7e Conférence Internationale de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 – Paris- France
« Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination.».

EFFICIENT AND EFFECTIVE REACTIVE SCHEDULING OF MANUFACTURING SYSTEM USING SARSA-MULTI-OBJECTIVE-AGENTS

N. AISSANI*, B. BELDJILALI

Department of Computer sciences
University of Oran ES-Sénia
BP 1524 El M'naour, Oran Algérie
aissani.nassima@yahoo.com,
bouzianebeldjilali@yahoo.fr

D. TRENTESAUX

LAMIH, Department of Production Systems.
University of Valenciennes, Le mont Houy, F-59313
Valenciennes cedex 09, France.
damien.trentesaux@univ-valenciennes.fr

ABSTRACT: *The current market trends, characterized by a great competitiveness with more and more complex and contradictory constraints, make the researchers of the field conceive an adaptive control system not only able to react effectively, but also have more and more ability to adapt to the fast evolution of demand and at fixed cost. This is by using resources available as well as possible to optimize this adaptation (efficiency). This paper, presents our step to work out a heterarchical system which is more reliable insofar as we respect much kind of performances: effectiveness and efficiency. With this intention, we use a multi-agent system of which the decisions taken by the system are the result of agents group work. These agents ensure in the same time a continuously improvement of effectiveness and efficiency performances thanks to the reinforcement learning technique and particularly SARSA-Multi objective algorithm which was introduced to them. This technique of learning makes it possible to agents to learn the best behaviour in their various roles (answer the requests, self-organization, plan…) without attenuating the on-line system. A computer implementation and experimentation of this model are provided in this paper to demonstrate the contribution of our approach.*

KEYWORDS: ***effectiveness and efficiency performances, dynamic scheduling, reinforcement learning, manufacturing system modeling, multi-agents system***

1. INTRODUCTION

The current market trends, characterized by a great competitiveness with more and more complex and contradictory constraints, make the researchers of the field conceive an adaptive control system not only able to react effectively, but also have more and more ablility to adapt to the fast evolution of demand and at fixed cost. This is by using available resources as well as possibly optimizing this adaptation. In Bousbia and Trentesaux, (2002) an analysis of the whole of these points was carried out and in particular highlighted the interest to adopt an approach of self-organized and heterarchical control system. The heterarchy term describes a relation between the same hierarchical level entities (Duffie and Prabhu, 1996), initially, proposed in the medical biology field (McCulloch 1945) then experimented in several domains (Maione and Naso, 2003; Prabhu, 2003; Haruno and Kawato, 2006). This term is relatively close to the concept of "distribution" in the multi-agent field (distributed systems). However, from our point of view, the concept of distribution of the decision capacities does not mean which multi-agent system is organized in a heterarchical way (even if it is often the case). However it is the assumption that we wish to make in this paper. From our point of view, this assumption is justified by the dynamics and the volatility of information which made the approach purely hierarchical and partially unsuited in the objective previously announced (Bousbia and Trentesaux, 2002).

The objective is thus to obtain not only an effective operation (reactivity) but also efficient in the direction where the real potentialities of production resources are taken into account. This is in order to improve continuously the already effective solutions. According to Senechal (2004) definitions of performances, the effectiveness is the adequacy of results and objectives can be very often evaluated thanks to quality indicators. And efficiency is the adequacy of the means and results, its deal with the system output. This performance is primarily judged during system run phase. It generates control or management decision. Few works were interested in efficiency (Butalaand and Sluga, 2002; Denkena *et al.* 2006) and even less the combination of both of performances effectiveness/efficiency particularly in reactive and flexible manufacturing systems. However, the relationship of influence may exist between some effectiveness and efficiency indicators.

In this paper we will focus on effectiveness and efficiency performances in an adaptive system based on reactive scheduling approach.

We will start first of all introducing the problems. Following the state of the art analysis, we will present the adopted modelling approach and the learning technique which we used. We will then propose our system model. Lastly, we will present our models, implementation and an experimentation, from which an analysis will be carried out. Then, a conclusion summarizes our contribution and presents a certain number of prospects.

2. BACKGROUND

In this part we propose an analysis of the state of the art in the field of scheduling problems in order to not only propose an effective solution (having a minimal level of performances) but also efficient (able to improve this constant level with means) and highlight some points that can converge. The importance of machine learning integration in the scheduling of production will be highlighted

1.1 Dynamic scheduling

In manufacturing control, scheduling constitutes the most important function. And, in this paper, we focus on dynamic scheduling.

Csaji and Monostori (2006) classify dynamic scheduling into three categories:
If the solution aims at generating the resource allocation off-line in advance, then it is called *predictive*. Thus, predictive solutions assume a deterministic environment. When it is an unforeseeable environment there are some data (e.g., the actual durations) that will only be available during the plan execution. According to the usage of this information, two basic types of solution techniques will be identified. A solution which will consider environment uncertainties is called *proactive*. A proactive solution allocates the operations to resources and defines the orders of the operations without precise starting times, because the durations are uncertain. This kind of technique can be applied when only the durations of the operations are stochastic, but, the states of the resources are known perfectly (e.g. stochastic job-shop scheduling) (Bidot *et al.* 2007).

Finally, On-line scheduling is called *reactive*. The resource allocation process actually evolves so more information becomes available in order to make decision in real-time (Trentesaux *et al.* 2001; Pujo and Brun-Picard, 2002; Csaji and Monostori, 2006; Aissani *et al.* 2008). Naturally, a reactive solution is not a simple objective function, but instead a resource allocation policy (a mapping from states to actions) which controls the process. In this paper we focus on reactive solutions only.

1.2 Reinforcement learning to improve performances

These last decades, the researchers in scheduling were inspired by the artificial intelligence after the methods used were exclusively based on operation research algorithms with exponential complexity. Taking into account several types of performances (effectiveness and efficiency) which means optimization of several criteria will increase even more the problem complexity. The artificial intelligence has allowed to solve these problems by giving satisfactory solutions even if not always most optimal.

Maione and Naso, (2003) used genetic algorithms to adapt the decision strategies of autonomous controllers. The control agents use pre-assigned decision rules only for a limited amount of time, and obey a rule replacement policy propagating the most successful rules to the subsequent populations of concurrently operating agents. It can not be a solution for reactive scheduling. So, we need to integrate a reactive technique which allows us to control system in real time. Reinforcement learning may be a suitable technique for giving quasi-real-time solutions and improve them over time.

Reinforcement learning is learning to act by trial and error. In this paradigm, an agent can perceive its state and perform actions. After each action, a numerical reward is given. The goal of the agent is to maximize the total reward it receives over time.

Katalinic and Kordic (2004) used reinforcement learning in the optimization of the resources use in a very expensive production of electric motors. These Systems are characterized by their varieties of products (produced upon request) that require a great flexibility and adaptability. Consequently, the units of assembly must be autonomous and modular, from where the difficulty of performance control and development. Katalinic and Kordic (2004) considered these units as a colony of insects able to organize themselves to carry out a task. This can reduce the number of used resources, to solve more easily the problems involved in production risks.

The most used reinforcement learning algorithm is Q-learning, Wei and Zhao (2005), extended it by using reward function based on scheduling criteria EMLT (The Estimated Mean LaTeness) which is an effective criteria without taking care about efficiency. Aydin and Özteme (2000) proposed an intelligent agent-based scheduling system. They employed Q-III to dynamically select dispatching rules. Their state determination criteria consist of the mean slack time of the queue and the buffer size of the machine. They take advantage of domain knowledge and experience in the learning process. Therefore, the principal learning mechanism of RL (Reinforcement Learning) is omitted.

Almost of those works resolve resource allocation satisfying criteria which are related to effectiveness and omit efficiency. In this paper, Our approach deals with both of effectiveness and efficiency performances by developing a system which gives an optimal scheduling solution using in best all resources and minimizing the reiteration of job requests.

In the next section, we will present our system architecture.

2 THE PROPOSED CONTROL SYSTEM ORGANISATION

A multi-agent system is a distributed system with localized decision-making and, usually, localized storage. An agent is basically an autonomous entity with its own value system and a means to communicate with other such objects. For a general survey on the application of multi-agent systems in manufacturing, see (Aissani *et al.* 2008). Our step aims at working out a system which is more powerful insofar as we respect effectiveness and efficiency performance: The decisions taken by the system are the result of an agents group work in a heterarchical organisation in first. Our approach consists in equipping the agents of our control system "**SPART/MCSRR**" (**S**ystem de **P**ilotage par **A**pprentissage par **R**enforcement en **T**emps reel/ **M**anufacturing **C**ontrol **S**ystem using **R**einforcement learning in **R**eal time) proposed in (Aissani *et al.* 2008) with a reactive multi objectives learning module as shown in figure 1.

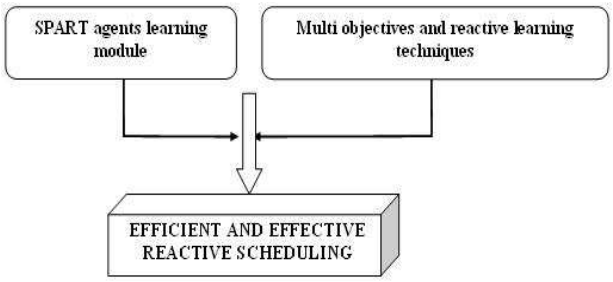

Figure 1. Integration of multi objectives and reactive learning techniques in SPART agent for an efficient and effective reactive scheduling

2.1 The SPART modelling

2.1.1 The SPART architecture

SPART architecture is based on resources agents (machines…), parts agents and an observer agent (see Figure 2)

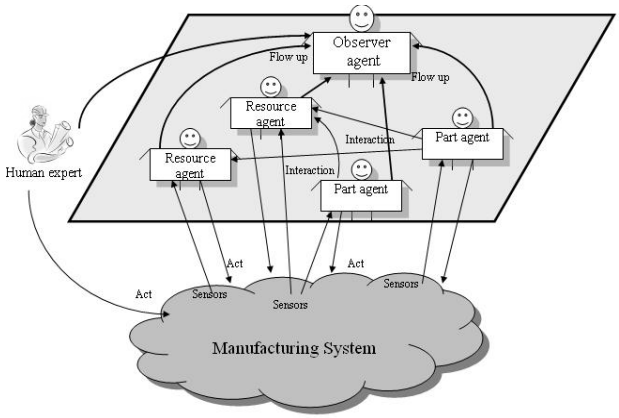

Figure 2. SPART architecture

The general idea underlying this approach is to consider both resources and physical parts as active entities. And, according to Alaadin modelization (Ferber and Gutknecht 1998), those agents have: ***Role***, They must have initially knowledge on their capacities: the name of the resource, names of their realizable operations and their properties, the list of these proper tools: ***Properties***, and the vicinity which represents the resources which are directly connected to this resource: ***Group*** (according to the cell fitting). And moreover, we equipped them with a ***Learning module***. Where a system state is coded **(St)** by the ***Perception*** module, agent chose an action **(A_t)** to execute by the ***Action*** module and according to what this action will generate, the agent perceives a numerical reward which can be positive or negative **(R_t)** To reward or punish the executed action. ***t*** is a given instant (see Figure 3).

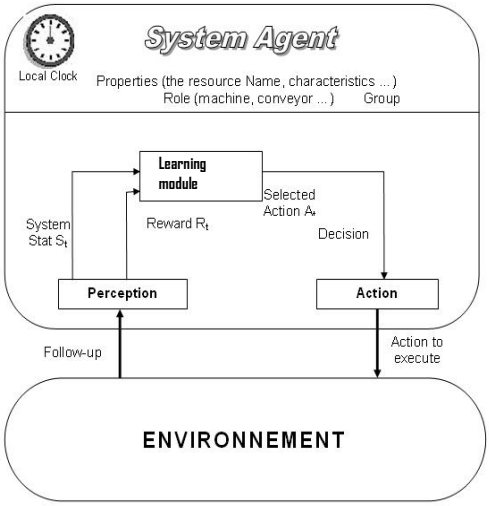

Figure 3. SPART/MCSRR generic architecture

Observer agent has a total sight on the system, the state variables which it observes is the indicators of efficient and effective performances. This agent observes all decision making centre.

2.1.2 Adopted Reinforcement learning technique

We propose to use a reactive and multi objective learning technique to propose efficient and effective scheduling solution in quasi-real-time and which improve over the time.

2.1.3 Markov Decision Process

A manufacturing system is regarded as a dubious, dynamic and unforeseeable environment, because it is subject to internal stresses (production risks…) and external constraints (forced market, unforeseeable orders…). The decision in such environment is a Markov decision process MDP according to Russell and Norvig, (1995). Markov decision process model is composed of two distinct elements: on the one hand, a Markov process modelling the environment which progress in time like a probabilistic automat and other share an element controller which examines the current system state to choose an action to execute in order to maximize the reward that it can obtain. A reasoning containing MDP can bring to the following speech: "I know that I am in *such situation* and if I make *such action* there are so many chances so that I find myself in this *new situation* by obtaining *such profit*."

In a Markov decision (see Figure. 4): in a perceived system state (or environment stat) $\mathbf{S_i}$ the agent executes the action $\mathbf{A_i}$ which according to the transition function **T** leads the system to the state $\mathbf{S_{i+1}}$ and the agent receives an $\mathbf{R_i}$ reward where **i** is a given moment.

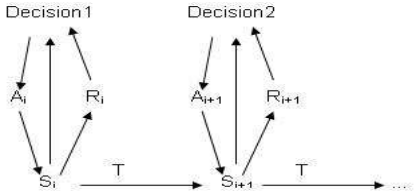

Figure 4. Markov Decisional Model

The policy of action π in a Markov decision process is a probability of choosing the action $\boldsymbol{a} \in A$ where A is the whole of the actions in the state $\boldsymbol{s} \in S$ where S is the whole of the states.

$$\pi : S \times A \to [0;1]$$

$$\pi : (s,a) = P(a|s)$$

A policy in a MDP is a kind of probabilistic plan. Bellman (1957) established a value function to estimate the utility of a state according to a reward which it can obtain by executing an action:

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s') \right]$$

T transition function, R Reward. V^{π} Utility value of the state according the π Policy. γ Rate of calculation, positive constant lower than 1 used to give more weight to the current reward than that of the future. This equation shows also the relation between the value of a state and that of its successors. It is this equation which will give rise to thereafter the Q function of GM-SARSA(0).

To learn optimal policy in Markov environment, where goal is known (to produce as soon as possible (effectiveness) and with lower cost by assuring no machine idle (efficiency)) and where learning parameters are not known: we do not have base of example beforehand conceived and when we cannot model the environment, i.e., when we do not have the transition function which models the system passage from states to states. So, we try to proceed by test - error and we speak then about reinforcement learning (Russell and Norvig, 1995).

2.1.3.1 Multiple goal SARSA learning for an efficient and effective scheduling

Sprague and Ballard (2003) proposed the most suitable learning algorithm for multi-agent system where the agent must discover an interleaved policy that adequately satisfies an ensemble of unordered sub goals when there are multiple reward signals. Sprague and Ballard (2003)'s approach replaces Q-learning with an on-policy learning algorithm Sarsa(0) (Singh and Sutton, 1996) which have the following update rule:

$$Q_i(s_i,a) = (1-\alpha) Q_i(s_i,a) + \alpha (r_i + \gamma Q_i(s_i', a')) \quad (1)$$

(Where Q_i is the *Q* value for module ***i*** and s_i is the observed stat for module ***i*** , a the selected action for s_i , a' the selected action for s_i' which is the new observed stat, γ Rate of calculation, positive constant lower than 1 used to give more weight to the current reward than that of the future and α a learning rate) Which replace the max over Q-values with the Q-value of the state action pair that is actually observed on the next step. Since updates are based on the actions that are actually taken, rather than on the best possible action, Sarsa(0) based modules discover Q-values that are closer to the true expected return under the composite policy. And for the composite policy **G**reatest **M**ass-**Sarsa(0)** is used, the suggestion is to generate an overall Q-value as a simple sum of the Q-values of the individual modules:

$$Q(s,a) = \sum_{i=1}^{n} Q_i(s_i,a)$$

The action with the maximum summed value is then chosen to execute.

This approach is based on modular architecture (see Figure5)

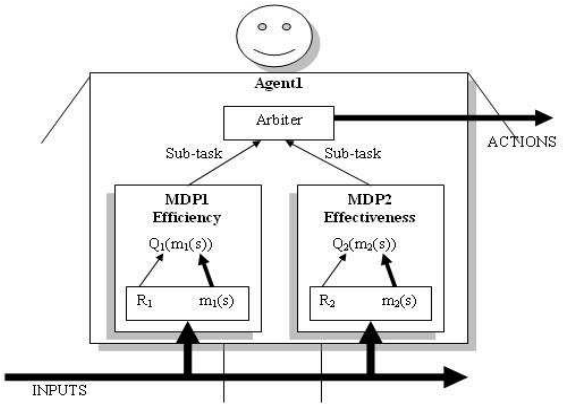

Figure 5. Modular MDP architecture

Thus the agent receives inputs from environment and executes actions to change the environment stat. However, the inputs are separated into modules which then learn in much smaller state-spaces. So, we define for each module ***i: mi*** a selection function which selects in the state representation of the part which relates to the module in question; ***Ri*** is a reward function related to this module goal; ***Qi*** is a Q-function called also *local function* that estimates expected reward from the module reward function. Arbiter contains the *global* Q-function $Q = \sum Q_i$

2.1.3.2 Modules definition

We have defined two modules for each agent, the first one deal with Effectiveness, guide the system towards a complete execution of all the jobs within the shortest times and minimize reiteration of job requests. The second one, deals with efficiency, guides the system towards the minimization of the machines downtime.

2.1.3.3 Selection function

The agent's inputs determine all the system stat S= {s1,s2,…,sn}, but each MDP module should receive the data which concern him, So, the selection function makes the agent to be able to identify what inputs are relevant to which module. In production environment, the inputs are: current time ***t*** ∈ ***0…T*** and the inventory of products ***p1… pn*** and their state ***Sp1… Spn*** (in progress, stand by…) on resources ***M1… Mm*** and their state ***Sm1…Smm*** (Working, stopped…) with the duration of production (***dp1…dp***) as well as the downtimes of the machines (***idl1…idlm***). The sub-tasks correspond to reaching one of the goals: minimization of the production duration (make span) and minimization of the machines downtime. Therefore, for the effectiveness module the relevant inputs are: current time, the inventory of products and their state on machine and the duration of production, and for the efficiency module the relevant inputs are: current time, the inventory of products on machine and the duration of the machines downtime with the machines state. So, we define S_i a subset of inputs corresponding ***i*** module

$$S_i = \{ s \,|\, s \in S, m_i(s) = s_i \}$$

S_i can be thought of as an abstraction of the global state-space and in machine learning, it is typically referred to as hidden state. However, sometimes hidden stat can be useful to an agent, eliminating irrelevant detail from the stat description; Agre (1988) described this case as passive abstraction.

2.1.3.4 Reward function

In a single MDP, the reward function will simply assign no reward to most states and some positive reward to a goal state. In a complex MDP composed of multiple sub-modules, the reward function might consist of some combination of several reward functions evaluating the different sub-tasks. The total reward of the agent is:

$R(s) = \sum_{i=1}^{n} R_i(m_i(s))$ where *n* is the number of modules

For the effectiveness module, a reward is computed from the parts stat.

$$R_{effess} = \begin{cases} 1 & if \quad S_{p_i} = \text{Launched} \\ 0 & if \quad S_{p_i} = \text{in progress} \\ -1 & if \quad S_{p_i} = \text{standby} \end{cases}$$

where R_{effess} is the reward function for effectiveness, S_{p_i} is the p_i stat.

While, for the efficiency module a reward function depends on the resources employment particularly the resources stats.

$$R_{effcy} = \begin{cases} 1 & if \quad S_{m_i} = \text{started} \\ 0 & if \quad S_{m_i} = \text{working} \\ -1 & if \quad S_{m_i} = \text{stopped} \end{cases}$$

where R_{effcy} is the reward function for effectiveness, S_{m_i} is the m_i stat.

2.1.3.5 Sarsa Learning Function

Each module has its own state-space and its own reward function, and is thus equipped to learn its sub-task using Sarsa(0) previously introduced. We can thus write the update formula for the local Sarsa Q-function as an extension of the formula (1):

$$\begin{aligned} &Q_{effess/effcy}(m_{effess/effcy}(s), a) = \\ &(1-\alpha) Q_{effess/effcy}(m_{effess/effcy}(s), a) + \\ &\alpha (R_{effess/effcy}(m_{effess/effcy}(s)) + \\ &\gamma Q_{effess/effcy}(m_{effess/effcy}(s'), a')) \end{aligned} \qquad (2)$$

Therefore, after the various MDP functions presentation, we can give a general diagram of the learning progression in Figure 6

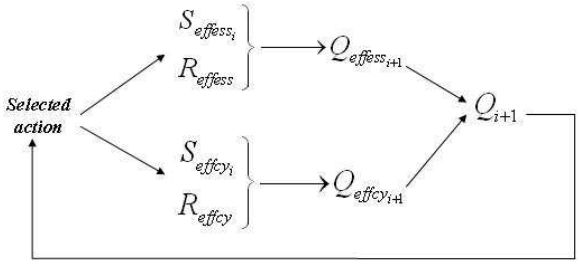

Figure 6. The modular learning progress

The modular Sarsa algorithm proceeds along the same lines as the regular Sarsa algorithm (see Figure 7), differing only in how the policy is updated and the action to execute is selected. In the first phase of the algorithm called ***exploration,*** decisions that are for us the placement of jobs on resources is done randomly according to the Boltzmann probability (corresponding to the choice made in (Aissani *et al*. 2008)) in order to try all opportunities, while in the second phase called ***exploitation*** placement is based on the Q value which have more meaning in this phase.

1- In the current stat s, select the action to execute which the Q value is raised and according to a probability by respecting a Boltzmann probability distribution

2- Execute selected action a, leading to the new stat s'

3- Update each module according to the update formula (2)

4- Repeat, with s' set to the current state (go to (1))

Figure 7. Modular Sarsa algorithm

However, it is important to note that it is more difficult to model the system as a MDP and to determine the parameters of learning: the representation of the system state can be very complex and the Development of reward function can be difficult when the indices of performance are more complex.

3 SIMULATION AND EXPERIMENTS

This simulation was implemented on Borland JBuilder because it offers very great possibilities of communication as it facilitates the threads programming and especially because we used MADKIT platform of J Ferber for the SMAs development (Madkit).

The reinforcement-learning algorithms advantage is that they allow an evaluation during learning. To carry out this evaluation, we conceive the following criteria.

3.1 Effectiveness Criteria

1- The Makespan of production ***Cmax*** which is the Max of the production time. The most relevant scheduling criteria

2- The percentage ***pSucc*** of tasks carried out successfully (without deadlines). $\text{pSucc} = \frac{\text{NTsucc}}{\text{NT}} \times 100$

Where: ***NTSucc*** is the number of tasks carried out successfully and ***NT*** is the total number of tasks. Inspired by Charton *et al*. (2003)

This two criteria deal with scheduling solution quality, minimizing the production time and the part waiting time.

3.2 Efficiency Criteria

The efficiency performance deals with the manner in which the resources are used to achieve an aim. So, we use the average of machines downtime ***aIdle***

$$aIdle = \frac{\sum_{i=1}^{Nm} Idle_i}{Nm}$$

Where ***i*** is the machine and ***Nm*** is the number of machines $\boldsymbol{Idle_i}$

By defining these criteria, we can see that the 2 criteria can be in relation: *Cmax* and *aIdle* because normally there is a correlation between minimizing makespan and increasing machine utilization. So, effectiveness and efficiency can have some goals that converge.

3.3 Experiences and investigation

As an experimental test base we used Hurink et al. (1994) set of data which was developed by Dauzère-Pérès and Paulli, (1997), this Benchmark is composed of: *edata*, *rdata*, *sdata* and *vdata* sets of data and where files are *File M x P* (file of M resources and P parts problem). L01 is 5×10, L10 is 5×15 L20 is 10×10 and L30 is 10×20, *edata* are the original problems (from Hurink et al. (1994) and Adams et al. (1988)). Each set M_i is equal to the machine to which operation i is assigned in the original problem, plus any of the other machines with a given probability. Depending on this probability Hurink et al. generated three sets of test problems: *edata*, *rdata*, and *vdata*. The first set *edata* contains the problems with the least amount of flexibility (close to the original job-shop scheduling problems), whereas the average size of M_i is equal to 2 in *rdata* and m/2 in *vdata*. For more details, see Hurink et al. (1994).

3.4 Improvement of performances

Firstly, we used m06 from *edata* to see how our approach can improve scheduling solution quality in term of effectiveness and efficiency with wire of time by learning.

m06 is 6x6 (6 machines and 6 jobs with one operation on each machine). The first graph (Figure 8) shows the *Cmax* and *pSucc* evolution. What is interesting to notice, is that at 5000 and 6000 iterations the scheduling solutions are given with a large *Cmax*, up to 83, in reinforcement learning. This phase is called an exploration phase where actions are carried out by chance according to Boltzmann probability; while the solutions suggested towards and after the 10000 iterations, *Cmax* is very interesting 47 we compared this Cmax with the maximum sum of the job running times Max_{jt} $(\max_{j=0}^{Part\_nbr}\{\sum_{i=0}^{Task\_nbr} Run\_Time(i)/p_j\})$ which is 47 here. Moreover, we notice a convergence because the interval of the suggested solutions *Cmaxs* is [59,47]. This phase is called an exploitation phase, where the choice of actions is based on Q values. The light width of the interval [59,47] is explained by the fact that our approach deals with several objectives, therefore, in certain decision making for example: the minimization of resources down time got a better rewards than the minimization of *Cmax* so, selected action is that dictated by the first goal. *pSucc* is closely related to *Cmax* by nature, bus *Cmax* is small as well as the Jobs do not make latency as well as the *pSucc* is large and screw-poured. It is why, *pSucc* in exploration phase reached 20%, but in exploitation phase *pSucc* reached 55% within [55%,35%] interval.

Figure 8. The pSucc and Cmax for the m06 set of data

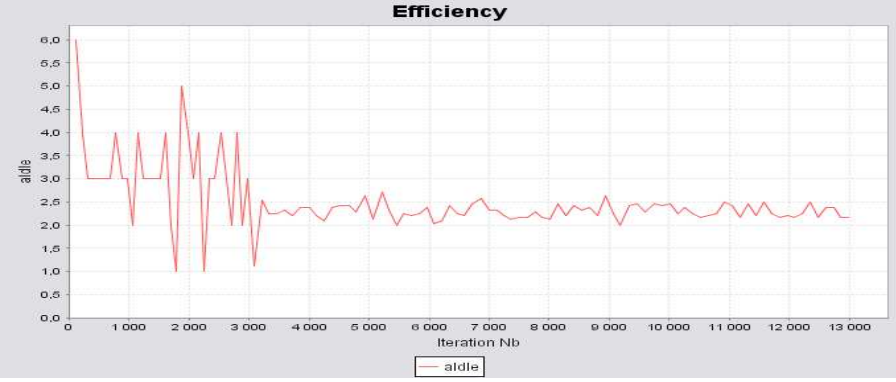

Figure 9. Graph of aIdle for the m06 set of data

Cmax, *pSucc* and *aIdle* converge to their optimum after many learning iteration, so, scheduling solutions given in exploitation phase are more optimized in term of effectiveness and efficiency (translated by *Cmax*, *pSucc* and *aIdle*) than the given one in exploration phase, for example in time of 1910 iteration *Cmax* of the given solution is 67 and the *pSucc* is 30% and efficiency criteria *aIdle* is 5, However, in 11950 iteration they have improved as *Cmax* decrease to 49, *pSucc* improve to 55% and *aIdle* is 2,15 units of time.

We notice that we obtained these results in 28mn34s on a CPU Mobility 1,73Ghz and a memory of 512Mb. This is a very good performance; it is the advantage of reinforcement learning, it offers solutions at any time and converges very quickly towards best solutions.

The improvement of effectiveness and efficiency performances can be illustrated by showing the difference between a scheduling suggested after 245 iterations i.e. with the beginning of learning and the other one suggested after 12450 iterations in a 3x3 problem (small sized instance for graphical readability) (Figures 10 and 11)

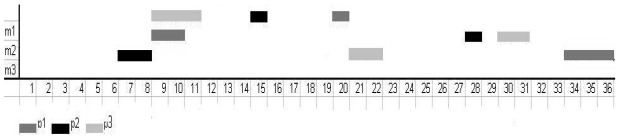

Figure 10. Scheduling solution in the beginning of learning

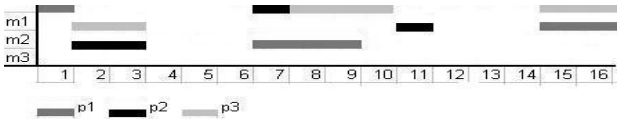

Figure 11. Scheduling solution after many learning iteration

The Makespan in the first solution (36) is much bigger than the given one (16) in the second one. Moreover, the machine down time (27 on average) is more important in the first solution than the second (10 on average). This proves that there is an improvement in the given scheduling solutions by learning in terms of effectiveness and efficiency emphasizing the relationship that may exist between the two.

We carried out an experiment by exploiting *exploration* and *exploitation* stages; we caused the same disturbance (broken down of all machines) twice at 1000^{th} iteration and 10000^{th}.

We noticed in figure 12, that in the exploitation stage, the disturbance is quickly deadened and the system is brought back to its most significant performances: $C\max \approx BestC\max$.

Lastly, these results show that our system is able to learn an optimal policy of control which is in continuous improvement while trying to reduce the attempts to carry out right tasks within their times.

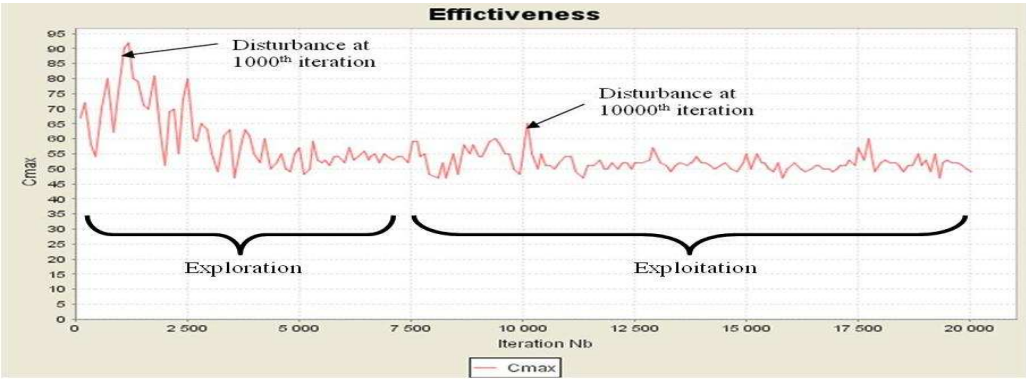

Figure 12. Obtained results with caused disturbances (Cmax)

3.5 Results comparison

We observe our system on 25000 iterations; iteration is a new launch to the scheduling problem. The following table shows all experiments results. *Best Cmax* : Calculated according to our approach; *Best Cmax of H and D* : Calculated by Hurink et al. (1994) et Dauzère-Pérès and Paulli, (1997) « Tabu reaserch »; Max_{jt} : Total execution time without any waiting time; *Itération Nbr* : The number of iterations in which we gained our Best Cmax; *Best pSucc* : Calculated according to our approach; *Itération Nbr* : The number of iterations in which we gained our Best pSucc; *Best adle*: Calculated according to our approach; *Itération Nbr* : The number of iterations in which we gained our Best aIdle; *Runing time for 25000 iteration* : The time to execute iteration 25000.

The above table 1 shows these experiments results by using the Benchmark previously represented, Firstly we can see that the results obtained with *edata* are less good then the obtained results with *rdata* which are less good then the obtained with *sdata* and *vdata*, it can be explained by the flexibility (increase of resources number which are able to execute a job) increase in *rdata*, *sdata* and *vdata*. It should be noted that BestCmax in our approach is obtained by comparing all Cmaxs on 25000 iterations, but nothing tells us that this is the best solution we can get, except that we are assuming that at this stage learner agent reaches the maturity to find the most optimal scheduling. When *Cmax* is compared to the Max_{jt} the average of this difference is 293 with *edata* and 124 with *rdata*, *sdata* and 135 with *vdata*. The same thing for *aIdle*, the average is 10 with *edata*, 4 with *rdata*, 5 with *sdata* and 2 with *vdata*. *pSucc* increases most of time with the flexibility increase, the *psucc* average with *edata* is 74% and is 51% with *rdata*, 98% with *sdata* and 31% with *vdata*. Our approach is based on a multi-agent system; the number of agents' increases with the configuration complexity generating density in communication which can explain this disturbance in performances, but most of time the increase of resources number which are able to execute a job provides flexibility in the placement of jobs. Nevertheless, our agents give very good results in term of Makespan compared to the best solutions given by Hurink et al. (1994) and Dauzère-Pérès and Paulli, (1997), the boldfaced *Cmax* shows the best solution within our approach solutions and the above ones. Our approach gives 6 best solutions in term of *Cmax* on 24 experiments, which is very promising.

		Effectiveness						Efficiency		
		Best Cmax	Best Cmax of H and D	Max somme of job production time Max_{jt}	Iteration nb1	Best pSucc	Iteration nb1	Best aIdle	Iteration nb1	Running time for 25000 iteration
edata	L01	**605**	609	413	10335	84%	11817	0	11817	21mn11s
	L10	866	866	443	20597	86%	20761	2	20761	28mn39s
	L20	1343	**857**	756	25243	51%	22018	16	14221	47mn21s
	L30	1264	**1147**	726	10414	51%	20306	35	20306	42mn41s
	M06	**47**	55	47	18655	85%	18557	2	16646	28mn34s
	M10	**673**	871	655	22734	84%	22734	7	17389	38mn06s
rdata	L01	**421**	570	413	11760	52%	11760	3	11760	26mn17s
	L10	804	804	443	22270	51%	22780	2	22780	31mn38s
	L20	774	**756**	756	16893	50%	16893	12	17950	18mn26s
	L30	1068	1068	726	12538	50%	12538	5	13184	20mn57s
	M06	47	**47**	47	20552	50%	20663	1	21883	30mn17s
	M10	**675**	679	655	18957	51%	18957	3	23476	18mn50s
sdata	L01	417		413	12066	100%	15662	1	23885	29mn03s
	L10	804		443	11089	100%	11089	5	23345	30mn34s
	L20	773		756	23792	98%	10142	8	22957	43mn30s
	L30	1068		726	20633	91%	21053	17	18529	29mn43s
	M06	47		47	19817	100%	19817	1	19817	36mn12s
	M10	679		655	19880	100%	21560	2	19880	18mn20s
vdata	L01	**461**	570	413	16387	30%	17412	1	16387	22mn15s
	L10	804	804	443	19801	45%	19801	5	19801	20mn53s
	L20	810	**756**	756	18745	29%	19980	3	20145	30mn17s
	L30	1068	1068	726	21056	32%	21056	5	21056	32mn50s
	M06	49	**47**	47	23430	34%	24490	1	24490	27mn26s
	M10	660	**655**	655	20102	40%	20102	4	20102	18mn10s

Table1. The experiments results

3.6 Multi criteria satisfaction

The dotted lines in Table 1 represent solutions that meet two criteria at the same time, for example, experience in *edata/L01* in 11817 iteration gives a solution with the best *pSucc* 84% and the best *aIdle* 0, therefore, it is a solution which is effective and efficient. These kinds of solution, which belong to the pareto frontier represent 50% of the generated solutions (experiments). The dark lines represent solutions that meet all the criteria and these solutions are 21% of solutions. And only 29% of the solutions are not effective and efficient at the same time with the best results (as we are looking for best *Cmax*, best *pSucc* and best *aIdle*). This is the strong point of our approach; it creates scheduling solutions that can be effective and efficient at the same time thanks to the multi-objective learning algorithm in light of the experiences gained on the benchmark presented above.

4 CONCLUSION AND FUTURE WORK

This paper has shown how to apply reinforcement learning and especially Sarsa-multi-objective to jobshop scheduling problems by formulating them as two MDP modules, one concerning the effectiveness and the second efficiency. The paper has also presented how to design manufacturing system as a multi-agent learning system (according to Aissani et al. 2008) using Alaadin concepts and MadKit plateform. Finally, we have presented simulation results on a benchmark for a variety of flexible workshops. Those experiences prove that there is an improvement in the given solution performances in term of effectiveness and efficiency. We note also the correlation that can connect some of effectiveness and efficiency criteria. By comparing the *Cmax* of our scheduling solutions with those obtained by linear algorithms, the results were very promising. And last, we saw that the solutions obtained by our approach converge to scheduling that meets the efficiency and effectiveness criteria at the same time in 50% of cases.

In the future we plan to extend the learning algorithm by using modular and monolithic system (Karlsson, 1997) to more optimize the solution. We expect also to take account of the market variation as constraints to define the reward function. In other words, arrivals date of orders, volume and promised delivery times. And we will compare our results with those of approaches based on artificial intelligence as genetic programming. And as our system is a multi-agent system at the base, we think involve more other decision-making entities as decision support systems (DSS) that will be an additional learning centre. Then we will try to apply our approach to real-world examples and other types of workshops, in other words: benchmarks.

REFERENCES

Adams, J., Balas, E., and Zawack, D,(1988), "The shifting bottleneck procedure for job-shop scheduling", Management Science, Vol 34, p 391-401.

Agre. P. E, 1988, The dynamic structure of everyday life. PHD thesis, Massachusetts Inst of Tech cambridge (MIT) artificial Intelligence Lab

Aissani. N, D. Trenteseaux and B. Beldjilali. 2008. Use of Machine Learning for Continuous improvement of the Real Time Manufacturing control system performances. *IJISE: International Journal of Industrial System Engineering*, Vol 3, No 4, 2008 (To be appeared)

Aydin. M. E, Özteme. E, (2000), Dynamic job-shop scheduling using reinforcement learning agents, Robotics and Autonomous Systems, Vol 33, No 2, p 169-178

Bellman. R, (1957), 'Dynamic Programming'. Ed Princeton University Press, p 83.

Bhattacharya, B, Lobbrecht Ah and Solomatine Dp (2002) 'Control of water levels of regional water systems using reinforcement learning' Proc. 5th International Conference on Hydroinformatics, Cardiff, UK, July 1-5, 2002, p 952-957

Bidot J, T. Vidal, P. Laborie and J. C. Beck, 2007, A General Framework for Scheduling in a Stochastic Environment. Proc International Joint Conference on Artificial Intelligence IJICAI07, P. 56-61

Bousbia S and Trentesaux D (2002) 'Self-Organization in Distributed Manufacturing Control: state-of-the-art and future trends', IEEE International conference on Systems, Man & Cybernetics, Vol 5, El Kamel, Mellouli & Borne (eds), CD-Rom, ISBN: 2-9512309-4-X, (Hammamet, Tunisie, Octobre 2002), paper #WA1L1, 6 p.

Butalaand. P and Sluga. A, (2002), Dynamic structuring of distributed manufacturing systems, Advanced Engineering Informatics, Vol 16, No 2, P 127-133

Charton. R, Boyer. A and Charpillet. F (2003). 'Learning of Mediation Strategies for Heterogeneous Agents Cooperation', in : 15th IEEE International Conference on Tools with Artificial Intelligence - ICTAI'2003, Sacramento, Californie, USA, IEEE Computer Society, B. Werner editor, Nov, 2003, p. 330-337.

Csaji B. C and L. Monostori. 2006. Adaptive algorithms in distributed resource allocation. Proc of the 6th International Workshop on Emergent Synthesis (IWES-06), August 18–19, The University of Tokyo, Japan, p. 69-75

Dauzère-Pérès, J and Paulli, (1997), An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research Vol 70, p. 281-306.

Denkena. B, L.-E. Lorenzen and A. Battino (2006), Increased Production Flexibility and Efficiency through Integration of Process Planning and Production Control, CIRP Journal of Manufacturing Systems, Vol 35, No 5.

Ferber. J, Gutknecht.O (1998) 'Un méta-modèle organisationnel pour l'analyse, la conception et l'execution de systèmes multi-agents' *Proc of 3rd International Conference on Multi-Agent Systems ICMAS'98,* p 128-135

Hurink, J., B. Jurisch, and M. Thole, (1994)'Tabu search for the job-shop scheduling problem with multi-purpose machines', *OR Spektrum*, No 15, p 205-215.

Karlsson. J, (1997), Learning to solve Multiple Goals, Doctorat thesis, University of Rochester

Katalinic. B and Kordic. V (2004) 'Bionic assembly system: concept, structure and function' Proc of the 5th International Conference on Integrated design and Manufacturing in Mechanical Engineering, IDMME 2004, Bath, UK, April 5-7, 2004

MadKit, Multi agents Development Kit, http://www.madkit.org/downloads

Maione. G and Naso. D, (2003), 'Discrete-event modeling of heterarchical manufacturing control systems', Systems, Man and Cybernetics, 2004 IEEE International Conference, Vol 2, 10-13 Oct. 2004, p 1783 - 1788

Marthi. B, Russell. S, Latham and D, Guestrin. C, (2005),' Concurrent Hierarchical Reinforcement Learning', proc of International Joint Conference on Artificial Intelligence, IJCAI-05, p 779-785

Oliveira L. and J. Diolino. An Approach to Design Control Systems for Distributed Production Systems as a Collaborative Architecture. Printed Notes of ICAPS'03 Doctoral Consortium. Trento, Italy

Pujo. P and Brun-Picard. D, 2002, Pilotage sans plan prévisionnel ni ordonnancement préalable , Méthodes du pilotage des systèmes de production, Hèrmes, 2002. p 129- 162.

Prabhu. V.V, (2003). "Stability and Fault Adaptation in Distributed Control of Heterarchical Manufacturing Job Shops," IEEE Transactions on Robotics and Automation, Vol. 19, No. 1, p. 142-147.

Russell S. Norvig P. (1995) 'Artificial Intelligence: A Modern Approach', The Intelligent Agent Book. Prentice Hall Series in Artificial Intelligence, 1995.

Senechal O, (2004), Pilotage des systèmes de production vers la performance globale, Habilitation à diriger des recherches, l'université de valenciennes et du Hainaut cambresis,

Singh. S and R. Sutton, (1996), Reinforcement learning with replacing eligibility traces. MachineLearning, Vol 22, p1-3

Sprague. N, D. Ballard, (2003), MultipleGoal Reinforcement Learning with Modular Sarsa(0), International Joint Conference on Artificial Intelligence IJCAI 2003 , p 1445

Trentesaux D., M. Gzara, S. Hammadi, C. Tahon and P.Bo. (2001). D-Sign: un cadre méthodologique pour l'ordonnancement décentralisé et réactif. Journal Européen des Systèmes Automatisés. p. 933-962

Watkins, C. J. C. H, (1989) 'Learning from delayed rewards', PhD thesis, Cambridge University, Cambridge, England.

Wei Y-Z and Zhao M-Y, (2005), A reinforcement learning-based approach to dynamic Job-shop scheduling, Acta automarica sinica, Vol 31, No 5, p 765-771.