Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

Tipo de artículo: Artículo original
Temática: Inteligencia artificial
Recibido: 30/11/2015 | Aceptado: 10/10/2016

Adapting a Reinforcement Learning Approach for the Flow Shop Environment with sequence-dependent setup time

Adaptación de un Algoritmo del Aprendizaje Reforzado para el Flow Shop con tiempos de configuración

Yunior César Fonseca-Reyna[1*], Yailen Martínez-Jiménez[2]

[1] Departamento de Informática, Universidad de Granma, Km 18 ½ Carretera Manzanillo, Bayamo, Granma, Cuba, fonseca@udg.co.cu

[2] Departamento de Ciencia de la Computación, Universidad Central de las Villas, Carretera a Camajuaní Km 5 ½, Santa Clara, Villa Clara, Cuba, e-mail: yailenm@uclv.edu.cu

[*] Author for correspondence: fonseca@udg.co.cu

Abstract

The tasks scheduling problem on linear production systems, Flow Shop Scheduling Problems, has been a great importance in the operations research which seeks to establish optimal job scheduling in machines within a production process in an industry in general. 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 Cmax. Furthermore, its considerate setup-time between two jobs and initial preparation times of machines. 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. In this paper presents an adaptation of 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. Finally, the algorithm is tested with problems of different levels of complexity in order to obtain satisfactory results in terms of solutions quality.

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

Resumen

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

El *Flow Shop Scheduling* es un problema de optimización que se presenta con frecuencia en sistemas de producción convencionales automatizados. Este es un problema común donde está involucrada la toma de decisiones con respecto a la mejor asignación de recursos a procesos de información en los cuales se tienen restricciones de temporalidad. 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 tienen en común una misma ruta. En este artículo se presenta una adaptación de un enfoque del Aprendizaje Reforzado conocido en la literatura como Q-Learning para resolver problemas de scheduling de tipo *Flow Shop* con tiempos de configuración entre trabajos y tiempos iniciales de preparación de las máquinas, teniendo como objetivo minimizar el tiempo de finalización de todos los trabajos, conocido en la literatura como *makespan* o Cmax. Por último, se presentan casos de pruebas para comprobar la validez de dicha adaptación de este algoritmo al problema de secuenciación de tareas.

Palabras claves: Aprendizaje reforzado, flow-shop, makespan, optimización, secuenciación.

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 (Akhshabi y Khalatbari, 2011; Wu, et al., 2005). 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 (Pinedo, 2008; Šeda, 2007). 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 dealing with a shop problem, and depending on the constraints it presents, it can be classified as Open Shop, Job Shop, Flow Shop, etc (Brucker, 2007; Doulabi, et al., 2010; Seido Naganoa, et al., 2012).

In this research we focus on manufacturing scheduling where all jobs share de 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.

The scheduling literature is abundant with solutions procedures for the general flow shop scheduling problem for developing permutation schedules to minimize the makespan or another criteria. Ruiz and Moroto (Ruiz y Moroto, 2005), and Mehmet and Betul (Mehmet y Betul, 2014) have presented an extensive review and evaluation of many exact methods, approximation methods, heuristics and meta-heuristics for the flow shop scheduling problem with the makespan criterion.

Due to the NP-hard (Ancâu, 2012; Čičková y Števo, 2010; Garey, et al., 1976) 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 (Johnson, 1954). Researchers have tried to extend this notorious result to obtain polynomial time algorithms for more general cases (Betul y Mehmet Mutlu, 2008; Kubiak, et al., 2002; Li, et al., 2011; Tavares-Neto y Godinho-Filho, 2011). Other outhors proposed a mathematical models for flow shop scheduling based on a mixed integer programming model (Ramezanian, et al., 2010; Šeda, 2007).

Ancâu(Ancâu, 2012) 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. (Framinan, et al., 2002) proposed two heuristics based on the NEH heuristic(Nawaz, et al., 1983) 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 (Brucker, 2007).

Nagar et al. (Nagar, et al., 1995) 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 (Sayın y Karabatı, 1999) 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.

Parviz et al.(Parviz, et al., 2014) demostrate the efficience of B&B methodology. The considered objective is to minimize the completion time of all products (makespan). In this research, some lower and upper bounds are developed to increase the efficiency of the proposed algorithm.

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(Yamada, 2003) applied GA, 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(Ling Wang, et al., 2006) 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(Varadharajan y Rajendran, 2005) 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(Nagar, et al., 1995). 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 and obtained good solutions(Akhshabi y Khalatbari, 2011; Álvarez, et al., 2008; Chaudhry y Munem khan, 2012; Fonseca, et al., 2014; Ling Wang, et al., 2006; Reeves, 1995; Sadegheih, 2006; Y. Zhang, et al., 2009).

The Ant Colony Optimization (ACO) approach has been used to solve combinatorial optimization problems. Xiangyong Li et. al(Li, et al., 2011) 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 (Betul y Mehmet Mutlu, 2008, 2010; Rajendran y Ziegler, 2004; Tavares-Neto y Godinho-Filho, 2011).

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(Taillard, 1993; Tasgetiren, et al., 2007). On the other hand, Quan-Ke(Quan-Ke, et al., 2008) applied a discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Rahimi-Vahed and Mirghorbani(Rahimi-Vahed y SM., 2007) 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.

Zhang and Xiaoping(Yi Zhang y Xiaoping, 2011) applied an Hybrid Estimation of Distribution Algorithm (EDA) for permutation flow shops. This method improved 42 out of 90 current best solutions for Taillard benchmark instances.

Based on idea of adaptative learnig, Anurag Agarwal et. al(Anurag, et al., 2006) 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.

All the previous approaches focus on optimization problems that are actually a very simplified version of reality. The exclusion of real-world constraints prevent the applicability of those methods. The industry needs systems for optimized production scheduling which adjust to the conditions in the production plant and generate good solutions in a short time. In this paper we tackle a flow shop scheduling problem with sequence dependent setup time and initial preparation times of machines with the criterion of total completion time minimization. This criterion is more realistic than the more common makespan minimization, as it is known it increases productivity while at the same time it reduces the work-in-progress.

Computational Methodology

Flow Shop Scheduling Description

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

The Flow Shop Scheduling is one of the most important problems in the area of production management (Čičková y Števo, 2010). 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 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 (Álvarez, et al., 2008; Toro, et al., 2006; Toro, et al., 2006b).

The problem investigated in this paper is conventionally given the notation $\boldsymbol{n|m|p|C_{max}}$ (Reeves, 1995) 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 setup-times of the jobs on machines are considerate.
- The machines are continuously available.
- The initial preparation times of machines are considerate.

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 *r(j)* as the machine *j* preparation time, *p(i, j)* as the processing time of job *i* on machine *j*, *s(i, k, j)* as the setup-time between job *i* and job *k* 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:

$$C(J_1,1) = r(1) + p(J_1,1)$$
$$C(J_i,1) = s(J_{i-1}, J_i, 1) + C(J_{i-1},1) + p(J_i,1) \quad for\ i = 2,\ldots,n$$
$$C(J_1,j) = max\{r(j),\ C(J_1,j-1) + p(J_1,j)\} \quad for\ j = 2,\ldots,m$$
$$C(J_i,j) = max\{C(J_{i-1},j) + s(J_{i-1}, J_i, j),\ C(J_i,j-1) + p(J_i,j)\} \quad for\ i = 2,\ldots,n;\ for\ j = 2,\ldots,m$$
$$C_{max} = C(J_n,m)$$

In other words, $\boldsymbol{C_{max}}$ is the time of the last operation in the last machine (Ríos-Mercado, 1999, 2001).

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

Reinforcement Learning and Multi-Agent Systems

The ideas involved in Reinforcement Learning (RL) were originally developed by Sutton and Barto (Sutton y Barto, 1998) 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 (Martínez, 2012).

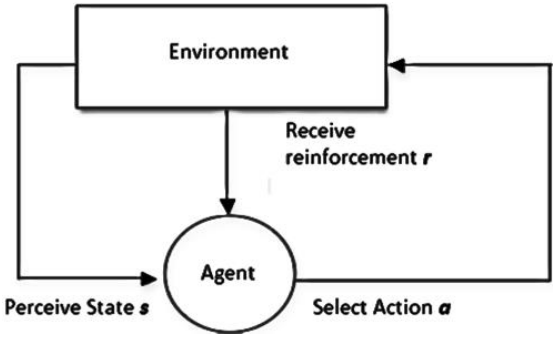

Figure 1. The standard Reinforcement Learning model.

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 (C. Watkins, 1989; C. Watkins y Dayan, 1992).

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 RL 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 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 $\delta(s, a)$. 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 (Sutton y Barto, 1998), 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

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

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 (Moriarty, et al., 1999).

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 (Moriarty, et al., 1999). In this sense, some authors have applied RL approaches to solve scheduling problems. Bert Van Vreckem et. al (Bert Van Vreckem, et al., 2013) proposed a method based on Learning Automata to solve Hybrid Flexible Flowline Scheduling Problems (HFFSP) with additional constraints like sequence dependent setup times, precedence relations between jobs and machine eligibility. Experiments on a set of benchmark problems indicate that this method can yield good results. On the other hand, Suarez (Suárez, 2010) introduce an alternative to solve the Job Shop Scheduling Problem with Parallel Machines using the QL algorithm. The results obtained by the alternative proposed are compared with the results reported by some other approaches. Bargaoui and Belkahala (Bargaoui y Belkahla, 2014) opted for a Multi-agent architecture based on cooperative behavior allied with the Tabu Search meta-heuristic to solve FSSP. The proposed approach has been tested on different benchmarks data sets and results demonstrate that it reaches high-quality solutions.

In this paper QL algorithm is first described and then applied to the solution of the $\boldsymbol{n|m|p|C_{max}}$ sequencing problem. In order to validate the quality of the solutions, computational results will be presented and compared with the optimum values of test problems.

Q-Learning Algorithm

A well-known RL algorithm is Q-Learning (Martínez, 2012), 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:

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

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

In this expression $\boldsymbol{\alpha} \in [0, 1]$ is the learning rate and $\boldsymbol{r}$ the reward or penalty resulting from taking action $\boldsymbol{\alpha}$ in state $\boldsymbol{s}$. 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 (Tsitsiklis, 1994).

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

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., *ε-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 $\boldsymbol{a}$ in the current state $\boldsymbol{s}$). The probability $\boldsymbol{\epsilon}$ determines when to choose a random action; this allows some balance between exploration and exploitation.

Adapting Q-Learning 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

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

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 $\boldsymbol{n|m|p|C_{max}}$ 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 (Martínez, 2012).

In our approach, we have one agent that will execute ***n*** actions (one operation from each of the ***n*** jobs). According to (Gabel y Riedmiller, 2007), 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 (Martínez, 2012). 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 :
$Q(s,a) = \{\}tten$
$Best = \{\}$
for each episode step **do**
Initialize *s* = { }
while not_finished**(**all jobs)
Choose the job 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'*)
$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
$s \leftarrow s'$
end while
if makespan(*s*) < **makespan**(*Best*)
Best ← *s*
end for

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

Computational Results

FSSP benchmark problems have been defined by several authors and widely used by many researchers in the scheduling field to test their solutions and compare them with solutions of other approaches. They are available online (Beasley, 1990; Taillard, 1993). However, there are no benchmark problems available for the flow shop scheduling problems with sequence-dependent setup time and initial preparation times of machines. For this reason, all data for the computational experiments are generated randomly. In order to test the proposed algorithm, ten different cases are used. Taking into account that the search space of the problem is *n!*, these instances were created with small dimensions in order to perform an exhaustive search in this space determining the optimal solutions and compare them with those obtained by QL algorithm. There were 10 instances. Their size are 5x3, 5x4, 5x5, 7x6, 7x7, 8x8, 9x4, 9x9, 10x8 and 10x10. We generated some random numbers to create the initial preparation time of machines and the setup-time between two jobs.

To determinate the quality of our solutions, the Relative Error (RE) is defined as:

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

Where *MK* is the best makespan obtained by our approach and *OP* is the optimum. The MRE takes into account the *RE* of the whole instances.

Tables 1, 2 and 3 shows the processing times, setup-times and initial preparation time of machines for 5x5 instance.

Table1. Processing times.

Machine	Processing time				
M0	10	11	6	8	11
M1	15	9	14	10	14
M2	12	11	9	10	6
M3	8	4	8	9	12
M4	6	6	8	6	3

Table 2. Preparation time machine.

Machine	Preparation Time
M0	9
M1	3
M2	8
M3	16
M4	23

We coded the Q-Learning algorithm in Java, running on a PC with Core i3 3.5 GHz CPU with 2 GB RAM. Figure 2 shows the solution for the instance 5x5 where C_{max} = 114. Table 5 shows the experimental results in relation to the optimal values for the instances set.

Revista Cubana de Ciencias Informáticas
Vol. 11, No. 1, Enero-Marzo 2017
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 41- 57

Table 3. Setup times between jobs per machines.

Machine	Setup-time				
M0	0	2	4	6	5
	3	0	7	9	4
	3	2	0	5	6
	2	4	6	0	7
	3	3	2	5	0

Machine	Setup-time				
M1	0	3	6	8	9
	2	0	5	4	1
	1	1	0	3	4
	4	4	5	0	7
	5	4	2	10	0

Machine	Setup-time				
M2	0	4	6	3	3
	2	0	5	7	3
	4	7	0	3	5
	8	8	5	0	12
	4	4	3	10	0

Machine	Setup-time				
M3	0	2	4	6	5
	3	0	7	9	4
	3	2	0	5	6
	2	4	6	0	7
	3	3	2	5	0

Machine	Setup-time				
M4	0	4	6	3	3
	2	0	5	7	3
	4	7	0	3	5
	8	8	5	0	12
	4	4	3	10	0

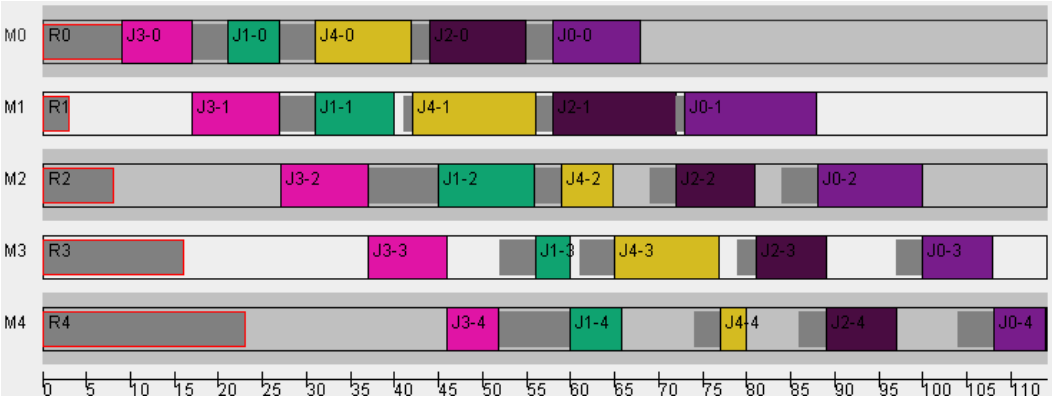

Figure 2. Gantt Diagram for instance 5x5 where C_{max} = 114

From table 5 we can see that the proposed algorithm is able to obtain good results. The algorithm obtains 5 optimal results and 5 slightly worse values for the instances set. The MRE for all instances was less than 0.03% taking into account the optimal values.

Table 5. Q-Learning algorithm results for the FSSP instances with setup-time

Instance	Optimal	QL-C_{max}	RE(%)	Instance	Optimal	QL-C_{max}	RE(%)
5x3	98	98	0.000%	**8x8**	198	208	0.050%
5x4	100	100	0.000%	**9x4**	163	176	0.079%
5x5	114	114	0.000%	**9x9**	233	233	0.000%
7x6	168	177	0.053%	**10x8**	227	241	0.057%
7x7	186	186	0.000%	**10x10**	254	260	0.024%
MRE: 0.026%							

Conclusions and Perspectives

We implemented an algorithm based on Reinforcement Learning, known as Q-Learning. This algorithm was adapted to FSSP with sequence-dependent setup-times and evaluated taking into account ten test cases of this problem. This algorithm provides good scheduling sequence FSSP for the test cases. The obtained result leads to the following conclusions:

- This approach constitutes an interesting alternative to solve complex mathematic problems.
- The Q-Learning adaptation for the FSSP with setup-time between jobs and initial preparation times of machines yielded good results taking into account the optimal values for the instances set of problems.
- It is important to mention that we are currently studying the main parameters of the QL algorithm and we can add a new reward function to our learning algorithm in order to construct alternative solutions and adapt other methods to generate initial solutions such as NEH, AG, and PSO. At the same time, we are considering other real world constraints and larger benchmarks.

References

Akhshabi, M. y Khalatbari, J. Solving flexible job-shop scheduling problem using clonal selection algorithm. Indian Journal of Science and Technology, 2011, 10(4): p. 1248-1251.
Álvarez, M.; Toro, E., et al. Simulated Annealing Heuristic For Flow Shop Scheduling Problems. Scientia et Technica, 2008, XIV(40): p. 159-164.
Ancâu, M. On Solving Flow Shop Scheduling Problems. Proceedings of the Romanian Academy, 2012, 13(1): p. 71-79.
Anurag, A.; Selcuk, C., et al. Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. European Journal of Operational Research, 2006, 169 p. 801-815.
Bargaoui, H. y Belkahla, O. Multi-Agent Model based on Tabu Search for the Permutation Flow Shop Scheduling Problem. Advances in Distributed Computing and Artificial Intelligence Journal, 2014, 3(1): p. 29-38.
Beasley, J. E. (1990). OR-Library Retrieved January 14, 2014, from http://people.brunel.ac.uk/~mastjjb/jeb/info.html
Bert Van Vreckem, B.; Borodin, D., et al. A Reinforcement Learning Approach to Solving Hybrid Flexible Flowline Scheduling Problems. En: 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA). Gent, Belgium: 2013, p. 402-409.
Betul, Y. y Mehmet Mutlu, Y. Ant colony optimization for multi-objective flow shop scheduling problem. Computers & Industrial Engineering, 2008, 54: p. 411-420.
Betul, Y. y Mehmet Mutlu, Y. A multi-objective ant colony system algorithm for flow shop scheduling problem. Expert Systems with Applications, 2010, 37 p. 1361-1368.
Brucker, P. Scheduling Algorithms. Berlin, Springer-Verlag, 2007, 378.

Čičková, Z. y Ševo, S. Flow Shop Scheduling using Differential Evolution. Management Information Systems, 2010, 5(2): p. 008-013.
Chaudhry, I. A. y Munem khan, A. Minimizing makespan for a no-wait flowshop using genetic algorithm. Sadhana, 2012, 36(6): p. 695-707.
Doulabi, S. H. H.; Jaafari, A. A., et al. Minimizing weighted mean flow time in open shop scheduling with time-dependent weights and intermediate storage cost. International Journal on Computer Science and Engineering 2010, 2(3): p. 457-460.
Fonseca, Y.; Martínez, Y., et al. Behavior of the main parameters of the Genetic Algorithm for Flow Shop Scheduling Problems. Revista Cubana de Ciencias Informáticas, 2014, 8(1): p. 99-111.
Framinan, J. M.; Leisten, R., et al. Efficient heuristics for flowshop sequencing with objectives of makespan and flowtime minimization. European Journal of Operational Research, 2002, 141: p. 561-571.
Gabel, T. y Riedmiller, M. On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. En: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. Honolulu, USA.: I. Press, 2007, p. 68-75.
Garey, M. R.; Johnson, D. S., et al. The Complexity of Flowshop and Jobshop Scheduling. Mathematics of Operations Research, 1976, 1(2): p. 117-129.
Johnson, S. M. Optimal two and three stage production schedules with setup times included. Naval Research Logistics Quarterly, 1954, 1: p. 402-452.
Kubiak, W.; Blazewicz, J., et al. Two-machine flowshop with limited machine availability. Eur. J. Oper. Res, 2002, 136: p. 528-540.
Li, X.; Baki, M. F., et al. Flow shop scheduling to minimize the total completion time with a permanently present operator: Models and ant colony optimization metaheuristic. Computers & Operations Research, 2011, 38: p. 152-164.
Ling Wang, L.; Zhang, L., et al. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Computers & Operations Research, 2006, 33 p. 2960 - 2971.
Martínez, Y. A Generic Multi-Agent Reinforcement Learning Approach for Scheduling Problems. PhD Thesis, Vrije Universiteit Brussel, Brussel, 2012.
Mehmet, Y. y Betul, Y. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. Omega, 2014, 45 p. 119-135.
Moriarty, D.; Schultz, A., et al. Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research, 1999, 11: p. 241-276.
Nagar, A.; Heragu, S., et al. A branch and bound approach for two-machine flowshop scheduling problem. Journal of the Operational Research Society, 1995, 46: p. 721-734.
Nawaz, M.; Enscore, E., et al. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. OMEGA - The International Journal of Management Science, 1983, 11(1): p. 91-95.
Parviz, F.; Seyed Mohammad, H. H., et al. A branch and bound algorithm for hybrid flow shop scheduling problem with setup time and assembly operations. Applied Mathematical Modelling, 2014, 38: p. 119-134.
Pinedo, M. Scheduling Theory, Algorithms, and Systems. New Jersey, Prentice Hall Inc., 2008, 586.
Quan-Ke, P.; Fatih, M. T., et al. A discrete particle swarm optimization algorithm for the no -wait flowshop scheduling problem. Computers and Operations Research, 2008, 35 (9): p. 2807-2839.
Rahimi-Vahed, A. y SM., M. A multi-objective particle swarm for a flowshop scheduling problem. Journal of Combinatorial Optimization, 2007, 13(1): p. 79-102.

Rajendran, C. y Ziegler, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan_total flowtime of jobs. European Journal of Operation Research, 2004, 115: p. 426-438.
Ramezanian, R.; Aryanezhad, M. B., et al. A Mathematical Programming Model for Flow Shop Scheduling Problems for Considering Just in Time Production. International Journal of Industrial Engineering & Production Research, 2010, 21(2): p. 97-104.
Reeves, C. R. A genetic algorithm for flowshop sequencing. Computers & Operations Research., 1995, 22(1): p. 5-13.
Ríos-Mercado, Z. An enhanced TSPbased heuristic for makespan minimization in a Flowshop with setup times. Journal of Heuristics, 1999, 5(1): p. 57-74.
Ríos-Mercado, Z. Secuenciando óptimamente líneas de flujo en sistemas de manufactura. Revista de Ingenierías, 2001, IV(10): p. 48-67.
Ruiz, R. y Moroto, C. A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operation Research, 2005, 64: p. 278-275.
Sadegheih, A. Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance. Applied Mathematical Modelling, 2006, 30: p. 147-154.
Sayın, S. y Karabatı, S. A bicriteria approach to the two-machine flowshop scheduling problem. European Journal of Operational Research, 1999, 112: p. 435-449
Šeda, M. Mathematical Models of Flow Shop and Job Shop Scheduling Problems. World Academy of Science, Engineering and Technology, 2007, 1(31): p. 122-127.
Seido Naganoa, M.; Almeida da Silva, A., et al. A new evolutionary clustering search for a no-wait flow shop problem with set-up times. Engineering Applications of Artificial Intelligence, 2012, 25: p. 1114–1120.
Suárez, Y. Solución al problema de secuenciación en máquinas paralelas utilizando Aprendizaje Reforzado Universidad Central de las Villas, Villa Clara, 2010.
Sutton, R. y Barto, A. Reinforcement Learning (An Introduction). Cambridge, Massachusetts, The MIT Press, 1998, 312.
Taillard, E. Benchmarks for basic scheduling problems. European Journal of Operational Research, 1993, 64(2): p. 278-285.
Tasgetiren, M. F.; Liang, Y. C., et al. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research, 2007, 177: p. 1930-1947.
Tavares-Neto, R. F. y Godinho-Filho, M. An ant colony optimization approach to a permutational flowshop scheduling problem with outsourcing allowed. Computers & Operations Research 2011, 38: p. 1286-1293.
Toro, M.; Restrepo, G., et al. Adaptación de la técnica de Particle Swarm al problema de secuenciación de tareas. Scientia et Technica UTP, 2006, XII(32): p. 307-313.
Toro, M.; Restrepo, G. Y., et al. Algoritmo genético modificado aplicado al problema de secuenciamiento de tareas en sistemas de producción lineal - Flow Shop. Scientia et Technica, 2006b, XII(30): p. 285-290.
Tsitsiklis, J. Asynchronous stochastic approximation an Q-learning. Machine Learning, 1994, 16: p. 185-202.
Varadharajan, T. y Rajendran, C. A multi-objective simulated-annealing algo-rithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. European Journal of Operational Research, 2005, 167: p. 772-795.
Watkins, C. Learning from delayed rewards. PhD Thesis, University of Cambridge, 1989.
Watkins, C. y Dayan, P. Technical Note: Q-Learning,. Machine Learning 1992, 8: p. 279-292.

Wu, T.; Ye, N., et al. Comparison of distributed methods for resource allocation. International Journal of Production Research, 2005, 43(3): p. 515-536.

Yamada, T. Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems. Tesis Doctoral, Kyoto University, Kyoto, Japan, 2003.

Zhang, Y.; Li, X., et al. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. European Journal of Operational Research, 2009, 196: p. 869-876.

Zhang, Y. y Xiaoping, L. Estimation of distribution algorithm for permutation flow shops with total flowtime minimization. Computers & Industrial Engineering, 2011, 60: p. 706-718.