Schedae Informaticae Vol. 25 (2016): 165–176
doi: 10.4467/20838476SI.16.013.6194

tfml 2017
theoretical foundations
of machine learning, Kraków

Online Supervised Learning Approach for Machine Scheduling

Bartosz Sądel, Bartłomiej Śnieżyński
AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunication
Department of Computer Science
al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: *sadel@agh.edu.pl, bartlomiej.sniezynski@agh.edu.pl*

Abstract. Due to rapid growth of computational power and demand for faster and more optimal solution in today's manufacturing, machine learning has lately caught a lot of attention. Thanks to it's ability to adapt to changing conditions in dynamic environments it is perfect choice for processes where rules cannot be explicitly given. In this paper proposes on-line supervised learning approach for optimal scheduling in manufacturing. Although supervised learning is generally not recommended for dynamic problems we try to defeat this conviction and prove it's viable option for this class of problems. Implemented in multi-agent system algorithm is tested against multi-stage, multi-product flow-shop problem. More specifically we start from defining considered problem. Next we move to presentation of proposed solution. Later on we show results from conducted experiments and compare our approach to centralized reinforcement learning to measure algorithm performance.

Keywords: supervised learning, reinforcement learning, scheduling, multi-agent system

1. Introduction

Machine scheduling problem is almost as old as the first Ford's assembly lines, dating back to twenties of 19th century. Rapid development of manufactures sparked

Received: 11 December 2016 / Accepted: 30 December 2016

efforts to make this process more efficient. Early on heuristic methods have caught the attention. Those included Just in Time(JIT), Kanban and few others [1]. Those methodologies emerged from car manufactures in Japan and soon become popular in other industries. IT business had initially adapted those concepts to team management process, to become scheduling problem solutions later on. Simultaneously pull and push systems where introduced in [2].

Machine scheduling may be considered as an optimization problem. Unfortunately, due to being NP-hard, finding optimal solution is not an easy task. Therefore, many algorithms were developed to solve it with increasing effectivity. Along finding new methods, problem also evolved to become its on-line version in which the goal is to optimize scheduling as continuous process with orders/tasks being generated when system is already working. In the past few years, agent-based solutions with use of reinforcement learning have been proposed for such cases [3, 4].

In this paper we consider agent-base approach with another machine learning strategy, that is supervised learning. Results in other domains show that in complex environment this type of learning gives improvements faster than reinforcement learning [5, 6].

In this paper, we will begin with defining on-line machine scheduling problem. Next, we will propose our solution and provide necessary theory. This will be followed by conducted experiments and conclusions. Our aim is to check whether supervised learning can replace reinforcement in dynamic environments.

2. Problem Representation

In this paper dynamic version of *flow-shop* problem is considered. It should reflect real manufacturing process where we have company producing products of n different types. Each product requires processing on m different stations. Single station may contain any number of machines. It's enough for a product to be processed just by one machine on every station. To make similarity with traditional *job-shop*, we can name process of producing product a *job* and processing a product by a certain machine a *task*. Later on we will use these terms interchangeably. Each product needs to pass all the stations. Additionally path alongside stations have to be the same for all product types. Thanks to these restrictions we can number stations by the sequence in which jobs will be passing through those. Since in real life manufacturing situation usually there is a delay between finishing processing of a product on one machine and starting on the next one, we have to introduce buffers between stations, where we can store the waiting products. As orders usually consist of more than just one unit of product, we need additional buffer for finished jobs where products are stored until whole order is completed. Sample configuration is presented in Figure 1. To simplify demonstration without loss of generosity we assumed different types of products to be marked as circles, pentagons and octagons. Every product in presented example must proceed sequentially by *Station1*, *Station2* and *Station3*. Finished products are stored in last buffer, called *Completed*.

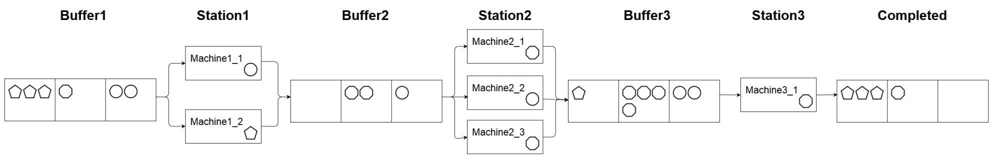


Figure 1. Example of model representing considered problem.

Every order o_i in our model can be defined as:

$$o_i = (p_i, c_i, d_i, r_i, f_i, ty_i, q_i) \tag{1}$$

where p_i stands for priority, c_i for creation time, d_i for due time, r_i for reward received after order completion, f_i for penalty received if order is not completed on time, ty_i for the type of the product order demands and q_i for quantity of this product. λ_{ty} is Poisson parameter describing the frequency with which orders of type ty are arriving. Number of product units q_i is drawn from uniform distribution $U(a, b)$ where a and b are minimum and maximum values. After arriving, orders are stored in the queue sorted by their priority, so the most important ones are delivered first. When new order comes to the system, products of which it is composed are inserted into $Buffer_1$. Then machines from $Station_1$ takes products from it and work on them. After processing is finished products are stored in $Buffer_2$. Next the same happens for the next stations until products finally gets to $Completed$ buffer. When there will be enough products in buffer, we are delivering first order from our queue. Depending if we managed to deliver order before due time it is possible to get reward or penalty. Machine is defined as:

$$M_{i,j} = (H_{i,j}, V_{i,j}, t_{i,j}, ty_{i,j}) \tag{2}$$

where i is number of station, j is number of machine in station, $H_{i,j}$ is the health of machine, $V_{i,j}$ is the table containing velocity of processing of different product types on this machine, $t_{i,j}$ is the time is takes to reconfigure machine and $ty_{i,j}$ is type of product machine is configured to process. Machines are allowed to change the type of product they are processing only when they are idle. Each time, machine have ty different actions to choose, where each one corresponds to processing different product type. When action of changing processed type is chosen, machine needs to remain idle for a fixed period of time which simulates machine reconfiguration. In our model every machine can process just one product at a time. Thanks to this property of our model, sometimes it is more beneficial for a machine to wait for the product of the currently configured type, instead of changing. Additionally each machine can process each product type with different speed. This property makes certain machines better at processing one type of a product, where another one may work better with other type. Although we allow for situations where single machine or all machines in station needs no time for processing certain type of products, we still demand this product to pass through that station. In our environment every machine has chance to break down. When it happens currently processed product is pushed back to the previous buffer. Machine remains than idle for a $t_{i,j}$ of turns, simulating fixing process, after which it starts working again.

Important thing in every model is optimization criteria which is used to describe system effectiveness. In problem we are considering there are many different parameters we may want to optimize. The most common would be profit maximization or minimization of product amounts in buffers. First option is reflecting most business scenarios. This way we don't only look for most valuable orders to complete, but we also have to the handle the ones with the highest penalty. The second option chooses what to do basing on number of products waiting in all buffers excluding the *Completed* one. This way we are minimizing idle time of machines.

Table 1. Notation of parameters.

T	Number of product types
ty	Product type
λ_{ty}	Poisson parameter describing frequency of orders with product type ty
$M_{i,j}$	Machine j in station i
$H^t_{i,j}$	Health of machine j in station i in turn t
$V_{i,j}$	Table with velocities of processing different product types on machine j in station i
$t_{i,j}$	Time it takes to reconfigure machine j in station i in turn t
o_i	I-th order
p_i	Order priority
r_i	Order reward
f_i	Order penalty
c_i	Order creation time
d_i	Order due time
q_i	Quantity of product units in order
$Buffer^t_{i,ty}$	Quantity of products ty in buffer of ı-th station in turn t
$Station_i$	I-th station
P_{limit}	Threshold used in supervised learning
u_{ty}	Unit price of product type ty
s_{ty}	Switch cost of product type ty

In this paper we chose to use the last option as it seems interesting for manufactures which don't want machines to be idle. Example of such machine may be foundry furnace which is shut down only once every twenty years and we would like to use it as much as we can since it can't be turned off. Used notation by us is gathered and presented in Table 1.

3. Proposed Solution

Our solution takes usage of approach proposed by Śnieżyński in [6]. Namely, we will try to employ supervised learning algorithms to solve problem previously defined in section 2. It's done by introducing learning agents with special architecture allowing them for taking usage of machine learning algorithms, designed for static *problems*.

First we need to model our problem as a multi-agent system. To do so we will treat every machine like a separate intelligent agent. Each of them will have it's own knowledge base and own classifier. Decisions about changing processed product type by machine $M_{i,j}$ will depend only on the decision of the agent connected with this machine itself.

Algorithm implemented by our system is presented on Figure 3. In this diagram we can see flow of simulation in implemented solution. The first step in every turn is order generation. Whether order should be generated this turn and if so, how big should it be depends on λ_{ty} and parameters of experiment establishing lower and upper boundary of order sizes. Next, products form generated orders are added to the buffer of first station. After that for each machine in station we check if it has finished processing of any product in previous turn. If so, we collect it so later we can add it to the buffer in next station. Next step is to fire learning process on machine's classifier. In our experiments we do this every 5 turns. Later, every machine has a chance to change type of processed product. The change is impossible, if machine is already processing some product. In case of changing product type we save that information in a form of entry in our knowledge base. It's value is calculated using equation 3. In other case we continue to process currently processed product. In this places we are also checking if machine should break in this turn. Next, we go on to the next machine in station. After processing of all machines in a single station we move on to the next station but this time products added to the buffer are ones collected in previous station. When all stations are finished we start the next turn.

Architecture of single agent is presented in Figure 2. This model introduces two main modules. First of them, named Processing module, is responsible for perceiving of the environment. It receives informations from environment and other agents. Later on, performs transformation of those observations into format appropriate for the classifier. Afterwards, learning module is asked if it knows what to do in a current situation basing on previously learned knowledge. Learning module communicates with classifier which role is to classify received data into one of the T classes where T is number of different product types considered in problem instance. If this classifier can choose proper action with probability higher than P_{limit} it simply responds with this action. If it's unsure what to do it can trigger learning process using gathered training data. After that learning module uses classifier again and returns the best choice without checking against p_{limit}. Finally, processing module puts chosen action into practice.

Although from outside this may look as reinforcement learning it's quite different. Where in reinforcement learning agents have the knowledge about previous states hidden in trained classifier, here we can collect history of environment states and

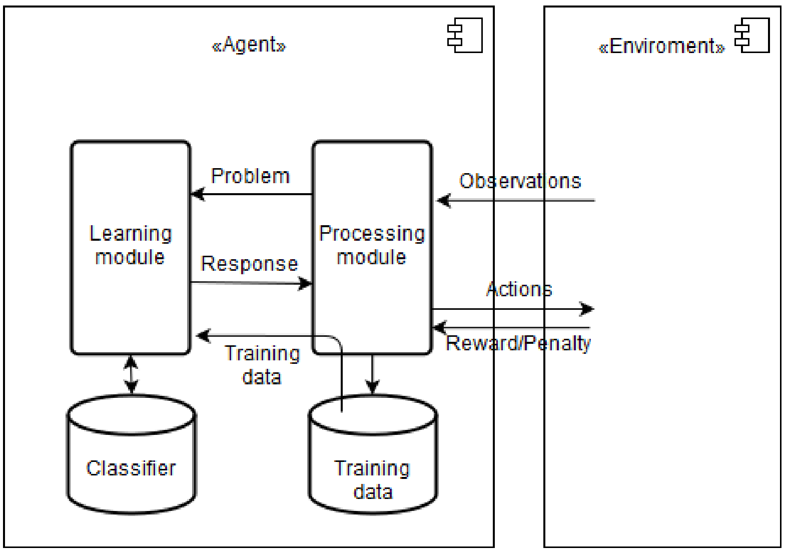


Figure 2. Used agent architecture.

taken actions (decisions made by agents) in form of training set. This way we can explicitly see experiment history and learn from it.

Each entry in the training set consists of two parts. The first of them is data and the second is a label, also called a category, which is description of the data part. There are two common ways to construct entries in problems like one considered here. In first of them data part of each entry consists of buffers from each station and health property of every machine. Product type which should be produced in this state is used as category in this approach. This method is often used as it's simple to implement and directly tells us, what should we do in given state. The second method combines data part and category from previous method to create the data part. As a label in this option we are using quality of the decision. Although harder to implement, this method offers us benefits, in a form of a way to express how good each decision is. Thanks to this, we can connect the same state with two different actions, where in the first method this would be impossible. In our work we are using second approach as we hope that ability to express value of moves, will improve proposed solution.

$$f(ty,i,j,t) = \begin{cases} u_{ty} * sgn(Buffer^t_{i,ty}), & \text{if } ty = ty_{i,j} \\ u_{ty} * sgn(Buffer^t_{i,ty}) - s_{ty}, & \text{otherwise} \end{cases} \tag{3}$$

Each entry is composed of amounts of products in buffers from 1 to m, health of all machines and a chosen action, which takes one of ty values. Value of our entry is calculated using equation 3. Entries are added to training set after every machine decision.

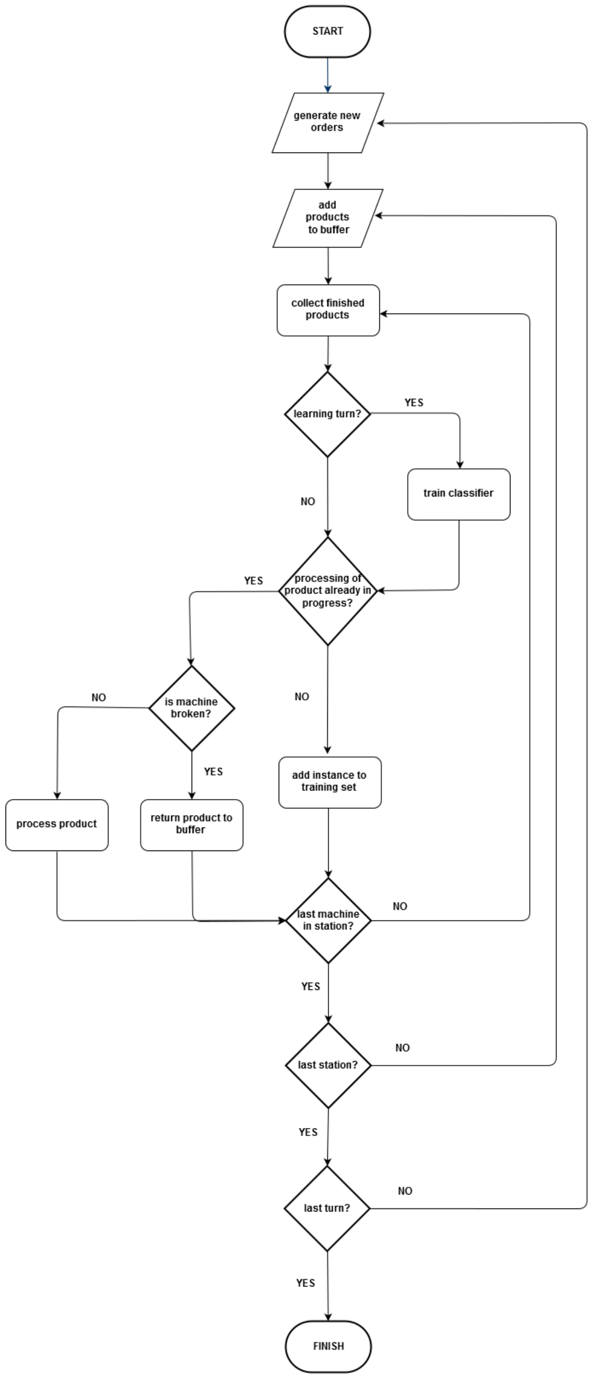


Figure 3. Flow diagram of implemented simulation.

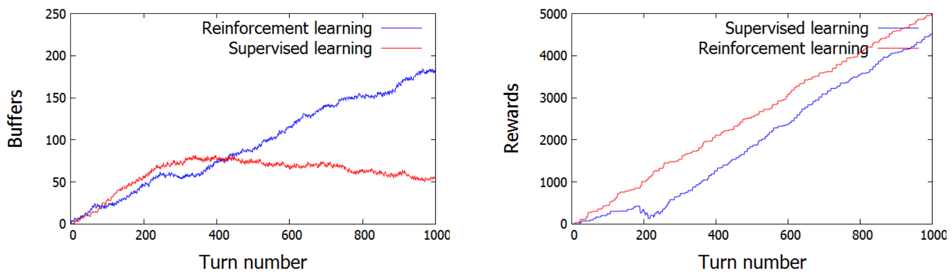
Learning process can be started every l turns or when quality of the best action is lower than a certain threshold. Depending on tools used it may mean constructing new classifier based on whole training set or just improve it using previously unseen examples. We have to take in to account that training classifier may be computationally expensive depending on the number of stations and machines in each of them, as they make the amount of collected entries significantly larger. This property makes supervised learning worse than reinforcement learning in problems where products are processed fast and there is no much time for decision-making. However in cases when processing takes hours or even days, learning may be run parallel to machine work thus be imperceptible.

4. Results

In order to test our approach various experiments were conducted. In all of them our method was tested against multi-agent reinforcement learning proposed in [4]. Parameters used in simulations are presented in Table 2. Every test was conducted 10 times and the presented results are the mean of received ones. Although system is able to carry out much bigger simulations we chose to present smaller ones. This choice is motivated by similar configurations used in [4], which allows us for a better comparison between both solutions.

4.1. Comparison in a 2x2 System

First test were run in a simple configuration containing only 2 stations with 2 machines each. In this setup algorithms should learn quickly.



(a) (b)

Figure 4. Results for 2x2 configuration.

In Figure 4a we can see results of both algorithms running in the same problem composition. We can notice that although supervised learning algorithm had slower

Table 2. Experiment parameters.

Parameter	**Value**	
T	2	
ty	0	1
s_{ty}	40	45
u_{ty}	25	30
λ_{ty}	3	3
r_i	Sampled from uniform distribution $U(0,60)$	
f_i	$d_t - t$	
Machines health	Sampled from uniform distribution $U(0,1)$ with 0,05 probability to break	
d_i	Sampled from uniform distribution $U(15,25)$	
Min order size	1	3
Max order size	1	3
Time span between learning	5 turns	
Supervised learning algorithm	J48 (C4.5)	
Reinforcement learning algorithm	Watkins	

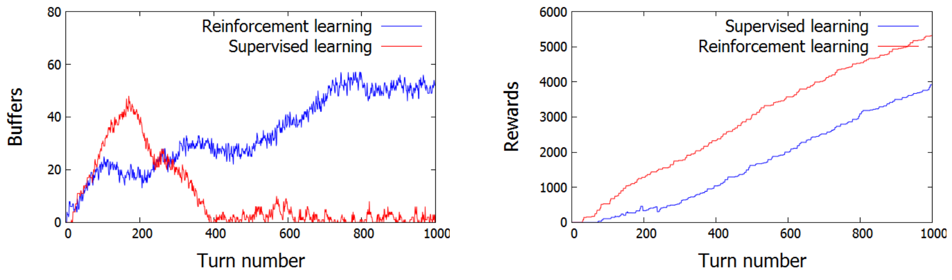
start it took the lead before end of experiment. Although rough start it managed to stabilize and even decrease number of products waiting in buffers, where reinforcement learning struggled with that much harder. Slower start of supervised system version was probably caused by lack of the knowledge at the start of the experiment. Reinforcement learning performed better in that situation because it does not need as much previous experience to work with as supervised learning. Situation has changed when the supervised algorithm gathered enough entries in the training set and greatly improved it's performance.

While reward optimization was not a goal of learning, we also made statistics of reward collected by both solutions. Results achieved for 2x2 configuration are presented on Figure 4b. Since supervised learning gathers knowledge slower it's also starting to gain profits later. Despite rough start, our approach manages to leverage it's experience and start to achieve profits equal to those gathered by reinforcement learning.

4.2. Comparison in a 3x3x3 System

Second experiment was conducted in model with 3 layers where each of them contained 3 machines. This case allows to test algorithms capabilities in more complex environment, although we have to remember that even the simplest examples from real manufacturing will probably be much more complicated than this one. Received results are presented in Figure 5a. Both methods performed better than in previous example. Reinforcement learning managed to stabilize quickly and keep the number of products in buffers on almost the same level for the whole simulation. Supervised learning like before had problems on the beginning. However, around turn 200 it had break through and managed to decrease the number of products waiting for processing to just a few. With such performance it is maintaining an advantage over reinforcement learning to the end of the simulation.

In Figure 5b we can see rewards achieved by learning algorithms during experiment. As we can see even without maximizing received rewards, both algorithms managed to work out profits. Reinforcement learning again achieved better results on this chart. It's due to the fact that it has been evenly working for the whole time. Supervised algorithms despite of having almost empty buffers in second half of the algorithm didn't manage to catch up. Although in later stages of experiment profit gain of both algorithms was equal, penalties for the first few orders put supervised learning too much behind.



(a) (b)

Figure 5. Results for 3x3x3 configuration.

5. State of the art

Since machine scheduling problem is one of the most crucial ones for today's manufacturing it has gathered a lot of attention in the recent years. As a result of this, vast diversity of algorithms aiming to optimize the whole process has emerged. Solutions based on various approaches included among others mixed-integer programming presented in [7] to solve flowshop problem using two criterias, makespan and flow-time. Genetic algorithm approach was considered to solve n-jobs, m-machines setup. Results of this try compared with simulated annealing and neightbourhood search were presented in [8].

There are several works considering reinforcement learning with Q-learning in particular as a possible solution to machine scheduling problem. Amongst them usually multi-agent approach, where each machine serves as agent, is proposed as possible solution. In [9] author employed multi-agent reinforcement learning for job shop problem in a real life enviroment. It brings good results but operates on small action space containig just 3 elements. Another work which employs reinforcement learning is [4] where centralized version of *reinforcement learning* approach is proposed for scheduling in online flow-shop problem. This solution despite taking longer time for single step evaluation, converges faster than multi-agent variation. There are less works considering supervised learning as solution. In [10] framework trying to learn rules corresponding with creation of optimal solution is introduced. In [11] support version machines (SVM) algorithm was used to solve resource constraint scheduling problem which is generalization of flow-shop problem considered by us.

Unfortunately, since labelling cost is high as it needs a lot of time from the experts and not always is even possible to be done, standard approach of *supervised learning* doesn't fit on-line version of scheduling problem too well. In reactive environments where there is no previously defined training set, most of algorithms are unable to learn. Luckily, proposed architecture allows agents to apply supervised learning autonomously and on-line, so they can tackle such problems too.

6. Conclusion

In this paper we have proposed a multi-agent solution for a *dynamic flow-shop* version. Agents apply autonomously supervised learning on line. Every agent collects experience which forms training data used to learn a classifier applied to chose what type of product should be processed on every machine. Later on experiments were conducted in which presented approach was tested against multi-agent reinforcement learning solution. The goal of the agents was to decrease number of products in buffers. As we can conclude after experiments, supervised learning outperformed reinforcement learning in terms of the assumed goal. Although it needed more time to take a grasp and converge with time it's getting better and better. It stabilizes earlier and can even decrease the number of product waiting in buffers where reinforcement learning was unable to do so in tested scenarios. What it means is that supervised learning is viable solution for dynamic problems if the right architecture is used.

Further research can focus on various directions. One direction is to try to employ presented approach using single-agent system instead of multi-agent one used in this thesis. Second way to go is to introduce some form of communication between agents, to enhance received result even further. The last direction is to use this approach in different problems with dynamic nature and see if it's gathering as good results as in this one.

Acknowledgements

The research presented in this paper was supported by the Polish Ministry of Science and Higher Education under AGH University of Science and Technology Grant 11.11.230.124.

7. References

[1] Sendil Kumar C., Panneerselvam R., *Literature review of jit-kanban system.* The International Journal of Advanced Manufacturing Technology, 2007, 32 (3), pp. 393–408.

[2] Olhager J., Östlund B., *An integrated push-pull manufacturing strategy.* European Journal of Operational Research, 1990, 45 (2), pp. 135–142.

[3] Ouelhadj, D., Petrovic, S., *A survey of dynamic scheduling in manufacturing systems.* Journal of Scheduling, 2008, 12 (4), pp. 417.

[4] Qu S., Chu T., Wang J., Leckie J.O., Jian W., *A centralized reinforcement learning approach for proactive scheduling in manufacturing.* In: *ETFA*, IEEE, 2015, pp. 1–8.

[5] Śnieżyński B., *Agent strategy generation by rule induction.* Computing and Informatics, 2013, 32 (5).

[6] Śnieżyński B., *A strategy learning model for autonomous agents based on classification.* International Journal of Applied Mathematics and Computer Science, 2015, 35 (3), pp. 471–482.

[7] Selen W.J., Hott D.D., *A mixed-integer goal-programming formulation of the standard flow-shop scheduling problem.* Journal of the Operational Research Society, 1986, pp. 1121–1128.

[8] Reeves C.R., *A genetic algorithm for flowshop sequencing.* Computers & operations research, 1995, 22 (1), pp. 5–13.

[9] Beke T., *Multi-agent reinforcement learning in a flexible job shop environment: the vcst case.* Master's thesis, Gent Universiteit, Gent, Belgium 2013.

[10] Ingimundardottir H., Runarsson T.P., *Supervised learning linear priority dispatch rules for job-shop scheduling.* In: *Learning and Intelligent Optimization: 5th International Conference.* Springer 2011 pp. 263–277.

[11] Gersmann K., Hammer B., *Improving iterative repair strategies for scheduling with the {SVM}.* Neurocomputing, 2005, 63, pp. 271 – 292.