Application of Learning Automata for Stochastic Online Scheduling

Yailen Martinez[1,2], Bert Van Vreckem[1*], David Catteeuw[1], and Ann Nowe[1]

[1] Computational Modeling Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium {ymartine,bvvrecke,dcatteeu,ann.nowe}@vub.ac.be
[2] Central University of Las Villas, Santa Clara, Cuba, yailenm@uclv.edu.cu

Summary. We look at a stochastic online scheduling problem where exact job-lenghts are unknown and jobs arrive over time. Heuristics exist which perform very well, but do not extend to multi-stage problems where all jobs must be processed by a sequence of machines.

We apply Learning Automata (LA), a Reinforcement Learning technique, successfully to such a multi-stage scheduling setting. We use a Learning Automaton at each decision point in the production chain. Each Learning Automaton has a probability distribution over the machines it can chose. The difference with simple randomization algorithms is the update rule used by the LA. Whenever a job is finished, the LA are notified and update their probability distribution: if the job was finished faster than expected the probability for selecting the same action is increased, otherwise it is decreased.

Due to this adaptation, LA can learn processing capacities of the machines, or more correctly: the entire downstream production chain.

1 Introduction

Multi-stage scheduling over parallel machines

Batch chemical plants usually consist of a series of one or more processing stages with parallel processing units at each stage. A new trend in production processes is to operate flexible, adaptive multi-purpose plants. We look at an application based on the chemical production plant of Castillo and Roberts [1, 2]. It is a two-stage process with four times two parallel machines, see Figure 1.

Each order (created at P_1 and P_2) must be handled first by a 'stage-1' machine M_{1-} and afterwards by a 'stage-2' machine M_{2-}. At each stage, a scheduler must choose between two parallel machines. *Parallel machines* can handle the same type of tasks, but may differ in speed. The possible choice in

* This research is partially funded by the IWT-SBO Project (DiCoMAS) "Distributed Collaboration using Multi-Agent System Architectures"

parallel machines is depicted by the arrows in the figure. All machines have a FIFO-queue and execute jobs non-preemptively.

Stochastic online scheduling

The length of the jobs varies according to an exponential distribution. Only the average joblength is known by the schedulers. Also, the machines' speeds are unknown. Even the expected processing time of the jobs is unknown. However, when a job is finished, the scheduler has access to its exact processing time.

Moreover, it is not known in advance when a new order will arrive. I.e. we have an *online* scheduling problem. In an offline problem, all product orders are known in advance. An optimal algorithm will find the best feasible schedule if time and memory restrictions allow it to be computed. In an online scheduling problem, an algorithm has to make decisions based on the history (i.e. information of already released or finished jobs) and the current product request. It is obvious this makes for a more challenging problem. Moreover, no algorithm can find the optimal schedule for all possible input sequences.

Approaches

This problem is particulary hard since it is stochastic, online and multi-stage at the same time.

There exist heuristics for online stochastic scheduling in the single-stage scenario. But these cannot be easily mapped to a multi-stage problem, in this case we do not only need the information about the immediate available machines, but also the information about the machines of the coming stages and this, of course, increases the complexity. In Section 3 we discuss one such heuristic.

In the next section, we introduce Reinforcement Learning and Learning Automata. We propose to apply these techniques for difficult scheduling problems such as the one described above. Later, we will compare Learning Automata to the heuristic of Section 3 in an easy setting.

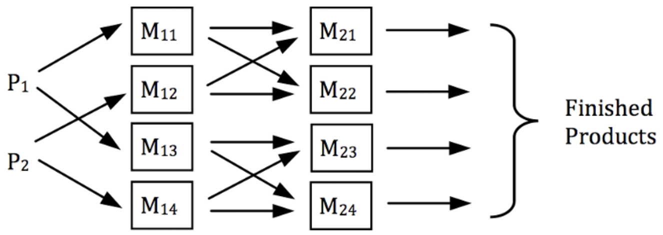

Fig. 1. A two-stage chemical production plant. For both product types P_1 and P_2, there are two parallel machines at the first stage. At the second stage of the process, there are also two parallel machines.

2 Reinforcement Learning

Reinforcement Learning (RL), as noted by Kaelbling, Littman and Moore in [4], 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.

RL is learning what to do in which situation to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trial-and-error. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of RL [3].

In the standard RL paradigm, an agent is connected to its environment via perception and action, as depicted in Figure 2. In each step of interaction, the agent senses the current state s of its environment, and then selects an action a which may change this state. The action 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 reward.

One of the challenges that arise in RL is the trade-off between *exploration and exploitation*. To obtain a high reward, an RL 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.

In many cases the environment is stochastic. This means, (i) rewards are drawn from a probability distribution and (ii) for each current state s and the chosen action a there is a probability distribution for the transition to any other state. As long as the environment is stationary (i.e. the transition and reward probabilities do not change over time) RL agents can learn an optimal policy. This was e.g. proven for the well-known Q-Learning algorithm [5].

In the next section we look at a particular RL method: Learning Automata.

2.1 Learning Automata

Learning Automata (LA) [6] keep track of a probability distribution over their actions.[3] At each timestep an LA selects one of its actions according to its

[3] Here we look only at 'Linear Reward' LA, there are many more described in literature [6], but this is probably the most widely used.

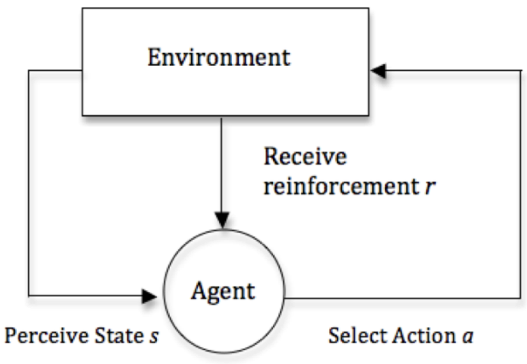

Fig. 2. The RL Paradigm: an agent repeatedly perceives the state of the environment and takes action. After each action the agent receives a reinforcement signal. The goal of the agent is to collect as much reward as possible over time.

probability distribution. After taking the chosen action i, its probability p_i is updated based on the reward $r \in \{0, 1\}$, see Equation 1, first line. The other probabilities p_j (for all actions $j \neq i$) are adjusted in a way that keeps the sum of all probabilities equal to 1 ($\sum_i p_i = 1$), see Equation 1, second line. This algorithm is based on the simple idea that whenever the selected action results in a favorable response, the action probability is increased; otherwise it is decreased.

$$\begin{aligned} p_i &\leftarrow p_i + \alpha r(1 - p_i) - \beta(1 - r)p_i \ , \\ p_j &\leftarrow p_j - \alpha r p_j + \beta(1 - r)\left(\tfrac{1}{n-1} - p_j\right), \quad \forall j \neq i. \end{aligned} \tag{1}$$

The parameters α and β ($\alpha, \beta \in [0, 1]$) are the reward and penalty learning rate. In literature, three common update schemes are defined based on the values of α and β:

- Linear Reward-Inaction (L_{R-I}) for $\beta = 0$,
- Linear Reward-Penalty (L_{R-P}) for $\alpha = \beta$,
- Linear Reward-ϵ-Penalty ($L_{R-\epsilon P}$) for $\beta \ll \alpha$.

2.2 Application to Scheduling

To apply LA to a scheduling problem we need to define the actions of all agents, the rewards and the problem's state space. We define an *action* of the LA as submitting a job to one of the parallel machines. Thus, for the problem described in Section 1 we have 6 agents: two receive product orders P_1 and P_2 and decide which 'stage-1' machine will be used. The other four agents receive partially processed jobs from a 'stage-1' machine M_{1-} and send them to a 'stage-2' machine M_{2-}. Note, the agent cannot wait to submit a job and cannot stop a job preemptively. In other settings these could be added as extra actions.

When a job j is completely finished, the two agents that decided the path of that job are notified of this. Based on the completion time C_j and the release times R_j for both stages a *reward* $r \in \{0, 1\}$ is created, see Equation 2. Note, (i) completion time is the time at which the job has finished both stage 1 *and* stage 2, and (ii) release times are the times at which the job starts stage 1 *or* stage 2 depending on the agent.

$$r = \begin{cases} 0 & \text{if } T > T_{avg}, \\ 1 & \text{otherwise,} \end{cases} \tag{2}$$

where the flowtime $T = C_j - R_j$ and T_{avg} is the average flowtime over the last n number of jobs. The larger n is the more accurate the LA's belief of the average flowtime of the jobs. The smaller n the faster the LA will adapt his belief of the average flowtime when for example a machine breaks down or the performance of a machine increases.

For this problem, it is unnecessary to define a *state-space*. From the agents' point of view the system is always in the same state.

3 WSEPT Heuristic for Stochastic Online Scheduling

We do not know of any good approximation algorithm for scheduling problems that are online, stochastic and multi-stage at the same time. For the single-stage case, however, there exists a very good heuristic: Weighted Shortest Expected Processing Time (WSEPT) [7].

It works in the following setup: orders are arriving over time and must be processed by one out of several parallel machines. The objective is to reduce the total weighted completion time ($\sum_j w_j C_j$, for all jobs j). Each time an order arrives, the WSEPT rule submits the job to the machine that is *expected* to finish it first. To this end it polls each machine for its current expected makespan (including the new job). If, for example, all jobs have equal expected processing time, and each machine the same average speed, then the expected makespan is the queuelength (including the currently processed job if any). In [7] lower bounds on the total weighted completion time ($\sum_j w_j C_j$) are given for the WSEPT heuristic.

In the next section we will compare the WSEPT and the Learning Automata in a simple single-stage scheduling task.

4 Experimental Results

4.1 WSEPT Heuristic versus Learning Automata

We ran some experiments on single-stage scheduling with $N = 4, 5$ or 6 identical machines. One scheduler receives a sequence of jobs. The joblengths are generated by an exponential distribution with average $\mu = 100$. The identical machines have unit processing speed $s_i = 1$, for $i = 1, \ldots, N$. I.e. a machine needs 100 timesteps to process an average job.

To make sure the system can actually handle the load, we set the probability of creating a job at any timestep to 95% of the total processing speed divided by the average job length: $0.95 \sum_i s_i/\mu$. To keep things easy, all jobs have unit weight $w_j = 1$.

We tested the following agents on the same sequence of orders:

RND: uniformly distributes the jobs over all machines,
WSEPT: uses the WSEPT heuristic as described in Section 3,
LA: a Learning Automaton as described in Section 2.1 with $\alpha = \beta = 0.02$.

Results

The experiments show that the LA clearly performs better than the RND scheduler. This is not at all to be expected. The optimal (but static) distribution of jobs of equal expected processing length on identical machines is the uniform distribution. Which is exactly what RND uses. However, due to the high variance in processing times, adapting the load distribution is more efficient at keeping the queues short.

Obviously, WSEPT outperforms LA. Note, the heuristic uses information which both LA and RND cannot access. The length of the queues over time show that WSEPT balances the load better: queues are 4 to 5 times shorter. On the other hand, the total weighted completion time ($\sum_j w_j C_j$) does not show huge differences between WSEPT and LA (in the order of 0.001 to 0.01).

Although the WSEPT heuristic outperforms the Reinforcement Learning approach, the LA are not useless. WSEPT requires access to more information and only works in single-stage loadbalancing problems. In the next section, we test LA in the multi-stage setting as described in Section 1.

4.2 Multi-Stage Scheduling

We look at two slightly different settings, see Table 1. Setting 1 is copied from [2]. In both cases, the average joblength is 100 and the jobrate is 1/45 for both product types P_1 and P_2. The performance is measured by total flowtime of the jobs through entire processing chain.

The first type of LA we test are Linear Reward-Inaction LA (L_{R-I}). After some time, the queues started growing indefinitely. This was caused by some automata converging prematurely to a pure strategy. I.e. they end up selecting the same action forever. This is due to the fact that L_{R-I} never penalize bad actions ($\beta = 0$). Although this may be favorable for many RL problems, it will almost never be for load-balancing. The only obvious exception is when one machine is able to process all jobs before any new order arrives.

The $L_{R-\epsilon P}$ generated better and better results when ϵ is increased. Finally, when $\epsilon = 1$ we have L_{R-P}, where penalty and reward have an equally large influence on the probabilities. This gives the best results.

The value of α and β, which determines the learning speed, seems best around 0.01 for this problem. Table 2 shows the average policy for each of the six agents. For example, the fourth agent receives jobs partially finished by machine M_{13} and distributes them over M_{23} and M_{24}.

The second setting shows that the agents take into account the time needed for a job to go through all stages. Machines M_{13} and M_{14} are 10 times faster as in the first setting. This does not increase the total system capacity, since the machines in the second stage would create a bottleneck. The result is that the first two agents still favor M_{11} and M_{12}, but slightly less. For example, the first agent in Table 2 distributes 71% of the load on M_{11} in the second setting, as opposed to 74% in the first setting.

Table 1. Processing speeds of all machines for two different settings.

Machine	M_{11}	M_{12}	M_{13}	M_{14}	M_{21}	M_{22}	M_{23}	M_{24}
Speed setting 1	3.33	2	1	1	3.33	1	1	1
Speed setting 2	3.33	2	10	10	3.33	1	1	1

Table 2. Average probabilities of all agents through an entire simulation.

Machine	M_{11}	M_{13}	M_{12}	M_{14}	M_{21}	M_{22}	M_{23}	M_{24}	M_{21}	M_{22}	M_{23}	M_{24}
Setting 1	.74	.26	.69	.31	.76	.24	.50	.50	.77	.23	.50	.50
Setting 2	.71	.29	.61	.39	.77	.23	.50	.50	.78	.22	.50	.50

5 Discussion

Following advantages of LA make them very applicable in difficult scheduling scenarios:

- They can cope with uncertainty: unknown joblengths, unknown future jobs and unknown machine speeds.
- The decisions based on the probability distribution and the updates of those distribution are very straightforward. They can be performed in a minimum of time and require only very limited resources.

The performed experiments show that LA can learn processing capacities of entire downstream chains. Note however that the rewards are delayed.

While waiting for a submitted job to be finished, other jobs must already be scheduled. In our scheduling problem, this is not a problem for the LA. When more stages would be added to the system, the LA could be equipped with so-called eligibility traces [3].

Since LA are very adaptive, it should even be possible to detect changes in processing speed, such as machine break downs.

Finally, when applying any randomization technique (such as LA) to balance a load, one is always better off with many short jobs than very few long ones (cf. the law of large numbers). It remains to be seen how large the effect of fewer but longer jobs will be in our setting.

References

1. Castillo I, Roberts CA (2001) Real-time control/scheduling for multi-purpose batch plants. Computers & Industrial Engineering,
2. Peeters M (2008) Solving Multi-Agent Sequential Decision Problems Using Learning Automata. PhD Thesis, Vrije Universiteit Brussel, Belgium.
3. Sutton R, Barto A (1998) Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.
4. Kaelbling LP, Littman M, Moore A (1996) Reinforcement Learning: a survey. Journal of Artificial Intelligence Research 4: 237-285.
5. Watkins C, Dayan P (1992) Technical note Q-Learning. Machine Learning. Springer Netherlands 8: 279-292.
6. Narendra KS, Thathachar MAL (1974) Learning automata - A survey. IEEE Transactions on Systems, Man, and Cybernetics, SMC-4(4): 323-334.
7. Megow N, Uetz M, and Vredeveld T (2006) Models and algorithms for stochastic online scheduling. Mathematics of Operations Research, 31(3): 513-525.