informs
https://pubsonline.informs.org/journal/opre

OPERATIONS RESEARCH

Vol. 00, No. 0, Xxxxx 0000, pp. 000–000

Submitted to *Operations Research*

ISSN 0030-364X, EISSN 1526-5463

Bootstrap Enhanced Scenario Optimization, a Case Study in Two-Echelon Logistics for Large-Scale Retail

Livio Fenga
Center for Simulation, Analytics and Modelling, University of Exeter Business School, Exeter, UK, L.Fenga@exeter.ac.uk

Vittorio Maniezzo
Department of Computer Science, University of Bologna, 47521 Cesena, Italy, vittorio.maniezzo@unibo.it

Abstract. We present a data-driven approach to scenario generation based on empirical distributions derived from univariate data series forecasts. The method is demonstrated using a real-world case study of supply chain optimization at the tactical level, focusing on the final stage of two-echelon logistics support for a large retail chain. The problem is modeled as a single-stage stochastic problem and framed as a prescriptive analytics case. Our approach uses Maximum Entropy Bootstrap (MEB) with bagging for univariate time series forecasting to predict demand scenarios that incorporate the inherent uncertainties of short, non-stationary time series data. These scenarios are then integrated into a deterministic equivalent model to optimize inventory allocation across distribution centers. The result is accountable and optimized prescriptions for inventory quantification and warehouse floor sizing. Our findings, which are also validated against an extended benchmark set of artificial instances, demonstrate the significant value of integrating MEB-enhanced forecasting with scenario-based optimization. They also provide the basis for a forecast-and-optimize framework, called Bootstrap Enhanced Scenario Optimization (BESO), whose generality extends beyond the specific use case.

Key words: Two-echelon logistics, Stochastic programming, forecasting, Two-echelon logistics, Maximum Entropy Bootstrap, Short time series, Combinatorial optimization

1. Introduction

Decision-making under uncertainty is a pervasive challenge in operations research, particularly in complex systems such as logistics networks. Scenario-based optimization has long been used to address this challenge by capturing the variability of future events through a finite set of possible

outcomes. However, the reliability of such methods depends heavily on how accurately these scenarios represent the underlying stochastic processes. In this paper, we propose a novel method — Bootstrap Enhanced Scenario Optimization (BESO) — that integrates bootstrap-based forecasting with scenario-based linear optimization to improve the robustness and realism of stochastic modeling. A new data-driven method for empirically estimating the distribution of stochastic variables and embedding it in scenario generation for robust and stochastic optimization may offer significant advantages over standard distribution-aware approaches. Traditional methods often rely on strong assumptions about the underlying probability distribution — typically normality or other parametric forms — which may not accurately reflect the true behavior of real-world uncertainties. In contrast, data-driven techniques directly leverage historical or observed data to construct empirical distributions, capturing nuances such as asymmetry, multimodality, or heavy tails that standard models might miss. This increased fidelity can lead to more realistic and effective scenario generation, improving both the robustness and relevance of the resulting optimization solutions. Furthermore, by adapting to the structure present in the data, data-driven approaches are naturally more flexible and can better accommodate nonstationarities or structural changes in the stochastic environment.

The BESO framework leverages bootstrapped time series to generate diverse and data-driven future scenarios, which are then embedded within constrained stochastic optimization problem, in which a stochastic objective function is optimized, possibly subject to robust, worst-case constraints. Unlike traditional stochastic programming methods that may rely on assumptions of distributional form or limited scenario diversity, BESO incorporates the empirical structure and seasonality of historical data to generate richer and more representative scenario sets.

To demonstrate the practical value and performance of BESO, we apply the method to a real-world two-echelon logistics network problem. In this setting, uncertain outbound freight demands must be managed across multiple depots and customer zones. By using BESO to define demand scenarios and solve for optimal routing and allocation decisions, we show how the method improves both solution robustness and operational efficiency compared to conventional techniques.

The remainder of the paper is structured as follows: Section 2 reviews previous works related to the different research areas intersected by this study. Section 3 describes the case study that motivated our research. Section 4 and 5 present the BESO methodology in detail. Section 6 discusses the computational results obtained, and Section 7 concludes with key insights and future research directions.

2. Literture review

The paper intersects different research threads, both in terms of the theoretical contribution and the application use case.

2.1. Scenario-Based Robust Optimization

Stochastic programming has long served as a fundamental framework for modeling decision-making under uncertainty. The early foundations of stochastic programming primarily considered expected-value models and recourse formulations **??**. In contrast, robust optimization **??** proposes to reformulate the model to a deterministic optimization problem where constraints consider a worst-case outcome. As research matured, hybrid models emerged paying attention to the interplay between constraint satisfaction and objective optimization, with major applications in fields such as energy systems, finance, supply chain management, and transportation planning.

A primary branch is Chance-Constrained Programming (CCP, also Chance-Constrained Optimization) **??** where uncertain constraints must be satisfied with at least a prescribed probability level, i.e., they must be satisfied for a high percentage of possible uncertain outcomes, rather than absolutely all of them. The survey in **?** provides an overview of CCP, considering also cases with limited distributional information and discussing reformulations based on sampling and distributional robustness.

On the contrary, hard-constrained stochastic programming **??** refers to models in which certain constraints must be satisfied in all cases, regardless of the realization of uncertain parameters. These constraints can for example represent physical, safety, or regulatory requirements. The nomenclature tends to be less normative in these cases, but usually when there is some tolerance on constraint violation the name becomes Robust chance-constrained optimization **?** and distributionally robust chance constraints when the distribution itself is uncertain **??**. The name distributionally robust optimization (DRO) is also used when looking for the solution most resilient against the worst case probability distribution for the uncertain parameters **?**. DRO has been proposed and applied in various logistics and supply chain contexts **??**.

A primary approach to stochastic programming is Scenario-Based optimization. This approach addresses uncertainty by discretizing the continuous probability distribution of uncertain parameters into a finite set of specific *scenarios*, each with an associated probability. The optimization problem is then formulated to find decisions that perform well across this set of scenarios.

The Sample Average Approximation (SAA) method is a pivotal technique that addresses this **?**. SAA approximates the true underlying distribution with an empirical distribution derived from a sample of independent and identically distributed realizations, i.e., scenarios. By replacing the expectation with a sample mean, the original stochastic problem can be transformed into a deterministic equivalent problem that can then be solved using standard techniques.

The optimization framework of interest for this paper, where we seek to minimize expected costs while ensuring that constraints are satisfied across all generated scenarios, belongs to the category of scenario-based robust optimization. This area is of significant scientific interest. Recent contributions include **?**, a paper introducing the Robust Stochastic Optimization Made Easy (RSOME) framework. This integrates scenario-tree-based stochastic linear optimization with distributionally robust optimization, and is available as a GitHub package **?**. **?** addresses the stochastic inventory routing problem by integrating scenario-based methods with distributionally robust optimization. It focuses on handling uncertainties in demand by considering various scenarios and ensuring robust solutions that perform well across these scenarios, and **?** proposes a scenario-based robust optimization approach that combines elements of stochastic programming and robust optimization. This approach emphasizes constructing probabilistic scenarios and protecting against adversarial perturbations within discrete uncertainty sets.

2.2. Forecast and optimize

The integration of forecasting and scenario-based optimization is not new *per se*. In fact there is a growing body of literature that combines the two, where typically time series generates forecast paths that become scenario sets. This integration goes by different names, including predict-then-optimize, stochastic programming with forecast scenarios, Forecast-informed optimization or Scenario-based stochastic optimization.

The typical workflow of these contributions goes through a forecasting phase, using time series models such as SARIMA, GARCH, LSTM, Prophet, XGBoost, etc. to generate point forecasts and/or predictive distributions, followed by a scenario generation phase that simulate or sample possible future paths based on forecast distributions and finally embed the scenarios into a stochastic program (single or multi-stage, see **???**) and more recently into adaptive robust optimization solutions (**?**). Unlike this traditional two-stage workflow, our bootstrap-based approach unifies forecasting and scenario generation: scenarios are not a post-processing step but emerge directly from the forecasting model itself. This integration eliminates the need for ad hoc scenario generation

techniques, ensuring that the optimization model operates on trajectories that are simultaneously consistent with historical dynamics and forecast uncertainty.

The sequential paradigm, in which forecasting models first estimate future parameters and the resulting predictions are then used in optimization models, has also been questioned, suggesting that it could result in suboptimal decisions when prediction errors significantly impact the optimization outcomes. Alternative, frameworks have been proposed where forecasting models are trained not only to minimize predictive error, but also to improve decision quality, with the optimization objectives embedded in the training loss of the forecasting models (**??**). However, this approach places a significant computational burden on model training that is not justified in cases where forecasts are sufficiently reliable, such as the one of interest to us.

2.3. 2-echelon logistics networks

Multi-echelon inventory management is a supply chain control strategy that coordinates inventory across multiple levels, called "echelons," of the chain (**?**). Each echelon — typically identified as suppliers, manufacturers, distribution centers, and retailers — may hold inventory. The objective is to optimize inventory and routing across all levels of the supply chain rather than addressing each level in isolation. The literature on this strategy has expanded significantly, covering various aspects such as inventory control policies, replenishment strategies, and inventory risk management, among others.

The implementation of effective forecasting and optimization processes represents a fundamental element of any well-defined and effective management policy. Specifically, in the context of 2-echelon logistics optimization, prescriptions grounded in predictive analytics are known to provide benefits by refining demand forecasting, inventory allocation, transportation planning, and related decisions (**???**). Forecasting, particularly demand forecasting, has garnered considerable attention in this field (**?**). Different forecasting paradigms have been proposed (**???**), demonstrating that accurate demand forecasts can be achieved by analyzing historical data and that it is possible to integrate these insights into management models. Nevertheless, precise demand forecasting, especially within the context of dynamic retail environments characterized by fluctuating trends and seasonal variations, remains a significant obstacle. Multi-echelon demand forecasting commonly employs time series analysis, incorporating a variety of models. However, the efficacy of these methods frequently hinges on assumptions of stationarity and specific distributional characteristics, which are often violated in empirical data. The application of prescriptive analytics in 2-echelon

logistics systems is still an area of research and presents several challenges, including how to deal with uncertainties in demand and supply, how to properly model historical series, and how to efficiently formalize the related management decision processes to be supported.

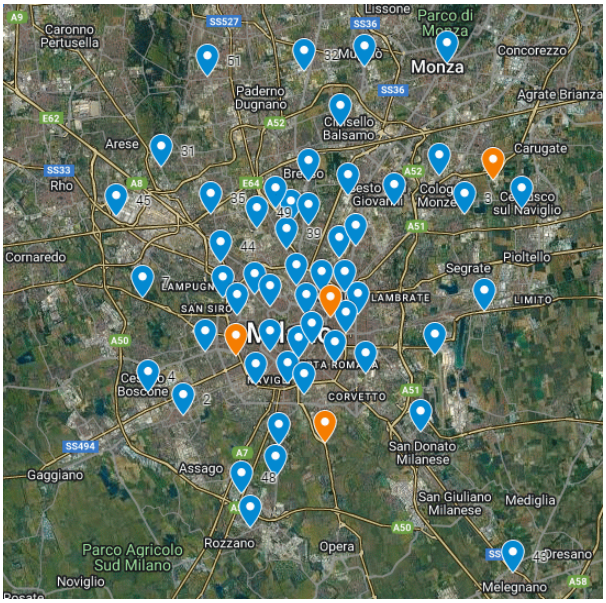
When we focus on the scope of our case study, effective multi-echelon inventory management, we face the objective of minimizing the average total inventory cost by promoting coordination and cooperation among supply chain members in terms of allocating stores to distribution centers (DCs). Each time a DC makes an allocation decision, it can seek advance demand information to reduce uncertainty and associated costs. This approach allows the DC to update its market forecast and anticipate the structural attributes of the model, ultimately providing valuable management insights (**?**). The majority of cases documented in the literature regarding the implementation of such multi-echelon inventory policies are based on examples of instances with generic demand, generated according to a variety of distributions (**??**). Our work presents a real-world use case in which all instance elements are derived from actual records.

3. The 2-echelon test case

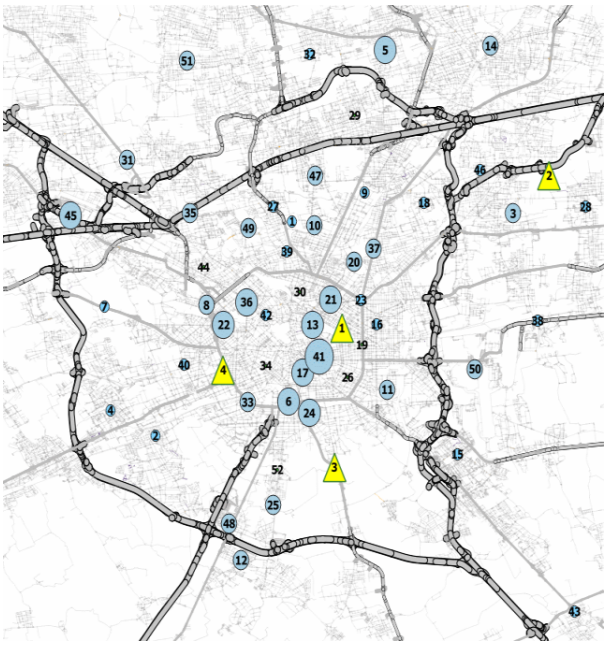
Our proposed test case comes from an LSRT company that manages multiple retail stores in a large city in northern Italy. Among these, 52 stores provided aggregate sales time series data spanning at least 45 months prior to the forecast period, which is set at a three-month horizon, as detailed in the following. Each store maintains a small inventory, replenished by deliveries from one of four different distribution centers (DCs) in the area. The products are categorized into three lines: fresh, dry, and household. In this context, the focus has been exclusively on the logistics of dry and household goods, including product categories ranging from personal care items to basic electronics, from household goods to holiday decorations. For this product line, DCs are required to maintain an aggregate inventory sufficient to cover a demand for a period of two weeks.

The supply chain consists of a large distribution center (DC) in the northern suburbs, a smaller DC in the south, and two small satellite facilities closer to the city center. While the satellites have limited storage capacity, they offer lower-cost and more timely service to the majority of the city's stores. Figure 1:a shows the (anonymized) geolocation of the DCs and stores and 1:b the GIS model of the locations (see also **?**), where the DCs are the orange triangles and the stores are the azure blobs whose size is proportional to the corresponding request.

Each store must be allocated to a distribution center (DC), ensuring that the total requests made to a DC do not exceed its storage capacity. Both capacities and requests are quantified in terms of



(a) gmap view



(b) GIS view

Figure 1 Stores and distribution centers locations.

pallets, with no further granularity at this analytical level. The cost of servicing a store from a DC is assessed based on the travel time required to transport goods between the two entities. Notably, the three smaller DCs are fully owned by the company, whereas the fourth DC is situated in a leased facility.

The ultimate goal of the analysis is to determine how much space to rent during the next peak demand period, the Christmas season. To achieve this, we first optimize the allocation of stores to DCs, taking into account transportation and storage costs, and from the solution it is immediate to derive the space required in each DC, therefore the space to rent in the largest one.

Another requirement characterizes the problem we are addressing: large stores located near small DCs often lead to rapid saturation of the capacity of central DCs. This forces nearby stores to be served by comparatively distant DCs. Therefore, it has been proposed to allow tentative split service for certain selected stores to facilitate what-if analysis. The problem we address thus involves a parametric number of possible assignments for each store, where which store should be allowed multiple (double) assignments is an input parameter, not a decision variable.

The analysis was conducted during the summer of 2022 in preparation for the subsequent Christmas season. At the time of the analysis, the anticipated demand requests were unknown and required forecasting. The analysis underwent iterative refinements throughout the summer,

culminating in the final forecast produced in September. Consequently, the data presented in this paper correspond to a three-month forecasting case.

The same data as this test case were used in (**?**), but a fundamentally different approach was taken in the previous work, a deterministic Lagrangian matheuristic that did not consider bootstrapping and its stochastic/robust optimization integration. In this paper, we therefore propose a genuinely novel approach that greatly expands the significance of the test case. The extended benchmark set is also entirely new.

All codes and data are available in an anonymized format from the project repository (**?**). Specifically, the file containing the request time series can be downloaded from the repository, where all series are complete up to the most recent actual values. However, it should be noted that the last three values were unknown at the time the final forecast was generated.

According to the classification proposed in **?** our contribution is described by the following typology string: 2,D,D,G|F,C|D,G|b,F,N|SC|E,O standing for: *System specification*: 2 echelons, Divergent material structure, Discrete events, Global information; *Resources specifications*: bounded storage, Constant delivery time; *Market specifications*: Discrete stochastic demand, Guaranteed service; *Control type specifications*: installation base stock policy, Flexible release quantities, No other means to satisfy unexpected requirements; *Performance specifications*: meeting service requirements and minimization of costs; *Generic scientific aspects*: Exact techniques, Optimization.

4. The predictive module

Data-driven approaches to estimating the distribution of stochastic variables have gained increasing attention as alternatives to classical parametric modeling, particularly in settings where assumptions of stationarity or known distributional forms are invalid. Unlike traditional techniques that rely on a priori specification of distribution families (e.g., Gaussian or Poisson), data-driven methods infer distributional characteristics directly from historical or observed data, making them especially suited for real-world applications with complex, noisy, or limited information.

Among the most prominent techniques are resampling-based methods such as the Bootstrap **?**, which generates empirical distributions by repeatedly sampling from the data with replacement. The Maximum Entropy Bootstrap **?** extends this idea by producing replicates that preserve dependence structures in time series data, making it particularly valuable for short or non-stationary sequences.

Recent work has explored the integration of these empirical distribution estimates into stochastic and robust optimization pipelines. Notable contributions include nonparametric scenario generation **?**, distributionally robust optimization using empirical Wasserstein distances **?**.

In our proposal, we utilize the Maximum Entropy Bootstrap (MEB) methodology in conjunction with the ensemble learning technique known as bagging (Bootstrap Aggregating). This combination enhances the accuracy and robustness of predictive models through the generation of multiple bootstrapped datasets from the original data. By training individual models on these datasets and aggregating their predictions through averaging, we adhere to MEB's principles while capturing the underlying statistical variability. Maximizing entropy under empirical constraints allows the integration of bagging with MEB to produce more robust models. In more details, each model trained on distinct bootstrapped samples incorporates uncertainty characteristics from the original data, allowing for a comprehensive representation of its variation.

The incorporation of MEB into the bagging framework not only bolsters the performance of predictive models but also facilitates uncertainty quantification in the predictions. Each model produces a set of predictions that can be used to construct confidence intervals, thereby offering insights into the variability and reliability of the forecasted outcomes. The ability to assess the spread of predictions from multiple bootstrapped datasets presents a more nuanced understanding of potential prediction intervals, which is particularly valuable in the context of the present paper, characterized by short, non-stationary, and possibly non linear time series surrounded by a significant amount of uncertainty.

Furthermore, this synergistic approach allows for dealing with noisy datasets and mitigating overfitting, as bagging inherently provides a mechanism to increase the robustness of the predictive model by leveraging the diversity across the bootstrapped samples. Since MEB generates bootstrapped datasets based on the maximum entropy principle, the resulting models avoid being overly sensitive to any one particular realization of the data.

Mathematically, if we denote the bootstrapped replicas of the dataset as $D_1^*, D_2^*, \ldots, D_B^*$, where B represents the number of bootstrapped samples, the resulting predictive framework can be expressed as: $\hat{y}_{BAG} = \rho(\hat{y}_b(x))$, where $\hat{y}_b(x)$ is the prediction made by the $b-th$ model trained on the bootstrapped sample D_b^* and ρ a generic centrality parameter. This averaging process effectively reduces variance by smoothing out the predictions of the individual models, leading to enhanced stability and generalization capabilities in the final aggregated model. The parameter ρ is often defined as either a simple average or the median, which leads to the following bagging predictors:

$$\hat{y}_{BAG} = \frac{1}{B}\sum_{b=1}^{B}\hat{y}_b(x) \tag{1}$$

$$\hat{y}_{BAG} = \begin{cases} \hat{y}_b(x)\left[\frac{h+1}{2}\right] & \text{if } h \text{ is odd} \\ \frac{\hat{y}_b(x)\left[\frac{h}{2}\right]+\hat{y}_b(x)\left[\frac{h}{2}+1\right]}{2} & \text{if } h \text{ is even} \end{cases} \tag{2}$$

here, $\hat{y}_b(x)$ is sorted in ascending order whereas h is the number of future values to be predicted.

4.1. Resampling techniques

In recent years, resampling techniques, especially the bootstrap method, have become valuable for overcoming the limitations of conventional forecasting methods (**?**). The bootstrap method creates simulated time series through resampling from observed data. This approach is useful for non-stationary, non-linear, intermittent, and asymmetric cyclical patterns, as well as low signal-to-noise scenarios (**???**). Moreover, resampling methods help address the effects of limited sample sizes (**?**) and model uncertainty (**?**).

The utility of bootstrap methods in model selection has been extensively validated. Fenga and Politis (**?**) introduced a bootstrap-based approach for autoregressive moving average (ARMA) order selection, demonstrating its effectiveness in enhancing model accuracy with finite sample sizes. Their work (**?**) extended this to threshold autoregressive models (SETAR), validating bootstrap techniques for optimal model specifications. Furthermore, Fenga and Politis (**?**) explored the LASSO technique for sparse autoregression, illustrating a bootstrap framework for model selection in high-dimensional contexts. These studies collectively show how bootstrap methods can enhance order selection across statistical modeling frameworks, underscoring their relevance in modern statistical analysis.

The standard bootstrap assumes independent and identically distributed (i.i.d.) data — an assumption that fails for time series, where observations are dependent and ordered.

This failure is more pronounced when the series is short, making dependence patterns more influential, the data exhibits autocorrelation, seasonality, or trend and when resampling destroys the structure and underrepresents variability.

In these cases Maximum Entropy Bootstrap (MEB) guarantees the following advantages:

- *Preservetion of Dependence Structure* MEB maintains the temporal dependence (e.g., auto-correlation, trend, seasonality) by using rank-based and interpolation techniques that reorder and smooth the data. Unlike i.i.d. bootstrap, it doesn't break time series structure.
- *Better Handling of Short Time Series* In short samples, structural properties (e.g., local trends) are easily lost by standard resampling. MEB keeps these features by reconstructing plausible time series paths that are consistent with observed data and uncertainty.

- *No Need for Model Specification* Unlike block bootstrap or parametric resampling, MEB is nonparametric and doesn't require selecting block sizes or fitting ARMA models — a key benefit for short samples where such models may be unstable or overfitted.
- *Generation of Smooth Pseudo-Series* MEB produces continuous, smoothed pseudo-series through linear interpolation, making it well-suited for statistical inference tasks like forecasting, confidence intervals, and hypothesis testing.
- *Maximum Entropy Principle* It applies the maximum entropy principle, meaning the generated bootstraps are the least biased (i.e., most non-committal) estimates consistent with the known constraints (the observed data). This is especially valuable when little information is available — like in short series.
- *Support of Stationary and Non-Stationary Series* MEB is flexible with both stationary and non-stationary time series, whereas other bootstrap techniques often require stationarity or explicit transformation to achieve it.

The resampling scheme adopted here is called Maximum Entropy Bootstrap (MEB) (**?**). MEB offers a principled and flexible approach to bootstrapping time series data, leveraging the Maximum Entropy Principle (MEP) (**?**) to address the inherent limitations of conventional bootstrap methods, thus addressing many limitations of traditional block bootstrap methods. The essence of MEP is grounded in the idea that in the absence of specific information, the most rational probability distribution to adopt is one that maximizes entropy, reflecting the greatest uncertainty about the system (**?**). This is mathematically articulated through Shannon entropy, defined as $S(p) = -\sum_{i=1}^{n} p_i \log(p_i)$, where p_i denotes the probability associated with the i-th outcome within a discrete sample space of size n.

The pivotal advancement provided by MEB lies in its ability to formulate bootstrapped samples that adhere to a spectrum of constraints dictated by the original dataset, such as predetermined moments, probabilities, or other statistical characteristics (**?**). Specifically, under typical circumstances, these constraints can be specified in the form of normalized probabilities and empirical expectations, expressed as $\sum_{i=1}^{n} p_i = 1$ to ensure that the probabilities form a valid distribution, and $\sum_{i=1}^{n} p_i f_i = \bar{f}$, where f_i represents measurements or statistics relevant to the analysis, with $\bar{f}$ being the empirical mean of these measurements. The formulation of the Maximum Entropy problem consequently leads to a constrained optimization framework. The optimization problem can be succinctly encapsulated in the Lagrangian: $L(p, \lambda, \mu) = S(p) + \lambda \left(1 - \sum_{i=1}^{n} p_i\right) + \mu \left(\bar{f} - \sum_{i=1}^{n} p_i f_i\right)$, where λ and μ are Lagrange multipliers that adjust the balance under the constraints of normalizing

the distribution and matching the empirical mean, respectively. The solution to this maximization problem yields a probability distribution, which appears as $p_i = \frac{e^{\mu f_i}}{\sum_{j=1}^{n} e^{\mu f_j}}$, where the parameter μ is determined through the constraint equations, specifically via a numerical root-finding approach that satisfies the moment condition $\sum_{i=1}^{n} p_i f_i = \bar{f}$.

This formulation transforms the problem into one that allows for flexible adaptation to various types of data and constraints imposed by the empirical observations. Once this maximum entropy distribution is established, resampling via bootstrapping can commence by selecting m samples according to the derived p_i. This produces synthetic datasets that not only reflect the variability and uncertainty characteristic of the original data but also adhere to the specific constraints imposed, which may include higher moments or even domain-specific statistics.

This resampling process can be iterated to generate a comprehensive set of bootstrapped samples, enabling a multitude of statistical analyses, including confidence interval estimation, hypothesis testing, and model validation (**?**). We adopted this procedure to generate the bootstrap sets of dataseries upon which scenarios were defined as detailed in the following section.

5. The prescriptive module

The results of the predictive module become a structural component of the prescriptive module, whose goal is to optimize the allocation of stores to DCs thus to permit the quantification of space to rent in a third-party DC. The results of the optimization are used to evaluate the modeled *scenario*, where scenarios are defined for specific what-if analyses. For example, the objective of the what-if analyses could be the effect of freezing some allocations while leaving the rest free, or possibly accepting some multiple, in our test case double, allocations for some selected stores.

The core optimization problem is a split allocation problem. This variant of the basic allocation problem has already received attention in the optimization literature, mainly in the context of survivable communication network design (**??**), and more generally as the splittable capacitated multiple allocation hub location problems (**?**), and finds applications in fiber optic access networks (**?**), split delivery routing (**?**), and even in school timetabling (**?**).The problem we are interested in is closely related to one that is presented in (**?**), but it is approached in a very different way.

The mathematical model that we use is a direct extension of the standard model for the allocation problem. As the framework we propose is valid beyond the specific case study described in the paper, in the following we will refer to the application elements (stores and data centres) using standard allocation terminology. We will use the term 'clients' for stores and 'servers' for data

centres to emphasise the generality of the analysis. In our context, assignment constraints can accept a parametric number of assignments, enabling requests to be serviced across multiple servers. The notation used in the model is as follows:

- n number of clients (stores)
- J index set of clients, $J = \{1, \ldots, n\}$
- m number of servers (DCs)
- I index set of servers, $I = \{1, \ldots, m\}$
- c_{ij} global cost for allocating client $j \in J$ to server $i \in I$
- Q_i storage capacity of server $i \in I$
- d_i unit storage cost of server $i \in I$
- req_j total request of client j.
- b_j maximum number of servers (DCs) client j can be assigned to
- x_{ij} binary decision variable equal to 1 iff client j is allocated to server i, $i \in I$ and $j \in J$
- q_{ij} integer decision variable representing the amount (number of pallets) of request of client j provided from server i

In the final test case, the amount requested by each client was always set to be equal to the corresponding predicted value, but in some scenarios, the requests of some clients could be explicitly split between two servers, so the requested q_{ij} amounts could vary.

The mathematical model used for this problem was as in the following formulation FD.

$$(FD) \qquad z_{FD} = \textbf{min} \sum_{i \in I} \sum_{j \in J} (c_{ij} x_{ij} + d_i q_{ij}) \tag{3}$$

$$\textbf{subject to} \sum_{i \in I} q_{ij} = req_j \qquad j \in J \tag{4}$$

$$\sum_{j \in J} q_{ij} \leq Q_i \qquad i \in I \tag{5}$$

$$\sum_{i \in I} x_{ij} = b_j \qquad j \in J \tag{6}$$

$$q_{ij} \leq req_j x_{ij} \qquad i \in I, j \in J \tag{7}$$

$$x_{ij} \in \{0, 1\} \qquad i \in I, j \in J \tag{8}$$

$$q_{ij} \in \mathbb{Z}_0^+ \qquad i \in I, j \in J \tag{9}$$

Constraints 6 are formulated as equalities because of the need to impose specific splits in what-if settings. If this is not of particular interest, these constraints could be replaced by less-than-or-equal-to inequalities combined with inequalities that force allocations to at least one server.

Note that when $b_j = 1$ for all clients, i.e., when each client $j \in J$ can be served by only one server, constraints 7 and 4 force the entire quantity requested by client j, req_j, to be supplied by a single server. The constraints 7 thus become equations, the constraints 4 become redundant, and the constraints 5 can be expressed by replacing $\sum_{j\in J} q_{ij}$ with $\sum_{j\in J} req_j x_{ij}$. Problem P reduces to the standard GAP and can be solved by any of the methods described in **?**.

However, we are interested in the general case that allows some multiple assignments, and since the cost is derived from the assignments, the problem can be considered a fixed-cost, split-assignment allocation problem. Moreover, since future client requests can only be forecast, the problem can be more conveniently formulated as a stochastic optimization problem (**??**), where future requests are random variables ρ_j, $j \in J$, whose distributions can be derived from the forecast results. The problem becomes a single-stage stochastic problem, since there is no subsequent recourse, whose formulation is as follows.

$$(FS) \qquad z_{FS} = \textbf{min} \sum_{i\in I}\sum_{j\in J}(c_{ij}x_{ij} + d_i q_{ij}) \tag{10}$$

$$\textbf{subject to} \sum_{i\in I} q_{ij} = R(\rho_j) \qquad j \in J \tag{11}$$

$$\sum_{j\in J} q_{ij} \le Q_i \qquad i \in I \tag{12}$$

$$\sum_{i\in I} x_{ij} = b_j \qquad j \in J \tag{13}$$

$$q_{ij} \le R(\rho_j)x_{ij} \qquad i \in I, j \in J \tag{14}$$

$$x_{ij} \in \{0,1\} \qquad i \in I, j \in J \tag{15}$$

$$q_{ij} \in \mathbb{Z}_0^+ \qquad i \in I, j \in J \tag{16}$$

where the values $R(\rho_j)$ in the constraints 11 and 14 refer to any same realization of the random variable ρ_j.

Different methods can be used to deal with this case (**?**). We transformed the formulation (FS) into its *deterministic equivalent* (**?**) using the forecasts computed on the bootstrap sets. This

transformed the original stochastic problem into a deterministic optimization one. We generated multiple scenarios each containing a realization of the stochastic variable ρ, which is defined as a forecast of one series from the bootstrap set in use. For each scenario s, we thus created a version of the constraint with b_i^s as the right-hand side. The final solution must satisfy these constraints across all scenarios.

The objective function now incorporates the various quantities that must be delivered to each client in different scenarios. We therefore consider the expected cost as a weighted average across the scenarios and include it in the objective function.

The resulting formulation is as follows. The random variables ρ_j are sampled in a finite number of scenarios. Let ρ_j^s denote the realization of ρ_j in scenario s, $s \in S$, and let $p_s = \frac{1}{|S|}$ be the probability associated with scenario $s \in S$. The variables ϵ_{js} are slack variables for the customer service constraints.

ATTENTION: The following displayed equation, in its current form, exceeds the column width that will be used in the published edition of your article. Please break or rewrite this equation to fit, including the equation number, within a column width of 240 pt / 84.67 mm / 3.33 in (the width of this red box).

$$(DE) \qquad z_{DE} = \textbf{min} \sum_{s\in S}\left(p_s \sum_{j\in J}\left(\sum_{i\in I}(c_{ij}x_{ij} + d_i q_{ij}^s) + M\epsilon_{js}\right)\right) \tag{17}$$

$$\textbf{subject to} \quad \sum_{i\in I} q_{ij}^s + \epsilon_{js} = \rho_j^s \qquad j \in J, s \in S \tag{18}$$

$$\sum_{j\in J} q_{ij}^s \le Q_i \qquad i \in I, s \in S \tag{19}$$

$$\sum_{i\in I} x_{ij} = b_j \qquad j \in J \tag{20}$$

$$q_{ij}^s \le \rho_j^s x_{ij} \qquad i \in I, j \in J, s \in S \tag{21}$$

$$x_{ij} \in \{0,1\} \qquad i \in I, j \in J \tag{22}$$

$$q_{ij}^s \in \mathbb{Z}_0^+ \qquad i \in I, j \in J, s \in S \tag{23}$$

$$\epsilon_{js} \in \mathbb{Z}_0^+ \qquad j \in J, s \in S \tag{24}$$

In this formulation, since each variable ρ_j^s derives from a specific realization of the corresponding random variable ρ_j, the amounts to be delivered to each store $j \in J$ are scenario-specific. However, it is required to determine a unique allocation that allows to serve all stores in any proposed scenario. This last requirement may be infeasible, so the slack variables ϵ_{js} are needed in the formulation. They are lexicographically penalized in the objective function by appropriately high costs, in order to implement the primary concern of achieving feasibility.

Optimizing formulation (DE) makes it possible to determine the quantities that each server must be able to supply to each client, i.e. the total quantity that each server must handle. In terms of our logistical application, since each DC must maintain sufficient inventory to meet the anticipated demand during the designated period, the optimization of the client (retailer) allocation translates directly into a quantification of the DC inventory and therefore of its required minimum size. The solution of the allocation problems permits therefore to directly solve the dimensioning question that motivated this research.

6. Computational results

The predictive and prescriptive modules are run in sequence; the first generates elements for the second. This section reports the results obtained when validating each module according to the previously described specifications.

6.1. Predictive module tests

The objective of the predictive module is to forecast demand for each store, asking for a forecast three months ahead. The available data consists of a time series for each of the 52 stores. Each series contains 45 values corresponding to four years of monthly data on relevant requests for each store, minus the values for the last three months to be forecast.

The core approach set out in this paper is the bootstrap method, which is described in Section 4. The results obtained using this method were validated against those obtained using several alternative forecasting algorithms: SARIMAX, Holt-Winters, MLP, RNN (LSTM), Random Forest and Gradient Boosting (XGBoost). The relevant hyperparameters were determined by grid search for integer-valued algorithms such as SARIMAX, and by Hammersley sampling (**?**) for real-valued algorithms.

All series were pre-processed using a 'log-diff' transform to approach stationarity, and a max–min scaler was applied to the artificial intelligence-derived methods. Python version 3.11 was used for

all implementations and the codes were run on a Windows PC workstation equipped with four Intel Core i7-4790 CPUs running at 3.60 GHz and with 32 GB of RAM.

The focus of this part of the research is on the bootstrap method. To this end, we tested the approach with the following settings:

- bootstrap sets containing 75, 125 and 175 series;
- bootstrap sets generated by simple autoregressive AR(p) with p=5 model (the default in python's statsmodels), where parameters are estimated using Yule-Walker equations (**?**), and by ARIMA, where autoarima is run on each of the seed series and the optimized orders are then applied to each boosted series;
- the forecasting of all the boosted series generated as above to obtain the boosted forecasts was tested with AR(p), Random Forest or ARIMA. In the case of ARIMA, autoarima was only repeated on the first series of each bootstrap set. In the case of AR, the parameter p was set for each series after a loop on different p values, keeping the one that maximised the corresponding AIC value.

We also attempted to generate bootstrap sets with and without backcasting, as well as with simple extraction and repetitions among the residuals of the seed series, and compared these with their scrambling. We did not report the results of these tests because backcasting worsened the results, while scrambling the residuals showed no difference compared to extraction with repetitions. Therefore, we decided to use the latter approach, in line with the literature.

The forecasts were compared using three cost functions: mean absolute error (MAE), mean square error (MSE) and bias. The results are reported by ranking the corresponding forecasts for each series according to the quality of the considered function, to emphasise how the algorithms perform relative to each other, and then averaging the rank of each function over all series. The Appendix reports absolute values for the error measures, not rank-based values, for all boostset sizes. Table 1 presents the results of comparing the forecasts obtained using standard algorithms with those obtained using boosting. The boosted series were computed using an AR(5) model based on Yule-Walker equations. The forecasts of these series were obtained using a simple autoregressive model. The boosted sets consisted of 75, 125, and 175 series, which are labelled YW_AR_75, YW_AR_125, and YW_AR_175, respectively.

The columns show the average ranking of each cost function (MAE, MSE or bias) for each boostset size, while on the rows we report:

- $fcast\_avg$: boost forecast obtained as the average of the corresponding ranks;

	YW_AR_75			YW_AR_125			YW_AR_175		
	MAE	MSE	bias	MAE	MSE	bias	MAE	MSE	bias
fcast_avg	2.67	2.67	0.08	2.58	2.58	0.05	3.83	3.83	0.04
fcast_50	3.04	3.04	0.06	3.19	3.19	0.04	2.50	2.50	0.03
yhw	5.42	5.42	0.08	6.06	6.06	0.04	5.88	5.88	0.03
yarima	5.73	5.73	0.05	6.04	6.04	0.03	6.08	6.08	0.02
ysvm	5.88	5.88	0.08	5.50	5.50	0.05	5.50	5.50	0.03
ylstm	6.02	6.02	0.06	5.52	5.52	0.04	6.27	6.27	0.02
yar	6.06	6.06	0.04	6.54	6.54	0.02	6.79	6.79	0.02
ymlp	6.38	6.38	0.05	5.98	5.98	0.03	6.17	6.17	0.02
yrf	6.46	6.46	0.11	6.31	6.31	0.06	5.48	5.48	0.04
yxgb	7.33	7.33	0.12	7.29	7.29	0.07	6.50	6.50	0.05

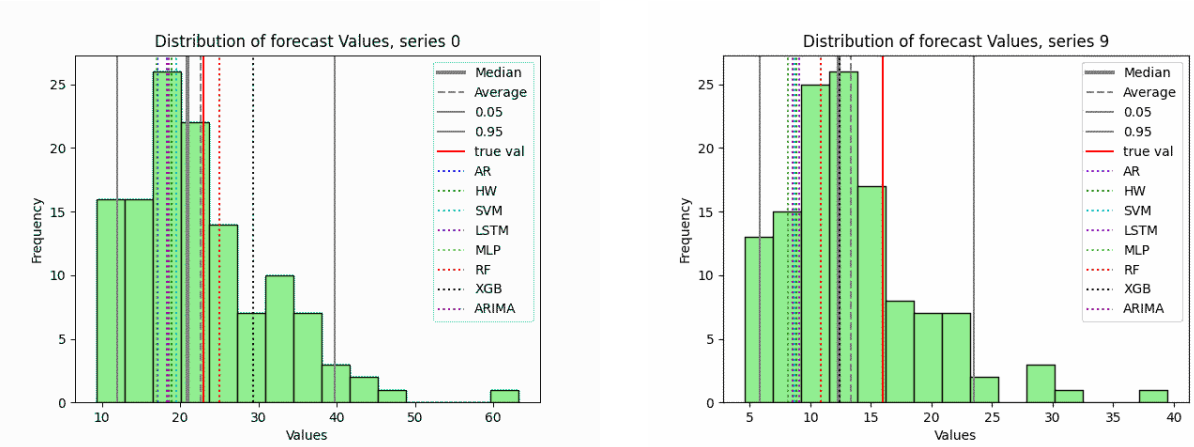
Table 1 Average rankings for different boostset size.

- $fcast\_50$: boost forecast obtained as the median of the corresponding ranks;
- yar: non-boost forecast obtained by an AR model;
- $yarima$: non-boost forecast obtained by an ARIMA model;
- yhw: non-boost forecast obtained by a Holt and Winters model;
- $ymlp$: non-boost forecast obtained by a multilayer perceptron model;
- $ysvm$: non-boost forecast obtained by a SVM model;
- $ylstm$: non-boost forecast obtained by a LSTM model;
- yrf: non-boost forecast obtained by a random forest model;
- $yxgb$: non-boost forecast obtained by a XGboost model;

The table shows that boosting is consistently the most reliable method for minimizing forecast errors, even when forecasts are obtained from a model as simple as AR[p], while ARIMA produces the least bias.

Figure 2 shows two examples of forecast distributions obtained for two different series. The green bars correspond to the distribution of the results obtained by the AR[p] model over the differently recolored alternatives of a series. Lines representing the results of other forecasting algorithms applied to the original series are superimposed on the distribution.

The relative merits of the different forecasting approaches are better highlighted by critical difference diagrams based on the non-parametric Friedman test followed by a post-hoc Nemenyi test. They allow for a visual comparison of performance ranks, and show whether differences between classifiers are statistically significant (**?**). The graphs, generated in our case by their SciKit version (**?**), plot the average rank across all series for each algorithm along the x-axis and connect by a horizontal bar those that could not be considered statistically different.



(a) 75 bootset *(b) 125 boostset*

Figure 2 Distribution of bootstrap forecasts.

Figures 3 and 4 show a comparison of the ranks on MAE for the 75 series boostset and on MSE for the 175 series boostset. In both cases, as in all those tested, the accuracy of the boost forecasts is significantly higher than that obtained with alternative models. The tables in the Appendix confirm that the comparison gives consistent results across all boostset sizes and cost functions, except for bias. The bias rankings are in fact unrelated to those obtained with the other cost functions, but this is due to the fact that the bias is very small in all cases, less than 1% of the average of the empirical values, which ensures that all the forecasting models are free of systematic errors on our data and makes the difference in the rankings insignificant.

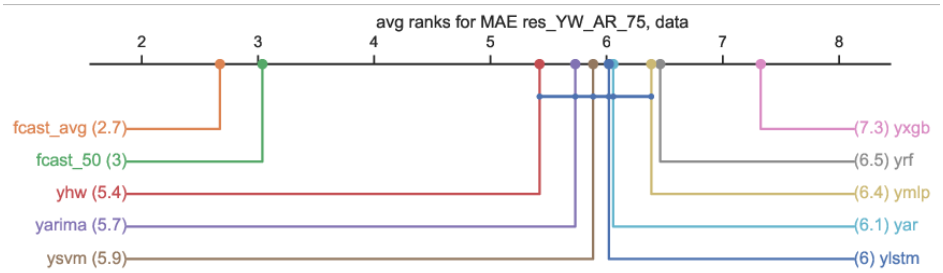


Figure 3 Average ranks for MAE on instance res_YW_AR_75

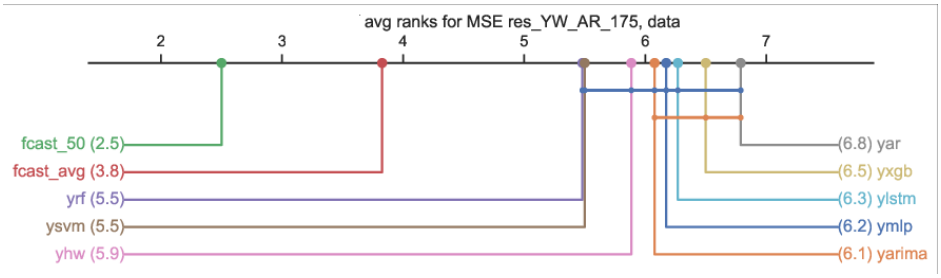


Figure 4 Average ranks for MSE on instance res_YW_AR_175

Our relatively short time series require a forecasting approach that can handle both limited data and non-stationarity. Although sophisticated machine learning models such as recurrent neural networks (RNNs) and gradient boosting machines (GBMs) have become more popular in recent years, they can perform less well when there is limited data available, particularly when the data has non-stationary properties. This can lead to overfitting. The autoregressive (AR) model combined with maximum entropy bootstrap offers a powerful and promising alternative. This technique performed excellently in the context of this study, consistently outperforming more complex machine learning models. It offers an approach based on robust statistical principles that mitigates the risk of overfitting inherent in many AI approaches. Moreover, the AR model combined with the bootstrap methodology provides reliable forecasts ideally suited to integration into the subsequent optimization module.

In addition to its superior accuracy in our case, the choice of an autoregressive (AR) model offers several crucial advantages over more complex machine learning approaches. Firstly, the AR model's inherent simplicity contributes to its ease of implementation and interpretation. Unlike many machine learning algorithms that operate as 'black boxes', the parameters of the AR model have clear statistical interpretations that provide insight into the underlying dynamics of the time series data. Secondly, the well-established theoretical framework of the AR model and the readily available diagnostic tools allow for robust validation and assessment of model performance.

6.2. Prescriptive module tests

The results reported in section 6.1 suggested instantiating the optimization on the results provided by the bootstrap forecasts. The prescriptive module was first run in a deterministic setting, to define a benchmark solution, then extended to the stochastic setting described in section 5.

When the problem to be solved is modeled as in the (FD) formulation, using the average predictions obtained by boosting as deterministic data, the instance is easily solved by any state-of-the-art MILP solver, thus ensuring the viability of dealing with a significant deterministic equivalent of the stochastic formulation (FS). The optimal deterministic value, z_{FP} =15553, is used as a benchmark in the following tests.

We moved on to the (DE) formulation generating a value for each of the stochastic variables ρ_j in correspondence to each bootstrap forecast. Given these input data, we ran experiments to assess the impact of the variation of the remaining sets of parameters of the formulation, i.e., cardinality of the boostsets, number of splittable clients, and assumptions on inventory costs.

All reported tests were run on the same machine as in subsection 6.1, a Windows PC workstation equipped with four Intel Core i7-4790 CPUs running at 3.60 GHz and 32 GB of RAM, but in this case the implementation language was C++ to allow finer control over the MILP solver used, which was CPLEX v. 22.11.

Increasing boostset size The size of the boostset helps to stabilize the results during forecasting, but it has a deep impact on the optimization. In fact, while the number of variables and the number of constraints of the formulation DE do not depend on the boostset size, larger boostsets are more likely to contain unusual values or combinations thereof, thus making constraints (18) progressively harder to satisfy, thus more costly, and possibly rendering the instance infeasible.

We conducted experiments to ascertain this using 75, 125 and 175 series boostsets and a 600 second time limit on the optimization process. Figure 5 shows a boxplot diagram depicting the distribution of the optimized cost over a number of tests equal to 1/5 of the number of series in the boostset, i.e., 15, 25, 35, respectively.

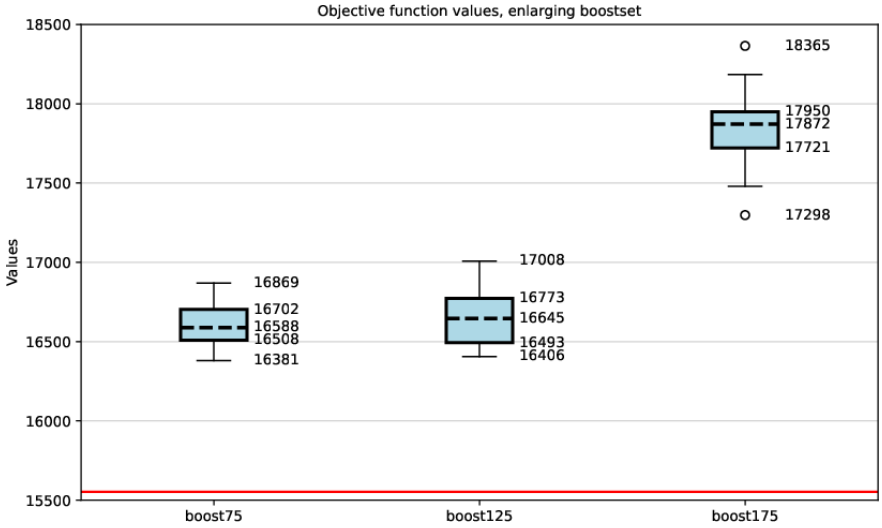


Figure 5 Allocation cost for increasing boostset size

Figure 5 has a red horizontal line at the value of the cost of the deterministic case. The increase in cost can therefore be considered as a quantification of the value of the information about the storage requests. We notice that the cost is essentially stable in the cases of the 75 and 125 series sets, but increases significantly in the case of the 175 boostset. This is due to the increased difficulty of the larger boostset instances, which leads to some instances being solved only heuristically within the time limit. All instances were feasible, but 7 out of 35 still had a gap between lower and upper bound after 600 seconds of CPU time.

Finally, we note that if we take the optimal solution obtained with the coefficients imposed by the 75 series and evaluate it with the coefficients of the 125 series, we obtain (as expected) a more costly solution than the optimal one computed with the coefficients of the 125 series, but more interestingly, we have a solution that is not feasible with the coefficients of the 175 series.

Increasing number of splittable clients A defining feature of the problem we face is the possibility of accepting split service for some clients. This is unusual for allocation problems and results from the requirement to perform specific what-if analyses on the results. The number of clients allowed to split, and which ones they are, is given in input. In the actual case, the service to a client, if splittable, could be split between at most two servers, and the quantities delivered had to be integer, as specified in all formulations..

The clients to split are chosen from the most requesting clients. We ran a series of tests based on 75 series boostsets, allowing the 0, 4, 8, and 12 most requesting clients to be split. Figure 6 shows the corresponding boxplots.

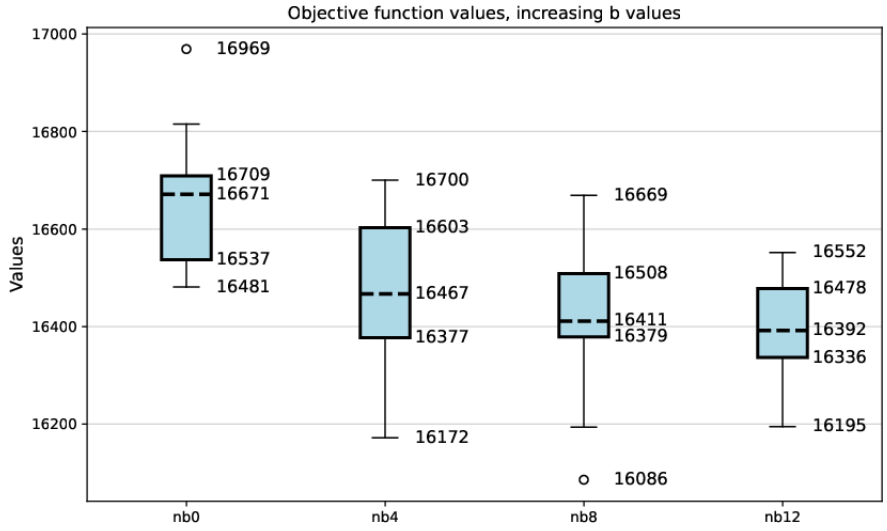


Figure 6 Allocation cost for increasing number of splittable clients

Allowing the service to a client to be split is a partial relaxation of the corresponding assignment constraint, so the optimal cost is expected to decrease. Indeed, this can be seen in figure 6, where the more clients are allowed to split, the lower all distribution moments are. However, the marginal decrease becomes smaller as the number of splittable clients increases, since splitting a small client may result in only a small contribution, if any.

Impact of different inventory cost assumptions A further set of tests was conducted to determine the impact of inventory costs d_i, $i \in I$ on the search process. Recall that in the original

case study, one of the objectives was to determine how much space should be rented in the only DC not owned by the company, where storage space needs to be leased. The case study reported in **?** did not include storage costs. Here, we consider storage costs that are inversely proportional to the average distance of the DC from the stores. We consider three settings, one with no storage costs, one where only the most distant DC incurs costs, and one where all DCs need to lease space. Costs are proportional to the quantities stored and are therefore influenced by the allocation policy.

All tests were performed on the 75 series boostset. The interest of the analysis does not lie in the increase of the optimal costs, since an increase of the costs of the decision variables clearly leads to an increase of the global costs, but rather in the variation of the complexity of the search, as it is reflected in the CPU time to reach the final result, and in the optimality or feasibility of the solutions. To assess the effectiveness of increasingly longer searches, we report results after 600, 1200 and 3600 seconds of CPU time.

The results are summarized in the table 2. All results are based on 15 repetitions, and the columns show:

- n: number of clients (stores);
- m: number of servers (DCs);
- n_{boost}: number of series in the boostset;
- n_q: number of servers incurring leasing costs;
- n_{inf}: number of infeasible solutions at the end of the search;
- GAP lin: average gap between the linear bound and the best solution at the end of the search;
- GAP fin: average gap between the best lower bound and the best solution at the end of the search;
- t.CPU: average CPU time (in sec) of the search process.

Table 2 shows an interesting feature of the fitness landscape of this split-cost allocation problem, the complexity of the search is not monotonic with respect to the number of leased servers. In fact, CPLEX was able to solve all instances to optimality in the case of 0, 3, and 4 leased servers, while in the case of 1 or 2 leased servers it was never able to prove the optimality of the best solution found within the allowed time limit of 3600 seconds. Moreover, the instances with only 1 leased server are more difficult than those with 2, as evidenced by the gap between the lower and upper bounds, both between the linear relaxation bound and the best solution found, and between the best lower bound at the end of the search and the best solution. This feature is further discussed in subsection 6.3.

n	m	n_{boost}	n_q	n_{inf}	GAP lin	GAP fin	t.CPU
52	4	75	0	0	0.01	0.00	333.95
52	4	75	1	1	0.18	0.17	600
52	4	75	1	0	0.15	0.14	1200
52	4	75	1	0	0.14	0.13	3600
52	4	75	2	0	0.04	0.04	600
52	4	75	2	0	0.01	0.03	1200
52	4	75	2	0	0.01	0.03	3600
52	4	75	3	0	0.01	0.00	211.03
52	4	75	4	0	0.01	0.00	13.00

Table 2 Gap and CPU time for increasing number of leased servers.

As a final note, we point out that the objective function was structured to lexicographically first strive for feasibility and then optimize within the feasible domain, in fact all instances always have resulted in feasible solutions. By reducing the cost coefficient of the request slack variables, it is possible to explore a trade-off between solution cost and the probability of reaching an infeasible solution. However, we did not explore this further.

Test case solution To further evaluate the benefits of the proposed approach, we compared the effectiveness of the robust BESO-optimized solution with that obtained using a parametric method that assumed a Gaussian distribution of the forecasts of the request datasets. This comparison was run against scenarios derived from a Gaussian distribution based on the confidence intervals of the forecasts provided by a Holt-Winters ETS model (HKOS08).

Figure 7 shows the distributions of the forecasts of the 75 boostset that BESO and ETS obtained. Note that the BESO distribution, which is data-driven, spans a wider range of values than the ETS distribution.

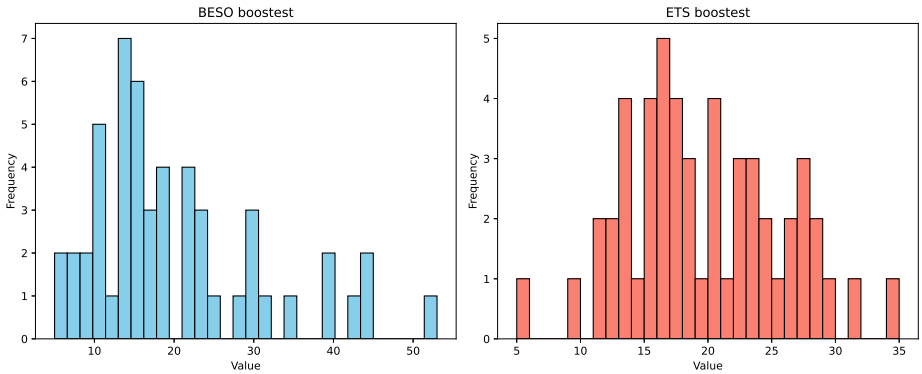


Figure 7 Comparison of distributions on 75 bootstrapped series

	No rental cost		With rental cost	
	BESO	ETS	BESO	ETS
Assign. Cost	17781	15423	5446720	4378380
Optimality gap%	0	0	16.07	0.69
Infeasibilities	0	1	0	1
t.Cpu	33	29	3600	3600

Table 3 Solution cost with and without rental cost

	No rental cost		With rental cost	
	BESO	ETS	BESO	ETS
DC 0	67 / 120	119 / 120	82 / 120	116 / 120
DC 1	571 / 1000	430 / 1000	532 / 1000	428 / 1000
DC 2	257 / 300	301 / 300	257 / 300	303 / 300
DC 3	128 / 180	173 / 180	152 / 180	176 / 180

Table 4 DC usage with and without rental cost

The wider distribution has an impact on the proposed allocation. We had empirical proof of this when we tested the BESO and ETS solutions on the validation data obtained by the end of the year — i.e. when we finally received the data on which we were asked to provide a forecast.

Table 3 presents a comparison of solutions in the first two columns when no rental cost was added for the use of leased floor space and in the last two columns when the rental cost was included in the objective function. As expected, given that the constraints were less restrictive, the ETS solution is less expensive in both cases. However, in both cases, the solution was infeasible as it did not provide all the goods requested by one of the retailers. This result was obtained in a single instance; therefore, it may not have happened. However, it is significant that, even in the first instance in which we tested our model, its robustness proved important in dealing with a limit case.

Table 4 details the proposed usage of the DCs inthe different cases. It appears that DC 2, the biggest owned one, is the overloaded one in both cases, even though by small amounts.

Finally, we note that Table 3 shows that including the rental cost makes otherwise identical instances much harder to solve. In fact, the time limit of 3600 CPU seconds was too short to reach optimality, and a duality gap was left in the proposed solutions, a substantial one in the case of the BESO instance.

6.3. Extended deterministic benchmark set

All analyses reported in sections 6.1 and 6.2 refer to the case study at the heart of this paper, but the interest of the method is not limited to this particular case study as it can be effective on any instance of this 2-echelon supply chain problem. To test the generality of the results, we generated a benchmark set with varying dimensions and coefficients. The study was conducted only with respect to the deterministic setting, and is therefore significant for assessing the computational complexity of the split-allocation problem as a deterministic combinatorial optimization problem.

We generated 5 instances for each configuration of number of clients (n), number of servers (m), and number of splittable clients. The travel costs were obtained by randomly selecting m rows and n columns from a large distance matrix, whose values were computed as the actual road distances between points located in the same area as the test case. Server inventory costs were inversely proportional to the average travel cost from the server to the clients and client requests were inversely proportional to the average travel cost from the client to the servers. Note that the distance matrix is asymmetric, as it was computed within a city area. The number n_b of splittable clients was progressively increased as the number n_q of servers with nonzero inventory costs. Specifically, the range of the defining parameters were:

- n: {50, 52, 100, 200, 300};
- m: {4, 5, 10, 25, 50};
- n_b: {0, 4, 5, 10, 25, 50}
- n_q: {0, 1, 2, 3, 4, 5}

Not all combinations of values were generated, the larger values were used only for instances with higher number of clients. In particular, the values 4 and 52 were only used to generate instances with the same dimensionality as the case study, in order to check the generality of the results obtained. The first instance of this set was kept to be the case study one.

The computational results in the case of no costly server, reported as averages over the 5 generated instances, are summarized in table 5, whose columns show:

- n, m: number of clients and number of servers;
- n_b: maximum number of splittable clients;
- $ncols$, $nrows$: number of columns and number of rows in formulation FD;
- $gaplb$: average percentage gap between the linear relaxation lower bound and the cost of the best solution found;

- $gapfinal$: average percentage gap between the best lower bound at the end of the search (smallest cost of an unexpanded node) and the cost of the best solution found;
- $ninf$: number of instances for which no feasible solution was found;
- $nopt$: number of instances solved to proven optimality;
- $tcpu$: average CPU time for solving the instance, time to optimality or 3600 if optimality could not be proven;

The table shows two trends. One, with respect to instance size, shows an expected increase in complexity. While instances the size of our test case are easily solved to optimality by modern solvers, increasing the size, as defined by the number of variables and the number of constraints, quickly leads to instances that could not be solved to optimality within the 3600-second time limit. The number of splittable clients has an important effect on the complexity of the instance. Instances of the same absolute size, but with more splittable clients, usually have a looser linear relaxation (LP) bound and have more difficulty reaching optimality, as can be seen by the decreasing number of instances out of the 5 from each group that could be solved to optimality.

Note that the optimality gap of the LP bound is usually small, except for the n=100 and m=50 instances, but the search still struggles to close this gap. Interestingly, the LP bound gets tighter as n increases, but this does not make the search any easier. The least-cost solution when optimality was not proven was usually found within the first 600 seconds of CPU time. However, we note that designing efficient heuristics for this problem is beyond the scope of this research. We could have tweaked the solver configuration to favor heuristic search at the expense of efficiency in proving optimality, or designed custom heuristics, but the focus here is on assessing the fitness landscape when road network costs are involved.

Table 6 reports results about tests varying the number of costly servers, the columns show:

- n, m: number of clients and number of servers;
- $ncols$, $nrows$: number of columns and number of rows in formulation FD;
- lb0, t0: linear relaxation bound and CPU time in the case of no costly servers;
- lb1, t1: linear relaxation bound and CPU time in the case of 1 costly server;
- lb2, t2: linear relaxation bound and CPU time in the case of 2 costly servers;
- lb3, t3: linear relaxation bound and CPU time in the case of 3 costly servers;
- lb4, t4: linear relaxation bound and CPU time in the case of 4 costly servers;
- lb5, t5: linear relaxation bound and CPU time in the case of 5 costly servers;

n	m	n_b	$ncols$	$nrows$	$gaplb$	$gapfinal$	$ninf$	$nopt$	$tcpu$
52	4	0	416	316	0.21	0.01	0	5	0
50	5	0	500	355	0.39	0.00	0	5	3
50	5	5	500	355	0.40	0.00	0	5	4
50	10	0	1000	610	1.32	0.00	0	5	383
50	10	5	1000	610	1.34	0.00	0	5	378
50	10	10	1000	610	1.35	0.13	0	4	1150
100	10	0	1000	1210	0.50	0.12	0	2	1920
100	10	5	1000	1210	0.54	0.05	0	3	1514
100	10	10	1000	1210	0.57	0.09	0	3	2063
100	50	0	10000	5250	5.23	3.06	0	0	3600
100	50	5	10000	5250	4.77	2.70	0	0	3600
100	50	10	10000	5250	4.55	2.42	0	0	3600
100	50	25	10000	5250	4.52	2.61	0	0	3600
200	10	0	4000	2410	0.17	0.05	0	2	2369
200	10	5	4000	2410	0.17	0.09	0	1	2893
200	10	10	4000	2410	0.18	1.01	0	0	3600
200	50	0	20000	10450	1.66	1.26	0	0	3600
200	50	10	20000	10450	1.68	1.29	0	0	3600
200	50	25	20000	10450	1.71	1.37	0	0	3600
300	10	0	6000	3610	0.09	0.03	0	3	1798
300	10	5	6000	3610	0.09	0.07	0	0	3600
300	10	10	6000	3610	1.02	0.47	0	0	3600
300	50	0	30000	15650	0.94	0.73	0	0	3600
300	50	10	30000	15650	0.93	0.76	0	0	3600
300	50	25	30000	15650	0.96	0.84	0	0	3600

Table 5 Extended benchmark set, deterministic case, no costly server

n	m	ncols	nrows	lb0	t0	lb1	t1	lb2	t2	lb3	t3	lb4	t4	lb5	t5
52	4	416	316	0.86	0.2	0.17	0.81	0.08	0.22	0.01	0.02	0.00	0.02	-	-
50	5	500	355	1.33	0.06	2.77	0.06	1.46	0.13	0.11	14.31	0.06	7	0.05	0.93
100	5	1000	705	0.41	0.24	0.09	6.39	0.06	3.2	0.01	0.06	0.02	4.8	0.02	2.71
200	5	2000	1405	0.14	1.43	0.01	1.79	0.01	0.25	0.01	0.17	0.01	0.11	0.01	7.84
52	4	416	316	0.86	0.36	0.24	0.22	0.22	0.14	0.26	0.09	0.05	1.9	-	-
50	5	500	355	1.33	0.16	0.52	0.07	0.50	5.8	0.13	10.11	0.05	1.37	0.02	0.22
100	5	1000	705	0.41	0.33	0.19	8.58	0.04	6.14	0.02	0.41	0.02	0.58	0.02	3.78
200	5	2000	1405	0.14	1.55	0.07	39.07	0.04	9.44	0.01	0.12	0.02	4.48	0.01	6.66

Table 6 Extended benchmark set, deterministic case, increasing number of costly servers

The table reports results on two sets of instances. The top 4 rows contain aggregate results on a set of instances, 5 for each configuration, where inventory costs are directly proportional to average travel distances, while the bottom 4 rows have inventory costs inversely proportional to average travel distances.

The complexity of solving instances increases non-monotonically with the number of costly servers, for both increasing and decreasing inventory costs. While the 0-cost instances are comparatively easy to solve, the CPU time required to prove optimality increases with the addition of the first costly servers and then decreases as more servers are paid to store inventory. However, this is not true of the gap between the linear relaxation bound and the cost of the optimal solution, where the instances with no costly servers always have the largest gap. As expected, the CPU time required to prove optimality increases with the dimension of the case, and the case with an inverse correlation between storage cost and travel cost is more difficult to solve than the case with a positive correlation.

7. Conclusions and future directions

This paper uses a case study in distribution logistics to propose a general framework for prescriptive analytics called Bootstrap Enhanced Scenario Optimization (BESO). The case study is based on a two-echelon supply chain in which retail stores are allocated to distribution centers (DCs). The problem analyzed was how to minimize the allocation cost and the corresponding aggregated demand for each DC when retailer requests are forecast based on historical data. This process enables us to determine the inventory required for each DC and the size of the DCs needed to accommodate it.

The study advocates the integration of forecasting and optimization. We propose basing the forecasting phase on bootstrapped autoregressive models, as these have been shown to be more effective than alternative methods. Bootstrapped forecasting identifies the probability distribution of future retailer requests, thereby supporting the stochastic optimization of allocations while taking into account both costs and forecast uncertainty. This integration eliminates the need for ad hoc scenario generation techniques, ensuring that the optimization model operates on trajectories that are simultaneously consistent with historical dynamics and forecast uncertainty. We thus contribute a methodological bridge that reduces the gap between predictive modeling and decision-making under uncertainty, offering both conceptual simplicity and practical robustness.

We considered cases where clients had to be fully allocated to a server, as well as a split allocation version. We validated our framework using transportation logistics data from a real-world use case, as well as artificial data to better analyse the fitness landscape and reveal which parameters most impact instance complexity. We obtained results showing how complexity changes when different parameters defining problem instances vary.

We believe that this study will be of wide interest, both because it is an innovative application of predictive analytics, and because the Maximum Entropy Bootstrap (MEB) forecasting methodology and the deterministic equivalent based on the bootstrapped series can offer valuable insights from a managerial perspective.

Future research could explore integrating MEB methods into preprocessing procedures, such as Box-Cox transformations, to enhance predictive accuracy. Additionally, investigating ensemble methods, including weighted averaging, may yield robust predictions in variable environments. The interplay between predictive analytics and real-time decision-making deserves attention, particularly regarding how strategies informed by bootstrapped predictions influence logistics efficiency. Lastly, the closure properties of the MEB framework allow for enhancements, such as incorporating Bayesian prior information, refining estimates by integrating external knowledge. Exploring hybrid models that merge Bayesian updating with maximum entropy may yield richer representations of uncertainty and enhanced predictive accuracy.

Appendix. Bootstrap sets error costs

This section reports the absolute values of the various cost functions, which were calculated using the forecast errors obtained by averaging the corresponding errors for each of the 52 test case series described in the paper.

The functions reported in the columns are: bias, Means Absolute Percentage Error (MAPE), Mean Error (ME), Mean Absolute Error (MAE), Mean Percentage Error (MPE) and Root Mean Square Error (RMSE). Table 7 gives the cost values for the case of a boost set composed of 75 series, table 8 for the 125 series and table 9 for the 175 series sets.

All tables show significantly consistent results across all boost set sizes.

model	bias	MAPE	ME	MAE	MPE	RMSE
fcast_50	-0.27	0.09	-1.02	1.63	-0.05	2.12
fcast_avg	0.02	0.07	0.07	1.25	0.01	1.68
yar	0.36	0.34	1.36	5.86	0.07	9.75
yhw	0.63	0.34	2.41	5.99	0.13	10.59
ysvm	0.69	0.36	2.66	6.20	0.14	10.99
ylstm	0.56	0.35	2.13	6.01	0.11	10.41
ymlp	0.47	0.36	1.79	6.26	0.09	10.60
yrf	1.50	0.44	5.73	7.79	0.31	13.54
yxgb	2.28	0.58	8.74	10.12	0.49	17.05
yarima	0.44	0.34	1.67	5.80	0.09	9.77

Table 7 Error costs for the 75 series boostset.

model	bias	MAPE	ME	MAE	MPE	RMSE
fcast_50	-0.56	0.09	-1.08	1.59	-0.06	2.25
fcast_avg	0.02	0.07	0.05	1.18	0.01	1.65
yar	-0.59	0.25	-1.13	4.36	-0.07	6.31
yhw	-0.13	0.24	-0.26	4.33	-0.02	6.39
ysvm	-0.06	0.24	-0.11	4.18	-0.01	6.39
ylstm	-0.30	0.24	-0.58	4.15	-0.04	6.34
ymlp	-0.47	0.24	-0.90	4.22	-0.05	5.99
yrf	1.27	0.28	2.44	5.01	0.14	7.48
yxgb	2.52	0.38	4.85	6.52	0.28	9.46
yarima	-0.40	0.25	-0.76	4.44	-0.05	6.83

Table 8 Error costs for the 125 series boostset.

model	bias	MAPE	ME	MAE	MPE	RMSE
fcast_50	-0.04	0.07	-0.07	1.22	0	1.51
fcast_avg	0.99	0.12	1.90	2.04	0.12	2.76
yar	-0.22	0.35	-0.42	5.84	-0.03	7.64
yhw	0.25	0.34	0.48	5.75	0.02	7.90
ysvm	0.30	0.33	0.58	5.66	0.03	7.76
ylstm	0.10	0.34	0.20	5.76	0	7.74
ymlp	-0.19	0.34	-0.36	5.69	-0.02	7.57
yrf	1.72	0.37	3.31	6.39	0.18	9.66
yxgb	3.01	0.46	5.78	7.96	0.33	11.83
yarima	-0.04	0.34	-0.08	5.69	-0.01	7.63

Table 9 Error costs for the 175 series boostset.