A REINFORCEMENT LEARNING APPROACH FOR SCHEDULING PROBLEMS

Yunior César Fonseca Reyna[*1], Yailen Martínez Jiménez[**] , Juan Manuel Bermúdez Cabrera[**] ,.Beatriz M. Méndez Hernández[**]
[*]Universidad de Granma, Bayamo, Granma, Cuba.
[**]Universidad Central de las Villas, Santa Clara, Villa Clara, Cuba
[1]Departamento de Informática, Universidad de Granma, Km 18½ Carretera Manzanillo, Bayamo, Granma, Cuba, e-mail: fonseca@udg.co.cu

ABSTRACT
Scheduling problems are an important class of sequencing problems that can be found in many real life situations, especially in the field of production planning. The problem considered in this work is to find a permutation of operations 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 can, in some scenarios, share the same route or not. The following research presents a Reinforcement Learning algorithm known as Q-Learning to solve scheduling problems, specifically Job Shop and Flow Shop. 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 were compared with the reported optimal results.

KEYWORDS: scheduling; job shop; flow shop; reinforcement learning; multi-agent systems

MSC: 68T20, 68T05, 90C59

RESUMEN
El problema de secuenciación 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. Considerando lo anterior, en este trabajo se presenta una alternativa de solución a problemas de secuenciación, específicamente a problemas tipos Job Shop y Flow Shop utilizando el algoritmo Q-Learning del Aprendizaje Reforzado. Finalmente, se realiza un estudio experimental utilizando instancias de problemas 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 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 [11]. In this work we focus on manufacturing scheduling, where the schedule construction process plays an important role, as it can have a major impact on the productivity of the company. 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. These scheduling problems are NP-complete and they are usually solved by approximation or heuristic methods [16, 17, 1].
RL, as noted by Kaelbling, Littman and Moore in [8], dates back to the early days of cybernetics and work in statistics, psychology, neuroscience, and computer science. It has attracted increasing interest in the machine learning and artificial intelligence communities during the past fifteen years.
In this research we follow the ideas of the approach of Gabel and Riedmiller [6] and implement a RL algorithm know as Q-Learning algorithm to solve the Job Shop Scheduling Problem (JSSP) and presented a way to solve

[1] fonseca@udg.co.cu

the Flow Shop Scheduling (FSSP). Q-Learning is based on the idea that a makespan of a schedule is minimized if as few as possible resources with queued jobs are in the system. Therefore they proposed to use in the q-learning update rule as cost function the number of jobs that are in the queues in the system. This means that high costs are incurred when many jobs that are waiting for further processing are in the system and, hence, the overall utilization of the resources is poor. It is important to note here that we are talking about costs instead of rewards, then, small q-values correspond to "good" state actions pairs.
In this paper, we will describe the QL algorithm and then apply it to the solution of the JSSP sequencing problem and will recommend a way to solve the FSSP. To validate the quality of the solutions of JSSP, the computational results will be presented and compared in terms of solution quality with test cases of the specialized literature.

2. SCHEDULING PROBLEMS

2.1. Job Shop Scheduling

JSSP is a well-known manufacturing scheduling scenario, it involves a set of jobs and a set of machines with the purpose of finding the best schedule, that is, an allocation of the operations to time intervals on the machines that has the minimum duration required to complete all jobs (in this case the objective is to minimize the makespan)[7].
The *n* x *m* minimum-makespan general jobs hop scheduling problem, designated by the symbols $\boldsymbol{n|m|G|C_{max}}$ and hereafter referred to as the JSSP, can be described by a set of $\boldsymbol{n}$ jobs $\{J_i\}_{1 \le i \le n}$ which is to be processed on a set of m machines $\{M_r\}_{1 \le r \le m}$. The problem can be characterized as follows [21]:

1. Each job must be processed on each machine in the order given in a predefined technological sequence of machines.
2. Each machine can process only one job at a time.
3. The processing of job J_j on machine M_r is called the operation O_{jr}.
4. Operation O_{jr} requires the exclusive use of M_r for an uninterrupted duration p_{jr}, its processing time; the preemption is not allowed.
5. The starting time and the completion time of an operation O_{jr} is denoted as s_{jr} and c_{jr} respectively. A *schedule* is a set of completion times for each operation $\{c_{jr}\}_{1 \le j \le n;1 \le r \le m}$ that satisfies above constraints.
6. The time required to complete all the jobs is called the makespan, which is denoted as C_{max}. By definition, $C_{max} = \max_{1 \le j \le n;1 \le r \le m} c_{jr}$

The problem is "general", hence the symbol *G* is used, in the sense that the technological sequence of machines can be different for each job as implied in the first condition and that the order of jobs to be processed on a machine can be also different for each machine. The predefined technological sequence of each job can be given collectively as a matrix $\{T_{jk}\}$ in which $T_{jk} = r$ corresponds to the *k*-th operation O_{jr} of job J_i on machine M_r .
The objective of optimizing the problem is to find a schedule that minimizes C_{max}.
The total number of possible solutions for a problem with n jobs and m machines is m*n!.

2.2. Flow Shop Scheduling

FSSP is one of the most important problems in the area of production management [4, 5]. There are a set of $\boldsymbol{m}$ machines and a set of $\boldsymbol{n}$ jobs. Each job comprises a set of $\boldsymbol{m}$ operations which must be executed on different machines. The problem investigated in this paper is conventionally given the notation $\boldsymbol{n|m|p|C_{max}}$ [12] and is defined as follows:

1. All the jobs have the same processing order when passing through the machines.
2. There are no precedence constraints among operations of different jobs.
3. Operations cannot be interrupted and each machine can process only one operation at a time.
4. 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.

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, C_{max} is the time of the last operation in the last machine [13, 14, 2].

3. REINFORCEMENT LEARNING

The ideas involved in RL were originally developed by Sutton and Barto [15] and applied to topics of interest for 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 [9].

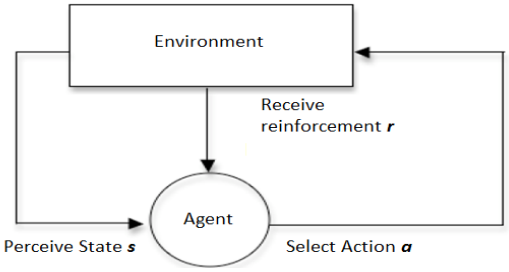

Figure 1.The standard reinforcement learning model.

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 [19, 20].

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 $\mathbb{R}$.

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 $\delta(s, a)$. 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 [15], 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 [10].
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. For example, a common RL task is robot control. Designers of autonomous robots often lack sufficient knowledge of the intended operational environment to use either the planning or the supervised learning regime to design a control policy for the robot. In this case, the goal of RL would be to enable the robot to generate effective decision policies as it explores its environment [10].

3.1. **Q-Learning**

A well-known RL algorithm is Q-Learning [9], 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 **a** 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 [18].
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.

Algorithm 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 ***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.

4. APPLYING QL TO SOLVE THE FSSP AND THE JSSP

When applying RL to solve a scheduling problem, in this case the JSSP, an agent is associated to each of the *m* resources (machines), this agent locally decides on elementary actions. For an agent taking an action means deciding which job to process next out of a set of currently waiting jobs at the corresponding resource. At each step, for selecting an action, each agent takes into account only the jobs that can be processed in that moment according to the problem constraints. Basically, we are keeping two kinds of sets; one containing the resources which are able to process some job (agents able to select an action); and the other important set is the one

containing the jobs that each agent can select as its next action. This means that each agent will have a set of this type, from which it will select, step by step, the actions to perform. This set is continuously updated; as well as the set containing the agents able to select an action, because once a job is finished in one resource, the status of the system changes.

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 $\boldsymbol{n|m|p|C_{max}}$ 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 [9]. We recommended one agent that will execute $\boldsymbol{n}$ actions (one operation from each of the $\boldsymbol{n}$ jobs).

According to [6], the set of states for the agent is defined as: $S_i = (A_i^r)^2$, this give raise to $|S_i| = 2^n$ local states for every agent $\boldsymbol{i}$, in our case, $\boldsymbol{i}$ = 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 [9]. We recommended 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.

In the JSSP, an agent $\boldsymbol{k}$ is associated to each of the $\boldsymbol{m}$ resources (machines). It is important to remember that an agent cannot take an action at discrete time step $\boldsymbol{t}$, but only after its resource has finished the execution of the current operation, because each resource can only process one operation at a time, besides, each job can only be processed by a single machine at any given time, meaning that only one of its operations can by in execution at time *t*.

At each time step, the possible actions an agent can select take the problem constraints into account, such that only operations that can be processed at that time can be chosen. Basically, we are keeping two sets; one containing the resources which are able to process an operation (agents able to select an action); and another one containing the operation that each agent can select as its next action, which is also what defines its current state.

These sets are continuously updated, because once an operation is finished on one resource, the status of the system changes. More specifically, the queue assigned to the resource where the next operation of the corresponding job will be executed is affected.

As previously mentioned, an agent is associated to each resource and it can execute $\boldsymbol{n}$ action. The set of states for every agent $\boldsymbol{i}$ is 2^{36} possible system states if we have, for example, 6 jobs and 6 machines. The action selection mechanism and feedback signal are the same that we recommended for the FSSP.

Algorithm 2 shows the pseudo code of the Q-Learning method to solve the JSSP:

Algorithm 2 Pseudo Code of the QL for the FSSP:

```
Initialize_Parameters – Number of cycles, learning rate, discount factor, epsilon, instance path;
Read_from_file – Read the data from the instance received as parameter;
Initialize_Q-values;
For 0 to number of cycles do:
        C_max = Execute_Q-Learning;
        If C_max < Best_found then Best_found = C_max;
Print_Solution;

Execute_Q-Learning {
Initialize s;
While set Possible_Agents != Ø
        For each agent in Possible_Agents
                Choose action using e-greedy policy;
```

[2] P(x) denotes the power set of x

Take action a, observe cost and s'

Update Q-value;

Update sets (Possible_Agents and the corresponding jobs in the queues)

End while;

}

5. EXPERIMENTAL RESULTS

In order to test the algorithm, we used the benchmark instances located in the OR-Library [3]. The OR-Library is a library of problem instances covering various Operation Research problems. For this, we used different cases that serve to compare the obtained results with the solution offered in this investigation.
Table 1 shows the different instances for the JSSP, 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.
Instances are grouped per number of jobs and machines. The obtained results are compared with the optimal solutions and with three other aproaches.

	Instance	Optimum	Our approach
10 x 5	la01	666	666
	la02	660	667
	la03	597	610
	la04	590	611
	la05	593	593
15 x 5	la06	926	926
	la07	890	890
	la08	863	863
	la09	951	951
	la10	958	958
20 x 5	la11	1222	1222
	la12	1039	1039
	la13	1150	1150
	la14	1292	1292
	la15	1207	1259

Table 1. Q-Learning algorithm results for the JSSP

It is possible to see that the proposed algorithm is able to obtain good results compared to the known lower bounds, obtaining in 11 cases the know optimum and very close solutions for the rest of the instances.

6. CONCLUSIONS

In this work, an algorithm based on Reinforcement Learning was implemented to solve scheduling problems. This algorithm was evaluated using scheduling benchmarks available on the internet in the OR-Library. A comparison was made in terms of solution quality for the job shop scenarios and some more experiments are being developed for the flow shop case. The obtained results lead to the following conclusions:

- This method constitutes an interesting alternative to solve complex scheduling problems.
- The Q-Learning adaptation for the JSSP yields results of excellent quality providing work sequences within the limits proposed by the literature, more experiments will be developed, mainly using the FSSP instances.

RECEIVED MARCH,2014.
REVISED JANUARY, 2015.

BIBLIOGRAPHY

[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] BEASLEY, J. E. (1990): OR-Library. Disponible en http://people.brunel.ac.uk/~mastjjb/jeb/info.html. **Consulted** January 14, 2014.
[4] BLAZEWICZ, J., K. ECKER, E. PESCH, G. SCHMIDT and J. WEGLARZ (2007): **Handbook on Scheduling from Theory to applications**. Springer-Verlag, Berlin.
[5] ČIČKOVÁ, Z. and S. ŠTEVO (2010):Flow Shop Scheduling using Differential Evolution. **Management Information Systems,** 5, 008-013.
[6] 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
[7] GONZÁLEZ, M. (2011): **Soluciones Metaheurísticas al Job-Shop Scheduling Problem with Sequence-Dependent Setup Times**. PhD. Thesis, Universidad de Oviedo, 281 p.
[8] KAELBLING, L. P., M. LITTMAN and A. MOORE (1996):Reinforcement Learning: a survey. **Journal of Artificial Intelligence Research,** 4, 237-285.
[9] MARTÍNEZ, Y. (2012): **A Generic Multi-Agent Reinforcement Learning Approach Scheduling Problems**. PhD Thesis, Vrije Universiteit Brussel, 169 p.
[10] MORIARTY, D., A. SCHULTZ and J. GREFENSTETTE (1999):Evolutionary Algorithms for Reinforcement Learning. **Journal of Artificial Intelligence Research,** 11, 241-276.
[11] PINEDO, M. (2008): **Scheduling Theory, Algorithms, and Systems**. Prentice Hall Inc., New Jersey.
[12] REEVES, C. R. (1995): A genetic algorithm for flowshop sequencing. **Computers & Operations Research.,** 22, 5-13.
[13] RÍOS-MERCADO, Z. (1999): An enhanced TSPbased heuristic for makespan minimization in a Flowshop with setup times. **Journal of Heuristics,** 5, 57-74.
[14] RÍOS-MERCADO, Z. (2001):Secuenciando óptimamente líneas de flujo en sistemas de manufactura. **Revista de Ingenierías,** IV, 48-67.
[15] SUTTON, R. and A. BARTO (1998): **Reinforcement Learning (An Introduction)**. The MIT Press, Cambridge, Massachusetts.
[16] 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.
[17] 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.
[18] TSITSIKLIS, J. (1994):Asynchronous stochastic approximation an Q-learning. **Machine Learning,** 16, 185-202.
[19] WATKINS, C. (1989): **Learning from delayed rewards**. PhD Thesis, University of Cambridge, 152 p.
[20] WATKINS, C. and P. DAYAN (1992):Technical Note: Q-Learning,. **Machine Learning,** 8, 279-292.
[21] YAMADA, T. (2003): **Studies on Metaheuristics for Jobshop and Flowshop Scheduling Problems**. PhD Thesis, Kyoto University, 120 p.