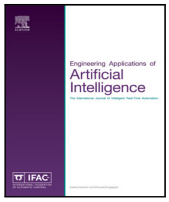
Engineering Applications of Artificial Intelligence 112 (2022) 104868

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



Reinforcement Learning in Manufacturing Control: Baselines, challenges and ways forward

Vladimir Samsonov [a,*], Karim Ben Hicham [a], Tobias Meisen [b]

[a] Institute of Information Management in Mechanical Engineering, RWTH Aachen University, Aachen, Germany
[b] Chair of Technologies and Management of Digital Transformation, University of Wuppertal, Wuppertal, Germany

ARTICLE INFO

Keywords:
Combinatorial optimization
Reinforcement Learning
Manufacturing Control
Job shop scheduling
Production simulation
Multi-Instance Ranked Reward

ABSTRACT

The field of Neural Combinatorial Optimization (NCO) offers multiple learning-based approaches to solve well-known combinatorial optimization tasks such as Traveling Salesman or Knapsack problem capable of competing with classical optimization approaches in terms of both solution quality and speed. This brought the attention of the research community to the tasks of Manufacturing Control (MC) with combinatorial nature. In this paper we outline the main components of MC tasks, select the most promising application fields and analyze dedicated learning-based solutions available in the literature. We draw multiple parallels to the current state of the art in the NCO field and allocate the main research gaps and directions on the perception, cognition and interaction levels. Using a set of practical examples we implement and benchmark common design patterns for single-agent Reinforcement Learning (RL) solutions. Along with testing existing solutions, we build on the ranked reward idea (Laterre et al., 2018) and offer a novel Multi-Instance Ranked Reward (m-R2) approach tailored to MC optimization tasks. It minimizes the reward shaping effort and defines a suitable training curriculum for more stable learning by separately tracking the agent's performance on every scheduling task and rewarding only policies contributing towards better scheduling solutions. We implement all solution design patterns as a set of interchangeable modules with a shared API, unified in a benchmarking framework with the focus on standardization of training and evaluation processes, reproducibility and simplified experiment lifecycle management. In addition to the framework, we make available our discrete-event simulation of a job shop production.

1. Introduction

Efficient Manufacturing Control (MC) is a crucial tool for production companies to meet demanding and fast-changing customer requirements. To grasp the potential of flexible production setups with high levels of automatization to its full extent, it is important to have autonomous decision making in the production planning and execution phase. At the same time, it proves to be a challenging task to have both resilient and efficient real-life MC in place. One aspect of the challenges related to MC problems of combinatorial nature is the NP-hardness, meaning that no known algorithms can find solutions using at most polynomial amount of computational resources with respect to the problem size. Finding solutions requires the usage of problem-tailored optimization algorithms traditionally developed in the field of Operation Research (OR). Recently, methods of Machine Learning (ML) (Bishop, 2006) and Reinforcement Learning (RL) (Sutton and Barto, 2018) have been used to address classical problems of OR such as combinatorial optimization (Mazyavkina et al., 2020; Vesselinova et al., 2020). At the same time, ML and RL methods succeeded in complex games, such as Go, where super-human performance was believed to be hardly achievable, if not impossible, for computers. These developments for applications with prohibitively large solution spaces and long planning horizons allow us to assume that new ML- and RL-based methods may achieve in the future the level of flexibility, adaptivity, speed, and precision required to efficiently manage the wide variety of MC tasks with no or little human involvement.

High research interest in RL applications for MC tasks resulted in considerable recent progress in the field (Altenmüller et al., 2020; Hu et al., 2020; Liu et al., 2020; Park et al., 2020; Gannouni et al., 2020; Zhang et al., 2020). Yet, multiple research possibilities remain open. A **first contribution** of this work is a systematic overview of recent advances and existing challenges in the field of RL applications in MC. In doing so we conduct a systematic analysis of MC tasks (Section 2), general state of the art in the field of neural combinatorial optimization (NCO) (Section 3), as well as the current state of RL solutions in MC (Section 4). A comparative analysis of the current state

Abbreviations: m-R2, Multi-Instance Ranked Reward; L2D, Learning to Dispatch

* Corresponding author.
E-mail addresses: vadimir.samsonov@ima.rwth-aachen.de (V. Samsonov), meisen@uni-wuppertal.de (T. Meisen).

https://doi.org/10.1016/j.engappai.2022.104868
Received 26 April 2021; Received in revised form 16 March 2022; Accepted 3 April 2022
Available online 3 May 2022

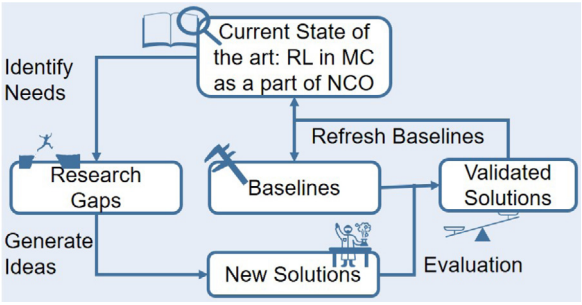


Fig. 1. General workflow of the study.

in considered fields allows us to draw multiple parallels between NCO and RL applications in MC and outline important solution requirements as well as possible research directions (Section 5).

Conducted research in the first part of the work highlights the need for a set of common baselines to significantly reduce the effort required for the development and comparative evaluation of new ideas in the field. Currently only a limited number of implemented RL solutions for MC are made available, while shared implementations tend to be highly integrated solutions that are hard to refactor and compare. Addressing this gap with a set of common solution patterns for RL in MC unified into one benchmarking framework introduced in Section 6 is a **second contribution** of our study. Modular design and unified API of our benchmarking framework allow to combine multiple ideas without much development overhead. Predefined training and evaluation routines combined with reproducibility ensure the comparability of conducted experiments. Structured analysis, enhancement and combination of proposed common solution patterns result in a baseline solution introduced in Section 7. We ensure that proposed solutions satisfy the main requirements allocated in Section 5. Additionally, we implement a discrete-event simulation of job shop production, featuring a modular design and allowing to compare different state space, action space, reward shape and state embedding designs with few changes. Instead of implementing everything from scratch, the designed flexibility of the benchmarking framework enables simple integration of existing solutions, as demonstrated with the approach proposed by Zhang et al. (2020), or to extend the functionality of existing code to test new ideas with minimum effort. Rather than merely enforcing a fixed evaluation methodology with benchmarking instances for a certain job shop production use case, we ensure that custom evaluation approaches and benchmarking tasks can be integrated into the framework. This creates a flexible baseline that can be rerun in different settings and adjusted freely to current research needs.

A **third contribution** of our work is a new Multi-Instance Ranked Reward (m-R2). It is an extension of the ranked reward approach proposed by Laterre et al. (2018) and allows us to enhance the provided baseline solution for RL in MC. The proposed reward design is tailored specifically for MC tasks. Conducted experiments demonstrate that it can provide consistently good results while keeping the reward engineering effort low. All implemented key design solutions, as well as the benchmarking framework, are made publicly available. We hope it will foster further development of openly accessible baselines for the key RL applications in MC and enable faster design iterations and robust comparative evaluations of new ideas in the future.

Fig. 1 summarizes the workflow of our work discussed in detail in the following sections.

2. Manufacturing control and beyond

MC is a key part of in-house Production Planning and Control (PPC) responsible for achieving the targets set by production planning despite disruptions (Wiendahl, 1997). Production disruptions are often unforeseeable (e.g., machine breakdown) and require a timely response to maintain high logistic performance and minimize related logistic costs. As stated by Lödding (2013), internal logistic performance can be defined as throughput time or makespan, due date deviation and due date reliability. Aspects of internal logistic cost are defined as inventory, utilization rate and output rate. Both logistic performance objectives and logistic costs are well formalized and influence the competitiveness of a company, turning it into a set of suitable objective functions in optimization tasks for MC.

MC includes three main activities (Lödding, 2013):

- Order release: decision on the point in time and the sequence of orders to be released for production, therefore determining the real input into the production.
- Sequencing: arranges the sequence in which order's operations are produced on machines.
- Capacity control: determines the required working hours to produce the planned orders and assigns workers to machines.

A combination of order release and sequence planning activities defines the actual production schedule. Therefore, actual short-term production scheduling is an important output of MC.

MC tasks can have various levels of complexity and flexibility depending on the type of manufacturing processes. Manufacturing organization principles oriented towards single-parts or small series production can offer the highest potential for self-controlled logistic and production processes. On the contrary, the production principle of flow production, which is more oriented towards large series, with its mostly rigid interlinking of production and transport systems, leaves too little flexibility to completely grasp the concept of autonomous decision making in production (Scholz-Reiter and Höhns, 2007).

Current studies concentrate mainly on applications of RL to the components of MC responsible for a significant part of short-term in-house PPC (Altenmüller et al., 2020; Gannouni et al., 2020; Hu et al., 2020; Liu et al., 2020; Park et al., 2020; Zhang et al., 2020). However, MC goes hand in hand with the external PPC component covering planning, coordination, and control of the external supply chain. Additionally, a more long-term perspective of PPC responsible for the pre-planning of the production program for several planning periods (production program planning) and estimating the required resources and materials (production requirements planning) play an important role in fulfilling the key logistic objectives while minimizing the costs (Luczak and Eversheim, 1999). We believe that similar to RL in MC, RL methods can be adapted to such tasks as planning and control of global manufacturing and logistic networks. Those tasks combine the essence of combinatorial optimization with high uncertainty levels and constitute open research possibilities for the development of learning-based planning and scheduling solutions.

3. RL in combinatorial optimization

Scheduling tasks can be formulated as combinatorial optimization problems. Along with traditional heuristic and metaheuristic methods, RL can be used to learn solution strategies for combinatorial optimization problems by itself. This direction of research is often referred to as NCO. Even if applied to different problems, the ideas of learning-based methods designed specifically for combinatorial optimization tasks are often transferable to the area of MC. One of the examples of successful idea transfer from NCO to MC domain set in this work is the development of an m-R2 reward design based on the idea coming from the NCO study for knapsack problem by Laterre et al. (2018). Moreover, analysis of such applications can highlight current "blind spots" and future research directions for RL applications in MC. We elaborate more on the possible connections between NCO and RL applications in MC in Sections 5 and 7.3.4. This section summarizes briefly the main ideas and recent developments in the NCO field of the potential relevance for the MC domain. For a more comprehensive survey on the topic please

Table 1
Overview of NCO RL-based solutions.

Study	Problem	Algorithm	State-space representation and related embeddings	Action space and mapping to the problem
Bello et al. (2016)	Traveling Salesman Problem (TSP) and Knapsack problems.	REINFORCE with pointer network.	Sequence of graph nodes (knapsack items or cities), no handcrafting is involved. *Problem size invariant* state embedding.	Direct mapping of input (graph nodes) to output (graph nodes) with pointer network. *Problem size invariant* because of the neural architecture.
Khalil et al. (2017)	Minimum Vertex Cover, Maximum Cut and TSP problems.	Q-learning with structure2vec graph embedding network.	Sequence of graph nodes (vectors), no handcrafting is involved. *Problem size invariant* state embedding.	Direct mapping of actions to graph nodes. *Problem size invariant* because of the neural architecture.
Kool et al. (2019)	Vehicle Routing Problem (VRP), Orienteering Problem and Prize Collecting TSP.	Enhanced approach from Bello et al. (2016). Encoder utilizes solely an attention mechanism.	Sequence of graph nodes (vectors), no handcrafting is involved. *Problem size and sequence invariant* state embedding.	Direct mapping of input (graph nodes) to output (graph nodes) with encoder/decoder architecture. *Problem size invariant* because of the neural architecture.
Ma et al. (2020)	TSP with time windows.	REINFORCE with hierarchical graph pointer network.	Matrix of vectors from considered city to all other cities, no handcrafting is involved. *Problem size invariant* state embedding.	Direct mapping of input (graph nodes) to output (graph nodes) with graph pointer network. *Problem size invariant* because of the neural architecture.
Pierrot et al. (2019)	Tower of Hanoi puzzle.	AlphaZero and Neural Programmer-Interpreters.	Handcrafted vector representation of the puzzle state. Problem size and sequence sensitive state embedding.	Indirect mapping of actions to disks and towers of the puzzle. *Problem size invariant.*

refer to dedicated reviews (Mazyavkina et al., 2020; Vesselinova et al., 2020).

One of the first dedicated NCO approaches is proposed by Bello et al. (2016) and is based on REINFORCE RL with a pointer network to solve the Knapsack- and traveling salesman problem (TSP). The pointer network (Vinyals et al., 2015) deployment allows to read the input sequence step by step and sequentially construct the output sequence over the input using the pointer mechanism, making the approach agnostic to the input size.

Khalil et al. (2017) look into solving graph problems of a combinatorial nature on examples of Minimum Vertex Cover, Maximum Cut and TSP problems. A combination of the Q-learning algorithm with the structure2vec graph embedding network (Dai et al., 2016) allows to exploit graph structure and generalize to problem instances of different sizes. The final solution is iteratively built by greedily picking one node that has not been used yet at each planning step.

Kool et al. (2019) propose to replace the *structure2vec* network with a Graph Attention Network (Veličković et al., 2018) and use an attention-based encoder–decoder for policy parameterization. While using a single set of hyperparameters, the proposed End-To-End RL framework learns solution strategies yielding results close to highly optimized heuristics for such problems as Vehicle Routing Problem (VRP), Orienteering Problem (OP) and Prize Collecting TSP (PCTSP).

Ma et al. (2020) propose the idea of hierarchical RL with pointer networks and graph embedding layers to solve TSP with time window constraints. Instead of learning a single generic policy fulfilling all task requirements, several policy levels arranged in a hierarchical structure address different tasks, from ensuring constraint satisfaction to finding the near-optimal solution of a TSP instance. Another approach using hierarchy and modularity through the adaptation of the Neural Programmer-Interpreters concept (Reed and Freitas, 2016) in NCO is proposed by Pierrot et al. (2019). It uses a set of learned programs that can be combined and recomposed according to the tasks at hand. Strong generalization capabilities of the proposed RL framework are demonstrated on the Tower of Hanoi puzzle. While being trained on a puzzle with two disks, the algorithm generalizes to tasks with considerably more disks.

The NCO field maintains a high pace of scientific research progress and demonstrates multiple outstanding achievements. Recent developments in the field concentrate on methods with extended scalability and generalization capabilities. This is enabled mainly via extending the ability to deal with optimization problems of various sizes without retraining, both on the side of the state-space representation and action space, as well as treating the input elements of optimization tasks and generated solutions as data with underlying structure (graphs), as unordered data (sets), or as a mix of both. Table 1 offers a summary of considered RL-based works in the field of NCO with the focus on key design decisions.

4. RL in manufacturing control

Manufacturing requires a production plan covering a certain planning horizon to coordinate logistics, ensure material availability, and provide sufficient personal resources. At the same time, unexpected changes (e.g., machine breakdown) can render current production plans unviable. Therefore, it is essential to have fast optimization methods capable of adjusting to production deviations within minutes or seconds. RL methods have capabilities to facilitate building such solutions for MC tasks, ensuring a high level of objective satisfaction and low time for solution generation. The first systematic evaluation of possibilities to address MC tasks with Machine Learning (ML) are dated back to the nineties (Aytug et al., 1994). Study of Zhang and Dietterich (1995) looks into addressing the Job Shop Scheduling Problem (JSSP) with RL on the example of installation and testing of payloads in the cargo bay of a space shuttle. Mahadevan and Theocharous (1998) demonstrate adaptation of RL for line production control with the goal of maximum demand fulfillment and low inventory levels. However, a rise of RL use across multiple domains started from the introduction of new End-To-End RL methods (Mnih et al., 2015) capable of achieving super-human performance on multiple tasks with no expert input.

Following the developments of learning-based methods, Waschneck et al. (2018) look into dynamic scheduling of semiconductor wafer processing with RL. It is characterized as a frontend-of-line production with four machines and three technology classes (product types) processed in batches. The optimization goal is to minimize the cycle time spread (CTS) for each of the three technology classes. A custom rule-based scheduling approach is selected as a baseline. The stochastic component that naturally arises in production through small variations is modeled by varying the processing time for each operation. The authors report difficulties with adopting a single RL agent relying on a Deep Q-Network (DQN) (Mnih et al., 2015) to control the job shop floor. A Multi-Agent Reinforcement Learning (MARL) setup with each machine controlled by a separate dispatching RL agent is introduced as a mitigation strategy. Every agent has a view on the production state,

including statuses of machines and work in progress (jobs). Additionally, information on machine capabilities and availabilities is given. All state-related information is assembled in a vector representation and given directly to each RL agent. Available actions allow the machine to remain idle or to directly choose the index of a certain waiting job to start processing.

Altenmüller et al. (2020) extend the given approach by introducing time constraints for each operation. Park et al. (2020) address a similar optimization task with an enhanced MARL setup. It uses a selection rule based on maximum Q-values in case several agents compete for the same resource. Unexpected machine breakdowns introduce a stochastic component into the task. Along with a set of rule-based scheduling approaches, a Genetic Algorithm (GA) metaheuristic is deployed as a performance baseline for makespan minimization. Additionally, generalization capabilities of the intelligent scheduling system are investigated by applying RL agents to new scheduling instances with more machines and without any additional training.

Lang et al. (2020) propose to split the selection of operation sequences and machines in a job shop production between two RL agents. All elements of the task are considered to be deterministic. Agents are trained on a discrete-event simulation and supplied with separate vectors of handcrafted state representations. The proposed solution is compared against the Greedy Randomized Adaptive Search Procedures (GRASP) metaheuristic (Rajkumar et al., 2010) and is shown to yield better solutions for JSSP instances of sizes up to eight machines and eight orders.

Kuhnle et al. (2020) use MARL setup with Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) agents to control the transportation of orders between an entry buffer and intermediate buffers on machines in a job shop environment. The stochastic nature of the manufacturing is modeled via random machine breakdowns and deviations in processing ties. Baselines are represented by a set of rule-based scheduling approaches. The authors present several designs of state space, action space, and reward, as well as an evaluation of their influence on the resulting performance of the RL system. All proposed state-space representations are consolidated in a vector form directly given to dispatching agents and include information on available actions, dispatcher positions, states of machines, remaining processing times, loaded buffer spaces, as well as statistics on the average waiting time per machine. The action space depends directly on the number of entry and machine buffers. Additionally, a reward design for multi-objective optimization covering maximization of machine utilization and minimization of order waiting time is attempted.

Liu et al. (2020) solve job shop scheduling by using Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) in a MARL setup with one separate agent assigned to each machine. Instead of directly mapping agent actions to orders, an agent chooses a dispatching rule, such as picking an order with the shortest processing time (SPT) or first in first out (FIFO). The state space consists of a processing time matrix, a Boolean matrix containing assigned operations to related machines, and a Boolean matrix denoting finished orders. State embedding is achieved by transforming all three state matrices via a Convolutional Neural Network (CNN). This way the authors attempt to capture discriminative state features and to facilitate the agent's learning. Additionally, the adapted reward shape follows an inverse hyperbolic function. The reward value is defined by the gap between the current makespan and the optimum makespan or some estimated value. This reward design can incentify even small improvements in near-optimum regions by introducing a strong reward gradient. A set of rule-based scheduling approaches serves as benchmarks. JSSP instances used for the RL training involve no stochastic changes. Nevertheless, part of the evaluation demonstrates that the resulting RL solution can work with changing processing times and machine breakdowns.

Table 2 provides a compact overview of the mentioned works in the field relying on multi-agent RL approaches. In the overview we pay particular attention to the capabilities of provided solutions to deal with different problem sizes, embrace the properties of unordered and structured data, as well as the level of handcrafting and expert input involved in the learning. Along with numerous multi-agent solutions, several studies demonstrate successful applications of single-agent RL solutions for various tasks of MC summarized in Table 3.

Gannouni et al. (2020) address order sequencing for a single-step production process with sequence-depending setup costs. Considered problem involves no stochastic elements. No alternative benchmarks are involved in the study. Minimization of setup waste is formulated as a Capacitated Vehicle Routing Problem (CVRP) allowing a direct adaptation of the RL solution proposed by Nazari et al. (2018). The involved neural architecture is a pointer network modification (Vinyals et al., 2015) relying on an input size invariant embedding and attention layers coupled with a recurrent neural network (RNN) decoder. Similar to the pointer network, the output size is determined by the input size, making it applicable to problems of various sizes. However, the given architecture does not cover precedence constraints important for more complex production scenarios.

Zheng et al. (2020) demonstrate training of an RL agent in a one flow shop production setting (source domain) for lateness and tardiness minimization with a subsequent transfer of the learned policy to a different flow shop setting (target domain). The transfer is achieved through learning state projections between the source and target domains. The stochasticity of the manufacturing environment is modeled via the random arrival of new orders. No alternative benchmarks are considered in the study.

Samsonov et al. (2021) demonstrate the use of a single RL agent for order sequencing in a job shop environment for makespan minimization. The proposed action space maps available orders to a predefined set of normalized processing durations of fixed size. This transforms the action space to be invariant to the problem size. The study relies on several rule-based scheduling approaches as benchmarks. No stochastic elements related to the manufacturing environment are considered.

One innovative development for scheduling is the use of graph networks for state embedding. Hu et al. (2020) propose a combination of a Petri Net manufacturing simulation and a Graph Convolutional Network (GCN). The usage of machines and sequential processing of the orders within a flexible manufacturing system (FMS) is modeled with a subclass of Petri Nets referred to as a system of simple sequential processes with resources (S^3PR). A specially adapted graph convolutional layer embeds the current state, dynamics, and structure of FMS. Based on the state embedding from GCN, an RL agent conducts online sequencing within FMS to minimize the total makespan. The action space is directly mapped to possible transitions in the system and therefore is dependent on the number of work-cells in the system. Several customized rule-based scheduling approaches and an alternative RL solution are selected as benchmarks. Random arrival of production orders adds stochasticity to the manufacturing environment.

Zhang et al. (2020) propose a problem size invariant approach for solving JSSP with RL. This is achieved by combining a disjunctive graph representation of the JSSP and embedding it into a state representation of fixed size with a Graph Neural Network (GNN). The size of the action space is variable and is equal to the number of orders available for scheduling. At every planning step the policy network iterates over individual state representations for each available order consisting of the given order embedding (graph node) and the whole graph embedding. The output is a probability distribution over all available orders. The proposed approach allows to achieve problem size invariance and is close to a MARL setting with policy parameter sharing. JSSP instances involve no stochastic elements. The developed solution is benchmarked against multiple rule-based scheduling methods and an exact solver.

Additionally, several works use similar RL- or ML-based solution designs in manufacturing-related tasks such as batching and picking orders in a warehouse to minimize the number of tardy orders (Cals et al., 2020) or machine allocation satisfying transportation constraints of automated material handling systems (Kim et al., 2020).

In the next section we reflect on the main findings of Sections 3 and 4, highlight the observed research gaps and outline the main directions considered in our work.

Table 2
Overview of MARL solutions for MC applications.

Study	Application	Goal	Algorithm	State-space representation and related embeddings	Action space and mapping to the problem
Waschneck et al. (2018)	Order sequencing in front-end-of-line production settings.	Cycle Time Spread minimization.	One DQN agent per machine.	Local view per machine, handcrafted vector state representation with *size dependent on the number of orders*, no specific state embeddings.	Direct mapping of actions to orders, *sensitive to the number of orders*.
Altenmüller et al. (2020)	Order sequencing in job shop production settings.	Minimization of time constraints violations.	One DQN agent per machine.	Local view per machine, handcrafted vector state representation with *size dependent on the number of orders*, no specific state embeddings.	Direct mapping of actions to orders, *sensitive to the number of orders*.
Park et al. (2020)	Order sequencing in job shop production settings.	Makespan minimization.	One DQN agent per machine, centralized policy.	Local view per machine, handcrafted vector state representation with *size dependent on the number of orders*, no specific state embeddings.	Direct mapping of actions to orders, *sensitive to the number of orders*.
Lang et al. (2020)	Order sequencing in job shop production settings.	Makespan and total tardiness minimization.	Two separate DQN agents for operation and machine selection.	Global view on the production state, handcrafted vector state representation with *size dependent on the number of orders and machines*, no specific state embeddings.	Direct mapping of actions to orders and machines, *sensitive to number of orders and machines*.
Kuhnle et al. (2020)	Order sequencing and route planning in job shop production settings.	Machine utilization maximization and waiting time minimization.	One TRPO agent per transportation unit.	Global view on the production state, handcrafted vector state representation with *size dependent on the number of machines*, no specific state embeddings.	Direct mapping of actions to source and destination machines, *sensitive to the number of machines*.
Liu et al. (2020)	Order sequencing in job shop production settings.	Makespan minimization.	One DDPG agent per machine.	Global view on the production state over 2D matrices with processing times, required machines and completion information *size dependent on the number of orders and machines*, state embeddings with a CNN.	Indirect mapping of actions to orders and machines , *problem size invariant*.

Table 3
Overview of single-agent RL solutions for MC applications.

Study	Application	Goal	Algorithm	State-space representation and related embeddings	Action space and mapping to the problem
Gannouni et al. (2020)	Order sequencing in single-step production settings with sequence-dependent setup costs.	Setup cost minimization.	REINFORCE with encoder–decoder network (Nazari et al., 2018).	Global view on the production state formulated as graph representation of Multiple Traveling Salesman Problem. No handcrafting is involved. *Problem size and sequence invariant* state embedding.	Direct mapping of input (operations) to output (operations) with encoder–decoder architecture. *Problem size invariant* because of the neural architecture.
Zheng et al. (2020)	Transfer of learned policy for order sequencing to unseen single-step flow shop setting.	Minimization of the average lateness and tardiness.	Unspecified RL algorithm.	Global view on the production state, handcrafted vector state representation with *size dependent on the number of orders and machines*, no specific state embeddings (unless transferred to another domain).	Direct mapping of actions to orders, *sensitive to the number of orders*.
Samsonov et al. (2021)	Order sequencing in job shop production settings.	Makespan minimization.	DQN.	Global view on the production state, handcrafted vector state representation with *size dependent on the number of orders and machines*, no specific state embeddings.	Indirect mapping of actions to orders. *Problem size invariant*.
Hu et al. (2020)	Order sequencing in job shop production settings.	Makespan minimization.	DQN.	Global view on the production state represented as a Petri Net. No handcrafting is involved. *Problem size dependent* state embedding.	Direct mapping of actions to orders, *sensitive to the number of orders*.
Zhang et al. (2020)	Order sequencing in job shop production settings.	Makespan minimization.	Modified Proximal Policy Optimization (PPO) (Schulman et al., 2017).	Global view on the production state formulated as a disjunctive graph. No handcrafting is involved. *Problem size and order invariant* state embedding.	Direct mapping of actions to operations. *Problem size invariant* because of the customized RL algorithm.

5. Analysis of research gaps

Analysis of the main solution qualities in NCO (see Table 1) and RL in MC (see Tables 2 and 3) demonstrates that current RL in MC methods lag behind in important solution aspects, such as the capability to deal with different problem sizes without retraining (changing sizes of the action and state spaces), End-to-End learning of optimization strategies without handcrafting, and exploiting the underlying structure of the problem. This is hard to achieve while relying on established multi-agent or single-agent RL algorithms.

There are multiple possibilities to adopt the established NCO methods to the tasks of MC as demonstrated by Gannouni et al. (2020). One example of building on the ideas originating from the NCO field is the recent work by Zhang et al. (2020). This work offers a novel solution for order sequencing in job shop production settings that simultaneously incorporates problem size invariance, exploits the underlying structure of the problem and eliminates the need for a handcrafted state-space representation.

Yet it is difficult to quantify what improvements new methods can offer compared to the existing solutions. Several factors make a direct comparison difficult. Firstly, to the best of our knowledge, only a limited number of studies made the source code openly available (Zhang et al., 2020; Kuhnle et al., 2020). Secondly, assuming the availability of the code will improve over time, it will remain challenging to select an appropriate benchmark solution among many studies reporting results with customized evaluation approaches and benchmarking instances. Solutions made available are often hard to adjust for particular needs since research-oriented implementations are not primarily designed for easy refactoring and tend to be tightly integrated solutions.

It is common practice in multiple fields to define publicly available baseline problems to facilitate the development, evaluation and comparison of new solutions. This can be accomplished by analyzing common solution designs with subsequent implementation, evaluation and consolidation in a single solution. At the same time, we believe the availability of one or several baseline solutions, evaluation methodology and a set of benchmarking problems is not the ultimate solution. It is far more important to augment the common baselines with a versatile toolset making it possible to adjust all elements of the solution to concrete research needs, rerun baselines on new problems with new evaluation strategies, as well as to incorporate new solutions in baselines with little effort.

This work addresses the outlined problem for one of RL's most studied application cases in MC, covering the order release and sequencing in job shop production environments known as JSSP. The variety of published studies with few publicly available implementations and no unified evaluation strategies makes it especially hard to select the most promising solution approaches and build on them in future research to create more generic RL-based solutions with better performance and the ability to deal with various problem sizes. Each JSSP instance considered in this work can be characterized by a set of jobs $J = \{J_1, \dots, J_n\}$ to be processed on a machine set $M = \{M_1, \dots, M_m\}$. A job J_i consists of an ordered set with n_i operations $O_i = \{O_{i1}, \dots, O_{in_i}\}$ defining the processing precedence constraints. Each operation O_{ij} is to be processed on one machine from a machine set M with a non-negative processing time t_{ij}. The goal is to find a schedule J on M, s.t. the latest endtime of any job J_i from J, known as makespan C_{max}, is minimized (Błazewicz et al., 1994; Brucker, 2007).

The developed baseline is integrated into a benchmarking framework to enable the discussed need for flexibility and compatibility between various RL solutions. It can serve as a strong baseline for rapid development and evaluation of new RL-based solutions. Our contribution is focused on the order release and sequencing tasks of MC (see Section 2) in jobs shop environments. However, the third aspect of MC, namely capacity control, represents another important research direction for learning-based optimization applications in manufacturing. While we do not contribute directly to this aspect, we believe our work can facilitate the development of further baselines solutions. Developed in this work benchmarking framework, manufacturing simulation, and benchmarking workflow provide a well-suited set of tools for addressing the aspects of capacity control activities and more production types with RL while facilitating the simplicity of the development process.

6. Benchmarking framework and methodology

Along with the motivation for a benchmarking framework provided in Section 5, our study relies on multiple experiments to compare common designs of RL solutions in MC. This necessitates the generation of comparable results while keeping the implementation effort low. The whole experimental setup is configured via a single YAML file. This facilitates the comparison between different experimentation runs and the creation of new experiments via an incremental change and combination of the existing configurations. Containing data defines all aspects of the initialization of the relevant RL agent and manufacturing environments, as well as the training and evaluation routines. The provided benchmarking framework features an abstraction layer with object constructors for separate parts of RL-related solutions wrapped in a set of unified APIs for seamless interchangeability, as well as a set of process managers responsible for training and evaluation routines. A factory method design pattern is used to accommodate logic specific to a particular RL solution or manufacturing simulation. The agent factory is responsible for initializing any RL agent implemented into the benchmarking framework, while the agent manager factory contains the logic for training, selecting an action, saving, and loading for each RL agent. Multiple training and evaluation routines can be integrated as interchangeable blocks with the training routine and evaluation routine constructors responsible for ensuring consistent interfaces and selecting the procedures relevant to the given experimentation setup. This allows us to assemble multiple experiments relying on different RL solutions and Deep Learning libraries from a limited set of experimental blocks. One of the central pieces of the framework is a discrete-event simulation of job shop production implemented with maximum flexibility in mind and full support of the OpenAI gym[1] API. It is based on a *SimPy*[2] discrete-event simulation framework (Müller et al., 2020), offers high modularity and facilitates the use of multiple reward, action and state-space designs considered in this study. The experimentation workflow and the interplay between the multiple blocks of the RL solution are summarized in Fig. 2.

Reproducibility and comparability are ensured by storing all run parameters in the single configuration file, fixing random seeds during training runs and using containerized training environments. Scalability and parallelization are achieved by running experiments on a Kubernetes cluster. It is also possible to run the experiments on a local machine without containerization. Experiment lifecycle management is provided through the integration of the Weight&Biases toolkit (Biewald, 2020) into the framework.

All of this reduces the effort required for adopting new RL solutions in the field of MC and allows us to directly compare them against existing baselines. The benchmarking framework, production simulation, configuration files for all experiments and containerized environments are openly available.[3]

Unless stated otherwise, elements of the baseline RL solution inspired by work of Samsonov et al. (2021) and Zheng et al. (2020) are used. It features an RL solution based on a DQN implementation from the Stable-Baselines 2 framework (Hill et al., 2018) with the policy learned by a feed-forward neural network. Total makespan minimization is chosen as the optimization objective. The baseline state-space

[1] https://gym.openai.com.

[2] https://simpy.readthedocs.io/en/latest/.

[3] https://github.com/v-samsonov/optimization-with-rl-in-manufacturing-control.

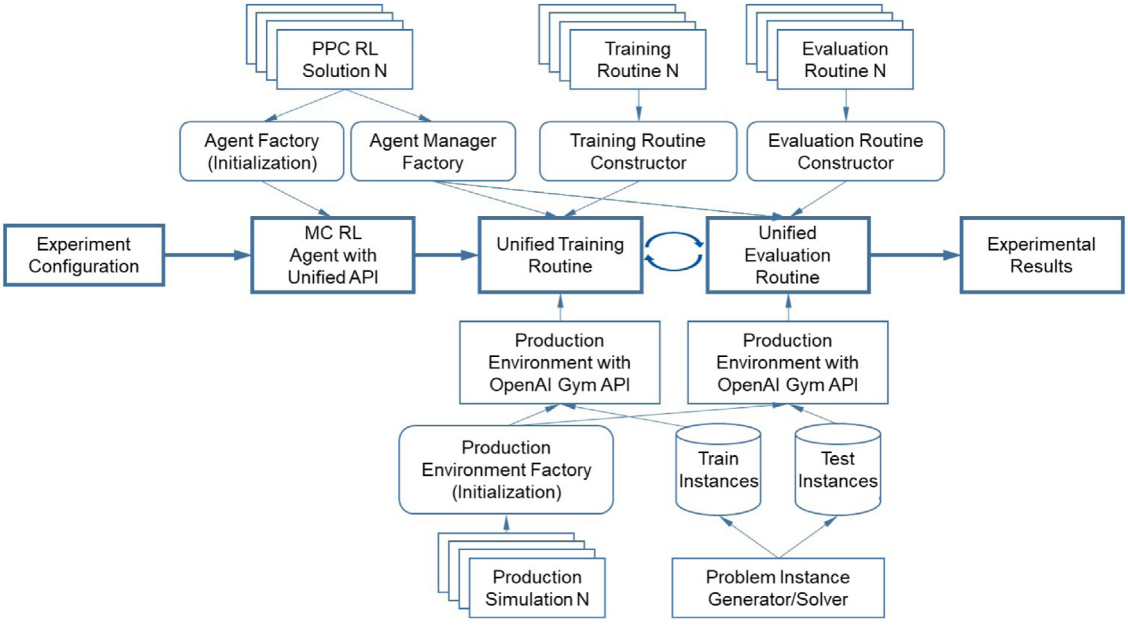


Fig. 2. Outline of the workflow and benchmarking framework architecture. Building blocks with standardized APIs and flexible training/evaluation routines allow the configuration of multiple experiments with minimum changes.

representation consists of a one-dimensional vector including order-centric information on current processing state, remaining backlog, next required machine and duration of pending operations per order, as well as machine-centric information describing machine states and processing backlogs per machine. The action space is problem-size agnostic and relies on an indirect mapping to the scheduling problem as discussed in Section 7.2.1.

We adopt an optimality gap measure as our default metric δ_{opt}. It indicates how much is the makespan C_{max} of a given scheduling solution differs from the optimal solution $C_{max,opt}$ (see Eq. (1)).

$$\delta_{opt} = \frac{C_{max} - C_{max,opt}}{C_{max,opt}} \cdot 100 \tag{1}$$

For training and benchmarking purposes we generate sets of JSSPs covering problem sizes of $6 \times 6 \times 6$, $8 \times 8 \times 8$, $10 \times 10 \times 10$ and $15 \times 15 \times 15$ (*number of orders* $\times$ *number of operations* $\times$ *number of machines*) with 1000 instances each. JSSP generator and solver implementations are provided together with the benchmarking framework. Operation durations are sampled uniformly from the integer interval [1..11]. Reference solutions with minimal total makespan are generated with the CP-SAT solver from the OR-Tools Library (Perron and Furnon, 2020). Given up to two hours of computation time per instance, the CPS-SAT solver manages to solve optimally most of the JSSP instances including the instances of size $15 \times 15 \times 15$.

During the training process the performance of the RL agent is regularly evaluated on a part of the seen training data. The RL agent is saved during training at the point of the best observed evaluation performance to be used for further evaluations and comparisons. Final reported performance per agent is estimated on a set of validation data with 100 JSSP instances not used for the training and is consistent through all experiments for a given problem size. All experiments involve three runs with fixed random seeds.

7. Designing baselines

Every RL application relies on a number of key design decisions concerning the choice of the training *environment*, formalization of *interaction* between the agent and the environment, agent's *perception* of the current state of the environment, as well as the *cognition* capabilities

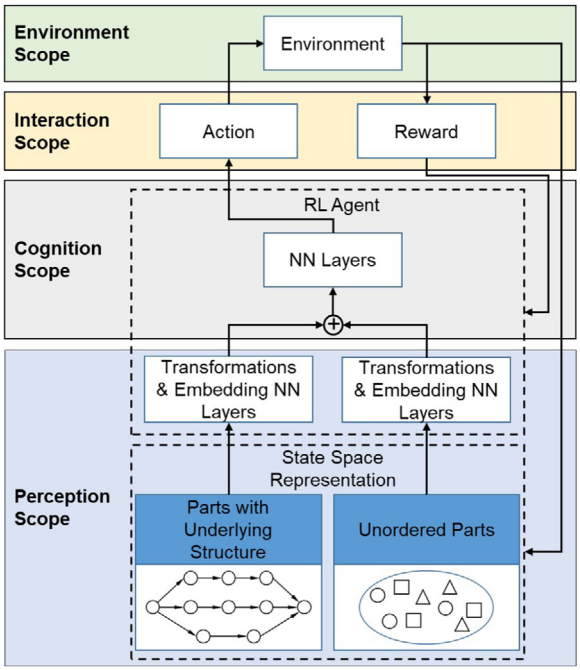


Fig. 3. Breakdown of RL design aspects into environment, interaction, perception and cognition levels.

of the agent ensuring correct evaluation of the raw or prepossessed state information and finding an appropriate solution strategy (see Fig. 3).

We use several practical demonstrations to compare existing ideas, highlight related advantages and challenges, as well as to test suitable solutions.

7.1. Solution designs on RL environment level

It is possible to build a custom RL environment mimicking a certain production setup from scratch (Zhang et al., 2020; Rummukainen and Nurminen, 2019) or use customized timed Petri Nets for production

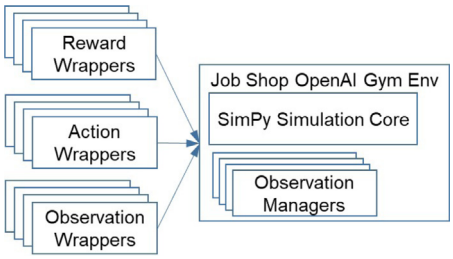


Fig. 4. Modular structure of the proposed SimPy-based production simulation allows for an easy adaptation to the MC task at hand.

simulations (Hu et al., 2020). While offering potentially unlimited flexibility, development and maintenance efforts are growing prohibitively expensive with the increasing complexity of related production processes. Alternatively, a production environment can be modeled using available discrete-event simulation solutions. Established commercial discrete-event simulation frameworks such as Plant Simulation or AutoMod can be used for RL training (Hwang and Jang, 2020). While offering high functionality and optimized performance, license restrictions and limited access to the source code make commercial solutions hard to acquire, share and adjust. Additionally, missing or limited containerization possibilities often make such solutions hard to deploy in the cloud. These facts reduce the reproducibility and thus also the comparability of the results. The use of open-source discrete-event simulation solutions such as *SimPy* (Meurer et al., 2017) and *salabim* (Van der Ham, 2018) alleviates the mentioned restrictions and therefore have a potential to be widely adopted as a standard in the research community (Kuhnle et al., 2020; Lang et al., 2020).

Creation of realistic and complex production environments is a highly relevant research direction. A well-suited simulation should be easily reconfigurable to mimic different production types, offer a high level of scalability, low runtimes, ensure repeatable behavior, have an OpenAI gym interface for communicating with RL agent/s. Apart from that, it should be openly accessible to the research community. Currently only a few production simulations, designed to train RL agents for a narrow set of manufacturing types and MC tasks, are made openly available (Kuhnle et al., 2020; Hoffman et al., 2020). Available solutions tightly integrate simulation logic, reward design, as well as interaction and state-space representations. This creates a substantial entry barrier into the research field, makes existing approaches hard to compare, and turns the development of new ideas into a tedious process.

Similar to the set of OpenAI Gym benchmarks covering classic control tasks or Atari games, a set of reference tasks to compare and develop new RL approaches in the field of MC are needed. A simulation with a set of pre-configured realistic production scenarios, possibly following the Graham classification (Graham et al., 1979), covering different MC tasks and offering various complexity levels can serve as a good baseline to measure and facilitate the progress in the field. Within the scope of this work we develop a fast and flexible *SimPy*-based simulation together with a number of solved scheduling instances of various sizes (see Section 6). Simulation logic, state-space representation, reward and interaction are decoupled from each other in separate modules (see Fig. 4) allowing implementations of new ideas with minimum changes.

The given simulation can mimic different RL environments for MC tasks. For example, along with using the original simulation proposed by Zhang et al. (2020) relying on graph representations of the JSSP, we demonstrate the possibility to mimic it with the proposed simulation by implementing a dedicated observation manager and reward wrapper without changing the core of the simulation itself. Currently our simulation setup is capable of simulating the job shop type of manufacturing only. Nonetheless, the use of *SimPy* discrete-event simulation framework as a backbone of the proposed solution allows extending it to other production types.

An additional line of research might be dedicated to the deployment of RL agents in real production environments. Every real-life deployment will require a simulation closely depicting the considered production setup. The development of simulations for a particular use case requires deep knowledge in simulation and production domains, as well as a high amount of implementation effort. Creation of process models from data with the help of process mining techniques (Van der Aalst, 2016) and their subsequent use for automated generation of discrete-event simulations might significantly reduce the amount of effort required to deploy an RL solution in production. This approach has the potential to quickly increase the number of realistic benchmarks for RL solutions in MC.

7.2. Solution designs on interaction level

Interaction between an RL agent and environment is defined through actions chosen by the agent and resulting reward signals returned by the environment (see Fig. 3). Both the design of the action space and the shaping of the reward signal need to be tailored to a particular use case and can require extensive handcrafting.

7.2.1. Action-space design

Scheduling formulated as a Markov Decision Process (MDP) (Howard, 1960) requires an agent to incrementally schedule operations for all orders over the required machines. This can be achieved through a direct mapping or indirect mapping of the action space to the scheduling problem discussed below.

Direct mapping of action space to the scheduling problem:

In the most generic case, a scheduling system requires an action space directly linked to the set of orders J of the size n_J, set of operations per order O of the size n_O, and set of machines M of the size n_M. It can be encoded into one discrete action vector with the dimensionality equal to the Cartesian product $J \times O \times M$ (Hu et al., 2020; Zheng et al., 2020), or into a set of three independent discrete action vectors of size n_J, n_O, n_M controlled by one or several agents (Kuhnle et al., 2020; Lang et al., 2020). Depending on the scheduling task at hand the dimensionality of the action space can be reduced. JSSP is characterized by strict precedence constraints and direct operation to machine mapping. This allows to reduce the corresponding action space down to a single action vector of dimensionality n_J. Further complexity reduction of the action space is possible through the masking of invalid actions at every planning step.

Indirect mapping of action space to the scheduling problem:

Alternatively, action space can be decoupled from the scheduling problem size as proposed by Samsonov et al. (2021) by introducing an abstraction level similar to the one used in common dispatching rules. Instead of directly picking the next operation an RL agent determines the relative duration of the next operation. An available order with the closest processing time of the pending operation is chosen. For an arbitrary JSSP, the resulting action-space dimensionality is defined solely by the number of time intervals that the maximum operation duration is split into. No action masking is required. Nevertheless, in the case of a more generic scheduling problem, such as Flexible Job Shop Scheduling Problem (FJSSP), additional logic is required to deal with multiple alternative machines.

Results of three independent training and evaluation runs as described in Section 6 for two identical baseline RL agents are presented in Fig. 5. Along with the visualization, the Wilcoxon signed-rank test (Rey and Neuhäuser, 2011) is conducted to investigate the observed differences in performance statistically. The null hypothesis assumes no difference between the medians of the two investigated populations. Observed better performance of indirect mapping compared to the direct action space for the size of $6 \times 6 \times 6$ cannot be

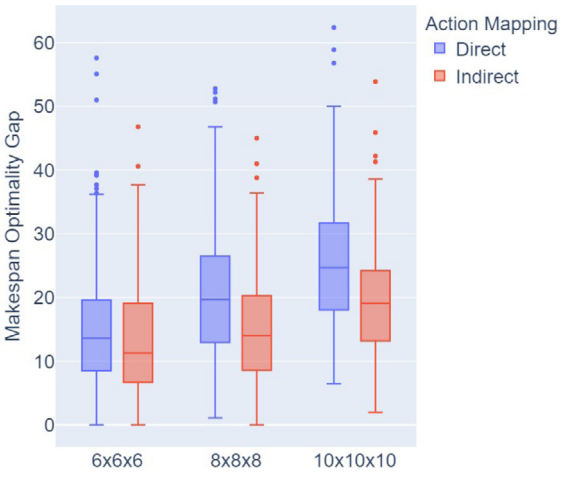


Fig. 5. Comparison of identical baseline RL agents trained with direct and indirect action-space mapping. Three independent training runs per setup with evaluation on 100 unseen $6 \times 6 \times 6$, $8 \times 8 \times 8$ and $10 \times 10 \times 10$ JSSP Instances.

statistically confirmed or rejected since received p-values for the null hypothesis across three different random seeds exceed the threshold value of 0.05. However, for bigger problem sizes the p-value goes well below the 0.05 significance level (max observed p-value for the $8 \times 8 \times 8$ problem size is $1.2 \cdot 10^{-8}$ and $2.9 \cdot 10^{-6}$ for the $10 \times 10 \times 10$ problem size). This proves that the simplicity of the indirect action space results in a noticeable improvement over more common action-space representation for bigger problem sizes. Combined with the ability to deal with changing problem sizes, indirect action space mapping becomes a performant solution.

7.2.2. Reward shape design

A suitable reward shape is expected to incentivize the fulfillment of one or multiple logistic objectives at the same time. The main logistic objectives on the level of job shop are minimization of throughput time, minimization of work in progress (WIP), maximization of machine park utilization and high due date reliability (Lödding, 2013). Minimization of a total makespan for a set of orders is another common optimization objective from the field of operations research (OR) which is tightly coupled with common logistic objectives. As motivated in Section 2, in this study we work on establishing a common baseline for a combination of two key activities in MC: order release and sequencing. Since most of the studies applying RL to components of MC choose makespan as the optimization objective (see Tables 2 and 3), we select it as the optimization objective for all investigations in this study.

An incremental way of solving a scheduling task decomposes it into a large number of smaller decisions on the level of individual orders and operations. Scheduling involves a long planning horizon and requires multiple steps to complete the task. Considering the settings, a well-designed dense reward giving feedback to the RL agent after each step alleviates the credit assignment problem, and allows the RL agent to learn a good policy faster. Multiple RL-based NCO studies (Khalil et al., 2017; Kool et al., 2019; Ma et al., 2020) define a dense reward as a negative cost for taking a certain action. Zhang et al. (2020) build on this idea by defining a reward $r(S_t, a_t)$ from a scheduling action a_t at state S_t as the maximum negative increase of the makespan lower bound (LB) C_{LB_i} for every order $J_i \in \{J_1, \dots, J_n\}$. LB for an order is defined as a cumulative duration of its unscheduled operations in Eq. (2). In this study we refer to the given reward shape as *dense LB reward*.

$$r(S_t, a_t) = max_{J_i \in \{J_1, \dots, J_n\}} \{C_{LB_{J_i}}\} \tag{2}$$

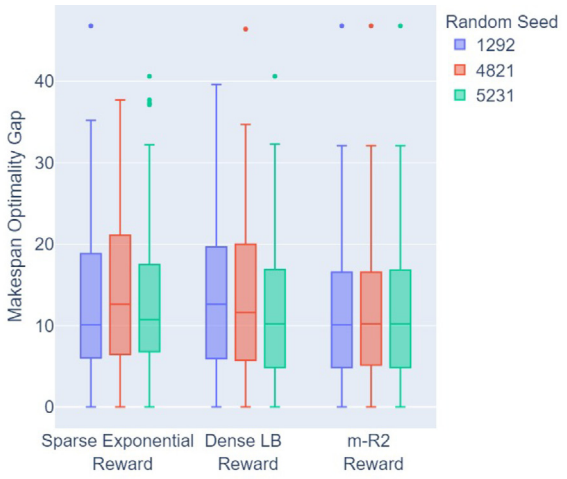
However, it is often hard to estimate the impact of a single action on the overall performance before the end of the scheduling process. For example, the simultaneous start of processing for multiple orders with first operations distributed over different machines is a reasonable planning strategy. Nevertheless, if some of them subsequently arrive at a machine at the same time, a bottleneck is formed. The impact will not manifest itself directly after the action responsible for it and will depend on the subsequent scheduling actions. Since the correct credit assignment to a single action could be difficult to implement, we argue that evaluating all actions at the end of the episode with a single reward can be a better option for learning good scheduling policies. Deployed in this study the baseline reward is a sparse reward proposed by Samsonov et al. (2021). It is given at the end of each episode and calculated as in Eq. (3). We refer to this reward shape as *sparse exponential reward*. It ensures high rewards for solutions with makespan C_{max} near to the optimum $C_{max,opt}$ and exponentially decreasing rewards with solution quality moving away from the optimum. In this way, increasingly difficult to find small improvements of the schedule in the near-optimal range are additionally rewarded. However, the main limitation for the given reward is the need to know optimal or near-to-optimal solutions for all training instances. In our work, we use the training instances solved with the CP-SAT solver provided as part of the training and evaluation framework. Finding optimal solutions for a wide range of training tasks might require extensive computational resources. Another option to quickly estimate a reference makespan value is using a handcrafted empirical logic for makespan estimation based on the sum of all operation times per job, as proposed by Pol et al. (2021).

$$r(T) = 1000 \times 1.025^{C_{max,opt} - C_{max}} \tag{3}$$

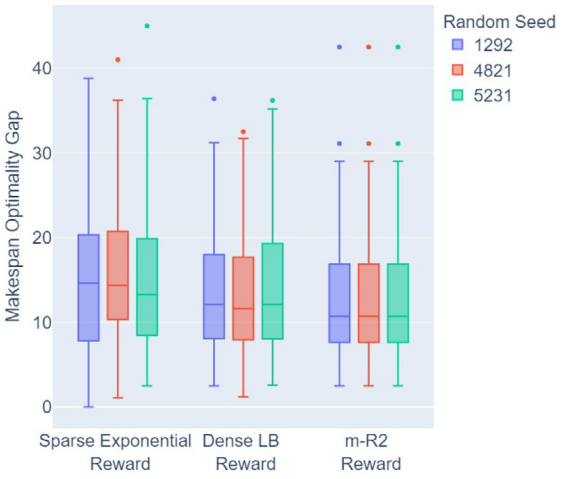
Additionally, we propose a new *multi-instance ranked reward* (m-R2) originating from the ranked reward (R2) approach by Laterre et al. (2018). Conventional ranked reward uses a buffer to track the recent performance of the agent during training. To achieve a positive reward, the agent needs to demonstrate reward levels matching or exceeding a certain threshold r_α calculated as a percentile $\alpha \in (0, 100)$ of the past episode rewards stored in the buffer. We adopt negative makespan C_{max} as a proxy to the agent performance $r_{C_{max}}$. Since optimal makespans vary considerably across JSSP instances, we track performance and calculate reward threshold $r_{\alpha,jssp}$ separately for each JSSP instance $jssp$ used in training. To prevent the agent from repeatedly choosing one potentially sub-optimal solution, we use an exploration incentive coefficient p_e that induces a penalty for solutions at the level of reward threshold with a probability in the range $[0, 1]$. Subsequently, observed agent performance $r_{C_{max}}$ is transformed into a ranked reward r_{m-R2} as demonstrated in Eq. (4).

$$r_{m-R2}(C_{max}, jssp) = \begin{cases} 1, & r_{C_{max}} > r_{\alpha,jssp} \\ -1, & r_{C_{max}} < r_{\alpha,jssp} \\ X, & r_{C_{max}} = r_{\alpha,jssp} \\ & P(X = -1) = p_e \\ & P(X = 1) = 1 - p_e \end{cases} \tag{4}$$

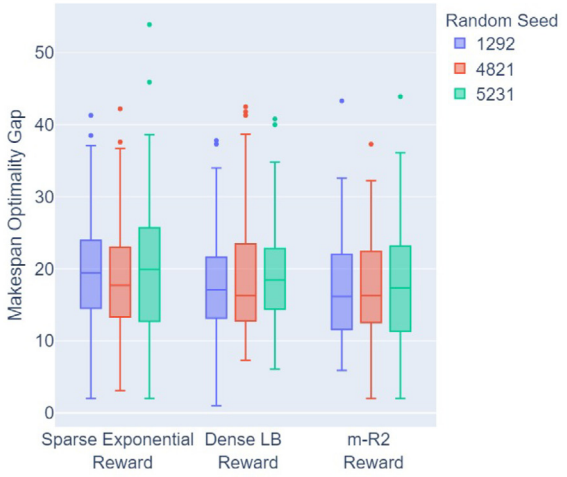
Fig. 6 provides a comparative evaluation of chosen reward shapes with baseline RL setup described in Section 6. All agents are trained for 3,000,000 training steps using 900 JSSP instances of sizes $6 \times 6 \times 6$, $8 \times 8 \times 8$ and $10 \times 10 \times 10$. Each training is repeated three times with different random seeds. After every 75,000 training steps the current performance of the agent is evaluated on a part of the training data. An RL agent with the smallest average optimality gap is selected from each run for further comparisons. The final comparison is conducted on a validation set containing 100 unseen JSSP instances. All experiments with m-R2 reward shape deploy a reward buffer size per scheduling instance, reward threshold $r_{\alpha,jssp}$ and exploration incentive coefficient p_e set to 250, 70% and 0.1 respectively. Only a limited amount of parameter tuning is conducted since the initially chosen parameter set demonstrated good performance over all considered problem sizes. In general, all three reward designs can guide learning towards a good scheduling policy. Yet, on average m-R2 reward results in a 1.3%,



(a) Optimality gaps for 100 unseen 6x6x6 JSSP instances.



(b) Optimality gaps for 100 unseen 8x8x8 JSSP instances.



(c) Optimality gaps for 100 unseen 10x10x10 JSSP instances.

Fig. 6. Comparison of reward shapes: m-R2 reward leads to solutions with lower optimality gaps on average over all considered JSSP sizes. The curriculum learning effect of m-R2 reward stabilizes learning and significantly reduces the observed differences in performance from run to run.

1.4%, and 0.6% lower optimality gaps for $6 \times 6 \times 6$, $8 \times 8 \times 8$, and $10 \times 10 \times 10$ JSSP sizes correspondingly compared to the next best dense LB reward. The Wilcoxon signed-rank test demonstrates that apart from one experimental run featuring the random seed of 5231 and sparse exponential reward, the observed m-R2 reward superiority is statistically significant (p-values below 0.05) for all runs with the problem size of $6 \times 6 \times 6$. At the same time, the observed superiority cannot be statistically confirmed or rejected for bigger problem sizes since the observed p-values exceed the 0.05 threshold. Better performance for smaller problem sizes demonstrated by m-R2 reward design can be explained by the challenges typical for all sparse reward designs related to the credit assignment problem. Rapidly increasing planning horizons with growing problem sizes significantly reduce the reward frequency and make distinguishing between good and bad steps difficult. Nevertheless, the m-R2 reward is a favorable reward design for solutions invariant to the problem size. In this case, a suitable policy can be learned on relatively small problem instances and directly transferred to larger instances without retraining, as demonstrated in this work. Additionally, the remarkably low difference in performance between independent runs with m-R2 reward across different random seeds sets it aside from other reward shape designs. This can be explained by the positive effect of a gradual difficulty increase along with the improving agent's performance similar to the idea of curriculum learning.

7.3. Solution designs on perception and cognition levels

Perception and cognition mechanisms are often integrated into a single neural architecture making it hard to draw a clear line of separation between them. In this study we consider the perception level to cover the information given to an agent to plan its actions, as well as any specific state-space embeddings aimed to enhance the representation of the related state information. Cognition scope is defined as the capability of an agent to evaluate the raw or preprocessed state information and derive the optimal course of action. We isolate those topics according to the given definitions as shown in Fig. 3 and elaborate on each of them in the following subsections.

7.3.1. State-space representation and related embeddings

The majority of current RL solutions in MC demonstrate common solution patterns for the state-space representation. One or multiple RL agents for scheduling are provided with information on the machine's current statuses and related backlogs along with the information on orders within the production system such as location, type of order, or the next pending operation. Additionally, the state-space representation is extended with reward-related performance metrics (e.g. current makespan or machine utilization rates). Primarily, a single consolidated

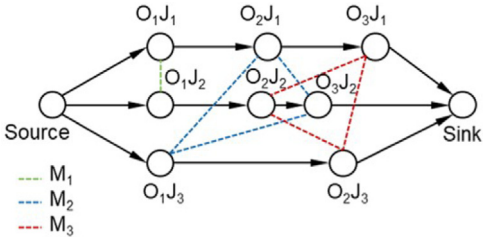


Fig. 7. Representation of a JSSP as a disjunctive graph.

vector covering all parts of the state-space representation is directly given to one or multiple RL agents, which have policy and/or value functions parameterized with feed-forward neural networks (NN) (Altenmüller et al., 2020; Cals et al., 2020; Kim et al., 2020; Kuhnle et al., 2020; Park et al., 2020; Rinciog et al., 2020; Shi et al., 2020; Waschneck et al., 2018).

In general, a job shop scheduling state can be formalized as a disjunctive graph $G = \{V, C \cup D\}$ (see Fig. 7), where $V = \{O_1, \dots, O_n\}$ is a vertex set denoting manufacturing operations for jobs $J = \{J_1, \dots, J_n\}$, source and sink are represented by dummy operations O_0 and O_{n+1} with zero durations, C is a set of directed arcs representing precedence constrains between operations of one job, and D is a set of undirected arcs connecting operations requiring the same machine from set $M = \{M_1, \dots, M_m\}$ (Błazewicz et al., 1994). Relevant information in the state-space representation can be divided into sets of unordered and ordered data. Embedding of those information parts might require the integration of multiple representation techniques.

A common challenge for both ordered and unordered parts of the state-space representation is their dependency on the problem size. This leads to the formulation of the first important condition as a requirement for a problem size agnostic embedding of the state-space representation:

Condition 1. *Input sets of variable sizes can be mapped to output of constant size: there exists a function $f : X \mapsto Y$ s.t. domain $X \subseteq \{\mathbb{R}^i\}$, domain $Y \subseteq \mathbb{R}^\ell$ and $\ell, i \in \mathbb{Z}$.*

Frequently the MARL approach is chosen for MC tasks with one agent controlling one machine (Altenmüller et al., 2020; Kuhnle et al., 2020; Liu et al., 2020; Park et al., 2020; Waschneck et al., 2018) to address the input size invariance Condition 1. This allows for an abstraction of the state-space representation independent from the number of machines, while introducing new challenges, such as non-stationarity and learning instabilities induced by the presence of other agents. The problem of changing state information size persists in the MARL setup because of the varying number of orders in the production system. To deal with the inputs of different sizes an upper bound for the input size can be chosen. Considering that production sites can accommodate a limited number of machines and orders, this design solution might prove to be a simple, yet viable solution. Any smaller state-space representations can be padded to fit the defined required size. In this study we refer to this technique as *state-space padding*.

7.3.2. Unordered parts of the state-space representation

Parts of the state-space representation that represent machine states and states of production orders are inherently unordered sets. The sequencing decisions of an RL agent should not be influenced by arbitrary permutations of orders and machines related to the state-space representation. The embedding condition for unordered parts of the state-space representation can be formulated as follows:

Condition 2. *The output is agnostic of any particular order of the input set: there exists a function $f : X \mapsto Y$ for some domain $X \subseteq \mathbb{R}^k$ and domain $Y \subseteq \mathbb{R}^\ell$ s.t. $f(x) = f(\pi(x))$ for any permutation π of $x \in X$ and $k, \ell \in \mathbb{Z}$.*

Fig. 8. Permutation of unordered parts of the state space slows down the learning on simple tasks, but leads to more robust learning process in general.

One approach to achieve a certain level of agnostic perception of the input sequence with conventional feed-forward neural networks is the constant permutation of unordered information. Exposing the agent to various permutations of unordered parts of the state space reduces the learning of unwanted sequence-related information and results in a certain level of permutation invariance. However, for one state-space representation of dimension n there are $n!$ equivalent permutations. This slows down the policy convergence speed with increasing problem size. At the same time, more complex learning tasks involving hundreds of JSSP instances benefit from random permutations of unordered parts of the state representation. Such permutations help to reduce the dependency of learned scheduling policies on a particular sequence of machines and orders in the state-space representation. In this study we refer to this technique as *state-space permutation*. This facilitates recognizing identical or similar states and results in higher generalization abilities (see Fig. 8).

Alternatively, enforcing certain ordering of the space state can help to achieve effects similar to permutation invariance. In this way, orders can be sorted according to the duration of the next operations or total processing times. Nevertheless, this approach will require an expert-based definition of the similarity measure and corresponding sorting strategy for each unordered set in the state space. For example, choosing a generic similarity measure and corresponding sorting strategy for parts of the state carrying machine information may prove to be challenging.

An input size padding combined with permutation or sorting of the state space can satisfy both Conditions 1 and 2. Several deep learning architectures such as Deep Sets (Zaheer et al., 2017), Janossy Pooling (Murphy et al., 2019) or Input-Invariant Transformers (Lee et al., 2019) inherit both sequence and input size agnostic properties. Since the adoption of such neural network architectures could not be covered within the scope of this work, we plan to integrate them into our framework in the near future and evaluate their broader implications in a later phase of our studies.

7.3.3. Parts of the state space with underlying structure

Information, such as operation sequence per order or production line configuration has a certain underlying structure in non-euclidean space. Several recent works look into exploiting this information while tackling sequencing tasks in MC. Liu et al. (2020), inspired by the success of deep learning on images, propose to embed current production state information with the help of 2D convolution operations on three matrices (similar to three color channels in images) containing information of assigned jobs to machines, processing times, and

completed orders. However, unlike images, information consolidated in handcrafted tabular form bear no relevant spatial information to learn. Despite the demonstrated success of the whole RL setup, the contribution of the 2D convolutional transformation is not investigated separately and may have a limited advantage. Hu et al. (2020) take another approach to encode the current state of a flexible manufacturing system (FMS). They present the production system as a Petri Net, which is encoded with a modified GCN. This allows for a compact encoding of the state space that covers the underlying dynamics of the system and speeds up the overall training process.

It is not only the current state of the production system can be represented as a Directed Acyclic Graph (DAG). As demonstrated in Fig. 7, information such as required machines and processing precedence constraints can be encoded, while preserving the topological information of its graph representation. Zhang et al. (2020) demonstrates solving JSSP with RL while relying only on the partial information from a disjunctive graph representation $G_{partial} = \{V, C\}$. We integrate the proposed approach in the experimental setup, while keeping the training procedure and hyperparameters unchanged. The setup relies on a direct action-space mapping and dense LB makespan reward. Sets of JSSPs for validation and the distribution of training data are consistent with the approach described in Section 6. The only major difference from the default training process established earlier in our study is the use of a new JSSP instance for every training episode instead of picking one JSSP instance from a limited pool of training data.

Fig. 9 summarizes evaluation results for JSSP instances of sizes $6 \times 6 \times 6$, $8 \times 8 \times 8$, and $10 \times 10 \times 10$. Each training and evaluation run is repeated three times with different random seeds. The considered approach is an End-To-End RL solution involving state embeddings with a graph neural network. The setup requires considerably more training parameters and leads to higher deviations of scheduling quality between independent runs. Achieved optimality gaps are close to the reported results in the original paper (Zhang et al., 2020). For the given problem sizes, we observe a 1 to 4% bigger optimality gap compared to our baseline solution and a 3 to 8% bigger optimality gap compared to the enhanced baseline setup with the m-R2 reward (see Fig. 10). Both baseline solutions have smaller neural architectures and a compact handcrafted vector space representation. The main assumption behind the observed result is the limited information available for the agent during planning. To keep the representation of the disjunctive graph sparse, the unordered parts of the disjunctive graph responsible for marking operations that require the same machines (see dashed lines in Fig. 7) are discarded. This makes it challenging for the agent to recognize and proactively avoid potential bottlenecks among unscheduled operations.

Performance of the considered End-To-End RL solution degrades considerably if the original dense reward signal is replaced with a sparse reward. This renders the proposed m-R2 reward as not applicable. The indirect action-space mapping chosen in this work for the baseline solution cannot be transferred directly to the given End-To-End RL solution. The original action space is similar to a multi-agent setup with policy parameter sharing. Separate scheduling decisions are derived from a set of independent value predictions. Each value prediction is based solely on an individual state-space representation covering the global state of the system together with the local state of the operation under consideration. The given approach can work with an arbitrary number of available operations. Action-space designs considered in this work are designed for single-agent solutions with one unified state-space representation and a fixed number of actions.

7.3.4. RL algorithms

Cognition level in this context is considered to cover the capability of an agent to evaluate the current state and derive an optimal course of action. Suitable methods to solve certain combinatorial optimization problems depend strongly on the problem formalization. Order sequencing or order release tasks can be seen as sequential decision-making problems and are often formulated as an MDP. In this case,

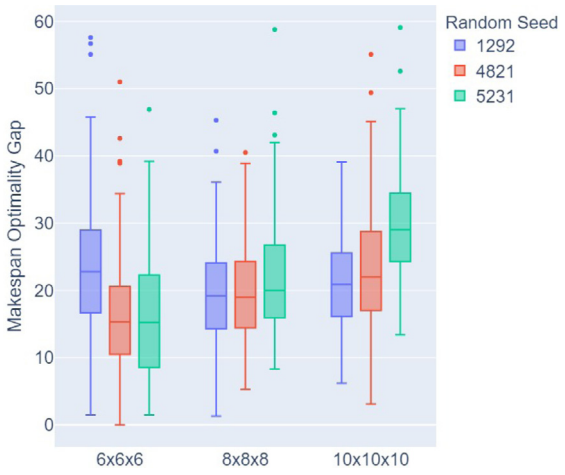


Fig. 9. Evaluation of Zhang et al. (2020) approach on 100 unseen $6 \times 6 \times 6$, $8 \times 8 \times 8$ and $10 \times 10 \times 10$ JSSP instances.

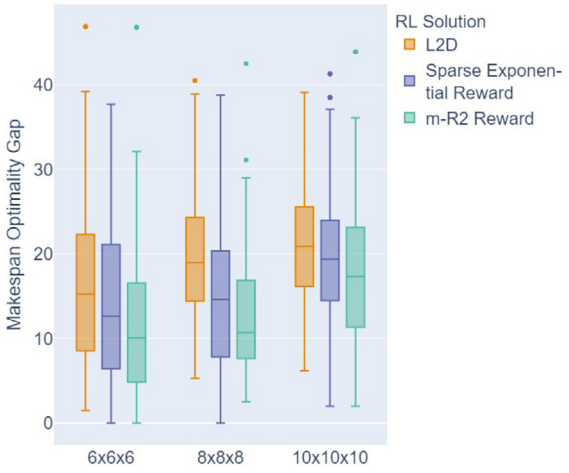


Fig. 10. Comparison of best runs over various problem sizes and approaches: Zhang et al. (2020) (L2D), baseline with sparse exponential reward and enhanced baseline with m-R2 reward. Evaluation on 100 unseen $6 \times 6 \times 6$, $8 \times 8 \times 8$ and $10 \times 10 \times 10$ JSSP instances.

the considered production system has a set of possible states S and actions A. Transition from current state $s \in S$ to the next state $s' \in S$ happens through execution of an action $a \in A$ under certain transition probability $P_a(s, s')$ resulting in a reward $R_a(s, s')$. Such a problem formalization allows for the direct use of a wide variety of RL methods to find a decision policy that maximizes the cumulative sum of rewards over a certain time-horizon.

This formalization approach justifies the common use of value-based RL methods, such as DQN with no problem-specific modifications in a single- or multi-agent setup while solving optimization problems in the area of MC (Altenmüller et al., 2020; Park et al., 2020; Shi et al., 2020; Waschneck et al., 2018). Several investigations are dedicated to the use of policy-based methods such as DDPG (Liu et al., 2020), PPO (Rummukainen and Nurminen, 2019), and TRPO (Kuhnle et al., 2020).

Since a combination of order release and sequence planning activities defines the actual production schedule, several mathematical problem formalizations for production scheduling can be used. For example, scheduling for a single-stage production on a single machine can be seen as a TSP with a salesman (machine) visiting (processing)cities (orders or operations) in a certain sequence (Lawler, 1985). A single-stage production scheduling for multiple machines can be formalized

as a VRP (Beck et al., 2003). This allowed to successfully transfer an RL-based solution for VRP problems from the NCO area to the MC context in a single-stage production with a sequence-dependent setup cost, as demonstrated by Gannouni et al. (2020). Depending on the planning scenario, a number of important constraints can be added to the VRP or TSP problem formulation, helping to model different constraints in a manufacturing environment. Capacitated constraints can limit the working time or the total number of processed orders per machine (Beck et al., 2002). Time window constraints can be used to model scheduling constraints related to due dates or precedence constraints between operations (Picard and Queyranne, 1978; Beck et al., 2002). Mentioned approaches for the formalization of sequencing and order release tasks as variations of TSP or VRP directly allows the use of numerous RL methods from the field of NCO, such as *structure2vec* Q-Learning (Khalil et al., 2017), attention-based networks (Kool et al., 2019) or hierarchical RL with pointer networks (Ma et al., 2020). Along with the discussed techniques for state-space representation, it is also a relevant investigation subject of which improvements can be achieved in the field of MC by adopting customized RL methods from the field of NCO. However, the adaptation of mentioned techniques for MC goes beyond the scope of this study and should be addressed in a set of separate studies.

7.3.5. Extended baseline RL solution and generalization to bigger problem sizes

Apart from the testing performance of a trained RL agent on new scheduling tasks similar to the tasks used in training, it is valuable to see how well a learned scheduling policy scales to bigger problem sizes without additional training. We evaluate the extended baseline RL solution relying on relative action-space mapping, enhanced with the proposed m-R2 reward, state-space padding and state-space permutation techniques. Training is conducted solely on the scheduling instances of the size $6 \times 6 \times 6$. Testing involves two sets with 100 problem instances each of sizes $6 \times 6 \times 6$ and $15 \times 15 \times 15$. Additionally we compare the observed performance of RL agents trained with three different random seeds against two common priority rule heuristics: shortest-processing-time (SPT) and longest-processing-time (LPT). The SPT rule schedules operations with the shortest processing time first. LPT, on the contrary, picks the longest operation time first. Fig. 11 demonstrates superior scheduling performance compared to both SPT and LPT heuristics for both the seen during the training problem size of $6 \times 6 \times 6$, as well as the unseen problem size of $15 \times 15 \times 15$. The remarkably steady performance of the RL agent independent of the random component of initialization and exploration is attributed to the curriculum learning effect of the m-R2 reward as shown in Section 7.2.2.

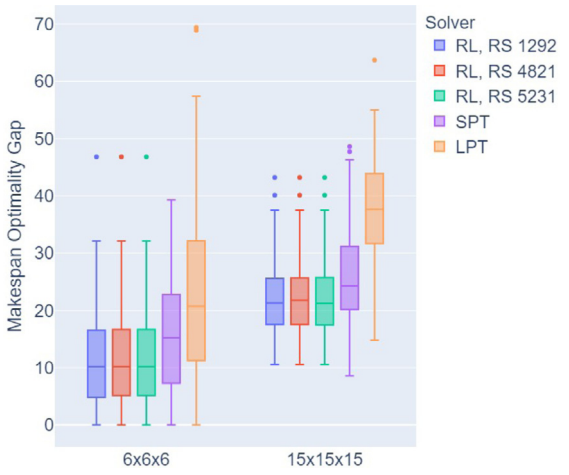


Fig. 11. Extended RL baseline solution trained on $6 \times 6 \times 6$ JSSP instances: performance on 100 unseen $6 \times 6 \times 6$ and $15 \times 15 \times 15$ JSSP instances.

8. Conclusion and outlook

Our work investigates and contributes to several research directions aimed at the application of RL in MC. We survey current RL-based approaches for scheduling in production settings and compare them to the current state of the art in the NCO field. Our work highlights the need for RL solutions in MC to be able to address different problem sizes without retraining, as well as to utilize the properties of the input data with underlying structure and unordered data (see Fig. 3).

We analyze common solution patterns on three different levels: perception level involving state-space representations and related embeddings, interaction level covering action and reward space designs, as well as cognition level covering different RL algorithms.

On the perception level we look into the nature of JSSP instance representation, related state-space designs and embedding techniques. With the example of a simple DQN agent, we demonstrate the advantages of giving the agent a certain level of sequence invariance to unordered parts of the state-space representation via using the state-space permutation technique. Combining it with the state-space padding allows to extend the range of problem sizes the agent can solve without retraining and to generalize to large problem sizes. Analysis of the solution patterns on the interaction level demonstrates the advantage of problem size agnostic action-space design proposed by Samsonov et al. (2021) over the commonly used direct mapping of the action space to separate operations.

We propose a new multi-instance ranked reward (m-R2) design. It automatically forms a suitable learning curriculum tailored to the difficulty of the involved scheduling tasks and the current agent's performance. A direct comparison with several commonly used reward designs demonstrates consistently better results. As a result, adding the m-R2 reward allows to create the strongest baseline solution considered in this work serving as a benchmark for future research. Provided in this work explanation, source code, and demonstrated results are sufficient to understand the idea and the principle of work m-R2 reward. Nevertheless, in the future, we intend to provide a more in-depth analysis of the m-R2 reward and work on its extension to tackle multi-objective optimization tasks.

Combining the state-space permutation, state-space padding, problem size agnostic action-space design and m-R2 reward results in a strong baseline scheduling solution. All solutions considered in our study are integrated into the benchmarking framework. Our benchmarking framework is implemented with the best software development practices and design patterns in mind, features a flexible modular structure with multiple interchangeable building blocks, and provides standardized training and evaluation routines. All of this ensures the low engineering effort required for the integration of new RL-solutions and production simulations. As a result, we contribute towards building a strong set of baselines with a unified API and extend the number of well-designed and publicly-available implementations. Currently it incorporates two RL-based scheduling baselines: the baseline solution developed in this study and the RL approach proposed by Zhang et al. (2020). The main advantage of the proposed benchmarking framework is the possibility to extend it to new RL-solutions and production simulations with minimal engineering effort. Additionally, we provide a SimPy-based simulation for job shop scheduling. The given simulation follows the modular pattern as well and utilizes multiple action, state space and reward wrappers. This allows us to execute all experiments listed in this study with no changes to the core of the simulation. The provided standardized implementations with high flexibility can be a good starting point for other researchers for implementing, testing, comparing and sharing new ideas.

Along with the extension of the m-R2 reward in the future, we intend to concentrate on enhancing the provided baselines with sequence and input size agnostic neural architectures (Zaheer et al., 2017;

Murphy et al., 2019; Lee et al., 2019) for the embedding of unordered state-space parts, as well as to work on the enhancing of embedding techniques for state-space parts with underlying structure as demonstrated by Hu et al. (2020) and Zhang et al. (2020). Another direction of future research is dedicated to the extension of the provided production simulation environment and benchmarking framework as a whole to more production types, while maintaining the current level of flexibility and performance. Further developments in both mentioned directions will facilitate the gradual development of RL applications of MC into production-ready solutions. Additionally, the explainability aspects and providing performance guarantees for RL-based systems in MC are crucial for increasing industry acceptance.

CRediT authorship contribution statement

Vladimir Samsonov: Conceptualization, Methodology, Software, Investigation, Validation, Visualization, Writing – original draft. **Karim Ben Hicham:** Software, Writing – review & editing. **Tobias Meisen:** Conceptualization, Supervision, Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Altenmüller, T., Stüker, T., Waschneck, B., Kuhnle, A., Lanza, G., 2020. Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints. Prod. Eng. 14, 319–328. http://dx.doi.org/10.1007/s11740-020-00967-8.

Aytug, H., Bhattacharyya, S., Koehler, G.J., Snowdon, J.L., 1994. A review of machine learning in scheduling. IEEE Trans. Eng. Manage. 41 (2), 165–171. http://dx.doi.org/10.1109/17.293383.

Beck, J.C., Prosser, P., Selensky, E., 2002. On the reformulation of vehicle routing problems and scheduling problems. In: Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation. Springer-Verlag, Berlin, Heidelberg, pp. 282–289.

Beck, J.C., Prosser, P., Selensky, E., 2003. Vehicle routing and job shop scheduling: What's the difference? In: Proceedings of the Thirteenth International Conference on International Conference on Automated Planning and Scheduling. In: ICAPS'03, AAAI Press, pp. 267–276.

Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S., 2016. Neural combinatorial optimization with reinforcement learning. arXiv Preprint arXiv:1611.09940. URL: https://arxiv.org/pdf/1611.09940.

Biewald, L., 2020. Experiment tracking with weights and biases. URL: https://www.wandb.com/.

Bishop, C.M., 2006. Pattern Recognition and Machine Learning. In: Computer Science, Springer, New York, NY, URL: http://www.loc.gov/catdir/enhancements/fy0818/2006922522-d.html.

Błazewicz, J., Ecker, K.H., Schmidt, G., Węglarz, J., 1994. Scheduling in Computer and Manufacturing Systems, second revised ed. Springer Berlin Heidelberg, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-642-79034-8.

Brucker, P., 2007. Scheduling Algorithms, fifth ed. Springer, Berlin and Heidelberg.

Cals, B., Zhang, Y., Dijkman, R., van Dorst, C., 2020. Solving the order batching and sequencing problem using deep reinforcement learning. arXiv Preprint arXiv:2006.09507. URL: https://arxiv.org/pdf/2006.09507.

Dai, H., Dai, B., Le Song, 2016. Discriminative embeddings of latent variable models for structured data. In: International Conference on Machine Learning (ICML).

Gannouni, A., Samsonov, V., Behery, M., Meisen, T., Lakemeyer, G., 2020. Neural combinatorial optimization for production scheduling with sequence-dependent setup waste. In: IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 2640–2647. http://dx.doi.org/10.1109/SMC42975.2020.9282869.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Korte, B.H., Johnson, E.L., Hammer, P.L. (Eds.), Discrete Optimization. In: Annals of Discrete Mathematics, vol. 5, North-Holland Pub. Co, Amsterdam and New York and New York, pp. 287–326. http://dx.doi.org/10.1016/S0167-5060(08)70356-X.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., 2018. Stable baselines.

Hoffman, W., Branding, F., Schwarz, C., Losse, A., Grundmann, T., 2020. Casymbda.

Howard, R.A., 1960. Dynamic programming and Markov processes.

Hu, L., Liu, Z., Hu, W., Wang, Y., Tan, J., Wu, F., 2020. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. J. Manuf. Syst. 55, 1–14. http://dx.doi.org/10.1016/j.jmsy.2020.02.004.

Hwang, I., Jang, Y.J., 2020. Q(l) learning-based dynamic route guidance algorithm for overhead hoist transport systems in semiconductor fabs. Int. J. Prod. Res. 58 (4), 1199–1221. http://dx.doi.org/10.1080/00207543.2019.1614692.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Le Song, 2017. Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems (NeurIPS).

Kim, H., Lim, D.-E., Lee, S., 2020. Deep learning-based dynamic scheduling for semiconductor manufacturing with high uncertainty of automated material handling system capability. IEEE Trans. Semicond. Manuf. 33 (1), 13–22. http://dx.doi.org/10.1109/TSM.2020.2965293.

Kool, W., Van Hoof, H., Welling, M., 2019. Attention, learn to solve routing problems! In: International Conference on Learning Representations (ICLR). URL: https://openreview.net/forum?id=ByxBFsRqYm.

Kuhnle, A., Kaiser, J.-P., Theiß, F., Stricker, N., Lanza, G., 2020. Designing an adaptive production control system using reinforcement learning. J. Intell. Manuf. http://dx.doi.org/10.1007/s10845-020-01612-y.

Lang, S., Lanzerath, N., Reggelin, T., Müller, M., Behrendt, F., 2020. Integration of deep reinforcement learning and discrete-event simulation for real-time scheduling of a flexible job shop production. In: Winter Simulation Conference (WSC).

Laterre, A., Fu, Y., Jabri, M.K., Cohen, A.-S., Kas, D., Hajjar, K., Dahl, T.S., Kerkeni, A., Beguir, K., 2018. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. arXiv Preprint arXiv:1807.01672.

Lawler, E.L. (Ed.), 1985. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. In: Wiley-Interscience Series in Discrete Mathematics, Wiley, Chichester, W. Sussex.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W., 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In: International Conference on Machine Learning (ICML). pp. 3744–3753.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv Preprint arXiv:1509.02971. URL: https://arxiv.org/pdf/1509.02971.

Liu, C.-L., Chang, C.-C., Tseng, C.-J., 2020. Actor-critic deep reinforcement learning for solving job shop scheduling problems. IEEE Access 8, 71752–71762. http://dx.doi.org/10.1109/ACCESS.2020.2987820.

Lödding, H., 2013. Handbook of Manufacturing Control: Fundamentals, Description, Configuration. Springer, Heidelberg, http://dx.doi.org/10.1007/978-3-642-24458-2.

Luczak, H., Eversheim, W., 1999. Produktionsplanung Und -Steuerung. Springer Berlin Heidelberg, Berlin, Heidelberg, http://dx.doi.org/10.1007/978-3-662-09472-3.

Ma, Q., Ge, S., He, D., Thaker, D., Drori, I., 2020. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. In: AAAI Workshop on Deep Learning on Graphs: Methodologies and Applications. URL: https://arxiv.org/pdf/1911.04936.

Mahadevan, S., Theocharous, G., 1998. Optimizing production manufacturing using reinforcement learning. In: Eleventh International Florida Artificial Intelligence Research Society Conference.

Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E., 2020. Reinforcement learning for combinatorial optimization: A survey. arXiv Preprint arXiv:2003.03600. URL: https://arxiv.org/pdf/2003.03600.

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, Š., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A., 2017. SymPy: symbolic computing in python. PeerJ Comput. Sci. 3, e103. http://dx.doi.org/10.7717/peerj-cs.103.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533. http://dx.doi.org/10.1038/nature14236.

Müller, K., Vignaux, T., Grayson, P., Scherfke, S., Lünsdorf, O., 2020. Simpy. URL: https://simpy.readthedocs.io.

Murphy, R.L., Srinivasan, B., Rao, V., Ribeiro, B., 2019. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In: International Conference on Learning Representations (ICLR).

Nazari, M., Oroojlooy, A., Snyder, L.V., Takáč, M., 2018. Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems (NeurIPS). URL: https://arxiv.org/pdf/1802.04240.

Park, I.-B., Huh, J., Kim, J., Park, J., 2020. A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities. IEEE Trans. Autom. Sci. Eng. 1–12. http://dx.doi.org/10.1109/TASE.2019.2956762.

Perron, L., Furnon, V., 2020. OR-Tools. URL: https://developers.google.com/optimization/.

Picard, J.-C., Queyranne, M., 1978. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. Oper. Res. 26 (1), 86–110. http://dx.doi.org/10.1287/opre.26.1.86.

Pierrot, T., Ligner, G., Reed, S., Sigaud, O., Perrin, N., Laterre, A., Kas, D., Beguir, K., Freitas, N.d., 2019. Learning compositional neural programs with recursive tree search and planning. In: Advances in Neural Information Processing Systems (NeurIPS).

Pol, S., Baer, S., Turner, D., Samsonov, V., Meisen, T., 2021. Global reward design for cooperative agents to achieve flexible production control under real-time constraints. In: Proceedings of the 23rd International Conference on Enterprise Information Systems. SCITEPRESS - Science and Technology Publications, pp. 515–526. http://dx.doi.org/10.5220/0010455805150526.

Rajkumar, M., Asokan, P., Page, T., Arunachalam, S., 2010. A GRASP algorithm for the integration of process planning and scheduling in a flexible job-shop. Int. J. Manuf. Res. 5 (2), 230. http://dx.doi.org/10.1504/IJMR.2010.031633.

Reed, S., Freitas, N.d., 2016. Neural programmer-interpreters. arXiv Preprint arXiv:1611.09940. URL: https://arxiv.org/pdf/1511.06279.

Rey, D., Neuhäuser, M., 2011. Wilcoxon-signed-rank test. In: Lovric, M. (Ed.), International Encyclopedia of Statistical Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1658–1659. http://dx.doi.org/10.1007/978-3-642-04898-2_616.

Rinciog, A., Mieth, C., Scheikl, P.M., Meyer, A., 2020. Sheet-metal production scheduling using AlphaGo zero.

Rummukainen, H., Nurminen, J.K., 2019. Practical reinforcement learning -experiences in lot scheduling application. IFAC-PapersOnLine 52 (13), 1415–1420. http://dx.doi.org/10.1016/j.ifacol.2019.11.397.

Samsonov, V., Kemmerling, M., Paegert, M., Lütticke, D., Sauermann, F., Gützlaff, A., Schuh, G., Meisen, T., 2021. Manufacturing control in job shop environments with reinforcement learning. In: International Conference on Agents and Artificial Intelligence (ICAART). pp. 589–597. http://dx.doi.org/10.5220/0010202405890597.

Scholz-Reiter, B., Höhns, H., 2007. Selbststeuerung logistischer prozesse mit agentensystemen. In: Schuh, G. (Ed.), Produktionsplanung Und -Steuerung. In: VDI-Buch, Springer, Dordrecht, pp. 745–780. http://dx.doi.org/10.1007/3-540-33855-1_18.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization. In: International Conference on Learning Representations (ICLR).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv Preprint arXiv:1707.06347v2.

Shi, D., Fan, W., Xiao, Y., Lin, T., Xing, C., 2020. Intelligent scheduling of discrete automated production line via deep reinforcement learning. Int. J. Prod. Res. 58 (11), 3362–3380. http://dx.doi.org/10.1080/00207543.2020.1717008.

Sutton, R.S., Barto, A., 2018. Reinforcement Learning: An Introduction, second ed. In: Adaptive Computation and Machine Learning, The MIT Press, Cambridge, Massachusetts and London, England.

Van der Aalst, W., 2016. Process Mining: Data Science in Action, second ed. Springer, Heidelberg, http://dx.doi.org/10.1007/978-3-662-49851-4.

Van der Ham, R., 2018. salabim: discrete event simulation and animation in python. J. Open Source Softw. 3 (27), 767. http://dx.doi.org/10.21105/joss.00767.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y., 2018. Graph attention networks. In: International Conference on Learning Representations. URL: https://openreview.net/forum?id=rJXMpikCZ.

Vesselinova, N., Steinert, R., Perez-Ramirez, D.F., Boman, M., 2020. Learning combinatorial optimization on graphs: A survey with applications to networking. IEEE Access 1. http://dx.doi.org/10.1109/ACCESS.2020.3004964.

Vinyals, O., Fortunato, M., Jaitly, N., 2015. Pointer networks. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 2692–2700.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmuller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Deep reinforcement learning for semiconductor production scheduling. In: Advanced Semiconductor Manufacturing Conference (ASMC). IEEE, pp. 301–306. http://dx.doi.org/10.1109/ASMC.2018.8373191.

Wiendahl, H.-P., 1997. Fertigungsregelung: Logistische Beherrschung von Fertigungsabläufen auf Basis des Trichtermodells. Hanser, München.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J., 2017. Deep sets. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 3391–3401, URL: http://papers.nips.cc/paper/6931-deep-sets.pdf.

Zhang, W., Dietterich, T.G., 1995. A reinforcement learning approach to job-shop scheduling. In: 14th International Joint Conference on Artificial Intelligence. pp. 1114–1120.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C., 2020. Learning to dispatch for job shop scheduling via deep reinforcement learning. In: Advances in Neural Information Processing Systems (NeurIPS). URL: http://arxiv.org/pdf/2010.12367v1.

Zheng, S., Gupta, C., Serita, S., 2020. Manufacturing dispatching using reinforcement and transfer learning. In: Machine Learning and Knowledge Discovery in Databases. pp. 655–671.