Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Deep Reinforcement Learning for a Multi-Objective Online Order Batching Problem

Martijn Beeks,[1] Reza Refaei Afshar,[1] Yingqian Zhang,[1] Remco Dijkman,[1] Claudy van Dorst,[2] Stijn de Looijer[2]

[1] Eindhoven University of Technology, Eindhoven, Netherlands
[2] Vanderlande Industries, Veghel, Netherlands
m.s.beeks@student.tue.nl, {r.refaei.afshar, yqzhang, r.m.dijkman}@tue.nl, {claudy.van.dorst, stijn.de.looijer}@vanderlande.com

Abstract

On-time delivery and low service costs are two important performance metrics in warehousing operations. This paper proposes a Deep Reinforcement Learning (DRL) based approach to solve the online Order Batching and Sequence Problem (OBSP) to optimize these two objectives. To learn how to balance the trade-off between two objectives, we introduce a Bayesian optimization framework to shape the reward function of the DRL agent, such that the influences of learning to these objectives are adjusted to different environments. We compare our approach with several heuristics using problem instances of real-world size where thousands of orders arrive dynamically per hour. We show the Proximal Policy Optimization (PPO) algorithm with Bayesian optimization outperforms the heuristics in all tested scenarios on both objectives. In addition, it finds different weights for the components in the reward function in different scenarios, indicating its capability of learning how to set the importance of two objectives under different environments. We also provide policy analysis on the learned DRL agent, where a decision tree is used to infer decision rules to enable the interpretability of the DRL approach.

Introduction

Warehouse fulfillment solutions are a crucial factor for achieving successful supply chain management. Warehouse operations consider receiving, storing, picking, and consolidation of goods, where a sub-problem, the order batching and sequencing problem (OBSP), studies how to collect all orders that arrive at a warehouse such that certain objectives such as minimizing tardiness of orders are optimized (De Koster, Le-Duc, and Roodbergen 2007). The OBSP is considered as one of the most important operations as it accounts for 55% of the total warehousing costs (Bartholdi III and Hackman 2018).

Different orders can be grouped in one batch (called picking-by-batch), and each batch is assigned to a single picker, who will then collect all the required items in the storage system. For some orders, it might be better not to combine them in any batch (called picking-by-order) for a faster collecting process to meet the deadline (i.e., cut-off time). This is especially desirable in the e-commerce market, where fast deliveries are expected. In this paper, we study an online order batching problem. We decide whether each arriving order should be batched or not, i.e., picking-by-batch or picking-by-order, and if it is batched, which batches to assign it. The (offline) order batching problem is known to be NP-hard when the number of orders per batch is larger than two (Gademann and Velde 2005).

In the e-commerce era, companies face the challenge of assembling numbers of time-critical picking orders. They are investigating new solutions of order batching and picking systems to meet the requirements of significantly increased orders with different patterns of arrivals and balance various optimization objectives such as fast deliveries and low picking costs. The existing heuristics have difficulties in providing good solutions in a dynamic environment, either because they assume an offline (static) setting where all orders are known in advance, or the handcrafted heuristics cannot grasp important relations in stochastic processes (Boysen, De Koster, and Weidinger 2019). Recently, Cals et al. treated the online order batching problem as a sequential decision-making problem and proposed the first reinforcement learning approach to learn the best batching decision to minimize the number of tardy (late) orders.

In this paper, we extend the work of Cals et al. by considering an additional objective, i.e., minimizing order picking costs. While picking an order within a small batch will result in a short lead time, the order picking efficiency will decrease. When using RL for solving optimization problems, RL aligns the optimization objectives with learning through its reward function. Human modelers have some intuitions on the priority of one objective over the other in specific scenarios. For example, when many orders have their cut-off times approaching, the batching decisions should prefer minimizing tardy orders to picking costs. However, it is hard for human modelers to specify weight values for two objectives for different situations. To learn how to balance the trade-off between two objectives (i.e., tardy orders and picking costs), we propose a Bayesian optimization framework to shape a parametrized reward function, such that the weights (or importance) of these objectives are adaptive to different situations. Moreover, unlike in Cals et al., we model a true online setting for learning where at every time step, new orders arrive that change the state of the environment. Furthermore, to explain the learned policy of the DRL agent, a decision tree is extracted from the policy network. We compare our approach with several well-performing batching rules and heuristics from the literature. We test our approach in several realistic scenarios with a throughput rate between 2000-4000 orders per hour, in comparison to 300-500 orders in (Cals et al. 2021). We summarize our contributions as follows.

- We show the Bayesian optimization approach to shape reward functions effectively finds a good trade-off between two optimization objectives for the DRL agent.
- The proposed DRL based approach (DRL+RS) results in a better performance in terms of both objectives than the state-of-the-art heuristics for the order batching problem.
- We transform the learned deep neural network into a decision tree to explain the learned policy. Interestingly, using the set of transparent heuristics rules given by the distilled decision tree, we obtain better-batching decisions than the tested benchmark heuristics.
- As a practical implication, we show the proposed DRL approach can handle real-world size instances. Together with the approach of distilled decision trees for interpretability, we demonstrate the DRL approach is very promising to be applied and adapted to optimize warehousing operations in practice.

Related Work

Order Batching and Sequencing Problem. The majority of existing work on the OBSP assumes both a single objective and an offline setting where all the orders are known in advance. Won and Olafsson propose two heuristics for the OBSP which minimize the weighted average of the total picking time of batches and the holding time of orders in the batch. The work of Li, Huang, and Dai studies a method for joint optimization of order batching and picker routing in the online retailer's warehouse in China. The authors propose an online order batching strategy that uses a similarity coefficient that represents the similarity between two orders. The main idea is to combine similar orders as one to obtain a high order picking efficiency. The authors of Bustillo et al. propose a General Variable Neighborhood Search (GVNS) that performs an online batching operation. Chen, Wei, and Wang study an online order batching problem with the assumption that order splitting and batch modifying are possible. The authors of Pinto and Nagano introduce two genetic algorithms to adjust better the trade-off between two objectives, namely, the level of customer service and the effectiveness of a warehouse. Gil-Borrás et al. propose a greedy randomized adaptive search algorithm that constructs an initial solution and then a variable neighborhood descent procedure to improve. The objective is minimizing picking time.

DRL for Solving Optimization Problems. Deep reinforcement learning (DRL), which combines reinforcement learning and deep learning (Mnih et al. 2015), has demonstrated impressive results for solving combinatorial optimization problems and dynamic sequential decision-making problems. In the domain of warehousing management, Cals et al. propose the first reinforcement learning approach in optimizing warehousing operations. The authors use an actor-critic DRL algorithm for order batching with a single objective, i.e., minimizing tardy orders. The problem size of their approach is rather small, with 300-500 orders per hour, and the learning algorithm does not fully cope with the uncertain, dynamic arrivals of orders.

We extend the work of Cals et al. by considering two objectives. Human decision-makers have intuitions about the relative importance of weights of two objectives in different situations, however, it is hard for them to pre-define fixed weights. We treat the weight learning problem as a hyper-parameter optimization problem and use Bayesian Optimization (BO) (Snoek, Larochelle, and Adams 2012) to learn appropriate weights in the reward function that lead to high objective values. BO jointly tunes more hyper-parameters with fewer experiments. Moreover, it evaluates the objective function as less as possible and is robust to noisy evaluations. BO has been very popular in automated machine learning for algorithm selection (Hutter, Hoos, and Leyton-Brown 2011), and its applicability in automated reinforcement learning is discussed in (Refaei Afshar et al. 2022). However, to the best of our knowledge, the (Bayesian) optimization approach has not yet been applied to learn the reward function that contains parametrized objectives for enhancing learning behavior of optimization problems in multi-objective reinforcement learning (Hayes et al. 2021).

Problem Description

The considered warehousing concept consists of a PtG (Person-to-goods) and a GtP (Goods-to-person) storage areas (Figure 1). In the PtG storage area, a picker has to walk to pick goods using a picking cart. In the GtP storage area, goods are automatically retrieved from the warehouse using an Automated storage and retrieval system (AS/RS), one tote at a time. The PtG storage area leverages flexible picking capacity, whereas the GtP storage area provides faster picking times. In both storage systems, two different picking strategies can be enforced: Pick-by-order or Pick-by-batch. During a pick-by-order picking tour, only items from a single order are retrieved, and logically, during a pick-by-batch picking tour, orders from multiple orders (batch) are retrieved. A pick-by-batch picking strategy has relatively short picking times per order but has a higher lead time. A pick-by-order strategy has relatively long picking times per order (no synergy of picking several orders simultaneously) but short lead times. After picking, orders are transferred to consolidation workstations: Pack station, Sort-to-order station, and Direct-to-order station.

A worker at a DtO workstation removes items from a product tote and collects items of a single order in an order carrier. If an order consists of multiple order lines, multiple product totes are provided. All associated product totes arrive in sequence, and a picker places the items in an order carrier. Besides fulfilling orders in order carriers, the DtO workstation can batch items. In this process, product totes from the GtP area arrive, a picker places all items in a batch tote which is stored again in GtP area. After that, the stored batch tote in the GtP area can be requested by

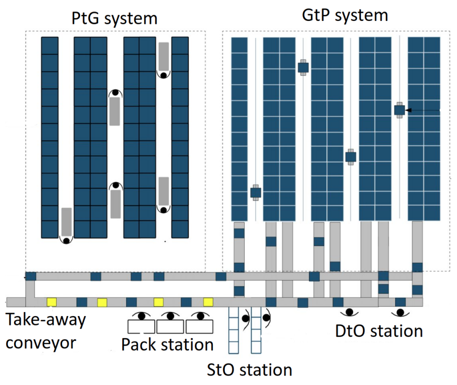

Figure 1: The warehousing concept (Cals et al. 2021)

either a pack station or a Sort-to-Order (StO) station. The StO station includes three processes: sorting, buffering, and packing. Within the sorting process, a picker removes items from a batch tote and sorts them in a put wall. After that, the operator places the sorted orders into carton boxes and transfers them. Lastly, the pack station processes batch totes consisting of only Single-item-Orders (SiO). A picker removes items from batch totes and places them in carton boxes that are subsequently sent for shipping. We refer to (Beeks 2021) for more detailed descriptions of the processes.

This work extends the work of (Cals et al. 2021) by transforming the original OBSP formulation to a multi-objective problem in an online setting. Given a warehousing setting, the proposed method needs to assign customer orders to pick orders and, subsequently, sequence batched orders such that capacity constraints are satisfied, maximize the orders that are shipped on time, and minimize order picking costs. An order is shipped on time when all picking and consolidations steps are performed before the respected cutoff time. The order-picking costs are defined by the amount of time that a picker takes to pick a single order within a batch.

Consider a time horizon T and a set of orders O, We define the decision variables as follows. For order $o \in O$ at $t \in T$, $PbO_o^t \in \{0,1\}$ represents the picking-by-order decision and $PbB_o^t \in \{0,1\}$ denotes the picking-by-batch decision. Similar to (Cals et al. 2021), the sequencing decision is performed using the Earliest Due Date sequencing heuristic. For each order $o \in O$, it has arrival time $a_o \in \{T\}$, its cutoff time (or due time) $d_o \in \{T\}$, its cutoff category based on urgency $e_o \in \{e_1, e_2, e_3\}$, and whether it is a single order or multiple item order $c_o \in \{SIO, MIO\}$. An order has its storage location $l_o \in \{PtG, GtP, both\}$. We have a set of resources $X = \{p, g, d, v, d\}$ where p represents PtG, g for GtP, d for DtO, v for StO and b for packing, and cap_x is the capacity of $x \in X$. We denote the batch size as N.

Several parameters need to be computed, depending on the state of orders that are already being picked and decisions that were made concerning these orders. They are: the service time of an order ($s_o \in \{T\}$); the picking time attributed to an order ($p_{o,x}$); the batch information to which an order is assigned ($ba_{o,b}$); and the number of resources that are occupied (occ_x^t).

Parameter	Description
O_{c_i,l_j,e_k}	Orders that need to processed divided in order categories with characteristics as order composition (c_i), storage location (l_j) and time until cutoff moment (e_k)
occ_x^t	Capacity information on resources
n, t and u	number of processed orders (n), number of tardy orders so far (t), simulation time (u)

Table 1: State space definition

An order becomes tardy, i.e., $tardy_o = 1$, if $a_o + s_o > d_o$. Each order is assigned to one and only one picking decision, i.e., $\sum_{t\in T} PbO_o^t + PbB_o^t = 1$ ($\forall o \in O$). Orders can only be picked after they have arrived, i.e., $PbO_o^t \cdot t + PbB_o^t \cdot t \geq a_o$ ($\forall o \in O, t \in T$). Resource abide the predefined capacities, i.e.,$occ_x^t \leq cap_x$ ($\forall x \in X, t \in T$). The maximum batch size of batches is respected, i.e., $\sum_{o\in O} ba_{o,b} \leq N$ ($\forall b \in B$). Lastly, orders cannot be split up and be allocated to multiple batches, i.e., $\sum_{t\in T} PbB_o^t = \sum_{b\in B} ba_{o,b}$ ($\forall o \in O$).

Given the above mentioned constraints and stochastic parameters, two objectives are minimized: number of tardy orders $f_1 = \sum_{o\in O} tardy_o$, and picking costs $f_2 = \sum_{o\in O} p_{o,x}$. In order to minimize f_1, orders need to be processed before their respective cutoff time. In order to minimize f_2, orders should have a pick-by-batch strategy. Hence, there might be a trade-off between these objectives.

Solution Methods

SMDP Formulation for the OBSP

Based on a specific state of the system, an agent can take a certain action that causes the system to go into a different state. The SMDP variant of a Markov Decision Process is defined by τ : the transition time, S : a finite state space, A : a set of actions, and R : the reward function. In an SMDP, the transition of time to the next state will depend on changes in the environment. The state (S) is represented based on information of orders that still need to be processed O_{c_i,l_j,e_k}, available resources in the system occ_x^t, and other useful information for learning (see Table 1).

The action space (A) is based on a pick-by-order or pick-by-batch decision for a specific order category. Based on the configuration of an order (SiO c_1 or MiO c_2) and the storage location l (PtG: l_1, GtP: l_2 or PtG & GtP: l_3), there are five different order categories (single-item-orders only have a single storage location, so PtG & GtP is not applicable). This results in an action space of 10 actions. Lastly, the 11th action is a wait action that the agent can take in case there are no resources left to process a certain set of order categories. This wait action 'waits' until the state of the system changes. This can happen either by the arrival of an order or by a resource becoming idle.

The reward function is displayed in Equation 1. The first two components account for the first objective, the minimization of the number of tardy orders, and provide a negative reward to the agent every time an order leaves the

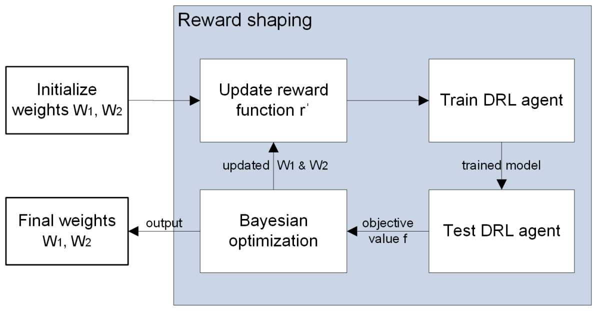

Figure 2: Reward shaping with Bayesian optimization

warehouse after its cutoff time. The second component provides a reward when an episode terminates where $u = \sum_{o \in O} tardy_o$ is defined as the number of tardy orders, and $|O|$ is defined as the total number of orders. With fewer tardy orders, this reward grows larger. The third component in this reward function penalizes the agent when it makes an infeasible action. This can happen when it provides a picking action for a certain order category where no resources are available. In this case, it ideally provides a wait action until resources become available. The fourth component provides an incentive to the agent to pick large batches and minimize the order picking costs. In this equation, $v = \sum_{o \in O} ba_{o,b}$ is defined by the actual batch size and N is defined by the maximum batch size. In this case, an agent is not punished when the number of orders within a batch is equal to the maximum batch size. In any other case, the agent is punished. Lastly, if no other components provide a positive or negative reward, the agent receives reward 0.

$$r(s, a, s') = \begin{cases} -1.5 & \text{if tardy order } (tardy_o = 1) \\ (1 - {}^{u}/_{|O|})^2 & \text{If an episode terminates} \\ -0.5 & \text{if infeasible action} \\ -\frac{(1 - {}^{v}/_{N})}{50} & \text{If order is picked} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The Proximal Policy Optimization (PPO) algorithm ((Schulman et al. 2017)) is used to interact with the environment and strikes a favorable balance between sample efficiency, simplicity, and computational efficiency. The PPO is an extension of the Trust Region Policy Optimization (TRPO) algorithm proposed by Schulman et al. and the Actor Critic with experience replay (ACER) presented by Wang et al.. Compared to off-policy learning methods such as DQN, PPO is an on-policy learning method that learns directly from experiences the agent has encountered. The environment has been modeled using a discrete event simulation method that represents the dynamics of the system and is used to interact with the DRL agent is implemented with the help of Hill et al..

Bayesian Optimization for Reward Shaping

Finding an aligned reward function for multi-objective optimization problems is difficult (Nguyen et al. 2020). The essential components of a reward function can be hand-crafted by domain experts based on their knowledge. However, the exact weights for different components representing different optimization objectives are difficult for human experts to define. Therefore, We propose to use Bayesian optimization (BO) to find appropriate weights of different performance objective measures in the reward function (i.e., reward shaping); see Figure 2. The idea of using an optimization algorithm to enhance the learning behavior of a DRL approach is adapted from Chiang et al. where the authors used Bayesian optimization for hyperparameter tuning of the DRL approach to a navigation task.

Bayesian optimization has been widely used for hyperparameter optimization in supervised learning, where an initial set of hyper-parameters is selected, and then a model is trained and tested. Based on the testing results, the BO updates the surrogate function, and a new set of hyperparameters is provided. Very often, a Gaussian Process is used as a surrogate function. We aim to find an aligned reward function for DRL in our case. Hence, two weights W_1 and W_2 are considered as hyper-parameters, reflecting the importance of corresponding two optimization objectives (i.e., minimizing tardy orders and minimizing picking costs) in the reward function for learning. Subsequently, we update the reward function in Equation (1) to assign weights to corresponding components as follows.

$$r'(s, a, s') = \begin{cases} W_1 \times (-1.5) & \text{if} tardy_o = 1 \\ W_1 \times (1 - \frac{u}{|O|})^2 & \text{If episode terminates} \\ -0.5 & \text{if infeasible action} \\ W_2 \times (-\frac{(1 - {}^{v}/_{N})}{50}) & \text{If order is picked} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

By optimizing the weights in the reward function for a specific scenario, the DRL agent can cope with different environments throughout the day. In other words, the BO framework will suggest a set of weights for the reward function based on each scenario. The starting weights and ranges have been set by a domain expert.

We illustrate the Bayesian optimization approach, adapted from O'Hagan, for reward shaping in Figure 2 and Algorithm 1. The hyper-parameter space X consists of two parameters, W_1 and W_2, which range between 0.5 and 1.5. This range is determined with a set of preliminary experiments. The objective function evaluates the solution by training and testing a DRL agent with the newly proposed weights. The two optimization objectives are transformed into a single objective function f using a multiplication scalarization method: $f = \frac{\sum_{o \in O} tardy_o}{|O|} \times \frac{\sum_{o \in O} p_{o,x}}{|O|}$. In this way, an increase or decrease in each of the objectives attributes proportionally to the single objective value.

The algorithm (Algorithm 1) first samples an initial set of weights (x_0) from the hyper-parameter space and evaluates it with the objective function f after testing the trained DRL agent. Then, the algorithm iterates until the predefined maximum number of evaluations are reached. In each iteration, first, a new set of hyper-parameters (weights) is selected with the expected improvement (EI): $u_{EI}(x) = (f(x^* - \mu(x))\Phi(Z) + \phi(Z)$, where Φ is the standard normal cumu-

Algorithm 1: Pseudo-code of Bayesian optimization for reward shaping

```
**Input** hyper-parameter space $X = W_1, W_2$, objective
  function $f(x)$, max evaluations $n_{max}$
**Output** $x^*$, $y^*$
Select an initial hyper-parameter configuration $x_0 \in X$
Evaluate the initial score of $x_0$ by training and testing the
  DRL approach: $y_0 = f(x_0)$
Set $x^* = x_0$ and $y^* = f(x_0)$
**for** $n \in \{1, ..., n_{max}\}$ **do**
    $u_{EI}(x) = (f(x^* - \mu(x))\Phi(Z) + \phi(Z)$
    Select a new hyper-parameter $x_n \in X$ by maximizing
      the expected improvement $u_{EI}(x)$
    $x_n = \arg\max_x u(x)$
    Evaluate $f$ for $x_n$ to obtain a new objective score
      $y_n = f(x_n)$
    Update the Gaussian Process with $(x_n, y_n)$
    **if** $y_n < y^*$ **then**
        $x^* = x_n$ and $y^* = y_n$
```

lative distribution function, ϕ its derivative, $Z = \frac{f(x^* - \mu(x))}{\sigma(x)}$ and μ is the current posterior predictive mean function of the Gaussian Process. Hereafter, this newly sampled set of weights is evaluated using the objective function $(f(x_n))$. The weights and objective values are appended to S, and herewith, the Gaussian Process distribution is updated. Finally, x^* and y^* are updated if the obtained objective value is the best so far.

Distillation of DRL Policy

A noteworthy disadvantage of DRL methods is the lack of clarity as to how specific actions are selected. The root of this difficulty lies in the distributed nature of the representations embedded in the hidden layers of the DNNs (Frosst and Hinton 2017). Decision trees offer an alternative where actions can be traced back through sequences of decisions based directly on input data. Using structured data that has been computed by the learned policy, a decision tree is fitted on certain state-action combinations. It infers a set of rules that separates different actions from each other (Che et al. 2016). We will use this approach to increase the explainability of learned DRL models.

Experiments

Experiment Setup

The dataset used for order arrivals is from a large e-commerce company in The Netherlands. This dataset includes 257,585 orders with around 376,522 items, where 70% of the orders have only 1 item, 14% of the orders have two items, 9% of the orders have three, and 4% of the orders have four items. This is a well-known pattern in the e-commerce business. For this scenario, it is assumed that 100,000 SKUs (Stock Keeping Units) are in the warehouse. For these experiments, hourly cutoff times from 17:00 until 24:00 are randomly allocated to each arriving order within 3 hours to provide a challenging context. It is ensured that a cutoff moment of an order is planned a minimum of 30 minutes after the arrival time of the order. So orders that arrive before 23:30 are included in the model and have to be processed before 24:00. This is in line with most of the delivery requirements of these e-commerce companies where they ensure next-day delivery, and this represents real-world characteristics where orders leave the warehouse in specific groups. With this set of rules for determining the cutoff moments, a distribution of cutoff moments per day is depicted in Figure 3. In total, 30% of the SKUs are stored in the PtG storage location, 40% of the SKUs are stored in the GtP storage location, and lastly, 30% of the SKUs are stored in both storage locations. In cases where a Multi-item-Order, for example, consists of an item stored in the PtG area and an item that is stored in both areas, the model picks the item that is stored in both areas in the PtG area. We vary several parameters in experimental settings. Firstly, to reflect real-world characteristics, throughput rates between 2000 and 4000 orders per hour are used. Secondly, based on the selected throughput setting, an appropriate set of resources are adjusted to the desired throughput rate such that the tardy orders and picking costs are at reasonable levels in a preliminary analysis. Thirdly, total simulation time consists of either two hours or an entire day. In both options, it is interesting to analyze the behavior of the proposed methods and observe whether these methods can cope with changing characteristics in terms of a changing order arrival pattern. Lastly, orders are released in real-time, making this an online problem. We have four experimental settings.

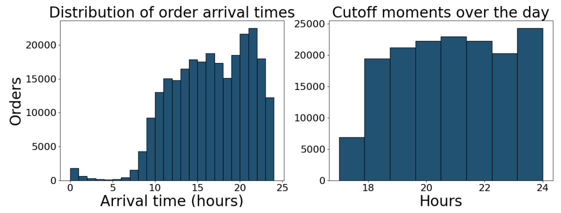

Figure 3: Distributions of order data

- Setting A: Processing 6000 orders from 15:00 - 17:00
- Setting B: Processing 7000 orders from 20:00 - 22:00
- Setting C: Processing 6500 orders from 22:00 - 24:00
- Setting D: Processing 50,000 orders from 10:00 - 24:00

Benchmark Heuristics

We compare the proposed DRL approach, including reward shaping (DRL+RS), with three heuristics from literature. The first is the LST rule that sorts orders on the due date, create batches with a batch size of N, and computes the slack time by subtracting the processing and arrival time from the cutoff time. If there is a negative slack time, the batch size is reduced iteratively.

The second heuristic proposed by Li, Huang, and Dai for joint optimization of online order batching and picker routing in the online retailer's warehouse in China. The BOC heuristic uses a similarity coefficient between two orders to obtain a high order picking efficiency. To optimize also

Algorithm 2: Adapted BOC heuristic

```
Input Set of unprocessed orders $I$, batch capacity constrains, max simulation time $t_{max}$
Output Set of batched orders $B$
$B \longleftarrow$ list
$t \longleftarrow 0$
while $I$ is not empty and $t < t_{max}$ do
    Choose the order with the most imminent cutoff time $i^*$
    while $i^*$ does not violate capacity constraints do
        Compute $S_{i^*j} = \frac{\text{size of } A_{i^*} \cap A_j}{\text{size of } A_j}$ for all $j \in I$
        Sort orders by $S_{i^*j}$ in descending order
        Select order $(i^*j)$ with the highest $S_{i^*j}$
        if Multiple orders have same similarity then
            Choose the order $j$ with most imminent cutoff time
        Combine order $j$ and $i^*$ as a new order $i^*$
    Append $i^*$ to $B$
    Remove all orders within $i^*$ from $I$
update simulation time $t$
```

for the number of tardy orders, we adapt BOC by initiating the seed order with an order that has the most imminent cutoff time. Furthermore, in the case of an equal similarity score, orders with an imminent cutoff time have priority. The adapted BOC is shown in Algorithm 2. The algorithm outputs a set of batches to which all orders have been assigned. This process iterates until there are no orders left in I and when the simulation time exceeds the max simulation time t_{max}. The first step of the batching operation is to select a seed order and subsequently iterate until the capacity constraints of a batch are reached. Within this iteration, the similarity coefficient is computed, and a suitable order is selected based on this coefficient. This selected order is combined with the seed order, and this process iterates until the batch-specific capacity constraint has reached. If so, the compiled batch is appended to a list B, and the included orders are removed from the unprocessed order list I.

The third benchmark heuristic, GVNS (General Variable Neighborhood Search) algorithm by Bustillo et al., exploits the idea of neighborhood change in a systematic way. The aim is to descend to a local optimum or, alternatively, to escape from the basin of attraction from that local optimum. To adapt this algorithm for optimizing two objectives, the evaluate function computes both expected order pickings costs and the expected tardy orders (algorithm 3).

Starting from an initial batching solution s based on the Earliest Due Date (EDD) batching heuristic, the entire approach iterates until the maximum simulation time (t_{max}) has been reached. First of all, given a solution S, the shaking procedure performs a swap move that generates a new solution $S^{'}$ in k consecutive times where items between orders are exchanged. The insert move will transfer a certain item to a new order. After this shake operation, the VND operation performs both of the local search moves in order to improve the solution. If one of the two moves improves the current solution, it is accepted as the new current solution. The solutions are evaluated using a function that computes average picking time and expected tardy orders. After the VND operation, all solutions are evaluated, and a new current solution is selected in the function $NeighborhoodChange$.

Algorithm 3: Adapted GVNS algorithm

```
Input Largest neighborhood to be explored $k_{max}$, max computing time $t_{max}$ and initial solution $S$
Output Improved solution $s$
while $t < t_{max}$ do
    $k \longleftarrow 1$
    while $k < k_{max}$ do
        for $i \in \{0, ..., k\}$ do
            $S^{'} \longleftarrow Swap(S)$
        $k \longleftarrow 1$
        while $k < k_{max}$ do
            $S^{''} \longleftarrow Insert(S^{'}), S^{'''} \longleftarrow Swap(S^{'})$
            if $evaluate(S') < evaluate(S^{''})$ then
                $S^{'} = S^{''}$
                $k \longleftarrow 1$
            else if $evaluate(S^{'}) < evaluate(S^{'''})$ and $evaluate(S^{''}) < evaluate(S^{'''})$ then
                $S^{'} = S^{'''}$
                $k \longleftarrow 1$
        $NeighborhoodChange(S, S^{''}, k)$
    $t \longleftarrow CPUTime()$
```

The implementation of our approach is available at https://github.com/ai-for-decision-making-tue/drl-order-batching.

Results

The model parameters for training a PPO agent are mostly similar to the original parameters of the work of Schulman et al.. The training consists of 4-16 million steps according to the specific experiment. This training is done in an environment where four agents are trained in parallel. Real-world order data is used to fit an order arrival distribution function, which is then used to sample training data at each iteration. So training was done on the different data sets over iterations. The actor and critic network is updated by all the agents simultaneously, whereby the agent can learn more state-action pairs at the same time. The discount factor has been set at 0.9999 as the episodes can be fairly long. The neural network consists of fully-connected multi-layer perceptrons with two hidden layers of 64 units and tanh activation layers. The PPO algorithm has been implemented using the python package Stable-baselines ((Hill et al. 2018)). This package has been built upon the Tensor flow framework.

Table 2 shows the results of the proposed DRL approaches with reward shaping (DRL+RS), without reward shaping (DRL), and the heuristics (BOC, LST, GVNS) on solving the multi-objective OBSP in terms of tardy orders and order picking costs. These results are obtained by simulation per setting for respectively 228, 554, 168, 172 runs. The values between brackets represent the standard deviation. For the DRL+RS approach, an agent was trained and tested for experiment settings A, B, and C. Then, for setting D, the obtained weights from the previous settings are used for training the agent due to the computational complexity of experiment D. During the different time windows of the day, dif-

Model	Setting A TA	Setting A PC	Setting B TA	Setting B PC	Setting C TA	Setting C PC	Setting D TA	Setting D PC
BOC	3.17	46.80	6.24	49.00	11.80	45.42	23.91	35.40
LST	17.6	65.12	18.7	68.83	21.15	67.21	25.76	53.34
GVNS	6.82	47.41	13.05	48.92	18.78	45.48	26.46	35.52
DRL	2.03	48.65	4.18	50.70	6.65	**42.22**	13.39	34.15
DRL + RS	**1.68**	**43.02**	**2.60**	**46.87**	**5.53**	42.90	**12.40**	**33.96**

Table 2: Experimental results of the DRL approach with and without reward shaping (RS) and three heuristics where TA: Tardy orders(%) and PC: Picking Costs

ferent weight settings are applied to the reward function of the agent for setting D. In this way, the agent's reward function is dynamically shaped according to the moment in time of the day. The mean values of all approaches for setting A, B, and C are tested using a student t-test. These provide statistical evidence that the means are different from each other with an α of 0.05. For every setting, the DRL approach with reward shaping outperforms all the tested heuristics in both objectives. Only DRL without reward shaping outperforms the DRL+RS slightly in setting C in terms of picking costs.

In the DRL approach with reward shaping and setting A, after 50 iterations, the value of 0.86 for tardy orders and 0.90 for picking costs have been found. The best set of weights for setting B is 1.34 for tardy orders and 0.52 for picking costs. For setting C, the weight of tardy orders seems to have a slightly larger impact on the objective value than in the other two settings. This could be explained by the fact that within this setting, all orders have their respected cutoff time at the end of the setting, and therefore, the agent focuses on picking these orders in time. For setting C, the best set of weights are 1.20 for tardy orders and a weight of 0.45 for order picking costs. The weight for order picking costs is decreasing over setting A, B, C. This could indicate that order picking costs become less important near the end of a day and tardy orders become more important. For DRL without reward shaping, the weights have been defined using expert knowledge, which reflects the business objectives of a typical system owner. Weight number 1 that punishes tardy orders was set to 1, and weight number 2 that stimulates efficient order picking, was set to 0.5. All heuristics have used a similar same weight definition as for DRL without reward shaping provided by a domain expert.

To assess robustness, all experiment settings have been adjusted so that 10% more orders require to be processed with the same resources. These results have been displayed in Table 3. For setting A, B, and D, the picking costs increased slightly, where it is assumed that this can be attributed to the lower percentage of tardy orders. For the DRL approach in setting C, both the percentage of tardy orders and the picking costs are lower than the BOC heuristic results. The DRL approach with reward shaping outperforms the BOC heuristic in almost all settings based on tardy orders and picking costs. As the DRL+RS approach remains the dominant method, this is somewhat robust to small changes in their current experiment settings.

Model	Setting A TA	Setting A PC	Setting B TA	Setting B PC	Setting C TA	Setting C PC	Setting D TA	Setting D PC
BOC	1.98	46.83	5.91	**48.80**	10.23	45.23	23.83	35.34
DRL	1.23	48.29	3.54	50.65	6.83	41.74	15.28	35.37
DRL + RS	**0.9**	**45.69**	**1.9**	49.65	**5.82**	**41.5**	**9.4**	**33.12**

Table 3: Robustness analysis DRL approach where TA: Tardy orders(%) and PC: Picking Costs

Lastly, this section presents the learned weights by the DRL+RS approach for each setting A: (0.86, 0.94), B: (1.13, 0.52), and C: (1.28, 0.45). These weights represent the priority for tardy orders W_1 and the weight for order picking costs W_2 respectively. For setting C, the weight of tardy orders has a larger impact on the objective value than the other two settings. This could be explained by the fact that by the end of this setting, all orders must have left the warehouse in order to be on time and that the agent prioritizes this over efficient order picking. The weight for order picking costs is decreasing over all settings. This could indicate that order picking costs become less important near the end of a day and that tardy orders become more important.

Explaining DRL policy with DT Decision trees (DT) can make learned policies more interpretable and provide explanations for certain predicted actions on given state representations. The work of Che et al. provides insights in a DRL policy using these DT and is used to provide insights to the DRL approach with reward shaping for setting A in Figure 4. Using the DRL approach with reward shaping trained on experiment setting A, a dataset has been compiled with a state representation of 20 features (state space representation) and 10-class (action space representation) predicted actions. This experiment is conducted between 15:00 and 17:00 and does not contain a lot of cutoff moments. Using this structured dataset, a classification approach is taken, and a decision tree is fitted. The dataset is unbalanced where the following list indicates the occurrence per class [7224, 13008, 20441, 59091, 1550, 1778, 9463, 20415, 17601, 5284]. Using a 10-fold cross validation evaluation method, the decision tree obtained an f1-score of 47%. Using this decision tree in Figure 4, several 'rules' can be

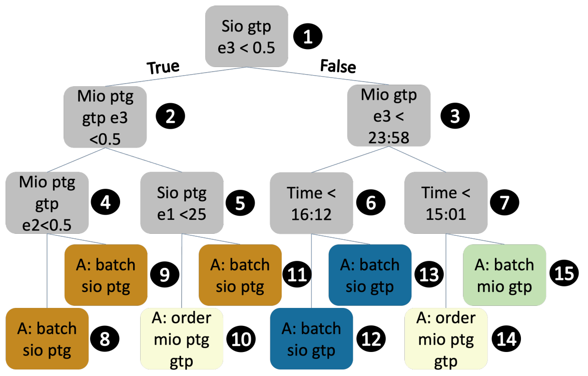

Figure 4: Decision Tree trained on DRL agent of setting A

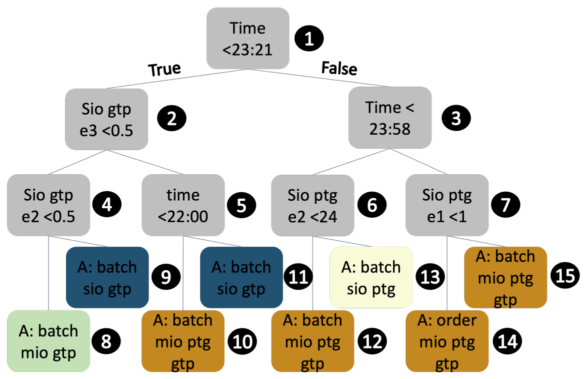

Figure 5: Decision Tree trained on DRL agent of setting C

inferred where the colors represent similar predicted actions and the entropy in all nodes refers to the 'disorder' of this node in providing a certain decision rule when classifying the data. First of all, the decision tree makes a split based on whether there are orders within the order category sio-ptg-e3. This order category contains orders that are single-item, are stored in the PtG storage area, and have a cutoff time that is more than 40 minutes from the current moment. Furthermore, in node 2, the decision tree splits the data based on orders within order category mio-ptg-gtp. If there are orders within this order category and the condition in node 5 is true, a pick-by-order action for order category mio-ptg-gtp is predicted. In all other cases, the decision tree predicts a pick-by-batch action for order category sio-ptg (node 4, 8, 9, 5, 11). On the right side of figure 4, the decision tree makes a split based on order category mio-gtp-e3 where if there are less than 25 orders in this category, the decision tree predicts a pick-by-batch action for order category sio-gtp (node 6, 12, 13). If this is not the case, the model makes a cut in node 7 where based on the simulation time, either a pick-by-batch action for order category mio-gtp is made of a pick-by-order action for order category mio-ptg-gtp.

Experiment C is conducted between 22:00 and 24:00 and contains a lot of cutoff moments for orders arriving within this window and poses a different dynamic than in experiment A. Using the DRL + RS approach, a generated dataset with 20 features and 10-class predictions is used to fit a decision tree. Similar to the setup for setting, this yielded an unbalanced dataset but still managed to obtain a f1-score of 56 %. The DT in Figure 5 makes the first split based on the simulation time, indicating the importance of time dependency in selecting actions learned by the DRL approach. Then it is checked whether there are orders within the order category sio-gtp-e3 in node 2. This order category contains orders that are single-item, are stored in the GtP storage area and have a cutoff time that is more than 40 minutes from the current moment. If there aren't any orders within this category, the DT prescribes to choose a batching action for mio-gtp orders in node 4. Most of the time, there are orders within this category (node 4), and the tree describes picking a batch of SiO GtP orders. On the other side of the tree, the simulation time is again used for forming two branches where it is checked whether the current simulation time is lower than 23:58 and higher than 23:21 in node 3. Furthermore, on the right side of the tree, a split is made on the sio-ptg-e2 order category in node 6, where if there are less than 24 orders within this category, the tree describes picking a batch of MiO orders with storage areas PtG and GtP in node 12. The only pick-by-order action is predicted for order category mio-ptg-gtp in node 14 when there are very few orders in order category sio-ptg-e1. This appears to be logical as the system does not have to spend pick-by-order action to this popular order category and can allocate its resources somewhere else. With some of the decisions of the DT explained, some dependencies can be identified based on simulation time and the number of orders within order categories.

Model	Setting A Tardy orders (%)	Setting A Picking costs	Setting C Tardy orders (%)	Setting C Picking costs
BOC	3.17 (1.1)	46.8 (0.8)	11.8 (1.8)	45.4 (0.9)
DRL	2.03 (0.8)	48.65 (0.8)	6.65 (0.8)	**42.22 (0.7)**
DRL + RS	**1.68 (0.7)**	**43.02 (0.6)**	**5.53 (1.1)**	42.90 (0.6)
Distilled DT	3.06 (0.9)	45.59 (0.7)	9.24 (1.0)	44.84 (0.7)

Table 4: Results of the distilled decision tree as heuristic

The DT infers batching rules from the DRL agent and is subsequently used as a heuristic for solving the OBSP for experiment settings A and C. Table 4 shows the results. In terms of tardy orders and order picking costs, the distilled DT outperforms the BOC heuristic significantly ($\alpha = 0.05$). As decision trees are not able to capture all behavior of DRL approaches, there is no equal performance.

Conclusion

Optimizing operations in warehousing has become very challenging for online retailers. In this work, we develop a deep reinforcement learning approach with reward shaping that can solve a multi-objective variant of the online order batching problem. We show the proposed DRL approach can handle real-world size instances and gives better solutions than the existing heuristics in terms of both performance measures, i.e., reducing tardy orders to increase customers' satisfaction and reducing picking costs to save operation costs. This shows the advantage of learning in tackling stochastic sequential decision-making problems. After training, the model can be used in the real-time warehousing system. Together with the approach of distilled decision trees for interpretability, we demonstrate the DRL approach is very promising to be applied and adapted to optimize warehousing operations in practice.

In this work, we use a rather straightforward method to extract decision rules from the policy network. As future work, we will investigate more tailored approaches to infer DRL agent's policy.

References

Bartholdi III, J.; and Hackman, S. 2018. Warehouse & distribution science: release 0.96. Atlanta, GA: The Supply Chain and Logistics Institute, School of Industrial and Systems Engineering, Georgia Institute of Technology.

Beeks, M. 2021. *Deep reinforcement learning for solving a multi-objective online order batching problem.* Master's thesis, TU Eindhoven.

Boysen, N.; De Koster, R.; and Weidinger, F. 2019. Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, 277(2): 396–411.

Bustillo, M.; Menéndez, B.; Pardo, E. G.; and Duarte, A. 2015. An algorithm for batching, sequencing and picking operations in a warehouse. In *2015 international conference on industrial engineering and systems management (iesm)*, 842–849. IEEE.

Cals, B.; Zhang, Y.; Dijkman, R.; and van Dorst, C. 2021. Solving the Online Batching Problem using Deep Reinforcement Learning. *Computers & Industrial Engineering*, 107221.

Che, Z.; Purushotham, S.; Khemani, R.; and Liu, Y. 2016. Interpretable deep models for ICU outcome prediction. In *AMIA annual symposium proceedings*, volume 2016, 371. American Medical Informatics Association.

Chen; Wei, Y.; and Wang, H. 2018. A heuristic based batching and assigning method for online customer orders. *Flexible Services and Manufacturing Journal*, 30(4): 640–685.

Chiang, H.-T. L.; Faust, A.; Fiser, M.; and Francis, A. 2019. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2): 2007–2014.

De Koster, R.; Le-Duc, T.; and Roodbergen, K. J. 2007. Design and control of warehouse order picking: A literature review. *European journal of operational research*, 182(2): 481–501.

Frosst, N.; and Hinton, G. 2017. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*.

Gademann, N.; and Velde, S. 2005. Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE transactions*, 37(1): 63–75.

Gil-Borrás, S.; Pardo, E. G.; Alonso-Ayuso, A.; Duarte, A.; et al. 2020. GRASP with Variable Neighborhood Descent for the online order batching problem. *Journal of Global Optimization*, 1–31.

Hayes, C. F.; Rădulescu, R.; Bargiacchi, E.; Källström, J.; Macfarlane, M.; Reymond, M.; Verstraeten, T.; Zintgraf, L. M.; Dazeley, R.; Heintz, F.; Howley, E.; Irissappane, A. A.; Mannion, P.; Nowé, A.; Ramos, G.; Restelli, M.; Vamplew, P.; and Roijers, D. M. 2021. A Practical Guide to Multi-Objective Reinforcement Learning and Planning. arXiv:2103.09568.

Hill, A.; Raffin, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; Traore, R.; Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; and Wu, Y. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines. Accessed: 2021-04-16.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, 507–523. Springer.

Li, J.; Huang, R.; and Dai, J. B. 2017. Joint optimisation of order batching and picker routing in the online retailer's warehouse in China. *International Journal of Production Research*, 55(2): 447–461.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Nguyen, T. T.; Nguyen, N. D.; Vamplew, P.; Nahavandi, S.; Dazeley, R.; and Lim, C. P. 2020. A multi-objective deep reinforcement learning framework. *Engineering Applications of Artificial Intelligence*, 96: 103915.

O'Hagan, A. 1978. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(1): 1–24.

Pinto, A. R. F.; and Nagano, M. S. 2019. An approach for the solution to order batching and sequencing in picking systems. *Production Engineering*, 13(3-4): 325–341.

Refaei Afshar, R.; Zhang, Y.; Vanschoren, J.; and Kaymak, U. 2022. Automated Reinforcement Learning: An Overview. *arXiv*, 2022.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International conference on machine learning*, 1889–1897.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; and de Freitas, N. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*.

Won, J.; and Olafsson, S. 2005. Joint order batching and order picking in warehouse operations. *International Journal of Production Research*, 43(7): 1427–1442.