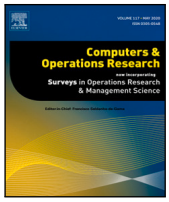
Computers & Operations Research 170 (2024) 106760

Contents lists available at ScienceDirect

Computers and Operations Research

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



Solving the Online On-Demand Warehousing Problem

Sara Ceschia [a,*], Margaretha Gansterer [b], Simona Mancini [c,b], Antonella Meneghetti [a]

[a] Polytechnic Department of Engineering and Architecture, University of Udine, Italy
[b] University of Klagenfurt, Department of Operations, Energy, and Environmental Management, Klagenfurt, Austria
[c] University of Palermo, Department of Engineering, Palermo, Italy

ARTICLE INFO

Dataset link: https://bitbucket.org/sceschia/online-on-demand-warehousing-problem

Keywords:
On-demand warehousing
Combinatorial optimization
Online algorithms

ABSTRACT

In On-Demand Warehousing, an online platform acts as a central mechanism to match unused storage space and related services offered by suppliers to customers. Storage requests can be for small capacities and very short commitment periods if compared to traditional leasing models. The objective of the On-Demand Warehousing Problem (ODWP) is to maximize the number of successful transactions among the collected offers and requests, considering the satisfaction of both the supply and demand side to preserve future participation to the platform. The Online ODWP can be modeled as a stochastic reservation and assignment problem, where dynamically arriving requests of customers must be rapidly assigned to suppliers. Firstly, an online stochastic combinatorial optimization framework is adapted to the Online ODWP. The key idea of this approach is to generate samples of future requests by evaluating possible allocations for the current request against these samples. In addition, expectation, consensus, and regret, and two greedy algorithms are implemented. All solution methods are compared on a dataset of realistic instances of different sizes and features, demonstrating their effectiveness compared to the *oracle* solutions, which are based on the assumption of perfect information about future request arrivals. A newly proposed approach of risk approximation is shown to outperform alternative algorithms on large instances. Managerial insights regarding acceptance and rejection strategies for the platform are derived. It is shown how requests with large demand, long time frame, not very long spanning time, and average compatibility degree, are very likely to be rejected in the optimal solution.

1. Introduction

In many industries, manufacturing companies are experiencing increased price pressure and therefore, are required to reduce fixed costs. One option is to better exploit owned storage capacities and make use of shared storage spaces (Unnu and Pazour, 2023; Ceschia et al., 2023), also known as on-demand warehouses. In on-demand warehousing, available space during low inventory periods is made available to other companies facing storage demand peaks. Compared to traditional warehousing models, such as constructing or outsourcing/leasing a facility, the on-demand one offers: capacity granularity, since storage space is measured in pallet positions rather than square units; commitment granularity, which is based on weekly/monthly periods of space availability or request rather than years; access scalability, i.e. more favorable locations in terms of distance from clients and related transport costs (Pazour and Unnu, 2018).

Several providers of such shared storage spaces are on the market (e.g., *Flexe*[1]). Typically they are run as platform-based companies, where the objective is to enable as many as possible successful matches of customer requests and available supplier capacities. Our study addresses exactly this matching problem in an online setting, which we denote as the Online On-Demand Warehousing Problem (OODWP). In contrast to the offline version introduced in Ceschia et al. (2023), no information on future requests is available during the acceptance or rejection process of dynamically arriving customer requests. For each request two dimensions are considered: duration and capacity consumption. Also, the spanning time (which is the time slack until a newly arrived request has to be serviced) and compatibility between requests and suppliers are taken into account.

This study contributes to the existing literature as follows.

- The decision process is formulated as a stochastic reservation and assignment problem, where dynamically arriving requests of customers must be assigned to suppliers in a given time horizon.
- Several solution approaches are implemented. These are based on sampling methods and greedy approaches. A newly proposed

* Corresponding author.
E-mail addresses: sara.ceschia@uniud.it (S. Ceschia), margaretha.gansterer@aau.at (M. Gansterer), simona.mancini@unipa.it, simona.mancini@aau.at (S. Mancini), antonella.meneghetti@uniud.it (A. Meneghetti).

[1] www.flexe.com.

https://doi.org/10.1016/j.cor.2024.106760
Received 7 February 2024; Received in revised form 20 June 2024; Accepted 25 June 2024
Available online 28 June 2024

method approximates the risk of creating "capacity holes", which are probably difficult to fill with future requests.
- In an extensive computational study based on realistic instances, we benchmark all approaches against *oracle* solutions, which are generated based on full knowledge about all incoming requests. Particularly for large instances, the newly proposed approach outperforms all alternative ones.
- Managerial insights regarding recommendation on acceptance and rejection strategies are derived, as, for instance, the characteristics of requests that are typically rejected in optimal solutions.

The remainder of the paper is organized as follows. In Section 2 we embed our study in the existing body of literature. Details on the decision problem are provided and formalized in Section 3. In Section 4, the newly proposed risk approximation method is presented, while Section 5 is dedicated to online stochastic algorithms. The computational study is provided in Section 6, followed by our conclusions in Section 7.

2. Literature review

In the literature, ODW is a rather new research topic. The first studies address on-demand warehousing characteristics (Pazour and Unnu, 2018; Tornese et al., 2020), costs (Unnu and Pazour, 2019), and business models (Parodos et al., 2022).

An emerging stream of research is mainly focused on the decision to adopt ODW beside more traditional storage and distribution systems. The decision can be considered as the first step before joining an ODW platform by both suppliers and customers. In Shi et al. (2021) the participants decide about the portion of storage capacity to allocate for ODW and to reserve for private usage in each time frame. The authors propose a mathematical formulation and provide theoretical properties of the problem. Unnu and Pazour (2022) develop a multi-period facility location model to identify the best solution among self-distribution, 3PL/leasing, and ODW. A mixed-integer linear program (MILP) determines location–allocation decisions of these different distribution center types with varying commitment granularities and capacity adjustments. Results show how a hybrid network adopting ODW in addition to traditional options reduces distribution costs by increasing capacity utilization. Since the above MILP model is only able to solve small-sized problems, the authors develop a heuristic to solve large-scale instances (Unnu and Pazour, 2023), in order to capture more dispersed facility locations of smaller capacity granularities with different commitment durations and capacity levels. Correia and Melo (2022) present an MILP model to solve a multi-period distribution network redesign problem, which integrates temporary warehouse lease contracts as a flexible strategy for sizing storage space to face market fluctuations. Results assess how a company can benefit from ODW as compared to the adoption of warehouse lease agreements with limited flexibility and scalability. Lee et al. (2024) simultaneously investigate demand and supply inherent uncertainties and the property of commitment in the design of an e-commerce supply chain network. To this end, a two-stage stochastic programming model is developed. In the first stage, the e-commerce retailer selects suppliers and warehouses, the latter chosen among a self-owned single-located warehouse, a set of warehouses connected to the ODW platform, and a single emergency warehouse with higher holding and transportation costs. In the second stage, commitment decisions for the facilities are determined. The authors conclude that ODW has the same effect of expanding the capacity flexibly, thus leading to cost savings for the supply chain.

Another research stream is based on the platform's perspective, thus facing the problem of matching suppliers with customers. Zhong et al. (2020) study the case of an on-demand service platform, in which two types of customers are considered: (i) the congestion-sensitive, whose request to the platform is influenced by the level of congestion of the platform itself and the related response time and perceived service quality, and (ii) those not affected. During the acceptance decision process, this sensitivity is considered by the platform. Different strategies are proposed and compared in order to maximize the platform's profit, but no optimization models are developed to support and analyze them. The study by Aouad and Saban (2023) addresses the online matching problem for a two-sided platform. As soon as a customer's request arrives, it is processed by the system, which offers the customer a set of possible options to choose from. This approach is customer-centered, but it can also provide suboptimal matching, which might lead to lower profit for the platform, as well as a negative impact on the satisfaction level of both customers and suppliers. Conversely, a more platform-oriented approach is proposed by Ceschia et al. (2023), who maximize the number of transactions gained by the platform by matching requests in a given time horizon, while at the same time preserving future participation and profits by taking customers' and suppliers' satisfaction as a secondary objective. If there is a tie, the secondary objective maximizes the number of suppliers matched with at least one customer and the number of customers that have matches within a specific threshold with respect to the minimum achievable cost. Our problem differs from this, as we are now solving the *online* variant of the problem. While in Ceschia et al. (2023) requests are all known in advance and a static optimization problem is solved, we work in a dynamic setting. Thus, as soon as a request is received, we have to decide whether to accept it or not and, in case of acceptance, which supplier to assign it to, without having information about future requests.

In online problems, sequential decisions must be made knowing only the current state of the system, without any information about the future. The literature on online decision problems is very rich. The largest part of the works deals with approximation algorithms, which are shown to be rather powerful for, e.g., knapsack problems, bin packing problems (Coffman et al., 2013; Christensen et al., 2017), and scheduling (Diedrich et al., 2009).

More recently, research on online decision problems moved to search policies and learning algorithms. Powell (2022) was the first to introduce a unified framework for sequential optimization problems. One of the most common techniques to take into account the potential impact of a decision on the future state of the system, is to sample future scenarios, solve them and extract information to guide the decision in the current state. This approach was introduced by Van Hentenryck et al. (2010) as Online Stochastic Combinatorial Optimization (OSCO). The authors test this approach on several combinatorial optimization problems showing very good performances. However, the main drawback of OSCO is that it requires considerable computational effort at each decision step, making it not suitable for large or very complex problems that have severe time limits. Another class of learning algorithms is known under the name of *dynamic lookahead policies*, in which simulation is used not only to derive a decision at the current state but to extract information that could guide future decisions for similar states, by means of the creation of a look-up table (Brinkmann et al., 2019). A parameterized lookahead policy was designed by Thul and Powell (2021) to deal with vaccines and testing-kits allocation during the COVID-19 pandemic. In Mancini et al. (2023) the authors provide a stochastic lookahead policy, in which all the computational effort is moved to a pre-processing phase, where sampled scenarios are analyzed in order to derive a deterministic policy to apply at each decision stage. This kind of approach is shown to be very effective and more scalable with respect to other simulation-based approaches.

Another popular category of algorithms for online problems is the cost function approximation (CFA) (Powell, 2022). In this approach, the objective function contains an additional term, which takes into account the expected impact of the current decision on the future states, by approximating it. An application of CFA in aircraft maintenance scheduling is reported in Deng and Santos (2022), while an application on energy storage with a rolling forecast is studied in Ghadimi and Powell (2024). Providing accurate approximations is a very challenging task since they strongly depend on the problem structure. The key point is to understand which are the potential negative consequences of each choice. An alternative viable option is to determine a set of rules to identify potentially risky decisions, that seem to be good at the moment but might turn out to be bad in the future, excluding potentially attractive opportunities in the future. An example of this approach can be found in Romero et al. (2011), where the authors provide a set of rules for the *tetris* game, to quantify the *goodness* of a decision with respect to the current reward and the impact on the system state, which itself impacts future decisions.

3. The on-demand warehousing problem

In this section, the offline ODW problem (ODWP) is firstly introduced by its mathematical formulation. Then, the corresponding online problem (OODWP)is formulated and a generic solution framework is presented, while in the last part the decision process is described in detail.

3.1. The offline problem

We consider a simplification of the ODWP introduced by Ceschia et al. (2023) which is defined in terms of time slots T, storage capacity suppliers K, and customers I. A supplier declares to an ODW platform the storage space available in each time slot Q_{kt}. Then, each customer can submit to the platform a single request, which is characterized by a demand q_i and a time frame $\tau_i \in [\tau_i^{start}, \tau_i^{end}]$. Therefore, it is introduced $\hat{q}_{it}$, which is equal to q_i for the time slots included in the time frame τ_i and 0 otherwise. In addition, there is a compatibility relationship between each customer/supplier pair specified by parameter φ_{ik}, that is equal to 1 if customer i can be assigned to supplier k, and 0 otherwise. The goal is to allocate a subset of customers I to the suppliers K, such that the capacities of the suppliers are not exceeded in any time slot and the number of accepted customer requests is maximized.

The mathematical programming formulation proposed by Ceschia et al. (2023) is an extension of the Temporal Knapsack Problem, that associates with each customer i and supplier k a binary decision variable y_{ik}, whose value is 1 if the request of customer i is allocated to supplier k and 0 otherwise, and the binary decision variable x_i with value 1 if the request of customer i is served and 0 if it is rejected by the platform. The integer programming model can be expressed as follows.

$$\max z = \sum_{i \in I} x_i \tag{1}$$

$$\sum_{i \in I} \hat{q}_{it} y_{ik} \leq Q_{kt} \qquad \forall k \in K, t \in T \tag{2}$$

$$\sum_{k \in K} y_{ik} = x_i \qquad \forall i \in I \tag{3}$$

$$y_{ik} \leq \varphi_{ik} \qquad \forall i \in I, k \in K \tag{4}$$

$$x_i, y_{ik} \in \{0, 1\} \qquad \forall i \in I \tag{5}$$

The objective function (1) maximizes the number of accepted customer requests, while constraint (2) ensures that available capacities are not exceeded. In constraint (3) the two assignment decisions are connected, while compatibility of customers and suppliers is defined in constraint (4). Variable domains are given in (5).

3.2. The online problem and a generic solution framework

Let us consider an ODW platform that operates in real time and therefore processes customer requests as soon as they arrive. Requests are not known a priori, thus, we define a sequence of requests $\boldsymbol{\xi} = [\xi_1, \dots, \xi_I]$, each one arriving at epoch i. Notice that epochs can have different lengths.

In detail, at epoch i requests $\boldsymbol{\xi_i} = [\xi_1, \dots, \xi_i]$ are known, those of customers $1, \dots, i-1$ have been already processed by the system because they appeared in the past, and the platform must decide whether and how to serve request ξ_i.

Following Van Hentenryck et al. (2009) and in order to describe the generic online optimization algorithm, we assume the existence of a dummy supplier $\perp$ with infinite stocking space available during all the planning horizon ($Q_{\perp t}, \forall t \in T$), which is compatible with all customers ($\varphi_{i\perp} = 1, \forall i \in I$). The dummy supplier is used to assign the non-selected requests and the symbol $K_\perp$ is used to denote $K \cup \perp$.

Symbol σ indicates a solution, σ_i is a partial solution at epoch i, and $\sigma[\xi_i \leftarrow k]$ denotes the assignment of supplier k to the request of customer i in a solution. Finally, for simplicity, $\sigma_\perp$ denotes the requests assigned to the dummy supplier, that are those rejected by the platform.

The generic framework for solving the online optimization problem is shown in Algorithm 1.

Algorithm 1 OnlineOptimization($\boldsymbol{\xi}$)

```
1: σ_0 := σ_⊥;                                   ▷ empty allocation
2: for i ∈ I do
3:     k ← ChooseAllocation(σ_{i−1}, ξ_i);       ▷ select the supplier
4:     σ_i ← σ_{i−1}[ξ_i ← k];                   ▷ update the solution
5: end for
6: return σ_{|I|};                               ▷ last assignment
```

The framework receives as input the list of requests $\boldsymbol{\xi}$ and initializes the solution σ_0 by assigning all requests to the dummy supplier $\perp$ (line 1). Then, at each epoch i corresponding to the arrival of the new request ξ_i, it selects the allocation k for ξ_i also considering the previous assignments σ_{i-1} (line 3), and finally updates the solution (line 4). The procedure stops when all requests have been processed and it returns a complete solution.

The function ChooseAllocation can be implemented in different ways resulting in distinct algorithms. For example, to obtain a *first fit* online algorithm (later denoted as FirstFit) the function returns the first supplier compatible with the request and with sufficient residual capacity. For *best fit* (later denoted as BestFit), the algorithm chooses the supplier that currently has the smallest total residual capacity throughout the customer time frame, given the assignment σ (Benoist et al., 2001).

More sophisticated algorithms that exploit stochastic information are presented in Section 5; they all assume to have available two components: (i) a function OptSol(σ, R) that provides the optimal allocation for a set of requests R, taking into account the past assignments included in the solution σ, and (ii) a function GetSample(t) that generates a set of requests over time frame $[t, |I|]$ sampling from the arrival distribution. In addition, the algorithms use a function FeasibleAssignment(σ_{i-1}, ξ_i, k) that checks the feasibility of the assignment of request ξ_i to supplier k, considering both the compatibility φ_{ik} and the residual capacity during the entire time frame τ_i. Finally, the function RandomBreakTie implements a random tie-break strategy, which selects an assignment in a uniform random way in case of equal values.

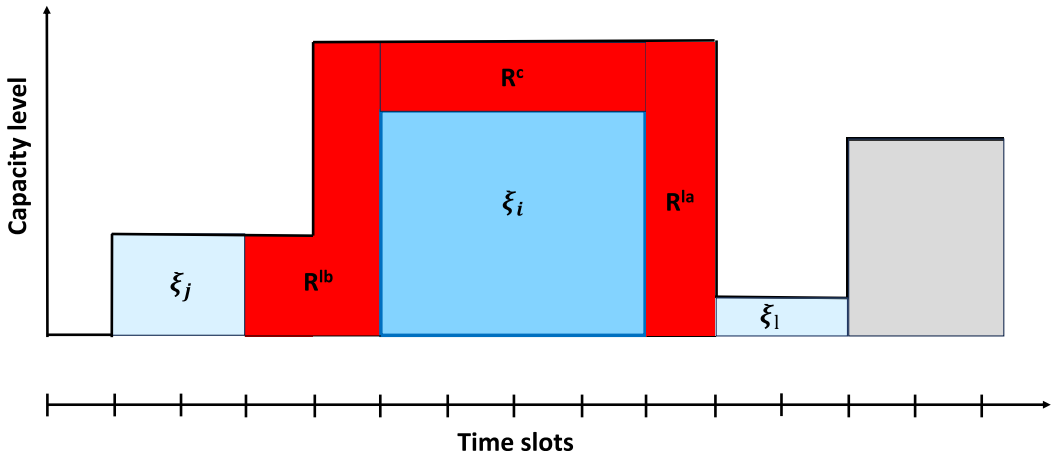


Fig. 1. An example of risky areas. If the newly arrived request (dark blue) is assigned as suggested, three risky areas are generated (red) either due to small capacity or due to closeness to currently assigned ones (light blue) such that only very few time slots are left over in between. Grey areas are free capacity not considered in the decision about the current request since (too far from ξ_i and there is another request in the middle, so there is no interaction with the current request).

3.3. Decision process and system state

Using the notation introduced by Powell (2022), state μ_i of the system at epoch i, i.e., at the moment in which request ξ_i arrives, is defined by the residual capacity for each supplier on each future day, generally referred to as Q_i^{RES} which is a matrix of size $(|H| - \tau_i^{arrival}) \times |K|$, where $\tau_i^{arrival}$ is the time slot in which ξ_i arrives, the starting time τ_i^{start} of request ξ_i, representing the day on which the request needs to be served, and the spanning time (expressed in time slots) between the arrival and the start of the request, T_i^S. The decision to make at epoch i, χ_i, consists of assigning request ξ_i to a supplier k, or rejecting it. The reward obtainable by a decision is equal to 1 if the request is accepted, whereas it is equal to 0 in case of rejection. The reward obtained does not depend on the supplier matched with the request. The transition phase among state i and $i+1$, consists of updating the residual capacity of the supplier k, to which request ξ_i has been assigned, according to the capacity absorbed by ξ_i, or to not modify the residual capacity of any supplier if ξ_i is rejected, obtaining a matrix Q_{i+1}^{RES}. The stochastic information ω_{i+1} comprises the newly arrived request ξ_{i+1}. The new state in $i+1$ is then $S_{i+1} = (Q_{i+1}^{RES}, \xi_{i+1})$.

4. The RISKY heuristic

The main idea behind this algorithm is to identify potentially *risky* areas that can be generated by assigning a customer request i to a supplier k. The larger the risky area, the worse the assignment. These areas are portions of the supplier capacity that, due to the current assignment, present a high risk of remaining unused. This can happen for two reasons: (1) the residual capacity is so small that with a high probability, there will be no further requests fitting it; (2) the time interval between the current request and the previous (or the following) one is so limited, that it would be very hard to find a request which matches that interval. The risky area of the first type is denoted as R^h, while the second type of risky areas, referred to as the requests processed before and after the current request, are denoted as R^{lb} and R^{la}, respectively. We consider risky the areas of type h if the residual capacity is included in the interval $[\alpha_{min}; \alpha_{max}]$. Similarly, we consider risky the areas of type l if their time frame is included in the interval $[\beta_{min}; \beta_{max}]$. An illustrative example is reported in Fig. 1.

We consider risky all areas with a residual capacity lower than the average demand of a request α_{max} or with a time span lower than the average requests length β_{max}. However, areas of large size can also be identified by a very small residual capacity for a very large number of time slots, which can be represented by a rectangle with a very large length and a very small height. This could be identified as a potentially risky area, but actually it is not the case, since we would not be able to fill such a small capacity, independently of the number of time slots for which it is available. To overcome this issue we set a minimum value of both residual capacity and time span (α_{min} and β_{min}) under which the respective areas are not considered risky.

Each time a new request arrives, we compute, for each feasible assignment to a supplier k, the global risky area $R_k^A = R_k^h + R_k^{lb} + R_k^{la}$. Then, a penalty p_k associated with the assignment is defined as $p_k = R_k^A / A^{EXP}$, where A^{EXP} is the expected area covered by a request, computed as the expected demand multiplied by the expected duration of a request. However, we use this additional penalty term only for instances in which we expect a relatively high number of requests. In fact, if the number of requests is small, then we should not penalize request acceptance since the probability of filling the related capacity with several future requests is small and we run the risk of leaving unused capacity due to our decisions to reject certain opportunities.

The assignment score s_k is then computed as

$$s_k = 1 - p_k(T_i^S / |T|), \tag{6}$$

where T_i^s is defined as the spanning time between the time on which the request starts and the time in which it enters the system, $\tau_i^{start} - \tau_i^{arrival}$.

This allows us to consider that, if a request starts immediately after the time slot in which it is received, no risky areas are generated as the capacity could not be used anyway. Conversely, the larger the time interval between the request arrival and the starting time, the higher the probability that future requests, that do not fit the residual capacity, have to be rejected. The score for rejecting a request (i.e., assigning it to the dummy supplier), is fixed equal to 0. Requests are always assigned to suppliers yielding the highest score. A request is rejected only if there are no compatible suppliers, or if the risky areas generated by its acceptance are so large that the corresponding penalty is higher than the benefit achievable by accepting it.

For instances in which the number of expected requests is much higher than the number of suppliers ($|I| \geq \rho|K|$), where ρ is a parameter of the algorithm, we also penalize requests larger than the expected demand q^{EXP}, updating p_k by $p_k = p_k + (q_i - q^{EXP})/q^{EXP}$. This

allows us to penalize the acceptance of large items, which could occupy areas that could potentially fit more than one request. Since the goal of the problem is to maximize the number of accepted requests, this additional penalty correction pushes the algorithm to prefer accepting several small requests instead of one large request. This is of practical relevance, as the price per pallet per day is constant for each supplier, but there are additional duties paid by the customer for the service. Such duties are fixed costs, which do not depend on the size of the request. Thus, accepting several small requests filling exactly the same capacity as a single large request, yields higher profits for the platform.

The pseudocode of the algorithm is reported in Algorithm 2.

Algorithm 2 RISKY (σ_{i-1}, ξ_i)

1: best_allocation $\leftarrow 0$;
2: $s_\perp \leftarrow 0$
3: **for** $k \in K$ **do**
4: $R_k^h, R_k^{lb}, R_k^{la} \leftarrow$ COMPUTERISKYAREAS(σ_{i-1}, ξ_i)
5: $R_k^A \leftarrow R_k^h + R_k^{lb} + R_k^{la}$
6: $p_k \leftarrow R_k^A / A^{EXP}$ ▷ compute the penalty
7: **if** $q_i \geq q^{EXP} \wedge |I| \geq \rho |K|$ **then**
8: $p_k \leftarrow p_k + (q_i - q^{EXP})/q^{EXP}$ ▷ penalty correction for large requests
9: **end if**
10: $s_k \leftarrow 1 - p_k(T^s/|T|)$ ▷ compute the score
11: **end for**
12: $s_\perp = 0$ ▷ score of the dummy supplier
13: best_allocation = $\text{argmax}_{(k \in K_\perp)}\, s_k$
14: **return** best_allocation;

Each epoch i corresponds to the arrival of a new request, the time elapsed between two epochs is not constant but depends on the requests' arrival time. In other words, we do not monitor the system at fixed points in time, but each time a new event occurs (i.e., a new request arrives). The state of the system μ_i at epoch i is uniquely identified by the residual capacity profile for each supplier in each time slot successive to the arrival time slot of the request ξ_i, defined as Q_{i-1}^{RES}. At each epoch RISKY approximates the cost of assigning request ξ_i to supplier k given the current state of the system μ_i. Such cost function approximation is based on computing the reward associated with the acceptance of ξ_i and the expected loss of reward based on the probability that one or more future requests could not be assigned to the supplier, due to lack of capacity generated by the assignment of ξ_i to k. The cost function value for the rejection choice (assignment to the dummy supplier) is 0, since no reward is achieved by rejecting a request, but no future opportunities of collecting reward are missed due to this choice, since no capacity is consumed. If the cost function value is negative for all the suppliers, then the best choice is to reject the request.

5. Online stochastic algorithms

The framework reported in Algorithm 1 can be executed using different CHOOSEALLOCATION functions. In particular, we present two classical methods, i.e., an *expectation* and *consensus* algorithm, as well as a new *regret* algorithm specifically proposed for the OODWP.

5.1. The EXPECTATION algorithm

The idea behind the EXPECTATION algorithm (Algorithm 3) is to evaluate, for each request, the assignment to all possible suppliers against different scenarios of future requests obtained by sampling, and then to choose the supplier for which the expected value is maximized (Van Hentenryck et al., 2009).

Algorithm 3 CHOOSEALLOCATION-E(σ_{i-1}, ξ_i)

1: best_allocation $\leftarrow 0$; best_reward $\leftarrow 0$; ▷ initialization
2: **for** $k \in K_\perp$ **do**
3: $f(k) \leftarrow 0$;
4: **end for**
5: **for** $s \leftarrow 1, \ldots, \mathcal{O}/|K_\perp|$ **do**
6: $R_{i+1} \leftarrow$ GETSAMPLE$(i+1)$; ▷ generate a scenario of future requests
7: **for** $k \in K_\perp$ **do**
8: **if** FEASIBLEASSIGNMENT(σ_{i-1}, ξ_i, k) **then**
9: $\sigma^* \leftarrow$ OPTSOL$(\sigma_{i-1}[\xi_i \leftarrow k], R_{i+1})$;
10: $f(k) \leftarrow f(k) + 1$; ▷ update the evaluation
11: **if** $f(k) >$ best_reward **then**
12: best_allocation $\leftarrow k$, best_reward $\leftarrow f(k)$;
13: **else**
14: **if** $f(k) =$ best_reward $\wedge$ RANDOMBREAKTIE() **then**
15: best_allocation $\leftarrow k$, best_reward $\leftarrow f(k)$;
16: **end if**
17: **end if**
18: **end if**
19: **end for**
20: **end for**
21: **return** best_allocation;

The algorithm is initialized in lines 1–4, it generates $\mathcal{O}/|K_\perp|$ scenarios of future requests, where $\mathcal{O}$ is a control parameter that counts the number of calls of the optimization procedure. The parameter $\mathcal{O}$ must be greater than the number of suppliers, otherwise, the number of scenarios is less than 1. Subsequently, the algorithm considers each supplier k, including the dummy (line 7), and it checks if the accommodation of request ξ_i to k is feasible considering the previous allocations stored in σ_{i-1} and the input data (line 8). The algorithm implicitly allocates ξ_i to k, and calls the optimization algorithm (line 9), which is the implementation in CPLEX of the model of Section 3.1. The evaluation of supplier k is updated in line 10 by adding the value of the objective function of the optimal solution σ^*. The best allocation (and best reward) are eventually updated in lines 11–17, adopting a random breaking time strategy in case of parity. For each request, all suppliers are evaluated against all scenarios and the supplier with the highest evaluation is returned (line 21). If the selected supplier is the dummy one, the request is rejected.

Note that the EXPECTATION algorithm runs $\mathcal{O}$ times the optimization procedure. However, each request is evaluated with respect to $\mathcal{O}/|K_\perp|$ scenarios. The main drawback of this algorithm is that when $\mathcal{O}$ is small due to time constraints (e.g., in online problems), each request is only tested against a limited number of scenarios, thus the algorithm does not collect much information. As a consequence, the EXPECTATION algorithm is appropriate when computational times are not critical, and $\mathcal{O}$ can be large enough to guarantee high-quality results.

5.2. The CONSENSUS algorithm

The CONSENSUS algorithm calls the optimization procedure without the pre-assignment of request ξ_i to any supplier; on the contrary, the supplier k^* to which request ξ_i is assigned in the optimal solution is

Table 1
Features of dataset 1.

Group	Weeks	Suppliers	Customers		Daily occupancy	
			min	max	min	max
A	4	10	19	36	0.13	0.48
B	8	10	13	31	0.12	0.31
C	12	10	20	37	0.13	0.28
D	4	10	36	61	0.42	0.72
E	8	10	37	58	0.36	0.52
F	12	10	47	61	0.29	0.71
G	4	10	84	116	0.80	1.60
H	8	10	87	119	0.76	1.27
I	12	10	88	118	0.69	1.02

rewarded, while all other suppliers receive no credit (Van Hentenryck et al., 2009).

Algorithm 4 ChooseAllocation-C(σ_{i-1}, ξ_i)

1: best_allocation ← 0; best_reward ← 0; ▷ initialization
2: **for** $k \in K_\perp$ **do**
3: $f(k) \leftarrow 0$;
4: **end for**
5: **for** $s \leftarrow 1, \dots, \mathcal{O}$ **do**
6: $R_i \leftarrow \xi_i \cup$ GetSample($i+1$);
7: $\sigma^* \leftarrow$ OptSol(σ_{i-1}, R_i) ▷ requests include ξ_i
8: $k^* \leftarrow \sigma^*(\xi_i)$;
9: $f(k^*) \leftarrow f(k^*) + 1$; ▷ only the evaluation of the best supplier is incremented
10: **if** $f(k^*) >$ best_reward **then**
11: best_allocation ← k^*, best_reward ← $f(k^*)$;
12: **else**
13: **if** $f(k^*) =$ best_reward $\wedge$ RandomBreakTie() **then**
14: best_allocation ← k^*, best_reward ← $f(k^*)$;
15: **end if**
16: **end if**
17: **end for**
18: **return** best_allocation;

Note that in line 6 the set of future requests R_i includes also the current request ξ_i, whose optimal allocation is denoted by k^*. Only the evaluation of supplier k^* is incremented by 1 (line 9). Finally, the algorithm returns the supplier with the best reward, i.e., the one that has been chosen in the highest number of scenarios. The main advantage of the Consensus algorithm is that each request is examined against $\mathcal{O}$ scenarios, instead of $\mathcal{O}/|K_\perp|$. This way the available samples are not partitioned between the requests, which is a substantial advantage when $\mathcal{O}$ is small and/or the number of suppliers is large. On the contrary, its limitation is that only the information about the best supplier is used, while other "intermediate" suppliers are ignored.

5.3. The Regret algorithm

The Regret algorithm as originally proposed by Van Hentenryck et al. (2009) is similar to the Consensus algorithm. However, instead of giving a reward only to the best supplier, an approximation is used to compute suboptimal solutions, where the current request is allocated to each feasible supplier. As for the Consensus algorithm, each supplier receives an evaluation in each scenario, performing $\mathcal{O}$ optimizations for each request.

Algorithm 5 shows the pseudo-code of our implementation of the Regret algorithm specifically designed for the OODWP.

Algorithm 5 ChooseAllocation-R(σ_{i-1}, ξ_i, ν)

1: best_allocation ← 0; best_reward ← 0; feasible_assignment ← **false**;
2: **for** $k \in K_\perp$ **do** ▷ initialization
3: $f(k) \leftarrow 0$;
4: **end for**
5: **for** $s \leftarrow 1, \dots, \mathcal{O}$ **do**
6: $R_i \leftarrow$ GetSample($i+1$); ▷ requests without ξ_i
7: $\sigma^* \leftarrow$ OptSol(σ_{i-1}, R_i)
8: **for** $k \in K$ **do**
9: reward ← FeasibilityWithFutureRequests(σ^*, ξ_i, k)
10: $f(k) \leftarrow f(k)$+ reward;
11: **if** $f(k) >$ best_reward **then**
12: best_allocation ← k, best_reward ← $f(k)$;
13: feasible_assignment ← **true**;
14: **else**
15: **if** $f(k) =$ best_reward $\wedge$ RandomBreakTie() **then**
16: best_allocation ← k, best_reward ← $f(k)$
17: **end if**
18: **end if**
19: **end for**
20: **if** feasible_assignment = **false then**
21: $f(\perp) \leftarrow f(\perp) + 1$;
22: **if** $f(\perp) >$ best_reward **then**
23: best_allocation ← $\perp$;
24: **end if**
25: **end if**
26: **end for**
27: **if** best_allocation $\neq \perp \wedge$ best_reward $< \mathcal{O}/\nu$ **then**
28: best_allocation ← $\perp$, best_reward ← $f(\perp)$;
29: **end if**
30: **return** best_allocation;

The key idea behind this algorithm is to generate $\mathcal{O}$ scenarios for each request and to solve the optimization problem without the current request (line 7). Once the optimal solution σ^* is obtained, we evaluate the allocation of request ξ_i to any supplier k. The function FeasibilityWithFutureRequests(σ^*, ξ_i, k) in line 9 tries to assign request ξ_i to supplier k given the past assignments of requests $i \in \{1, \dots, i-1\}$ and those related to the future requests R_i generated by the sampling and stored in σ^*. If the allocation is feasible, the reward of k is 1, otherwise its reward is the fraction of satisfied demand during the time frame ($0 \leq$ reward ≤ 1). In addition, the algorithm has a control parameter ν (line 27) that is used to avoid early assignments to suppliers and preserve space for future requests. The algorithm returns the allocation with the best reward.

6. Computational experiments

In this section we validate through a set of computational experiments, the effectiveness of the proposed algorithms. In Section 6.1, we describe test instances and parameter settings, while Section 6.3 compares the performance of the different algorithms and benchmarks the procedures with the *oracle* solution. In Section 6.5 we provide managerial insights.

All algorithms were coded in C++ and resort to IBM ILOG CPLEX APIs (version 22.11) for ILP routines, with the default optimization parameters. Experiments were run on an Intel 8-Cores i7-7700 CPU (3.60 GHz) with Ubuntu Linux 22.04.

Table 2
Features of dataset 2.

Group	Weeks	Suppliers	Customers		Daily occupancy	
			min	max	min	max
J	4	10	233	275	2.73	4.39
K	8	10	226	272	1.75	2.79
L	12	10	219	270	1.73	2.56
M	4	10	457	538	4.35	7.41
N	8	10	469	548	3.86	5.84
O	12	10	449	540	3.40	4.85

6.1. Test design

In order to study the performance of the proposed solution approaches, we construct a test bed varying some input parameters systematically. In detail, we consider a planning period $|T|$ of 28, 56, and 84 time slots (corresponding to 4, 8 and 12 weeks), an expected number of customers equal to $\{25, 50, 100, 250, 500\}$, and a fixed number of capacity suppliers $|K| = 10$. We thus compose a full factorial design of experiments, leading to 15 different classes of instances (A–O), grouped in two datasets depending on the number of customers: dataset 1 (classes A–I) with up to 100 customers, and dataset 2 (classes J–O) with a number of customers between 250 and 500. For each class, we generated 10 instances resulting in 150 test cases in total.

Regarding the generation procedure, the arrivals of customer requests follow a Poisson process with an arrival rate λ computed as the ratio between the expected number of customers and the number of time slots. As a consequence, the actual number of customers $|I|$ can be slightly different from the expected one, given as an input to the instance generator. The arrival processes for requests are independent.

The demand q_i of each customer i is randomly chosen within 10 and 2500 pallets. The first day τ_i^{start} of the requested time frame is randomly selected between the arrival of the request and the last time slot of the horizon. Then, the length of the time frame $\tau_i^{end} - \tau_i^{start}$ is drawn within the minimum booking period (set to 7-time slots) and the remaining time slots. If the remaining time slots are less than the minimum booking period, the length of the time frame is set equal to the remaining time slots.

The supplier availability is generated in two steps for 7 consecutive time slots (one week): we first decide if there is some space available in that week with probability 0.5, then, in case, the values of stocking space available Q_{kt} are drawn between 2500 and 5000 pallets. The compatibility matrix is drawn with a density of 0.9. We only consider suppliers within a certain radius from the customer location to ensure that the matching provided by the platform meets the customer's needs.

The selection of the values of parameters used by the instance generator is inspired by the real-life scenario and the generation procedure described in Ceschia et al. (2023). All random values (except the arrivals) are drawn from a discrete uniform distribution.

Tables 1 and 2 show the main features of the two datasets grouped by classes. The daily occupancy is the average ratio between the daily demand and the daily supply. For the number of customers and the daily occupancy, the minimum and the maximum values in the class are reported.

Instances and the instance generator coded in Python are available at https://bitbucket.org/sceschia/online-on-demand-warehousing-problem.

6.2. Parameters

Based on preliminary experiments, for the Expectation, Consensus, and Regret algorithms, parameters $\mathcal{O}$ and ν were fixed to 50 and 2, respectively. The mathematical programs are solved by CPLEX with a time limit of 30 s. The mathematical solution can be found within the time limit for all instances of dataset 1, except groups G, H and I. For Oracle, we set a time limit of one hour, such that CPLEX is able to find the optimal solution for all the instances of dataset 1. For larger instances of dataset 2, Oracle consistently reaches the time limit.

For the RISKY algorithm, α_{min} and β_{min} are set equal to $0.1A^{EXP}$ whereas α_{max} and β_{max} equal to A^{EXP}

6.3. Comparison of the algorithms

Table 3 depicts the aggregated results on instances of groups A–I of dataset 1; run times are in seconds. Each instance is solved five times for Expectation, Consensus, and Regret because of the non-deterministic nature of the sampling. For FirstFit (FF), BestFit (BF) and Risky running times are negligible, always inferior to one second.

We observe that the Consensus algorithm outperforms all the other methods on all groups but the smallest one (A), where the best score is obtained by Regret. For all the other test cases, the Regret algorithm does not bring any benefit over Consensus and Expectation that are consistently superior. This proves the value of stochastic information to guide decisions about the future. This behavior is more evident in Fig. 2, which plots the average optimality gap with respect to the optimal offline solution (Oracle) for all groups of dataset 1. It can be noticed that on small and medium instances (groups A-F), FirstFit and BestFit perform well, while their performances deteriorate as the number of customers increases. On the other hand, Risky exhibits good quality results through all instances of dataset 1: it is within 12% of the Oracle on average and it is never worse than 18% of it.

When comparing the run times of the different methods, it is evident that Expectation, Consensus, and Regret algorithms require long computational times; as a consequence, these methods become unusable for instances with more than 80 customers. In contrast, FirstFit, BestFit and Risky are very fast, requiring less than one second for their execution, and scale very well, since their computational times are only marginally affected by the instance's size. The disparity in running times is primarily due to the necessity of online stochastic optimization algorithms to evaluate each new request against a minimum number of scenarios to guarantee good-quality results.

We thus compare the performance on dataset 2 (which includes instances of 250 and 500 customers) only using FirstFit, BestFit, and Risky. The results are presented in Table 4.

For these larger datasets, we can see that the performance of FirstFit and BestFit are comparable, although surprisingly the former is slightly superior. Notably, the newly proposed RISKY strongly dominates both of them for all the groups of large instances (J-O), demonstrating its efficiency and scalability compared to the other heuristic methods.

6.4. Performance guarantee

The RISKY heuristic does not guarantee an approximation ratio, since for instances with particular features the optimal solution tends to infinity with increasing number of suppliers $|K|$, while RISKY always provides a solution with an objective function value equal to 0 (all the requests are rejected). We show this behavior by means of an illustrative example. Consider having $|K|$ suppliers with identical capacities all compatible with all requests. Consider further to have $|I|$ identical requests, with $|I| > |K|$ consuming the largest part of the capacity of the suppliers, as shown in Fig. 3. For each request i, RISKY identifies a potentially risky area (shown in red), which yields a negative score s_k for all the suppliers, therefore, it is always assigned to the dummy supplier $\perp$, i.e., it is rejected. This would result in a solution in which all requests are rejected and therefore the cumulative reward achieved by the company would be 0. The optimal solution of this instance consists of accepting $|K|$ requests and assigning each one of them to a different supplier, which leads to a solution with an objective function value of $|K|$. When $|K|$ grows to infinity, the optimal solution value tends to infinity as well, while the value provided by RISKY is always equal to 0. This proves that RISKY does not admit any approximation

Table 3
Comparative results (objective function and run times in seconds) on dataset 1 with all proposed algorithms (FF, BF, and RISKY times are always <1 s).

Group	Oracle	FF	BF	Risky	Expectation		Consensus		Regret	
	z	z	z	z	z	Time (s)	z	Time (s)	z	Time (s)
A	24.2	22.5	22.7	22.4	22.2	9.4	22.8	19.7	**22.8**	18.3
B	22.4	21.8	21.6	21.2	21.2	4.8	**21.9**	11.6	21.7	9.4
C	23.3	22.1	21.8	21.4	21.8	6.0	**22.5**	12.6	22.3	12.2
D	39.1	33.4	33.3	33.6	35.1	190.2	**36.2**	480.8	33.8	633.3
E	40.7	36.5	36.2	35.3	36.9	63.6	**37.8**	152.7	36.4	142.6
F	41.9	37.5	37.0	35.1	37.84	48.4	**39.0**	124.1	37.3	106.6
G	57.4	44.6	42.3	49.2	49.75	4905.5	**53.8**	13 452.7	47.6	10 530.5
H	67.7	54.4	54.5	56.9	58.8	8071.5	**62.0**	25 462.5	58.0	21 859.3
I	71.1	58.9	58.6	59.4	62.1	7052.9	**64.9**	20 236.3	61.4	14 261.2
Avg	43.1	36.9	36.4	37.2	38.4	2261.4	**40.1**	6 661.4	37.9	5 285.9

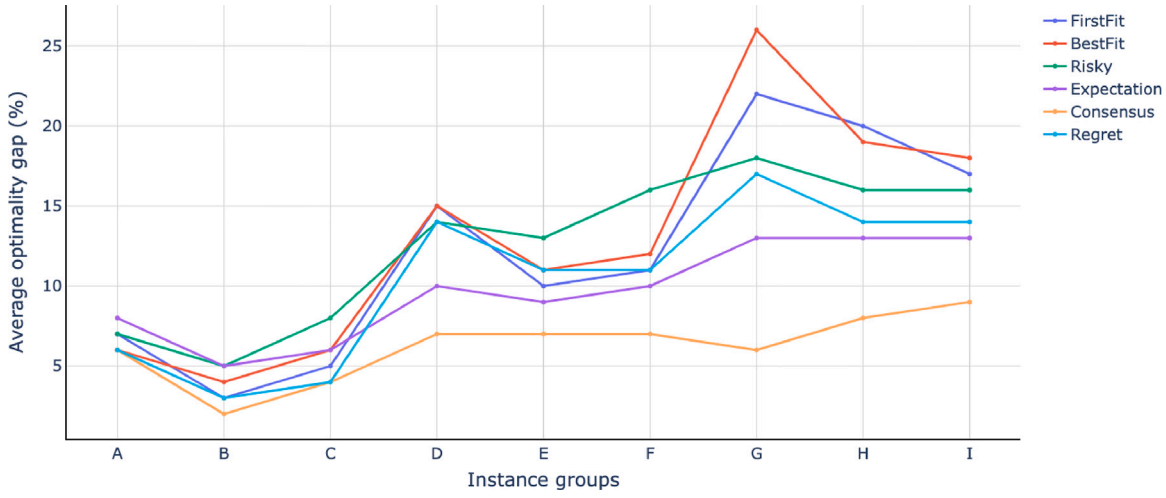


Fig. 2. Optimality gap for dataset 1.

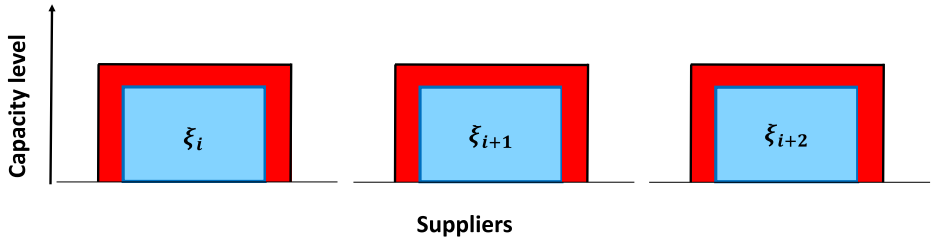


Fig. 3. Optimal solution for an illustrative example.

Table 4
Comparative results on dataset 2.

Group	Oracle	FirstFit	BestFit	Risky
	z	z	z	z
J	97.5	60.9	57.8	**74.4**
K	110.1	76.0	73.5	**86.1**
L	124.1	90.0	87.2	**100.4**
M	139.0	78.0	74.0	**87.0**
N	167.8	101.5	98.6	**120.4**
O	180.5	114.3	114.6	**134.8**
Avg	136.5	86.8	84.3	**100.5**

ratio. Nevertheless, our extensive computational study shows that it systematically obtains very good results. In fact, the instances for which it obtains poor results (as the one in this example) have very specific features that are very unlikely to occur in realistic instances.

6.5. Managerial insights

In this section, we present two analyses devoted to derive interesting managerial insights. The first analysis aims at assessing the benefit achievable by exploiting two relevant features of Risky, namely, (i) the additional penalty associated with very large requests, which takes into consideration that the space occupied by a large request could be more profitably filled with several small requests, and (ii) the penalty correction factor depending on the spanning time between the arrival and the start of the request, T_i^S, which reduces the effect of the penalty when T_i^S tends to be zero, considering the fact that the residual alternative opportunity to fill the space occupied by that requests is decreasing when approaching the starting time (see Section 4).

We compare the results of Risky including these two features with two modified versions of the algorithm, in which we do not consider additional penalties for large requests (NO_LARGE_PEN) and no correction factor based on the spanning time (NO_TIME_PEN). In the latter case, the penalty for generating risky areas is included independently

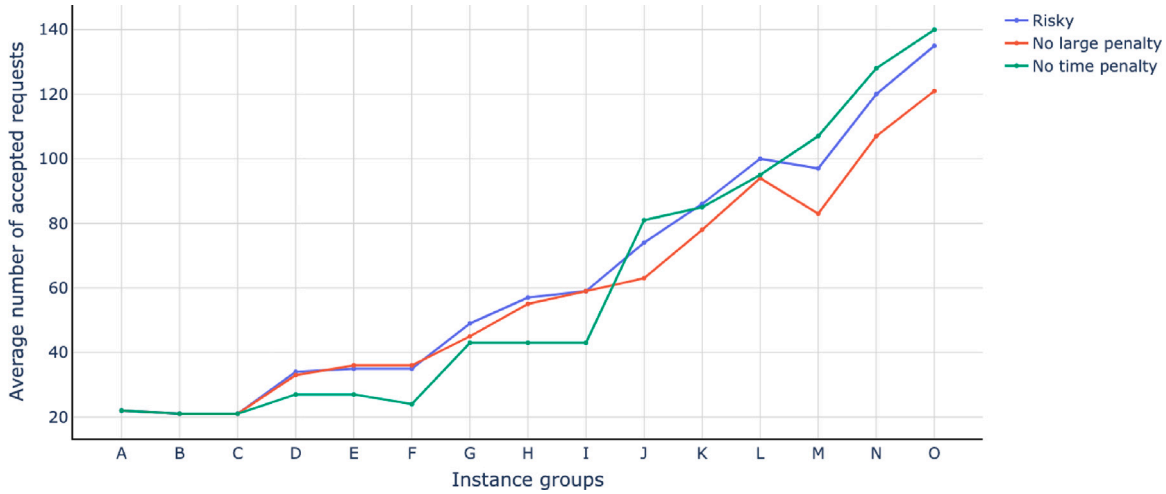


Fig. 4. Comparison of Risky in three different version grouped by class.

Table 5
Analysis of the Risky algorithm in 3 different versions based on the objective function.

Group	Risky	NO_LARGE_PEN	NO_TIME_PEN
A	21.9	21.9	21.9
B	21.2	21.2	21.2
C	21.4	21.4	21.4
D	33.5	33.2	27.1
E	35.3	35.9	27.4
F	35.1	36.1	23.8
G	48.9	45.0	43.1
H	56.9	54.8	43.4
I	58.8	58.9	43.2
Avg dataset 1	37.0	36.5	30.3
J	74.4	63.4	81.2
K	86.1	78.1	85.4
L	100.4	93.7	95.4
M	96.5	83.3	107.1
N	120.4	106.6	127.6
O	134.8	121.0	139.9
Avg dataset 2	102.1	91.0	106.1
Avg all	63.0	58.3	60.6

of the spanning time. Comparisons for all classes of instances A–O are provided in Table 5.

From these results we can evince that both features are needed as Risky outperforms, on average, all the other algorithms. However, from a more detailed analysis, we can see that while on small and medium-sized instances (dataset 1) Risky is strongly preferable, on larger ones (dataset 2), NO_TIME_PEN performs slightly better. Conversely, NO_LARGE_PEN is almost always slightly dominated by Risky with the proposed settings. This trend is highlighted in Fig. 4 which plots the number of accepted requests by group. If we collect instances by time horizon length rather than by request group, as depicted in Fig. 5, we observe that Risky strongly outperforms both algorithms. This means that the effectiveness of both features (i) and (ii) vary only according to the number of requests, while the time horizon length does not significantly affect it. Since the number of suppliers and the total available capacity do not depend on the number of requests, we can argue that what actually influences the utility of these features (in particular the time-related penalty) is the ratio between the number of requests and the capacity. The larger this ratio, the lower the benefit of the penalty, which even becomes negative for very large values of this ratio. When the ratio is large, the number of future alternative opportunities to fill the capacity is large too and therefore, we can be more selective when accepting requests and we can reject requests that generate large risky areas. On the contrary, when the ratio is small, and therefore the future opportunities are limited, we are more likely to accept requests, even those which generate risky areas, since we risk ending up with a large percentage of unused capacity.

Table 6
Analysis of rejected requests for all algorithms and for Oracle.

	Demand	Time frame	Spanning	Compatibility
Consensus	56.42%	23.72%	−14.67%	−1.46%
Expectation	46.68%	15.43%	−27.44%	−0.93%
Regret	52.48%	11.07%	−3.01%	−0.90%
FirstFit	31.40%	8.01%	−34.08%	−0.86%
BestFit	32.33%	9.24%	−33.79%	−0.68%
Risky	37.43%	8.88%	−18.61%	−0.71%
Oracle	50.20%	24.05%	−3.18%	−0.43%

Concerning the time-related penalty correction factor, it should be noted how it considerably decreases the penalty value if the spanning time is small, see Eq. (6). Thus, we push the system to accept requests within a very short spanning time, regardless of the generated risky areas. This is a rational behavior, since, when the spanning time is small, the number of future opportunities is limited, especially when the ratio between the number of requests and the available capacity is low. Conversely, when the ratio is high, the number of future opportunities is still quite high even when the spanning time is small, and therefore, we can afford to reject a risky request. We believe this is an interesting managerial insight.

The second analysis concerns the study of the features of rejected requests, in order to provide the decision maker with an identikit of the requests that should be rejected. For this, we consider four features of each request: (i) the demand, (ii) the time frame length, (iii) the spanning time, and (iv) the compatibility degree between the request and the suppliers. The latter is defined as follows for each request $i \in I$.

$$\frac{\sum_{k \in K} \varphi_{ik}}{|K|} \tag{7}$$

In Table 6 we report, for each feature, the average percentage variation with respect to the average value of the features in all the requests. Results are provided for all algorithms and the Oracle solution.

We observe that the average demand of rejected requests is clearly above the overall average demand for all the algorithms, even if the greedy heuristics (FirstFit, BestFit, and RISKY) tend to accept smaller (but still above average sized). Regarding the time frame, we

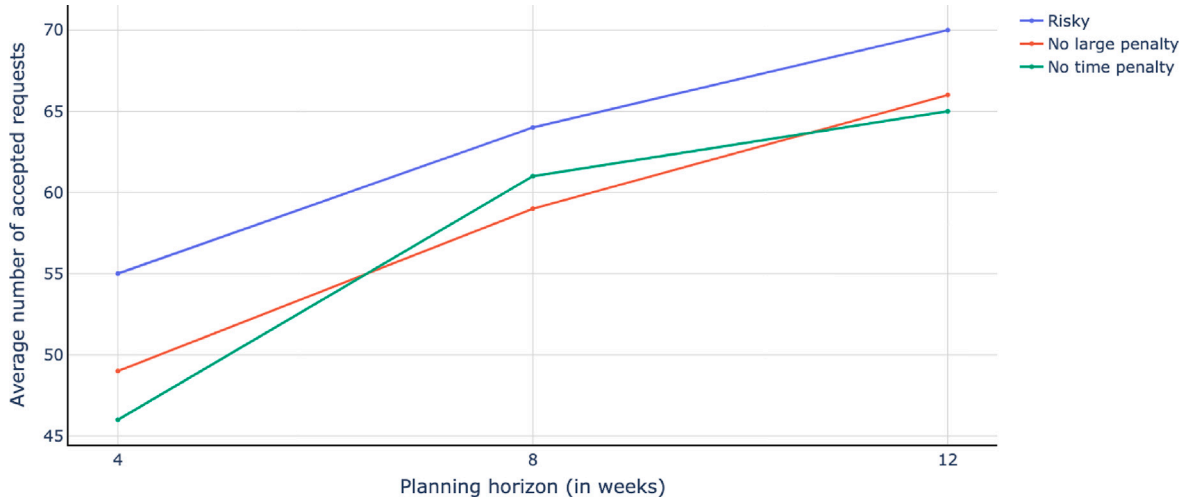


Fig. 5. Comparison of Risky in three different versions grouped by the length of the time horizon.

can observe that on average requests rejected by Oracle are 24.05% longer than the overall average time frame. The only algorithm, which matches this behavior is Consensus with 23.72%, while all the other algorithms reject shorter requests, especially the three greedy ones (only 9% above average).

On average, requests accepted by Oracle have a spanning time just slightly lower than the average time (−3.18%). The only algorithm matching this behavior is Regret, while the others reject requests with much shorter spanning times on average. This result is totally counter-intuitive, as it seems more reasonable to reject requests with large spanning times in order to profit from future opportunities to better fill the supplier capacity. When investigating more deeply this aspect, we notice that most of the rejections, especially in the later parts of the time horizon, are not preemptive, but occur because, due to wrong choices made in the previous epochs, the incoming requests are not compatible with the residual capacity of any supplier and must be rejected.

Finally, rejected requests show an average compatibility degree approximately equal to the overall average degree for all the algorithms. Therefore, this feature seems not to be relevant for the recommendation which requests to reject.

In summary, we find that a request with large demand, long time frame, not very long spanning time, and average compatibility degree, is very likely to be rejected in the *oracle* solution. This finding can be seen as a valuable rule of thumb for acceptance or rejection decisions.

7. Conclusions

We have addressed the Online On-demand Warehousing Problem (OODWP), which is grounded in the paradigm of the *Sharing Economy*. It arises if providers of shared storage space platforms have to match incoming customer requests with the available capacities of suppliers. The objective is to accept and assign as many as possible customer requests, where these requests have two dimensions that have to be considered, i.e., duration and capacity consumption. Also, the spanning time (which is the time slack until a newly arrived request has to be serviced) and compatibilities between requests and suppliers are taken into account.

We formalized the decision process as a stochastic reservation and assignment problem and presented several solution algorithms. These include greedy as well as sampling-based approaches. A newly proposed idea is to reject or accept incoming requests based on the estimation of the risk their features (time and required capacity) might impose requests arriving later.

In an extensive computational study, we compared all proposed algorithms and benchmarked them against an *oracle* solution, where full information availability is assumed. Results revealed, that on small test instances, one of the sampling-based methods outperforms all other algorithms. However, due to heavy run times, this approach is not suitable for larger instances. In these cases, the newly designed risk approximation-based approach is dominant, outperforming the alternative algorithms in all tested groups of instances. Finally, we also derived valuable managerial insights regarding acceptance and rejection strategies.

Future work could analyze the problem from a bilevel optimization perspective or apply sophisticated learning techniques. Furthermore, it could be interesting to address a setting in which we do not have to give an immediate reply to the customers but to wait for a specified amount of time before making a decision for a larger set of requests. While this might be viable approach in regards of valuable matching, one has to take into account the loss of customers, who are not willing to accept the postponement of decisions. Observing current market behavior, covering such a setting has to include in-depth investigations of platform providers' business models. Another interesting future development concerns the exploitation of data about the average number of expected compatible requests for each supplier, which can strongly differ among suppliers. For instance, suppliers located on a very densely populated area could be filled more easily compared to those located in rural zones.

CRediT authorship contribution statement

Sara Ceschia: Writing – original draft, Supervision, Software, Methodology, Data curation, Conceptualization. **Margaretha Gansterer:** Writing – original draft, Supervision, Software, Methodology, Data curation, Conceptualization. **Simona Mancini:** Writing – original draft, Supervision, Software, Methodology, Data curation, Conceptualization. **Antonella Meneghetti:** Writing – original draft, Supervision, Software, Methodology, Data curation, Conceptualization.

Data availability

Instances are available at https://bitbucket.org/sceschia/online-on-demand-warehousing-problem.

References

Aouad, A., Saban, D., 2023. Online assortment optimization for two-sided matching platforms. Manage. Sci. 69, 2069–2087.

Benoist, T., Bourreau, E., Caseau, Y., Rottembourg, B., 2001. Towards stochastic constraint programming: A study of onine multi-choice knapsack with deadlines. In: International Conference on Principles and Practice of Constraint Programming. Springer, pp. 61–76.

Brinkmann, J., Ulmer, M., Mattfeld, D., 2019. Dynamic lookahead policies for stochastic-dynamic inventory routing in bike sharing systems. Comput. Oper. Res. 106, 260–279.

Ceschia, S., Gansterer, M., Mancini, S., Meneghetti, A., 2023. The on-demand warehousing problem. Int. J. Prod. Res. 61, 3152–3170.

Christensen, H., Khan, A., Pokutta, S., Tetali, P., 2017. Approximation and online algorithms for multidimensional bin packing: A survey. Comp. Sci. Rev. 24, 63–79.

Coffman, E., Csirik, J., Galambos, G., Martello, S., Vigo, D., 2013. In: Du, D.-Z., Pardalos, P.M., Graham, R.L. (Eds.), Handbook of Combinatorial Optimization. pp. 455–531, chapter Bin Packing Approximation Algorithms: Survey and Classification.

Correia, I., Melo, T., 2022. Distribution network redesign under flexible conditions for short-term location planning. Comput. Ind. Eng. 174, 108747.

Deng, Q., Santos, B., 2022. Lookahead approximate dynamic programming for stochastic aircraft maintenance check scheduling optimization. European J. Oper. Res. 299, 814–833.

Diedrich, F., Jansen, K., Schwarz, U., Trystram, D., 2009. A survey on approximation algorithms for scheduling with machine unavailability. Lecture Notes in Comput. Sci. 5515, 50–64.

Ghadimi, S., Powell, W., 2024. Stochastic search for a parametric cost function approximation: Energy storage with rolling forecasts. European J. Oper. Res. 312, 641–652.

Lee, J., Ko, C., Moon, I., 2024. E-commerce supply chain network design using on-demand warehousing system under uncertainty. Int. J. Prod. Res. 62, 1901–1927.

Mancini, S., Ulmer, M., Gansterer, M., 2023. Dynamic Assignment of Delivery Order Bundles to In-Store Customers. Technical Report. Working Paper 12/2023, Otto-Von-Guericke Universität Magdeburg.

Parodos, L., Tsolakis, O., Tsoukos, G., Xenou, E., Ayfantopoulou, G., 2022. Business model analysis of smart city logistics solutions using the business model canvas: The case of an on-demand warehousing e-marketplace. Future Transp. 2, 467–481.

Pazour, J., Unnu, K., 2018. On the unique features and benefits of on-demand distribution models. In: 15th IMHRC Proceedings (Savannah, Georgia. USA). pp. 1–9.

Powell, W., 2022. Reinforcement Learning and Stochastic Optimization. Wiley.

Romero, V.I.M., Tomes, L.L., Yusiong, J.P.T., 2011. Tetris agent optimization using harmony search algorithm. Int. J. Comput. Sci. Issues 8, 1–22.

Shi, Y., Yu, Y., Dong, Y., 2021. Warehousing platform's revenue management: A dynamic model of coordinating space allocation for self-use and rent. European J. Oper. Res. 293, 167–176.

Thul, L., Powell, W., 2021. Stochastic optimization for vaccine and testing kit allocation for the covid-19 pandemic. European J. Oper. Res. 304, 325–338.

Tornese, F., Unnu, K., Gnoni, M., Pazour, J., 2020. On-demand warehousing: main features and business models. In: XXV Summer School "Francesco Turco" - Industrial Systems Engineering. pp. 1–23.

Unnu, K., Pazour, J., 2019. Analyzing varying cost structures of alternative warehouse strategies. In: IISE Annual Conference and Expo 2019. pp. 480–485.

Unnu, K., Pazour, J., 2022. Evaluating on-demand warehousing via dynamic facility location models. IISE Trans. 54, 988–1003.

Unnu, K., Pazour, J.A., 2023. A large-scale heuristic approach to integrate on-demand warehousing into dynamic distribution network designs. Comput. Ind. Eng. 186, 109752.

Van Hentenryck, P., Bent, R., Mercier, L., Vergados, Y., 2009. Online stochastic reservation systems. Ann. Oper. Res. 171, 101–126.

Van Hentenryck, P., Bent, R., Upfal, E., 2010. Online stochastic optimization under time constraints. Ann. Oper. Res. 177, 151–183.

Zhong, Y., Pan, Q., Xie, W., Cheng, T., Lin, X., 2020. Pricing and wage strategies for an on-demand service platform with heterogeneous congestion-sensitive customers. Int. J. Prod. Econ. 230, 107901.