Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

DISPATCHING IN REAL FRONTEND FABS WITH INDUSTRIAL GRADE DISCRETE-EVENT SIMULATIONS BY DEEP REINFORCEMENT LEARNING WITH EVOLUTION STRATEGIES

Patrick Stöckermann
Alessandro Immordino
Thomas Altenmüller

Infineon Technologies AG
Am Campeon 1-15
81726 Neubiberg, GERMANY

Georg Seidel

Infineon Technologies Austria
Siemensstraße 2
9500 Villach, AUSTRIA

Martin Gebser
Pierre Tassel

University of Klagenfurt
Universitätsstraße 65-67
9020 Klagenfurt am Wörthersee, AUSTRIA

Chew Wye Chan
Feifei Zhang

D-SIMLAB Technologies Pte Ltd
8 Jurong Town Hall Road, #23-05
Singapore, 609434, SINGAPORE

ABSTRACT

Scheduling is a fundamental task in each production facility with implications on the overall efficiency of the facility. While classic job-shop scheduling problems become intractable when the number of machines and jobs increase, the problem gets even more complex in the context of semiconductor manufacturing, where flexible production control and stochastic event handling are required. In this paper, we propose a Deep Reinforcement Learning approach for lot dispatching to minimize the Flow Factor (FF) of a digital twin of a real-world, stochastic, large-scale semiconductor manufacturing facility. We present the first application of Reinforcement Learning to an industrial grade semiconductor manufacturing scenario of that size. Our approach leverages self-attention mechanisms to learn an effective dispatching policy for the manufacturing facility and is able to reduce the global FF of the fab.

1 INTRODUCTION

Semiconductors play an indispensable role in contemporary society, being at the heart of almost every electronic device. The recent global shortage of semiconductor chips has highlighted their significance as a vital enabler of digital transformation for the economy and society, as well as their value in foreign affairs. This shortage, brought about by the COVID-19 pandemic, has stressed the reliance on semiconductor supply for the general public and policy makers. Increasing capacity by building new factories is a long-term investment requiring substantial time and asset. Hence, there is a need for methods enhancing the production capacity using currently available facilities (Ramani et al. 2022; Mohammad et al. 2022; Leslie 2022).

The manufacturing of chips takes place in large-scale production facilities called fabs, where wafers containing multiple chips are produced and then separated into individual units before packaging. These wafers are organized into lots, which serve as the basic production unit in the fab. Since the semiconductor manufacturing process is highly complex and stochastic, planning and optimizing the production is difficult. In fact, scheduling tasks are characterized by very high and diverse volumes of products, with hundreds of production steps and some machines that are scarce due to their high cost and low availability. The production steps additionally incorporate re-entrant flow due to frequent re-occurrences of processing sequences. On

top of that, the production process is highly stochastic, with uncertainty linked to operation durations, the availability of machines, and the next steps to be performed. The latter is due to lots randomly revisiting production steps, for example, if a quality control step fails. Lot dispatching is a central problem in complex and flexible job shops for semiconductor manufacturing, where optimized dispatching is essential for an efficient Work In Progress (WIP) flow and better utilization of fab resources.

In this paper, we propose a Deep Reinforcement Learning (DRL) approach to dispatch lots in a digital twin of a real-world, stochastic, large-scale semiconductor manufacturing facility. Our contributions can be summarized as follows:

- We present the **first application of Reinforcement Learning (RL)** to a **large-scale industrial grade scenario** of lot dispatching in semiconductor manufacturing.
- We propose a **heavily parallelized scalable setup** that considers multiple dimensions of stochasticity in the training data.
- We propose a **neural network architecture** based on self-attention which is size-agnostic regarding the length of the queue.
- We **evaluate** the proposed approach and neural network architecture on **several real-world scenarios** and test the **generalization properties** of the method.

2 FUNDAMENTALS AND RELATED WORK

In this section, we introduce the fundamentals of the semiconductor manufacturing process, including current optimization methods as well as previous work on DRL in the context of semiconductor manufacturing.

2.1 Complex Job Shops in Semiconductor Manufacturing

Semiconductor manufacturing consists of two main parts: *(1)* the frontend, where the blank wafers are processed to completed wafers, and *(2)* the backend, where the wafers are cut into separate chips and packaged. The frontend of semiconductor manufacturing can be characterized as a complex job-shop system (Mönch et al. 2013), realizing the flexible production of diverse and continuously changing product mixes and volumes within a single production facility. Notably, the environment is highly stochastic, with processing, transport times, and machine breakdowns being the most important sources of uncertainty. Another critical, impactful stochastic event is the re-entrant flow of wafers through the fab. The re-entrant flow has significant consequences as certain machine groups (especially the lithography machines) are a bottleneck of the fab. Further complexity for production control is due to batching processes (Knopp et al. 2017), where the parallel processing of jobs on the same machine is crucial for effective resource utilization (Mathirajan and Sivakumar 2006).

In the fab, queues of waiting lots are deliberately tolerated to achieve high utilization of the expensive tools. The processing sequence of the waiting lots from the perspective of machines is specifically optimized to increase the efficiency of the fab by achieving a good tradeoff between throughput and cycle time. This task is also called Semiconductor Factory Scheduling Problem (SFSP), which is often solved as a set of subproblems such as tool scheduling (*single machine job-shop*), work center scheduling (*parallel machines job-shop*), or work area scheduling (*flexible job-shop*) (Mönch et al. 2011). The classic Job-Shop Scheduling Problem (JSSP) has been proven to be NP-hard (Applegate and Cook 1991), and as all more advanced variants can express the standard JSSP, they are also NP-hard (Gupta and Sivakumar 2004).

Restricted subproblems are easier to solve and permit the use of mathematical optimizers (Klemmt et al. 2009). However, these approaches are static and have to calculate a new solution in case of changes in the wafer fab, such as unscheduled downtime of tools, delayed lots due to deviations from expected processing times, or changes in the loading. In production, an optimal solution is recomputed every few minutes to react to the highly dynamic environment. This real-time requirement limits the feasible scope of the optimization (sub)problem, as the time to find and optimize a solution can be at most the cycle time

of the recalculation interval. Hence, optimization needs to be limited to manageable subproblems, where combining their local minima may yield suboptimal performance at the overall fab level.

A well-known phenomenon is the emergence of WIP waves, an accumulation of lots that can cause issues over multiple weeks. Such an accumulation of lots carries on from one work center to the next, exceeding some work centers' capacities and under-utilizing others. The high utilization causes a delay in the production process as the cycle time at the fully utilized work center increases over-proportionally and explodes in case of tool failures. Therefore, a smooth distribution of lots over different work centers is desirable. As on-time delivery is often the most important factor in the wafer fab, optimization at the fab level is desirable to prevent delays and achieve optimal utilization of the tools (Waschneck 2020).

Previous approaches regarding the application of RL for dispatching in semiconductor manufacturing include a multi-agent model of four work centers parameterized with historical data (Waschneck et al. 2018), as well as a single-agent framework considering three work centers (Kuhnle et al. 2021). For the conceptually simpler case of JSSP, Park et al. (2021) apply a Graph Neural Network (GNN) scheduler. This design is motivated by the very good generalization capability of GNNs and the natural representation of JSSP as a disjunctive graph. Moreover, Liu et al. (2020) propose an RL approach to solve JSSP utilizing an actor-critic network. Both aforementioned JSSP approaches achieve good results on commonly referenced benchmark problems that are deterministic and contain no more than 20 machines. In contrast to this, Lee et al. (2019) utilize a simulator modeling a real fab with 160 machines and 50 products.

The SMT2020 testbed (Kopp et al. 2020) models modern wafer fabs including over 1000 tools. Based on an academic, open-source SMT2020 simulation tool, Tassel et al. (2023) propose a self-supervised, RL-based approach to improve the global dispatching of a fab. The scale of the addressed problem goes well beyond the small test datasets for JSSP. However, the SMT2020 fab models have some shortcomings compared to real manufacturing data. In fact, our scenario's load mix is much more diverse, and the tool dedications are more complex. While there are up to 10 different products in the SMT2020 scenarios, our dataset contains more than ten times the number of products. Another important detail that is missing in SMT2020 is that a specific operation for the same lot can have flexible processing times on different tools.

2.2 Evolution Strategies

Salimans et al. (2017) explore the use of Evolution Strategies (ES) as an alternative to derivative-based RL algorithms, such as policy-based Proximal Policy Optimization (PPO) (Schulman et al. 2017; Mnih et al. 2016) or value-based Deep Q-Network (DQN) (Mnih et al. 2013) approaches. RL is a subfield of machine learning that has recently gained significant attention in the semiconductor industry, as it enables the development of intelligent systems that can autonomously learn how to make decisions in complex and dynamic scenarios through interaction with the environment.

However, ES offer multiple advantages over derivative-based RL methods. Salimans et al. (2017) show that ES perform well on benchmark problems such as MuJoCo and Atari. Furthermore, this approach scales very well with the number of used CPU cores as different configurations of the parameter set, defining the policy, can be tested in parallel. The second important advantage is the nature of ES, which can be interpreted as a set of black-box algorithms. Each parameter set is evaluated using only one overall reward for each episode, thus mitigating the credit assignment problem (Sutton and Barto 1998) in RL. ES also do not require back-propagation, which significantly simplifies the implementation. As the parameter configurations are generated by adding Gaussian noise to the parameters, ES show good exploration behavior. ES are attractive for long-lasting effects of the actions chosen by the policy and suboptimal value function estimates. However, ES will scale poorly with the number of parameters θ.

The version of ES used by Salimans et al. (2017) belongs to the class of Natural Evolution Strategies (NES), presented by Wierstra et al. (2008) with F as the objective function evaluating a policy π_θ parameterized by θ. The objective function can be defined on θ and directly optimized using stochastic gradient ascent. The score function, estimated by Equation 1 (Salimans et al. 2017), can be approximated by adding Gaussian noise with the fixed covariance $\sigma^2 I$ in the parameter space. Using this blur, we smoothen

the discrete behavior of the environment and actions:

$$\nabla_\theta \mathbb{E}_{\varepsilon\sim N(0,I)} F(\theta+\sigma\varepsilon) = \frac{1}{\sigma}\mathbb{E}_{\varepsilon\sim N(0,I)}\{F(\theta+\sigma\varepsilon)\varepsilon\} \tag{1}$$

As our environment is highly stochastic, we introduce further smoothing by evaluating each parameter set $\theta+\sigma\varepsilon$ on multiple environments with different random seeds and averaging over their return values.

The resulting Algorithm 1 loops over two phases. The first one generates perturbations of the parameters defining the policy and evaluates the resulting policies in the environment. The second phase combines the results of all executed episodes to calculate a stochastic gradient estimate and update the parameters:

Algorithm 1 Evolution Strategies (Salimans et al. 2017)

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0, number n of parallel agents, maximum iterations T
2: **for** $t = 1,\ldots,T$ **do**
3: Sample $\varepsilon_1,\ldots\varepsilon_n \sim N(0,I)$
4: Compute return values $F_i = F(\theta_t+\sigma\varepsilon_i)$ for $i = 1,\ldots,n$
5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\varepsilon_i$
6: **end for**

While this approach explores the parameter space with random perturbations of the parameters, the Covariance Matrix Adaptation (CMA) approach (Hansen and Ostermeier 2001) samples while considering the covariance of the parameters and aiming for faster convergence. The key feature of CMA-ES is the adaptation of the covariance matrix, which captures the relationships between the variables in the optimization problem. The covariance matrix is updated in each algorithm generation based on the current population's success or failure. This adaptive feature of CMA-ES enables it to explore the search space efficiently and converge to the global optimum.

CMA-ES fixes the problem by tracking pairwise dependencies between the samples in the distribution with a covariance matrix C. The new distribution parameter becomes

$$\theta = (\mu,\sigma,C), p_\theta \sim N(\mu,\sigma^2 C) \sim \mu + \sigma N(0,C)$$

where σ controls the overall scale of the distribution, often known as step size (Hansen and Ostermeier 2001). At each iteration, the CMA-ES algorithm updates the mean vector μ_i and the covariance matrix C to generate the population of candidate solutions w_i. The mean vector is updated using the average of the best candidate solutions of the previous generation (Hansen and Ostermeier 2001):

$$\mu_{t+1} = \mu_t + \alpha_\mu \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (w_i^{t+1} - \mu_t)$$

N_{best} is a hyperparameter denoting the subset of the most promising candidates w_i, where the importance is provided by the score $F(w_i)$ given by the fitness function and α_μ, usually set to 1, controls how fast the mean μ should be updated. The CMA-ES method is summarized in Algorithm 2:

Algorithm 2 Summarized method for CMA-ES presented by (Hansen and Ostermeier 2001)

Input : Step size σ_0, covariance matrix $C_0 = I$, mean vector μ_0, number N_{best} of most promising solutions considered for optimization, number n of parallel agents, maximum iterations T
for $t = 1, \ldots, T$ **do**
  Sample $w_1, \ldots w_n \sim N(\mu_t, \sigma^2 C_t)$
  Compute return values $F_i = F(w_i)$ for $i = 1, \ldots, n$
  $\sigma_{t+1} \leftarrow \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (w_i - \mu_t)$
  $\mu_{t+1} \leftarrow \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} w_i$
end for

3 APPROACH

In this section, we describe the RL environment to interact with the fab's digital twin alongside the observation and action space and the evaluation function. Moreover, we detail our attention-based neural network architecture, the training procedure, and the setting used to train the policy.

3.1 Reinforcement Learning Architecture

The architecture that is used for the training of the policy consists of three main components: *(1)* the simulator, *(2)* the gym environment, and *(3)* the optimizer. In our experiments, the optimizer creates multiple parallel gym environments, where each of them communicates with its own simulator instance. Since our simulation is very elaborate and takes more than 99 % of the execution time of the training, the parallelization of the simulation environments is essential. The simulator models the wafer fab, and the gym environment defines how the interaction with the simulator is handled. This includes extracting the features for the observation and the execution of the actions chosen by the policy. The optimizer tunes the parameters of the policy to maximize the cumulative reward over an episode.

3.1.1 Discrete-Event Simulation

We utilize a highly realistic, state-of-the-art commercial simulator with over a thousand pieces of equipment for our experiments (Seidel et al. 2017; Seidel et al. 2020). The simulator is parameterized using historical data from the underlying real fab to extract statistical properties such as equipment failure distributions. Moreover, the loading in the simulator is the same as the planning data for the real fab. The simulation model is integrated into a production pipeline and parameterized with a scalable, automated data farming approach, and it is calibrated and validated to be about 90 % accurate regarding fab level Key Performance Indicators (KPIs) as well as work center KPIs like utilization (Seidel et al. 2017). In the real fab, mathematical optimization is used for specific, local optimization subproblems. This cannot be simulated because the computation of such local optima would slow down the simulator to almost real time. As the wafer fab level results were proven to be highly accurate, we trust the calibrated parameters and heuristics used in the simulation to provide a sufficient reference for comparing our approach to reality.

As the scope of the simulation is extensive, detecting bottlenecks and critical elements in the WIP flow through data analysis is crucial. The optimization can be applied more effectively and targeted to these bottlenecks, in our case, the lithography clusters. This means that we only control the dispatching decisions for lithography systems. The other tools run with their default dispatching rules. We selected the lithography clusters because they have high utilization and high throughput. The impact on the fab level is therefore expected to be high.

To utilize the existing discrete-event simulation, an interface is used, which handles the communication between the optimization algorithm in Python and D-SIMLAB's D-SIMCON simulator (D-SIMLAB Technologies 2023). Every time a dispatching decision has to be made at the initially defined bottleneck

tools, the simulator triggers a callback to the agent to decide on the next lot to dispatch according to the current policy.

The optimization performance is measured by comparing the resulting KPIs to the ones achieved by applying the default dispatching rules in the simulation. Those are a combination of conditional, static, handcrafted dispatching rules such as Earliest Due Date (i.e., selecting the lot with the closest deadline), First-In First-Out (i.e., selecting the lot that waits longest since its previous operation), and several rules for prioritized lots. This combination of multi-level dispatching heuristics is carefully designed by domain experts, where especially the batching tools rely on elaborate handcrafted heuristics. We validated with simulation experiments that the combined multi-level dispatching heuristics outperform each of the mentioned individual rules.

3.1.2 State Space

All tools and lots in the fab contribute to the state of the simulation. However, it is infeasible to base the decisions of the RL agent on all theoretically available information, as the number of parameters for the neural network defining the policy would be too high, and the communication overhead would exceed the simulation execution time by far. Therefore, we define a subset of the state to be included in the agent's observation. The observation space represents the portion of the simulator state taken as input for the agent at every decision.

Each agent interaction corresponds to a dispatching decision at one of the lithography cluster tools, as described in subsubsection 3.1.1. The simulator triggers a callback to the agent requesting a dispatching decision every time one of the tools controlled by the agent switches its state from unavailable to available, either because an operation has been processed or a repair or maintenance has been completed on the tool.

The agent receives only the representations of the lots queuing in front of the current tool requesting a dispatching decision. That is, the entirety of all lots queuing at the work center that the tool belongs to is filtered for the lots that can be processed by the specific tool. The following attributes are used as a representation of the individual lots in the queue of the current tool:

- Time to fab due date
- Time to step due date
- Waiting time at current step
- Current lithography layer
- Number of alternative tools
- Boolean flag indicating whether a faster tool exists
- Expected processing time
- Expected remaining cycle time

The fab due date refers to the planned completion of the lot that is promised to the customer. The step due date is determined by calculating the expected cycle times for each step and serves as a target date for the completion of the current step to meet the fab due date. Waiting time indicates the duration for which a lot has been pending since its previous operation.

The current lithography layer serves as an indicator for the upcoming process steps and the number of process steps already completed. Additionally, the number of tools that can process the lot at the given step is included. To enhance the utility of alternative tool information, we also indicate whether another tool could speed up the processing time, which may occur when it is able to process more wafers in parallel. The expected processing time is referring to the current step, and the remaining cycle time to the expected duration for completing all yet to perform operations.

The lithography tools included in the simulation can pre-load multiple reticles, allowing the next reticle to be loaded without losing time while the current wafers are still processed. Therefore, there are no major setup times that need to be avoided, and information about the reticles is not included in the observation space. However, it could happen that a reticle cannot be loaded because all reticles of that type are used on other machines. This state is typically very short, as the processing times are only a few minutes and one specific reticle is usually not used more than a handful of times in a row. We quantified such effects by comparing simulation runs and found them to be statistically negligible.

3.1.3 Policy

The policy maps the observation to an action. It consists of a neural network with an attention mechanism and preprocessing steps, where one of the key challenges is the variable queue of waiting lots, so that the policy's input varies in length. Figure 1 displays an example input with four queuing lots in front of a tool. Similar to (Tassel et al. 2023), the proposed architecture, through its attention mechanism, allows to be size and order agnostic.

The observation is fed to the network lot by lot and combined as the dot product of projections. This architecture allows our approach to scale for large instances easily and to efficiently deal with the number of trainable parameters. In particular, the attention mechanism permits to inject the context of the other lot representations without depending on the absolute number of lots.

It requires three projections for the queries, keys, and values. The Scaled Dot-Product Attention (Vaswani et al. 2017) is computed by applying the softmax function on the transposed dot product of the query Q with all keys K, scaled by dividing by the key projection dimension d_k and multiplying it with the value V:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Key and query lot representations are projected to a lower dimensional space to reduce the required parameters. We observe that the projection from eight to six values is a good trade-off between loss of information and the number of trainable parameters as it achieves the best results in our experiments.

The individual input parameters are preprocessed with z-score normalization. We calculate a running mean and standard deviation for each input parameter during the training. Those values are asynchronously exchanged over parallel simulations. This guarantees that all values are within a reasonable range and,

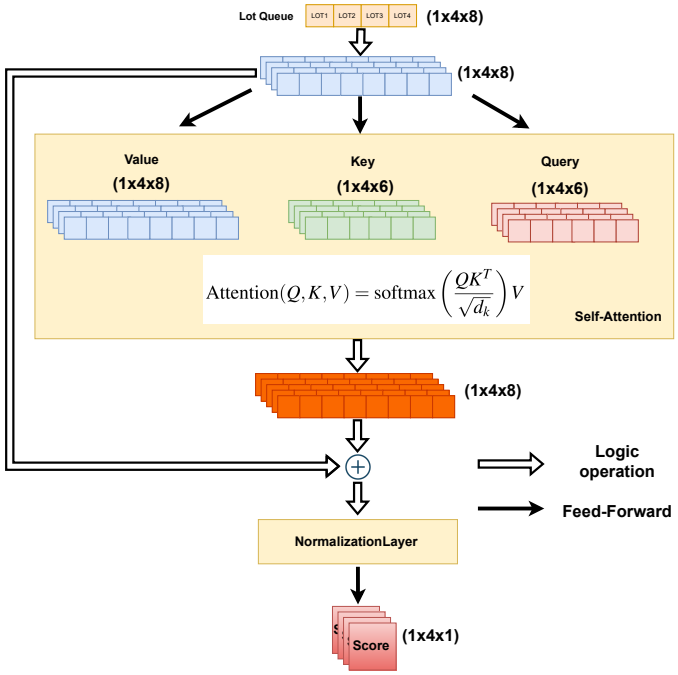

Figure 1: The architecture of the deep neural network includes the self-attention mechanism. Lot representations are initially projected and subsequently fed trough the attention mechanism as well as normalization and feed-forward layers.

therefore, usable for the neural network. We do not use batch normalization because the values would lose the context of the entire environment if they were only normalized among each other in one queue. More specifically, this means that the model should recognize that a waiting time is long compared to other lots in the same queue, and whether it is a long waiting time compared to all queues it encountered before.

The result of the attention mechanism is normalized and then passed to a feed-forward layer to eventually calculate a score for each lot.

3.1.4 Action Space

The action space defines the possible actions that the agent can choose from. In our case, the action is to choose one of the lots from the queue, to be dispatched next on the currently considered tool, which just became available. After the policy scores each lot with consideration of the other lots, we greedily pick the lot with the highest score.

3.1.5 Reward

The reward is based on the global Flow Factor (FF) of the fab, which is defined as the overall cycle time divided by the raw processing time. The difference between these two values is due to the time that the lots wait in queues for processing steps. Ideally the FF would be equal to 1, meaning that the lots are never waiting. However, fabs are always run with a greater FF to achieve a higher throughput and better utilization of the expensive tools. As the loading is fixed, we want the FF to be as low as possible without yielding heavily delayed individual lots.

3.2 Optimization Algorithm

We use ES to optimize the cumulative reward generated by the agent policy and compare both algorithms described in section 2. While policy-gradient-based approaches are commonly used to train RL agents, our domain imposes multiple challenges that make using such approaches difficult.

The first challenge is the definition of a value function, estimating how desirable some state is. This is the basis for the advantage estimator, which is a core concept of PPO. It assesses whether actions are advantageous by considering the cumulative reward and comparing the estimated values of the states before and after an action. The problem here is that the simulation's state is too large to be taken entirely as an input to a value function estimator. Therefore, it would need to be encoded, or individual features must be selected by hand based on deep domain knowledge.

The second main challenge is the credit assignment problem. While the ES approach only considers the overall resulting KPIs of an episode, which is easily accessible information, PPO requires a reward every few thousand decisions to converge. Since the overall effect of decisions is delayed, a meaningful intermediate reward is difficult to define. Any such frequent reward would be have to be linked to an optimal solution for the entire episode, while ES directly optimizes based on the rewards for entire episodes.

3.3 Parallelization

Due to the complexity of the simulation, most of the execution time is spent running and communicating with the simulation. It takes multiple hours to simulate one episode of two months in the fab, including the control of over 50 bottleneck tools. For a good result, dozens of episodes are required. Multiple parallel simulations with different random seeds for each parameter set are executed on a Load Sharing Facility (LSF) to mitigate this problem. The number N of parallel simulations is the product of the number of different random seeds r, and the number of new policy parameter sets p. Therefore, for four different random seeds and 16 different perturbations of the parameters θ, we need 64 cores to execute everything in parallel. The workers are deployed on different hardware instances on the cluster, grouped in nodes of at least 4 cores per physical instance.

4 RESULTS

For the first experiment, we train a dispatching policy using Algorithm 1 presented by Salimans et al. (2017). During the training, the mean FF steadily improves over the episodes, surpassing the static reference dispatching heuristics. To test whether the found policy works for different random seeds, we evaluate the policy with frozen weights on 32 simulations with different random seeds. This influences all stochastic events, such as tools uptime, the lots that must be reworked, split, or scraped, and operations' processing times. In our experiments, we were able to improve the average fab level FF for a specific loading scenario by 2.54 % compared to the reference run described in subsubsection 3.1.1, while the tardiness is reduced by 18.92 %. We observe similar improvement if the method is trained and tested on another simulation scenario with different loading. However, we achieve no improvement over the reference when training on one scenario and testing on another scenario, where each scenario refers to a specific interval of time from the past two years in the modeled wafer fab. In fact, different loading scenarios vary in the number of free tools, the total number of WIP lots, and the product mix.

To mitigate the generalization problem, we train a model on two loading scenarios in parallel (see Figure 2). Each policy is therefore evaluated over different loading scenarios within each episode. When training on two loading scenarios in parallel, we observe that, during the testing, the strategy generalizes for those two scenarios over different random seeds and previously unseen scenarios (see Table 1). We additionally introduce a tardiness penalty as we find that the pure optimization of the FF sometimes leads to increased total tardiness. If the tardiness is higher than for the reference, the reward is the FF discounted by the ratio at which the tardiness is worse relative to the reference. Our results show that this is effective in preventing increased tardiness.

When comparing the ES approach presented by Salimans et al. (2017) with the CMA-ES approach, we observe that the latter leads to faster convergence and better solutions. The approach of Salimans et al. (2017) does not yield any improvement over the reference when training on multiple scenarios, even after 15 episodes, while CMA-ES already outperforms the reference after less than 10 episodes. This can be explained by the fact that CMA-ES is considerably more sample-efficient as it is not searching using random perturbations, but is adaptive in step size and direction.

Training on two sufficiently different scenarios yields a much more stable policy than training on just one with different random seeds. However, some scenarios show considerably better improvement than

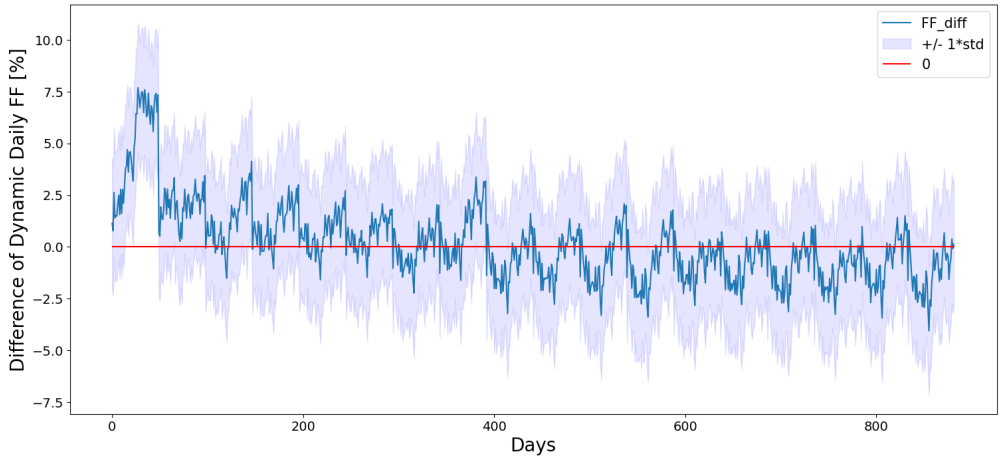

Figure 2: The improvement of the FF during the training. The graph shows the mean difference and standard deviation between the FF of the RL runs compared to their respective reference runs with the same random seed in percent. A negative value means improvement over the reference. After 50 days, the simulation is reset and a new episode is started with the same initial conditions.

Table 1: Evaluation of a policy trained with 64 CPUs cores on loading scenarios 1 and 2 in parallel. The reward is the FF including the mentioned tardiness penalty. The results are compared to the respective reference runs with the corresponding random seeds. The policy is evaluated on the two scenarios it was trained on as well as two previously unseen scenarios 3 and 4. Each scenario is tested with 16 random seeds, while it was trained on two random seeds for each of the two scenarios used for training. We report the percentage of improvement over the reference and its standard deviation. For all metrics, positive values represent an improvement.

Relative Improvement	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
FF	$2.72\% \pm 0.34$	$0.88\% \pm 0.28$	$5.00\% \pm 0.22$	$0.60\% \pm 0.31$
Tardiness	$36.92\% \pm 1.10$	$22.84\% \pm 1.15$	$26.91\% \pm 1.87$	$17.93\% \pm 1.43$
Completed Wafers	$0.35\% \pm 0.11$	$0.79\% \pm 0.09$	$0.66\% \pm 0.05$	$0.13\% \pm 0.16$

others. As the reward is the mean over all loading scenarios and random seeds, the optimizer may achieve good results by improving one scenario much more than the other during training. This seems to be the case for scenario 1 in Table 1. As scenario 1 has the lowest reference FF and scenario 3 the highest, it might be the case that it is easier to beat the heuristics in more extreme cases. Note that even a one-digit percentage improvement means a great success since the dispatching heuristics in the fab are already highly optimized. With more computational resources, more than two scenarios can be used for training, making it even harder to overfit one scenario. To verify the usefulness of the architecture used (see subsubsection 3.1.3), we also compared experiments using a single-layer feed-forward network only. Our architecture was significantly better in each experiment, thus supporting the design displayed in Figure 1.

5 CONCLUSION

The highly relevant problem of optimizing lot dispatching in semiconductor manufacturing is frequently addressed with RL approaches in academia. However, this work is the first to use a simulation fed with recent and real data from a production facility of that size. The difficulties related to the complexity and enormous runtime that arise from our industrial grade scenario are overcome by strong parallelization, a queue-size-agnostic and parameter-efficient neural network architecture as well as the sample-efficient CMA-ES optimization algorithm.

We show that an ES optimizer combined with an RL architecture is able to find optimized dispatching strategies for individual scenarios that generalize over different loading scenarios and randomized system behavior. The found strategies outperform the dispatching heuristics that are currently used in a real frontend fab. When training on two loading scenarios in parallel, the found policy is able to generalize over previously unseen scenarios during the testing. Our experiments show that the approach can optimize global KPIs in highly stochastic and complex simulations by controlling decisions for a subset of its components. The robustness and generalization capabilities of the trained policy could be further improved by training on more scenarios and random seeds in parallel, which is mainly limited by the available computational resources. Furthermore, different neural network architectures for the policy can be explored and compared, which we aim to investigate in future work.

For further assessment, the simulation has to be examined in more detail for its compliance with reality. While the simulation accuracy has been validated regarding its fab level KPIs, some details at tool level can be improved to fit the reality even more closely and to ensure that a strategy trained in simulation will also generalize to the real fab.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their helpful comments. This work was partially funded by FFG project 894072 (SwarmIn) as well as KWF project 28472, cms electronics GmbH, FunderMax

GmbH, Hirsch Armbänder GmbH, incubed IT GmbH, Infineon Technologies Austria AG, Isovolta AG, Kostwein Holding GmbH, and Privatstiftung Kärntner Sparkasse.

REFERENCES

Applegate, D. L., and W. J. Cook. 1991. "A Computational Study of the Job-Shop Scheduling Problem". *INFORMS Journal on Computing* 3(2):149–156.

D-SIMLAB Technologies 2023. "Forecaster". http://www.d-simlab.com/category/d-simcon/products-d-simcon/forecaster-and-scenario-manager.

Gupta, A. K., and A. I. Sivakumar. 2004. "Job Shop Scheduling Techniques in Semiconductor Manufacturing". *The International Journal of Advanced Manufacturing Technology* 27(11-12):1163–1169.

Hansen, N., and A. Ostermeier. 2001. "Completely Derandomized Self-Adaptation in Evolution Strategies". *Evolutionary Computation* 9(2):159–195.

Klemmt, A., S. Horn, G. Weigert, and K.-J. Wolter. 2009. "Simulation-Based Optimization vs. Mathematical Programming: A Hybrid Approach for Optimizing Scheduling Problems". *Robotics and Computer-Integrated Manufacturing* 25(6):917–925.

Knopp, S., S. Dauzère-Pérès, and C. Yugma. 2017. "A Batch-Oblivious Approach for Complex Job-Shop Scheduling Problems". *European Journal of Operational Research* 263(1):50–61.

Kopp, D., M. Hassoun, A. Kalir, and L. Mönch. 2020. "SMT2020—A Semiconductor Manufacturing Testbed". *IEEE Transactions on Semiconductor Manufacturing* 33(4):522–531.

Kuhnle, A., J. Kaiser, F. Theiß, N. Stricker, and G. Lanza. 2021. "Designing an Adaptive Production Control System Using Reinforcement Learning". *Journal of Intelligent Manufacturing* 32(3):855–876.

Lee, W., B. Kim, K. Ko, and H. Shin. 2019. "Simulation Based Multi-Objective Fab Scheduling by Using Reinforcement Learning". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H. G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, 2236–2247. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Leslie, M. 2022. "Pandemic Scrambles the Semiconductor Supply Chain". *Engineering* 9:10–12.

Liu, C., C. Chang, and C. Tseng. 2020. "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems". *IEEE Access* 8:71752–71762.

Mathirajan, M., and A. I. Sivakumar. 2006. "A Literature Review, Classification and Simple Meta-Analysis on Scheduling of Batch Processors in Semiconductor". *The International Journal of Advanced Manufacturing Technology* 29(9-10):990–1001.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. 2016. "Asynchronous Methods for Deep Reinforcement Learning". In *Proceedings of the 2016 International Conference on Machine Learning*, edited by M.-F. Balcan and K. Q. Weinberger, 1928–1937. New York: Proceedings of Machine Learning Research.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. 2013. "Playing Atari with Deep Reinforcement Learning". *CoRR* abs/1312.5602.

Mohammad, W., A. Elomri, and L. Kerbache. 2022. "The Global Semiconductor Chip Shortage: Causes, Implications, and Potential Remedies". *IFAC-PapersOnLine* 55(10):476–483.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. 2011. "A Survey of Problems, Solution Techniques, and Future Challenges in Scheduling Semiconductor Manufacturing Operations". *Journal of Scheduling* 14(6):583–599.

Mönch, L., J. W. Fowler, and S. J. Mason. 2013. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*, Volume 52 of *Operations Research/Computer Science Interfaces Series*. New York: Springer.

Park, J., J. Chun, S. H. Kim, Y. Kim, and J. Park. 2021. "Learning to Schedule Job-Shop Problems: Representation and Policy Learning Using Graph Neural Network and Reinforcement Learning". *International Journal of Production Research* 59(11):3360–3377.

Ramani, V., D. Ghosh, and M. S. Sodhi. 2022. "Understanding Systemic Disruption from the Covid-19-Induced Semiconductor Shortage for the Auto Industry". *Omega* 113:102720.

Salimans, T., J. Ho, X. Chen, and I. Sutskever. 2017. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". *CoRR* abs/1703.03864.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. "Proximal Policy Optimization Algorithms". *CoRR* abs/1707.06347.

Seidel, G., C. F. Lee, A. M. Kam, B. P. Gan, C. W. Chan, A. Naumann, and P. Preuss. 2017. "Harmonizing Operations Management of Key Stakeholders in Wafer Fab Using Discrete Event Simulation". In *Proceedings of the 2017 Winter Simulation Conference*, edited by V. W. K. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. H. Page, 3670–3678. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Seidel, G., C. F. Lee, A. Y. Tang, S. L. Low, B. P. Gan, and W. Scholl. 2020. "Challenges Associated with Realization of Lot Level Fab Out Forecast in a Giga Wafer Fabrication Plant". In *Proceedings of the 2020 Winter Simulation Conference*,

edited by K.-H. G. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, 1777–1788. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Sutton, R. S., and A. G. Barto. 1998. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: Massachusetts Institute of Technology Press.

Tassel, P., B. Kovács, M. Gebser, K. Schekotihin, P. Stöckermann, and G. Seidel. 2023. "Semiconductor Fab Scheduling with Self-Supervised and Reinforcement Learning". In *Proceedings of the 2023 Winter Simulation Conference*, to appear.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. "Attention is All you Need". In *Proceedings of the 2017 Annual Conference on Neural Information Processing Systems*, edited by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, 5998–6008. New York: Curran Associates, Inc.

Waschneck, B. 2020. *Autonome Entscheidungsfindung in der Produktionssteuerung komplexer Werkstattfertigungen*. Ph. D. thesis, University of Stuttgart, Germany.

Waschneck, B., A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek. 2018. "Deep Reinforcement Learning for Semiconductor Production Scheduling". In *Proceedings of the 2018 Annual SEMI Advanced Semiconductor Manufacturing Conference*, 301–306. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Wierstra, D., T. Schaul, J. Peters, and J. Schmidhuber. 2008. "Natural Evolution Strategies". In *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, 3381–3387. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

PATRICK STÖCKERMANN is a Ph.D. candidate at Infineon Technologies in Munich, GERMANY, and the University of Klagenfurt, AUSTRIA. His current research interests focus on utilizing reinforcement learning methods for semiconductor manufacturing. His email address is patrick.stoeckermann@infineon.com.

ALESSANDRO IMMORDINO is a Master student at Infineon Technologies in Munich, GERMANY. His current research interests focus on utilizing reinforcement learning methods for semiconductor manufacturing. His email address is alessandro.immordino@infineon.com.

THOMAS ALTENMÜLLER is vice president for manufacturing analytics at Infineon Technologies in Munich, GERMANY. He started to work in semiconductor manufacturing in 1996 and operated in various functions in technology, planning, strategy and manufacturing analytics. In the last years, he investigated the opportunities of deep reinforcement learning for frontend WIP flow management and maintenance scheduling. His email address is thomas.altenmueller@infineon.com.

GEORG SEIDEL is senior manager at Infineon Technologies in AUSTRIA. Since 2000, he has been actively engaged in various industrial engineering topics, including simulation and WIP flow management. His responsibilities include overseeing the deployment of fab simulation across multiple Infineon sites, specifically in Kulim, Regensburg, and Villach. His email address is georg.seidel@infineon.com.

MARTIN GEBSER is professor for Production Systems at the University of Klagenfurt and Graz University of Technology, AUSTRIA. His research addresses the practical implementation and application of modern solving technology for various complex tasks, such as planning and scheduling, product configuration, and system design. His email address is martin.gebser@aau.at.

PIERRE TASSEL is a Ph.D. candidate at the University of Klagenfurt, AUSTRIA. His current research interests include utilizing reinforcement learning methods for solving combinatorial optimization problems and constraint programming. His email address is pierre.tassel@aau.at.

CHEW WYE CHAN is a software engineer of D-SIMLAB Technologies (Singapore). He holds a Master of Computing degree from the National University of Singapore. He is currently engaged as a doctoral student at the School of Computer Engineering at Nanyang Technological University, Singapore. His research interests include machine learning, data science, and simulation-based optimization. His email address is chew.wye@d-simlab.com.

FEIFEI ZHANG is a product manager at D-SIMLAB Technologies (Singapore). He is responsible for lot scheduling and dispatching products for customers in the semiconductor industry. He holds a Bachelor of Computer Science from the National University of Singapore. His email address is zhang.feifei@d-simlab.com.