problem to obtain improved solutions. This algorithm obtained good solution for several benchmark problem sets.

3. FLOW SHOP SCHEDULING

The FSSP is one of the most important problems in the area of production management[8]. It can be briefly described as follows: There are a set of ***m*** machines and a set of ***n*** jobs. Each job comprises a set of ***m*** operations which must be executed on different machines. All the jobs have the same processing order when passing through the machines. There are no precedence constraints among operations of different jobs. Operations cannot be interrupted and each machine can process only one operation at a time. The problem is to find the job sequences on the machines that minimize the makespan, which is the maximum completion time of all the operations. The flow shop scheduling problem is NP-complete and thus it is usually solved by approximation or heuristic methods [39, 40, 1].
The problem investigated in this paper is conventionally given the notation ***n|m|p|C***$\boldsymbol{n|m|p|C_{max}}$ [28] and is defined as follows:

- 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 machines are sequence independent and are included in processing times.
- The machines are continuously available.

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 $\{J_1, J_2, \ldots, J_n\}$, then we calculate the completion times $C(J_i, j)$ as follows:
$C(J_1, 1) = p(J_1, 1)$
$C(J_i, 1) = C(J_{i-1}, 1) + p(J_i, 1) \quad for\ i = 2, \ldots, n$
$C(J_1, j) = C(J_1, j-1) + p(J_1, j) \quad for\ j = 2, \ldots, m$
$C(J_i, j) = max\{C(J_{i-1}, j), C(J_i, j-1) + p(J_i, j)\} \quad for\ i = 2, \ldots, n;\ for\ j = 2, \ldots, m$
$C_{max} = C(J_n, m)$

In other words, $\boldsymbol{C_{max}}$ is the time of the last operation in the last machine [29, 30].

4. REINFORCEMENT LEARNING AND MULTI-AGENT SYSTEMS

The ideas involved in Reinforcement Learning (RL) were originally developed by Sutton and Barto [35] and applied to topics of interest to researchers in Artificial Intelligence. 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 [18].

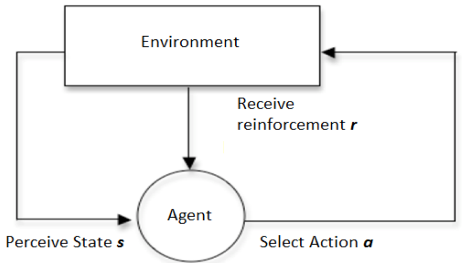

Figure 1. The standard reinforcement learning model.

A typical type of the environment is one that possesses the Markov property. In such an environment, what will happen in the future depends on the current state of the environment and the action and only on this. Most reinforcement learning researchers have been focusing on learning in this type of environment, coming up with a number of important reinforcement learning methods such as the Q-Learning algorithm [43, 44].

One of the challenges that arise in reinforcement learning and not in other kinds of learning is the trade-off between exploration and exploitation. To obtain a high reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate its expected reward. Proper control of the tradeoff between exploration and exploitation is important in order to construct an efficient learning method.

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 $\mathbb{R}$.
- 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 *s*at time t.

The reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state. A RL agent's sole objective is to maximize the total reward it receives in the long run. The reward function defines which the good and bad events are for the agent. Besides reinforcement learning (RL), intelligent agents can be designed by other paradigms, notably planning and supervised learning, but there exist some differences between these approaches. In general, planning methods require an explicit model of the state transition δ(s, a). Given such a model, a planning algorithm can search through the state-action space to find an action sequence that will guide the agent from an initial state to a goal state. Since planning algorithms operate using a model of the environment, they can backtrack or “undo” state transitions that enter undesirable states. In contrast, RL is intended to apply to situations in which a sufficiently tractable action model does not exist. Consequently, an agent in the RL paradigm must actively explore its environment to observe the effects of its actions. Unlike planning, RL agents normally cannot undo state transitions. Of course, in some cases it may be possible to build up an action model through experience [35], enabling more planning as experience accumulates.

So basically there are two approaches:

– Model based approach: learn the model, and use it to derive the optimal policy.
– Model free approach: derive the optimal policy without learning the model.

Agents can also be trained through supervised learning. In supervised learning, the agent is presented with examples of state-action pairs, along with an indication that the action was either correct or incorrect. The goal in supervised learning is to induce a general policy from the training examples. Thus, supervised

learning requires an oracle that can supply correctly labeled examples. In contrast, RL does not require prior knowledge of correct and incorrect decisions. RL can be applied to situations in which rewards are sparse, for example, rewards may be associated only with certain states. In such cases, it may be impossible to associate a label of correct or incorrect on particular decisions without reference to the agent's subsequent decisions, making supervised learning infeasible[20].
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[20].
In this paper, we will first describe the QL algorithm and then apply it to the solution of the $\boldsymbol{n|m|p|C_{max}}$ sequencing problem. To validate the quality of the solutions, the computational results will be presented and compared in terms of solution quality with test cases of the specialized literature.

5. Q-LEARNING

A well-known RL algorithm is Q-Learning (QL) [18], which works by learning an action-value function that expresses the expected utility (i.e. cumulative reward) of taking a given 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'. The update rule for the state action pair ***(s, a)*** is the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma\, max_{a'}\, Q(s',a') - Q(s,a))]$$

In this expression $\boldsymbol{\alpha} \in [0, 1]$ is the learning rate and ***r*** the reward or penalty resulting from taking action $\boldsymbol{\alpha}$ in state ***s***. The learning rate $\boldsymbol{\alpha}$ 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 restrictions [41].
Algorithm 1 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.

Algoritm 1 Q-Learning Algorithm

```
Initialize $Q$-values arbitrarily
for each episode do
    Initialize $s$
    for each episode step do
        Choose $a$ from $s$ using policy derived from $Q$(e.g., $\varepsilon$-greedy)
        Take action $a$, observe state $s'$ and $r$
        Update $Q$-value, $Q(s,a) \leftarrow Q(s,a) + \alpha\,[r + \gamma\, max_{a'}\, Q(s',a') - Q(s,a))]$
        $s \leftarrow s'$
    end for
end for
```

The agents need to balance between exploration and exploitation. The ϵ-greedy action selection method instructs the agent to follow the current policy π most of the time, but sometimes, to choose an action at random (with equal probability for each possible action ***a*** in the current state ***s***). The probability $\boldsymbol{\epsilon}$ determines when to choose a random action; this allows some balance between exploration and exploitation.

5.1. APPLYING QL TO SOLVE THE FSSP

In the FSSP all the jobs have the same processing operation order when passing through the machines. This model takes the processing times of the operations as input parameters, with the objective of finding certain job sequence that minimizes the idles time, in the long run.
To fit the QL method, it is reasonable to define states as job sequences, or more precisely job precedence relations. State-changes (or actions) are defined as changes in the relations. An action step is performed by a permutation operator, which sets up a job sequence according to precedence preferences. At the beginning no preferences are given, so states are randomly traversed. As learning proceeds, preferences are updated, which, in turn, influences the action selection policy converging to the found quasi-optimal job sequence. From this respect the learning algorithm is a directed search procedure.

In this research, we take into account ***n|m|p|C***$n|m|p|C_{max}$ where we have only one agent associated with a first resource (machine). This agent will make decisions about future actions. For this agent taking an action means deciding which job to process next from the set of currently available jobs. When a job is selected, this is processed by all the machines. The agent can select the best job taking into account the associated *q-value* (exploration), or can select one job randomly (exploration). The action selection mechanism is executed by an *ε-greedy* strategy described in [18].
In our approach, we have one agent that will execute ***n*** actions (one operation from each of the ***n*** jobs). According to [12], the set of states for the agent is defined as: $S_i = (A_i^r)$, this give raise to $|S_i| = 2^n$ local states for every agent ***i***, in our case, ***i*** = 1, which results in an upper limits of $|S_i| \leq 2^6 = 64$ possible system states if we have, for example, 6 jobs.
There are different possible feedback signals that can be used when solving a scheduling problem [18]. We are using cost as reward signal, meaning that the lower the cost the better the action, which is based on the idea that a makespan of a schedule is minimized if not many resources with queued jobs are in the system.

The proposed algorithm is summarize as:

Algorithm 2 Applying QL to solve FSSP

```
Initialize :
  $Q(s,a) = \{\}$
  $Best = \{\}$
  **for** each episode step **do**
      Initialize $s = \{\}$
      **while** not_finished(all jobs)
          Choose the jobs with the largest processing time $J_m$
          Initialize *actions* as the possible insertion set points of $J_m$ in $s$
          Choose $a$ from $s$ using policy derived from $Q$(e.g., *ε-greedy*)
          Take action $a$, observe state $s'$ and $r$ as 1/**makespan**($s'$)
              $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a'} Q(s',a') - Q(s,a))]$
            $s \leftarrow s'$
      **end while**
      **if** **makespan**($s$) < **makespan**($Best$)
          $Best \leftarrow s$
  **end for**
```

6. EXPERIMENTAL RESULTS QL-FSSP

We used the FSSP benchmark instances available in the OR-Library [4]. The OR-Library is a library of problem instances covering various Operation Research problems. There were 120 instances proposed by E. Taillard [36] grouped in 10 according to the number of jobs and machines for FSSP problem. These problems introduce the processing times for all jobs in each machines.
In order to test the algorithm, we selected randomly different cases that serve to compare the optimal results with the solutions offered in this investigation. There were 32 instances from Taillard's datasets, 4 of each of the sizes(jobs x machines) 20x5, 50x5, 100x5, 20x10, 100x10, 20x20, 50x20, 100x20. The best-known upper bounds for these problems were used for comparison purposes. We coded the Q-Learning algorithm in Java, running on a PC with Core i3 3.5 GHz CPU with 2 GB RAM.
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 [18]:

- C1: episodes $= n * m$, α = 0.1, γ = 0.8, ε = 0.2
- C2: episodes $= n * m$, α = 0.1, γ = 0.9, ε = 0.1
- C3: episodes $= n * m$, α = 0.1, γ = 0.8, ε = 0.1
- C4: 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:

- C3: episodes= $n * m$, α = 0.1, γ = 0.8, ε = 0.1

For this, we taken 10 instances from the group selected previously and we realized a nonparametric test to determinate the best combination using the results from these executions. Then, we applied Iman-

Davenport test to determinate significant differences and Holm's test to multiple comparison. The results obtained shown that C3 is the better combination. Table 1 summarized these results.

Table 1 Iman-Davenport and Holm's test results

Combination	Ranking
C3	3.15
C1	2.45
C4	2.35
C2	2.05

After this, we ran the Q-Learning algorithm *n* x *m* iterations for each Taillard's problem and we present the relative performance for the problems, in comparison with the benchmark solutions for the makespan objective. Table 2 shows the different instances for the FSSP, which was the problem selected to test the performance of the proposed approach. In this case, we selected instances conformed by different complexity problems in terms of the number of jobs and machines. This table summarizes the results for all the 32 Taillard's instances. Every test was repeated with 10 runs for each instance and the best solution was selected. Hence, there were 320 runs in total. Instances are again grouped per number of jobs and machines and the Mean Relative Error (MRE) is shown at the bottom of the results. The Relative Error (RE) is defined as:

$$RE = \left[\frac{MK - UB}{UB} * 100\right]$$

where MK is the best makespan obtained by our approach and UB is the upper bound. The MRE takes into account the average of the results for the whole group of instances.

Table 2 Q-Learning algorithm results for the FSSP

	Instance	**Optimal**	**QL-C$_{max}$**	**RE**		**Instance**	**Optimal**	**QL-Cmax**	**RE**
20x5	ta001	1278	**1278**	0,00%	20x10	ta011	1582	1583	0,063%
	ta002	1359	**1359**	0,00%		ta012	1659	1659	0,603%
	ta005	1236	**1236**	0,00%		ta017	1484	**1484**	0,000%
	ta008	1206	**1206**	0,00%		ta020	1591	1598	0,440%
MRE				**0,00%**	**MRE**				**0,276%**
20x20	ta022	2099	**2099**	0,000%	50x5	ta035	2863	**2863**	0,000%
	ta024	2223	2229	0,270%		ta036	2829	2831	0,071%
	ta027	2273	2295	0,968%		ta037	2725	2728	0,110%
	ta029	2237	2241	0,179%		ta040	2782	**2782**	0,000%
MRE				**0,354%**	**MRE**				**0,045%**
50x20	ta052	3715	3728	0,350%	100x5	ta061	5493	**5493**	0,000%
	ta053	3668	3692	0,654%		ta063	5175	5177	0,039%
	ta057	3716	3760	1,184%		ta067	5246	**5246**	0,000%
	ta059	3765	3810	1,195%		ta070	5322	**5322**	0,000%
MRE				**0,845%**	**MRE**				**0,009%**
100x10	ta071	5770	5807	0,641%	100x20	ta082	6241	6329	1,410%
	ta073	5676	5691	0,264%		ta085	6377	6457	1,255%
	ta077	5595	5634	0,697%		ta086	6437	6511	1,150%
	ta080	5845	5903	0,992%		ta089	6358	6455	1,526%
MRE				**0,648%**	**MRE**				**1,335%**
AVERAGE: 0.439									

7. RESULTS AND DISCUSSION

Based on the results from Table 2 we can see that the proposed algorithm is able to obtain good results. The average of RE for all the instances was less than 1.00 percent. For 11 instances representing a 34.38%, Q-Learning algorithm matched the best-known upper bound. For 7 set of problems, Q-Learning algorithm MRE was minor at 1.00%. Only one set of problems, MRE was larger at 1.00%. Hence, we observed that Q-Learning algorithm for large instance (case of 100x20) does not obtain the expected results.
For the test cases, we present in the Figure 2 the quality and precision of QL algorithm.

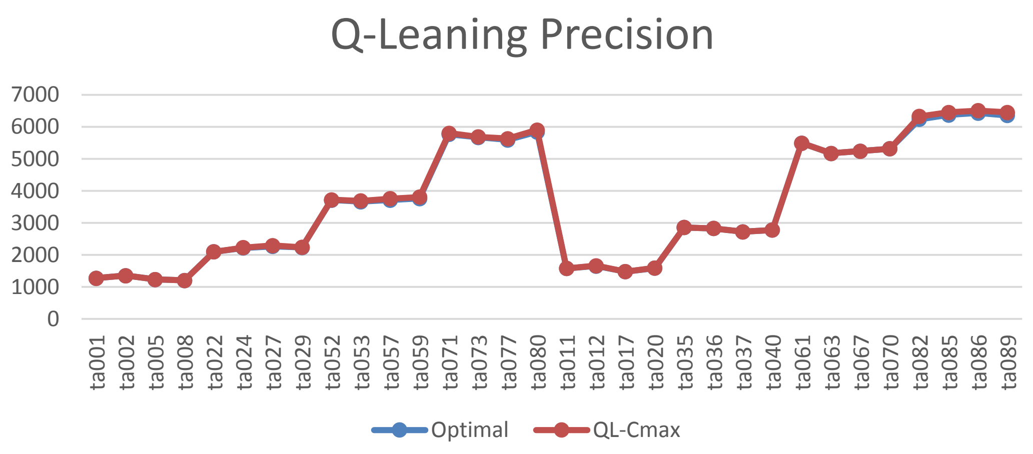

Figure 2. Q-Learning Precision

Of the previous figure, we can conclude that the accuracy and quality of obtained results by our approach are comparable to the optimum values proposed by the literature.

8. CONCLUSIONS AND FUTURE WORK

This paper presented a RL algorithm known as QL to solve the flow shop scheduling problem. 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. 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 Q-Learning algorithms, and a new reward function could be added to our learning algorithm in order to construct alternative solutions to obtained better solutions for FSSP large instances and compare with other obtained results by different approaches.

RECEIVED: MAY, 2017
REVISED: NOVEMBER, 2017

REFERENCES

[1] ÁLVAREZ, M., E. TORO and R. GALLEGO (2008):Simulated Annealing Heuristic For Flow Shop Scheduling Problems. **Scientia et Technica,** XIV, 159-164.
[2] ANCÂU, M. (2012):On Solving Flow Shop Scheduling Problems. **Proceedings of the Romanian Academy,** 13, 71-79.
[3] ANURAG, A., C. SELCUK and E. ERYARSOY. (2006):Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. **European Journal of Operational Research,** 169 801-815.
[4] BEASLEY, J. E. (1990): OR-Library. Disponible en http://people.brunel.ac.uk/~mastjjb/jeb/info.html. **Consulted** January 14, 2014.
[5] BETUL, Y. and Y. MEHMET MUTLU (2008):Ant colony optimization for multi-objective flow shop scheduling problem. **Computers & Industrial Engineering,** 54, 411-420.
[6] BETUL, Y. and Y. MEHMET MUTLU (2010):A multi-objective ant colony system algorithm for flow shop scheduling problem. **Expert Systems with Applications,** 37 1361-1368.
[7] BRUCKER, P. (2007): **Scheduling Algorithms**. Springer-Verlag, Berlin.
[8] ČIČKOVÁ, Z. and S. ŠTEVO (2010):Flow Shop Scheduling using Differential Evolution. **Management Information Systems,** 5, 008-013.
[9] CHAUDHRY, I. A. and A. MUNEM KHAN (2012):Minimizing makespan for a no-wait flowshop using genetic algorithm. **Sadhana,** 36, 695-707.
[10] FONSECA, Y., Y. MARTÍNEZ, A. E. FIGUEREDO and L. A. PERNÍA (2014):Behavior of the main parameters of the Genetic Algorithm for Flow Shop Scheduling Problems. **Revista Cubana de Ciencias Informáticas,** 8, 99-111.

[11] FRAMINAN, J. M., R. LEISTEN and R. RUIZ-USANO (2002):Efficient heuristics for flowshop sequencing with objectives of makespan and flowtime minimization. **European Journal of Operational Research,** 141, 561-571.

[12] GABEL, T. and M. RIEDMILLER (2007): On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. **IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning**. **Honolulu, USA.** I. Press

[13] GAREY, M. R., D. S. JOHNSON and R. SETHI (1976):The Complexity of Flowshop and Jobshop Scheduling. **Mathematics of Operations Research,** 1, 117-129.

[14] JOHNSON, S. M. (1954):Optimal two and three stage production schedules with setup times included. **Naval Research Logistics Quarterly,** 1, 402-452.

[15] KUBIAK, W., J. BLAZEWICZ, P. FORMANOWICZ and G. SCHMIDT (2002):Two-machine flowshop with limited machine availability. **Eur. J. Oper. Res,** 136, 528-540.

[16] LI, X., M. F. BAKI and Y. P. ANEJA (2011):Flow shop scheduling to minimize the total completion time with a permanently present operator: Models and ant colony optimization metaheuristic. **Computers & Operations Research,** 38, 152-164.

[17] LING WANG, L., L. ZHANG and D.-Z. ZHENG (2006):An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. **Computers & Operations Research,** 33 2960 - 2971.

[18] MARTÍNEZ, Y. (2012): **A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems**. PhD Thesis, Vrije Universiteit Brussel, 169 p.

[19] MEHMET, Y. and Y. BETUL (2014):Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. **Omega,** 45 119-135.

[20] MORIARTY, D., A. SCHULTZ and J. GREFENSTETTE (1999):Evolutionary Algorithms for Reinforcement Learning. **Journal of Artificial Intelligence Research,** 11, 241-276.

[21] NAGAR, A., S. HERAGU and J. HADDOCK (1995): A branch and bound approach for two-machine flowshop scheduling problem. **Journal of the Operational Research Society,** 46, 721-734.

[22] NAWAZ, M., E. ENSCORE and I. HAM (1983):A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. **OMEGA - The International Journal of Management Science,** 11, 91-95.

[23] PINEDO, M. (2008): **Scheduling Theory, Algorithms, and Systems**. Prentice Hall Inc., New Jersey.

[24] QUAN-KE, P., M. T. FATIH and L. YUN-CHIA (2008):A discrete particle swarm optimization algorithm for the no -wait flowshop scheduling problem. **Computers and Operations Research,** 35 2807-2839.

[25] RAHIMI-VAHED, A. and M. SM. (2007):A multi-objective particle swarm for a flowshop scheduling problem. **Journal of Combinatorial Optimization,** 13, 79-102.

[26] RAJENDRAN, C. and H. ZIEGLER (2004):Ant-colony algorithms for permutation flowshop scheduling to minimize makespan_total flowtime of jobs. **European Journal of Operation Research,** 115, 426-438.

[27] RAMEZANIAN, R., M. B. ARYANEZHAD and M. HEYDAR (2010):A Mathematical Programming Model for Flow Shop Scheduling Problems for Considering Just in Time Production. **International Journal of Industrial Engineering & Production Research,** 21, 97-104.

[28] REEVES, C. R. (1995): A genetic algorithm for flowshop sequencing. **Computers & Operations Research.,** 22, 5-13.

[29] RÍOS-MERCADO, Z. (1999): An enhanced TSP based heuristic for makespan minimization in a Flowshop with setup times. **Journal of Heuristics,** 5, 57-74.

[30] RÍOS-MERCADO, Z. (2001):Secuenciando óptimamente líneas de flujo en sistemas de manufactura. **Revista de Ingenierías,** IV, 48-67.

[31] RUIZ, R. and C. MOROTO (2005):A comprehensive review and evaluation of permutation flowshop heuristics. **European Journal of Operation Research,** 64, 278-275.

[32] SADEGHEIH, A. (2006):Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. **Applied Mathematical Modelling,** 30, 147-154.

[33] SAYIN, S. and S. KARABATI (1999):A bicriteria approach to the two-machine flowshop scheduling problem. **European Journal of Operational Research,** 112, 435-449

[34] ŠEDA, M. (2007):Mathematical Models of Flow Shop and Job Shop Scheduling Problems. **World Academy of Science, Engineering and Technology,** 1, 122-127.

[35] SUTTON, R. and A. BARTO (1998): **Reinforcement Learning (An Introduction)**. The MIT Press, Cambridge, Massachusetts.

[36] TAILLARD, E. (1993):Benchmarks for basic scheduling problems. **European Journal of Operational Research,** 64, 278-285.

[37] TASGETIREN, M. F., Y. C. LIANG, M. SEVKLI and G. GENCYILMAZ (2007):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.

[38] TAVARES-NETO, R. F. and M. GODINHO-FILHO (2011):An ant colony optimization approach to a permutational flowshop scheduling problem with outsourcing allowed. **Computers & Operations Research,** 38, 1286-1293.

[39] TORO, M., G. RESTREPO and M. GRANADA (2006): Adaptación de la técnica de Particle Swarm al problema de secuenciación de tareas. **Scientia et Technica UTP,** XII, 307-313.

[40] TORO, M., G. Y. RESTREPO and E. M. GRANADA (2006b):Algoritmo genético modificado aplicado al problema de secuenciamiento de tareas en sistemas de producción lineal - Flow Shop. **Scientia et Technica,** XII, 285-290.

[41] TSITSIKLIS, J. (1994):Asynchronous stochastic approximation an Q-learning. **Machine Learning,** 16, 185-202.

[42] VARADHARAJAN, T. and C. RAJENDRAN (2005):A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. **European Journal of Operational Research,** 167, 772-795.

[43] WATKINS, C. (1989): **Learning from delayed rewards**. PhD Thesis, University of Cambridge, 152 p.

[44] WATKINS, C. and P. DAYAN (1992):Technical Note: Q-Learning,. **Machine Learning,** 8, 279-292.

[45] WU, T., N. YE and T. ZHANG (2005):Comparison of distributed methods for resource allocation. **International Journal of Production Research,** 43, 515-536.

[46] YAMADA, T. (2003): **Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems**. Tesis Doctoral, Kyoto University, 120 p.

[47] ZHANG, Y., X. LI and Q. WANG (2009):Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. **European Journal of Operational Research,** 196, 869-876.