Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 277 (2026) 3463–3472

Procedia Computer Science

www.elsevier.com/locate/procedia

7th International Conference on Industry of the Future and Smart Manufacturing
(former International Conference on Industry 4.0 and Smart Manufacturing)

Planning and Scheduling in Flexible and Adaptive Microfactories Using Reinforcement Learning

Fabrizio Albertetti*, Thibault Barthelet, Stéphane Beurret, Luca Laissue, Damien Chappatte, Nabil Ouerhani

Haute Ecole Arc Ingénierie, University of Applied Sciences and Arts Western Switzerland (HES-SO), Rue de la Serre 7, 2610 St-Imier, Switzerland

Abstract

In the context of Industry 4.0, the need for frequent and rapid adaptation of production is becoming increasingly critical. This paradigm shift requires systems capable of autonomously adjusting to dynamic conditions and responding in near real time to the factory's operational state changes. Existing manufacturers, particularly in precision manufacturing where small batch production is prevalent, face significant difficulties in meeting these requirements. In this paper, we focus on two main challenges: (a) the limited flexibility of production systems, which are often not designed for frequent hardware or software reconfiguration, and (b) complexity in the optimization of production flow, which requires advanced planning and scheduling algorithms. To address these challenges, we propose an innovative method based on reinforcement learning (RL) for planning and scheduling optimization. Reinforcement learning enables systems to adapt efficiently and remain robust to changes in the face of change. We validated our method through simulations conducted on a flexible microfactory, *MiLL*, developed at Haute Ecole Arc. The RL agent's performance surpasses a realistic baseline by an average of 8.1%, demonstrating its effectiveness in both planning and scheduling microfactory production. Our method is not confined to precision applications; it can also be applied to any scenario requiring robustness and flexibility.

Keywords: Microfactory; Industry 4.0; Artificial intelligence; Reinforcement learning; Production scheduling

1. Introduction

The integration of the 4th industrial revolution technologies into precision manufacturing demands solutions that address the specific constraints of customized products and low-volume production. This requires highly flexible and rapidly re-configurable production means, both in software and hardware. Manufacturing Execution Systems (MES) must support frequent and swift adaptations of production workflows. To meet these demands, production systems must: (a) be capable of automatic reconfiguration to accommodate new workflow requirements, and (b) enable real-

*Corresponding author.
E-mail address: fabrizio.albertetti@gmail.com

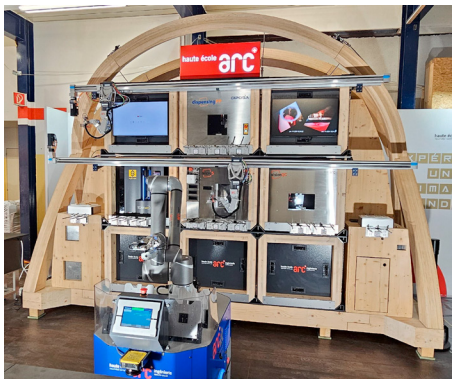

Fig. 1. The MiLL microfactory, designed and implemented by Haute Ecole Arc. Picture from the SIAMS exhibition, April 2024

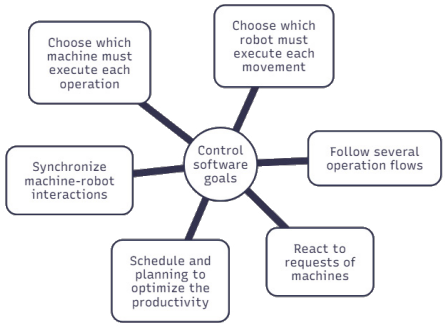

Fig. 2. Goals of the control software of the MiLL

time modifications to production flows –including parts and consumables– through on-demand reconfigurable transit systems.

The Haute Ecole Arc Ingénierie University has launched the MicroLean Lab project to help SMEs in the microtechnology domain to leverage and implement relevant Industry 4.0 concepts. One outcome of this project is the MiLL microfactory prototype. The MiLL microfactory features nine fully reconfigurable slots, each capable of hosting an autonomous machine, and a robotic system for transporting material and parts between slots. The entire setup is managed by a software platform specifically designed to support high flexibility requirements. Furthermore, we have standardized the interface between the control software and the machines to implement the plug&produce concept [1]. This interface defines the command protocols required for synchronizing material transfer operations between robots and machines, ensuring seamless integration and coordination (fig. 1). The MiLL serves as a test and validation environment for different studies and developments, in particular the one presented here.

This article describes the works and results related to the development of a control software that meet the needs listed in fig. 2. This software is composed of several modules, from the scheduling and planning that use AI algorithms [1] to the interface with machines and robots to control them. To our knowledge, no similar concept has yet been implemented in this field.

After a review of the current state of the art in this domain, the methodology applied to the conception of this software is described. Then the main implementation features are presented. Finally, this paper shows and describes the results reached and the futures perspectives of this project.

2. Literature Review

Production scheduling optimization is an old industrial problem [2], and it continues to be an active area of research today [3, 4].

Classical approaches to solving such problem include Branch and Bound (B&B) algorithms, Mixed Integer Linear Program (MILP), and Constraint Programming (CP).

Each of these approaches faces challenges when applied to the scheduling optimization of complex systems [5] : B&B is generally unsuitable for dynamic environments with flexible machines. MILP rarely leads to high-quality solutions, and although CP typically performs better than MILP, it can struggle to find near-optimal solutions.

To move from classical linear production lines to distributed, self-organizing production systems require significant transformations int the development process [6]. These transformations are possible thanks to emerging technologies such as Cyber-Physical Systems [7, 8] and the Internet of Things [9].

[1] Source available here: https://zenodo.org/records/16894482

Physical robots and machines organizations can be abstracted as physical multi-agent systems. Leveraging agent-based models is a modern strategy to support the realization of Industry 4.0 [10, 11, 12].

To build such dynamic systems, the ability to predict and manage perturbations and failures is essential [13, 14].

Recent research has demonstrated the significant potential of reinforcement learning (RL) algorithms in tackling manufacturing scheduling problems. [15] introduced the application of RL to job-shop scheduling using temporal difference learning, while [16] extended this approach to dynamic production environments. More recent studies have demonstrated the effectiveness of deep reinforcement learning techniques–such as actor-critic methods and Proximal Policy Optimization algorithms– in managing complex scheduling constraints [17]. For instance, [18] applied deep RL in smart manufacturing settings, achieving considerable performance improvements over traditional scheduling methods. Additionally, [19] proposed a real-time scheduling framework based on RL, capable of adapting to dynamic conditions in smart factories.

An emerging technique in this domain is the integration of reinforcement learning with digital twins. [20] developed a deep RL-based dynamic reconfiguration planning method for smart manufacturing systems, addressing the challenges of flexible production with reconfigurable machine tools. [21] provided a comprehensive review of AI applications in manufacturing, emphasizing the convergence of RL, digital twins, and cyber-physical systems as key enablers of Industry 4.0. Furthermore, [22] conducted a systematic literature review on the use of deep RL in production systems, identifying key research gaps and opportunities.

Together, these developments highlight the promise of RL as a robust and adaptive solution for scheduling in flexible and dynamic manufacturing environments.

3. Methodology

3.1. Multi-Agent Control Architecture

The control software (Fig. 3) contains all the logic required for the full operation of the microfactory MiLL. Its design is based on a multi-agent architecture with centralized decision-making. The machines and robots in the MiLL are each managed by a dedicated agent, referred to as *machine agent*. The machine agents handle communication with their respective physical counterparts, maintain their operational status up to date, and ensure safe functioning. Each machine agent provides real-time information to the *scheduler agent*, enabling the construction of an accurate and dynamic model of the overall MiLL's state.

The central component of the system is the scheduler agent, which ensures proper synchronization across all modules. The scheduler agent requires two distinct configurations. The first one defines the factory setup, including the specification of machines and robots. The second one outlines the operational flow –a sequence of tasks required to manufacture a part. Since multiple machines can perform the same task, the control software must determine the most suitable machine for each operation. The operational flow is structured as a tree to ensure that all requirements to execute a task are satisfied. The scheduler agent can recursively determine the next operation to perform by starting from the last one.

The scheduler agent communicates with the *AI agent* to determine the action to execute. For this purpose, the scheduler gathers the current state of the microfactory, including machines, robots, and pieces. It sends the data to the AI agent and receives the next action to perform.

The scheduler agent verifies that each new action aligns with the operational flow and ensures that it can be safely executed, i.e., without causing interference or physical conflicts with any device. If the AI agent issues an invalid command, the scheduler agent will continue executing the base scenario using a fallback algorithm. Finally, the validated action is transmitted to its target via the appropriate machine agent or *robot agent*, using the standard interfaces described below.

The machines used in the MiLL are sourced from various manufacturers. One requirement of this project is that all machines must be capable of communicating with the control software. To avoid developing custom interfaces for each machine, a standardized communication protocol is necessary. The Open Platform Communications Unified Architecture (OPC UA) has been chosen as the IT-OT interface protocol, as it is widely adopted in industrial automation and supports extensibility [23].

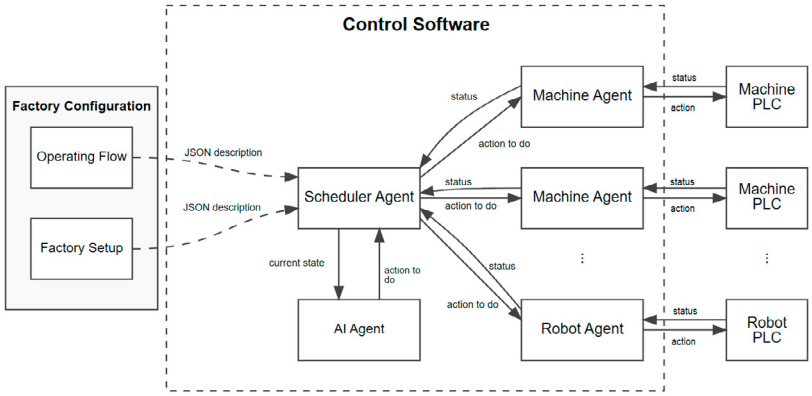

Fig. 3. Architecture of the Control Software and its agents. Bloc: software component; plain arrows: data exchanged; PLC: Programmable Logic Controller

In our implementation, an extension of the OPC UA protocol is needed to define a common semantic for issuing commands from the control software during interaction between robots and machines (e.g., loading material into a machine). This leads to a sequence of OPC UA calls represented in Fig. 4. Data can be sent at each call by providing parameters (e.g., type of material, operation to do). Upon completion of the action or in the event of an error, the device updates its status accordingly.

All machines and robots must implement these OPC UA methods and maintain their status variables throughout each operation. By adhering to this standard, new machines can be integrated into the microfactory without requiring any modifications to the control software. This capability is essential to fulfilling the plug&produce paradigm [1].

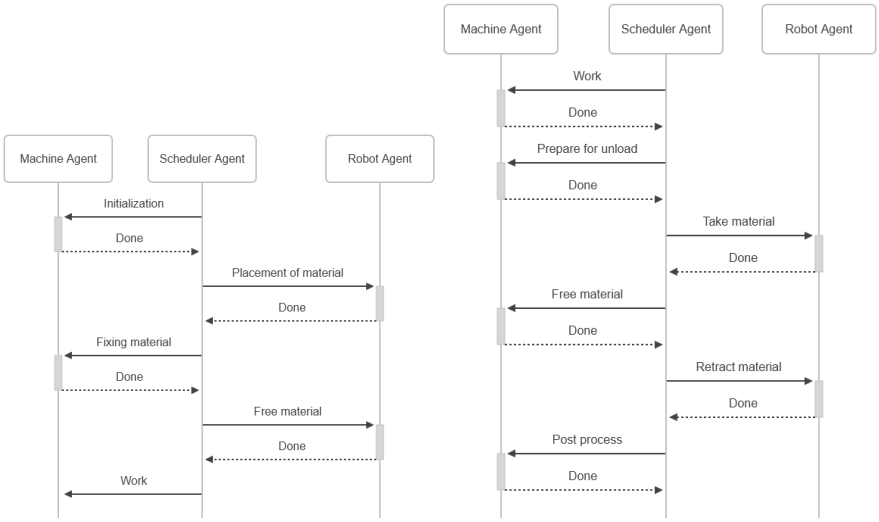

Fig. 4. Sequence diagrams of OPC UA calls between agents. Material input are on the left-hand size and material output are on the right-hand side. The dashed lines indicate the status updates of the machine and robot agents.

3.2. Reinforcement Learning-Based Production Scheduling

The complexity of scheduling in flexible microfactories requires advanced optimization techniques that can handle dynamic environments, managing multiple constraints, and handling uncertain conditions. In this section, we detail our proposed approach to developing an intelligent control system tailored to flexible microfactories. Our solution integrates an AI-based scheduling system using reinforcement learning with a standardized interface to optimize

planning and scheduling operations across the entire microfactory. We begin by establishing a simulation environment that accurately replicates the behavior and operational constraints of the MiLL microfactory. This environment serves as the foundation for training our RL agents and evaluating their performance under realistic conditions.

3.2.1. Agent Performance Evaluation Strategies

With both the hardware and control software clearly defined, the first step is to simulate the system within a controlled environment. It ensures greater consistency and facilitates the design of a suitable reward function.

Then, a critical aspect is to determine how to evaluate the performance of the scheduler agent. The primary metric used is the number of timesteps required to complete a workpiece, as the ultimate goal is to minimize the total completion time (also known as cycle time). However, identifying the optimal number of timesteps in advance is highly challenging due to the complexity of the system, and sometimes only a heuristic function is possible. Additional factors–such as unnecessary or parasitic movements– must also be considered. To address this, we present three complementary evaluation strategies.

The first strategy involves comparing the agent's performance with that of a human operator. This comparison is feasible only under a simplified configuration of the microfactory, with a limited number of machines, and a small set of workpieces. Under these conditions, the optimal number of steps can be determined manually, providing a reliable benchmark.

The second evaluation strategy involves comparing the agent's performance with a baseline. This baseline serves as a simplified approximation of the potential optimal number of steps for a given workpiece. The baseline is calculated using the following formula:

$$T_{total} = T_{load} + \sum_{i=1}^{n} D_i + \sum_{i=1}^{n-1} T_{trans}(M_i, M_{i+1}) + T_{final} + (N_{pieces} - 1) \times (D_1 + D_n) + C_{overlap} \tag{1}$$

where T_{total} is the total estimated time, T_{load} is the initial loading sequence time, D_i is the processing duration at machine i, $T_{trans}(M_i, M_{i+1})$ is the transfer time between machines, T_{final} is the time needed for the final transfer to output, N_{pieces} is the number of workpieces, D_1 and D_n are first and last machine durations, and $C_{ovelap} = \max(0, (N_{pieces} - 2)) \times \sum_{i=1}^{n} D_i$, the piece overlap compensation.

This baseline formula provides a theoretical approximation of the minimum time required to process all workpieces through the microfactory. It accounts for the loading sequence, individual machine processing times, transfer between machines, and the final output transfer. For scenarios involving multiple workpieces, the formula incorporates potential processing overlaps by considering only the durations of the first and last machines for each additional workpiece, assuming ideal pipelining of operations. This overlap accounts for the constraint that each machine can process only one workpiece at a time, thereby reflecting a more realistic lower bound on the total processing time.

The third strategy is a qualitative assessment conducted by a human operator. In this approach, the operator runs the agent on a specific configuration and operating flow, then evaluates its performance on observed behavior. While this method does not determine whether the agent achieves optimal performance, it reliably indicates whether the agent is effective. It is especially valuable for identifying performance issues or behavior anomalies, which can inform further refinement and improvement of the agent.

4. Implementation

This section outlines the design and implementation of the simulation environment, the configuration of the microfactory, and the learning objectives of the RL agent.

4.1. Simulation

The simulation is implemented in a discrete manner, from both a time- and movement perspective. The MiLL layout is represented as a two-dimensional matrix of size 3x5. The first column allows the positioning of the MiLL's input

queue. The second, third and fourth columns represent the 3x3 matrix of the actual microfactory and the respective nests and machines of each element of the grid. The last column represents the output queue.

The machines have the same basic characteristics as the real ones. They are either *free* or *working* with a workpiece. The workpieces are moved in the microfactory by the *SmartMoma*, an automatic guided vehicle (AGV) robot capable of moving through the entire microfactory. It can pick up and drop pallets that hold one piece each. Once the pallet is placed by the SmartMoma in front of a machine, a 5 axis robot *Meca500* can pick the piece on the pallet to fix it in the respective machine. Figure 5 shows an illustration of the simulation with different steps.

Fig. 5. A simplified representation of simulation steps.

This example shows five steps of the simulation. The first step shows the initial state of the simulation. The second step shows the SmartMoma moving to pick a pallet. The third step shows the SmartMoma moving the pallet in front of the machine. The fourth step shows the SmartMoma placing the pallet, and in the last step the Meca500 gets the piece from the pallet to the machine.

Building on this foundation, the various robots, namely the SmartMoma and the Meca500, were integrated into the simulation. The movement of each Meca500 robot is constrained to linear motion along its respective row, reflecting its physical limitations. In contrast, the SmartMoma is capable of moving in any direction but remains restricted by the grid of the factory. Once integrated, the action sets for each robot were defined (Table 1).

Robot	**action sets**
SmartMoma	Move North Move South Move East Move West Interact (pick/place) Load Pallet (from storage) Store Pallet (to storage)
Meca500	Move Left Move Right Interact (with a workpiece or a machine)
System	Skip (no action)

Table 1. Actions sets of the two different robot types available in the simulation.

Next, the configuration of the MiLL was defined. Each machine is assigned a specific position within the grid, along with a fixed processing time and a predefined probability of failure. Each machine is virtually housed within a nest, which can accommodate a limited number of pallets, determined by the number of indexers available. The manufacturing sequence for a workpiece is known and provided to the simulation as a directed graph outlining the sequence of required steps. Every workpiece begins in the input queue and ends in the output queue. The complete manufacturing configuration and microfactory setup are available in the paper's accompanying code repository.

4.2. Learning Objective of the RL Agent

In the reinforcement learning (RL) framework, the learning objective is to discover a policy π that maximizes the *expected return* across a distribution of trajectories. A trajectory τ is defined as a sequence of states and actions $((s_0, a_0), (s_1, a_1), \ldots)$. At each time step t, the agent receives a *reward* $R(s_t, a_t, s_{t+1})$ when transitioning from state s_t to state s_{t+1} by taking action a_t.

The *return* of a trajectory τ is the total discounted cumulative reward: $G(\tau) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})$, where $\gamma \in [0, 1]$ is the discount factor that attenuates the influence of future rewards. The RL agent's objective is to maximize the expected return: $J(\pi) = \mathbb{E}_{\tau \sim \pi}[G(\tau)]$, where the expectation is taken over trajectories induced by the policy π.

A critical component of the learning process is the design of a reward function that effectively guides the agent toward the optimal behavior. The reward is computed as the sum of the individual values defined in Table 2. Several variations of this reward function have been explored, and their implementations are available in the project's code repository.

Event	Value	Description
Time step penalty	-0.01	Base penalty per time step to encourage efficiency
Action cost	-0.003	Small penalty to all non-wait actions to discourage unnecessary or inefficient movements
Designated machine processing	+0.03	Small reward for each step a workpiece spends in the designated machine
Machine processing completion	+0.1	Reward for completing the processing of a workpiece in a machine
Taking a workpiece from input	+0.5	One-time reward for starting to process a new workpiece (unique per workpiece)
Completed workpiece to output	+1.0	Reward for delivering a completed workpiece to the output queue
Unloading from machine	+0.1	Reward for taking a processed workpiece from a machine
Loading to designated machine	+0.5	Reward for loading a workpiece into its designated next machine

Table 2. Components of the reward function. The reward function is designed to encourage the agent to complete the task as quickly as possible while avoiding unnecessary movements.

The reward values in Table 2 were determined through systematic experimentation and iterative refinement. Initially, only Time step penalty was implemented. To help the model learn critical sequences of action, operation-specific rewards were introduced, allowing faster convergence.

5. Experimentation

Our experimentation process consists of two phases: an initial exploration using genetic algorithms, followed by the development of reinforcement learning-based approaches. We initially implemented a genetic algorithm as a baseline to establish foundational performance metrics and to better understand the complexity of the sequential decision-making environment. Despite extensive optimization efforts, including parallelization across 64 CPU cores and incremental sequence length increases, the genetic algorithm proved inadequate for practical deployment.

This limitation motivated a transition to RL, where we systematically evaluated several algorithms, including DQN, PPO, Recurrent PPO, and A3C. Based on empirical performance and stability, we selected proximal policy optimization (PPO) as our primary approach. The following sections detail our implementation choices, training methodologies, and the iterative evolution of our reward function, all aimed at optimizing the agent performance in the microfactory scheduling task.

5.1. Genetic Algorithm Approach

As previously mentioned, the initial approach to solving the scheduling problem involved the use of a genetic algorithm to find optimal action sequences. The agent is represented as a list of discrete actions, which form the genome. To overcome the difficulties of the sequential nature of the environment, the algorithm incrementally increases the sequence length–up to 3000 steps–while preserving and evolving genomes from previous generations.

The algorithm uses tournament selection, two-point crossover, and uniform integer mutation. Candidate solutions are evaluated directly within the environment using a fitness function based on cumulative reward. To accelerate the

search process, the implementation leverages parallelization across 64 CPU cores and incorporates early stopping when a solution reaches a predefined fitness threshold.

Despite these optimizations, the genetic algorithm proved inefficient for practical use. This limitation is primarily attributed to the inherent difficulty of navigating the environment's sequential decision-making complexity.

5.2. Reinforcement Learning

To overcome the limitations of the genetic algorithm, a reinforcement learning (RL) approach is taken. It allows the training of an intelligent agent capable of learning optimal behaviors through interaction with the environment. By design, the RL agent can dynamically adapt to changes in the system–an essential capability for operating within the flexible and evolving context of the MiLL microfactory.

With the simulation environment in place, the next step is selecting the appropriate RL algorithm. Several approaches were explored, including both Q-based and actor-critics algorithms. After extensive experimentation, Proximal Policy Optimization algorithm was selected as the most suitable algorithm. PPO effectively captures the sequential nature of the simulation while remaining fully stateless, which facilitates its integration into the control software.

To ensure valid decision-making, action masking is employed to restrict the agent's choices to only actions that are feasible in the current state. The core architecture for both the policy and the value networks consists of a feed-forward neural network with three hidden layers containing 1024, 512 and 512 neurons. The ReLU activation function is used. Population-Based Training (PBT) is applied to optimize the agent's hyperparameters dynamically during training.

The training process exhibits distinct phases, which are clearly reflected in the learning curves (Figure 6). Initially, the agent undergoes an exploration phase, characterized by long episode lengths (approximately 2000 steps) and negative cumulative rewards. At around iteration 150, a significant performance breakthrough occurs as the agent discovers effective strategies, dropping episode length to approximately 800 steps and achieving positive rewards. Gradual improvements continue until iteration 500, after which the learning curve stabilizes, with occasional minor fluctuations attributed to the stochastic nature of the learning process. To enhance training stability and facilitate learning, curriculum learning is employed: the episode length is initially limited up to 1500 steps and later increasing up to 2000 steps once performance plateaus. This approach proved essential to help the agent overcome the complexity of the sequential manufacturing environment.

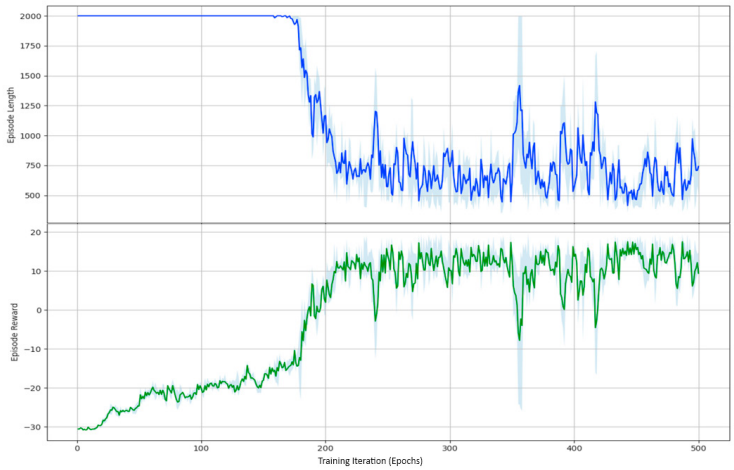

Fig. 6. Training curve of the RL agent over 500 iterations (x-axis), averaged across 6 parallel trials. In blue (top) the average episode length, and in green (bottom) the average episode reward.

6. Results

6.1. Results and Evaluation

Table 3 compares the performance of the RL agent against the baseline.

Environment			# of steps		Difference
# workpieces	# machines	advanced choices	baseline	RL agent	
1	1	no	102†	**97**	**-4.9%**
2	1	no	150†	150	0%
2	2	no	**218**	219	0.5%
2	3	no	262	**257**	**-1.9%**
2	4	no	**338**	344	1.8%
2	4	yes	338	**320**	**-5.3%**
4	4	yes	754	**673**	**-10.7%**
6	4	yes	1170	**1074**	**-8.2%**
Average			416.5	***391.8***	***-3.6%***

Table 3. Performance comparison of the RL agent against the baseline. The baseline represents the optimal time required to complete the task. It is either measured by a qualified human operator (denoted by †) or computed from the equation 3.2.1. A lower number indicates a better performance, and a negative percentage denotes that the agent outperforms the baseline. The # workpieces indicates the number of pieces that are to be produced. The # machines represents the number of machines (operations) a piece has to go through to be completed. Advanced choices denotes that the operating flow specifically allows the choice of order for some specific operations.

The trained agent demonstrates exceptional performance across multiple environments. In average, across all theoretical environments (i.e., without †), it outperforms the baseline by 4.0%, indicating that it successfully learned to perform its task optimally. Although the RL agent is slower than the baseline by 0.1% in the configurations where the agent does not have access to the advanced choices, the RL agent outperforms the theoretical baseline by 8.1% when it can. This indicates that the agent successfully learned to make the best choices to minimize the number of steps needed for the completion of a workpiece. For the two smallest baselines (denoted by †), the comparison indicates that the RL agent is 2.5% faster than a human operator on average.

Finally, the qualitative observation by human operator shows some key points of the algorithm. One of them is the preemptive strategy of the agent, that will move as many workpieces as possible as soon as possible. Another point is the observation of parasitic moves, such as making the Meca500 move back and forth for no apparent reason. This problem is especially prominent in the bigger experiments with 4 and 6 workpieces.

6.2. Discussion and future work

The performance of the agent is very encouraging. The RL agent outperforms a realistic baseline on average of 8.1%. This performance is faster than a naive theoretical optimal solution, indicating that the algorithm successfully both plans and schedules the production of the microfactory.

The agent has a potential for improvements. The parasitic moves are a big problem in an industrial standard. If working on reward scaling could probably help this for bigger experiments, it still proves to be unstable for more generic works. Also, a broader change of layout of the MiLL leads the agent to be completely lost and unable to finish the operational flow. As such, improving the algorithm reliability across multiple environments is the next main step to work on.

7. Conclusion

We proposed an innovative method based on reinforcement learning for the planning and scheduling of microfactories, with a main focus on adaptation and flexibility. Our method was validated on the MiLL microfactory under various scenarios, demonstrating an average improvement of 8.1% over a realistic baseline. However, some enhancements are necessary before it can be deployed in industrial contexts, such as support for higher production volumes, more complex operating flows (e.g. operation with multiple inputs), and human-performed operations. With these improvements, our proposed method could be virtually applied to any application with similar characteristics and needs.

Acknowledgements

The project described in this paper has been funded by the engineering and architecture faculty of the University of Applied Science and Art Western Switzerland (HES-SO) and by Innosuisse.

References

[1] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, J. Ota, Agile assembly system by "plug and produce", CIRP Annals 49 (1) (2000) 1–4. doi:https://doi.org/10.1016/S0007-8506(07)62883-2.
[2] S. M. Johnson, Optimal two- and three-stage production schedules with setup times included, Naval Research Logistics Quarterly 1 (1954) 61–68. doi:10.1002/nav.3800010110.
[3] J. Xie, L. Gao, K. Peng, X. Li, H. Li, Review on flexible job shop scheduling, IET Collaborative Intelligent Manufacturing 1 (2019) 67–77. doi:10.1049/iet-cim.2018.0009.
[4] A. Prashar, G. L. Tortorella, F. S. Fogliatto, Production scheduling in industry 4.0: Morphological analysis of the literature and future research agenda, Journal of Manufacturing Systems 65 (2022) 33–43. doi:10.1016/J.JMSY.2022.08.008.
[5] S. Dauzère-Pérès, J. Ding, L. Shen, K. Tamssaouet, The flexible job shop scheduling problem: A review, European Journal of Operational Research 314 (2024) 409–432. doi:10.1016/j.ejor.2023.05.017.
[6] I. Graessler, J. Hentze, Transformations in product development to enable globally distributed self-organizing production systems, Procedia CIRP 84 (2019) 474–479. doi:10.1016/J.PROCIR.2019.04.212.
[7] S. Zanero, Cyber-physical systems, Computer 50 (2017) 14–16. doi:10.1109/MC.2017.105.
[8] D. Serpanos, The cyber-physical systems revolution, Computer 51 (2018) 70–73. doi:10.1109/MC.2018.1731058.
[9] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, Industrial internet of things: Challenges, opportunities, and directions, IEEE Transactions on Industrial Informatics 14 (2018) 4724–4734. doi:10.1109/TII.2018.2852491.
[10] B. Vogel-Heuser, M. Seitz, L. A. C. Salazar, F. Gehlhoff, A. Dogan, A. Fay, Multi-agent systems to enable industry 4.0, at - Automatisierungstechnik 68 (2020) 445–458. doi:10.1515/auto-2020-0004.
[11] H. Ghorbel, J. Dreyer, F. Abdalla, V. R. Montequin, Z. Balogh, E. Gatial, I. Bundinska, A. Gligor, L. B. Iantovics, S. Carrino, Soon: Social network of machines to optimize task scheduling in smart manufacturing, in: 2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), IEEE, 2021, pp. 1–6. doi:10.1109/PIMRC50174.2021.9569644.
[12] M. Seitz, F. Gehlhoff, L. A. C. Salazar, A. Fay, B. Vogel-Heuser, Automation platform independent multi-agent system for robust networks of production resources in industry 4.0, Journal of Intelligent Manufacturing 32 (2021) 2023–2041. doi:10.1007/s10845-021-01759-2.
[13] A. S. Palau, M. H. Dhada, K. Bakliwal, A. K. Parlikad, An industrial multi agent system for real-time distributed collaborative prognostics, Engineering Applications of Artificial Intelligence 85 (2019) 590–606. doi:10.1016/J.ENGAPPAI.2019.07.013.
[14] T. T. Mezgebe, H. B. E. Haouzi, G. Demesure, R. Pannequin, A. Thomas, Multi-agent systems negotiation to deal with dynamic scheduling in disturbed industrial context, Journal of Intelligent Manufacturing 31 (2020) 1367–1382. doi:10.1007/s10845-019-01515-7.
[15] W. Zhang, W. Zhang, T. G. Dietterich, T. G. Dietterich, A reinforcement learning approach to job-shop scheduling, null (1995). doi:null.
[16] M. E. Aydın, M. E. Aydin, M. E. Aydin, E. Öztemel, E. Oztemel, Dynamic job-shop scheduling using reinforcement learning agents, Robotics and Autonomous Systems (2000). doi:10.1016/s0921-8890(00)00087-7.
[17] C. Liu, C.-L. Liu, C. C. Chang, C. C. Chang, C.-J. Tseng, C. J. Tseng, Actor-critic deep reinforcement learning for solving job shop scheduling problems, IEEE Access (2020). doi:10.1109/access.2020.2987820.
[18] L. Wang, L. Wang, K.-H. Kim, X. Hu, X. Hu, Y. Wang, Y. Wang, S. Xu, S. Xu, S. Ma, S. Ma, S. Ma, K. Yang, K. Yang, Z. Liu, Z. Liu, Z. Liu, W. Wang, W. Wang, Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning, Computer Networks (2021). doi:10.1016/j.comnet.2021.107969.
[19] Y.-R. Shiue, Y.-R. Shiue, Y.-R. Shiue, K.-C. Lee, K.-C. Lee, C. Su, C.-T. Su, Real-time scheduling for a smart factory using a reinforcement learning approach, Computers & Industrial Engineering (2018). doi:10.1016/j.cie.2018.03.039.
[20] J. Huang, S. Huang, S. K. Moghaddam, Y. Lu, G. Wang, Y. Yan, X. Shi, Deep reinforcement learning-based dynamic reconfiguration planning for digital twin-driven smart manufacturing systems with reconfigurable machine tools, IEEE Transactions on Industrial Informatics (2024). doi:10.1109/tii.2024.3431095.
[21] R. X. Gao, J. Krüger, M. Merklein, H.-C. Möhring, J. Váncza, Artificial intelligence in manufacturing: State of the art, perspectives, and future directions, CIRP annals (2024). doi:10.1016/j.cirp.2024.04.101.
[22] M. Panzer, M. Panzer, B. Bender, B. Bender, Deep reinforcement learning in production systems: a systematic literature review, International Journal of Production Research (2021). doi:10.1080/00207543.2021.1973138.
[23] W. Mahnke, S.-H. Leitner, M. Damm, OPC unified architecture, Springer Science & Business Media, 2009.