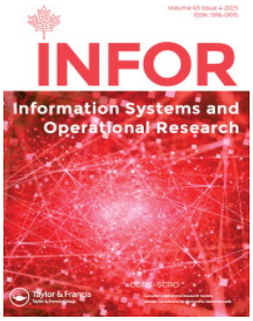
Taylor & Francis
Taylor & Francis Group



INFOR: Information Systems and Operational Research

ISSN: 0315-5986 (Print) 1916-0615 (Online) Journal homepage: www.tandfonline.com/journals/tinf20

Simulation-based generation of heuristics for decision-making in stochastic environments

Andreas Körner, Daniel Pasterk, Florian Stadler & Christine Zeh

To cite this article: Andreas Körner, Daniel Pasterk, Florian Stadler & Christine Zeh (26 Nov 2025): Simulation-based generation of heuristics for decision-making in stochastic environments, INFOR: Information Systems and Operational Research, DOI: 10.1080/03155986.2025.2592355

To link to this article: https://doi.org/10.1080/03155986.2025.2592355

Published online: 26 Nov 2025.

INFOR: INFORMATION SYSTEMS AND OPERATIONAL RESEARCH
https://doi.org/10.1080/03155986.2025.2592355

Simulation-based generation of heuristics for decision-making in stochastic environments

Andreas Körner, Daniel Pasterk, Florian Stadler and Christine Zeh

Institute of Analysis and Scientific Computing, TU Wien, Vienna, Austria

ABSTRACT

Decision-making in stochastic environments often requires a trade-off between performance and interpretability. Although Reinforcement Learning (RL) excels at creating adaptive policies, the resulting solutions are not transparent. Conversely, while heuristics offer transparency, they often lack optimality and adaptability. In this work, we present a general framework that combines the strengths of both approaches. First, we use RL to train a policy on a Markov Decision Process (MDP). Then, we extract transparent heuristics in the form of decision trees via interpretable learning (VIPER). To conclude our method, we apply pruning to the tree, aiming to simplify its structure and improve the generalisation of the resulting rule set. We demonstrate this approach using a logistics case study involving significant variability in production and demand. The resulting heuristics outperform expert-designed rules and match the performance of the original RL policy, offering transparency and robustness. This method allows for data-driven, explainable decision-making that does not require domain-specific expertise.

Nomenclature: $\mathcal{C}$: customers; $\mathcal{M}$: machines; $\mathcal{P}$: products; A: actions; c_p: production cost; $D_{c,t}$: demand; E: timesteps to store in inventory; $E(s)$: entropy; f_p: completion time of product; G_t: return at time t ; $I(N)$: impurity; $I(t,i)$: inventory; J: categories; $q_\pi(s,a)$: action value function; $q_*\pi(s,a)$: optimal action value function; R: rewards; r_c: revenue; R_i: selling price; S: states; T: max timestep; $v_\pi(s)$: value function; $w(s)$: sample weight

ARTICLE HISTORY
Received 29 August 2024
Accepted 17 October 2025

KEYWORDS
Decision optimisation; stochastic demand; reinforcement learning; Markov decision processes; explainable artificial intelligence; heuristics generation

1. Introduction

The advent of digitalisation and automation is transforming the landscape of business operations. These technologies are enabling the optimisation of processes to unprecedented levels, enhancing throughput and output (Vaidya et al. 2018). To remain competitive in this rapidly evolving environment, it is essential to prioritise process improvement while maintaining product quality (de et al. 2022). The optimisation of

CONTACT Andreas Körner andreas.koerner@tuwien.ac.at Institute of Analysis and Scientific Computing, TU Wien, Vienna, Austria

decision-making represents a crucial aspect of this process. There are different approaches to finding solutions, but one widely accepted approach is the use of heuristics. They have the advantage of being fast, transparent and cheap to use once the rule-set is defined (Deb and Gupta 2023). They are transparent, so it is also possible to evaluate their correctness. However, these benefits come at the cost of loss of generality and adaptivity. This is due to the process of generating a new heuristic, which is often done manually, requiring expert knowledge in the specific domain (Ridha 2015).

Automated approaches have been developed or adapted to replace traditional heuristic generation, thereby reducing the need for expert knowledge in each specific case (Deb and Gupta 2023). The simplest approaches are trial and error, or brute force, respectively. By defining the different dimensions for decision making, each combination is tested on a simulation. The best combination is then selected. A smarter approach is the use of metaheuristics, which adapt natural behaviour to browse the dimension space. A popular metaheuristic is the genetic algorithm (Nguyen et al. 2017). Its approach is to utilise machine learning algorithms to learn directly from a problem simulation the optimal behavioural strategy. The advantage of this approach is that it produces an automated and adaptive rule set, resulting in a nearly optimal solution. However, this adaptability comes with the drawback of being non-transparent and resource intensive. For more complex algorithms, it is also necessary to adjust the problem-specific hyperparameters.

It is clear that neither the manual nor the automatic approach is the perfect solution for decision optimisation. The aim is to combine all the benefits while minimising disadvantages and avoiding significant losses in potential optimisation. We achieve this by combining the automation and optimality of Machine Learning (ML) approaches with the benefits of heuristics. This eliminates the need for expert knowledge and focuses on data-driven information.

The methods introduced in this paper provide a systematic way to address problems that are modelled as Markov Decision Processes (MDPs). We use the characteristics of such a simulation to optimise the problem using machine learning, resulting in a non-transparent rule set. From this, we extract a classical heuristic, which is transparent, simple and can be validated.

One approach in the area of ML that has gained popularity in recent years for solving decision problems is the use of Reinforcement Learning (RL). For restrictive environments such as the Atari games (Mnih et al. 2013), it has been impressively demonstrated that RL and the use of Deep Neural Networks can achieve high performance results for sequential decision processes. Many methodological improvements (Hessel et al. 2018) have further increased the performance and the range of applications. Its use as an optimisation method for real-world problems has also been demonstrated in several ways (Gosavi 2015; Panzer and Bender 2022). In addition, industrial problems are often solved using Deep Reinforcement Learning (DRL) approaches. In recent years, several scheduling problems, where discrete heuristics couldn't solve the problem, have been optimised using Deep Q-Network (DQN) agents (Günther et al. 2016; Chen et al. 2019; Xie et al. 2019; Hubbs et al. 2020; Lee et al. 2020; Zhou et al. 2021). This makes DRL a suitable method for finding optimal solutions to a wide range of problems without the need for expert knowledge.

In order to extract a transparent heuristic from a non-transparent RL algorithm, it is necessary to represent the correlations between state information and the decisions that have been made. For this purpose, decision trees are an appropriate approach, as they are capable of extracting rule sets from data. The algorithm of Bastani et al. (2018), called Verifiability *via* Iterative Policy Extraction (VIPER), trains a decision tree on previously trained policies, the oracle. The Q-DAGGER algorithm extends the Dataset Aggregation (DAGGER) algorithm of Ross et al. (2011), which is then utilised to extract a decision tree policy. The extracted policy is shown to be verifiable in terms of correctness, stability and robustness. This verification is performed on the Atari Pong and Cart-Pole environments (Barto et al. 1983; Bellemare et al. 2013). These environments are well-known toy problems and are used in the machine learning community as benchmark simulations for novel RL algorithms.

Costa et al. (2024) introduce a novel approach of an ensemble algorithm for decision trees to directly train an explicable policy. The decision trees are augmented by several tree operations such as pruning, truncation, leaf or threshold modification. To exploit the power of imitation learning, three different initialisation approaches are presented. On top of a randomly initialised tree, a decision tree learned from an uninterpretable model, such as DRL, can be used. A novel pruning approach, reward-pruning, is introduced, which can also be applied to the initialisation trees. The training algorithm is evaluated on the benchmark environments Cart-Pole, Lunar Lander and Maize Fertilisation (Barto et al. 1983; Gadgil et al. 2020; Gautron et al. 2022). The best models for these environments outperformed other decision trees for RL, while still being simple and interpretable.

Using a Non-linear Decision Tree (NDT), Dhebar et al. introduce a two-step approach for extracting a policy and a heuristic from an uninterpretable model (Dhebar et al. 2024). For the initial training, an NDT is trained on a (state, action) pair dataset collected from the simulation run by the uninterpretable model. A recursive two-stage evolutionary algorithm is used for this training step. In the next step, the pruned NDT is re-optimised directly on the environment. The approach is successfully evaluated on the Cart-Pole, Car-Following, Mountain-Car and Lunar-Lander environments (Barto et al. 1983; Moore 1990; Nageshrao et al. 2019; Gadgil et al. 2020). Due to the non-linearity of the decision trees, more interpretable work needs to be done to understand the resulting policy. Furthermore, the evaluation environments have been restricted to small dimensional state and action spaces.

State-of-the-art heuristic generation methods often use genetic programming, which has been successfully applied across various combinatorial optimisation problems such as bin packing or scheduling (Burke et al. 2019). However, this approach comes with disadvantages such as difficulty in generalisation, limited interpretability, overfitting and sensitivity to stochastic environments (Bhattacharya and Nath 2001; Žegklitz and Pošík 2015). Another way to obtain heuristics is to create them manually, a process which often requires domain-specific knowledge. However, generating heuristics based on data-driven information eliminates the need for expert knowledge or labour-intensive manual processes. Imitation learning involves building a rule-based decision tree that contains all the information necessary to extract a heuristic. A survey found that imitation learning has emerged as a valid and increasingly

essential approach to programming intelligent behaviour (Zare et al. 2024). Therefore, we expect to avoid the issues of genetic programming by using imitation learning.

In this work, we present a framework for generating heuristics tailored to complex and stochastic environments. We train a DQN model using an LDP simulation in the form of an MDP. DQN models have proven to be highly effective in solving complex problems (Terven 2025). After the model has been successfully trained, we use imitation learning with the VIPER algorithm to extract a comprehensible policy in the form of a decision tree from the trained DQN model (Bastani et al. 2018). This tree is then used to generate the heuristic. During this process, we utilise the pruning functionality of decision trees to simplify the rule set. This results in a more generalised and robust heuristic that is more resilient to stochasticity in the environment. We compare our methodology for generating heuristics with an incremental approach, in which we manually and iteratively refine existing reasonable heuristics in this field using simulation-based optimisation. This process aims to mimic the behaviour of experts in improving existing heuristics.

We evaluate our general method and demonstrate its potential for optimality by conducting a case study on a demanding example with three challenging scenarios. This case study is conducted on an on-demand manufacturing problem simulation with an attached job-shop. It represents an optimal choice for a study because it offers a good compromise between complexity and intuition. The complexity stems from a high degree of variance and randomness, which is hard to automate. In comparison, these variants happen in blocks, which is easy for a human to understand and extract a meaningful heuristic from. We will refer to it by Logistics Decision Problem (LDP).

This paper is structured as follows. In the first section, we present the methods that we use in our methodology. We formally define and classify heuristics in the area of optimisation problems. Then, we describe the method of formulating a decision problem as an MDP and the RL algorithm we chose to optimise it. Finally, we explain how we will use imitation learning from a trained RL policy to build the decision tree and extract the heuristics.

The next section explains and defines the LDP that we have selected for our case study. We show how the mathematical problem formulation is adapted to an MDP simulation. We then set out the three scenarios we have chosen for our illustrative examples and describe how we have generated an expert heuristic manually.

We also highlight the problems that arise with normal heuristics in situations of high variance and explain why a stepwise approach is essential.

In the results, we apply our method to the formulated scenarios. We consider the training of the model, the extraction of the policies and the generation of heuristics. We then benchmark our heuristic with the trained policy as well as with expert heuristics. Furthermore, we demonstrate the derived decision tree and discuss its explicability. It is shown that the extracted heuristic consistently outperforms expert heuristics over multiple test runs and matches the performance of our trained DQN model. This demonstrates that a superior heuristic can be extracted with minimal expert knowledge.

2. Method

In this section, the individual building blocks of our proposed method are described in detail. We explain how a problem formulation is adapted and how a transparent heuristic is created using the described techniques. The dependencies between the components are also outlined.

The decision-making problem is modelled using the Markov Decision Processes framework, which enables the application of DRL. The transition probabilities of the states are not explicitly known, but their expected value can be approximated using simulation. The VIPER algorithm then enables the extraction of an interpretable strategy for solving the decision problem.

2.1. Heuristics for logistics decision problems

In the field of psychology, heuristics are an approach to decision-making in complex situations without having to process all the information available. This also includes decision-making under time pressure (Raue and Scholl 2018). When using machines, this approach is also often used to achieve reasonable results with relatively simple rules to implement. In particular, these methods should be used when dealing with problems that cannot be formulated in an analytical mathematical form or cannot be solved efficiently by optimisation solvers.

Heuristic algorithms are classified as approximate optimisation algorithms and can be further categorised into problem-specific heuristics and metaheuristics. Metaheuristics are currently the preferred approach given the increasing size and complexity of optimisation problems. These algorithms are methods for finding approximate solutions to a wide range of optimisation problems. They for example can automatically find a near-optimal combination of decisions by defining the possible values to select as a multidimensional real space. This solution space is then traversed and tested, utilising natural processes as a model, such as ant colonies or the gravitational force of planets. However, these metaheuristics do not return a rule set to be followed. Rather, they solve the problem provided *via* a simulation. This lack of generality prevents the solution from being applied to similar but different problem adaptations (Dragan et al. 2020). In such cases, the use of problem-specific heuristics becomes pertinent.

2.2. Markov decision models and reinforcement learning

Decision processes can often be represented mathematically as Markov Decision Processes. We choose this approach for our study. In an MDP, an agent interacts with an environment iteratively. After the agent makes a decision in an initial state, the environment responds with a new state s' and a scalar reward R. The agent then makes another decision and the cycle starts again. This is repeated until a defined terminal state is reached or a specified time has elapsed in a possible infinite run. Solving this system normally means to find some strategy how to maximise the cumulative reward, which is called (episodic) return. A significant limitation of the

model is the Markov property. All the information about the system must be encoded in the respective last state before an action is performed.

Formally, a finite MDP consists of a set of states $\mathcal{S}$ and actions $\mathcal{A}$, with $|\mathcal{S}| < \infty$ and $|\mathcal{A}| < \infty$, as well as transition probabilities

$$p(s', r|s, a) := \mathbb{P}(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a), \tag{1}$$

where S_t, A_t and R_t represent the state, the action and the reward received at time t (Puterman 1994). The agent's goal is therefore to maximise the return at time t

$$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T, \tag{2}$$

with the discount factor $\gamma \leq 1$ and $t \leq T$ as the last time step of the simulation. Since we are dealing with an episodic task, it is assumed that T is certainly a finite random variable.

We denote the value function as

$$v_\pi(s) := \mathbb{E}_\pi(G_t|S_t = s), \tag{3}$$

which measures the expected return of a state s, assuming that the agent follows a fixed policy

$$\pi(s|a) = \Pr(a_t = a, s_t = s) \tag{4}$$

. The action value function is defined analogously as

$$q_\pi(s, a) := \mathbb{E}_\pi(G_t|S_t = s, A_t = a). \tag{5}$$

If the optimal action value function $q_*(s, a) := \max_\pi q_\pi(s, a)$ is known, selecting the maximiser chooses

$$a^*(s) = \arg\max_{a \in \mathcal{A}} q(s, a) \tag{6}$$

in the state s an optimal guideline. Various methods are available to approximate $q_*(s, a)$ directly, such as dynamic programming, Monte Carlo methods or Q-learning (Sutton and Barto 2018). However, these only work in small state spaces and are therefore rarely applicable in practice.

At the cost of losing the convergence guarantee of improving the policy, many of these algorithms can be extended to large state spaces using function approximations. For example, parameterising an approximate action value function with Deep Neural Networks and performing gradient descent on the Q-learning error has proven successful in a variety of environments.

To extract a working policy from our simulation model, we choose the Deep-Q-Network algorithm referred to above. DQN combines the classical algorithm of Q-learning with a state-action approximation *via* Deep Neural Networks. It has been shown that this combination leads to solution strategies for complex tasks, even in

high-dimensional state spaces. The main extensions of the algorithm compared to state-of-the-art approaches at the time are the application of 3 techniques:

DQN uses a Deep Neural Network to approximate the Q-function. The network operates by taking the current state as input and generating Q-values for all possible actions. The network learns by minimising the mean square error between the predicted Q-values and the Q-values calculated from the Bellman equation. The Bellman equation establishes a recursive relationship, whereby the target Q-value is defined as the immediate reward, in addition to the discounted maximum Q-value of the subsequent state. By iteratively updating the network parameters in order to minimise this error, the DQN progressively improves its estimates of the true Q-values and thereby learns an effective policy for decision-making in complex environments.

To reduce the correlation between successive data and increase the stability of the learning process, DQN uses an experience memory (replay memory). Experiences (consisting of state, action, reward and next state) are stored in this memory and randomly selected for training the network. This reduces bias and variance in training by decorrelating sequential experiences. Furthermore, this makes learning more data efficient as each experience is reusable. The replay buffer also stabilises the training as forgetting is prevented by a diverse set of experiences for updates.

This also helps to prevent early convergence by reducing the correlation between concurrent training updates and promoting diversity. However, to reliably prevent early convergence on a local optimum due to the increased manifold of the search space, other approaches are also applied. Dropout, where a percentage of nodes are masked for the next layer, allows for a higher degree of generalisation, rather than focusing on a single aspect too early and heavily. Another RL-specific approach is the use of exploration. By selecting suboptimal actions during training, new decision paths can be explored to identify more effective strategies.

Another key concept is the Target Network, which is a secondary neural network that mirrors the main Q-Network, but with delayed regular updates. Unlike the main Q-Network, which weights are continuously updated during training, the Target Network remains fixed for a predefined number of steps. This avoids instabilities due to rapidly shifting targets, thus making the training more stable and reliable. Furthermore, by maintaining stable target values over short periods, the learning algorithm prevents the network from entering harmful feedback loops. These loops can cause the network to attempt to chase its own moving estimates. After a certain number of iterations, the weights of the Target Network become synchronised with those of the main Q-Network. This technique significantly reduces oscillations and divergence in the Q-value estimates, thereby increasing the stability of the learning process.

These enhancements led to improvements in the stability and efficiency of the learning process over some straight-forward approach *via* a naive neural network approximation.

2.3. Policy extraction

Interpreting trained policies is a major research topic in both supervised and Reinforcement Learning. However, the sequential nature of RL makes understanding trained policies even more challenging than it is for supervised learning models.

Decision trees, which are used in supervised learning, are inherently descriptive, as they refine decisions based on feature values (Jijo and Abdulazeez 2021). This results in a transparent policy based on multiple binary decisions that culminate in a single, unambiguous outcome. Bastani et al. (2018) exploit this property in their VIPER algorithm.

An initial training set for a decision tree classifier is aggregated by running the simulation on the optimal RL policy. In contrast to its predecessor Q-DAGGER (Ross et al. 2011) the importance of a state decision is also included in the policy generation. This is achieved by weighting each state by its maximal difference of q-values. This value indicates how much weight the decision at this time step has on the progressing episode. After an initial decision tree is trained on this weighted state-action data, it is used to run new episodes, revealing unseen states that are important to the tree's decisions. The training data is incrementally updated with the new states obtained by the episode traversing and combined with the corresponding actions of the optimal RL policy. This ensures a targeted training of emerging states.

In contrast to the VIPER algorithm, we train the decision tree directly using the maximum difference in q-values for each state. Using the q-value difference as the sample weight

$$w(s) = v_\pi(s) - \min_{a \in A} q_\pi(s, a) \tag{7}$$

allows the decision tree to compute a higher impurity value for higher weighted samples, increasing the chance of splitting the node and therefore allows omitting the resampling step of the VIPER algorithm. The q-values are not constrained to an interval and can therefore produce large weights. To compute the entropy of the nodes, the weights must be normalised. This is achieved by constraining them to the interval $[0, 1]$ with

$$w(s) = \frac{w(s) - \min_{\zeta \in S}(w(\zeta))}{\max_{\zeta \in S}(w(\zeta)) - \min_{\zeta \in S}(w(\zeta))}. \tag{8}$$

The entropy

$$E(S) = -\log_2\left(\sum_{s \in S} w(s)\right) \sum_{s \in S} w(s) \tag{9}$$

represents the information value of a set of states, which is used to calculate the impurity of policy action selections in nodes. The impurity

$$I(N) = E(\mathcal{S}^{(N)}) - \sum_{k=0}^{|\mathcal{A}|} E(\mathcal{S}_k^{(N)}) \tag{10}$$

is computed for each leaf node of the tree, where $\mathcal{S}^{(N)}$ are the states with the correct feature values for the constraints leading to that node, and $\mathcal{S}_k^{(N)}$ are the states with action k selected by the RL policy. The node with the highest purity gain obtained by the difference of a node's impurity and of a new split combination is selected for the split.

In the case of high variance in episode behaviour, it is important to generate sufficient training data for the imitation learning. To achieve this, the algorithm is parameterised by training episodes rather than by the number of trajectories. This ensures that all states influenced by previous states are included in the training data. Furthermore, the resulting decision tree is tested on at least the same number of test episodes to provide reliable mean rewards and standard deviation.

2.4. Heuristic generation

Having built a reasonable decision tree from training data, the process of heuristic generation can now be started. The policy extraction method generates a highly complex decision tree with minimal impurities. Extracting a heuristic from this data is not straightforward, due to its complexity. To solve this problem, the pruning functionality of decision trees can be utilised. It retains the most important rules while omitting minor, nuanced decisions. It compares the impurity of subtree roots to a defined threshold. If it exceeds it, the purity is reduced by merging the subtree into one leaf. This allows for better generalisation of the tree while simultaneously simplifying the rule set.

To control the pruning efficiency, the purity threshold is defined. Based on this parameter, each subtree with a higher purity is merged. Therefore, the parameter can be used to decide on the complexity of the resulting heuristic. The rule set can be directly derived from this pruned decision tree. This more generalised rule set can achieve in average higher rewards for high variance episodes than the more specialised policy and removes the machine learning model dependencies to benefit from a discrete decision. The structure of decision trees allows the time-dependent dynamics of highly stochastic environments to be reflected by adding nodes that act based on simulation time. It also enables important edge cases to be integrated by making the tree as deep as necessary and adding highly specific decisions. This level of detail for edge cases is adjusted by pruning.

In addition, key state dimensions and values can be directly extracted to gain a better understanding of the internal behaviour of the simulation.

3. Case study of logistics decision problem

In this section, we will demonstrate the entire process, from defining a problem to generating a reasonable heuristic based on expert knowledge. We intentionally choose a simulation model that is both realistic and easily understandable without requiring extensive specialist knowledge. To meet these requirements, we decided for a detailed simulation of a sales point with an attached production process in the form of a job-shop. In this example, various products need different machines for a specified duration. The core decision problem is to initiate the production of these products to meet a stochastic demand during peak times. The demonstration is much more complex than toy-problems and should show the applicability of the method on real-world problems.

First, we will describe the generic simulation model and then we will move on to specific scenarios which cover our requirements.

3.1. Problem formulation

The simulation uses a variety of products $\mathcal{P} \subset \mathbb{N}$ where each product $p \in P$ belongs to one of $J \subset \mathbb{N}$ categories $\mathcal{P}_i \subset \mathcal{P}, i \in J$. Each category differs in its unique sequence and duration of production steps and has a specified production cost c_p. The production steps are carried out on a set of different machines $\mathcal{M} \subset \mathbb{N}$. The predetermined sequence is defined by $P_{i,j} = m$ for production step j of product category i with machine m. Upon completion of the last machine, products are stored in the inventory I. Products can expire, when they are stored for more than E amount of time steps in the inventory, therefore the inventory is defined as

$$I(t, i) = \{p \in \mathcal{P}_i : f_p + E > t\},$$

where f_p is the completion time of product $p \in \mathcal{P}$.

Until a specified end time T, customers $\mathcal{C}$ arrive with a specific demand of multiple product categories $D_{c,t} \in \mathbb{N}^J, t < T$. The value for each dimension defines the number of products of a specific category. The demand $D_{c,t}$ can be satisfied if enough products of each category are present in the inventory,

$$\forall i \in J : |I(t, i)| > \pi_i(D_{c,t}),$$

with π_i being the projection on dimension i. If this holds, the products are sold and the revenue for this demand is positive and defined by the selling price $R_i, i \in J$, of the product categories,

$$r_c = \sum_{i \in J} R_i \cdot \pi_i(D_{c,t}).$$

If not enough products are in the inventory at this time step, the revenue for this customer is zero, $r_c = 0$.

The aim of the problem formulation is to maximise the revenue while minimising the production cost,

$$\max_{\mathcal{P}} \sum_{c \in \mathcal{C}} r_c - \sum_{p \in P} c_p.$$

3.2. Simulation model as Markov decision process

Based on this problem formulation, a simulation model was implemented as an MDP. The RL agent has access to various information channels, including inventory levels and the state of the production process. The agent can make decisions, such as ordering new products every time step Δt.

The factory is modelled as a MDP, with various variants evaluated for reward assignment. In these variants, successful sales are rewarded, while unproductive times or undesirable production patterns are penalised. This evaluation mechanisms is used to train the agent and optimise its decision-making.

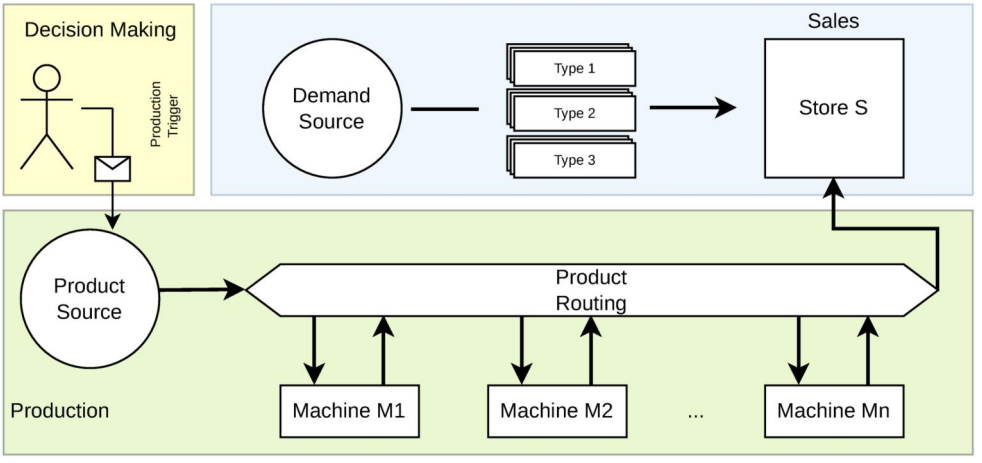


Figure 1. The production of a specific product is triggered by the decision of an RL agent or a production heuristic. A demand can be fulfilled if the desired products are available in the store. This generates a positive reward for the Markov decision model, which leads to a better policy in adaptive systems.

The model and the relationship between demand, sales and production are illustrated in Figure 1.

To meet the problem formulation, a certain number $\sum_i d_{\text{prod},i}$ of requested demand $d_{\text{prod},i}$ for each single product i are generated over a specified total runtime of t time units. These demands include a randomly generated number of products that are requested at the times $t_1 \ldots t_n$ within one episode. The points in time are defined by a changing rate r_n. These variations are configured to create several phases of intensive and less intensive demand. To generate the randomness of arrivals, a given vector d_{gen} is linearly scaled to the total runtime and normalised. Product specific demand $d_{\text{prod},i}$ for all products i is generated the same way and forms the product specific demand per arrival. Together, this constitutes the probability density of the requests occurring. To illustrate this, we refer to Figure 2. In addition, the exact quantity of products demanded at times t_i for $i \in 1, \ldots, n$ is also generated with weightings that differ slightly between the products.

In the sales process model, demand is processed linearly. Products that are available at these times are sold at a price $R_{\text{sell},k}$ for product k. Furthermore, demands are also 'partially' satisfied. If the quantity of individual products in a demand cannot be fully satisfied by existing products, parts of the demand are covered by existing products. The transport and handling time from the production site is described by the parameter t_{trans}. Randomness is generated across various runs using probability distributions based on the parameters and distributions shown in Figure 3.

In addition, some simplifications are made to the model. The storage space for products is technically unlimited, but is not used in the scenarios considered, except during the learning phase, in which its use is also penalised.

Each product is modelled by a recipe that includes the sequence of the required machines and their processing time. Each machine has an input queue with a length of l_{in} and an output queue with a length of l_{out}. If the output queue is full, no further

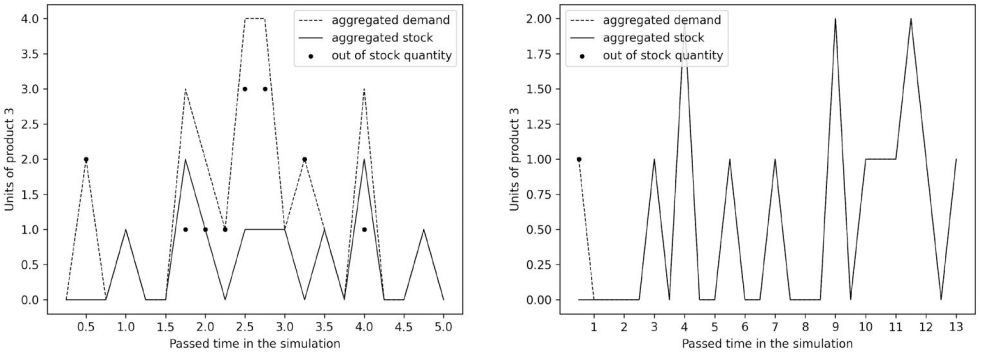


Figure 2. These figures show the history of the aggregated demand, the aggregated stock levels and out of stock values of product 3 in a single episode of the short Scenario A (2a) and the long Scenario B (2b). The out of stock values are only calculated if an out of stock situation occurs in an interval. In the Scenario B, there is almost no out of stock situation, as demand is congruent with inventory levels. Furthermore, stockpiling does not happen.

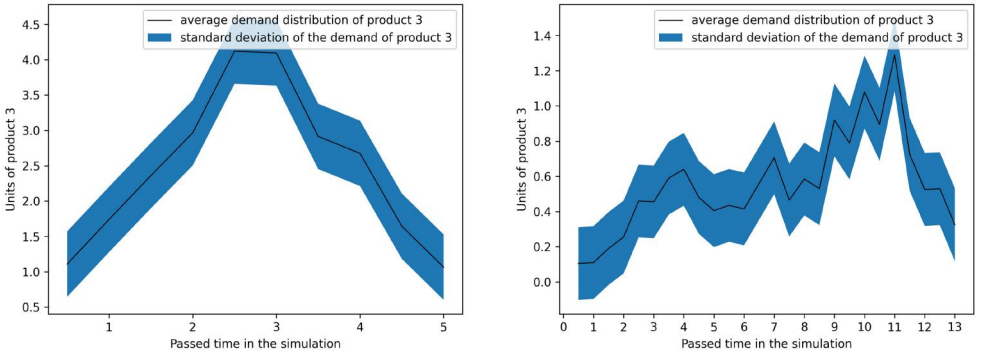


Figure 3. These figures show the aggregated demand distribution of the short (Scenario A) in (3a) and long configuration (Scenario B) in (3b)of our model. Aggregation takes place in 30-minute intervals. The demand waves of both configurations are clearly visible.

production can take place in this machine. Each machine is connected to a router which receives the job and moves it to the next required station. At the end of the production line there is a warehouse that forwards the finished products to the sales department individually with a delay of t_{trans} as soon as they are available. In the examples evaluated, no machine down-times or maintenance breaks are modelled. In addition, there is only one machine of each type per production line, so that production is not carried out simultaneously with the same type.

In the case of complex scenarios, the mathematical requirements of a Markov Decision Process can often only be approximated. The state vector, which the agent receives before a new action can be performed, includes the simulation time, the current stock levels of the products, the current utilisation of the machines and the number of products in current production. We are aware that the Markov property cannot be fully guaranteed by our selection of state space variables. Respecting this property would increase the state space enormously and this must then be simplified again by a suitable approximation on the learning algorithm side. Since neural

network approximations currently lack convergence guarantees, only the theoretical relationship between the model and the learning algorithm can be adjusted. Therefore, we choose the pragmatic approach and use information that is also available to a purely heuristic-based system. The available actions in our case study cover the ordering of new products. The RL agent can trigger the production of a specific product with the action A_i for product i and A_0 for no product. This is repeated every time step Δt. The simulation model accepts the decision and starts the defined subsequent processes. In our reward model, the different products are configured with a sales price $R_{\text{sell},i}$ and a negative production price $P_{\text{prod},i}$ for each product i. A one-off penalty per product of -10 is applied, if the storage space of more than 30 is exceeded. Additionally, a reduction in the sales price by a factor of 0.8 was added if the product is stored for longer than 1.0 time units.

The simulation-model was implemented using an event-driven simulation paradigm, where the simulation is controlled step-by-step from the outside by executing the actions of the RL agent. To enable a precise evaluation of the system dynamics and the interactions between the various components, the model was implemented as a compound of process-orientated state machines. This allows complex interactions and state transitions to be captured precisely.

3.3. Generation method for stochastic demand

The application examples considered are characterised by a very strong dependence on randomness. This section discusses the generation of demand, which is essential for the dynamic behaviour of the simulation.

The demand for products is defined per product group by a weight vector d_p for product p and by the requested quantity m of the product. The values used can be found in Table 1. The arrival rates of customers are also relevant for determining when the products are needed.

The specific times and quantities are determined using the following multi-stage process.

1. The demand vectors d_p are linearly interpolated over the time period T. The length of the vector is not related to T, but this enables the most accurate control possible. The linear interpolations are normalised to the area 1 for the available time period and then used as the probability density for generation.
2. The arrival times of the customers are also linearly interpolated using the vector AR and transformed into a density function. P time points are then sampled from this density function. These represent the arrival times of customers in the stores.
3. Based on the arrival times, the number of products is now randomly generated from the density functions of the product groups. For each product p, the $d_{prod,p}$ pieces are distributed according to the density function to the arrival times of the customers.
4. As a result, a certain quantity of products is required at each arrival time.

Exemplary density functions for three vectors $v1, v2, v3$ is illustrated in Figure 4.

Table 1. The parameters for the simulation models.

Parameter	Description	Scenario A	Scenario B	Scenario C
t	Time of an episode	5	14	14
Δt	Action resolution	0.075	0.1	0.1
d_{gen}	Arrival vector	[2,3,4,3,2]	[1,1,2,1,2,4,4,2,2,1]	[1,1,2,1,2,4,4,2,2,1]
d_1	Demand vector prod. 1	[2,3,4,3,2]	[1,2,2,2,2,2,2,2,2,1]	[1,1,2,1,2,4,4,2,2,1]
d_2	Demand vector prod. 2	[2,3,4,3,2]	[1,1,3,4,3,1,1,3,2,1]	[1,1,2,1,2,4,4,2,2,1]
d_3	Demand vector prod. 3	[5,6,7,8,9]	[1,2,3,4,2,1,2,4,2,1]	[1,1,2,1,2,4,4,2,2,1]
$d_{prod}1$	Demand count prod. 1	15	40	20
$d_{prod}2$	Demand count prod. 2	15	25	15
$d_{prod}3$	Demand count prod. 3	25	15	45
t_{trans}	Transport time to shop	0.2	0.2	0.2
$prod_{route1}$	Prod. 1 machine-times	TypeA: 0.08 TypeB: 0.1	TypeA: 0.05 TypeB: 0.05	TypeA: 0.05 TypeB: 0.05
$prod_{route2}$	Prod. 2 machine-times	TypeA: 0.08 TypeB: 0.1 TypeD: 0.05	TypeA: 0.05 TypeB: 0.05 TypeD: 0.025	TypeA: 0.05 TypeB: 0.05 TypeD: 0.025
$prod_{route3}$	Prod. 3 machine-times	TypeA: 0.08 TypeC: 0.3 TypeD: 0.05	TypeA: 0.05 TypeC: 0.2 TypeD: 0.025	TypeA: 0.05 TypeC: 0.2 TypeD: 0.025
l_{in}	Maximum length of input queue of machines	10	10	10
l_{out}	Maximum length of output queue of machines	10	10	10
R_{sell1}	Selling price prod. 1	1	1	1
R_{sell2}	Selling price prod. 2	2	2	2
R_{sell3}	Selling price prod. 3	4	4	4
R_{prod1}	Production price prod. 1	−0.3	−0.3	−0.3
R_{prod2}	Production price prod. 2	−0.35	−0.35	−0.35
R_{prod3}	Production price prod. 3	−0.6	−0.6	−0.6
$t_{greeting}$	Store time greetings	0.007	0.007	0.007
$t_{checkout}$	Store time checkout	0.025	0.025	0.025

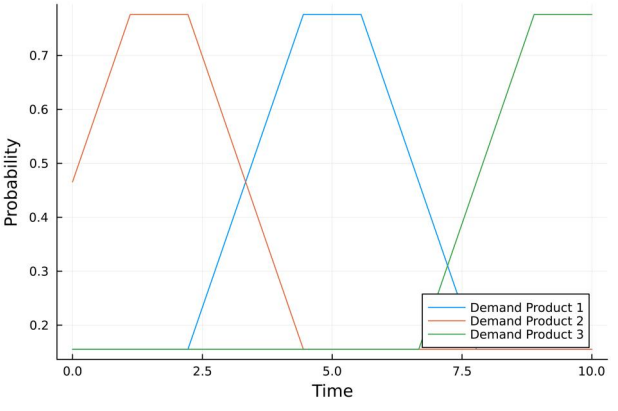


Figure 4. Demonstration of density function for demand generation with example vectors $v1 = [1, 1, 1, 3, 5, 5, 3, 1, 1, 1]$, $v2 = [3, 5, 5, 3, 1, 1, 1, 1, 1, 1]$, and $v3 = [1, 1, 1, 1, 1, 1, 1, 3, 5, 5]$.

The actual demand for products during a simulation can be seen in Figure 5. It can be seen that the density function controls the demand accordingly.

These examples are only intended to illustrate the process. The variables used in the examples can be found in Table 1 and create a much more complex stochastic

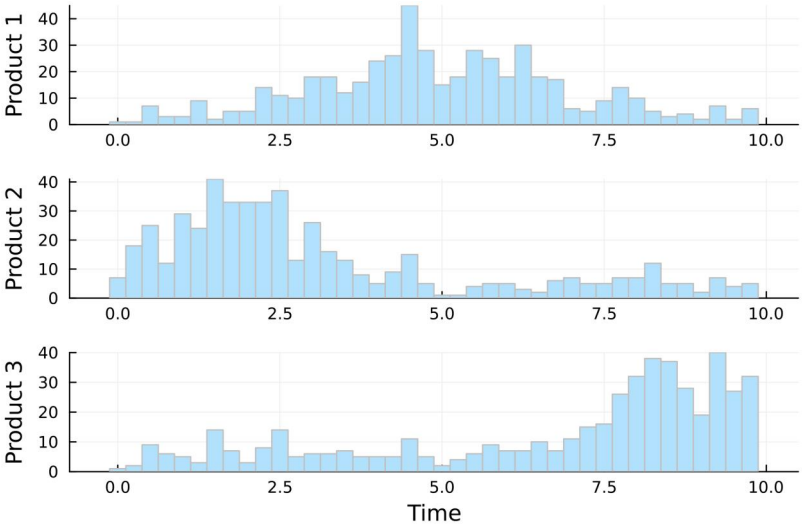


Figure 5. Demonstration of actual demand generation based on density function in simulation set up with three products.

Figure 6. Arrival times for demand vector $ar = [1, 10, 1, 1, 10, 1]$.

situation with a very large dispersion that is difficult to visualise. For illustration, arrival times for the vector $ar = [1, 10, 1, 1, 10, 1]$ are shown in Figure 6.

The demand of products was fixed in order to make the defined scenarios as controllable as possible. The number of products and the processing time, including stochastic fluctuations, are part of the scaling and adjustment to the desired boundary conditions. For a reasonable scaling, several variables must be coordinated with each other in order to reflect the credibility of a realistic scenario. The number of products is the variable that can be fixed most easily. We derive three different scenarios from this model setup, which we briefly motivate in the next two subsections. The associated parameters can be found in Table 1.

3.4. Description of Scenario A

The idea behind this model is to simulate a day of a small bakery that is solely responsible for producing and selling its products. The simulation lasts for five time units, to which we refer to as hours for improved interpretability and readability. During this time, production decisions must be made every 4.5 min. Three non-perishable products can be produced. As the shop is only open for five hours, we have decided that these products do not need to spoil. The product range contains one cheap product (Number 1) with a fast production time and two products (Numbers 2 and 3) with a longer production time and a higher selling price than the first one. To create a conflict of interest, the cheaper product shares some of the machines in the production process with the more expensive ones.

To model the demand, we simulate 50 customers coming into the shop with the intention of buying any number of any product. Over an episode their demand must accumulate to 15 units of product 1, 15 units of product 2 and 25 units of product 3. The demand is distributed so that each product has its peak in the third hour of the simulation. Figure 2a describes the production, demand and stock levels of product 3 in Scenario A. We have deliberately designed the demand so that not all demand can be met at the peak. This allows us to analyse, if and how the agent responds to peaks in demand.

3.5. Description of Scenario B and Scenario C

The longer model is designed to simulate a longer day in a similar shop to the bakery in the model above but with a total time span of 14 h and three perishable products. If a product stays in the shop for more than half an hour, its selling price is reduced by 10%. If the product is not sold for another 30 min, its selling price is reduced by a total of 20%. The idea behind this design choice is to simulate a shop that has a philosophy of selling only very fresh products.

In this model production decisions need to be made every 6 min. The product range is similar to Scenario A. However, the production times for all products are shorter than in the previous model. The aim of this adjustment is to be able to react more quickly to changes in demand and to avoid stockpiling, which is penalised due to perishing of products. Figure 2 shows this effect. The ratios between the production times of the three products remain the same.

In this longer scenario, 75 customers are simulated. They behave like the customers in the shorter model. However, over the course of an episode, their demand must accumulate to 40 units of product 1, 25 units of product 2 and 15 units of product 3. The demand patterns have been changed so that product 1 has a fairly constant demand over an episode, while the other products have two demand waves, one at the beginning of the simulation, and one towards the end. We expect to see a change in agent behaviour as the desired quantity of the most expensive product has been significantly reduced compared to the demand for the other products. Figure 2b describes the production, demand and inventory levels of product 3 in Scenario B.

Additionally, Scenario C is derived from Scenario B, with the only difference being the variation in product demand counts for each product, see Table 1. The purpose of this adaptation was to intentionally construct a simpler scenario, allowing us to evaluate how the method performs under reduced complexity and whether it produces a correspondingly simpler solution. Given that the demand for product 3 is highest it is also the most expensive one, we expect the solution to favour the production of product 3.

3.6. Generation of initial heuristics

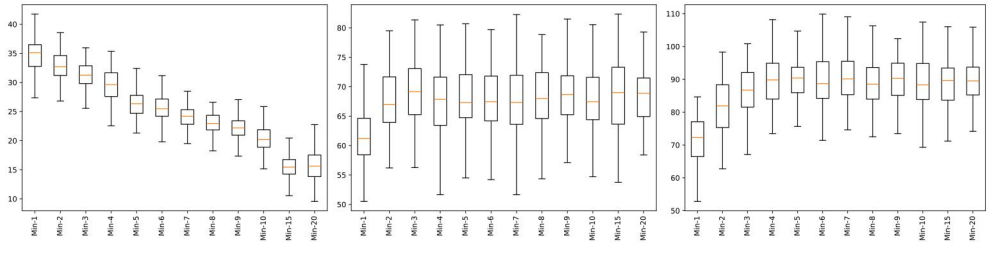
To showcase a typical heuristics generation, we want to identify a heuristic for our considered problem that can solve the underlying decision-making process. In a first

approximation, we define a threshold below which the production of the corresponding product is triggered.

This choice of a simple threshold-based heuristic is both operationally and economically justified. Threshold rules are widely used in inventory and production planning to create buffer stock and reduce the risk of stockouts under uncertain demand. They offer a good trade-off between performance and simplicity, keeping implementation costs low while maintaining responsiveness. Since the optimal threshold cannot be derived analytically due to the stochastic nature of the system, simulation-based evaluation provides a practical and efficient way to identify effective parameter values. As the heuristic is based on domain-specific knowledge and reflects widely adopted strategies in practice, it can be considered an effective substitute for expert behaviour under uncertainty. This makes the heuristic a robust and interpretable starting point for further refinement. The performance of the heuristics for the three simulation models is shown for different threshold values in Figure 7.

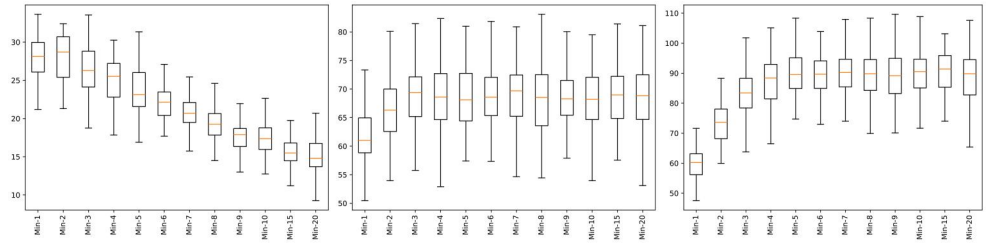
These heuristics serve as an initial basis for the examples analysed. However, a closer look at the underlying data shows that it is obviously not possible to satisfy demand at peak times, so the heuristic is too simple to account for lead times and fluctuating demand.

In a further iteration to improve the heuristics, we consider the fluctuations in demand over time for the individual products and dynamically adjust the threshold. In our scenarios, the times with higher demand can be covered by increased production in advance. The ideal point in time can be determined using classic simulation-based optimisation. We adjust the times accordingly and increase the production quantity between time steps 2 and 5 to the maximum capacities. As expected, this leads to an improvement in performance. As a possible further refinement, the expected profit per product could be included in product prioritisation. We consider this heuristic to be exhaustive because experts proactively plan based on anticipated demand fluctuations, rather than reacting to them. Dynamically adjusting the threshold based on expected demand over time reflects this forward-looking behaviour. Furthermore, iterative refinement using simulation-based optimisation reflects the strategy adjustments made by experts.



(a) Scenario A. (b) Scenario B. (c) Scenario C.

Figure 7. The return under different thresholds for the basic heuristics in the three considered scenarios. The term 'Min-x' refers to a policy that defines x as the lower threshold for a production start.



(a) Scenario A. (b) Scenario B. (c) Scenario C.

Figure 8. The return under different thresholds for the improved heuristics in the three considered scenarios. The term 'Min-*x*' refers to a policy that defines *x* as the lower threshold for a production start. Additionally, the requested demand was increased by 2 items in the time between 0 and 5.

Figure 8 shows the evaluation over 100 runs between the initial heuristic and improved one, we see that the proposed modification increases the total return accordingly and also reduces the standard deviation.

These two modifications and all future modifications require extensions of the models to include promising parameters, which can then be solved as part of parameter optimisation. This requires knowledge of the model to find the parameters.

4. Results

In this section, we will present the results of our method evaluated on a logistic decision problem. It is organised into two parts. First, we will discuss in detail, how the method can be applied on the problem. Resulting from this execution is a heuristic created by the decision tree. In the second part, we benchmark this extracted heuristic, to the problem specific expert heuristic.

4.1. Application of method on the example

To apply the method, we first present the training process used to optimise the decision problem with a learned policy. Then we will discuss the extraction of a decision tree and consequently a transparent heuristic that could be extracted from the learnt policy. Finally, we will examine this in the context of the simulation and illustrate the effects using an exemplary situation.

4.1.1. Learning

We use classic DRL with a slightly customised DQN algorithm to find the policy. It turned out that the high level of randomness leads to a very unstable learning behaviour. When fixing the randomness to a seed, we were able to observe a convergent behaviour in almost all cases. In order to calculate a useful policy for the stochastic case, hyperparameter tuning was performed mainly on learning rate and batch size. To achieve the desired stability, we slightly modified the DQN algorithm. Updates are withheld for 10 steps and only after that a gradient step is performed in the neural network. In our experience, this delay leads to a more stable behaviour at the expense of sampling efficiency. Since we are working with an existing simulation model,

Table 2. These hyperparameters lead to stable learning behaviour in the considered case-studies.

Net Config	Input	Output	Activation	Minibatch size	128
Layer 1	11	32	ReLu	Experience replay	10 000
Layer 2	32	32	ReLu	Learning rate	0.1
Layer 3	32	32	ReLu	Update frequency	50
Layer 4	32	4		Exploration	0.1
				Gamma	0.99

policy stability is prioritised. The complexity and capacity of the neural networks also had a major influence on the performance of the training. The parameters were applicable for all scenarios and can be found in Table 2.

Normalising the input accelerated learning but did not improve policy performance or convergence reliability. Figure 9 shows the learning progress of all scenarios. Convergence of the learning curve is clearly visible for all scenarios, as is the superiority of the approach compared to the heuristic built iteratively.

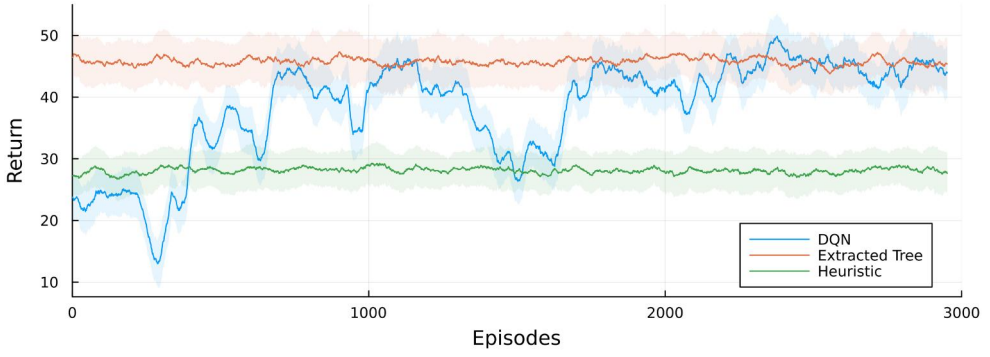
4.1.2. Heuristic generation

After training a DRL agent and identifying an optimal policy, we applied an iterative policy extraction method to derive interpretable heuristics in the form of decision trees. To account for stochasticity in the environment, we generated a dataset from 100 episodes, iteratively expanded it across ten runs, and selected the best-performing tree. A pruning similarity of 0.8 was chosen for Scenarios A and B to balance model simplicity and policy performance.

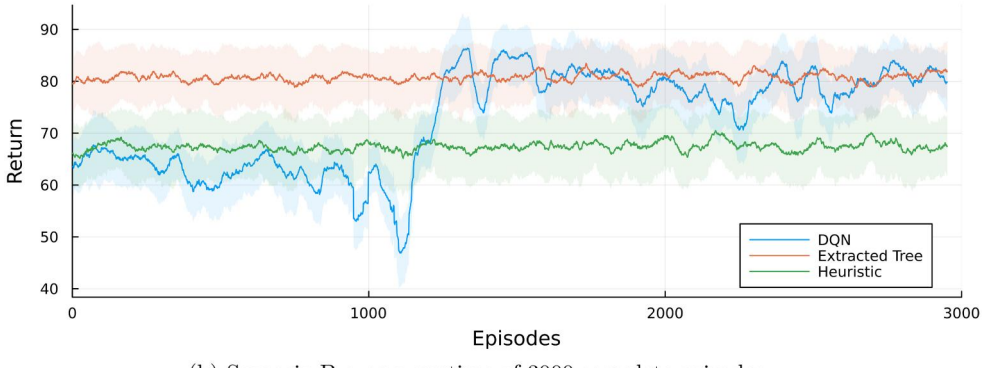
In Scenario A, the extracted decision tree (excerpt in Figure 10c) has a depth of 11 and comprises 62 specific actions following a sequence of decisions. The resulting heuristic achieves an average reward of 45.74, with a standard deviation of 4.75. The policy alternates between checking current production levels and simulation time. Products 2 and 3 are favoured in general, while product 1 is only produced during specific peak times. In this scenario, production stops when too many products of a specific type are being produced. The type and amount of product to be produced depends on the simulation time. This helps to keep the inventory within a reasonable range, since products do not perish and the simulation time is quite short.

In Scenario B, the extracted tree (excerpt in Figure 10a) uses all the provided actions to build a tree with a depth of 11, resulting in 48 constraints. This achieves a mean reward of 80.05, with a standard deviation of 5.48. The policy incorporates perishability by carefully balancing the production of product 3 with maintaining stock of products 1 and 2. The simulation starts with an inventory of product 3, which is then complemented by the other products once the product 3 store is filled or product 3 is currently being produced. The focus on product 3 during the earlier stages of the simulation is rechecked throughout the decision tree.

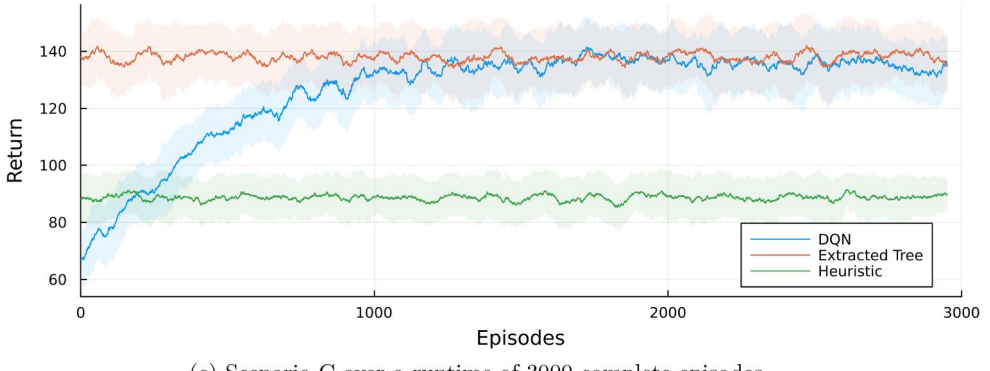
In Scenario C, which is derived from Scenario B with simplified demand dynamics, the extracted decision tree (Figure 10b) is the most compact. It uses three actions and three constraints across two features. Despite its simplicity, it achieves a mean reward of 138.04, with a standard deviation of 12.43. The heuristic primarily favours product 3 and dynamically adjusts production according to the inventory state of products 1 and 3.



(a) Scenario A over a runtime of 3000 complete episodes.



(b) Scenario B over a runtime of 3000 complete episodes.

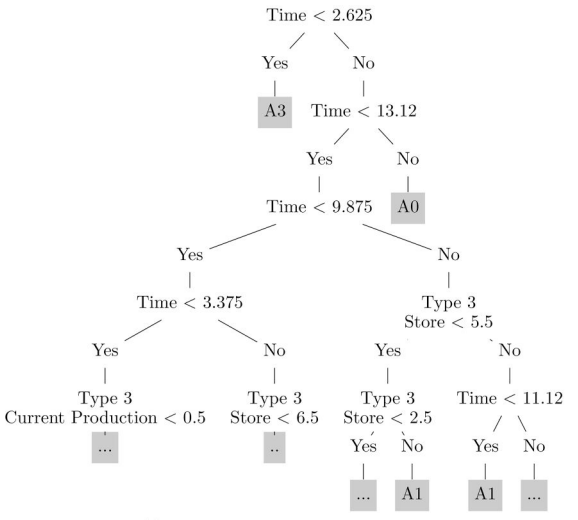


(c) Scenario C over a runtime of 3000 complete episodes.

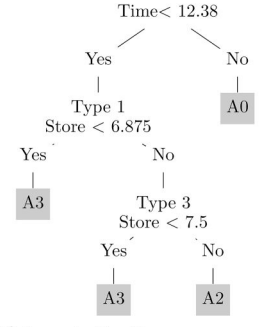
Figure 9. The trend of the episodic return including standard deviations over the learning process observed, along with the evaluation of the DQN policy, the extracted tree policy and the best improved heuristic.

The higher reward reflects the reduced complexity of the scenario and the efficiency of the extracted policy.

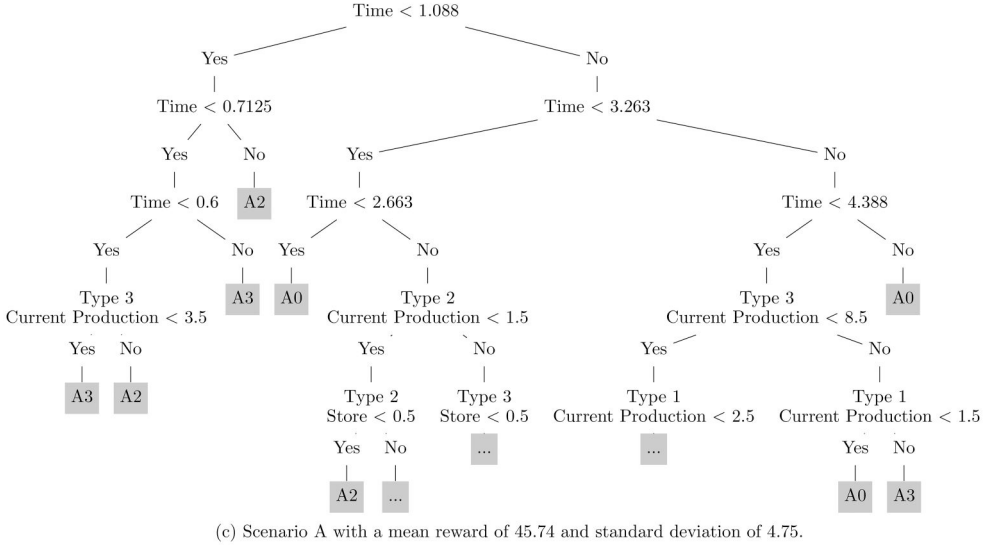
The heuristics extracted from all three scenarios demonstrate effective production scheduling, minimising overproduction while aligning with demand peaks and freshness constraints. Each decision tree reflects the key features of its respective scenario and adapts to perishability, production speed and demand structure. The resulting



(a) Scenario B with a mean reward of 80.05 and standard deviation of 5.48.



(b) Scenario C with a mean reward of 138.04 and standard deviation of 12.43.



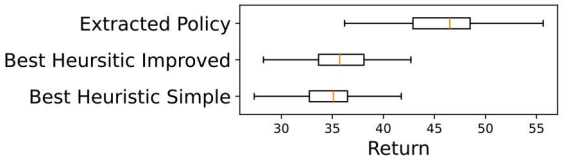
(c) Scenario A with a mean reward of 45.74 and standard deviation of 4.75.

Figure 10. The decision tree extracted by the iterative policy VIPER from the optimal policy of the DQN agent. The extracted heuristics mean reward is computed over 100 episodes. (a) Scenario B with a mean reward of 80.05 and standard deviation of 5.48. (b) Scenario C with a mean reward of 138.04 and standard deviation of 12.43. (c) Scenario A with a mean reward of 45.74 and standard deviation of 4.75.

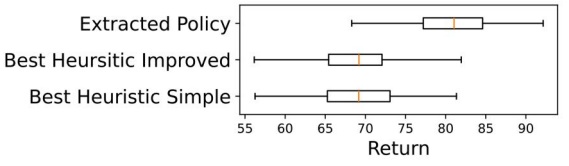
policies are interpretable, compact and capable of achieving strong performance under stochastic conditions.

4.2. Expert heuristic vs. extracted heuristic

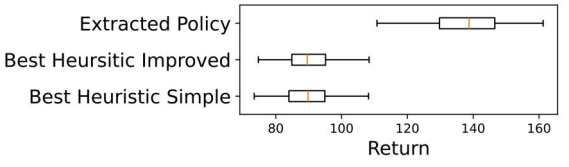
We evaluate the described method on the two defined examples over 1000 simulation runs. As a reference, we take the best of the expert heuristics found in section 3.6 and compare it with the extracted tree policy. The comparison is shown in Figure 11.



(a) Comparison of the extracted heuristics with the comparative heuristics in Scenario A.



(b) Comparison of the extracted heuristics with the comparative heuristics in Scenario B.



(c) Comparison of the extracted heuristics with the comparative heuristics in Scenario C.

Figure 11. This figure compares the best of the manually defined heuristics from section 2.4 with the extracted heuristics.

In both cases, the extracted heuristics are able to beat the expert heuristics. In the more complex example, it is easy to see that the interquartile range of the expert heuristic is rather large. We assume this is due to the higher complexity of the model. Even so, the extracted policy continues to perform robustly.

5. Discussion

In this paper, we propose a framework that integrates Policy Extraction, Reinforcement Learning, and Markov Decision Models to automatically derive effective heuristics. We successfully demonstrated the efficacy of this approach through a case study involving a logistics decision-making problem. This problem was chosen as it is a complex yet intuitively accessible example to illustrate the feasibility of this approach. The present study employs three scenarios of the problem to validate the efficacy and flexibility of the proposed approach in producing superior heuristics in different settings. To demonstrate the naive approach of creating heuristics based on domain-specific knowledge, we employed initial baseline heuristics commonly used in this field. To mimic expert behaviour in adapting a strategy, we then refined these heuristics iteratively using simulation-based optimisation to improve their performance and resilience to environmental stochasticity. In contrast, for our method neither domain-specific knowledge nor problem-specific customisation is needed. The

heuristic of our methodology was able to clearly outperform both the baseline and the improved heuristics. While it is evident that the naively created heuristics could be further improved with sufficient effort and innovation, such enhancements would contradict the fundamental purpose of heuristics—to be inexpensive and easily deployable. Furthermore, the study also revealed the limitations of relying on intuition to address the stochastic nature of various problems. In such cases, our method is clearly advantageous, as it automatically accounts for stochasticity when generating heuristics by extracting them from a trained DQN model, which is inherently suited well for handling stochasticity. Furthermore, the extracted heuristic avoids the black-box limitations of DQN models, making it comprehensible and, in our case, also superior to naive heuristics across all evaluated scenarios.

A number of challenges had to be addressed, including the optimisation of hyperparameters to ensure stable and reproducible learning behaviour, which required considerable effort. The training of the agents produced very different learning curves, which fortunately led to largely consistent results. Scenario C appears to be very easy to learn, while scenarios A and B are more difficult for the agent to learn due to the complexity of the policy. The instability of the learning curves is due to the learning algorithm. More than 3,000 episodes were simulated on a test basis, and the instability consistently disappeared after approximately 2,000 episodes. The shape of the learning curves and the associated difficulty are also consistent with the complexity of the decision trees. Moreover, the solution demonstrated the intrinsic constraints of the simulation model. Many potential actions within the simulation may not be essential for maximizing the reward, but may, in fact, impede it. It is evident that the behaviour of the trained agent can be restricted by additional constraints *via* reward engineering or more detailed mechanics within the model. Even without this modification, the method presented was able to extract an interpretable and superior strategy, which is difficult to find using classical methods.

Although the extracted policy's strategy initially seemed simple, it yielded a superior solution to that of expert-based, problem-specific heuristics in all three scenarios. One might argue that the scenarios were not complex enough. This may be true of Scenario A, which is shorter than Scenario B. However, we observed that adding Scenario C – which has the same settings as Scenario B but with a different demand distribution – resulted in a drastic reduction in decision tree complexity. This strongly suggests that Scenario C represents an edge case, while Scenario B is more complex. This is evidenced by the fact that its resulting decision tree is larger and incorporates more features than Scenario C's tree. These findings demonstrate that our method can effectively adapt to both edge-case and standard scenarios of an LDP, thereby underscoring the generalisability of our method as a robust framework.

The method has additional 'side effects' that may not be immediately apparent, but are nevertheless highly relevant for potential applications. The decision tree provides valuable insights into which features of the observation space are relevant for the underlying problem and how strong they influence the agent's decisions. Thus, our method effectively addresses a core challenge in the field of feature extraction. Moreover, our method is problem-independent, making it highly flexible and broadly applicable across different domains. As such, it enables a systematic evaluation of

feature selection in existing models. By analysing the structure of the resulting decision tree, one can assess whether specific features used by a model are meaningful or negligible. This paves the way for future research leveraging this method to perform in-depth diagnostics and improve interpretability in RL systems.

Despite the significant advancements in machine learning, the practical applications of RL have not met expectations, as evidenced by Paduraru et al. (2021). One potential explanation for this discrepancy is that the scientific benchmarks primarily encompass deterministic and predictable environments. Additionally, the practical implementation of RL in real-world scenarios is often challenging or even infeasible. It is challenging to implement superior, trained policies in the form of a non-transparent neural network without a clear understanding of the network's decision-making process under different conditions. The extraction of policies in the form of decision trees or heuristics enables the strategy to be checked and, in certain areas, implemented *via* classical controlling techniques.

The objective of further research is to improve the policy extraction method. The results obtained from decision trees are susceptible to high levels of variance when the training data is subject to minor alterations. Random forests, which are composed of multiple distinct and simpler decision trees that use a majority voting approach, are a state-of-the-art solution for tackling supervised learning problems and address this limitation. Our objective is to examine the impact of integrating random forests into our policy extraction algorithm, with the aim of accelerating the learning process of the decision tree and reducing the number of training iterations.

While our method demonstrates problem-independent applicability and strong interpretability benefits, further validation in larger and more complex simulations is necessary to fully ascertain its suitability for specialised heuristics and domain-specific applications. A potential avenue for further investigation within the presented application is to focus on the production mechanics itself. A multitude of functional and well-designed heuristics for the selection and timing of individual production steps exists. These could be improved through the application of machine learning and subsequent interpretation, potentially even without additional information dependency. It would be of interest to ascertain whether the complexity of the decision trees increases in proportion to the complexity of the simulation model. Such an investigation could provide valuable insights for further development in this area.

Disclosure statement

The authors report there are no competing interests to declare.

Use of generative AI

Generative AI was used to enhance the quality of the text. TU Wien offers members a pro licence for AI translation technology (deepl.com, version 06/2025) to produce high-quality language for research articles.

Funding

The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

ORCID

Andreas Körner http://orcid.org/0000-0001-7116-1707

Data availability statement

We are willing to make the underlying data publicly please contact us.

References

Barto AG, Sutton RS, Anderson CW. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans Syst, Man, Cybern. SMC-13(5):834–846. https://doi.org/10.1109/TSMC.1983.6313077

Bastani O, Pu Y, Solar-Lezama A 2018. Verifiable reinforcement learning via policy extraction. In: Bengio S, Wallach H, Larochelle H et al., editors. Advances in neural information processing systems. Vol. 31. Curran Associates.

Bellemare MG et al. 2013. The arcade learning environment: an evaluation platform for general agents. JAIR. 47:253–279. https://doi.org/10.1613/jair.3912

Bhattacharya M, Nath B 2001. Genetic programming: a review of some concerns. In: Alexandrov VN, Dongarra JJ, Juliano BA et al., editors. Computational science – ICCS 2001. Springer Berlin Heidelberg. p. 1031–1040.

Burke EK, Hyde MR, Kendall G et al. 2019. A classification of hyper-heuristic approaches: revisited. Springer International Publishing. p. 453–477. https://doi.org/10.1007/978-3-319-91086-4_14

Chen B et al. 2019. Improving cognitive ability of edge intelligent IIoT through machine learning. IEEE Netw. 33(5):61–67. https://doi.org/10.1109/MNET.001.1800505

Costa VG et al. 2024. Evolving interpretable decision trees for reinforcement learning. Artif Intell. 327:104057. https://doi.org/10.1016/j.artint.2023.104057

de J, Pacheco DA, Librelato TP. 2022. Optimising process and product performance in complex systems: a study in the automotive industry. *International Journal of Quality & Reliability Management*, Vol. 40 No. 4 pp. 922–941, doi: https://doi.org/10.1108/IJQRM-04-2020-0097.

Deb I, Gupta RK. 2023. Application of heuristics in production planning and job scheduling. Int J Res Publ Rev. 4(6):2531–2540. https://doi.org/10.55248/gengpi.4.623.45166

Dhebar Y et al. 2024. Toward interpretable-AI policies using evolutionary nonlinear decision trees for discrete-action systems. IEEE Trans Cybern. 54(1):50–62. https://doi.org/10.1109/TCYB.2022.3180664

Dragan D, Kramberger T, Popović V. 2020. Optimization methods and heuristics and their role in supply chains and logistics; p. 139–160.

Gadgil S, Xin Y, Xu C. 2020. Solving the lunar lander problem under uncertainty using reinforcement learning. ArXiv:2011.11850 [cs].

Gautron R, Padrón EJ, Preux P et al. 2022. gym-DSSAT: a crop model turned into a reinforcement learning environment. ArXiv:2207.03270 [cs].

Gosavi A. 2015. Solving Markov decision processes via simulation. In: Fu MC, editor. Handbook of simulation optimization. Springer. p. 341–379.

Günther J et al. 2016. Intelligent laser welding through representation, prediction, and control learning: an architecture with deep neural networks and reinforcement learning. Mechatronics. 34:1–11. https://doi.org/10.1016/j.mechatronics.2015.09.004

Hessel M et al. 2018. Rainbow: combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1), 3215-3222. https://doi.org/10.1609/aaai.v32i1.11796

Hubbs CD et al. 2020. A deep reinforcement learning approach for chemical production scheduling. Comput Chem Eng. 141:106982. https://doi.org/10.1016/j.compchemeng.2020.106982

Jijo B, Abdulazeez M. 2021. A classification based on decision tree algorithm for machine learning. J Appl Sci Technol Trends. 2:20–28.

Lee S, Cho Y, Lee YH. 2020. Injection mold production sustainable scheduling using deep reinforcement learning. Sustainability. 12(20):8718. https://doi.org/10.3390/su12208718

Mnih V, Kavukcuoglu K, Silver D et al. 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602. https://arxiv.org/abs/1312.5602.

Moore AW. 1990. Efficient memory-based learning for robot control. https://api.semanticscholar.org/CorpusID:60851166.

Nageshrao S, Costa B, Filev D. 2019. Interpretable approximation of a deep reinforcement learning agent as a set of if-then rules. In 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA). p. 216–221.

Nguyen S, Mei Y, Zhang M. 2017. Genetic programming for production scheduling: a survey with a unified framework. Compl Intell Syst. 3(1):41–66. https://doi.org/10.1007/s40747-017-0036-x

Paduraru C, Mankowitz DJ, Dulac-Arnold G et al. 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Mach Med Learn. 110:2419–2468.

Panzer M, Bender B. 2022. Deep reinforcement learning in production systems: a systematic literature review. Int J Prod Res. 60(13):4316–4341. https://doi.org/10.1080/00207543.2021.1973138

Puterman ML. 1994. Markov decision processes: discrete stochastic dynamic programming. 1st ed. Wiley. https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316887.

Raue M, Scholl SG. 2018. The use of heuristics in decision making under risk and uncertainty. In: Raue M, Lermer E, Streicher B, editors. Psychological perspectives on risk and risk analysis: theory, models, and applications. Springer International Publishing. p. 153–179. https://doi.org/10.1007/978-3-319-92478-6_7

Ridha MB. 2015. The role of heuristic methods as a decision-making tool in aggregate production planning. IJBA. 6(2), 68-76. https://doi.org/10.5430/ijba.v6n2p68

Ross S, Gordon GJ, Bagnell JA. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In: Gordon G, Dunson D, Dudík M, editors. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics; (Proceedings of Machine Learning Research. Vol. 15. PMLR. p. 627–635.

Sutton RS, Barto AG. 2018. Reinforcement learning: an introduction. In: Adaptive computation and machine learning series. 2nd ed. The MIT Press.

Terven J. 2025. Deep reinforcement learning: a chronological overview and methods. AI. 6(3):46. https://www.mdpi.com/2673-2688/6/3/46. https://doi.org/10.3390/ai6030046

Vaidya S, Ambad P, Bhosle S. 2018. Industry 4.0 – a glimpse. Procedia Manuf. 20:233–238. https://doi.org/10.1016/j.promfg.2018.02.034

Xie S, Zhang T, Rose O. 2019. Online single machine scheduling based on simulation and reinforcement learning.

Zare M et al. 2024. A survey of imitation learning: algorithms, recent developments, and challenges. IEEE Trans Cybern. 54(12):7173–7186. https://doi.org/10.1109/TCYB.2024.3395626

Žegklitz J, Pošík P. 2015. Model selection and overfitting in genetic programming: empirical study. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. Association for Computing Machinery. p. 1527–1528. https://doi.org/10.1145/2739482.2764678

Zhou T et al. 2021. Reinforcement learning with composite rewards for production scheduling in a smart factory. IEEE Access. 9:752–766. https://doi.org/10.1109/ACCESS.2020.3046784