POLITECNICO DI TORINO
Repository ISTITUZIONALE

Scheduling autonomous robots for an intralogistic application: A comparison of lookahead-based ADP strategies

Original
Scheduling autonomous robots for an intralogistic application: A comparison of lookahead-based ADP strategies / Battistotti, Margherita; Brandimarte, Paolo; Mazzi, Nicolò; Giancola, Francesca. - In: EXPERT SYSTEMS WITH APPLICATIONS. - ISSN 0957-4174. - ELETTRONICO. - 271:(2025), pp. 1-14. [10.1016/j.eswa.2025.126590]

Availability:
This version is available at: 11583/2997032 since: 2025-01-29T15:30:13Z

Publisher:
Elsevier

Published
DOI:10.1016/j.eswa.2025.126590

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

23 February 2025

Expert Systems With Applications 271 (2025) 126590

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Scheduling autonomous robots for an intralogistic application: A comparison of lookahead-based ADP strategies

Margherita Battistotti [a], Paolo Brandimarte [b,*], Francesca Giancola [c], Nicolò Mazzi [d]

[a] AMBS - University of Manchester, Booth Street West, Manchester M15 6PB, United Kingdom
[b] DISMA - Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
[c] Spindox S.p.a., via Bisceglie 76, 20152 Milano, Italy
[d] Spindox Labs S.r.l., via alla Cascata 56C, 38123 Trento, Italy

ARTICLE INFO

Keywords:
Intralogistics
Scheduling
Approximate dynamic programming
Rollout strategies
Monte Carlo tree search

ABSTRACT

The increasing role of e-commerce has spurred a significant amount of research on optimization in warehousing management, including routing and scheduling issues. When material handling is rigidly automated, a deterministic scheduling problem arises, for which solution strategies have been proposed in the literature. A recent trend is the introduction of autonomous robots, which may interact with human operators and offer additional flexibility in item manipulation. The resulting problem is affected by uncertainty, due to the interaction between robots and human workers and the possible failure in items manipulation. In the paper, we propose an adaptation of approximate dynamic programming strategies with limited lookahead, namely, rollout strategies and Monte Carlo tree search. The idea can be interpreted as an intermediate approach between the solution of a deterministic problem, disregarding uncertainty and using a long lookahead, or the application of pure state-based dispatching rules with no lookahead. The proposed approaches are compared against exact dynamic programming on small-size instances, and then evaluated on larger instances, proving their viability.

1. Problem description and motivation

The increasing role of e-commerce, additionally boosted during the Covid period, has made efficient automatic warehouse management a crucial problem. In such a problem, we have to tackle a combination of scheduling and routing subproblems, along with some new and peculiar issues (Boysen et al., 2019a). In some applications, the system is highly automated, and the problem requires picking items and delivering to a workstation, where a human worker manually picks items from a set of trays and places them into boxes. The role of the worker is simply to assemble the shipment corresponding to a customer order, which may include multiple items (Boysen et al., 2023a). In such an automated setting, the resulting scheduling problem is essentially deterministic, although quite demanding in terms of synchronization issues. With other system configurations, there is an interaction with AGVs (Löffler et al., 2022) or robots (Boysen et al., 2023b). The relevance of human–robot interactions in a practical setting is also shown in Allgor et al. (2023).

The research described in this paper fits within the EU-funded research project DARKO,[1] a setting in which anthropomorphic robots operate autonomously, moving in a workspace where collision with human workers must be avoided, and not only pick items, but also deposit them into trays, where shipments are assembled. The relevant consequence is that the scheduling problem becomes stochastic:

- On the one hand, potential collision with human workers is detected and anticipated by stopping the robot for a time period. Hence, the traveling time between two points in the workspace may be stochastic, as a delay may be introduced.
- On the other hand, the robot may deposit each item in the tray from a safe location, or from a more distant point. This kind of action, referred to as *throwing*, increases throughput, but may result in a failure. Thus, a second stochastic factor comes into play, as in the case of a failure in a deposit action implies the loss of the item, so that a new item of the same type must be picked.

In this paper, we consider a simplified case featuring a single robot. The problem will be fully and formally specified in Section 3, but it

* Corresponding author.
E-mail addresses: margherita.battistotti@postgrad.manchester.ac.uk (M. Battistotti), paolo.brandimarte@polito.it (P. Brandimarte), francesca.giancola@spindox.it (F. Giancola), nicolo.mazzi@spindox.it (N. Mazzi).

[1] DARKO is an acronym for *Dynamic Agile production Robots that learn and optimize Knowledge and Operations*. See https://darko-project.eu/about/ for more information.

https://doi.org/10.1016/j.eswa.2025.126590
Received 11 September 2024; Received in revised form 15 January 2025; Accepted 16 January 2025
Available online 24 January 2025
0957-4174/© 2025 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

comprises both a scheduling and a routing component. The robot must fulfill a set of jobs, where each job comprises a set of items that must be collected from their store locations and deposited into a specific tray. There is a set of trays, each one corresponding to a parcel to be assembled, with a given priority. There is a scheduling component, as we must determine a sequence of pick and place actions, possibly interleaved among them. There is a routing component, as we must consider travel times between points in the space, accounting for the risk of delays due to collisions, and the risk of a failure if an item is "thrown" from a convenient but distant point.

When dealing with a stochastic routing/scheduling problem, different strategies may be applied:

1. One strategy is to ignore uncertainty and to plan actions optimally, solving a combinatorial optimization problem. This is an open-loop strategy. When the plan significantly deviates from reality, due to random disruptions, the control loop is closed and replanning occurs. In this way, on the one hand, we introduce a lookahead into the strategy, but this must be often revised, which may prove to be both inefficient and ineffective.
2. On the opposite side of the spectrum, we may give up any idea of a lookahead, and just define dynamic priority rules, which sequence actions on the basis of the current system status. This is a common approach in machine scheduling, but such a myopic approach may result in poor performance.
3. A predictive lookahead may be introduced by stochastic dynamic programming, which does not disregard the possible occurrence of stochastic disruptions. This yields a closed-loop policy based on system status but, unlike greedy priority rules, better accounts for the possible state evolution.

While the third approach is highly desirable, it is not quite practical due to the well-known curse of dimensionality in dynamic programming (Powell, 2011). Approximate Dynamic Programming (ADP) strategies may be applied, but they are still demanding. Moreover, since our problem is not recurring, as the portfolio of orders is continuously evolving, we have to learn a policy with some suitable frequency. The rationale is to find a compromise between adaptability and performance. The ADP learning algorithm is run at an intermediate frequency, so that the learned policy may be applied in front of failures, and it is updated when necessary (typically, when a tray is emptied and a new customer order must be served).

We will only consider lookahead-based ADP strategies, rather than, e.g., a pure Value Function Approximation (VFA) strategy. We should note that this work builds upon a previous unpublished study conducted by one of the authors as part of her master's thesis (Battistotti, 2024), and some material has been adapted and revised from that original work. In the thesis, a simpler version of the problem, disregarding priorities, was addressed, but an VFA technique (Approximate Policy Iteration) was also considered. Since VFA was outperformed by lookahead-based policies, we will not consider it here. For a discussion of the advantages of introducing a lookahead, including some theoretical arguments, we refer to Bertsekas (2020).

The plan of the paper is the following. In Section 2 we outline the relevant literature, in order to position our contribution. The problem we use as a case study is specified and then formally stated as a Markov Decision Process (MDP), amenable to solution by stochastic dynamic programming (DP), in Section 3. Three solution strategies are described in Section 4: exact DP, which is practical only for small-scale problems, myopic rollout, and Monte Carlo tree search. Experimental results are outlined in Section 5, and conclusions are drawn in Section 6.

2. Literature review

In order to provide useful and relevant references, and to position this paper, we have split the review in two subsections. First, in Section 2.1, we highlight the key role of proper warehouse management in e-commerce. Then, in Section 2.2 we summarize key concepts in approximate dynamic programming, in order to frame the algorithmic approaches that we have investigated. Finally, in Section 2.3, we discuss the positioning of our paper and its potential contributions, as well as its limitations.

2.1. Warehousing in e-commerce

Optimal warehousing management is a traditional topic in Operations Research, as illustrated by Boysen and de Koster (2024). Within an industrial setting, material handling is regarded as an operation that does not add value to the product. In an e-commerce setting, where timely delivery is a key service quality factor, warehousing has become a key factor to profitability. A recent survey with emphasis on e-commerce is provided by Boysen et al. (2019b).

The exact problem formulation depends on the exact system layout, and it often involves a combination of order batching and picking, which may be formulated as an integer programming problem; see, e.g., (Valle et al., 2017). Due to the difficulty of the resulting problem, heuristic approaches are typically applied. Pan et al. (2015) apply genetic algorithms, whereas (Cheng et al., 2015) propose a hybrid approach based on the particle swarm optimization and ant colony optimization. Optimal policies are discussed by Schiffer et al. (2022), including dynamic programming.

Warehousing systems may involve different levels of automation and interaction between human operators and devices. For instance, Löffler et al. (2022) discuss the problem of routing pickers in picking systems assisted by automated guided vehicles (AGVs). A recent tendency is to move from AGVs to AMRs, i.e., autonomous mobile robots. In such systems, due attention must be paid to the human–robot interaction (Allgor et al., 2023; Löffler et al., 2023). Zhang et al. (2022) propose a mixed-integer nonlinear model formulation of such problems, along with heuristic solution strategies.

Most model formulations are deterministic, and emphasis may be on routing and/or scheduling issues, depending on the nature of the system layout. On the contrary, Boysen et al. (2023b) consider uncertainty in the arrival stream of orders and consider limited look-ahead strategies. In our paper, we also consider uncertainty and limited lookahead strategies but, as we discuss later, we consider different risk factors.

2.2. Strategies for approximate dynamic programming

In principle, stochastic dynamic programming (SDP) is a suitable tool to tackle complex sequential decisions under uncertainty, like those involved in warehouse management under uncertainty. Unfortunately, it is well-known that SDP suffers from multiple curses, including, but not limited to the familiar curse of state dimensionality (Brandimarte, 2021; Powell, 2011). Nevertheless, several approximate dynamic programming (ADP) strategies are available, which are often able to yield high-quality decision policies under uncertainty. See, e.g., (Powell, 2009, 2010) for a general overview of ADP strategies, and Powell and Simao (2012) for a review geared towards transportation and logistics applications.

Such strategies are classified by Powell (2019) into four main approaches, which van also be hybridized. The most traditional idea is value function approximation (VFA), whereby the value function of the standard Bellman optimality equation is approximated in parametric or non-parametric form. A general survey is provided by Geist and Pietquin (2013). See, e.g., (Ulmer et al., 2018) for a discussion aimed at routing problems. Ulmer and Thomas (2020) propose an integration of parametric and non-parametric approximations. Another strategy, cost function approximation (CFA), relies on the approximation of a dynamic decision problem by a static one, where the objective function is modified in order to foster non-myopic decisions.

An alternative approach aims at building approximate policies directly, by policy function approximation (PFA). A policy function directly maps states into decisions. In complex warehousing and logistics applications, finding such a map may be far from easy, even though simple heuristics may be devised and interpreted as approximate policy functions. In order to improve performance, some form of lookahead may be introduced, which is the fourth ingredient in ADP and reinforcement learning (Bertsekas, 2019). Widely used approaches to improve a base decision policy by introducing a form of lookahead include rollout (Bertsekas, 2020; Goodson, 2013) and Monte Carlo tree search (Browne et al., 2012).

With more specific reference to the application of rollout strategies to combinatorial optimization and scheduling under uncertainty, we should also mention early papers such as (Bertsekas & Castanon, 1999; Bertsekas et al., 1997). The adaptation of ADP strategies to complex stochastic scheduling problems, such as those encountered in project scheduling, is illustrated in more recent papers like (Li & Womer, 2015; Solomon et al., 2019; Xie et al., 2021).

2.3. Paper positioning and contribution

Unlike most literature, we consider a stochastic problem, where uncertainty is not only related to the incoming order stream, but to the mission execution by autonomous mobile robots (AMRs). An AMR must avoid collision with human operators, which implies that they may have to stop along their way, and that routing should consider the potential presence of human operators. Moreover, to speed up operations, an AMR may throw an object into the tray from a certain distance from the ideal position. Since this may result into a partial mission failure, the tradeoff between a relatively risky action and a safe one should be accounted for. Hence, we believe that our paper (which is related to the UE-funded DARKO project), presents some distinguishing features with respect to the literature. We should mention that a similar approach to ours is adopted by Zhou et al. (2024), where stochastic dynamic programming is applied to provide a control policy for the immediate future, which is revised at a lower frequency. Dynamic programming is also applied in a similar context by Justkowiak et al. (2024), but a deterministic model is considered.

From a methodological viewpoint, we apply well-known lookahead-based ADP strategies, like myopic rollout and Monte Carlo tree search, which must be suitable adapted to the problem at hand. As we discuss in the paper, this also involves some care in shaping the appropriate rewards. Hence, we offer some contribution in terms of reward engineering literature.

What we describe is a limited case study, which is not meant to be a fully functional solution. Indeed, we solve a very specific subproblem (as described in Section 3 below). Moreover, we consider a single AMR. Hence, our contribution is methodological, as we investigate the suitability of specific stochastic ADP strategies to a problem with some new and interesting features.

3. Formal problem statement

We address the problem of scheduling tasks for an intralogistic application, where a robot must collect objects and carry them to specific destinations in a warehouse, while being constrained to a fixed maximum carrying capacity of c objects, regardless of their type. The list of object types to be moved to the destinations, henceforth also referred to as assembly spots, and the respective quantities are defined by a set of orders associated with a priority level, e.g., "*5 units of objectA and 8 units of objectD are demanded, with priority 1*". The arrival of orders into the system is stochastic, and every newly arrived order is placed in a sorted queue based on its associated priority level. The most urgent order exits the queue whenever an assembly spot is available; hence, the pairs order-destination characterize a dynamic mission. The robot's objective is to schedule a time-efficient and minimum-risk sequence of tasks to fulfill all orders that arrive before a fixed time horizon, while respecting their priorities.

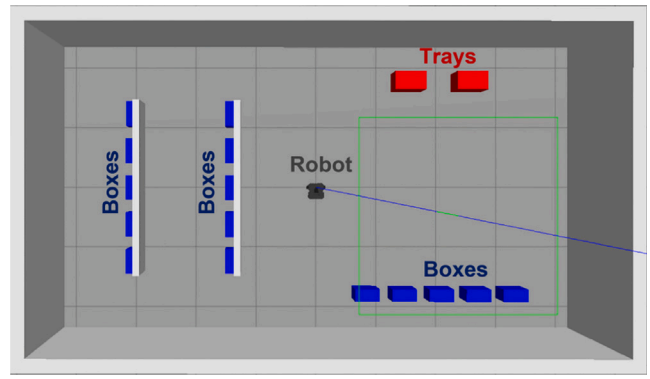

Fig. 1. Toy example of the research use case.

The environment, as illustrated by the toy example in Fig. 1, is indeed a three-dimensional space where a single agent, identified as the robot, can perform three main action types: it can move from one location to another, pick objects from boxes, and place into or throw them towards assembly spots. Note that not all actions guarantee deterministic outcomes, because collisions with humans or with shelving units may occur during navigation, while throws may fail. Indeed, collisions and failures represent the exogenous risk factors affecting the system, and the consequences of their occurrence are, respectively, a time delay and the loss of the object whose throw was attempted.

3.1. The problem setting

The system layout may be represented by a completely connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where each vertex $n \in \mathcal{N}$ represents either a picking or a throwing location. The set $\mathcal{E}$ of edges connecting the vertices represents the optimal paths to follow in terms of time-efficiency and risk-avoidance: in fact, the completely connected graph $\mathcal{G}$ is the result of a previously solved routing optimization problem. As depicted in Fig. 2, each edge $e = (n_0, n_1) \in \mathcal{E}$ is associated with two parameters: Δt_e, which is the time needed to move from vertex n_0 to the destination vertex n_1, and r_e, which is the probability that the robot will face a potential collision when traveling through e. Since the routing optimization problem defining the edges of the graph is solved once and before scheduling begins, both parameters r_e and Δt_e are assumed fixed during the scheduling problem resolution. This choice leads to a static view of the risk factors associated with moving actions, which are actually dynamic in the real world. Nevertheless, as we will later see, the time horizons for the orders' fulfillment are generally set to be short in our experiments, and a unique snapshot of the initial situation is a close approximation of the risk throughout the entire scheduling. If deemed necessary, an option would be to repeatedly solve the routing problem and dynamically adjust the data associated with the edges of graph $\mathcal{G}$.

3.2. Problem definition

Let $\mathcal{O}$ be the set of O object types and $\mathcal{D}$ the set of D destinations, from now on also referred to as trays, where objects might need to be carried to. Each object type is univocally associated with a box where it is stored and from where it can be picked. In our setting, box locations are defined by the picking vertices $n \in \mathcal{N}_{pick}$, while tray locations are associated with coordinates $(x_d, y_d), d \in \mathcal{D}$, that differ from the throwing vertices $n \in \mathcal{N}_{throw}$. Nevertheless, each vertex $n \in \mathcal{N}$ is also associated to a pair of coordinates (x_n, y_n).

Let then consider time as continuous, such that $t \in \mathcal{T} = [0, T]$ with T being a fixed a time horizon. As we aim to work with a discrete event

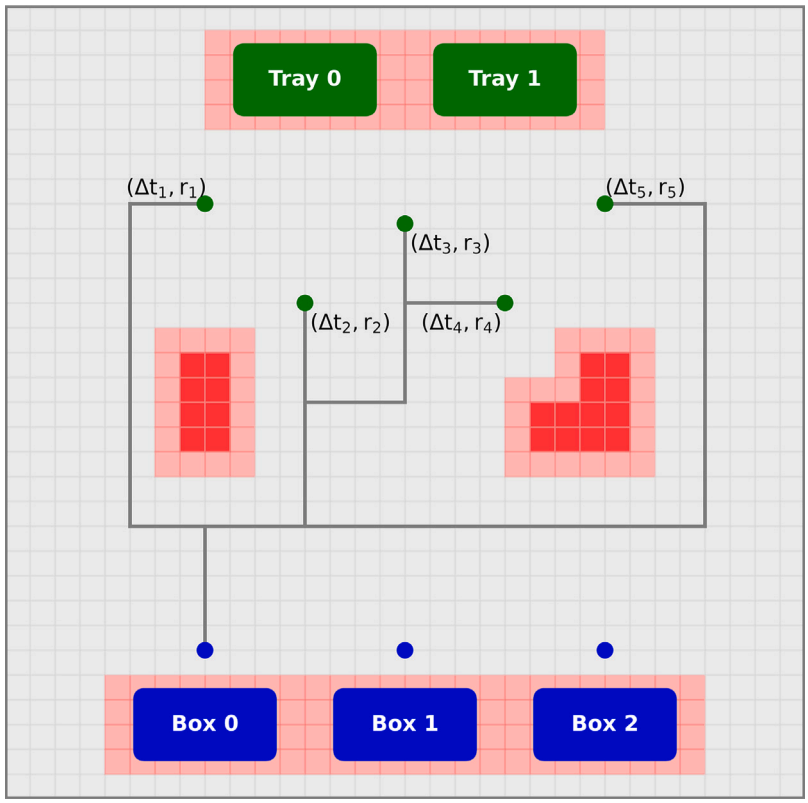

Fig. 2. Partial representation of graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$: each edge $\mathsf{e} \in \mathcal{E}$ is associated to the parameters pair $(\Delta t_{\mathsf{e}}, r_{\mathsf{e}})$. Central red zones represent risky areas, i.e., possible obstacles with which collisions occur with probability r_{e}.

system, we define a natural set of indexes $\mathcal{I} = \{0, 1, 2, \dots, I\}$ such that time flow is modeled by discrete decision epochs defined by the time instants $t_i \in \mathcal{T}$ at which decisions are made. We collect such t_i in an ordered set of increasing time instants $\mathcal{T}_{\mathcal{I}} \in \mathcal{T}$, with $t_0 = 0$ and $t_I \leq T$.

3.2.1. Mission, queue, and service priorities

Let us suppose that at the beginning of the scheduling at least D orders are already in the queue waiting to be served, so that the mission dynamically assigned to the robot is always defined by D order-tray pairs as $\mathcal{M} = \{(k_d, d), d \in \mathcal{D}\}$, where $k_d = \{(o, q_o), o \in \mathcal{O}\}$ is the order associated to tray d demanding a quantity of q_o items for object type o. For the arrival in the queue of other orders we assume a Poisson process with rate λ. The queue is indeed a dynamic list of orders sorted by their associated priority level $l \in \mathcal{L} = \{1, 2\}$, 1 being the most urgent. The levels of priority, together with arrival times, define a rule of precedence for serving the orders in the queue. Indeed, every time a tray becomes available again an order substitution takes place according to the selection of the order in the queue with the longest waiting time among the most urgent ones. Moreover, in order to avoid longer-waiting, lower-priority orders to be surpassed by newly arrived, higher-priority orders, after a predefined period of time priorities of all less urgent orders in the queue rank up by one level.

3.2.2. State space

We are ready to define the state space $\mathcal{S}$. Let $s \in \mathcal{S}$ be defined as an array of dimension $\Omega = 2 + O \times (D + 1)$, resulting from a concatenation of the following:

- $s^1 \in \mathcal{T}_{\mathcal{I}}$, which represents the time elapsed from the beginning of the initial mission in terms of decisional epochs;
- $s^2 \in \mathcal{N}$, which represents the position of the robot on the graph at time s^1;
- $\underline{s}^o \in \mathbb{N}_0^{(D+1)}$, defined $\forall\, o \in \mathcal{O}$, such that:
  - $\underline{s}^{o,1} \in \{0, \dots, \bar{O}\}$ is the number of objects of type o picked before time s^1, where $\bar{O} = \sum_{d=1}^{D} q_{o_d} \mid o_d = o$, represents the total number of objects of type o to pick during the mission defined at that time;
  - $\underline{s}^{o,d+1} \in \{0, \dots, q_{o_d}\}$ is the number of objects of type o placed in tray $d \in \mathcal{D} = \{1, \dots, D\}$ before time s^1, where q_{o_d} represents the total number of objects of type o to place in tray d during the mission defined at that time.

For example, let $O = 2$ and $D = 2$, so that $\Omega = 2 + 2 \times (2 + 1) = 8$. A generic state $s \in \mathcal{S}$ would be denoted as $s = (t_i \in \mathcal{T}_{\mathcal{I}}, n \in \mathcal{N}, \underline{s}^1, \underline{s}^2)$ and, given a suitable mission $\mathcal{M}$, one could have $\underline{s}^1 = (4, 2, 1)$ and $\underline{s}^2 = (1, 1, 0)$. Note that each state records information about which objects have been picked and/or placed up to a specific time, and enables a deduction of the remaining tasks required to complete the respective current mission. In fact, whenever an order is fulfilled and there is at least one other order waiting in the queue, a tray substitution is performed, hence the record regarding the just emptied tray is reset and the mission updated.

3.2.3. Action space

Let us now focus on the action space. As previously stated there are three main action types: move, pick, throw, which can further branch off into more specific actions by associating to each type additional information. For example, to a moving action type we shall associate a vertex where to move, and for a picking action type we shall specify which object has been chosen to be picked. We work with a set $\mathcal{A} = \mathcal{A}_{move} \cup \mathcal{A}_{pick} \cup \mathcal{A}_{throw}$, where $\mathcal{A}_{move} \subset \mathcal{N}$, $\mathcal{A}_{pick} \subset \mathcal{O}$ and $\mathcal{A}_{throw} = \{(o, d), o \in \mathcal{O}, d \in \mathcal{D}\}$, and various subsets $\mathcal{A}_s \subset \mathcal{A}$ only containing the admissible actions given a state $s \in \mathcal{S}$. Furthermore, each action is associated to the time duration of its execution, namely $\Delta t_{\mathsf{th}} = 5$ time units for all throwing actions, $\Delta t_{\mathsf{pck}} = 7$ for all picking actions, and Δt_{e}, as previously defined, for moving actions on edge $\mathsf{e} \in \mathcal{E}$.

3.2.4. Risk factors, immediate contributions and state transitions

Action types are also linked to rewards that depend on the state of the system, on the time at which the action is performed and on its outcome. We will refer to such rewards as immediate contributions and denote them as $C_{t_i}(s_{t_i}, a_{t_i}, w_{t_i+\Delta t})$, where $w_{t_i+\Delta t}$ is the realization of risk factors during the time interval subsequent to the time instant of the decision. In fact, being time flow described by time instants $t_i \in \mathcal{T}_{\mathcal{I}}$ corresponding to decisional epochs, and by uneven intervals $\Delta t \in \Delta = \{\Delta t_{\mathsf{pck}}, \Delta t_{\mathsf{th}}, \Delta t_{\mathsf{e}}, \mathsf{e} \in \mathcal{E}\}$ corresponding to actions' duration, given a state of the system s_{t_i} at time t_i and a succeeding state $s_{t_i+\Delta t}, \Delta t \in \Delta$, one can indicate the realization of risk factors in between the two as $w_{t_i+\Delta t}, \Delta t \in \Delta$.[2]

As previously mentioned, our system is affected by two risk factors directly associated to throwing and moving actions, while picking actions are deterministic.

The probability of success of a throwing action from throwing vertex $n \in \mathcal{N}_{throw}$ to tray d is given by:

$$p_{\text{throw}} = \begin{cases} 0, & \text{if } \|(x_n, y_n), (x_d, y_d)\|_2 \geq M \\ \frac{1}{M-m}\left(M - \|(x_n, y_n), (x_d, y_d)\|_2\right), & \text{otherwise,} \end{cases} \tag{1}$$

where M and m are threshold distances based on the specific warehouse dimensions and the robot's throwing capabilities considered in the application. A throw fails if the distance between the throwing vertex n and the tray d exceeds M, while $m \leq \min_{n,d}(\|(x_n, y_n), (x_d, y_d)\|_2)$ contributes to the normalization factor $M - m$, so that $p_{\text{throw}} \leq 1$ for all throwing actions. In our experiments, the parameters were based on our graph modeling and set to $M = 80, m = 8$. Hence, p_{throw} is inversely proportional to the distance between the throwing location and the destination, and completely independent of the object thrown. The possible outcomes arising from a throwing action are thus binary: success, denoted by $w_{t_i+\Delta t_{\text{th}}} = w_{i+1} = 1$, is expected with probability p_{throw}, while failure, $w_{t_i+\Delta t_{\text{th}}} = w_{i+1} = 0$, may occur with probability $1 - p_{\text{throw}}$.

For what concerns moving actions, parameter $r_{\mathsf{e}}, \mathsf{e} \in \mathcal{E}$ indicates a risk percentage, hence the probability of having a failure $w_{t_i+\Delta t_{\mathsf{e}}} = w_{i+1} = 0$, i.e., a collision, when traveling through the corresponding edge is

$$p_{\text{move}} = \frac{r_{\mathsf{e}}}{100}. \tag{2}$$

Instead, a smooth crossing of the same edge happens with probability $1 - p_{\text{move}}$, and is denoted by $w_{t+\Delta t_{\mathsf{e}}} = w_{i+1} = 1$.

A clarification on the outcomes of risk factors prompts to the definition of immediate contributions and state transitions, which depend on them.

For a throwing action $a_i = (o, d) \in \mathcal{A}_{s_i,throw}$, i.e., throw object $o \in \mathcal{O}$ in tray $d \in \mathcal{D}$, the immediate contribution is defined as:

$$C_i(s_i, a_i, w_{i+1}) = \begin{cases} 0, & \text{if } w_{i+1} = 0 \\ \frac{r_{\text{throw}}(2T - s_i^1)}{T} - \frac{\alpha T_d^e}{\max_d(T_d^e)+1} + \frac{\beta(s_i^1 - T_d^e)}{\min_d(s_i^1 - T_d^e)+1}, & \text{else,} \end{cases} \tag{3}$$

where T_d^e is the entering time in the mission of the order assigned to tray d, r_{throw} is a multiplicative factor common to all throwing actions and α and β are positive coefficients, set both equal to 1 in our experiments. The negative term suggests a growing penalty for orders that entered the system earlier, while the one weighted by β gives an incentive to throw objects demanded by orders that have been waiting for a longer time. However, both terms help defining a tray-filling (or order-completing) priority. For the same throwing action the state transition is ruled by equation

$$s_{i+1} = g_{t_i+\Delta t_{\text{th}}}(s_i, a_i, w_{i+1}), \tag{4}$$

such that

$$s_{i+1}^1 = s_i^1 + \Delta t_{\text{th}}, \quad \text{for } w_{i+1} = 0, 1, \tag{5}$$

$$\underline{s}_{i+1}^{o,1} = \begin{cases} \underline{s}_i^{o,1}, & \text{for } w_{i+1} = 1 \\ \underline{s}_i^{o,1} - 1, & \text{for } w_{i+1} = 0, \end{cases} \tag{6}$$

$$\underline{s}_{i+1}^{o,d+1} = \begin{cases} \underline{s}_i^{o,d+1} + 1, & \text{for } w_{i+1} = 1 \\ \underline{s}_i^{o,d+1}, & \text{for } w_{i+1} = 0. \end{cases} \tag{7}$$

Eq. (5) simply updates the elapsed time from the beginning of the scheduling, while Eqs. (6) and (7) respectively reflect that when a throw fails an object is lost and that, instead, a successful throw places the object in the corresponding tray.[3] Note that the dependence on just the previous state of the system rather than on the whole path is a common assumption better known as Markovian property, on which we hinge throughout our entire paper.

For a moving action $a_i = n \in \mathcal{A}_{s_i,move}$, i.e., move from s_i^2 to n along $e = (s_i^2, n) \in \mathcal{E}$, the immediate contribution is null in case of a smooth crossing of the edge and equals -2 when collisions occur. Moreover, the system transitions according to:

$$s_{i+1} = g_{t_i+\Delta t_{\mathsf{e}}}(s_i, a_i, w_{i+1}), \tag{8}$$

where

$$s_{i+1}^1 = \begin{cases} s_i^1 + \Delta t_{\mathsf{e}}, & \text{for } w_{i+1} = 1 \\ s_{i+1}^1 = s_i^1 + \Delta t_{\mathsf{e}} + 5, & \text{for } w_{i+1} = 0, \end{cases} \tag{9}$$

$$s_{i+1}^2 = n, \quad \text{for } w_{i+1} = 0, 1. \tag{10}$$

Eq. (10) indicates the transition from the initial location s_i^2 to the intended destination n, while Eq. (9) suggests a penalty of 5 time units in case of a collision. Finally, deterministic picking actions (such that $w_{t+\Delta t_{\text{pck}}} = w_{i+1} = 1 \ \forall i$) of the type $a_i = o \in \mathcal{A}_{s_i,pick}$, i.e., pick object $o \in \mathcal{O}$, define the deterministic immediate contribution

$$C_i(s_i, a_i, 1) = \frac{r_{\text{pick}}(2T - s_i^1)}{T} \tag{11}$$

with r_{pick} being a multiplicative factor common to all picking actions, and bring the system to a succeeding state

$$s_{i+1} = g_{t_i+\Delta t_{\text{pck}}}(s_i, a_i, 1), \tag{12}$$

accounting for one more picked item of type o after Δt_{pck} time units. In fact:

$$s_{i+1}^1 = s_i^1 + \Delta t_{\text{pck}}, \tag{13}$$

$$\underline{s}_{i+1}^{o,j} = \begin{cases} \underline{s}_i^{o,j}, & \forall j \neq 1 \\ \underline{s}_i^{o,j} + 1, & \text{for } j = 1. \end{cases} \tag{14}$$

Note that the decrease with time of the immediate contributions associated to both picking (11) and throwing (3) actions is needed to emphasize the importance of collecting and placing objects at the earliest convenient opportunity.

3.2.5. Problem objective

Let us finally state the problem objective. After the definition of a suitable terminal reward value function $F(\cdot) : \mathcal{S} \to \mathbb{R}$, that measures the quality of terminal states of the system, the natural way of stating the stochastic problem so far described should be:

$$\max_{\pi \in \Gamma} \quad \mathbb{E}\left[\sum_{i \in \mathcal{I}} \gamma^i C_i(s_i, a_i, w_{i+1}) + \gamma^I F(s_{t_I})\right], \tag{15}$$

$$\text{s.t.} \quad a_i \in \mathcal{A}_{s_i} \ \forall i \in \mathcal{I}, \tag{16}$$

[2] Hereafter, for the sake of notation's simplicity, wherever necessary we will substitute indexing by time t_i with just i, provided it does not create ambiguity. As examples: $s_{t_i} = s_i$, $s_{t_i+\Delta t} = s_{t_{i+1}} = s_{i+1}$, and $w_{t_i+\Delta t} = w_{t_{i+1}} = w_{i+1}$.

[3] We explicit state transition equations only for state entries that actually vary when an action is performed. Other entries, whose transition is implicit, remain unchanged.

$$s_{i+1} = \begin{cases} (4) \text{ if } a_i \text{ is a throw action,} \\ (8) \text{ if } a_i \text{ is a move action,} \quad \forall\, i+1 \in \mathcal{I}, \\ (12) \text{ if } a_i \text{ is a pick action,} \end{cases} \tag{17}$$

where $\gamma \in (0,1]$ is a discount factor, and Γ is the set of admissible policies. A policy is a sequence of functions $\pi = (\pi_i)_{i \in \mathcal{I}}$, such that each $\pi_i : \mathcal{S} \to \mathcal{A}$ maps a state of the system $s_i \in \mathcal{S}$ at time t_i to an admissible action $a_i \in \mathcal{A}_{s_i}$. The objective becomes the search for an optimal policy.

4. Solution strategies

The paradigm of Dynamic Programming (DP) has been chosen to innovatively solve the problem of intralogistic robot scheduling described in the previous section because of its notorious flexibility. Indeed, its applications span across various fields, from operations research to economics, from control theory to machine learning. DP is not a fixed and defined algorithm, but rather an optimization principle, and as such its implementation for a specific problem may require a considerable customization effort (Brandimarte, 2021) that counterbalances its appealing flexibility. Furthermore, it is as flexible as computationally expensive: curses of dimensionality are its Achilles' heel, and it might prove impractical for larger scale problems. For this reason, in our research, exact DP is only applied to small problem instances, in order to employ the output solutions as benchmarks to validate approximate implementations for larger scale instances.

Such approximate implementations are chosen as two Approximate Dynamic Programming (ADP) resolution techniques that fall under the category of Look-Ahead Policies (LAPs): Myopic Rollout (MR) and Monte Carlo Tree Search (MCTS). Another point in favor of the implementation of two LAPs is that they do not require exhaustive replanning when a new order starts being served. In fact, given our problem setting, when exploiting the exact DP paradigm we must repeat the complete enumeration of all the new possible states of the system whenever an order associated to a tray is substituted. This will become clearer in the next section, as it will the fact that with LAPs this extensive enumeration is not necessary. LAPs assist in decision-making in a certain state by simulating (a limited number of) plausible scenarios, thus eliminate the need of exploring the entire state system.

4.1. Exact dynamic programming

The main idea of exact DP is to recursively solve a multi-stage dynamic decision problem as the one presented by decomposing it into smaller sub-problems. The key procedure is to evaluate states based on their appeal through the use of value functions $V_i : \mathcal{S} \to \mathbb{R}$, that measure the quality of being in a certain state at time instant t_i. As for actions, their quality is somehow evaluated through the immediate contributions $C_i(\cdot)$, as in the general formulation (15).

Let us describe how the recursive resolution of the sub-problems is performed through a backward pass. In finite-horizon problems as ours, a value is assigned to all possible terminal states based on their quality through a specific terminal reward value function, chosen as

$$F(s) = (T - s^1) - \sum_d \sum_{o \in k_d} \left(q_o - \underline{s}^{o,d+1}\right) + \sum_o \underline{s}^{o,1}, \tag{18}$$

for our problem. The term $T - s^1$ linearly rewards the early completion of the mission with respect to the fixed time horizon T, the double summation is a penalization of one unit for every object that was supposed to be placed but has not (there is no penalty if the mission is completed before time horizon is reached), while the last summation is a prize of one unit for every picked object. Then, for each previous state that would bring the system to a terminal one, the assigned value is the result of an optimization problem over the admissible actions. The objective of the sub-problem is the expectation of the sum of the immediate contribution and the discounted future state value, conditional to the current state and the chosen action. The process is then repeated until the initial state is reached.

In brief, what just described is the recursive application of the Bellman's Equation:

$$\begin{cases} V_I(s_I) = F(s_I) \\ V_i(s_i) = \underset{a_i \in \mathcal{A}_{s_i}}{\text{opt}}\ \mathbb{E}\left[C_i(s_i, a_i, w_{i+1}) + \gamma V_{i+1}(s_{i+1}) | s_i, a_i\right],\ i \in \mathcal{I} \setminus \{I\}. \end{cases} \tag{19}$$

In conclusion, the final optimal solution given by the application of the Dynamic Programming principle is defined by one last pass, forward in time. Starting from a given initial state, for each time instant t_i at which a decision must be made, the final forward pass selects the optimal action as the argument satisfying (19). Note that in our problem setting the state space varies whenever a new order is assigned to a tray, leading to the necessity of repeating a backward pass as the one described above. The more orders to serve, the more tray's substitutions take place, thus more backward passes are needed. We will discuss in Section 5 that this frequent and thorough replanning considerably affects the execution time of the paradigm on our problem.

4.2. Lookahead-based ADP strategies

In this section, we describe how two look-ahead policies (LAPs) have been adapted to the specific problem we deal with. For the sake of completeness, we should mention the other common ADP techniques, based on Value Function Approximation (VFA) or Policy Function Approximation (PFA). A VFA approach is similar to LAPs in the sense that both work going forward in time, avoiding the expensive backward pass of exact DP. However, the former aims at building functions to approximate the values that would be associated with all the states within an exact DP paradigm. It does so by first simulating several sample paths from an initial state, forward in time, up to a *long* time horizon or a terminal state, in order to find suitable parameters to approximate the value functions to be used later during the decision-making process. One could say that it includes a learning phase, comprising simulations, separate from and prior to the actual decision-making. On the other hand, each decision made with LAPs when in a state is directly based on simulations starting from that state and ending after a limited number of steps, i.e., the lookahead. A PFA approach (in principle) is also feasible for our problem, as the action set is finite. Nevertheless, in this paper, we do not describe VFA and PFA approaches because, as mentioned in Section 1, we have investigated Approximate Policy Iteration in a previous study (Battistotti, 2024), in order to learn a suitable approximation of the optimal value function. Computational experiments shown that a larger computational effort is required to obtain a performance that is similar to LAPs. For further considerations on the advantages provided by lookahead, we refer again to Bertsekas (2020).

4.2.1. Myopic rollout

The adjective *Myopic* describes the activity of making a decision just by looking roughly into the future, without a crystal ball. However, MR represents an improved version of a more naive decision-making technique: a Myopic Policy that only relies on the values of immediate contributions to select the best action. The improvement is indeed reflected in the noun *Rollout*, which stands for the recursive procedure of rolling to the next state after making the myopic decision and then repeating the process for a fixed number of steps. During this phase an estimate of the value of being in the state from where the rollout has started is produced, based on a probable future path. If, given a state s, the MR procedure is performed for all the states reachable from s, as shown in Fig. 3, a sub-optimal action can be chosen based on the expectation of the myopic value produced for the state the action may lead the system to, according to

$$\arg\max_{a \in \mathcal{A}} \mathbb{P}(1)(C_i(s,a,1) + V'^a) + \mathbb{P}(0)(C_i(s,a,0) + V''^a), \tag{20}$$

where V'^a and V''^a are the values associated through MR to the states eventually succeeding s when action a is performed, in case of success

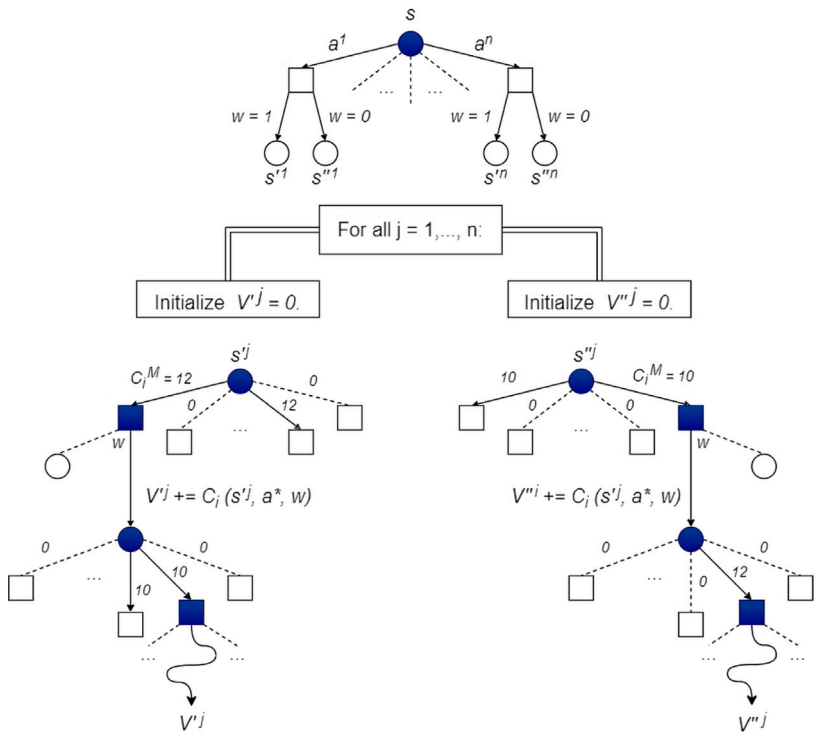

Fig. 3. Example of decision-making process through MR for generic state s. Square nodes represent the states of the system before the random realizations. For every state (round node) reachable from s a value is computed with a MR. The best action is then chosen as in Eq. (20).

and failure respectively, occurring with probability $\mathbb{P}(1)$ and $\mathbb{P}(0)$.

Given the simplicity of the algorithm its implementation only requires the definition of a few parameters. For example, it is essential to define how far in the future to "roll" and how to myopically choose between a set of available actions. For the decision rule we simply opt for the Myopic Policy that outputs an action solely based on a myopic version of its deterministic immediate contribution, i.e.,

$$a_i^\star = \arg\max_{a\in\mathcal{A}_{s_i}} C_i^M(s_i,a) = \arg\max_{a\in\mathcal{A}_{s_i}} \cdot \frac{r_a(2T - s_i^1)}{T}, \tag{21}$$

where $r_a \in \{r_{\text{pick}}, r_{\text{throw}}, r_{\text{move}}\}$ depends on action a. This choice directly affects the estimates of the values of the states produced during the rollout. In fact, after every myopic decision, the estimates are recursively defined as:

$$\bar{V}(s_i) = C_i(s_i, a_i^\star, w_{i+1}) + \gamma\bar{V}(g_{t_i+\Delta t}(s_i, a_i^\star, w_{i+1})). \tag{22}$$

As for the number of recursion steps to perform, denoted by R, its choice may vary depending on the problem: in general R should increase with the problem size. Specifically, we noticed that an increase in the value of parameter R does not negatively affect the performances of the algorithm on small scale problems, but, if significant, it may worsen them for larger scale problems. Supposedly, the further in the future we myopically look, the less accurate are the values produced by the MR. Accounting for the attentive considerations, we set $R = 10$ for all problem's instances.

Of course, once the rollout approaches the time horizon or reaches a terminal state it cannot proceed, and a precise value must be assigned to the terminal state reached. To this aim, we employ the same terminal value function defined in (18) for DP.

Finally, we conclude the subsection by summarizing the overall decision-making procedure through MR Pseudo-Algorithm 1.

Algorithm 1 Decision making through MR

```
procedure BestDecision(s)
    BestValue ← 0
    for a ∈ A_s do
        s' = g_{t_i+Δt}(s, a, 1)                ▷ State associated to action's success
        V' = MyopicRollout s', 0
        s'' = g_{t_i+Δt}(s, a, 0)               ▷ State associated to action's failure
        V'' = MyopicRollout s'', 0
        V = P(1)(C_i(s, a, 1) + V') + P(0)(C_i(s, a, 0) + V'')
        if V > BestValue then
            BestValue ← V
            BestAction ← a
        end if
    end for
    return BestAction
end procedure

procedure MyopicRollout(s, r)
    a* = (21)
    if r ≤ R then
        Exogenous factor realization. Simulate outcome w
        s' = g_{t_i+Δt}(s, a*, w)
        RealContribution = C_i(s, a*, w)
        r ← r + 1
        if s is not terminal then
            V' = MyopicRollout s, r
            V = RealContribution + γV'
        else V = F(s)
        end if
    else V = C_i^M(s, a*)
    end if
    return V
end procedure
```

4.2.2. Monte Carlo tree search

MCTS is a search method based on a randomized exploration of the state space. Its algorithm uses the results of previous explorations

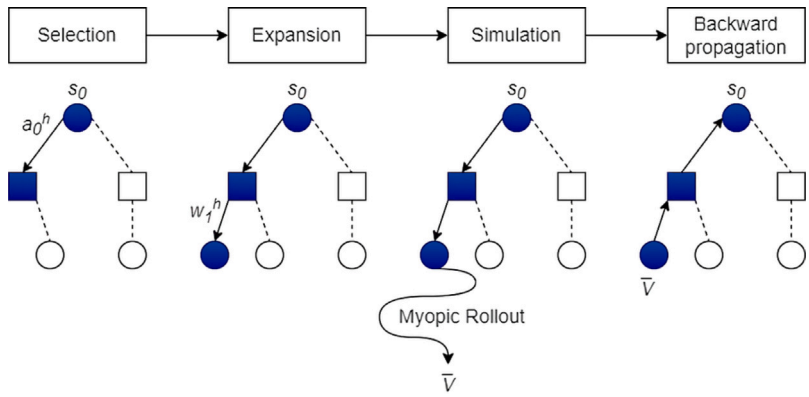

Fig. 4. The main steps of a MCTS at arbitrary iteration h. Round and square nodes represent pre-decision and post-decision states respectively. All white nodes represent states visited during previous iterations.

to gradually build up a tree in memory, hence it progressively becomes better at accurately estimating the values of the most promising actions (Winands, 2015).

As the name implies, the search is performed by means of a gradually constructed decision tree, but before introducing the overall procedure of building it, let us define its nodes. There are two types of nodes in a decision tree: the decision nodes, at which decisions are made, and the outcome nodes, at which new random information becomes available. In a DP context like the one we are dealing with, the decision nodes identify the standard states of the system, while the outcome nodes represent the post-decision states. A post-decision state is the state that the system intends to reach when a specific action is performed, as if there were no exogenous risk factors. For example, in our problem, the choice of moving action $a \in \mathcal{A}_{move}$ when in state $s_i | s_i^1 = t_i, s_i^2 = n$, is made with the intention of reaching state $s_{i+1} | s_{i+1}^1 = t_i + \Delta t_e, s_{i+1}^2 = a, e = (n, a) \in \mathcal{E}$, which is indeed a post-decision state, more precisely denoted by s_i^a to emphasize the dependence on chosen action a. After reaching a post-decision state, information on the realization of external factors becomes available and determines the actual transition to another standard state, that we refer to as pre-decision state. In the previous example, the next pre-decision state may be equal to the post-decision one, if no collision takes place during navigation, or it can result in $s_{i+1} \neq s_i^a$ when a failure occurs.

From now on, in our MCTS, the transition from a pre-decision state s_i (decision node) to the next one s_{i+1} is thus divided into two steps: first, an action a_i is chosen and the algorithm transitions to a post-decision state s_i^a (outcome node) following the transition equations defined in (4)–(10), (13), (14), with $w_{i+1} = 1$ and s_i^a replacing s_{i+1}); then, after the random realizations of the risk factors, it proceeds according to the same transitions, for which we use the novel general notation:

$$s_{i+1} = g^a_{t_i+\Delta t}(s_i, a_i, w_{i+1}). \tag{23}$$

The algorithm always follows four main steps iteratively. In fact, after having identified as the tree root the state at which the robot needs to choose the best action to perform, the MCTS begins and repeatedly undergoes the phases of selection, expansion, simulation and backpropagation (see Fig. 4).

Below a precise and problem-driven description for arbitrary iteration h.

1. **(Selection)** It aims at selecting the most suitable action to perform at a pre-decision state s_i^h, in order to keep exploring the tree. When the number of children for the decision node identified by s_i^h is null, the first action selected is the one with the highest deterministic immediate contribution. This initial choice, although quite myopic, guarantees that the effects of the apparently most appealing action are explored at any cost. Then, if the number of children is less than a fixed allowed offspring limit, the action is chosen among the available ones. This decision is based on a one-step simulation followed by a MR, as in:

$$a_i^{h,\star} = \arg\max_{a \in \mathcal{A}^h_{s_i}} C_i(s_i^h, a) + \textsc{MyopicRolloutg}_{t_i+\Delta t}(s_i^h, a, w_{i+1}^h). \tag{24}$$

On the other hand, if the offspring limit has already been reached in earlier iterations, the action is chosen among the previously visited ones, collected in $\bar{\mathcal{A}}^h_{s_i}$, according to the Upper Confidence bounding for Trees (UCT) (Kocsis & Szepesvári, 2006):

$$a_i^{h,*} = \arg\max_{a \in \bar{\mathcal{A}}^j_{s_i}} \hat{Q}(s_i^h, a) + \epsilon \sqrt{\frac{2 \ln N(s_i^h)}{N(s_i^{a,h})}}. \tag{25}$$

The exploration coefficient ϵ and the number of visits $N(s_i^h)$ and $N(s_i^{a,h})$ of the decision node identified by s_i^h and of the outcome node identified by $s_i^{a,h}$ respectively, define an exploration term voluntarily biased towards post-decision states that have been visited less frequently. On the contrary, the term $\hat{Q}(s_i^h, a) = C_i(s_i^h, a) + \hat{V}^a(s_i^{a,h})$, where $\hat{V}^a(s_i^{a,h})$ indicates the approximate value assigned to post-decision state s_i^h up until iteration h, steers the choice towards actions so far considered more promising.

2. **(Expansion)** Right after the action selection, it comes the expansion phase, whose procedure differs depending on earlier explorations. In fact:

   - if the selected action has never been tried before, the outcome node corresponding to the post-decision state is created. Then, an outcome is uniformly sampled among the available ones and the corresponding pre-decision state is created. At this point, the search enters its next phase;
   - if the selected action has already been tried, there are two further distinct situations:
     - all outcomes have been visited. In this case, an outcome simulation is performed, that will bring the search to a next pre-decision state, from which a new selection phase will begin;
     - not all outcomes have been visited. Therefore, an outcome is uniformly sampled among the ones not yet explored, and the corresponding next pre-decision state is created, leading the search to its next phase.

   Note that, while a limit is set for the number of actions to try at each state, i.e., for the offspring of the corresponding decision node, all outcomes are potentially explored. The choice is due to the fact that, in our specific problem, the admissible outcome

space given a state is at most binary. For this same reason, there is no negative effect in uniformly sampling the outcomes during the expansion phase, actually we think it may fasten the initial exploration.
Nevertheless, after having sampled all the possible realizations given an action, external risk factors are simulated according to their real probability distributions.

3. **(Simulation)** Whenever a new pre-decision state is created during expansion, this last phase stops and a simulation begins: a value estimated through a MR is associated to the state representing the new leaf node.
4. **(Backpropagation)** During this last phase of the MCTS, the newly simulated value associated to the newly created leaf node is back-propagated towards the parent-node, iteratively until the root, following the path sampled during the previous phases of the current iteration, h. In the meanwhile, also the counters of the number of visits for each node in the path are updated. The overall procedure is illustrated in Pseudo-Algorithm 2, where $\bar{\mathcal{W}}_a^h$ denotes the set of outcomes visited after the play of action a up until iteration h, and is similar to the approach proposed in Powell (2019), but specifically adapted for single temporal step updates.

Algorithm 2 Backpropagation phase of MCTS

procedure BackPropagation(s_i^h)
  $N(s_i^h) \leftarrow N(s_i^h) + 1$
  while $s_{i-1}^{a,h}$ is **not** null **do**
    $N(s_{i-1}^{a,h}) \leftarrow N(s_{i-1}^{a,h}) + 1$
    $\hat{V}^a(s_{i-1}^{a,h}) = \frac{1}{\sum_{w \in \bar{\mathcal{W}}_a^h} \mathbb{P}(w)} \sum_{w \in \bar{\mathcal{W}}_a^h} \mathbb{P}(w) \hat{V}(g_{t_i}^a(s_{i-1}^h, a_{i-1}^h, w))$
    reward $= C_{i-1}(s_{i-1}^h, a_{i-1}^h, w_i^h)$
    delta $= \hat{V}^{a,h}(s_{i-1}^{a,h})$
    $\hat{V}(s_{i-1}^h) \leftarrow \hat{V}(s_{i-1}^h) + \frac{(\text{delta} - \hat{V}(s_{i-1}^h))}{N(s_{i-1}^h)+1}$
    BackPropagations_{i-1}^h
  end while
end procedure

Once the four phases of the search are repeated for a fixed time of iterations, ($H = 50$ in our experiments), the policy, guiding the choice of the best action to perform when in state s_{i_0}, corresponding to the root node of the just created Monte Carlo Tree, is:

$$\pi^\star(s_{i_0}) = \arg\max_{a \in \mathcal{A}_{s_{i_0}}} \hat{Q}(s_{i_0}, a) = \arg\max_{a \in \mathcal{A}_{s_{i_0}}} C_i(s_{i_0}, a) + \hat{V}^a(s_{i_0}^a). \tag{26}$$

Let us conclude by focusing in detail on a couple of essential hyperparameters that have been mentioned when explaining the MCTS phases: the exploration parameter ϵ and the offspring limit, henceforth denoted by ρ.

There are two situations that are preferably to be avoided during a MCTS, and that can be partially dodged with a reasonable tuning of such parameters. The first inconvenient situation consists in iteratively lowering the approximate value of a state, up until its exclusion from further exploration, although it may lead to a very promising future state when the correct action is selected. In fact, during backpropagation, the value of a state can be repeatedly compromised by the values of its other, less promising offspring, if numerous. On the contrary, the algorithm might become interested in visiting states that only appear as favorable, when it neglects less frequently visited, yet better, actions.

The first scenario is likely to happen when ϵ and ρ are set to elevate values, whilst the second is mainly caused by the lowering of the former. Thus, it is naturally inferred that the two parameters shall be antithetically fixed: an excessive exploration factor should be accompanied by a conservative offspring limit, and viceversa. We opted for the first alternative and set $\epsilon = 3.5$ and $\rho = 5$ in all our experiments, conscious of the necessity of increasing both of them, and consequently the number of iterations H, when dealing with significantly larger scale problems.

Table 1
List of the orders considered for the Small Instance[1]. Each order is characterized by the quantity of items required for each object type.

	Obj 1	Obj 2	Obj 3	Obj 4
Order 1	1	1	1	0
Order 2	3	0	0	1
Order 3	0	3	0	0
Order 4	2	0	1	1

5. Experimental results

In this section we illustrate the performances of the methods presented. First of all, we validate the approximate approaches by comparing their results on small instances to the ones output by the exact DP paradigm described in Section 4.1. Then, since the latter cannot be applied to larger-scale problems, the analysis on larger instances is only performed for the approximate methods.

In order to understand what we mean by "large instance", it is important to recall the nature of our problem setting, which was pointed out in Section 3.1. We deal with a limited lookahead to be able to assume a static view of the risk factors related to collisions. If, while solving a problem instance, these risk factors change to the point that the graph $\mathcal{G}$ must be updated, we must define and solve a new problem instance. Hence, dealing with extremely large instances and an extended lookahead would be pointless. For this reason, the larger problem instances that we will consider may not be *large* in an absolute way, but we will refer to them as such because they are relative to the small ones (and to draw a line between what we may tackle by exact DP and what we cannot).

For experimental purposes, for each instance we define K orders and collect them in a set $\mathcal{K}$. We then consider a mission complete when either all orders have been served or the time horizon is reached, whichever occurs first.

We mainly focus on variations of the following two instances of the problem.

1. Small Instance:
   - $K = 4$ orders, listed in Table 1,
   - total number of items to collect $Q = 14$,
   - time horizon $T = 300$ seconds,
   - $O = 4$ object types,
   - $D = 2$ trays,
   - $|\mathcal{N}| = 6$ graph vertices;
2. Large Instance:
   - $K = 8$ orders, listed in Table 2,
   - total number of items to collect $Q = 42$,
   - time horizon $T = 1000$ seconds,
   - $O = 5$ object types,
   - $D = 3$ trays,
   - $|\mathcal{N}| = 8$ graph vertices.

All experiments account for a maximum robot capacity of $c = 4$ and are conducted with fixed deterministic rewards $r_{\text{pick}} = 10, r_{\text{throw}} = 12$, used as multiplicative factors in immediate contributions.

Table 2
List of the orders considered for the Large Instance[2]. Each order is characterized by the quantity of items required for each object type.

	Obj 1	Obj 2	Obj 3	Obj 4	Obj 5
Order 1	1	2	1	0	0
Order 2	3	0	0	1	2
Order 3	0	3	0	0	2
Order 4	0	0	3	3	0
Order 5	2	0	0	2	1
Order 6	2	0	1	1	1
Order 7	0	1	2	1	2
Order 8	4	0	1	0	0

5.1. Evaluation metrics

We present a set of experiments that aim at evaluating different characteristics of the results obtained by the different methods. Therefore, we first compare the approaches for a fixed sequence of orders, in terms of both times of arrival and associated priority levels. This choice allows us to reduce variance in the estimates and to focus on the intrinsic value of the compared methods. Then, a more general comparison of the methods regards the average results of a series of experiments for which the orders' arrival in the system and the assignment of a priority level are stochastic, as described in Section 3.2.1. This second experiment is carried out to assess the robustness of the methods.

Moreover, for all experiments we adopt two different ways for evaluating the results: the first consists of measuring the average time taken for the robot to fulfill an order. The time is either measured starting from the instant the order arrives into the system or the instant the order is assigned to an available tray, hence enters the mission. Respectively, the average waiting time since arrival V_a and the average waiting time since entrance V_e are defined as:

$$V_a = \frac{1}{\bar{K}} \sum_{k \in \bar{K}} T_k^c - T_k^a, \tag{27}$$

$$V_e = \frac{1}{\bar{K}} \sum_{k \in \bar{K}} T_k^c - T_k^e, \tag{28}$$

where $\bar{\mathcal{K}} \subseteq \mathcal{K}$ is the set of orders served before reaching the time horizon, T_k^c is the time instant at which order k is completed and T_k^a and T_k^e are respectively the arrival time and entrance time of order k. Of course, the smaller the time taken the more efficient the scheduling.

The other kind of assessment somehow measures the with priority levels, in the sense that the sequence S_e of orders assigned to an available tray, sorted by time, should coincide with the sequence S_c in which the orders are fulfilled. In scheduling terms, this is related with the minimization of work in progress. To this purpose, we introduced two evaluation metrics. The first one is based on the maximum shift in the two sequences, according to

$$V_{max} = \max_{k \in \bar{\mathcal{K}}} |i_k - j_k|, \tag{29}$$

while the second one defines a value based on the overall *late* shifting, as

$$V_{overall} = K - \sum_{k \in \bar{\mathcal{K}}} 0.5|_{j_k = i_k + 1} - \sum_{k \in \bar{\mathcal{K}}} (j_k - i_k - 1)|_{j_k > i_k + 1}, \tag{30}$$

where i_k and j_k are the positions of order k in sequences S_e, S_c, respectively. We wish for the value defined by (29) to be close to zero and for the one defined by (30) to be close to the total number of orders K.

Finally, of course, also algorithms execution times are analyzed and considered for deciding for the best algorithm overall.

5.2. Experimental setting

The setting considered is the one described in Section 3. Prior the beginning of the scheduling the first D orders defined for a specific instance are immediately assigned to the respective D trays available, hence their arrival time and entrance time in the system are set to 0. Furthermore, the positions in the entrance sequence S_e of such first D orders are all set to 1. Coherently, also the first D elements of the completion sequence S_c are associated to the position 1. For all others the indexing follows natural enumeration. If not specified otherwise, the $K-D$ remaining orders arrive into the system according to a Poisson process whose rate of arrival is $\lambda = \frac{K-D}{T}$.

Table 3
Summary of the hyperparameter values used in our experiments.

Method	γ	λ	R	ϵ	ρ	H
DP	1	$\frac{K-D}{T}$	–	–	–	–
MR	0.95	$\frac{K-D}{T}$	10	–	–	–
MCTS	0.95	$\frac{K-D}{T}$	10	3.5	5	50

All parameters values used for our experiments have been clarified so far, except for the discount factor γ: its value for the exact DP paradigm equals 1 while is set to 0.95 during application of the MR (and consequently of the MCTS). You can refer to Table 3 for a concise summary of the hyperparameter values set for all methods.

Please note that experiments are repeated a considerable number of times,[4] hence the presented results always refer to the mean values obtained over the multiple runs.

5.2.1. Experiments with a fixed sequence of order arrivals

One way we adopt to evaluate the implemented approaches is to compare them to each other when the sequence of arrival of the orders and their associated priority levels are fixed. To this purpose, an homogeneous arrival gap between the last $K - D$ orders is defined as equal to $\frac{T}{6}$, while the first D orders arrive into the system at time 0, as in the general stochastic-arrival case described so far. Furthermore, in order to work with the same priority levels throughout all the repeated runs regarding this experiment, we generated $K - D$ random levels of priority $\in \mathcal{L}$, to associate to the last $K - D$ orders, after having fixed a specific seed.

5.3. Results

Having clarified the instances on which experiments are conducted, the experimental setting, and the values fixed for the parameters, we are ready to present the results of our research.[5]

5.3.1. Small instance

As previously mentioned, conducting experiments on small instances of the problem is essential to validate the approximate methods with respect to the exact DP paradigm, which cannot be applied to larger scale problems due to its computational expense. Therefore, the first results we present refer to Instance 1 and are collected in Tables 4 and 5. The former exhibits the average values of the evaluation metrics for experiments conducted with fixed sequence of arrival of the orders, while the latter shows the same average evaluations for experiments conducted with a stochastic sequence of arrival.

As expected, in both cases, the exact DP paradigm performs better than the two approximate methods in terms of average time to fulfill an order once it has arrived into the system and/or entered the mission. Instead, it is quite surprising that, despite the longer average order's waiting time, serving priority is more respected by approximate methods. Moreover, by observing the results in Table 5, one can notice that

[4] The number of runs repeated to conduct experiments differ for each method and depend on its execution time. For the MR we repeated at least 30 runs for each experiment, for the DP and MCTS 15.

[5] For code and implementation details, please refer to https://github.com/margheritabattistotti/opt_robot_scheduling_with-lookahead-basedADP.git.

Table 4
Comparison of the performances of our approaches on Instance 1. All runs are executed with the same fixed sequence of arrival and priority levels of the orders.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
DP	98	95	1.20	3.40	58.37s
MR	153	123	0.80	3.63	0.13s
MCTS	130	122	0.06	3.96	6.11s

Table 5
Comparison of the performances of our approaches on Instance 1. All runs are executed with different stochastic sequences of arrival and priority levels of the orders.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
DP	78	76	0.33	3.83	39.12s
MR	128	116	0.16	3.91	0.13s
MCTS	142	126	0.05	3.975	5.79s

Table 6
Comparison of the performances of our approaches on Instance 2. All runs are executed with the same fixed sequence of arrival and priority levels of the orders.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
MR	246	234	0.83	7.55	0.99s
MCTS	250	236	0.6	7.63	38.97s

an higher compliance between the two sequences S_e and S_c, of orders entering the mission and being completed respectively, may suggest a small sacrifice in efficiency.

We can conclude that both approximate approaches present overall valuable results and can be considered as valid techniques to apply to our problem for its resolution. In fact, in the case of a fixed sequence of arrival, for which we analyze a specific scenario and thus are able to make direct comparisons, the approximate methods present, with respect to the exact DP, a delay of less than 30 seconds in the average serving time of an order once it has entered the mission (V_e). The contained nature of the delay is confirmed by the more robust analysis on experiments with varying sequences of arrival, although direct comparisons may be harder to make due to the dependence of the results on the associated the generated sequences.

As for the resolution speeds, we must present some considerations prior diving into conclusions. Note that the exact DP's resolution time technically coincides with the time needed to perform the backward passes. This means that once a computationally expensive backward pass is performed, having listed all the values associated to all possible states of the system, decisions are then taken instantly, even when failures occur. In our case, such pass must be repeated whenever an order in the queue is assigned to a newly available tray; with longer time horizons and many more orders to expect, it becomes indeed intractable to follow this approach. Nonetheless, for small instances with short time horizons like the one presented it proves to be the best alternative although its highest overall resolution time. On the other hand, both MR and MCTS execution times refer to the total time taken for the actual scheduling of a simulated scenario. In fact, the methods respectively require the application of a rollout and the construction of a tree whenever a decision must be made; this means that the average time to make a decision is negligible in the case of MR and extremely contained for the MCTS. Therefore, both approaches represent valuable faster alternatives to the DP paradigm, especially when it comes to larger scale problems, as further proved in the next section.

5.3.2. Large instance

Once validated the approximate approaches through comparisons with the exact DP paradigm, it is now time to analyze their performances on a larger scale. The section does not present experimental results for the DP paradigm since its application to problem Instance 2 would cause memory overload,[6] as predictable. In fact, Tables 6 and 7 show the results regarding the two sets of experiments performed with fixed and stochastic sequence of arrival, referred only to the two approximate methods.

Table 7
Comparison of the performances of our approaches on Instance 2. All runs are executed with different stochastic sequences of arrival and priority levels of the orders.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
MR	308	268	1.20	7.16	0.81s
MCTS	301	260	1.00	7.42	42.22s

The algorithms cause similar average waiting times per order, although the MCTS presents lower values of V_{max} and higher values of $V_{overall}$, suggesting a better compliance in terms of meeting priority levels, as in the previously analyzed smaller case. Recall that the MCTS performs a search on an iteratively constructed tree, exploring various future scenarios, hence leading to a more accurate and varied lookahead with respect to the MR, whose application guarantees the exploration of one only among the possible future paths. Moreover, such forward MR exploration is based on deterministic immediate contributions while real contributions are only employed for the definition of the final myopic value to associate to a state. On the other hand, not only during backpropagation the MCTS employs the real prioritized rewards, but also in the selection phase. Therefore, we may justify the results with such motivations.

For what concerns execution times, the same considerations made in the previous section hold for this one as well. In fact, the MR is definitely faster than the other method and its decision making process is almost instant. However, although less evident, the MCTS is also extremely time-efficient: for example, if we compute the time taken for making a single decision during the simulation of the fixed sequence of arrival scenario we get approximately 0.22 s, given it takes an average of 173 actions to complete the mission.

In conclusion, despite the MR presents lower waiting times for the orders in the fixed-sequence scenario, the more robust analysis performed on the scenarios with stochastic sequences of arrival proves that the MCTS is generally slightly preferable. However, the two methods perform very similarly and can be used interchangeably, unless specific requirements on execution speeds must be satisfied.

5.4. Further experiments

In the previous sections, we stressed the fact that, given our setting, we inevitably have to roll the horizon forward and solve the problem again when risk factors change significantly. Assuming such an adaptation is a relatively frequent task to perform, in we have limited our main experiments to instances that are only *relatively* large, because a longer lookahead would not make too much sense under such an assumption. However, for the sake of completeness, we believe it is valuable to assess the behavior of our approaches on longer time horizons, assuming, this time, that risk factors do not change as frequently and, thus, we can schedule ahead more orders. To this aim, we present a further experiment, labeled as *extra-large* instance, characterized as follows:

Instance 3:

- $K = 100$ orders, sampled at random from the ones in Table 2;
- total number of items to collect $400 \leq Q \leq 600$, depending on the sampling:
- time horizon $T = 12500$ seconds;
- $O = 5$ object types;
- $D = 3$ trays;

[6] On a machine equipped with RAM 16 GB.

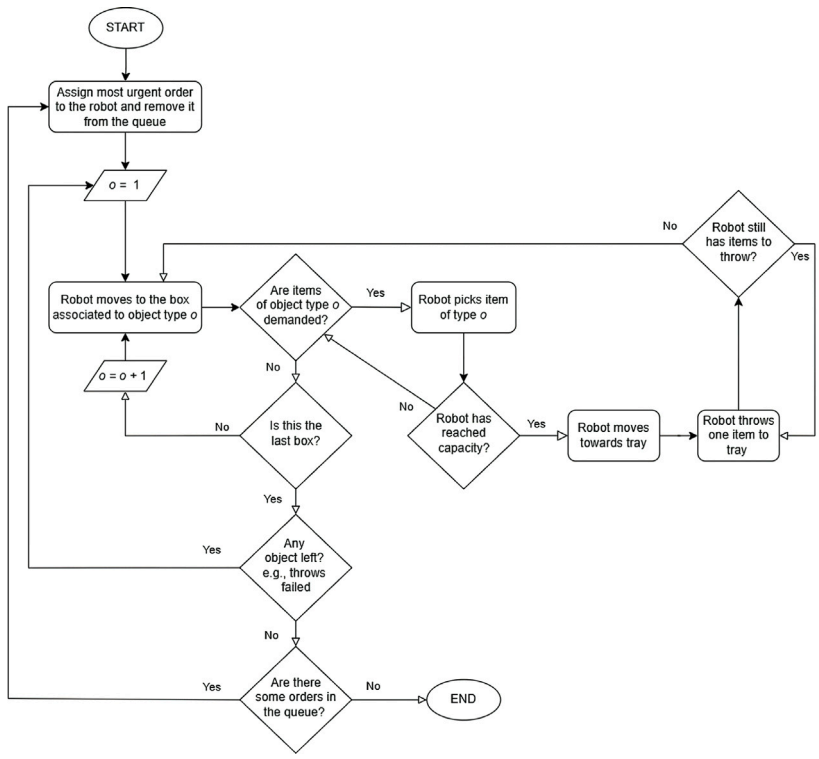

Fig. 5. Flowchart illustrating the sequential heuristic procedure. The condition on not having reached the time horizon is implicit: it is not shown in the flowchart but it is intrinsically verified at every step.

- $|\mathcal{N}| = 8$ graph vertices.

All hyperparameter values are set as in the previous experiments, except for the homogeneous arrival gap, adjusted to $\frac{T}{K-D+1}$, due to the longer list of orders to be completed. One may observe that we do not change the parameters R of the MR and ϵ and ρ of the MCTS, but we should note that keeping the same structure of the graph does not affect the local decision-making step of the lookahead approaches. In order to assess their quality, we compare the results of our approximate methods to those of a simple sequential heuristic that serves orders one at a time ($D = 1$). The heuristic procedure is illustrated by the flowchart in Fig. 5. Each time a robot is assigned an order, it is initially guided towards a box in a boundary location (i.e., the first or the last one). From there, it starts picking items of the object type associated with the box if any are demanded, otherwise it moves to the next adjacent box. When it reaches its capacity, it quits picking and moves towards the only tray in use to unload all the items collected. After the entire load has been thrown to the assigned tray, if there are missing items to meet the order's demand, the robot returns to the box from where the picking was interrupted. When the mission is completed, the robot recedes to the initial box location to repeat the procedure for the immediately succeeding order. The process is repeated for all orders in the queue or until a fixed time horizon is reached. The approach may be interpreted as a simple priority rule to sequence tasks associated with orders, much like dispatching rules in machine scheduling.

Let us start by analyzing Table 8, which shows the results of our experiments conducted on the extra-large Instance 3 when having fixed a sequence for the arrival of the orders and their priority (see Section 5.2.1). The MCTS presents lower average waiting times for the orders and better compliance with priority levels when compared to MR. We can draw similar conclusions when our methods deal with a stochastic sequence of arrival, for which results are reported in Table 9. Additionally, we note that their average waiting times with respect to arrival V_a and entrance V_e increased in the latter scenario with respect to the fixed-sequence one and that the compliance with priority levels is worse too. This behavior is already visible when analyzing the results of our approaches on (large) Instance 2, while it does not hold for (small) Instance 1. We conclude that our methods tend to struggle on longer time horizons when dealing with stochastic sequence of arrivals as more uncertainty is introduced.

Table 8
Comparison of the performances of our approaches on Instance 3. All runs are executed with the same fixed sequence of arrival and priority levels of the orders. The gap between arrivals was set to $\frac{T}{K-D+1} \approx 127$ seconds.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
MR	175	169	2.53	93.96	16.89s
MCTS	157	148	1.8	97.1	798.49s
Heuristic	144	115	0	100	0.43s

Both tables show that the average waiting time with respect to entrance V_e is lower when the heuristic is used. This is no surprise, as the heuristic deals with one order at a time, while our methods try to serve multiple orders simultaneously. For this reason, we are more interested in the average waiting time with respect to arrival V_a to make comparisons. This value is lower for the heuristic in the fixed-sequence case, but higher in the case of stochastic sequence of arrival, and suggests our methods present results similar to a simple heuristic. However, note that the value of V_e for the heuristic reflects the average time needed to serve an order once it is assigned to a tray, that is 115 seconds. In these experiments we set the fixed arrival gap to a value of approximately 127 seconds: when following the heuristic procedure, the robot has plenty of time to serve an order before another one arrives. In such a framework, the system is not stressed and the advantages of our methods are not shown. We repeated the experiments with a lower fixed arrival gap of 100 seconds and a higher Poisson rate equal to 0.01. Tables 10 and 11 confirm our intuition that, under a more stressful situation, our methods guarantee an average waiting time with respect to arrival lower than the one presented by the sequential heuristic.

Table 9
Comparison of the performances of our approaches on Instance 3. All runs are executed with different stochastic sequences of arrival and priority levels of the orders. The Poisson rate was set to $\frac{K-D}{T} \approx 0.008$.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
MR	458	276	3.6	80.16	18.98s
MCTS	412	262	3.6	86.7	703.26s
Heuristic	494	115	0	100	0.33s

Table 10
Comparison of the performances of our approaches on Instance 3. All runs are executed with the same fixed sequences of arrival and priority levels of the orders. The gap between arrivals was set to 100 seconds.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
MR	812	324	4.26	74.3	13.27s
MCTS	661	319	2.8	80.4	798.49s
Heuristic	835	115	0	100	0.48s

Table 11
Comparison of the performances of our approaches on Instance 3. All runs are executed with different stochastic sequences of arrival and priority levels of the orders. The Poisson rate was set to 0.01.

Resolution method	V_a	V_e	V_{max}	$V_{overall}$	Execution time
MR	918	312	4.0	76.55	12.99s
MCTS	913	311	3.0	82.1	612.99s
Heuristic	1174	115	0	100	0.48s

In the end, all these further experiments reinforce the conclusions drawn in the previous sections about the better performances reached by the MCTS when compared to MR. Nevertheless, they are both valuable alternatives to exact methods and heuristics, for their trade-off between solution quality and runtime. In fact, concerning runtime, we recall that both MR and MCTS execution times refer to the total time taken for the actual scheduling of a simulated scenario. Hence, on average, the MCTS only took 0.3 s per decision and the MR even less, given that during their execution they scheduled more than 2000 actions. However, we should note that for other instances, larger in terms of graph structure (number of trays and boxes), a MCTS would require a higher number of iterations for the construction of a decision tree with the same lookahead of MR – due to the extended action space – and this could result in slower decision making.

6. Conclusions

Throughout the text we have discussed the application of the Dynamic Programming paradigm, especially of its Approximate counterpart, on a specific intralogistic robot scheduling problem. We envisioned a scenario where a single agent is tasked with the transportation of objects from boxes to trays situated in a warehouse. We modeled the warehouse as a completely connected undirected graph with vertices corresponding to box locations, where the robot can pick the various object types, and other vertices corresponding to locations from where the robot can throw the collected items into the trays. We considered such trays as assembly spots for specific orders that randomly arrive into the system, requesting multiple object types in different quantities. We let the orders being associated with a (randomly assigned) priority level, essential for the definition of a sorted queue determining the assignment of an order to an available tray, if any. The limited number of assembly spots implied the definition of a dynamic mission collecting the various jobs assigned to the available trays. Finally, we considered the system to be subjected to static risk factors associated to moving and throwing actions and distributed as Bernoulli random variables. Then, the problem's main objective was to define a time-efficient and risk-aware scheduling of tasks for the robot, respecting natural priority rules.

It is well-known that the exact DP paradigm provides optimal scheduling of tasks but suffers from curses of dimensionality; on the other hand, approximate methods output less optimal decisions but are generally more time-efficient. Guided by this knowledge, we had the intuition of applying the exact paradigm only to small instances of the problem described above, and used the results as benchmarks for the validation of two lookahead policies: Myopic Rollout and Monte Carlo Tree Search. Indeed, extensive experiments conducted on small instances showed a significant difference between the execution times of the approximate approaches and the ones of the exact paradigm, without significantly compromising performances. Given the promising results on small instances, we repeated various experiments on larger instances for approximate methods only. By evaluating the average waiting times of the orders and the compliance with priority levels, we were able to confirm that Myopic Rollout and Monte Carlo Tree Search are valid techniques for the resolution of dynamic scheduling problems, like the one at hand.

In conclusion, the approaches we presented have not been previously explored in the literature, making our work a valuable contribution. As the automation of warehouses and/or fulfillment centers is currently a compelling topic for major technological companies, our findings offer significant insights in this area.

6.1. Further developments

Further developments could focus on even larger instances of the problem, on the automated tuning of the hyperparameters here manually chosen, on the implementation of speed-enhancing techniques, and on the effects of considering dynamic risks.

In fact, we treated risk factors as static in nature and thus maintained them fixed throughout our simulations. However, over longer time horizons, it is plausible that they may dynamically change, particularly those linked to moving actions involving collisions with humans, who are mobile as well. A future topic of research could indeed involve more extensive experimentation with dynamic risks and incorporate a suitable rolling procedure to our scheduling problem.

Regarding the implementation of the algorithms we opted for native Python language. However, speed-enhancing techniques, as leveraging Python packages like Numba, could be employed to further optimize execution times. Nevertheless, our choice of programming language does not compromise the validity of our conclusions: our experiments enable valuable comparisons under a time perspective despite the non-optimal runtimes.

CRediT authorship contribution statement

Margherita Battistotti: Conceptualization, Investigation, Methodology, Software, Writing – original draft. **Paolo Brandimarte:** Methodology, Supervision, Writing – review & editing. **Francesca Giancola:** Conceptualization, Supervision, Writing – review & editing. **Nicolò Mazzi:** Conceptualization, Project administration, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

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

Acknowledgment

This work has been partially funded by the European Union's Horizon 2020 research and innovation programme, under grant agreement No 101017274. See https://darko-project.eu/about/ for more information about the DARKO project.

Data availability

No data was used for the research described in the article.

References

Allgor, R., Cezik, T., & Chen, D. (2023). Algorithm for robotic picking in Amazon fulfillment centers enables humans and robots to work together effectively. *INFORMS Journal on Applied Analytics*, *53*, 266–282. http://dx.doi.org/10.1287/inte.2022.1143.

Battistotti, M. (2024). *Dynamic optimization of intralogistic robot scheduling*. Master's Degree in Mathematical Engineering: Politecnico di Torino -, URL http://webthesis.biblio.polito.it/id/eprint/30370.

Bertsekas, D. P. (2019). *Reinforcement learning and optimal control*. Athena Scientific.

Bertsekas, D. P. (2020). *Rollout, policy iteration, and distributed reinforcement learning*. Athena Scientific.

Bertsekas, D. P., & Castanon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, *5*, 89–108.

Bertsekas, D. P., Tsitsiklis, J. N., & Wu, C. (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, *3*, 245–262.

Boysen, N., & de Koster, R. (2024). 50 years of warehousing research—An operations research perspective. *European Journal of Operational Research*, http://dx.doi.org/10.1016/j.ejor.2024.03.026.

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

Boysen, N., de Koster, R., & Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, [ISSN: 0377-2217] *277*(2), 396–411. http://dx.doi.org/10.1016/j.ejor.2018.08.023, URL https://www.sciencedirect.com/science/article/pii/S0377221718307185.

Boysen, N., Schwerdfeger, S., & K., S. (2023). A review of synchronization problems in parts-to-picker warehouses. *European Journal of Operational Research*, *307*, 1374–1390.

Boysen, N., Schwerdfeger, S., & Ulmer, M. W. (2023). Robotized sorting systems: Large-scale scheduling under real-time conditions with limited lookahead. *European Journal of Operational Research*, *310*(2), 582–596.

Brandimarte, P. (2021). *From shortest paths to reinforcement learning: A MATLAB-based introduction to dynamic programming*. Springer.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*, 1–43.

Cheng, C.-Y., Chen, Y.-Y., Chen, T.-L., & Jung-Woon Yoo, J. (2015). Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. *International Journal of Production Economics*, *170*, 805–814.

Geist, M., & Pietquin, O. (2013). Algorithmic survey of parametric value function approximation. *IEEE Transactions on Neural Networks and Learning Systems*, *24*, 845–867.

Goodson, J. C. (2013). Rollout policies for dynamic solutions to the multivehicle routing problem with stochastic demand and duration limits. *Operations Research*, *61*, 138–154.

Justkowiak, J. E., Kovalyov, M. Y., & Pesch, E. (2024). A dynamic programming algorithm for order picking in robotic mobile fulfillment systems. *Networks*, *84*, 481–490. http://dx.doi.org/10.1002/net.22245.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In J. Fürnkranz, T. Scheffer, & M. Spiliopoulou (Eds.), *Machine learning: ECML 2006* (pp. 282–293). Springer, http://dx.doi.org/10.1007/11871842_29.

Li, H., & Womer, N. K. (2015). Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *European Journal of Operational Research*, *246*, 20–33.

Löffler, M., Boysen, N., & Schneider, M. (2022). Picker routing in AGV-assisted order picking systems. *INFORMS Journal on Computing*, *34*, 440–462. http://dx.doi.org/10.1287/ijoc.2021.1060.

Löffler, M., Boysen, N., & Schneider, M. (2023). Human-robot cooperation: Coordinating autonomous mobile robots and human order pickers. *Transportation Science*, *57*, 979–998. http://dx.doi.org/10.1287/trsc.2023.1207.

Pan, J. C.-H., Shih, P.-H., & Wu, M.-H. (2015). Order batching in a pick-and-pass warehousing system with group genetic algorithm. *Omega*, *57*, 238–248.

Powell, W. B. (2009). What you should know about approximate dynamic programming. *Naval Research Logistics*, *56*, 239–249.

Powell, W. B. (2010). Merging AI and OR to solve high-dimensional stochastic optimization problems using approximate dynamic programming. *INFORMS Journal on Computing*, *2*, 2–17.

Powell, W. B. (2011). *Approximate dynamic programming: Solving the curses of dimensionality* (2nd ed.). Hoboken, NJ: Wiley.

Powell, W. B. (2019). *Reinforcement learning and stochastic optimization: A unified framework for sequential decisions*. Wiley.

Powell, W. B., & Simao, B. (2012). Approximate dynamic programming in transportation and logistics: a unified framework. *EURO Journal of Transportation Logistics*, *1*, 237–284.

Schiffer, M., Boysen, N., Klein, P. S., Laporte, G., & Pavone, M. (2022). Optimal picking policies in E-commerce warehouses. *Management Science*, *68*, 7497–7517.

Solomon, S., Li, H., Womer, K., & Santos, C. A. (2019). Multiperiod stochastic resource planning in professional services organizations. *Decision Sciences*, *50*, 1281–1318.

Ulmer, M. W., Soeffker, N., & Mattfeld, D. C. (2018). Value function approximation for dynamic multi-period vehicle routing. *European Journal of Operational Research*, *269*, 883–899.

Ulmer, M. W., & Thomas, B. W. (2020). Meso-parametric value function approximation for dynamic customer acceptances in delivery routing. *European Journal of Operational Research*, *285*, 183–195.

Valle, C. A., Beasley, J. E., & Salles da Cunha, A. (2017). Optimally solving the joint order batching and picker routing problem. *European Journal of Operational Research*, *262*, 817–834.

Winands, M. H. M. (2015). Monte-Carlo tree search. In N. Lee (Ed.), *Encyclopedia of computer graphics and games* (pp. 1–6). Springer, http://dx.doi.org/10.1007/978-3-319-08234-9_12-1.

Xie, F., Li, H., & Zhe, X. (2021). An approximate dynamic programming approach to project scheduling with uncertain resource availabilities. *Applied Mathematical Modelling*, *97*, 226–243.

Zhang, S., Zhuge, D., Tan, Z., & Zhen, L. (2022). Order picking optimization in a robotic mobile fulfillment system. *Expert Systems with Applications*, *209*, Article 118338. http://dx.doi.org/10.1016/j.eswa.2022.118338.

Zhou, C., Stephen, A., Tan, K. C., Chew, E. P., & Lee, L. H. (2024). Multiagent *Q*-learning approach for the recharging scheduling of electric automated guided vehicles in container terminals. *Transportation Science*, *58*, 664–683.