Relazione finale collaborazione occasionale

Gianvito Losapio

July 12, 2023

Titolo della collaborazione

Distributed Reinforcement Learning for Industrial Production Systems

Responsabile

prof. Marcello Restelli

Sommario dell'attività svolta

The work carried out during the collaboration was focused on the analysis and the implementation of distributed reinforcement learning algorithms in the context of a specific problem in the industrial production systems - the job shop scheduling problem.

Starting from a specific use case provided by a company, suitable methods characterized by decentralized, asynchronous execution have been searched throughout the literature. A selection of algorithms have been implemented and tested over a simulator of the production system, trying to prove their feasibility and efficiency.

In the following paragraphs, a more detailed description of the problem is presented and the use of reinforcement learning in such context is motivated. Afterwards, the specific use case is introduced with a description of the simulator and the architecture of the reinforcement learning system.

- *The job shop scheduling problem in the era of Industry 4.0*

Industry 4.0 is revolutionizing the way companies manufacture, improve and distribute their products. Manufacturers are integrating new technologies, including Internet of Things, cloud computing and machine learning into their production facilities and throughout their operations. These smart factories are equipped with advanced sensors, embedded software and robotics that collect and analyze data and allow for better decision making.

Modern industrial production systems are thus characterized by increased automation, predictive maintenance, self-optimization of process improvements and, above all, a new level of efficiencies and responsiveness to customers not previously possible.

Industry 4.0 concepts and technologies can be applied across all types of industrial companies, including for example job shops, i.e., manufacturing systems that handle job production. Job shops are composed of different machines aggregated in shops by the

nature of skills and technological processes involved. Therefore, each shop may contain different machines, which gives this production system processing flexibility, since jobs are not necessarily constrained to a single machine.

Arising from the context of job shops, the job shop scheduling problem (JSSP) is an optimization problem belonging to the class of optimal job scheduling, in which we are given n jobs $J_1, J_2, \ldots, J_n$ and m machines $M_1, M_2, \ldots, M_m$ with varying processing times and power. The required output is a schedule, i.e., an assignment of jobs to machines, that should optimize a certain objective function (typically the makespan – the total length of the schedule). In the specific variant of JSPP, each job J_i consists of a set of operations $\{O_1, O_2, \ldots\}$ which need to satisfy the following constraints:

- the set of operations has to be processed sequentially in a specific order (this means that only one operation can be processed at a given time)
- each operation has a specific machine (or a set of machines) that it needs to be processed on

Today JSSP has wide applications acting as a high abstraction of modern production environment such as semiconductor manufacturing process, automobile or aircraft assembly process and mechanical manufacturing systems.

From a computational point of view, JSPP is one of the best known combinatorial optimization problems, and it has been proved to be NP-hard [1].

The most popular methods used to solve JSPP are dispatching rules based on heuristics or meta-heuristics such as genetic algorithms or particle swarm optimization [2]. They assume a static manufacturing environment where the information of job shops is completely known in advance, hence outputting a deterministic scheduling scheme without any modification during the entire working process. However, in today's complex and varying manufacturing systems, dynamic events such as the insertions, cancellations or modifications of orders, machine breakdowns, variations in processing times and so on, are inevitable to be considered.

For this reason, reinforcement learning has recently been utilized as an online solution to JSPP, with the objective of handling uncertain events in real time as well as scaling efficiency to larger production systems. Several algorithms have been proposed in the last few years (see [2] for a comprehensive list).

◦ *Solving the job shop scheduling problem with reinforcement learning*

Reinforcement learning (RL) is a type of machine learning in which algorithms learn solutions for sequential decision processes via repeated interaction with an environment.

A sequential decision process is defined by an agent which makes decisions over multiple time steps within an environment to achieve a specified goal. In each time step, the agent receives an observation from the environment and chooses an action. Consequently, the environment may change its state according to some transition dynamics and send a scalar reward signal to the agent. A typical learning objective is to maximise the expected discounted reward in each state of the environment.

The decision process is generally modeled by a Markov Decision Process (MDP), in which the agent observes the full state of the environment, or a Partially Observable Markov Decision Process (POMDP), in which the agent may observe only a part of the full state.

When more than one learning agent is involved in the decision process, a specific field is entered called Multi-Agent Reinforcement learning (MARL). MARL algorithms can be categorised based on various properties, such as assumptions about the agents' rewards (e.g. fully cooperative, competitive, or mixed) or assumptions about the training/execution mode.

In the context of this work, the job shop scheduling problem can be modeled as a Decentralized-POMDP and solved using MARL algorithms. Each machine in the production system corresponds to an agent having access only to local information. Constraints about the communication between agents and the availability of other agents' information are provided by the physical structure of the production system and by the objectives of the problem.

Training and execution are in this case decentralized because each agent must select its own action at each time step, without knowledge of the actions chosen or observations received by the other agents. Finally, the problem is partially observable because, while the formalism assumes that there exists a Markovian state at each time step, the agents do not have access to it. Instead, each agent receives a separate observation at each time step, which reflects its own partial and local view of the world.

◦ *Architecture of the system*

The production system consists of several machines $M_1, M_2, \ldots, M_m$, with specific connections between each other (as shown in Figure 1). Each job J_i consists of the production of a product, which requires a set of operations $\{O_1, O_2, \ldots\}$ called "skills" in this context. Each machine can perform a limited set of skills with a specific duration. Starting from a given entry point of the product in the system, the objective is to complete the job by minimizing the makespan (i.e., the total time required to complete the job).

A simulator of the production system (C++) executes the entire job based on a given configuration of the plant and of the products. A Python interface binds to the simulator and exchange information through URL requests and MQTT messages. Each agent runs in parallel and waits for messages from the simulator. Each step of the simulation is triggered by a MQTT message, which can require a single agent to perform some operations (e.g., selecting an action). The resulting setting is thus characterized by an asynchronous interaction of the agents with the environment, with the learning process distributed across agents.

In order to test state-of-the-art algorithms with external simulators, the client-server architecture provided by the library RLlib [3] has been implemented. In our case, each agent acts as a client and connects to its own server. Every time the agent is asked to perform an action, the agent sends a request to the server. Once the action is received and executed, the agent sends the corresponding reward to the server which periodically

executes the learning process.

Presentazione sintetica dei risultati ottenuti

Two different algorithms have been tested on the specific scenario shown in Figure 1.

The production system is composed of 9 machines arranged in 3x3 grid. The machines denoted as "X" are characterized by a random configuration. The duration of the skills are indicated in the top-right corner of each machine, while the duration of the transport between machines is constant and equals to 1s.

In this case there is only a single product P1 having a fixed entry point as denoted in the figure. The job consists of four different operations.

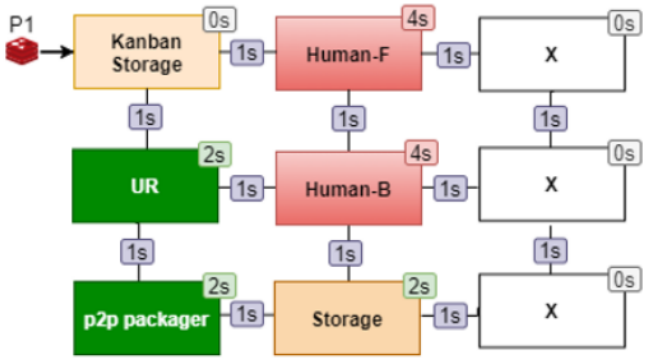


Figure 1: Scenario 1

Each agent decides where to forward the product by selecting one of its own ports connecting to other agents. The observation is given by the product name, the name of the current skill to be executed, all the remaining skills required for that product and a counter specifying how many times the product has been already seen with the same current skill. The reward is the negative duration of the transport executed by the agent.

The implemented algorithms are the following:

- *Distributed Q-learning.* Each agent i has a Q table Q_i. When agent i observes s, takes action a and receives reward $R(s,a)$, it updates the entry $Q_i(s,a)$ of its Q table as

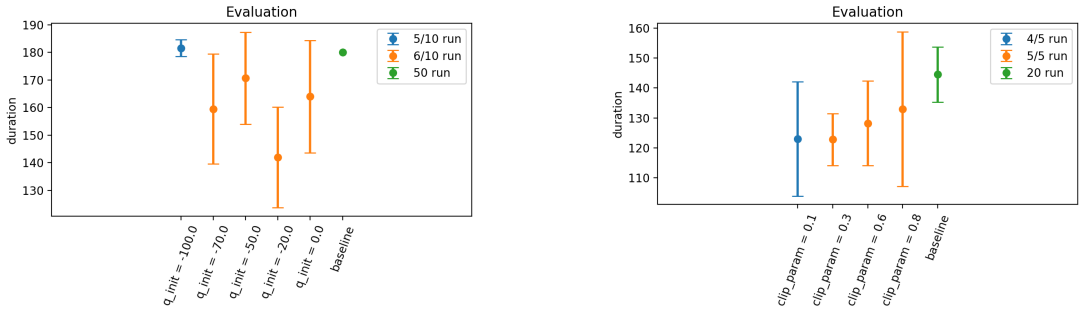
$$Q_i(s,a) = (1-\alpha)Q_i(s,a) + \alpha\Big[R(s,a) + \gamma \max_{a'} Q^{\text{next}}(s^{\text{next}}, a')\Big] \tag{1}$$

  where Q^{next} is the Q table of the next agent (receiving the product at the next turn) and s^{next} is the next state in which the next agent starts acting.

- *Independent PPO (IPPO).* In simpler terms, IPPO [4] is a straightforward implementation of PPO for multi-agent reinforcement learning tasks. Each agent runs

the same instance of the PPO algorithm in a total independent way, i.e., acting as a single-agent.

A Python routine for the grid-search of hyperparameters has been implemented and tested for both algorithms. Figure 2 reports an example of the results with two different parameters: on the left the initialization of the Q values, on the right the clip parameter of the PPO loss.



(a) (b)

Figure 2: (a) Distributed Q-learning, (b) Independent PPO

The results appear promising as in most cases both algorithms outperform the baseline, which is represented on the right by the random policy and on the left by the "minimum-hop" policy (i.e., given the set of agents which can execute the next skill, each agent chooses the closest agent in terms of hops). Notice that the "minimum-hop" policy may be suboptimal, since the closest agent may not be the fastest to perform that skill, thus resulting in a longer final makespan.

It must be noted that: (i) the duration shown in the figure is the sum of the duration of all the operations executed on the product, which may result in a poor correlation with the chosen reward (ii) the labels of the error bars denote the experiments which are concluded, the remaining ones getting stuck in an infinite loop (i.e., the job never gets completed).

In line of these preliminary tests, future works may include:

- Change of the reward to the negative sum of the duration of the operations executed by the agent plus the duration of the transport (to make the learning signal fully correlated to the final makespan).
- Comprehensive tuning of the hyperparameters involved in the algorithms in order to ensure the complete execution of the job (e.g., finding a better trade-off between exploration and exploitation)
- Test of other decentralized algorithms exploiting only local information such as [5], [6].
- Scaling of the results to more challenging scenarios, e.g., with a larger number of machines and with multiple products.

References

[1] Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. Mathematics of operations research, 1(2), 117-129.

[2] Luo, S. (2020). Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing, 91, 106208.

[3] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., ... & Stoica, I. (2018, July). RLlib: Abstractions for distributed reinforcement learning. In International conference on machine learning (pp. 3053-3062). PMLR.

[4] Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., & Wu, Y. (2022). The surprising effectiveness of ppo in cooperative multi-agent games. Advances in Neural Information Processing Systems, 35, 24611-24624.

[5] Zhang, Y., Qu, G., Xu, P., Lin, Y., Chen, Z., & Wierman, A. (2023). Global convergence of localized policy iteration in networked multi-agent reinforcement learning. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 7(1), 1-51.

[6] Kar, S., Moura, J. M., & Poor, H. V. (2012). QD-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus. arXiv preprint arXiv:1205.0047.