Industrial Engineering and Applications
L.-C. Tang (Ed.)

doi:10.3233/ATDE230059

Reinforcement Learning for Quality-Oriented Production Process Parameter Optimization Based on Predictive Models

Akshay PARANJAPE [a,1], Nils PLETTENBERG [a] Markus OHLENFORST [a] and Robert H. SCHMITT [b]

[a] *IconPro GmbH, Aachen, Germany*
[b] *WZL | RWTH Aachen Univerity*

Abstract. Production of low-quality or faulty products is costly for manufacturing companies since it wastes a lot of resources, human effort, and time. Avoiding such waste requires the correct set of process control parameters, which depends on the dynamic situation in the production processes. Research so far mainly focused on optimizing specific processes using traditional optimization algorithms, mainly evolutionary algorithms. To develop a framework that enables real-time optimization based on a predictive model for an arbitrary production process, this paper explores the application of reinforcement learning (RL) in the field of process parameter optimization. Inspired by the literature review on both, production process parameter optimization, and RL, a model based on maximum a posteriori policy optimization that can handle both numerical and categorical parameters is proposed. A validation study conducted on data sets from production fields compares the trained model to state–of–the–art traditional optimization algorithms and shows that RL can find optima of similar quality while requiring significantly less time.

Keywords. Reinforcement Learning, Process Parameter Optimization, Machine Learning, Deep Learning, Genetic algorithm, Particle Swarm Optimization

1. Introduction

Potential inspection cost reduction can be achieved by inspecting the quality of the part in the production with the help of a surrogate model. These surrogate models can either be a digital clone, FEM simulation, or an approximation model. Traditionally, statistical methods were used to build an approximation model, but in the last couple of decades significant work has been done in the field of machine learning (ML) to build an approximation model, also denoted as a predictive model. The predictive model is used to predict the quality of the part in the production with the help of input parameters (using a combination of raw material parameters, assembly parameters, process control parameters, production parameters, etc). A predictive model helps to predict the quality of the part,

[1]Corresponding Author: IconPro GmbH, Aachen, Germany; E-mail: akshay.paranjape@iconpro.com

thereby reducing the inspection cost but it does not focus on improving the quality of the part or reducing the scrap rate. The solution to the latter is termed process parameter optimization, where the objective is to find an optimized set of process control parameters. Although many traditional algorithms like particle swarm optimization (PSO), genetic algorithm (GA), simulated annealing (SA), basin hopping (BA) and differential evolution (DE) [1–3] are used in recent studies, they lack the potential for real-time in-process applications. The study presented here tries to answer the two research questions:

- How can reinforcement learning (RL) be used to find quality-oriented optimized control parameters for in-process optimization of a production process?
- How does the RL perform compared to traditional optimization algorithms?

2. Recent work

There are many approaches inspired by biological evolution to find the optimal parameter solution for a simulated production process with either a white-box model or a black-box model. According to a review by Weichert et al. [2], evolutionary algorithms are commonly researched for process optimization, mostly with particle swarm optimization (PSO) and genetic algorithm (GA). Besides these, other algorithms like differential evolution (DE), basin hopping (BH), and simulated annealing (SA) are also used for finding optimal parameters of the process given a simulation model [4–12].

A vast amount of research has been conducted in the last decade in the field of RL. The most notable example is the AlphaGo by Deepmind which could defeat the best Go player [13]. The adoption of RL has immensely helped the field of robotics, especially in the assembly of automobiles. Only a little research addresses the use of RL for process parameter optimization.

He et al. [14] described decision-making in textile manufacturing as a Markov Decision Process (MDP), allowing them to use a Deep Q-Network (DQN) to find optimal parameters for a textile color fading ozonation process. The results showed that RL is a promising technique to optimize the parameters of a production process, but the authors point out that the availability of data is limited in textile production, hampering the application of machine learning.

Hayashi and Hasaki [15] used a DQN to find the optimal design of a plane frame under static loads. To optimize two parameters, an action space was modeled that allowed to discretely increase or decrease either of the two-parameter values or keep them constant. The environment returned a binary reward, depending on whether the restrictions were met. As there is no natural terminal state for such a task, the episode length was set to a fixed number of steps. This allows batch processing in the exploration phase, speeding up the process significantly. The output of the DQN was further optimized by particle swarm optimization (PSO) and simulated annealing (SA), showing better results than using only traditional methods without RL [15].

Overall, the literature review revealed that RL is only rarely used to optimize the parameters of production processes. However, the initial attempts [14,15] with RL methods for parameter optimization showed promising results. As there is no general framework available to optimize parameters across different domains, the question arises of whether RL is a suitable method to efficiently find optimal parameters for different use cases.

3. The Problem

The problem of production process parameter optimization (P3Opt) can be divided into two separate parts, namely: Process Model, and Parameter Optimization.

3.1. Scope of the research

Within the scope of this paper, only those production processes are considered whose production data consists of either continuous/numerical or discrete parameters. For a typical production process, production data consists of raw material data, assembly parameters, environmental conditions, process parameters, and data generated during the process. A few examples of process parameters include tension of wire in the wire-EDM machine; roll gaps, forces, torques for a rolling machine, applied pressure for a cleaning process, temperature zones for a forming process, etc. With the availability of both quality and production data, machine learning (ML) algorithms can be used to predict quality parameters with production data by training a model. The model that can predict the quality of the produced parts with production data is termed a *predictive quality model (PQ–model)*.

3.1.1. Process Model

Process parameters can be further divided into controllable and uncontrollable parameters depending on whether they can be manipulated during the process [2]. Performing an optimization loop to find optimized controllable parameters is highly expensive and time–consuming in a real process and therefore a process simulation is a necessary prerequisite. On the other hand, it is also difficult to create a Finite Element Method (FEM) based simulation for a complicated production process. With the availability of historical production data and quality data, a PQ–model can be trained with ML algorithms [16] which can then be used as a surrogate model of the production process.

3.1.2. Control Parameter Optimization

In order to improve the quality of the produced parts and to nullify the production of faulty parts, a recommendation mechanism is required that can suggest possible changes for control parameters in a production process. Without loss of generality, all production parameters can be divided into controllable X_c and uncontrollable parameters X_u. For i^{th} part produced in the process under production, the goal is to find optimized X_c parameters that optimize the quality of the finished part in the next iteration. Quality–oriented optimization has twofold benefits: first, it overcomes the defect if the i^{th} iteration produced faulty parts, and second it enhances the quality by enhancing the confidence score of optimized values based on the PQ–model. The optimization problem represented in (1) finds optimized parameters X_c with the trained optimization model.

$$X_c = F_{P3Opt}(X^o, M_{process}) \tag{1}$$

where X_c denotes list of control parameters, F_{P3Opt} is the optimization model to be trained, X^o denotes the process parameters at i^{th} iteration and $M_{process}$ is the process model aka. PQ–model.

3.2. Out of scope

Time–series and image data are out of the scope of this paper. Time–series constitutes data coming from sensors, for example, the vibration of a table during a milling process, variation in temperature, force applied for a forming process, etc. A few examples of production use cases with image data include ultrasonic imaging of resistance spot welding, microscopic imaging of parts for technical cleaning, etc.

4. Reinforcement Learning

As evident from section 2, PSO and GA are the most widely used algorithms for finding optimized process parameters. In this work, RL approach is considered to find optimized control process parameters for real–time optimization. A comparative study of evolutionary algorithms and RL based algorithms is conducted in Section 6 to validate the approach.

ML can be broadly divided into supervised, unsupervised, and reinforcement learning (RL). Supervised learning uses labeled data, unsupervised learning learns patterns from unlabeled data, whereas RL learns by interacting with an environment. The goal of RL is to maximize a reward by taking optimal actions in an environment. The power of RL was recently demonstrated in various scenarios ranging from relatively simple applications like Atari games [17] to more complex ones like the game Go or even self–driving vehicles. Büscher et al. [18] recommended the possible usage of RL for resource efficient cleaning process. Three basic elements of an RL algorithm are agent, environment, and reward.

Environment and agent: As the overall goal of RL is to learn optimal decisions, the first thing required is an instance of making these decisions and learning from them. This decision–maker, which can also be considered a controller, is called the agent. The agent is continuously interacting with a controlled system, the environment. The environment sends a control signal, an observation, to the agent. Based on the observation, the agent decides on an action, which it transmits to the environment. The environment will take the action into account to compute a new state. Information about the new state is then again transferred to the agent in the form of an observation and in addition, the agent receives a reward for its action. These steps are repeated until a termination criterion is met.

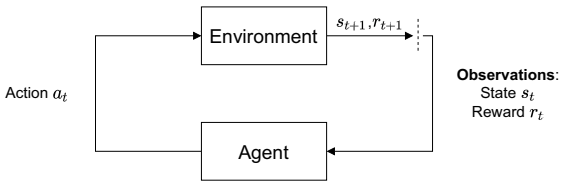

Figure 1. Interaction between agent and environment.

Although continuous time scenarios are possible, RL usually builds on the assumption of discrete time steps. For these steps $t \in \{0, 1, 2, ..., T\}$, the environment provides the agent with a representation of its current state $S_t \in \mathscr{S}$ where $\mathscr{S}$ is a set of possible states. The

agent selects an action $A_t \in A(S_t)$ where $A(S_t)$ is the set of all possible actions for S_t. In the next time step, the agent receives a numerical reward R_{t+1} and the updated state S_{t+1}, as shown in Fig. 1. Due to the abstract formulation of the RL framework, it is applicable to a wide range of decision–learning problems and has proven to be quite useful for most of them.

Rewards and Returns: Rewards are required to formalize the goal of the agent. The agent receives a reward $R_t \in$ IR at each time step and the goal is to maximize the total reward. Therefore, it must always take the action that will yield the highest cumulative future reward, which may not be the highest reward for the current time step. When modeling the reward, it is important to ensure that it reflects on the actual goal and not the way the goal should be achieved. For example, if a robot is trained to play football, it should be rewarded for winning a match, not only for scoring goals. The latter could lead to the robot scoring many goals but allowing the opponent to score even more, and ultimately failing to achieve the actual objective. Rewards are given for single–time steps, an additional term is used to formalize the objective of an RL task: the expected return G_t. For the simplest case, the expected return is just the sum of all future rewards $G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$ [19]. There are mainly two main approaches for solving RL problems, namely methods based on value function and methods based on policy search. These are described in the below subsections.

4.1. Value function

Value function methods are based on estimating the value (expected return G_t) of being in a given state. The state–value function $V_\pi(s)$ is the expected return when starting in states and following the policy (π) thereafter. Assuming $V^*(s)$ is the value function with optimal policy $\pi*$, optimal state value function can be defined as:

$$V^*(s) = \max_\pi V^\pi(s) \tag{2}$$

In practice, a different function, state–action value function which is also called as Q-function, is used rather than the value function since transition dynamics are generally unavailable. The Q–function $Q^\pi(s,a)$ is similar to V^π with the difference that an initial action a is given and the policy π is followed from the next state onwards [20]. Actual Q–learning is realized by defining the function as a Bellman equation [21].

4.2. Policy Search

As opposed to the value function, the policy search method directly searches for an optimal policy π^* rather than learning a value function. Parameters π_θ of policy π are updated to maximize the expected return $E[R|\theta]$ with either gradient–based or gradient–free approaches [22]. Gradient–free approaches are generally used for lower dimensional problems whereas gradient–based approaches have been found to perform better for higher dimensional problems.

4.3. Actor Critic Methods

Directly learning a policy rather than state–action values has some advantages, e.g., it enables learning for continuous and high–dimensional action spaces, a domain where

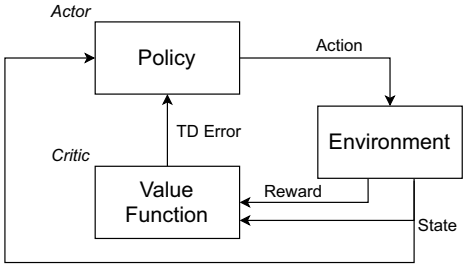

Figure 2. Actor Critic Method

pure value–learning usually struggles. However, they also have some shortcomings compared to value learning. To overcome these shortcomings, the actor–critic concept was introduced which combines value function with a representation of policy. It consists of two parts, the policy part, *actor* that selects actions, and the estimated value function, *critic*, that evaluates the action made by the actor. The major difference between actor–critic methods and other policy baseline methods is that actor–critic methods use the value functions as a baseline for policy. Actor–critic methods were already used over 50 years ago but did not play a major role since the introduction of Q–learning in the early nineties. However, this changed recently with the success of deep Q–learning, because in addition to the aforementioned handling of continuous action space, actor–critic methods only require a minimum of computation to select actions and they can explicitly learn stochastic policy/policies [19]. Fig. 2 describes the actor–critic method.

4.4. Model–based methods

Model–based reinforcement learning uses a model of the environment to make predictions about how the environment will behave. This allows the RL agent to learn from it's experience and make more informed decisions about what actions to take in the future. In contrast, model–free reinforcement learning does not use a model of the environment and instead relies on direct interactions with the environment to learn about the consequences of actions.

4.5. Deep Reinforcement Learning

While many approximation methods can produce satisfactory results, no other method has received attention in the last decade like Deep RL (DRL) has. Inspired by the success of deep learning in other domains, its application has become increasingly popular, making it the most common method that is used today. The popularity of DRL has been boosted by breakthroughs in several domains, starting with Atari games in 2013 [17] and the well–recognized AlphaGo, which beat the Go world champion in 2015 [13]. Some of the most important approaches for DRL are described in the below subsections.

4.5.1. Deep Q–learning

The Deep Q–Network (DQN) approach showed that, contrary to the general belief at the time, learning value functions using large non–linear approximators like neural networks

(NNs) can be stable. The stability of learning was achieved by two key ideas: first, the authors trained the network with data from an experience replay buffer. The experience replay buffer is a finite cache storing transition samples in the form of the tuple (s_t, a_t ,r_t , s_{t+1}). The transitions are sampled from interactions with the environment following an exploration policy. When the buffer is full, the oldest samples are discarded. Using a buffer addresses some issues that prevented the success of DRL up to this point.

4.5.2. Deep Deterministic Policy Gradient (DDPG)

Although DQN was a breakthrough for DRL, it has one major drawback: it is only able to handle discrete action spaces, i.e., the number of actions per dimension is finite. In reality, however, many control tasks require continuous control. To address this issue, Lillicrap et al. [23] introduced Deep Deterministic Policy Gradient (DDPG), an actor–critic method that builds up on DQN.

4.5.3. Maximum a posteriori Policy Optimisation (MPO)

Another actor–critic method that uses deep learning is maximum a posteriori policy optimization (MPO). It has some similarities with DDPG but is based on the coordinate ascent on a relative entropy objective. The key idea of MPO is to transform the RL problem into an inference problem. This makes it possible to apply inference methods such as the expectation maximization (EM) algorithm. The algorithm alternates between the E–step, and the M–step, which updates the policy using the weighted state–action samples from the E–step as targets for supervised learning.

5. Methodology

5.1. Surrogate Model for production process

A real process in a production unit can be prone to behavioral changes. This can be a result of various influencing factors like environmental temperature, changes in machining parameters, various human factors, etc. Thus, training of an RL–agent for optimization of the production process parameter on a real process can lead to training with an incomplete number of possibilities that occurred historically. Apart from that, training on the real process in most cases is not possible owing to the different production units, processing time, and evaluation time. Therefore, a simulation of the process is desired. A FEM simulation of the process which takes into consideration various influencing factors can be used as a surrogate model when available. In other cases, where historical production data and quality data are available, an ML–based predictive model can be used as a surrogate model provided it is able to find good relationships between input and output parameters. The model performance should be acceptable for the production process under consideration. The predictive model takes process input parameters and outputs the quality target, thus approximating the behavior of the production process. For the experiments mentioned in this paper, an `XGBoost` model was trained. In practice, this can be done with Automated Machine Learning (AutoML) solutions. The predictive model is used as an environment for the RL–agent interface to mimic the real production environment

5.2. Reinforcement Learning Models

The predictive model described in Fig. 6 acts as an environment for RL–agent. RL–agent was designed with `Hybrid--MPO` model proposed by Neunert et al. [24]. It can support both numerical as well as continuous action spaces. The general structure of the model is similar to DDPG, at its core are four NNs: one actor network π, one critic network Q, and the respective target networks. The actor–network has to jointly output continuous

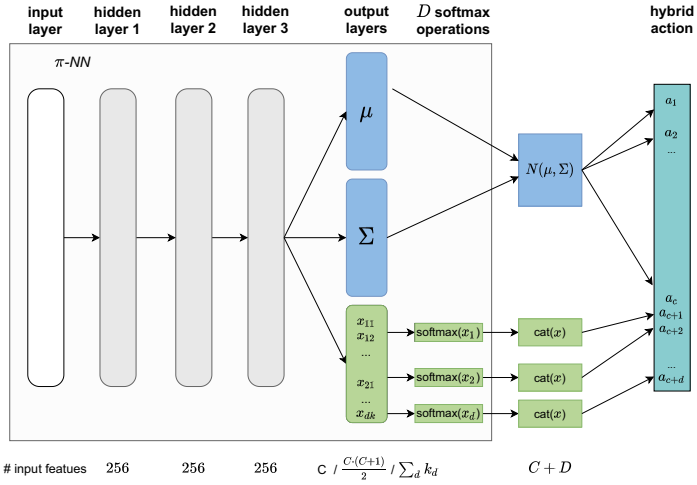

Figure 3. Architecture of the actor network π. The target network π' shares the same architecture, it's weights are updated at the end of each iteration.

and categorical actions. To do this, a fully–connected NN with three hidden layers is used, which is shown in Fig. 3. Each layer has a size of 256 neurons. After each layer, batch normalization is applied and the output is passed to a Rectified Linear Unit (ReLU) activation function. Choosing suitable layer sizes is a trade–off between the complexity of the model and its ability to handle complex data with many input and output dimensions. They were selected based on the original `Hybrid MPO` model. The input layer size matches the number of all parameters, both controllable and uncontrollable ones. The actor–network has three output layers. The first output layer returns the mean μ for all continuous actions c. The second output layer returns the covariance Σ for all continuous actions in form of a Cholesky matrix L. As L is a lower triangular matrix, the size of the second output layer is $\frac{c\cdot(c+1)}{2}$. Based on a normal distribution using μ and Σ, c continuous actions are sampled. The third output layer of the network contains the probabilities for all discrete actions. For each discrete action dimension d, the probabilities for all categories of this feature k_d are computed by applying D softmax operations on the corresponding outputs of the network. These probabilities are then used to sample D discrete actions. Both discrete and continuous actions are concatenated to one vector to be processed by the environment and the critic network. A simple fully connected network with three layers of size 512 each is used to approximate the value function of the critic Q. The size of the input layer is defined by the sum of both continuous and discrete actions and the number of state dimensions which is equivalent to the number

of all process parameters, both controllable and uncontrollable. The current state and the actions selected by the actor are simply concatenated to one input vector. The output of the critic network is a scalar, the estimated Q-value.

In addition to the networks, a replay buffer is used to generate statistically independent samples that allow stable training. The networks are trained iteratively. At the beginning of each iteration, trajectories are sampled to fill the replay buffer. This is done using the target actor π'. First, a sample is drawn from the training data set by the environment. Only samples that are not in the target class are considered. This way, the actor has to learn how to optimize poor samples. If it would get mainly samples from the target class, it might struggle to deal with parameters that have a low probability for the desired classification later. The environment clamps the features of s_0 before passing it to the target actor. The target actor selects an action that is processed by the environment to compute the new state s_{t+1} and reward r_{t+1}. The tuple $(s_t, a_t, s_{t+1}, r_{t+1})$ is saved in the replay buffer to be used for training the actor and critic later. This process is depicted in Fig. 4. It is repeated for a fixed number of steps, the episode length. The entire process described above is repeated for multiple episodes until the replay buffer is filled. The size of the replay buffer can be set manually as a hyperparameter. Once the replay buffer

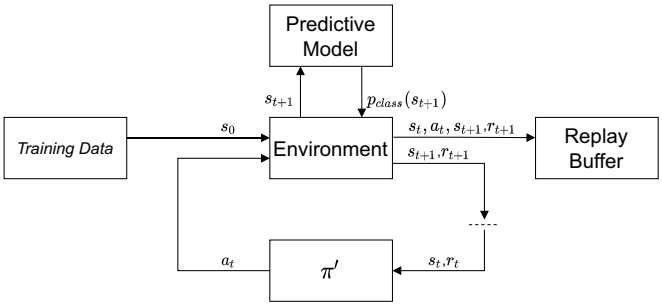

Figure 4. Trajectory sampling during training. Note that the target actor π' is used to select actions.

is filled, the next step is to update the critic using temporal difference (TD) learning. This is a distinct difference from the original hybrid MPO model where Retrace is used to update the critic network. To compute the loss of the Q network, the `SmoothL1` loss function $\ell(x,y)$ was used:

$$\ell(x,y) = \begin{cases} \frac{0.5(x-y)^2}{\beta} & |x-y| < \beta \\ |x-y| - \beta \cdot 0.5 & otherwise. \end{cases} \tag{3}$$

`SmoothL1` loss was chosen because it offers steady gradients for large values of x like `L1` and reduces oscillation for small updates of x like `L2` [25]. To ensure stable learning, both critic Q and target critic Q' were considered to compute the loss `L`, similar to the loss for DDPG.

$$L = \ell\left(R_{t+1} + \gamma Q(S_{t+1}, a), Q'(S_t, A_t)\right) \tag{4}$$

β was set to 1 as this is the value most frequently suggested in the literature. The discount factor γ was set to 0.99, which is also a value that was used for the original hybrid–MPO

model. The critic is optimized using "adaptive moment estimation" method, commonly referred as Adam, with a learning rate of 0.0003, which also matches the original model [24]

To train the critic, the states from the replay buffer are passed to the target actor. However, instead of getting the hybrid action vector, the distributions are stored (the second–to–last step in Fig. 3). These distributions are used to sample 64 new hybrid action vectors for each state which are then passed to the Q' to get the corresponding Q–values. To complete the E–step of MPO, q_{ij} are computed according to (5).

$$q_{ij} = q(a_i, s_j) = \exp\left(\frac{q^{\pi}(s_i, s_j)}{\eta}\right) / Z(j) \tag{5}$$

where $Z(j) = \sum_i exp(\frac{q^{\pi}(s_i,s_j)}{\eta})$ and η is the temperature parameter. Adam is used to optimize η. The constraint for the dual function ε_{dual} is set to 0.1. Finally, the M–step has to be applied. To compute the loss for the actor network π, the outputs μ, Σ and p are decoupled. For discrete actions, the average log probabilities and KL divergences for all discrete action dimensions D are computed; the categorical probabilities p_d are decoupled for each discrete feature d.

Both E–steps and M–steps are computed in batches of size 128. The batch size was deliberately chosen to be much smaller than the one used for the original model (3072) as large batch sizes might compromise the model's quality. Furthermore, the original model had to process state and action spaces of lower dimensions on a more powerful graphics processing unit (GPU). As this model is intended to be used for complex data sets on mid–range GPUs, a small batch size ensures that the entire batch fits into the memory of the GPU.

All tuples in the replay buffer were rerun five times for each iteration. In this way, fewer samples are required and the number of function evaluations of the predictive model is reduced, which speeds up the training. At the end of one iteration, the parameters of the target network are updated by copying the ones of the actor and critic network, respectively. Training is run for 50 iterations with early stopping if the mean reward does not improve over ten iterations. Once training is completed, it is possible to use the model to predict optimal parameters for a given sample. To do this, the model is provided with the original process parameters, both controllable and uncontrollable ones. The next steps are very similar to trajectory sampling to fill the replay buffer. Only for prediction, the action actor network π is used rather than the target π' and only states and corresponding rewards are stored in the buffer. 200 pairs of states and rewards are sampled, and the pair that has the highest reward is selected as optimal parameters. The prediction procedure is shown in Fig. 5.

While a high number of iterations for the prediction process generally improves the result, it is also possible to choose a number smaller than 200. Pre–tests showed that even with only ten iterations, the results are usually still acceptable. However, if function evaluations of the predictive model are not too expensive, it is preferable to consider a higher number of samples to increase the probability of finding an optimized point. Apart from the function evaluations, the prediction is very fast because the actor is implemented by a simple fully connected NN. Additionally, processing scales very well for multiple initial process parameters because of batch processing.

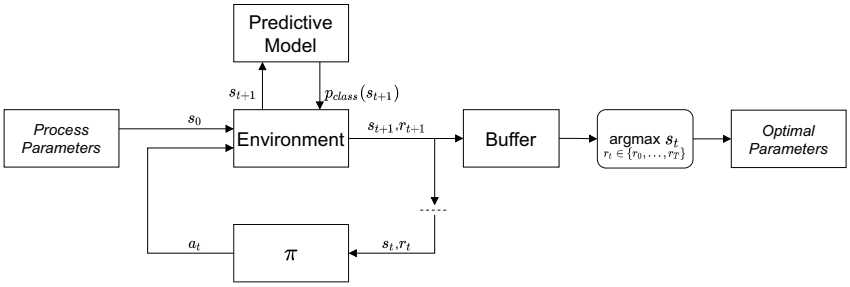

Figure 5. Flow chart for prediction of optimized process parameter using the trained policy and predictive model

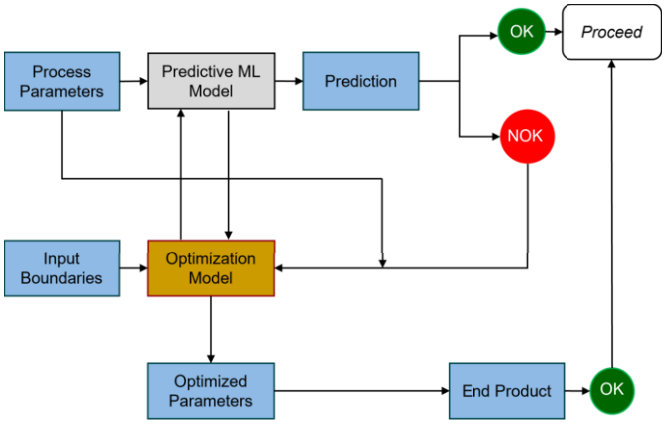

Figure 6. Experiment set–up for in–process process parameter optimization. Pre–trained predictive model is the requirement for the process parameter optimization model to start training.

6. Experiments and Results

6.1. Experimental set–up

There are three major sections of the experimental setup, namely the predictive model, the parameter optimization model, and the prediction endpoint. The experimental setup is depicted in Fig. 6. As a part of the preparation for P3Opt, production data and quality data from real production lines should be collected. The first step then is training a predictive model with production data and labels as quality data, the second step is training a process parameter optimization model and the last step is to use the process parameter optimization model for in–process optimization of the production processes. The setup shown in Fig. 6 is specifically for a tabular structured data set.

6.2. Data sets

The focus of the research is to find a generalized approach for P3Opt. The validation is not focused on a particular data set but rather on a group of seven production data sets that originate from various fields, including the production of turbochargers, LEDs, steel plates, 3D–printing, rotogravure printing, etc. These data sets vary not only in their origin

Table 1. Details of the production data sets that were used for testing. $\mathscr{F}$ is the number of input parameters, $\mathscr{C}$ the number of controllable continuous parameters, $\mathscr{D}$ the number of controllable discrete parameters, $\mathscr{P}$ denotes number of combinations for $\mathscr{D}$, and $\overline{\kappa}_{rel}$ is the mean condition number of the data sets.

Name	Type	Samples	$\mathscr{F}$	$\mathscr{C}$	$\mathscr{D}$	$\mathscr{P}$	score	$\overline{\kappa}_{rel}$
DF1	reg	75977	15	6	2	8	0.994	2.749
DF2	clf	50	12	7	1	2	1.000	0.000
DF3	reg	244	15	8	1	4	0.795	0.006
DF4	clf	3614	171	103	6	486	0.724	801.856
DF5	clf	1941	28	18	1	2	0.985	3.866
DF6	clf	277	35	15	9	12960	0.769	30.416
DF7	clf	6938	29	19	2	16	0.989	13.681

but also cover a broad range of complexity including the number of input features, task type i.e. regression or classification, and multiple categorical (discrete) features which lead to a higher number of total combinations for RL agent's action space. For all data sets, approximately 30% of the input features were selected as fixed. The remaining controllable features are split into continuous features $\mathscr{C}$ and discrete features $\mathscr{D}$. For all selected production data sets, the vast majority of controllable features were numerical. The value span for the controllable features was calculated from historical data and it was used to limit the search space for RL–agent. The accuracy or r2-score of the predictive models used for prediction was high for all data sets.

6.3. Results and Validation

Experimental results for eight production data sets are summarized based on three metrics, namely *probability score*, *distance*, and *stability*. All three together constitute the reward for the RL algorithm
Reasons for the choice of metrics:

- *objective score*: The aim of the predictive model is to inform how good are the input values to get the target output, which is represented by class probability for classification tasks or absolute difference for regression tasks. Probability score being the major scoring factor, it only makes sense to include it as a part of the reward.
- *distance*: For in–process optimization, one would like to avoid a drastic change in the control parameter. For example, it would be easier and more practical to change a temperature by 5° Celsius than by 50° Celsius. Hence, a distance factor is introduced in the reward function which considers the distance between the current state and the next state. This is done with one of the most frequently used distance metrics for data that contains both numerical and categorical features - Gower distance [26]. It assigns a similarity score s_{ij} between two samples i and j.
- *stability*: It is possible that the trained predictive model has singularities. To avoid getting trapped in the singularities of the predictive model, a stability check is introduced for the parameter solution as a part of the reward function. This stability check is conducted by calculating the condition number (κ_{rel}). A lower condition number implies higher stability for the parameter solution.

$$\kappa_{rel} = \limsup_{\tilde{x} \to x} \frac{||f(\tilde{x}) - f(x)||}{||f(x)||} \frac{||x||}{||\tilde{x} - x||} \tag{6}$$

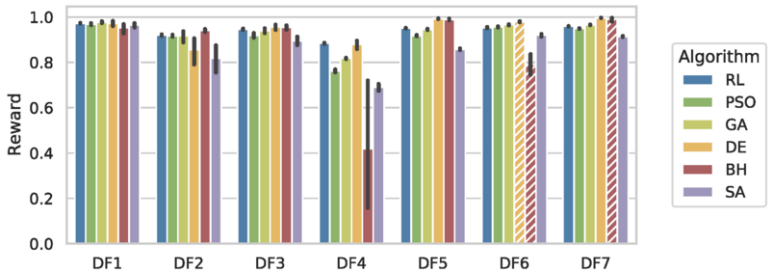

Figure 7. Total rewards for the different algorithms on a selection of production data sets. Error bars denote 95% confidence intervals. Crosshatched bars indicate incomplete results due to the time limit.

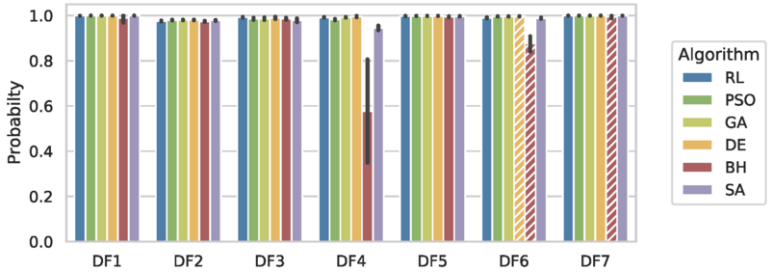

Figure 8. Probability for the different algorithms on a selection of production data sets. Error bars denote 95% confidence intervals. Crosshatched bars indicate incomplete results due to the time limit.

A comparison between the performance of RL–P3Opt with respect to other traditional optimization techniques like PSO, GA, SA, DE, and BH is presented and compared across these three metrics. Due to the complex nature of some production data sets, even with the limitation of only considering the most important categorical features for optimization with traditional algorithms, the runtime was very long for some of them. For example, optimizing 100 samples from DF4 with BH took over a week, and the optimization with DE was canceled due to a high expected runtime. A time limit of eight hours per algorithm was introduced for the test study. If an algorithm did not complete the optimization of all 100 samples within the time limit, it was stopped and impartial results are reported. Impartial results are marked with crosshatched bars in the plots. Fig. 7 shows the total reward obtained by the algorithms for the different data sets.

With some minor exceptions, all algorithms generally performed well. RL performed just as well as the evolutionary algorithms, but for most data sets, BH showed very good results too. However, its total reward was lower for more complex data sets like DF6 and DF4. The only algorithm that failed to find optima of the same quality as the others is SA. Looking at the reward, DF4 slightly stands out because all algorithms obtained a lower reward than for any other data sets. Nevertheless, RL performs at least as well as or better than all other algorithms, demonstrating its capability to handle complex data sets.

The bar chart in Fig. 8 shows the objective score for the optimized parameters from the predictive model. For a classification task, the objective score is the prediction prob-

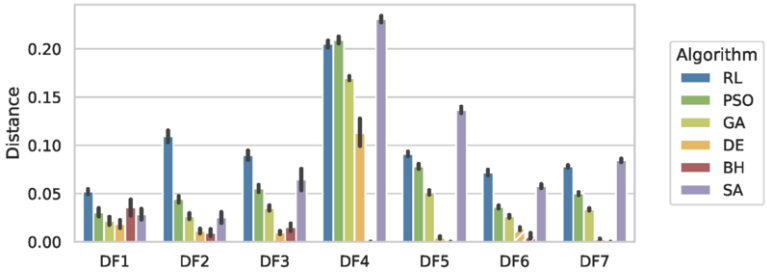

Figure 9. Distance for the different algorithms on a selection of production data sets. Error bars denote 95% confidence intervals. Crosshatched bars indicate incomplete results due to the time limit.

ability score and for a regression task, it is the normalized mean absolute error (MAE) (c.f. (7)).

$$\begin{aligned} O_{clf} &= p_{class} && \in [0,1] \\ O_{reg} &= 1 - \frac{|\hat{y} - p|}{y_{range}} && \in [0,1] \end{aligned} \tag{7}$$

where p_{class} is the desired class probability obtained from a classification model, $\hat{y}$ is the optimal numerical target value and p represents the predicted target value from a regression model, and y_{range} is the span of target parameter values $y_{range} = max\{\hat{y} - y_{min}, y_{max} - \hat{y}\}$. All algorithms could find optimized samples with an extremely high objective score. The only exception is BH with an acceptable mean probability well above 0.8 for DF6 and a very unstable result for DF4. However, BH struggled with that particular data set in general. It could only optimize six samples within the time limit of eight hours, which is why the numeric results are not very meaningful.

A weakness of the RL model is revealed in Fig. 9; it struggles to find optima that are as close to the original sample as those found by other algorithms. With the exception of DF4, where RL has some advantages because it is the most complex data set, the traditional algorithms are superior in this aspect. Even SA, which is the algorithm that generally shows the poorest results, was able to find optima that are much closer to the original sample for most data sets. Overall, the optimized points found by RL are still close. Furthermore, the distance for RL shows a low variance for all data sets, i.e. there are not many outliers and the results are reliable. Thus, the RL model's performance in this aspect is still satisfactory, even though traditional algorithms clearly have an advantage if the focus is on finding optima that are as close as possible to the original samples.

The stability of the data set is measured with its corresponding predictive model in the form of the relative condition number κ_{rel}. The mean condition number ($\overline{\kappa}_{rel}$) for a data set is the mean of relative condition numbers calculated on random samples from data set. Condition number varies depending on the data set. For example, DF4 has a very high mean condition number $\overline{\kappa}_{rel}$ of 801.856; a higher condition number implies a less stable data set. Hence, finding a stable point for such a data set is much harder than for data sets like DF2 or DF3, where $\overline{\kappa}_{rel}$ is close to 0. For this reason, Fig. 10 shows the condition numbers separated by data sets. The optimization results on the production

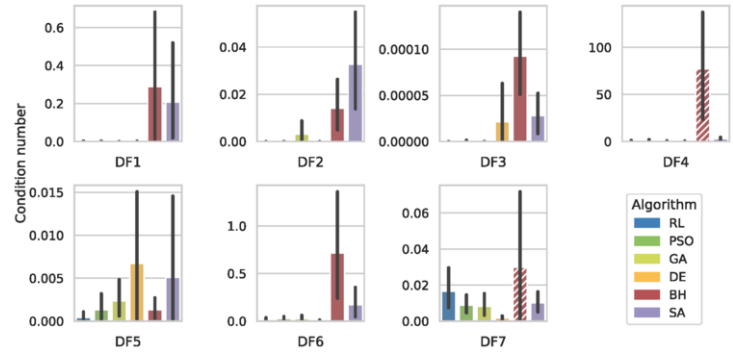

Figure 10. Condition for the different algorithms on a selection of production data sets. Error bars denote 95% confidence intervals. Crosshatched bars indicate incomplete results due to the time limit.

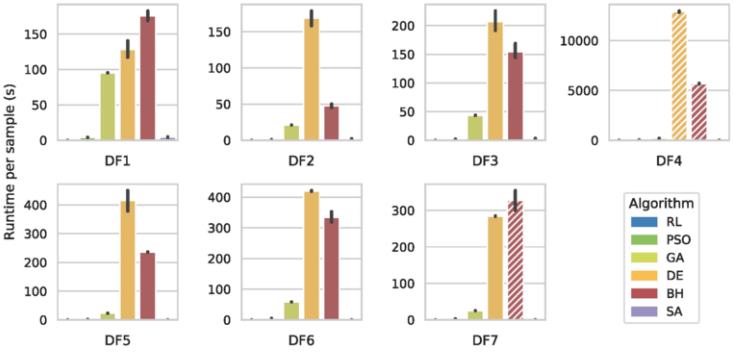

Figure 11. Runtime for optimization for the different algorithms on a selection of production data sets. Error bars denote 95% confidence intervals. Crosshatched bars indicate incomplete results due to the time limit.

data sets show that RL, PSO, and GA consistently find very stable optima. DE was, with few exceptions, also able to find stable optima, while SA and BH in particular struggled to find optima with a low condition number.

The time required to optimize a single sample correlates strongly with the complexity of the data set for the traditional algorithm. To increase the visibility of small values for simple data sets, the results in Fig. 11 are therefore also split by data sets. Despite being slower than RL, PSO, and SA also require significantly less time than the remaining three algorithms. Out of those, DE is the slowest on most data sets. This is especially the case for DF4, where DE could only optimize three samples before the time limit was exceeded. The optimization of a single sample took over 3.5 hours on the GPU server with an AMD Ryzen 7 3800X processor with eight physical cores and 64 GB of RAM. BH also takes much longer than the three fastest algorithms, but the runtime does not correlate as strongly with the data set complexity as for DE. While it finds optima of poorer quality than RL and evolutionary algorithms, it also failed to optimize all 100 samples on two data sets. Finally, GA is also much slower than RL, PSO, and SA, but scales much better than DE and BH. It is not affected strongly by the number of controllable features.

The runtimes on the production data sets emphasize the importance of batch processing. The RL model is able to process all 100 test samples in a single batch, making it

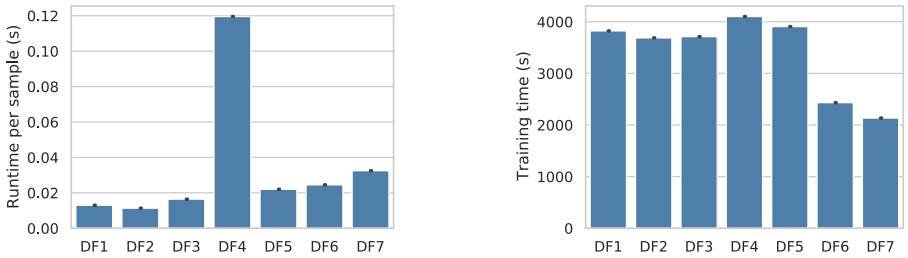

Figure 12. Runtime for the RL model on a selection of production data sets. Runtime is split into optimization prediction time (left) and training time (right).

by far the fastest algorithm, the PSO implementation is at least able to process the entire population for a single sample at once, but this is not the case for GA or DE. This is one reason why PSO is much faster than the other evolutionary algorithms. Fig. 12 shows the time it takes to optimize a single sample inside a batch of 100 for the different data sets. The results indicate that optimization runtime is highly correlated with the number of features in the data set. Training, on the other hand, is only very slightly impacted by the complexity of the data set. DF7, which is one of the more complex ones, in fact, has the shortest training time. It is possible that this counter–intuitive result is caused by a lower overall load of the server at the time of prediction.

7. Conclusion

While process parameter optimization is not a new field, the scope of RL in this field had not been explored extensively. Additionally, traditional process parameter optimization techniques, mainly evolutionary algorithms, are not suitable for real–time optimization due to their high computation time. This paper explores the possibilities of RL algorithms to find optimization results in real–time that can be used for in–process production optimization. To accommodate both continuous and discrete parameters, a Hybrid–MPO proposed by Neunert et al. [24] is recommended. The validation study results show that RL optimization is as good as traditional optimization algorithms in terms of reward and much better in terms of prediction time. For traditional optimization algorithms, the prediction time is the same as the optimization time since there is no training of the model. The study conducted in this paper does not focus on a particular production use case, it provides a general procedure for production parameter optimization which can be explored and fine–tuned as per the production use–case requirements. The proposed optimization model can help manufacturing companies optimize their processes which not only saves cost but also results in more efficient use of resources, ultimately improving the sustainability of production.

References

[1] L. Zhang, Z. Jia, F. Wang, and W. Liu, "A hybrid model using supporting vector machine and multi-objective genetic algorithm for processing parameters optimization in micro-EDM," *The International Journal of Advanced Manufacturing Technology*, vol. 51, no. 5-8, pp. 575–586, 2010.

[2] D. Weichert, P. Link, A. Stoll, S. Rüping, S. Ihlenfeldt, and S. Wrobel, "A review of machine learning for the optimization of production processes," *International Journal of Advanced Manufacturing Technology*, vol. 104, no. 5-8, pp. 1889–1902, 10 2019. [Online]. Available: https://link.springer.com/article/10.1007/s00170-019-03988-5

[3] B. Olson, I. Hashmi, K. Molloy, and A. Shehu, "Basin Hopping as a General and Versatile Optimization Framework for the Characterization of Biological Macromolecules," *Advances in Artificial Intelligence*, vol. 2012, pp. 1–19, 2012. [Online]. Available: http://downloads.hindawi.com/archive/2012/674832.pdf

[4] J. Pfrommer, C. Zimmerling, J. Liu, L. Kärger, F. Henning, and J. Beyerer, "Optimisation of manufacturing process parameters using deep neural networks as surrogate models," in *Procedia CIRP*, vol. 72. Elsevier B.V., 1 2018, pp. 426–431.

[5] R. V. Rao and P. J. Pawar, "Modelling and optimization of process parameters of wire electrical discharge machining," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 223, no. 11, pp. 1431–1440, 11 2009.

[6] E. Wei, D. Wicke, and S. Luke, "Hierarchical Approaches for Reinforcement Learning in Parameterized Action Space," Tech. Rep., 2018. [Online]. Available: www.aaai.org

[7] A. Peng, X. Xiao, and R. Yue, "Process parameter optimization for fused deposition modeling using response surface methodology combined with fuzzy inference system," *The International Journal of Advanced Manufacturing Technology*, vol. 73, no. 1-4, pp. 87–100, 2014.

[8] W. C. Chen, G. L. Fu, P. H. Tai, and W. J. Deng, "Process parameter optimization for MIMO plastic injection molding via soft computing," *Expert Systems with Applications*, vol. 36, no. 2 PART 1, pp. 1114–1122, 3 2009.

[9] K. Bouacha and A. Terrab, "Hard turning behavior improvement using NSGA-II and PSO-NN hybrid model," *International Journal of Advanced Manufacturing Technology*, vol. 86, no. 9-12, pp. 3527–3546, 10 2016. [Online]. Available: https://link.springer.com/article/10.1007/s00170-016-8479-6

[10] J. A. Silva, J. V. Abellán-Nebot, H. R. Siller, and F. Guedea-Elizalde, "Adaptive control optimisation system for minimising production cost in hard milling operations," *International Journal of Computer Integrated Manufacturing*, vol. 27, no. 4, pp. 348–360, 4 2014.

[11] D. S. Srinivasu and N. R. Babu, "An adaptive control strategy for the abrasive waterjet cutting process with the integration of vision-based monitoring and a neuro-genetic control strategy," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 5-6, pp. 514–523, 2008.

[12] Y. Rong, G. Zhang, Y. Chang, and Y. Huang, "Integrated optimization model of laser brazing by extreme learning machine and genetic algorithm," *International Journal of Advanced Manufacturing Technology*, vol. 87, no. 9-12, pp. 2943–2950, 12 2016. [Online]. Available: https://link.springer.com/article/10.1007/s00170-016-8649-6

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 1 2016.

[14] Z. He, K. P. Tran, S. Thomassey, X. Zeng, J. Xu, and C. Yi, "A deep reinforcement learning based multi-criteria decision support system for optimizing textile chemical process," *Computers in Industry*, vol. 125, 2021.

[15] K. Hayashi and M. Ohsaki, "Reinforcement learning for optimum design of a plane frame under static loads," *Engineering with Computers*, vol. 1, p. 3, 1 2020. [Online]. Available: https://doi.org/10.1007/s00366-019-00926-7

[16] A. Paranjape, P. Katta, and M. Ohlenforst, "Automated data preprocessing for machine learning based analyses," *COLLA 2022, The Twelfth International Conference on Advanced Collaborative Networks, Systems and Applications*, 05 2022.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 12 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[18] J. Büscher, A. Paranjape, R. Möhle, M. Polikarpov, N. Plettenberg, R. Zwinkau, J. Deuse, and R. H. Schmitt, "Bauteile ressourceneffizient reinigen mithilfe von ki," *JOT Journal für Oberflächentechnik*, vol. 63, no. 1, p. 40–43, 2022.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *CoRR*, vol. abs/1708.05866, 2017. [Online]. Available: http://arxiv.org/abs/1708.05866

[21] R. Bellman, "On the theory of dynamic programming," *Proceedings of the Na-*

tional Academy of Sciences, vol. 38, no. 8, pp. 716–719, 1952. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.38.8.716

[22] M. P. Deisenroth, G. Neumann, and J. Peters, *A Survey on Policy Search for Robotics*, 2013.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 9 2016. [Online]. Available: https://goo.gl/J4PIAz

[24] M. Neunert, A. Abdolmaleki, M. Wulfmeier, T. Lampe, J. T. Springenberg, R. Hafner, F. Romano, J. Buchli, N. Heess, and M. Riedmiller, "Continuous-discrete reinforcement learning for hybrid control in robotics," in *Proceedings of the Conference on Robot Learning*, L. P. Kaelbling, D. Kragic, and S. Komei, Eds., 2020, pp. 735–751.

[25] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[26] J. C. Gower, "A General Coefficient of Similarity and Some of Its Properties," *Biometrics*, vol. 27, no. 4, p. 857, 12 1971.