Politecnico di Torino

Politecnico di Torino

Master's Thesis in Management Engineering
November 2025

Optimizing Capacity Planning in Logistics by means of A Stochastic Bin Packing Model

Supervisor:
Prof. Guido Perboli
Co-supervisor:
Prof. Sara Khodaparasti

Candidate:
Ludovica Marino

Alla curiosità,

Che la mia famiglia mi ha insegnato a coltivare,

E che ogni giorno ritrovo nelle persone che mi stanno accanto.

Abstract

In real-world logistics and supply chain operations, decision-makers are often required to allocate resources before full information about demand is revealed, forcing them to make refined decisions once that data becomes fully available. This thesis addresses this challenge by proposing a two-stage stochastic model for a variant of the Variable Cost and Size Bin Packing Problem with Stochastic Items. The model reflects a realistic setting in which bins differ in cost and size, and aims to minimize total costs while ensuring feasible and efficient packing. The algorithm developed captures the trade-offs between early commitments and later adjustments under uncertainty, incorporating key constraints such as heterogeneous costs, item incompatibility, and multiple capacity dimensions. An extensive experimental plan is adopted to examine how variations in instance structure influence capacity planning, and to assess the impact of different levels of variability among the stochastic parameters associated with the items.

Table of Contents

List of figures

List of tables

1. Introduction

1.1 Background

Operations Research (OR) is a discipline that resists a single, universally accepted definition. It can be understood both as a body of problems, methodologies, and solutions, and as an applied process characterized by a systematic approach. Its fundamental role is to support decision-making through scientifically grounded methods to address complex real-world problems. OR relies on mathematical modeling, in which complex systems are represented by equations, inequalities, and functions, with the goal of identifying optimal solutions (Maggioni et al., 2016).

The modeling process typically involves three key stages. Formulation requires creativity and expertise to capture the most relevant aspects of the system in a simplified model. Deduction focuses on deriving solutions through rigorous algorithmic and mathematical techniques, ensuring precision and objectivity. Interpretation involves translating model outcomes into practical actions, a phase that demands human judgment to address aspects overlooked in earlier stages. Linear programming is a fundamental tool in Operations Research for modeling optimization problems (Tadei & Della Croce, 2019). It is based on an objective function—to be maximized or minimized—defined over a set of decision variables subject to constraints expressed as linear equations or inequalities. These variables, which may be continuous, integer, or binary, represent the system's controllable parameters. The objective function typically reflects economic goals, such as minimizing costs or maximizing profits, while the constraints delimit the feasible region of the problem. In this way, linear programming translates complex decision problems into mathematical models that allow systematic evaluation and identification of optimal solutions.

As such, OR models are powerful tools that combine creativity, mathematical rigor, and careful interpretation. They provide decision-makers with data-driven support for optimizing processes and strategies. Central to this discipline is the concept of optimization, which not only underpins OR itself

but also shapes contemporary society, where continuous improvements in technology and quality of life need ever more efficient tools and methods.

1.2 Context and motivation

Modern supply chains involve complex networks and frequently unpredictable demand patterns, forcing companies to make tactical capacity-planning decisions under uncertainty. Firms are required to make ex-ante decisions regarding the quantity of capacity units—such as shipping containers, vehicle loads, or warehouse space—to secure, accounting for the related fixed costs, yet in the absence of complete information concerning critical parameters, as the eventual volume of goods to be allocated or stocked.

This challenge can be effectively addressed through variants of the classical Bin Packing Problem (BPP), which models the allocation of a given set of items into the fewest possible number of bins (e.g., boxes, containers, trucks, shelves) without exceeding each bin's capacity. The BPP constitutes a fundamental NP-hard combinatorial problem with broad applications in scheduling, routing, and resource allocation (Martello & Toth, 1990). Traditionally, BPP assumes that all input parameters – such as item sizes, bin capacities, and costs - are known and fixed. In practice, however, planners frequently face uncertainty concerning item characteristics and future demand. Deterministic models that presuppose complete information are often inadequate, because they usually impose requirements that cannot be realistically satisfied. This gap between deterministic models and stochastic realities has motivated and led to numerous extensions of BPP that incorporate uncertainty in item sizes, availability, and related parameters, thereby fundamentally altering the nature of bin packing.

This frequent situation naturally maps onto a bin-packing problem with stochastic items, where capacity can be determined beforehand, but the set of "uncertain items" that serve as the pivotal factor in the problem can only be discovered in hindsight. The Variable Cost and Size Bin Packing Problem

with Stochastic Items (VCSBPSI) was proposed to formalize this class of problems (Crainic et al., 2014). In this problem formulation, decision-makers select a combination of capacity available before demand is realized and later respond with a recourse action if the planned capacity falls short. Since packing decisions are made tactically in advance and then adjusted when uncertainty resolves, the problem naturally fits a two-stage stochastic optimization framework, adequate to a variety of macro-logistics and supply chain contexts where bin-type flexibility and stochastic demand are inherent. The two-stage approach with recourse (Birge & Louveaux, 1999) is a cornerstone of stochastic bin packing models, as the recourse allows for an alternative to one-stage robust models, which produce a single packing solution that must remain feasible for all allowed variations, leaving extra slack for safety at the expense of higher costs.

1.3 Research aim

In light of the above, this thesis aims to extend the VCSBPSI by introducing additional constraints, with the objective of enhancing its flexibility and applicability across diverse real-world scenarios. The proposed elaboration is intended to ensure that the model can better adapt to current and emerging market dynamics and, consequently, to the correlated evolving logistical requirements. Furthermore, various solution approaches will be examined to evaluate their ability to achieve optimality, both in terms of the objective function and computational efficiency, with their performance benchmarked against a heuristic approach.

The objective of the experimental campaign is to assess the practical value of adopting a stochastic programming model and to investigate whether a fundamental structure for capacity planning and the dependence of the plan on attributes of the problem setting can be identified.

This work will start with a section regarding the possible practical applications of the model in question, followed by a chapter about the state of the art of bin packing problems, and then by a detailed problem description of the

VCSBPSI with additional constraints developed for the objective of this thesis, with a focus on the mathematical model. Then, the solution method will be presented, alongside a Best Fit Decreasing heuristic algorithm for performance comparison. In the final section, results will be analyzed in depth, highlighting the benefits of different solution methods and demonstrating that a two-stage stochastic approach is more advantageous in the presence of critical uncertainty.

2. Practical Applications

2.1 Logistics, transportation, and supply chain

Common applications of a two-stage stochastic bin packing model are inspired by logistics and supply chain, where packing problems generally arise in capacity planning, transportation, and warehousing.
A few key practical application fields of the VCSBPSI include intermodal transport, 3PL capacity contracting, and city logistics with e-commerce.

Intermodal freight transportation involves moving goods via a sequence of different transportation modes (e.g., trucks, trains, barges, ships) in a door-to-door journey (Li et al., 2015). A benefit of the approach is freight consolidation, which enables achieving higher load factors and lower unit costs. Consolidation centers combine shipments from multiple sources into outbound loads. From a packing perspective, the center operator must decide how many trucks or containers to have ready each period to consolidate incoming goods. If arrival volumes are uncertain, a stochastic bin-packing model can determine an optimal "buffer" (e.g., always have one extra truck on standby to accommodate demand spikes, because the expected cost of occasional rentals is lower than the risk of shipment delays).
Indeed, intermodal shipments can generate significant economies of scale through consolidation and better capacity utilization (Tawfik & Limbourg, 2018). The VCSBSPI framework plays a crucial role, as the benefits only materialize when capacity is planned and allocated wisely at the tactical level. However, demand variability makes this challenging; therefore, the best option is to approximate a plan as accurately as possible. By considering multiple bin types, the model captures real intermodal choices, while on the other hand, by considering stochastic items, it protects against variability in shipment volumes. Additionally, another practical application where a multi-period bin packing model (Crainic et al., 2021) could be useful is for consolidation along transport corridors, where regular shipments occur and irregular shipments can be handled with spot capacity, while evaluating at the same time the benefit of advanced planning across multiple periods, for the firm's marginality.

Another field of logistics to consider is third-party logistics providers (3PLs). In such relationships, the shipper (e.g., a manufacturer or retailer) negotiates medium-term contracts with a 3PL to ensure that sufficient transport and storage capacity will be available when needed (Aissaoui et al., 2007). These contracts often specify both the quantity and type of capacity reserved and the logistics services to be performed (Crainic et al., 2014). Tactical capacity contracting thus involves deciding how much capacity to book with the 3PL and in what forms, before actual shipment orders are realized. This contracting problem is inherently a multi-criteria decision balancing cost, service, and risk, as on the one hand, securing more capacity may guarantee lower unit costs and the ability to serve peak demand; on the other hand, over-committing might mean paying for unused capacity if demand underperforms relative to forecasts. Once again, by applying the VCSBPSI formulation to 3PL contracting, each possible capacity unit can be treated as a bin with a fixed cost and a size. The shipper must choose an optimal set of bins to minimize the expected total cost, while accounting for actual usage that may differ from projected levels. As (Crainic et al., 2014) highlight, the traditional deterministic assumption that all parameters of the decision are known at planning time is rarely valid and is unrealistic for capacity contracting, given the time delay between planning and operations. For instance, if demand exceeds the contracted agreement, the shipper must be able to purchase additional capacity on the market at a higher price. This commonly encountered situation fits the recourse scenario that VCSBPSI is designed to handle. The outcome should indeed be a tactical plan that balances the upfront contract costs against the penalty of unmet demand. By solving this as a two-stage stochastic bin packing problem, the firm can decide on an optimal mix of container bookings that minimizes expected total shipping cost, while keeping overflow —spot-market containers— rare. If demand turns out to be higher, the plan also tells the company how to use costly backup options. Therefore, tactical capacity contracting with 3PLs perfectly exemplifies the need for stochastic bin-packing models to help the firm decide on a portfolio of capacity under uncertainty, to satisfy future logistics needs at minimal cost and risk. Analogously, VCSBPSI can naturally accommodate multi-sourcing in procurement by including bins from different

providers, ensuring that the model chooses an optimal mix and allocation to minimize costs and risks (Aissaoui et al., 2007).

Furthermore, it is worth considering the application field of city logistics, which refers to the planning and execution of freight distribution in dense urban areas (Crainic et al., 2023). In this sense, urban environments pose special challenges: severe congestion, tight delivery time windows, regulations on vehicle types and emissions, and limited loading space. Concurrently, urban consumers – both businesses and residents – demand fast and frequent deliveries, a trend increasingly amplified by the rise of e-commerce. This surge puts enormous pressure on city distribution systems, exposing inefficiencies in traditional last-mile operations and warehousing practices (Liang et al., 2025). Similarly to the previously cited real-life applications, the VCSBPSI framework closely corresponds to this setting. A city logistics planner needs to ensure sufficient capacity to carry all items while minimizing costs and maintaining high service quality amid aggressive competition in e-commerce. Once again, bin type flexibility is crucial, as using a single vehicle type would be inefficient or infeasible given the diversity of urban deliveries. Selecting the right mix of those vehicle types to match forecasted demand resembles choosing bin types in the tactical plan. Additionally, high day-to-day variability and seasonality mean that a static plan can easily fail to meet expectations or waste resources. Therefore, stochastic planning helps by accounting for variability in the number of orders per day or per area. In parallel, robust capacity planning is also essential in city logistics because delivery reliability is a key performance metric; failed or delayed drop shipments due to a lack of capacity can seriously hurt service levels and incur penalties. Building on this topic, urban last-mile delivery is a key example, where the "bins" are not physical containers but rather the available transportation companies (TCs) or courier services the provider can use, each with its own cost and service quality profile (Baldi et al., 2019).

Beyond transport capacity, warehousing and packing are also part of e-commerce logistics that tie into the bin packing problem. (Liang et al., 2025) highlight that online retail orders involve diverse item combinations and real-time decision making, which traditional methods struggle to handle. From a

capacity planning standpoint, one might model the box stocking problem as a VCSBPSI by deciding how many of each box size -bin type- to order ahead of a sales season, without knowing the exact order composition. While this is a more fine-grained application, it underscores how stochastic bin packing concepts permeate e-commerce operations, from warehouse packing to delivery fleet planning, by precisely tailoring capacity to anticipated demand and providing flexible backup options. Taken together, the examined real-world applications—particularly those in city logistics and online retail—demonstrate the critical role of demand-driven, flexible planning in managing contemporary supply chains (Crainic et al., 2023). These problems can be seen as instances of the Variable Cost and Size Bin Packing Problem with Stochastic Items, reinforcing the model's relevance. By accommodating multiple bin types and stochastic item streams, the problem provides a unifying framework to approach capacity planning in freight systems and e-commerce fulfillment. Embracing such stochastic models is critical in these settings because deterministic plans would either often fail or be overly costly, whereas a stochastic solution seeks a cost-efficient plan that performs well across many possible futures.

2.2 Other industry fields

Moreover, beyond logistics, bin packing with uncertainty finds use in several other fields. One notable example is cloud computing and resource allocation. (Cohen et al., 2017). Here, "bins" might be servers or virtual machines with specific capacities (CPU, memory), and "items" are tasks or applications that require those resources. Uncertainty comes from the tasks' resource usage, which might be stochastic. For instance, assigning virtual machines to physical servers is a bin-packing problem; if actual workloads fluctuate, one might use a chance-constrained bin-packing model to allow a small probability of overflow on each server -overcommitment- to increase utilization safely (Yan et al., 2022). Researchers have indeed looked at robust bin packing for cloud services, where the aim is to pack more virtual machines on a server than guaranteed by worst-case analysis, while still remaining within safe limits with

high probability, while preventing joint placement of interfering applications (Martinovic et al., 2022).

3. Literature Review

The Bin Packing Problem (BPP) and its numerous extensions represent one of the most fundamental and widely studied classes of combinatorial optimization problems in operations research. At its core, the BPP consists of assigning a set of items, each with a specific size, to a finite number of bins of limited capacity to minimize the total number of bins used or the overall packing cost. This apparently simple formulation captures a vast range of real-world applications, as seen in the chapter above, and, at the same time, serves as a cornerstone for the development of both exact and heuristic optimization methodologies. Over the years, the classic deterministic BPP has evolved into a rich family of variants, each designed to capture additional layers of realism relevant to logistics and decision-making. More recent developments have incorporated realism even further, by introducing uncertainty, leading to stochastic and robust variants such as the Variable Cost and Size Bin Packing Problem with Stochastic Items (VCSBPSI), a two-stage stochastic formulation that explicitly models the decision process under demand and size variability (Crainic et al., 2014).

From a methodological perspective, this field has served as a fertile testing ground for a variety of optimization techniques, as well as heuristics and metaheuristics, due to the combinatorial explosion inherent to real-world problem sizes. In parallel, approximation algorithms and competitive analysis have provided theoretical performance guarantees for simplified or online versions of bin packing, further enriching the algorithmic landscape. Across the literature, a central theme emerges: the trade-off between optimality and practicality. As uncertainty and realism are incorporated, exact optimal solutions become less attainable, prompting the evolution from simple, single-dimension deterministic models toward complex formulations that integrate stochasticity, heterogeneous bin types, and multi-period decision structures—each inspired by real-world use cases, ensuring that research remained relevant to industry needs.

Accordingly, this literature review emphasizes conceptual and practical insights rather than formal derivations, illustrating how classical bin packing

theory has been extended to handle stochastic, multi-stage, and constraints-rich environments. These developments collectively shape the algorithmic framework and research direction adopted in this thesis.

Firstly, (Crainic et al., 2011) introduces efficient lower bounds and heuristics for the Variable Size Bin Packing Problem (VSBPP), consolidating a variant of the BPP by introducing variability in bin sizes. This matter deviates the following literature from standard Bin Packing Problem formulation to variability, and later on uncertainty, in the problem scenarios, as well as additional constraints to the basic formulation (Eliiyi & Eliiyi, 2009). Later on, (Baldi et al., 2012) introduce the Generalized Bin Packing Problem (GBPP), considering logistics applications, and therefore creating the need to consider both bin costs and item packing profits, splitting the item into compulsory and non-compulsory items. This problem lays the foundation for future advanced packing problems with elaborate mixed objectives, creating an example of problem scenarios that increasingly align with real cases. Additionally, it also yields a bridge between bin packing problems and knapsack paradigms (Martello & Toth, 1990). This research will be drawn upon by (Baldi et al., 2019) with the Generalized Bin Packing Problem with bin dependent item profits (GBPPI), used in urban last-mile parcel delivery service, where the objective is to minimize the cost due to the bins used minus the profit obtained from the loading of the items into the bins, which becomes a problem of trade-off between cost – bins used – and quality – number of items loaded -, demonstrating the flexibility of bin packing models to handle multi-criteria objectives as well as being a direct example of practical use in logistics. On the topic of additional constraints for the BPPs, we have numerous works in the literature that present those, such as early work from (Dawande, 2001), which introduces a color (class) constraint on BPP formulation, meaning that two or more items of the same color may not be placed into the same bin. Later, type and item incompatibility constraints will be further explored by (Bender et al., 2015), who work on a Separable Assignment Problem (SAP), meaning that items may be duplicated in subsets to find feasible fitting in bins. Moreover, (Capua et al., 2018) work on a Bin Packing Problem with Conflicts (BPPC), introducing again incompatibilities among items, though leaving the incompatibilities more to an instance matter rather than the

mathematical problem formulation itself. Alongside this works, also (Anand and Guericke, 2020) do research on mixing constraints, setting limits to the mix of items in the same bin, while (Ekici, 2022) studies the Variable-Sized Bin Packing Problem with Conflicts and Item Fragmentation (VSBPPC-IF), which has applications such as the delivery of incompatible items using a fleet of heterogeneous vehicles where split delivery is allowed. There, a set of items can be fragmented, and each fragment can be packed into a separate bin, even though the fragments of conflicting items cannot be packed into the same bin. Lastly, (Tsao et al., 2024) analyze another BPP with incompatible product categories, such as only one type of box can be selected for each bin, thus diverting the objective function into minimizing the total unused space of all the used bins rather than into minimizing the usage costs. Furthermore, (Christensen et al., 2017) present a survey in online and offline settings for multidimensional bin packing problems. One of their relevant takeaways is that introducing additional dimensions or constraints typically increases problem complexity, often requiring problem-specific heuristics or sacrificing some optimality for computational tractability.

Alongside these findings, on a parallel path, the literature offers a vast collection of works regarding different methods of approach, to align with the variability of scenarios and different realizations, across different moments in time (Perboli et al., 2014). (Crainic et al., 2014) introduces for the first time the Variable Cost and Size Bin Packing Problem with Stochastic Items, where items are drawn from a stochastic set rather than a deterministic one, and the algorithm's behavior involves recourse over time, simulating a tactical plan during uncertainty and an operational plan once reality is known. Stochasticity is also introduced through the use of scenarios to simulate different cases achievable, which are further explored. This work is also a strategic turning point, blending bin packing with supply chain contract planning.

Similar reasoning is highlighted in (Bódis & Balogh, 2019), where they present a study on online bin packing problems with scenarios that, though, do not rely on probabilistic assumptions or recourse mechanisms. Thus, it constructs a single packing plan that must remain feasible for every possible scenario, therefore filling a gap in BPP literature by considering uncertainty without

recourse actions. Moreover, (Crainic et al., 2021) illustrates a multi-period bin packing model, where different bins are available at different times, meaning that also the relative costs change based on, for example, spot-market capacity. Thus, instead of a single planning period, the authors consider a sequence of interdependent time steps where each period's bin allocation influences the next. Overall, their results confirm that the stochastic bin packing paradigm can effectively support tactical-to-operational logistics decisions under multi-period uncertainty. Another relevant consideration is brought to light in (Salem et al., 2023), which is the first dedicated survey of cutting and packing under uncertainty. In the article, the authors identify numerous open challenges and research opportunities, as pointing out that many realistic sources of variability remain understudied (e.g., last-minute order changes, estimation errors in item dimensions, etc.) and that existing stochastic models often suffer from tractability issues, preventing widespread industrial adoption. Therefore, by highlighting these gaps, the paper aims to guide future research, encouraging the development of more efficient algorithms and the exploration of uncertainty in complex packing settings. All these contributions to the state of the art mark a shift from static optimization toward data-driven, adaptive decision-making, reflecting the growing complexity and dynamism of modern logistics environments.

Additionally, the integration of machine learning into bin packing has opened new directions for research. Indeed, machine learning-based heuristics (Fedorov, 2023) have been proposed to predict high-quality initial solutions or guide search heuristics. Moreover, (Bruni et al., 2023) further exploit the topic of machine learning optimization, which integrates predictive analytics and prescriptive optimization for decision making in last-mile and third-party logistics contexts. The results published show that combining learning and optimization significantly improves both solution quality and computational efficiency, compared to purely stochastic or deterministic methods.

In synthesizing these works, it becomes evident that bin packing with uncertainty is a multidisciplinary topic. The reviewed papers collectively push the frontier of BPP in different directions, as focusing on modeling new variants (uncertainty, mixing, multi-dimensional), or on efficiency (heuristics, analysis), while many do both to some degree. On the whole, this chapter

sets the underlying motivations for the work of this thesis that, with the model proposed in the following section, tries to further align with real cases' peculiarities, managing even more possible forms of uncertainty, combining a two-stage stochastic algorithm based on VCSBPSI with additional constraints on items and bins characteristics, and on item types and categories. This development should broaden the practical applications of the VCSBPSI, making it a flexible option for different industrial scenarios, with the purpose of treating uncertainty from multiple points of view, ranging from stochasticity to attribute management to planning across different time horizons.

4. Problem Description

4.1 Model background

The model this thesis focuses on is the Variable Cost and Size Bin Packing Problem with Stochastic Items (VCSBPSI). This model is an evolution of the general bin packing problem (BPP), which can be defined as follows (Eliiyi & Eliiyi, 2009): the model has several containers or bins of the same size, and *n* items need to be packed in as few containers as possible, hence not including any variability or uncertainty of any kind. The BPP is an NP-hard combinatorial problem (Garey & Johnson, 1990), modeled using binary indicator variables. The mathematical formulation of the problem is the following:

$$\min z = \sum_{j=1}^{n} y_i \tag{1}$$

$$\text{s.t.} \sum_{j=1}^{n} v_i x_{ij} \leq V_j y_j, \quad 1 \leq i \leq m \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad 1 \leq j \leq n \tag{3}$$

$$x_{ij}, y_i \in \{0,1\}, \quad 1 \leq i \leq m, 1 \leq j \leq n \tag{4}$$

where the objective function (1) tries to minimize the total number of bins that are used for packing all items. The variable y_i takes the value of 1 if bin i is used for storage; otherwise, it is zero. Constraint set (2) ensures that the bin capacity is not exceeded for any bin. Variable x_{ij} is 1 if the item i is packed in the bin j, zero otherwise. Constraint set (3) sets that all items should be packed, but only assigned to one bin. Finally, constraint set (4) ensures binary variables. This formulation of the BPP is known as the one-dimensional or basic bin packing problem, and, as described previously in this thesis, it has many applications in supply chains, including container loading, loading trucks with weight capacity, and applications in manufacturing environments.

Beyond the basic model, with time, advances in research have developed new variants of the basic BPP, introducing variability and uncertainty, to improve

the model's alignment with real-world contexts. Therefore, we start by illustrating a variable cost and size bin packing problem (VCSBPP) (Crainic et al., 2011), which introduces variability in terms of characteristics of the bins, with costs and capacity changing for different bins, defined as follows:

$$\min z\,(y) = \sum_{j \in \mathcal{J}} c_j y_j \tag{1}$$

$$\text{s.t.} \sum_{j \in \mathcal{J}} x_{ij} = 1, \quad \forall i \in \mathcal{I} \tag{2}$$

$$\sum_{i \in \mathcal{I}} v_i x_{ij} \leq V_j y_j, \quad \forall j \in \mathcal{J}, \tag{3}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \tag{4}$$

$$y_j \in \{0,1\}, \quad \forall j \in \mathcal{J}. \tag{5}$$

The difference that we can notice from the one-dimensional bin packing problem is the introduction of the variable c_j , which represents the cost of each bin j. The author assumes, without any loss of generality, that the volumes and costs associated with the bins and items are integers, peculiar to each bin, while the rest of the model behaves as the basic BPP. In this version, the objective function (1) doesn't try to minimize the number of bins used, but the total fixed cost of the selected bins. Constraints (2) and (3) act as the model above, while relations (4) and (5) reinforce the integrality requirements for all decision variables.

4.2 Mathematical developed model

Moreover, as already mentioned in the introductory sections of this work, to further develop new structures of the problem and to enhance the applicability of the model to real-world cases, the literature has been focusing especially on the uncertainty of fact realizations in reality. It is for this reason that this thesis presents alterations to the VCBPSI, further developing means to manage uncertainty, as first introduced in the *Variable Cost and Size Bin Packing Problem with Stochastic Items* (Crainic et al., 2014). This work additionally increases the level and difficulty of constraints, approximately

trying to approach reality even more, allowing the model itself to be more flexible in terms of applicability to different scenarios of the supply chain industry, as well as different needs, where through small changes, the entity of uncertainty introduced in this thesis could be altered specifically for a peculiar field and related applications. Following the structure outlined by (Crainic et al., 2014), the problem is formulated as a two-stage stochastic model that accounts for both the availability of information and the uncertainty associated with items in capacity planning. The approach relies on a stochastic programming framework with recourse, which separates the tactical capacity planning decisions of the first stage (Monczka et al., 2020) from the operational decisions of the second stage. The latter consists of recourse actions, undertaken repeatedly over the planning horizon, that define the adjustments to be implemented once the relevant information becomes available.

We define J as a set of available bins in the first stage of the problem, while the set Ω is the sample space of random events, where $\omega \in \Omega$ defines the realization of a particular event. Vector ξ contains all the stochastic parameters defined in the model, meaning that $\xi(\omega)$ is a given realization of the random vector. The first stage binary variables y_j are equal to 1 if bin $j \in J$ is selected, 0 otherwise.
The problem analyzed in this work is the following:

$$\min_{y} \sum_{j \in J} f_j \, y_j + E_{\xi}\left[Q\left(y, \xi(\omega)\right)\right] \tag{1}$$

$$\text{s.t. } y_j \in \{0,1\}, \quad \forall j \in J \tag{2}$$

where $Q\left(y, \xi(\omega)\right)$ is the extra cost paid in the second stage if capacity is added when uncertainty is introduced, considering also the tactical plan y and the vector of different scenarios $\xi(\omega)$. The objective function (1) aims to minimize the fixed cost associated to the tactical plan, meaning the first stage decision of capacity allocation when uncertainty is still not revealed, plus the expected cost associated with the operational second stage, when extra capacity is

added; Constraint (2) simply states integrality requirements on y. The second stage aims to define function $Q(y, \xi(\omega))$ as:

$$Q(y, \xi(\omega)) = \min_{x,z} \sum_{k \in K(\omega)} f_k z_k \tag{3}$$

$$\text{s.t.} \sum_{j \in J} x_{ij} + \sum_{k \in K} x_{ik} = 1, \quad \forall i \in I(\omega) \tag{4}$$

$$\sum_{i \in I(\omega)} v_i(\omega) x_{ij} \leq V_j y_j, \quad \forall j \in J \tag{5}$$

$$\sum_{i \in I(\omega)} v_i(\omega) x_{ik} \leq V_k z_k, \quad \forall k \in K \tag{6}$$

$$\sum_{i \in I(\omega)} w_i(\omega) x_{ij} \leq W_j y_j, \quad \forall j \in J \tag{7}$$

$$\sum_{i \in I(\omega)} w_i(\omega) x_{ik} \leq W_k z_k, \quad \forall k \in K \tag{8}$$

$$\sum_{i \in I(\omega): \lambda(i) = t} x_{ij} \leq R_{jt}, \quad \forall j \in J, \forall t \in T \tag{9}$$

$$\sum_{i \in I(\omega): \lambda(i) = t} x_{ik} \leq N_{kt}, \quad \forall k \in K, \forall t \in T \tag{10}$$

$$\sum_{i \in I(\omega)} l_i(\omega) x_{ij} \leq L_j y_j, \quad \forall j \in J \tag{11}$$

$$\sum_{i \in I(\omega)} l_i(\omega) x_{ik} \leq L_k z_k, \quad \forall k \in K \tag{12}$$

$$\sum_{i' \in I(\omega); \lambda(i') \neq t} x_{i'b} \leq M(1 - x_{ib}), \quad \forall b \in J \cup K, i \in I, t \in T | \lambda(i) = t \tag{13}$$

$$M = card(\{i \in I(\omega) | \lambda(i) \neq t\}) \tag{14}$$

$$x_{ib} \in \{0,1\}, \quad \forall i \in I(\omega), \forall b \in J \cup K \tag{15}$$

$$z_k \in \{0,1\}, \quad \forall k \in K \tag{16}$$

where $\forall j \in J$, we have f_j, V_j, W_j, L_j which are respectively the fixed cost, the volume capacity, the weight capacity and the length of bins j. The same parameters are considered for the set K, which represents the set of available bins in the second stage, where $\forall k \in K$, we have f_k, V_k, W_k, L_k as defined above. $I(\omega)$ is the set of items to be packed in the considered scenario, where v_i, w_i, l_i are respectively the volume, weight and length of item $i \in I(\omega)$. Following these variables, constraints (5)-(8) and (11)-(12) verify that different kind of

capacities of the bins are not exceeded when inserting items. Constraint (4) ensures that an item is packed in only one bin, while decision variables (15) are equal to 1 if an item is packed in bins in either the first or second stage. On the other hand, decision variables (16) behave as constraint (2), guaranteeing that z_k is equal to 1 only if bin $k \in K$ is selected. Then, additional constraints are used to manage item types, which consist of the major and most significant changes applied to this model with respect to (Crainic et al., 2014). In this sense, constraint sets (9)-(10) are used to manage the maximum number of items per type allowed in a bin, governed by parameters R_{jt} and N_{kt}. These act as soft upper bounds that enforce operational segregation, ensuring balanced loading without exceeding thresholds that might compromise safety or quality.

However, following the item's types peculiarities, (13)-(14) introduce even a stronger separation mechanism, adding stronger barriers on type separation by enforcing bin homogeneity with respect to item type. This implies that if an item is assigned to a bin, then that bin can only contain items of the same type. This is deeply guaranteed by a *big-M* constraint, where the right part of the inequality goes to zero if an item of type t is assigned to a bin, therefore deleting the mathematical possibility of selecting other items of different types $\lambda(i) \neq t$. Conceptually, this creates a logical exclusion effect, limiting the bin to a single category type after its first assignment. These additions can be useful in real-case scenarios, as they are adaptable to the different needs of the supply chain, for example, handling obstacles in putting together different items from different suppliers, or in mixing different types of perishable foods, or materials that could cause hazardous chemical reactions. Thus, this contributes to enhancing the flexibility to meet the industry's different requirements. On the other hand, all additional rigid constraints exponentially increase the model's complexity, leading to a foreseeable prediction that the algorithm's overall performance will be less efficient compared to the standards in the literature or previous related works. Indeed, each new dimension (volume, weight, length) and categorical constraint (item type) expands the feasible search space exponentially. The *Big-M* type constraints in particular are known to increase the degeneracy of the integer program, potentially slowing down branch-and-bound procedures used by MIP solvers such as Gurobi or CPLEX.

To summarize, the model represents a multi-dimensional, two-stage stochastic optimization problem in which the first stage involves advance decisions on the number and types of bins to open, while the second stage determines how the items of each scenario are allocated within the available capacity. The model therefore captures the tension between robustness and flexibility inherent in modern supply chains, where companies can strategically reserve baseline capacity (first stage) and retain the option to react dynamically (second stage) if realized demand deviates from forecasts. This stochastic structure thus enables decision-makers to quantify and control the cost of uncertainty, measuring how much extra capacity is worth securing in advance versus acquiring it reactively.

In essence, the proposed algorithm in this thesis extends the classical VCSBPSI model along two key dimensions:

1. Multi-dimensional capacities, integrating volume, weight, and length constraints.
2. Item-type segregation, ensuring logical homogeneity and compliance with industrial requirements.

5. Solution Method

5.1 Solution approaches

The incorporation of uncertainty and additional constraints in bin packing makes these problems challenging to solve optimally. Nevertheless, in literature, a range of methodologies has been developed, from exact algorithms to heuristics and hybrid approaches. For this thesis, an exact optimization method approach will be used.

Exact optimization methods are exact algorithms that guarantee optimal solutions but are typically limited to smaller problem instances due to combinatorial explosion. For one-dimensional bin packing, Integer Linear Programming (ILP) formulation exists – e.g. using binary variables like x_{ij} to indicate if item i goes into bin j, with constraints to enforce capacity and item assignment. However, a straightforward ILP is impractical for large problems, as the number of potential bins and variables can be too massive (Bender et al., 2015). The introduction of uncertainty multiplies problem size: a two-stage stochastic Linear Programming often has copies of second-stage variables for each scenario, thus resulting in an exponential growth of computational complexity even for modest original problems. Therefore, one overarching theme is the tension between optimality and practicality, because as models incorporate uncertainty and realism, exact optimal solutions become less attainable. This is why the academic community has responded with increasingly advanced heuristic and hybrid methods, which also motivated the choice of this work to develop a heuristic algorithm as a benchmark to further evaluate the described model.

5.2 Optimization method

In combinatorial optimization, stochastic programming and robust optimization represent two complementary paradigms for addressing uncertainty (Maggioni et al., 2016). Stochastic programming models uncertain parameters as random variables or probability distributions and seeks solutions optimizing an expected or risk-adjusted objective, often with recourse actions. In contrast, robust optimization, or scenario-based planning, does not assume explicit probability distributions. Instead, it requires

solutions that remain feasible across a predefined set of scenarios. The scenario-based approach captures uncertainty via multiple discrete item sets or sizes; only one scenario will actually occur, but it is unknown beforehand, so the packing must work for any scenario that might occur. This yields a single robust packing that can accommodate every scenario, effectively safeguarding against worst-case or wide-ranging conditions. For this reason, in addressing the VCSBPSI, a two-stage stochastic programming approach is first formulated, where instances are generated from probability distributions, introducing uncertainty directly through the parameters. Secondly, sampling is applied to obtain a set of representative parameter scenarios, which are then used to evaluate the expected cost associated with the second-stage recourse actions. The scenario-based framework mitigates distributional assumptions and potential biases by ensuring feasibility across multiple scenarios.

Thus, the first step towards solving model (1)-(16) is to obtain a manageable approximation of function $E_{\xi}\left[Q\left(y,\xi(\omega)\right)\right]$, through set S, namely the set of representative scenarios aforementioned. These are used to approximate the expected cost associated with the operational decisions:

$$min \quad \sum_{j\in J} f_j y_j + \sum_{s\in S} p_s \left(\sum_{k\in K^s} f_k z_k^s \right) \tag{17}$$

$$s.t. \quad \sum_{j\in J} x_{ij}^s + \sum_{k\in K} x_{ik}^s = 1, \quad \forall i \in I^s, \forall s \in S \tag{18}$$

$$\sum_{i\in I^s} v_i^s x_{ij}^s \leq V_j y_j, \quad \forall j \in J, \forall s \in S \tag{19}$$

$$\sum_{i\in I^s} v_i^s x_{ik}^s \leq V_k z_k^s, \quad \forall k \in K, \forall s \in S \tag{20}$$

$$\sum_{i\in I^s} w_i^s x_{ij}^s \leq W_j y_j, \quad \forall j \in J, \forall s \in S \tag{21}$$

$$\sum_{i \in I^s} w_i^s x_{ik}^s \leq W_k z_k^s, \quad \forall k \in K, \forall s \in S \tag{22}$$

$$\sum_{i \in I^s} l_i^s x_{ij}^s \leq L_j y_j, \quad \forall j \in J, \forall s \in S \tag{23}$$

$$\sum_{i \in I^s} l_i^s x_{ik}^s \leq L_k z_k^s, \quad \forall k \in K, \forall s \in S \tag{24}$$

$$\sum_{i \in I^s : \lambda(i) = t} x_{ij}^s \leq R_{jt}, \quad \forall j \in J, \forall t \in T \tag{25}$$

$$\sum_{i \in I^s : \lambda(i) = t} x_{ik}^s \leq N_{kt}, \quad \forall k \in K, \forall t \in T \tag{26}$$

$$x_{ib}^s \in \{0,1\}, \quad \forall i \in I^s, \forall b \in J \cup K, \forall s \in S \tag{27}$$

$$z_k^s \in \{0,1\}, \quad \forall k \in K, \forall s \in S \tag{28}$$

$$y_j \in \{0,1\}, \quad \forall j \in J \tag{29}$$

$$\sum_{i' \in I^s, \lambda(i') \neq t} x_{i'b}^s \leq M\,(1 - x_{ib}^s), \quad \forall b \in J \cup K,\ i \in I,\ t \in T : \lambda(i) = t \tag{30}$$

$$M = card(\{i \in I^s | \lambda(i) \neq t\}), \tag{31}$$

where in (17) p_s defines the probability associated with scenarios $s \in S$, which follows a uniform distribution. Similarly to the stochastic model, $z_k^s = 1$ if bin $k \in K$ is selected in scenario $s \in S$, 0 otherwise; $x_{ij}^s = 1$ if item $i \in I^s$ is packed in bin $j \in J$ in scenario $s \in S$, 0 otherwise; and $x_{ik}^s = 1$ if item $i \in I^s$ is packed in bin $k \in K$ in scenario $s \in S$, 0 otherwise. The scenario-based approach allows the model to account for all scenarios when computing the optimal solution, ensuring greater flexibility and neutrality, even though it incurs a significant increase in computational complexity from multiple scenarios.

Basically, the two-stage stochastic model takes the important decisions, such as how many bins to open in the first stage, by weighing up the benefits and drawbacks of either allocating more capacity on hand when demand is still unknown or risking having to buy extra capacity at a higher cost. This is done for each instance by considering all the scenarios contained in the sample. That is carried out while still maintaining the overall functionality of an optimization algorithm, thereby finding an optimal solution that satisfies all the constraints and minimizes the cost.

5.3 Model testing

5.3.1 Instances generation

Through instances, this work investigates how diversity affects capacity planning, as well as the implications of accounting for different levels of variability and correlation in the stochastic parameters. Building on existing VCSBPSI instances (Crainic et al., 2014), a new set of 120 test instances was generated. This set incorporates three categories of bins, characterized as follows:

- Volumes: 50, 100, 120
- Weight: $500 * V$, where 500 [Kg/m^3] is the density of wood. The material was chosen purely for testing purposes, and it could be changed depending on different interests
- Length is fixed with a value of 16.5 *(Trasporto Europa, 2023)*
- Fixed cost of first-stage bins f_j is correlated to their volumes computed as $\sqrt{V_j}(1+\delta)$, where δ is uniformly distributed in the range [-0.3, 0.3].
- The number of bins of volume V defined in both stages is defined in both stages as the minimum number of bins of volume V needed to pack all items.
- The cost f_k for extra bins in the second stage is the original one increased by a fixed percentage factor α (5%, 10%, 20%, 40%).

With regard to the item set, the number of items in the second stage was assumed to follow a uniform distribution within the range [25, 50]. These items were classified into the following categories:

- Small (S): volume in the range [5, 10];
- Medium (M): volume in the range [15,25];
- Big (B): volume in the range [20,40].

Each item has its own length computed as $\sqrt[3]{v_i}$, while the weight is again $500 * v_i$. Items are divided into three classes of item spread, where the first one (Sp1) includes a high percentage of small items (S=60%, M=20%, B=20%). The second class (Sp2) includes a high percentage of medium items (S=20%, M=60%, B=20%), while the third class (Sp3) represents a high percentage of big items (S=20%, M=20%, B=60%). For each bin, its allowance for containing a maximum number of item types was computed by dividing the volume of the bin by 10 to compute the maximum number of small items that the bin can fit, by 25 for the medium ones, and by 40 for the big ones. Finally, for the final instances, 25 scenarios were generated, and for each parameter combination, 10 random instances were generated to ensure the robustness of the results.

5.3.2 Experimental setup

In order to test and evaluate the problem, this work has been conducted by using the experimental plan illustrated below.

The scenario-based stochastic problem (17)-(31) is solved by a commercial MIP solver, specifically Gurobi, with a maximum runtime of 30 minutes (1800 seconds). The time limit was set to this threshold to allow tests to be feasible for this thesis, but trials were also run with a time limit of an hour, though reporting almost irrelevant changes in the results. All computations were carried out on a processor Apple M3 (8-core, 4 performance + 4 efficiency), 16 GB RAM, up to 4.06 GHz. Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[arm] - Darwin 24.6.0 24G90) was chosen as the MIP solver (Gurobi Optimization, LLC, 2025), and it was run on Python 3.13.2.

5.3.3 Code Implementation

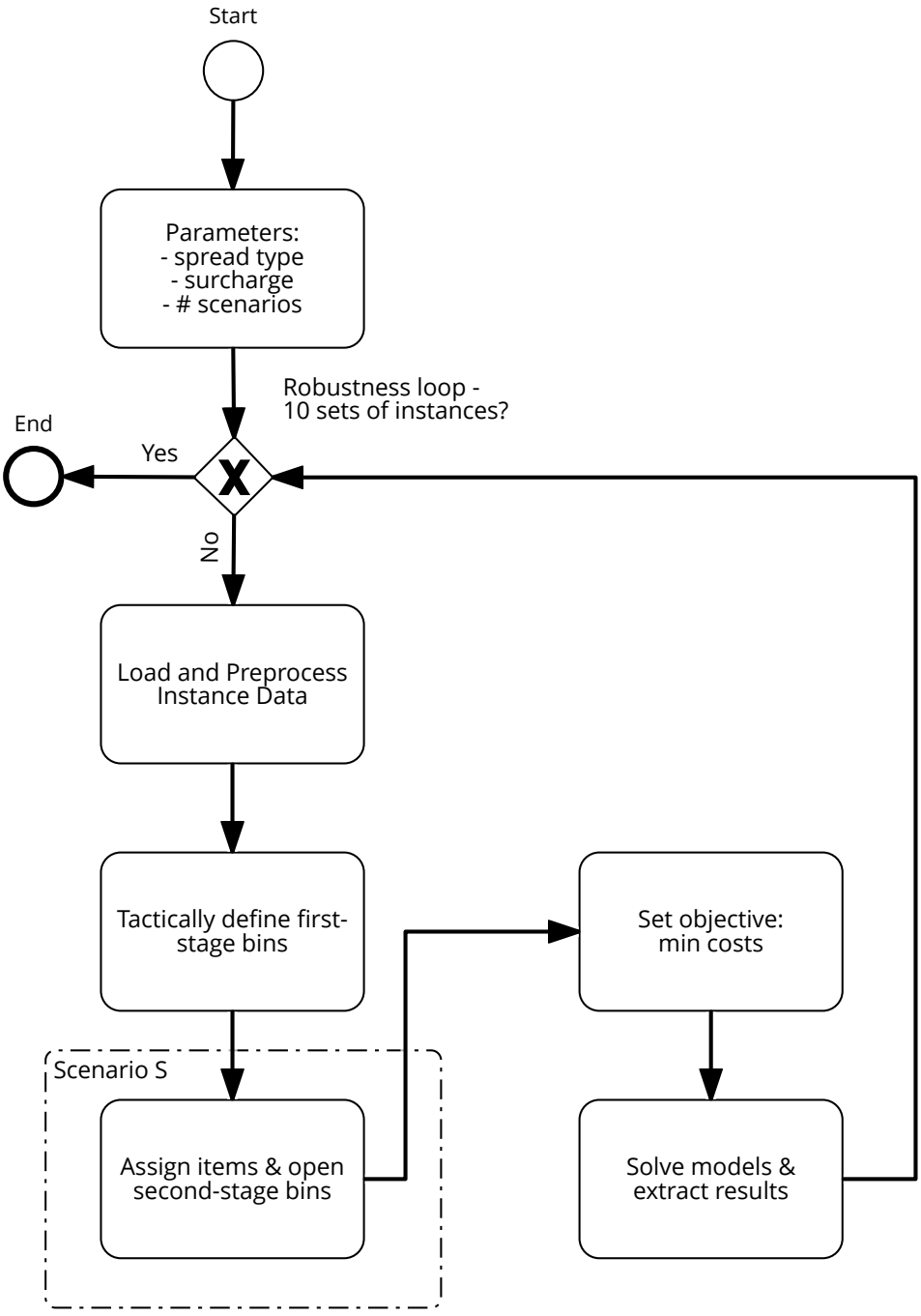

Figure 1 - Flowchart of Python code for 2-Stage Stochastic Model

In *Figure 1*, a flowchart of the structure of the algorithm of the Python code developed is reported. The first task represents the setting for the instances

to be used, with the parameters – items spread type, additional surcharge α, number of scenarios - that are on top of the peculiarities of the instances, as they drive the generation of the instances themselves. The robustness loop represents the cycle built to run the 10 instances generated to ensure no biases in the results due to the eventual peculiarity of one sample instance set. Then, for these 10 times, the algorithm processes an instance with the parameters manually selected and proceeds to load the data from the instance in question, allowing analysis of the scenarios and selection of the bins to allocate in the first stage of the model. Later, once first-stage bins are selected, for each scenario inside the instance, its items are assigned to bins either in first or second stage, depending on the availability of capacity and satisfaction of the various constraints. After the check on each constraint is completed, the minimization objective function is set, and therefore the model is solved, and the results are extracted. Once all 10 instances have been iterated over, the code process finishes with an exit code. The conceptual architecture of the two-stage stochastic model illustrated above is likewise reflected in the extract of the Python code observable below, used for the actual testing of the algorithm, which reports the crucial part of the decision-making model, implemented using the "gurobipy" library. The complete code can be found in *Appendix B*.

```
# start of the optimization model
model = Model("2Stage_Stochastic")
# decision variables creation for bins usage
y = model.addVars(J, vtype=GRB.BINARY, name="y")
z = {s: model.addVars(J, vtype=GRB.BINARY, name=f"z_{s}") for s in S}
x = {s: {} for s in S}
#items categories
T = ['S', 'M', 'B']
divider = {'S': 10, 'M': 25, 'B': 40}
#scenarios S iterations
for s in S:
    items = scenarios[s]
    I = list(range(len(items)))
    #items parameters
    v = {i: items[i]['volume'] for i in I}
    w = {i: items[i]['weight'] for i in I}
    l = {i: items[i]['length'] for i in I}
    lam = {i: items[i]['category'] for i in I}
    #decision variables for items
    x[s] = model.addVars(I, J, vtype=GRB.BINARY, name=f"x_{s}")
    #CONSTRAINTS
    # Each item assigned to only one bin
```

```
    for i in I:
        model.addConstr(quicksum(x[s][i, j] for j in J) == 1)
    # volume, weight and length constraint
    for j in J:
        model.addConstr(quicksum(v[i] * x[s][i, j] for i in I) <= V[j]
* y[j] + V[j] * z[s][j])
        model.addConstr(quicksum(w[i] * x[s][i, j] for i in I) <= W[j]
* y[j] + W[j] * z[s][j])
        model.addConstr(quicksum(l[i] * x[s][i, j] for i in I) <= L[j]
* y[j] + L[j] * z[s][j])
        # Type max constraints

        for t in T:
            max_items = V[j] // divider[t]
            model.addConstr(quicksum(x[s][i, j] for i in I if lam[i] ==
t) <= max_items)

        # Type homogeneity constraint (Big M)
        for t in T:
            for i in I:
                if lam[i] == t:
                    M = sum(1 for i2 in I if lam[i2] != t)
                    model.addConstr(quicksum(x[s][i2, j] for i2 in I if
lam[i2] != t) <= M * (1 - x[s][i, j]))

    # No bin can be used in 2nd stage if used in 1st -> bin can be used
in only 1 stage
    for j in J:
        model.addConstr(z[s][j] <= 1 - y[j])

# Objective: First stage cost + Expected second stage cost
model.setObjective(
    quicksum(F[j]*y[j] for j in J) +
    quicksum(prob_s * (F[j] * (1 +alfa/100))*z[s][j] for s in S for j
in J),
    GRB.MINIMIZE
)
```

Code 1 - Extract from Python code for the two-stage stochastic model

5.3.4 Performance Evaluation

The Expected Value Problem (EVP) is then computed. Hence, the approach consists of replacing the stochastic item set of each scenario with a deterministic approximation, obtained by averaging the counts and volumes of large, medium, and small items across scenarios. This yields a deterministic version of the problem, which is solved to derive the tactical plan, i.e. the first-stage decision. The resulting optimal solution is subsequently evaluated through a Monte Carlo simulation of the scenarios to estimate the true stochastic objective value associated with the chosen first-stage decisions. Additionally, the Percentage Value of the Stochastic Solution ($VSS_{\%}$) is

obtained. The VSS is adopted to quantify the gain a decision-maker achieves by relying on stochastic programming rather than the expected value problem (EVP) approach, and it is computed as

$$VSS_{\%} = \frac{|z_{EVP} - z_{2SP}|}{z_{2SP}} \cdot 100$$

where z_{EVP} and z_{2SP} are respectively the objective function values of the Montecarlo Simulation of the EVP and of the two-stage stochastic problem. Furthermore, as it will be explained with greater precision in the following chapter, additional data is collected to elaborate on the qualification of the two-stage stochastic model, i.e. the percentage of the objective function value given by the first and second stage, the percentage of the volume used in each bin loaded in the first and second stage, and finally the execution time of the algorithm.

5.4 Heuristic algorithm

As stated in the previous sections, exact methods provide vital benchmarks for optimization problems and sometimes components of heuristics (e.g., using an optimal solution of a relaxed problem as a guide). Though for most stochastic or variants of the BPPs, exact optimization beyond small sizes is intractable. Hence, heuristic algorithms, ranging from simple greedy methods to sophisticated metaheuristics, are widely used for large bin packing problems with uncertainty or additional constraints (Tsao et al., 2024). Indeed, many heuristics originally developed for classic bin packing have been adapted to those new variants.

For these reasons and for the purpose of this thesis, a Best Fit Decreasing greedy heuristic has been built. The algorithm orders the items by decreasing volumes and tries to fit them in the bins while respecting compliance with all the constraints between item and bin characteristics. This choice has been made with full awareness that greedy approaches may risk getting locked into a suboptimal arrangement due to the combinatorial nature of conflicts, as for instance, placing a large conflict-heavy item too early might fragment the solution. Although more advanced local search techniques are often required to overcome such limitations, in this work, the heuristic is employed solely for benchmarking purposes. Consequently, a simpler level of complexity has been adopted.

The proposed heuristic is based on a two-stage allocation decision that combines a stochastic tactical pre-selection of first-stage bins with second-stage bin allocation using a *Best-Fit Decreasing (BFD)* search strategy, adapted to account for bin costs and other constraints. The objective of the algorithm is to minimize the total cost of capacity allocation while satisfying feasibility requirements represented by various constraints.

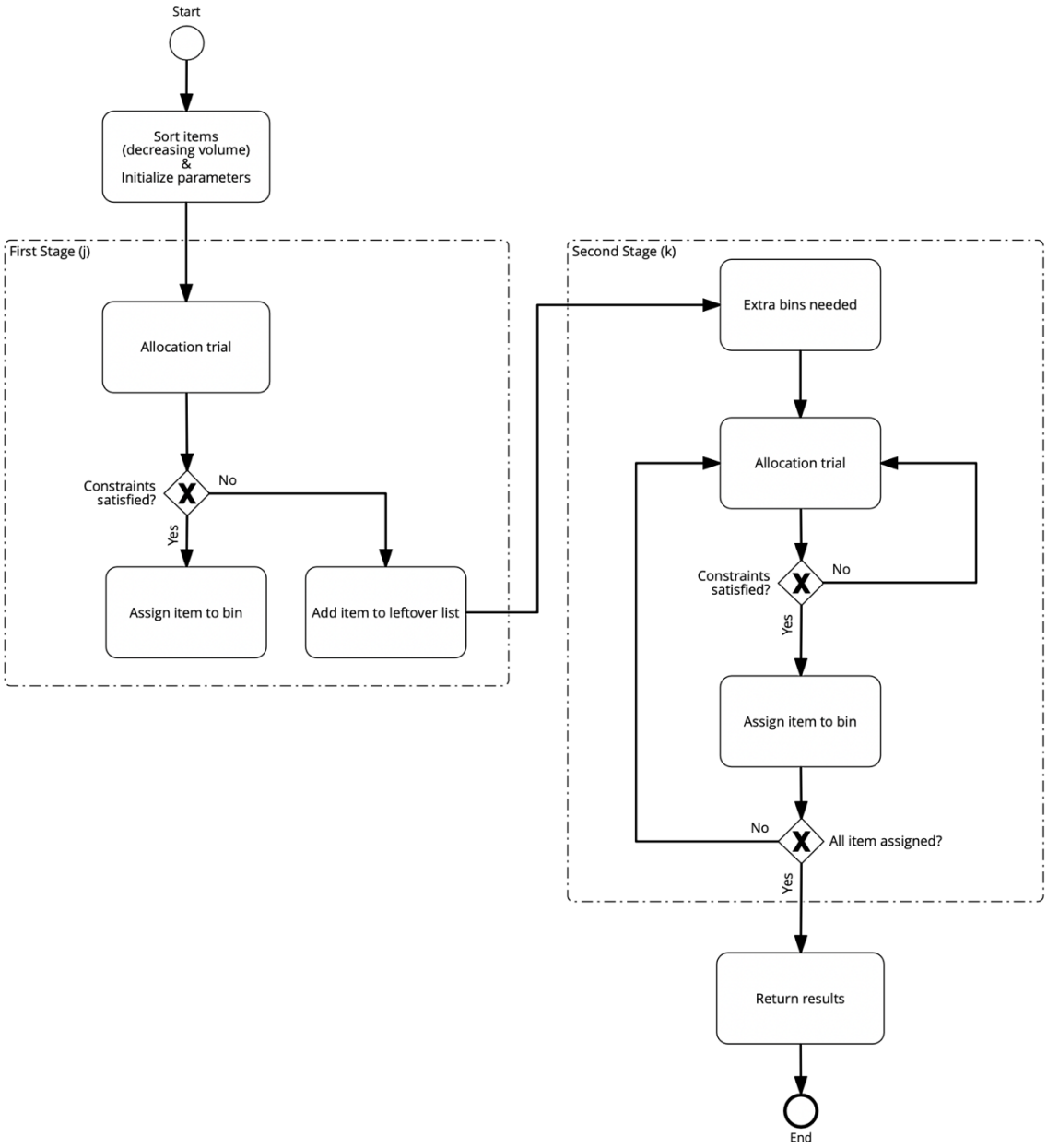

Figure 2 - Flowchart of Python code for heuristic algorithm

The focal decision point of allocating an item into a bin, as raffigurated in the flowchart in *Figure 3*, is represented by iterating through all the bins available and choosing, among those, the one which has the best tradeoff between a low cost and a high remaining volume, so as to attempt to fit other items into the already used bins, in order to dampen the fixed cost of using that bin for whichever number of items allocated inside.

Heuristic Best Fit Decreasing

```
1:  Sort the items in decreasing order of their volumes
2:  Initialize the state of all bins (remaining capacities, type counters, assigned type, list of
        items)
3:  Initialize empty assignment list and leftover list
    First Stage (J)
4:  for each item i in the sorted list of items do
5:    Find the bin j ∈ J with the lowest cost, and among those, the minimum remaining
        volume that can accommodate i, satisfying all the constraints
6:    if such a bin exists
7:      Assign item i to bin j and update bin state
8:    else
9:      Add item i to leftover list
10:   end if
11: end for
    Second Stage (K)
12: if leftover list is not empty
13:   for each item i in the leftover list do
14:     Find the bin k ∈ K with the lowest cost, and among those, the minimum remaining
        volume that can accommodate i, satisfying all the constraints
15:     if such a bin exists
16:       Assign item i to bin k and update bin state
17:     else
18:       Mark item i as unassigned
19:     end if
20:   end for
21: end if
22: Return results
```

Figure 3 - Pseudocode Two-stage Best Fit Decreasing heuristic

As can be seen above in the pseudocode of *Figure 4*, the second stage is opened only when needed, meaning only when some items still have to be allocated to a bin after having filled all the containers chosen for the first stage, simulating demand uncertainty in the case of scarce capacity on hand. Bins in the second stage could also be opened when capacity is still available in first-stage bins, but constraints on item types cannot be satisfied anymore, since all items must be assigned to a bin in any case.

Even though heuristic algorithms are generally more simple-structured than other types of models, such as optimization ones, the content of the presented heuristic is still very elaborate and complex, due to the high level of constraints intrinsic in the nature of the problem. Given that, along with the simple nature of the architecture of the algorithm developed for the heuristic, it is safe to assume that results will not be outstanding, because the structure requested by the model and the constraints would have required its complexity to be much higher and adequate. Though, given the purpose of

this work, where I don’t evaluate the heuristic algorithm on itself, but I use it as a comparison metric to optimization models, a straightforward and intuitive structure was considered acceptable. This topic will be further reflected upon in the results section.

6. Results

6.1 Overall metrics of evaluation

In this section, the results of the execution of the different solution algorithms will be presented. This last part of the thesis aims to explain, through data support, the practical value of adopting a stochastic programming model and to investigate whether a fundamental structure for capacity planning and the dependence of the plan on attributes of the problem setting can be identified. Moreover, the aim is also to further illustrate the complete and general behavior of the model developed and presented, thereby explaining its intrinsic dynamics explicitly.

Before presenting the results from the work carried out for this thesis, I'd like to present a brief introduction on the performance evaluation metrics that will be used to evaluate the algorithm developed.

- **Optimality gap:** or problems where an optimal or best-known solution is available (often from exact methods or small instances), the percentage gap of the algorithm's solution cost from the optimum is a primary metric, as it measures indeed the gap – the distance – between the best bound and the objective function. For this purpose, it is computed as

$$Gap = \frac{z^* - LB}{z^*} \cdot 100$$

  Where z^* is the best integer solution – objective function – and LB is the lower bound.
- **Computation Time:** it represents the time that the algorithm takes to generate a given solution. Since many bin packing decisions need to be made in operational environments, algorithms must be efficient. Though, it is known that exact algorithms have a problem of exponential complexity, therefore increasing the time needed for computations exponentially with respect to small changes in the size of the problem. This is also the reason why often metaheuristics can be tuned to trade off time and solution quality, motivating the choice of developing a simple, basic heuristic for benchmarking the optimization algorithm.

- **Value of Stochastic Solution:** the VSS% quantifies the benefit of using a stochastic model over a simpler deterministic approach. As done by (Crainic et al., 2014), I implicitly measured the VSS by comparing the stochastic solution cost to a plan based on expected deterministic demand, and I expressed it as a percentage improvement. The objective of this metric is to measure whether explicitly accounting for variability yields a notable cost reduction. As expressed above, this value is computed as

$$VSS_{\%} = \frac{|z_{EVP} - z_{2SP}|}{z_{2SP}} \cdot 100$$

  Where z_{EVP} is the expected value of the deterministic problem, while z_{2SP} is the objective function of the two-stage stochastic problem. This metric will be a key decision factor in evaluating the added value of using a stochastic programming model.

- **Utilization and Waste:** especially in bin packing applications, bin utilization, which is the fraction of bin capacity used on average, is an important metric, as high utilization means small wasted space, which is often correlated with cost-efficiency. In this section, utilization will be evaluated through the KPI of filling percentage levels, which are an average of how much bins are filled with items. If bins are only half full (e.g. <50%) perhaps the model is too constrained, implying that they are either too conservative or too constrained. The latter might be the case for the model developed in the thesis, as there are two rigid constraints on the items' compatibility with the bins.

- **Operational metrics:** in applied studies, additional metrics like the number of overflow events, such as how often extra bins beyond the tactical plan were needed, or the distribution of bin usage, in the sense of evaluating whether capacity was adequately planned. On this topic, a plan that frequently requires recourse bins might be considered operationally troublesome. Thus, there is sometimes a balance between cost and reliability, and it is one of the considerations that will have to be kept in mind when analyzing the results for different surcharge factors α. This evaluation will be done with a metric called the percentage of the first stage objective function O_{FS} , which indicates the percentage of the total

value of the objective function that is represented only by the usage of first stage bins.

In the end, the evaluation of these models and algorithms aims to demonstrate that they can handle realistic problem sizes and uncertainty factors, and that they provide a tangible improvement over simpler approaches. As seen at the start of this work, the literature shows a clear progression: early works provided basic models and theoretical results, and more recent works emphasize extensive computational experiments to validate the performances, often analyzing how instance structure and variability levels affect the solutions. This kind of experimentation builds confidence that the algorithms can cope with the diverse conditions encountered in practice and are not tuned to a single specific scenario.

6.2. Presentation of solution results

Table 1 compares the solutions obtained using the two methodologies described in the model testing paragraph. The table reports, for each combination of item spread (Column 1), second-stage bin cost increase parameter α (Column 2), and for a number of scenarios equal to 25, the percentage VSS (Column 3) and the average computational time required by the EVP (Column 4). The computational time of the two-stage stochastic model is not reported in this table because it exceeds the time limit of 30 minutes for each instance.

S=25			
	α (%)	VSS%	Time (s)
sp1	5	15,25	3683,10
	10	15,63	1958,50
	20	0,17	3000,43
	40	2,50	3324,00
sp2	5	1,32	2043,40
	10	5,29	1541,00
	20	0,03	1840,90
	40	10,28	3317,00
sp3	5	2,37	1351,50
	10	10,09	1562,10
	20	2,43	3858,40
	40	8,31	1505,81

Table 1 - Comparisons of the solutions obtained with the scenario-based stochastic problem and the EVP.

The results show that the $VSS_{\%}$ reaches its highest percentage when the parameter takes low values (5% and 10%), which supports the relevance of adopting a two-stage stochastic formulation under such conditions. However, as the cost of second-stage bins increases, the VSS progressively declines, reaching its lowest level at α = 20%. This trend can be explained by the fact that, when second-stage costs are higher, the EVP solution anticipates potential shortages by securing additional capacity in the first stage, thus reducing the benefit of stochastic optimization. On the other hand, we don't always see a low $VSS_{\%}$ value for higher values of α, as for higher numbers of spread of items – meaning sp2 and sp3 -, because since we have fewer small items, bin capacity is taken up faster. This peculiarity makes finding an optimal allocation of the items into the bins more difficult, as bins fill up almost immediately (e.g., bigger bins, with a volume of 120, are full after 3 big items are allocated). At the same time, though, less reasoning on the optimization of the capacity left in the bins used is needed, allowing for the possibility that the two-stage stochastic model may be only slightly more effective than the Monte Carlo simulation. This justifies why, in general, for sp2 and sp3 we see lower values of VSS%, independently of the second-stage bin cost increase

parameter. Though at the same time, we see higher values than expected for VSS% for a higher value of α - for example, equal to 40 -, which might be due to the fact that the results of the Monte Carlo simulation are mostly constrained by the average of the items, especially for the first-stage bins. This impedes the algorithm from reasoning further about an optimal allocation, especially because for these high-spread items, bins fill up quickly, and each new bin used in the second stage incurs a very high-cost increase. This justifies that, contrary to the above, these specific settings validate the advantage of using a two-stage stochastic model, as it can perform more computations for cost minimization by making a decision for the first-stage bins based on the stochastic instance, rather than using an average of item characteristics.

Overall, high values of VSS (e.g., 15,63% for sp1, and α=10%), suggest that when variability is moderate and the surcharge is non-trivial, the stochastic model is able to better anticipate future realizations of item characteristics and allocate resources more effectively. In contrast, very low VSS values (e.g., 0,03% in sp2, α=20%) suggest that the first stage solution derived from the deterministic EVP is already close to optimal for most scenarios. In these cases, uncertainty might have a limited impact on solution quality, or the scenario spread may be narrow enough to make pre-committed decisions relatively robust. Interestingly, at last, high VSS values (e.g., 10,28% in sp2, α=40%) reappear when the spread is wider and the penalty for the second-stage bins increases. This reflects common tensions in logistics planning, as when future uncertainty grows and penalties for reactive decisions increase, proactive planning through a stochastic approach becomes more valuable.

On the other hand, the EVP computational time is generally lower for sp3 because, as stated above, the higher presence of large items allows the model to reach a solution faster, due to fewer possible maneuvers when allocating items to the bins. In fact, for sp1, which has a bigger presence of small items, the execution time reaches the time limit for each level of parameter α.

Furthermore, the performance of the solution methods can be analyzed by going into the details of the results of the two-stage stochastic model and the

EVP, where *Table 2* and *Table 4* display the optimality gap reached by the MIP solver (Column 3), the percentage of the objective function due to the first stage (Column 4), the average filling levels of bins in the first stage (Column 5), and in the second stage (Column 6). The values are grouped by classes of item spreads (Column 1) and the additional cost factor of bins in the second stage (Column 2). The values reported in the tables are the average of the 10 instances used for each combination of parameters, computed separately for each instance.

S = 25					
2 - Stage Stochastic					
	α (%)	Gap (%)	O_fs (%)	F_fs (%)	F_ss (%)
sp1	5	34,68	71,36	63,50	56,04
	10	49,84	58,00	52,97	54,28
	20	40,67	74,54	59,70	49,81
	40	40,10	89,11	56,04	26,88
sp2	5	42,77	72,12	61,18	50,94
	10	35,04	76,68	65,12	49,05
	20	41,48	89,88	58,53	29,46
	40	37,93	89,55	59,36	26,86
sp3	5	26,42	76,17	66,55	52,33
	10	36,92	80,74	63,20	46,97
	20	38,89	85,20	59,55	35,14
	40	31,90	87,75	60,38	29,63

Table 2 - Summary of main experimental results for the two-stage stochastic model

The gaps of the scenario-based stochastic problem are very large (>25%). In general, high gap values mean that the solver wasn't able to find an optimal solution, but only an approximated feasible one. Indeed, as the lower bound is built by relaxing the integer bounds, in big and complex problems, it doesn't always accurately represent the true optimality. In this work, this might be due to two main explanations. Firstly, the high gap value could be related to the difficulty of the MIP solver in computing a good lower bound for large-sized integer problems with a lot of symmetry in the solutions, which is unfortunately typical of packing problems. Indeed, in bin packing problems,

there are usually many equivalent solutions; for example, if two identical items can be placed in two identical bins, the total cost is not affected, yet the solver will explore all the feasible possibilities regardless. Thus, the presence of many symmetric configurations slows down the construction of the lower bound, because the solver gets lost in branches that lead to the same logical solutions but with different items. Secondly, high gap values might also be due to the rigid constraints that characterize the algorithm developed in this thesis, especially those regarding item compatibility within the same bin. It is for this reason that a detailed focus on the algorithm's behavior when some of the constraints on item types are removed will be presented later. Hence, a high gap is not necessarily an indicator of a low-quality solution, but a technical limitation of the solver in having to deal with large, symmetric problems, as well as highly complex and constrained ones. This aligns with the intuition that stochastic models than encode more variables and constraints, inflate model size and hamper Gurobi's ability to explore the entire solution space efficiently. This brings obvious limitations for real-world adoption, given the usual extensive starting size. Additionally, in contexts where explainability and predictability of solver behavior are crucial, the deterministic EVP approach may still be preferred, despite being theoretically inferior.

Turning to the other columns, labeled O_fs(%), F_fs(%), and F_ss(%), they are displayed to help assess how well the solution found utilizes first-stage and second-stage bins, and to analyze how these metrics change across different item spreads and additional cost α.

In the stochastic model, the percentage of how much the objective cost is borne by first-stage bins (O_fs(%)), drops for lower values of α, especially for spread of items with less big items. This indicates that when the second stage brings with itself a higher additional cost, the algorithm correctly becomes more conservative, utilizing the fixed cost of the first-stage bins more effectively, resulting in a higher value of O_fs(%). This additionally implies that the model actively defers more packing decisions to the second stage, exploiting flexibility despite the surcharge. On the other hand, when items' spread contains more bigger items, the percentage of the cost deriving from the first stage is generally higher across all

levels of α, as in general, bigger bins to fit bigger items already cost more, once again leading to a more conservative and efficient usage of bins in the first stage. Moreover, filling levels confirm this, because when O_fs(%) is higher, filling levels for the second stage are lower, while the ones for the first stage are higher.

Overall, though, the filling levels percentages are slightly low. This outcome depends on the high number of constraints and on the limits on item compatibility, which already limit the full usage of the bins' capacity. For example, a bin with a medium volume from the instances used can only fit 2 big items, possibly leaving unusable space, which increases the overall cost of the solution as well as decreases the quality of the filling levels. To further sustain this argumentation, a peculiar focus on the item compatibility constraints has been reported in *Table 3*, where both a solution with no constraints of this type, and one with only no constraints of having to fit the same item type inside the same bin, have been represented. Basically, in the first row of results, both the constraints on the maximum number of item type packable in a bin, and bin homogeneity on item category have been removed, while in the second row, only the latter constraint is removed.

Constraints Modification	VSS%	Time (s)	EVP				Stochastic - 2 stage			
			Gap (%)	O_fs (%)	F_fs (%)	F_ss (%)	Gap (%)	O_fs (%)	F_fs (%)	F_ss (%)
No Compatibility Constraints	43,79	0,09	0,20	99,80	72,55	1,05	5,12	81,56	90,46	53,46
No Same Type Constraints	32,81	0,13	0,00	100,00	71,36	-	12,89	80,92	85,28	50,87
All Compatibility Constraints	2,50	3324,00	10,71	88,50	62,07	27,02	40,10	89,11	56,04	26,88

Table 3 - Results with modifications on item compatibility constraints of the EVP and two-stage stochastic model [sp1, α=40]

We can firstly notice that both the computational time for the EVP and the gap, both of the EVP and of the two-stage stochastic model, are significantly better when modifications on items' constraints are introduced, with a substantial absolute distance from the results with all the constraints to the modified ones. This consideration already builds upon the argumentations written above, acknowledging the complexity provided to the algorithm by the rigid constraints on items' compatibility, thereby worsening the expected results.

Furthermore, through these results, it is safe to state that these items' constraints are also the main contributors to having lower filling levels. Lastly, it is noticeable that for the EVP, the percentage of the objective function generated by the first stage is higher when modifications are made to the compatibility constraints, meaning that the deterministic initial values are a good approximation of the solution when the bins' capacity is not constrained excessively. On the other hand, for the two-stage stochastic model, values of O_fs(%) decrease, once again indicating that the latter defers more packing decisions to the second stage, being able to manage the items' allocation more efficiently, especially when the complexity is lower. This is also reflected in the VSS% values, which are extremely high when the complexity is lower, thanks to the absence of rigid constraints.
Moving on to the results of the EVP, a similar depiction of *Table 2* is represented below.

S = 25					
EVP					
	α (%)	Gap (%)	O_fs (%)	F_fs (%)	F_ss (%)
sp1	5	9,40	90,43	61,91	25,58
	10	4,07	92,46	63,09	25,68
	20	8,83	89,70	62,40	27,91
	40	10,71	88,50	62,07	27,02
sp2	5	19,46	93,01	67,71	28,57
	10	2,04	92,29	69,37	29,95
	20	3,94	91,27	68,42	32,18
	40	11,38	87,57	67,07	27,98
sp3	5	2,60	91,56	65,93	29,62
	10	3,41	91,73	65,39	28,61
	20	12,71	86,15	65,28	31,11
	40	1,55	89,15	64,84	30,01

Table 4 - Summary of main experimental results for the EVP.

As opposed to the results reported above for the two-stage stochastic model, we see different trends across the metrics displayed in *Table 4*. First of all, we overall see lower values for the gap (<20%), thanks to the lower complexity of choosing bins in the first stage, since the decision is made through deterministic data. Then, the percentage level of objective cost borne by the first decision stage are generally higher than the ones in the two-stage stochastic model, regardless of

the values of α. As it can be seen in the table, the values of O_fs(%) is always higher than 86%, indicating that the EVP algorithm in the Monte Carlo simulation leaves less room of operation to the second decisional stage, as even for low values of α the percentage value is very high, meaning that the algorithm doesn't efficiently take advantage of the fact that extra bins opened in the second stage only have an additional percentage cost of 5%. On the other hand, the high values of O_fs(%) could also be explained by the fact that the deterministic instance used to compute the first stage bins of the EVP, which are then used in a Montecarlo simulation to eventually open new bins in the second stage, is probably a good representation of the data in the stochastic instances, leading to a rare need of opening new bins in the second stage, as most items already fit in the fixed bins. This could also be motivated by the relative small size and distribution ranges of the stochastic instances, logically making the deterministic averaging an accurate approximation.

Lastly, before deciding to test the model with instances containing 25 scenarios, fewer scenarios were adopted for the initial experimental phase. Results from these trials are reported in *Table 5*. Once it was seen that when using 10 scenarios the results were not substantially different from the ones using 25 scenarios, it was declared that, since the results were stable, 25 scenarios were a good enough representation. More scenarios were not tested for two main reasons. Firstly, in (Crainic et al., 2014), where the VCSBPSI was tested on similar instances, 25 scenarios were reported to be a good approximation of representation when compared to 50 and 100 scenarios. Secondly, because of the complex nature of the model developed in this research work, which already causes gap values and computational times to be elevated, testing more than 25 scenarios would have only added complexity to the model, providing misleading results, biased by the elevated computational time and weakly formulated lower bound.

S	VSS%	Time (s)	EVP				2-Stage Stochastic			
			Gap(%)	O_fs (%)	F_fs (%)	F_ss (%)	Gap (%)	O_fs (%)	F_fs (%)	F_ss (%)
S = 3	22,72	374,94	0,82	96,90	59,86	24,58	44,58	70,43	64,62	60,65
S = 10	15,17	1786,13	2,69	93,53	60,84	22,48	34,57	71,25	62,95	55,33
S = 25	15,25	3683,10	9,40	90,43	61,91	25,58	34,68	71,36	63,50	56,04

Table 5 - Results of the EVP and two-stage stochastic model for changing number of scenarios [sp1, α=5]

Given the findings stated in this subchapter, from an operational standpoint, they suggest that the stochastic model better exploits capacity in reaction to item uncertainty. This can be especially useful in logistics systems and related applied aspects, where unused bin capacity is a key cost driver. At the same time, some level of dependence of the plan on attributes of the problem setting can be identified, as reported in the multiple tables above.

6.3 Heuristic Benchmark

As stated previously, a simple greedy heuristic algorithm was created to benchmark the results of the latter with the objectives produced by the two-stage stochastic optimization model. The main goal is to consider whether using an heuristic algorithm might be of help, if not even a better option, when treating large size data or multiple rigid constraints, which we know know that they make an optimization algorithm very intractable in the case of considering in how much time an optimal solution is generated, or how big is the gap between the objective function and the lower bound.

To understand if the greedy heuristic model could be a valuable substitute of the two-stage stochastic model, the research work mainly focused on a metric which can be found in the table below as "Gap cost". It is computed as the difference between the total cost of the two-stage stochastic algorithm – 2SS – and the one of the heuristic, divided by the two-stage model value of the objective function, which represents the base level for this benchmarking procedure. For this reason, higher values of the gap cost mean that the heuristic algorithm doesn't appropriately minimize the cost as much as the original model. Obviously, this statement will always be true, as heuristics are only good enough models, but often a small difference in the cost is acceptable, as it can be part of a trade-off. The more general trade-off considered, when deciding whether to use a heuristic or metaheuristic rather than an optimization exact algorithm, is often the one between cost and time. Indeed, heuristic algorithms, due to their simpler and more sequential structure, usually have very brief computational times, contrary to the normal runtime exponential behavior of exact models.

Before analyzing the specific case of this heuristic comparison, it is important to point out that the execution time of the 2-SS algorithm is always above 1800 seconds, which is also the time limit manually set for the code, while the one for the heuristic algorithm was always below 1 second in the testing run. This sets a starting point favourable to the heuristic, which operates far better on this topic. Secondly, a margin for improvement in the structure of the heuristic should be taken into account, especially for future comparisons, because, as already highlighted in the dedicated section, the heuristic algorithm built has an extremely simple architecture. A metaheuristic could also be further considered, as it might overall decrease the cost projected through the use of more advanced mechanisms in the decision-making steps of the algorithm.

sp1

Alpha	Method	2SS	Heuristic	Gap cost
5	*Total cost*	75,85	92,40	**22%**
	F_fs (%)	63,50	62,92	-
	F_ss (%)	56,04	59,24	-
10	*Total cost*	101,25	108,98	**8%**
	F_fs (%)	52,97	59,93	-
	F_ss (%)	54,28	61,85	-
20	*Total cost*	88,43	106,71	**21%**
	F_fs (%)	59,70	63,56	-
	F_ss (%)	49,81	55,86	-
40	*Total cost*	87,18	116,38	**33%**
	F_fs (%)	56,04	63,94	-
	F_ss (%)	26,88	67,98	-

Table 6 - Comparison of SS vs Heuristic Method under varying α values

In *Table 6* presented above, we see some values for the spread of items sp1, hence the one with the highest percentage of small items. In detail, we see the total cost and the percentage filling levels, split by type of algorithm – 2SS or Heuristic -, and by different values of surcharge α. Additionally, in the last column, the *Gap cost* can be observed. We see that for the highest value of surcharge – 40% - we have the highest value of gap cost. This might be

due to the fact that the 2SS algorithm can choose more smartly which bins to plan in the first stage, and therefore, by being more conservative for a higher surcharge, it results in a relatively lower cost, compared to the heuristic. This affirmation doesn't hold for a value of α= 10%, which we would expect to be higher than the gap cost of α = 5%, but is instead much lower. This types of inconsistency can be caused by the simplistic form of the heuristic, which, even if it selects more poorly the allocation between first and second stage bins, it might select bins that have overall little costs, such as for example the one with lower capacity in volume. These are the main behaviors that the heuristic algorithm adopts during its procedure, justifying macro trends noticeable in the results. On the other hand, its simplistic nature allows for exceptions in these trends, due also to peculiarities of the instances' sets.

sp2

Alpha	Method	2SS	Heuristic	Gap cost
5	*Total cost*	105,04	119,36	**14%**
	F_fs (%)	61,18	67,55	-
	F_ss (%)	50,94	59,50	-
10	*Total cost*	101,89	127,42	**25%**
	F_fs (%)	65,12	68,64	-
	F_ss (%)	49,05	53,99	-
20	*Total cost*	108,44	141,65	**31%**
	F_fs (%)	58,53	69,31	-
	F_ss (%)	29,46	61,58	-
40	*Total cost*	102,81	133,58	**30%**
	F_fs (%)	59,36	68,45	-
	F_ss (%)	26,86	66,20	-

Table 7 - Comparison of SS vs Heuristic Method under varying α values

For spread type 2, meaning an instance set with the highest percentage of medium-sized items, we see similar trends to the general expected one described previously. For growing values of the additional costs of extra bins, the gap cost increases. In this specific case, it is clear that also for a small value of surcharge, the 2SS works overall far better than the heuristic algorithm, since we still have a gap of 14% in the total cost of the solution for

a 5% incremental increase in the cost of selecting extra bins in the second stage of the problem.

sp3

Alpha	Method	2SS	Heuristic	Gap cost
5	*Total cost*	124,52	152,48	**22%**
	F_fs (%)	66,55	66,37	-
	F_ss (%)	52,33	56,37	-
10	*Total cost*	115,62	145,16	**26%**
	F_fs (%)	63,20	68,33	-
	F_ss (%)	46,97	50,24	-
20	*Total cost*	132,48	172,78	**30%**
	F_fs (%)	59,55	68,44	-
	F_ss (%)	35,14	55,54	-
40	*Total cost*	142,90	158,07	**11%**
	F_fs (%)	60,38	65,95	-
	F_ss (%)	29,63	52,68	-

Table 8 - Comparison of SS vs Heuristic Method under varying α values

Lastly, for sets of instances with spread type number 3, which have the highest percentage of large-sized items, we can notice the already known trend vertically across growing values of additional costs, with the exception of the value corresponding to α=40%. This can be possible due to the fact that both models choose conservatively and smartly enough the bins to open during the first stage – the tactical one – and, when extra bins were needed, the heuristic algorithm was able to effectively choose adequate and cheap bins rather than expensive ones, correctly balancing the trade off between the cost of the bin and the remaining capacity volume for next items. These are the advantages of using a best-fit decreasing greedy heuristic rather than a first-fit decreasing one.

Alpha	Gap cost
5	19%
10	19%
20	27%
40	25%

Table 9 - Gap cost average for different α values

To sum up, in *Table 9*, the average values of gap cost for all spread types, split by values of surcharge α are reported. It is important to highlight that, when no distinction about spread types is made, we can see with no exceptions that the trend of the results grows with positive correlation to the surcharge. More in detail, we can see a big distance between relatively low values of surcharge – 19% - and relatively higher ones – 25, 27% -, probably suggesting that 20% can be considered a sort of threshold for a more conservative behavior of the two-stage stochastic algorithm, not followed by the heuristic.

This last consideration allows us to better understand the behavior of the heuristic model, which, when it has to deal with more expensive additional costs, fails to be an eventual substitution of the 2SS model, having a too high gap cost.

As a whole, the illustrated overall results naturally lead to a statement that sustains that the heuristic algorithm developed in this thesis can't be considered a valid alternative to the two-stage stochastic model, because, even though the computational time is incomparably better, the incremental percentage of the cost of the solution is unsuitable, resulting in a weak performance. On the other hand, though, some logistics companies may opt for this kind of model to solve bin packing problems involving large numbers of items and bins, where the additional cost of the heuristic might become negligible.

7. Conclusion and Future Work

7.1 Conclusions

This thesis investigated the problem of bin packing under uncertainty, introducing a two-stage stochastic model tailored to an extension of the Variable Cost and Size Bin Packing Problem with Stochastic Items (VCSBPSI)**.** The formulation captures realistic complexities, such as heterogeneous bin dimensions, variable costs, and uncertain item characteristics (volume, weight, length, category), which are revealed only after initial decisions are made. The problem was addressed using a two-stage recourse approach, in which bin selections are made in the first stage, followed by item assignments and recourse bin activations in the second stage after scenario realizations. Key metrics, including the objective function value, Value of the Stochastic Solution (VSS)**,** weighted-average fill levels, and first-stage cost share, were systematically analyzed.

The results indicate that the stochastic approach consistently outperforms the deterministic equivalent in terms of cost efficiency and space utilization. This outcome is mainly due to the nature of the stochastic method compared to the latter, because it defers more packing decisions to the second stage, being able to manage the items' allocation more efficiently, especially when the additional cost of using extra bins is remarkable - both lower or higher - or when complexity is lower. The two-stage stochastic model can also efficiently allocate resources across increasing surcharge levels.

On the other hand, the heuristic benchmarking showed that the main algorithm consistently performed better in terms of the objective cost, even though the gain in computational time was not comparable.

Therefore, the two-stage stochastic model is overall the best alternative, with exceptions that might be due to explainability, where the EVP might be better, or to faster execution time, especially for larger data sets, where a heuristic algorithm might represent a better choice.

7.2 Field contribution

The thesis contributes to the scientific literature in several ways. Firstly, it extends the classical bin packing problem to account for uncertainty, heterogeneity in bin costs, and multidimensional item constraints simultaneously. This generalization captures real-world logistics challenges with greater fidelity than previous models.

Secondly, it demonstrates the operational value of stochastic decision-making through comprehensive simulations. The consistently positive $VSS_{\%}$ values highlight the quantitative advantage of anticipating uncertainty rather than relying on expected-value-based deterministic models. Moreover, by reporting not only cost metrics but also structural indicators (e.g., fill levels and bin usage rates), the analysis provides insights into the distribution of decision weight across stages, such as the shift in cost burden from the first to the second stage. For instance, the observed increase in the first-stage cost share (from approximately 58% to 88% across scenarios) indicates that the problem strategically postpones certain decisions to benefit from realized information, thereby reducing overall inefficiency.

7.3 Research Limitations

While the work of this thesis has advanced understanding of stochastic and scenario-based approaches to bin packing problems, several methodological and practical limitations should be acknowledged, as recognizing these constraints is essential for contextualizing the results and identifying promising avenues for further investigation.

First, computational complexity represents a significant challenge, as in several cases the solver failed to close the optimality gap within the time limit, particularly when dealing with large numbers of scenarios and highly symmetric feasible spaces. This limitation could also be associated with the use of personal items for this work, which are relatively weaker than professional processors, which usually have larger hardware and faster clock

speeds. Furthermore, because the focus was on assessing the practical value of adopting a stochastic programming model by computing the VSS% rather than achieving higher-quality results at the expense of longer computational time, the work did not increase the time limit for the Python testing.

Second, the scenario generation process relies on predefined distributions and uniform probabilities, which, while useful for controlled analysis, may not fully reflect the skewed or correlated patterns present in operational settings. Moreover, the model assumes independence among item characteristics, which may not fully capture real-world correlations.

Third, the current model assumes static, single-period planning, which may oversimplify decision-making environments where demand unfolds dynamically or where capacity acquisition and decommissioning have lead times and financial implications.

7.4 Future work

Future work could address several of the limitations outlined above. From a methodological standpoint, heuristic or metaheuristic strategies, more elaborated and adopting local optimization strategies, may also be explored to provide high-quality solutions within tighter computational budgets.

From a modeling perspective, extending the framework to a multi-period setting—where capacity and demand evolve over time—would enhance its applicability to long-term planning problems in logistics and supply chain management. Additionally, future studies should test the stochastic programming model using real-world industrial data or within collaborative projects with logistics firms, in order to obtain more realistic results and improve the robustness and relevance of the model's decision-support capabilities, enabling its transferability to large-scale industrial settings.

Operationally, embedding additional constraints related to lead times, bin reusability, and the cost of underutilized capacity could render the model

suitable for exploring cross-domain applications such as e-commerce warehousing, postal logistics, or third-party fulfillment centers.

References

Aissaoui N., Haouari M., Hassini E. «Supplier Selection and Order Lot Sizing Modeling: A Review». *Computers & Operations Research* 34: 3516–40 (2007). https://doi.org/10.1016/j.cor.2006.01.016.

Anand S., Guericke S. «A Bin Packing Problem with Mixing Constraints for Containerizing Items for Logistics Service Providers». *Computational Logistics* vol. 12433. Lecture Notes in Computer Science. Springer International Publishing, (2020). https://doi.org/10.1007/978-3-030-59747-4_22.

Baldi M. M., Crainic T. G., Perboli G., Tadei R. «The Generalized Bin Packing Problem». *Transportation Research* (2012). https://doi.org/10.1016/j.tre.2012.06.005.

Baldi M. M., Manerba D., Perboli G., Tadei R. «A Generalized Bin Packing Problem for Parcel Delivery in Last-Mile Logistics». *European Journal of Operational Research* (2019). https://doi.org/10.1016/j.ejor.2018.10.056.

Bender M., Thielen C., Westphal S. «Packing Items into Several Bins Facilitates Approximating the Separable Assignment Problem». *Information Processing Letters* (2015). https://doi.org/10.1016/j.ipl.2015.02.001.

Birge J. R., Louveaux F. «Introduction to Stochastic Programming». *Springer Verlag* (1999).

Bódis A., Balogh J. «Bin Packing Problem with Scenarios». *Central European Journal of Operations Research* (2019) 27: 377–95. https://doi.org/10.1007/s10100-018-0574-3.

Bruni M. E., Fadda E., Fedorov S., Perboli G. «A Machine Learning Optimization Approach for Last-Mile Delivery and Third-Party Logistics». *Computers & Operations Research* 157 (2023). https://doi.org/10.1016/j.cor.2023.106262.

Capua R., Frota Y., Ochi L.S., Vidal T. «A Study on Exponential-Size Neighborhoods for the Bin Packing Problem with Conflicts». *Journal of Heuristics* (2018) 24: 667–95. https://doi.org/10.1007/s10732-018-9372-2.

Christensen H.I., Khan A., Pokutta S., Tetali P. «Approximation and Online Algorithms for Multidimensional Bin Packing: A Survey». *Computer*

Science Review (2017) 24: 63–79. https://doi.org/10.1016/j.cosrev.2016.12.001.

Cohen M. C., Keller P.W., Mirrokni V., Zadimoghadddam M. «Overcommitment in Cloud Services Bin Packing with Chance Constraints». *ACM SIGMETRICS Performance Evaluation Review* 45, issue 1 (2017): 7–7. https://doi.org/10.1145/3143314.3078530.

Crainic T. G., Fomeni F. D., Rei W. «Multi-Period Bin Packing Model and Effective Constructive Heuristics for Corridor-Based Logistics Capacity Planning». *Computers & Operations Research* 132 (2021): 105308. https://doi.org/10.1016/j.cor.2021.105308.

Crainic, T. G., Gobbato L., Perboli G., Rei W., Watson J., Woodruff D. L. «Bin Packing Problems with Uncertainty on Item Characteristics: An Application to Capacity Planning in Logistics». *Procedia - Social and Behavioral Sciences*, vol. 111 (2014): 654–62. https://doi.org/10.1016/j.sbspro.2014.01.099.

Crainic T. G., Perboli G., Rei W., Rosano M., Lerma V. «Capacity Planning with Uncertainty on Contract Fulfillment». *European Journal of Operational Research* 314, issue 1 (2024): 152–75. https://doi.org/10.1016/j.ejor.2023.09.003.

Crainic T. G., Perboli G., Rei W., Tadei R.. «Efficient lower bounds and heuristics for the variable cost and size bin packing problem». *Computers & Operations Research* 38, issue 11 (2011): 1474–82. https://doi.org/10.1016/j.cor.2011.01.001.

Dawande M., Kalagnanam J., Sethuraman J. «Variable Sized Bin Packing with Color Constraints». *Electronic Notes in Discrete Mathematics 7,* (2011): 154-57. https://doi.org/10.1016/S1571-0653(04)00248-3

Ekici A. «Variable-Sized Bin Packing Problem with Conflicts and Item Fragmentation». *Computers & Industrial Engineering 163* (2022): 107844. https://doi.org/10.1016/j.cie.2021.107844

Eliiyi U. & Eliiyi D. T. «Applications of bin packing models through the supply chain». *International journal of business and management - 1,* issue 1 (2009).

Fedorov S. «Urban Logistics and Last Mile Applications: Models and Methods to Deal with Demand Uncertainty». *PhD dissertation in Computer and Control Engineering, Politecnico di Torino* (2023).

Garey M. R., Johnson D. S. «Computers and Intractability; A Guide to the Theory of NP-Completeness». *W. H. Freeman & Co.* (1990).

Salem K. H., Silva E., Oliveira J. F. «Cutting and Packing Problems under Uncertainty: Literature Review and Classification Framework». *International Transactions in Operational Research* 30, issue 6 (2023): 3329–60. https://doi.org/10.1111/itor.13318.

Li L., Lin X., Negenborn R. R., De Schutter B. «Pricing Intermodal Freight Transport Services: A Cost-Plus-Pricing Strategy». *Computational Logistics* vol. 9335. Lecture Notes in Computer Science. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-24264-4_37.

Liang K., Shan M., Liu H., Yang J., Gu C., Yin X. «Intelligent Optimization of E-Commerce Order Packing Using Deep Reinforcement Learning with Heuristic Strategies». *Applied Soft Computing* 179 (2025): 113283. https://doi.org/10.1016/j.asoc.2025.113283.

Maggioni F., Potra F., Bertocchi M. «A scenario-based framework for supply planning under uncertainty: stochastic programming versus robust optimization approaches». arXiv:1611.06514 (2016). https://doi.org/10.48550/arXiv.1611.06514.

Martello S. & Toth P. «Knapsack problems – Algorithms and Computer Implementation». *John Wiley & Sons,* (1990).

Martinovic J., Hähnel M., Scheithauer G., Dargie W. «An Introduction to Stochastic Bin Packing-Based Server Consolidation with Conflicts». *TOP* 30, issue 2 (2022): 296–331. https://doi.org/10.1007/s11750-021-00613-1.

Monczka R.M., Handfield R. B., Giunipero L. C., Patterson J. L. «Purchasing & Supply Chain Management». *7th edition. Cengage Learning*, (2020).

Perboli G., Gobbato L., Perfetti F. «Packing Problems in Transportation and Supply Chain: New Problems and Trends». *Procedia - Social and Behavioral Sciences* 111 (2014): 672–81. https://doi.org/10.1016/j.sbspro.2014.01.101.

Tadei R. & Della Croce F. «Elementi di ricerca operativa». *Società Editrice Esculapio,* (2019).

Tawfik C. & Limbourg S. «Pricing Problems in Intermodal Freight Transport: Research Overview and Prospects». *Sustainability* 10, issue 9 (2018): 3341. https://doi.org/10.3390/su10093341.

Trasporto Europa. «Motorizzazione chiarisce sulla lunghezza degli autoarticolati». (2023). https://www.trasportoeuropa.it/notizie/autotrasporto/motorizzazione-chiarisce-sulla-lunghezza-degli-autoarticolati/.

Tsao, Y., Tai J., Vu T., Chen T.. «Multiple Bin-Size Bin Packing Problem Considering Incompatible Product Categories». *Expert Systems with Applications* 247 (2024): 123340. https://doi.org/10.1016/j.eswa.2024.123340.

Yan J., Lu Y., Chen L., Quin S. «Solving the Batch Stochastic Bin Packing Problem in Cloud: A Chance-Constrained Optimization Approach». *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, (2022), 2169–79. https://doi.org/10.1145/3534678.3539334.

Appendix

In this section, a link to a public GitHub repository is reported, which contains a *Readme* file that contains the explanation of what each code developed for this thesis does, and what it produces. Additionally, an Excel file can also be found in this folder, which reports all the results obtained from the execution of all the algorithms, namely the EVP with the Monte Carlo simulation, the two-stage stochastic model, and the heuristic BFD. For completion these 3 main codes will also be fully reported in the text below.

Repository GitHub:
https://github.com/LudovicaM/Optimization_extended_VCSBPSI.git

Appendix A – EVP model with Monte Carlo Simulation

```
from gurobipy import *
import time
import json
import random

# Montecarlo simulation after having computed determinstically the
fixed bins available in the first stage.
# The scope is to analyze how the model reacts once uncertainty is
introduced through stochastic items.

#extract bins from the determinstic instance. From these I will
manually impose the results of the
# deterministic model as the fixed bins available in the first stage
[J], while the remaining bins
# will be available in the second stage [K] with the surcharge alfa
with open('instances_det_json/instance_Sp1_s38.json', 'r') as f:  #da
cambiare manualmente quando cambia il set deterministico
    data_det = json.load(f)

# Principal parameters
spread_type = data_det['spread_type']

# Bins' extraction from file
bins = data_det['bins']
expanded_bins = [] #full list of bins without the quantity, but as many
bins as required
counter = 0
for b in bins:
    bin_copy = b.copy()
    quantity = bin_copy.pop('quantity')  # rimuovo 'quantity'
    for _ in range(quantity):
        new_bin = bin_copy.copy()
        new_bin['id'] = counter  # aggiungo identificativo univoco
        expanded_bins.append(new_bin)
        counter += 1
```

```
fixed_bins_index = [7, 11, 13, 14, 15, 16, 17, 21] #MANUALLY IMPOSED
bins from result of deterministic 1°-stage model
fixed_bins = [] # will create set J
for bin in expanded_bins:
    if bin['id'] in fixed_bins_index:
        fixed_bins.append(bin)

extra_bins = [] #bins not fixed in the first stage -> will create set K
for b in expanded_bins:
    if b['id'] not in fixed_bins_index:
        extra_bins.append(b)

print("__FIXED BINS (J)__")
for b in fixed_bins:
    print(f"Bin id {b['id']}: Volume: {b['volume']}, Cost: {b['cost']},
"
          f"Length: {b['length']}, Weight: {b['weight']}")


print("__EXTRA BINS (K)__")
for c in extra_bins:
    print(f"Bin id {c['id']}:Volume: {c['volume']}, Cost: {c['cost']},
"
          f"Length: {c['length']}, Weight: {c['weight']}")

# manual parameters to be set for each combination of parameters -> to
be changed for different iterations
spread = 'Sp1'
surcharge = 5  #5,10,20,40
n_scen = 10

# extraction of stochastic instances to be executed 10 times for
robustness
for rep in range(5):
    filename =
f'instances_json/instance_{spread}_s{surcharge}_n{n_scen}_r{rep}.json'
    print(filename)
    with open(filename, 'r') as f:
        data = json.load(f)

        alfa = data['surcharge']
        scenarios = data['scenarios']
        S = list(scenarios.keys())
        prob_s = 1 / len(S)
        """
        for s_id, item_list in scenarios.items():
            print(f"\nScenario {s_id}:")
            for i, item in enumerate(item_list):
                print(f"Item {i}: Volume={item['volume']},
Category={item['category']},"
                      f" Length={item['length']},
Weight={item['weight']}")
        """
        # choose one scenario with a uniform distribution
        random_scenario_id = random.choice(list(scenarios.keys()))
        print(f"Running model for scenario {random_scenario_id}")
        items = scenarios[random_scenario_id]
        #print(items)
```

```
        #Parameters' sets of one scenario
        J = fixed_bins_index.copy() #first stage bins
        K = []                      #second stage bins
        for c in extra_bins:
            extra_bins_id = c['id']
            K.append(extra_bins_id)
    # Parameters of bins
    Vj = {b['id']: b['volume'] for b in fixed_bins}
    Lj = {b['id']: b['length'] for b in fixed_bins}
    Wj = {b['id']: b['weight'] for b in fixed_bins}
    fj = {b['id']: b['cost'] for b in fixed_bins}
    Vk = {b['id']: b['volume'] for b in extra_bins}
    Lk = {b['id']: b['length'] for b in extra_bins}
    Wk = {b['id']: b['weight'] for b in extra_bins}
    fk = {b['id']: (b['cost']* (1+ alfa/100)) for b in extra_bins}

    model = Model("VCSBPSI_Montecarlo")

    # Decision Variables
    #x = model.addVars(I, J + K, vtype=GRB.BINARY, name="x")
    y = {}
    for j in J:
        y[j] = model.addVar(vtype=GRB.BINARY, name=f"y_{j}")
        model.addConstr(y[j] == 1)  # logic constraint that forces y[j]
= 1
    #z = model.addVars(K, vtype=GRB.BINARY, name="z")
    x = {}
    z = {}
    for s in S:
        items = scenarios[s]
        I = list(range(len(items)))
        v = {i: items[i]['volume'] for i in I}
        w = {i: items[i]['weight'] for i in I}
        l = {i: items[i]['length'] for i in I}
        lam = {i: items[i]['category'] for i in I}
        T = ['S', 'M', 'B']
        R_jt = {(j, t): Vj[j] // {'S': 10, 'M': 25, 'B': 40}[t] for j
in J for t in T}
        N_kt = {(k, t): Vk[k] // {'S': 10, 'M': 25, 'B': 40}[t] for k
in K for t in T}

        x[s] = model.addVars(I, J + K, vtype=GRB.BINARY, name=f"x_{s}")
        z[s] = model.addVars(K, vtype=GRB.BINARY, name=f"z_{s}")


        # Constraints
        # (4) Each item assigned to one bin
        for i in I:
            model.addConstr(quicksum(x[s][i, b] for b in J+K) == 1)

        # (5) Volume constraint in first-stage bins
        for j in J:
            model.addConstr(quicksum(v[i]*x[s][i, j] for i in I) <=
Vj[j]*y[j])

        # (6) Volume in second-stage
        for k in K:
            model.addConstr(quicksum(v[i]*x[s][i, k] for i in I) <=
Vk[k]*z[s][k])

        # (7)-(8) Weight constraints
```

```
        for j in J:
            model.addConstr(quicksum(w[i]*x[s][i, j] for i in I) <=
Wj[j]*y[j])
        for k in K:
            model.addConstr(quicksum(w[i]*x[s][i, k] for i in I) <=
Wk[k]*z[s][k])

        # (9)-(10) Max items per type
        for t in T:
            for j in J:
                model.addConstr(quicksum(x[s][i, j] for i in I if
lam[i] == t) <= R_jt[j, t])
            for k in K:
                model.addConstr(quicksum(x[s][i, k] for i in I if
lam[i] == t) <= N_kt[k, t])


        # (11)-(12) Length constraints
        for j in J:
            model.addConstr(quicksum(l[i]*x[s][i, j] for i in I) <=
Lj[j]*y[j])
        for k in K:
            model.addConstr(quicksum(l[i]*x[s][i, k] for i in I) <=
Lk[k]*z[s][k])

        # (15)-(16) Type homogeneity (big-M)
        for b in J + K:
            for t in T:
                for i in I:
                    if lam[i] == t:
                        M = sum(1 for i2 in I if lam[i2] != t)
                        model.addConstr(
                            quicksum(x[s][i2, b] for i2 in I if lam[i2]
!= t) <= M * (1 - x[s][i, b])
                        )


    # Objective - minimization
    model.setObjective(
        quicksum(fj[j] * y[j] for j in J) +
        quicksum(prob_s * fk[k] * z[s][k] for s in S for k in K),
        GRB.MINIMIZE
    )


    # Time limit [can be removed if not necessary]
    model.setParam("OutputFlag", 1)
    model.setParam("MIPGap", 0.01)
    model.setParam("TimeLimit", 1800)
    model.setParam("Threads", 4)
    start_time = time.time()
    # Optimize
    model.optimize()

    end_time = time.time()
    # computation of execution time (could be useful for comparison
with heuristics)
    execution_time = end_time - start_time

    # time limit reached -> approximated solution
    if model.Status == GRB.TIME_LIMIT:
```

```
        print("Tempo limite raggiunto, soluzione approssimata.")
    # infeasibility case
    elif model.status == GRB.INFEASIBLE:
        print("Model is infeasible. Computing IIS...")
        model.computeIIS()
        model.write("model.ilp")  # esporta il modello
        model.write("infeasible.ilp")  # esporta il sistema infeasible
    else:
        print("\n✅ OPTIMAL SOLUTION FOUND")
    if model.Status == GRB.OPTIMAL or model.Status == GRB.TIME_LIMIT:
        total_cost = model.ObjVal
        print(f"Objective function: {total_cost}")

        # First stage bin cost
        cost_fs = sum(fj[j] for j in J if y[j].X == 1)
        percentage_fs = (cost_fs / total_cost) * 100
        print(f"First-stage cost: {round(cost_fs, 3)}
({round(percentage_fs, 2)}%)")

        # Aggregated stats for 1st and 2nd stage
        prob_s = 1 / len(S)  # uniform probability
        total_fill_1st = 0.0
        count_1st = 0
        total_z = 0.0  # weighted total number of second-stage bins
used
        total_fill_2nd = 0.0
        count_2nd = 0

        for s in S:
            items = scenarios[s]
            I = list(range(len(items)))
            v = {i: items[i]['volume'] for i in I}

            fill_sum_1st = 0
            fill_count_1st = 0

            # First-stage bins fill
            for j in J:
                volume_used = 0
                if y[j].X == 1:  # bin j is used
                    print(f"Bin {j} used; Original volume {Vj[j]}")
                    items_in_bin = [i for i in I if x[s][i, j].X == 1]
                    for vv in items_in_bin:
                        volume_used += v[vv]
                    print(f"  Items: {items_in_bin}; Filling level of
bin: {((volume_used / Vj[j]) * 100)}")
                    if items_in_bin:
                        fill = sum(v[i] for i in items_in_bin) / Vj[j]
* 100
                        fill_sum_1st += fill
                        fill_count_1st += 1

            # somma la media dei fill di questo scenario pesata per la
sua probabilità
            if fill_count_1st > 0:
                total_fill_1st += prob_s * (fill_sum_1st /
fill_count_1st)
                count_1st += 1

            # Second-stage bins usage and fill
            z_used = 0
```

```
            fill_sum = 0
            fill_count = 0
            for k in K:
                if z[s][k].X == 1:
                    z_used += 1
                    print(f"Bin {k} used; Original volume {Vk[k]}")
                    items_in_bin = [i for i in I if x[s][i, k].X > 0.5]
                    fill = sum(v[i] for i in items_in_bin) / Vk[k] *
100
                    fill_sum += fill
                    fill_count += 1

            total_z += prob_s * z_used
            if fill_count > 0:
                total_fill_2nd += prob_s * (fill_sum / fill_count)
                count_2nd += 1

        avg_fill_1st = total_fill_1st if count_1st > 0 else 0
        print(f"Weighted avg. fill level (1st stage bins):
{round(avg_fill_1st, 2)}%")
        avg_fill_2nd = total_fill_2nd if count_2nd > 0 else 0
        print(f"Weighted avg. second-stage bins used: {round(total_z,
2)}")
        print(f"Weighted avg. fill level (2nd stage bins):
{round(avg_fill_2nd, 2)}%")
    else:
        print("No optimal solution found.")
    print("Resolution time: {}\n".format(execution_time))
```

Appendix B – Two-stage stochastic optimization model

```
from gurobipy import *
import time
import json

# this code runs the 2-stage stochastic optimization model for VCSBPSI
with additional constraints, using Gurobi.
# The model extracts data from an instance with the following
combinations of parameters
# (spread type, alfa surcharge, number of scenarios) for 10 times, in
order to grant robustness in the results.
# The instances are generated previously in another python project
"Instances_generation".

# combination of parameters to be changed manually for the right
istance
spread = 'Sp1'  # spread type
surcharge = 5  # alfa 5, 10, 20, 40
n_scen = 10  # number of scenarios S

# extraction of stochastic instances through .json reading function
for rep in range(5):
    filename =
f'instances_json/instance_{spread}_s{surcharge}_n{n_scen}_r{rep}.json'
    print(filename)
```

```
    with open(filename, 'r') as f:
        data = json.load(f)

        # Extraction of bins from opened instance
        bins_raw = data['bins']
        print("__BINS__")
        for b in bins_raw:
            print(f"Volume: {b['volume']}, Cost: {b['cost']}, "
                  f"Length: {b['length']}, Weight: {b['weight']},
Quantity: {b['quantity']}")
        alfa = data['surcharge']
        scenarios = data['scenarios']
        S = list(scenarios.keys())
        prob_s = 1/len(S)

        bins = []  # list which will contain the full number of bins.
This will be used from now on
        counter = 0
        for b in bins_raw:
            bin_copy = b.copy()
            quantity = bin_copy.pop('quantity')  # remove field
'quantity'
            for _ in range(quantity):
                new_bin = bin_copy.copy()
                new_bin['id'] = counter  # add unique id to be able to
track the bin
                bins.append(new_bin)
                counter += 1

        #create set J which contains all bins
        J = [b['id'] for b in bins]

    # Bin params
    V = {b['id']: b['volume'] for b in bins}
    L = {b['id']: b['length'] for b in bins}
    W = {b['id']: b['weight'] for b in bins}
    F = {b['id']: b['cost'] for b in bins}

    # start of the optimization model
    model = Model("2Stage_Stochastic")
    # decision variables creation for bins usage
    y = model.addVars(J, vtype=GRB.BINARY, name="y")
    z = {s: model.addVars(J, vtype=GRB.BINARY, name=f"z_{s}") for s in
S}
    x = {s: {} for s in S}
    #items categories
    T = ['S', 'M', 'B']
    divider = {'S': 10, 'M': 25, 'B': 40}
    #scenarios S iterations
    for s in S:
        items = scenarios[s]
        I = list(range(len(items)))
        #items parameters
        v = {i: items[i]['volume'] for i in I}
        w = {i: items[i]['weight'] for i in I}
        l = {i: items[i]['length'] for i in I}
        lam = {i: items[i]['category'] for i in I}
        #decision variables for items
        x[s] = model.addVars(I, J, vtype=GRB.BINARY, name=f"x_{s}")
        #CONSTRAINTS
        # Each item assigned to only one bin
```

```
        for i in I:
            model.addConstr(quicksum(x[s][i, j] for j in J) == 1)
        # volume, weight and length constraint
        for j in J:
            model.addConstr(quicksum(v[i] * x[s][i, j] for i in I) <=
V[j] * y[j] + V[j] * z[s][j])
            model.addConstr(quicksum(w[i] * x[s][i, j] for i in I) <=
W[j] * y[j] + W[j] * z[s][j])
            model.addConstr(quicksum(l[i] * x[s][i, j] for i in I) <=
L[j] * y[j] + L[j] * z[s][j])
            # Type max constraints

            for t in T:
                max_items = V[j] // divider[t]
                model.addConstr(quicksum(x[s][i, j] for i in I if
lam[i] == t) <= max_items)

            # Type homogeneity constraint (Big M)
            for t in T:
                for i in I:
                    if lam[i] == t:
                        M = sum(1 for i2 in I if lam[i2] != t)
                        model.addConstr(quicksum(x[s][i2, j] for i2 in
I if lam[i2] != t) <= M * (1 - x[s][i, j]))

        # No bin can be used in 2nd stage if used in 1st -> bin can be
used in only 1 stage
        for j in J:
            model.addConstr(z[s][j] <= 1 - y[j])

    # Objective: First stage cost + Expected second stage cost
    model.setObjective(
        quicksum(F[j]*y[j] for j in J) +
        quicksum(prob_s * (F[j] * (1 +alfa/100))*z[s][j] for s in S for
j in J),
        GRB.MINIMIZE
    )

    model.setParam("OutputFlag", 1)
    model.setParam("MIPGap", 0.01)
    model.setParam("TimeLimit", 1800) # time limit 30 minutes
    model.setParam("Threads", 8)
    start_time = time.time()
    model.optimize()
    print(f"Gurobi runtime (reported): {model.Runtime:.2f} sec")

    end_time = time.time()
    # computation of execution time (could be useful for comparison
with heuristics)
    execution_time = end_time - start_time
    # time limit reached -> approximated solution
    if model.Status == GRB.TIME_LIMIT:
        print("TIME LIMIT REACHED. APPROXIMATED SOLUTION")
    # infeasibility case
    elif model.status == GRB.INFEASIBLE:
        print("Model is infeasible. Computing IIS...")
        model.computeIIS()
        model.write("model.ilp")  # esporta il modello
        model.write("infeasible.ilp")  # esporta il sistema infeasible
    else:
        print("\nOPTIMAL SOLUTION FOUND")
```

```
    if model.Status == GRB.OPTIMAL or model.Status == GRB.TIME_LIMIT:
        total_cost = model.ObjVal
        print(f"Objective function: {total_cost}")

        # First stage bin cost
        cost_fs = sum(F[j] for j in J if y[j].X == 1)
        percentage_fs = (cost_fs / total_cost) * 100
        print(f"First-stage cost: {round(cost_fs, 3)}
({round(percentage_fs, 2)}%)")

        fill_1st, fill_2nd, total_j, total_z = 0.0, 0.0, 0.0, 0.0
        for s in S:
            items = scenarios[s]
            I = list(range(len(items)))
            v = {i: items[i]['volume'] for i in I}

            fill_j, count_j = 0, 0
            fill_k, count_k = 0, 0
            j_used = 0
            z_used = 0

            for j in J:
                if y[j].X > 0.5:
                    print(f"Fixed Bin {j} used; Original volume
{V[j]}")
                    items_in_j = [i for i in I if x[s][i, j].X > 0.5]
                    j_used += 1
                    if items_in_j:
                        print(f" Items: {items_in_j}")
                        usage = sum(v[i] for i in items_in_j) / V[j]
                        fill_j += usage * 100
                        count_j += 1
                    else:
                        print(" Items: none (bin open but unused in
this scenario)")
                elif z[s][j].X > 0.5:
                    items_in_j = [i for i in I if x[s][i, j].X > 0.5]
                    print(f"Extra Bin {j} used; Original volume
{V[j]}")
                    print(f" Items: {items_in_j}")
                    z_used += 1
                    usage = sum(v[i] for i in items_in_j) / V[j]
                    fill_k += usage * 100
                    count_k += 1

            if count_j > 0: fill_1st += prob_s * (fill_j / count_j)
            total_j += prob_s * j_used
            if count_k > 0: fill_2nd += prob_s * (fill_k / count_k)
            total_z += prob_s * z_used

        print(f"Weighted avg. first-stage bins used: {round(total_j,
2)}")
        print(f"Weighted avg. fill level (1st stage bins):
{round(fill_1st, 2)}%")
        print(f"Weighted avg. second-stage bins used: {round(total_z,
2)}")
        print(f"Weighted avg. fill level (2nd stage bins):
{round(fill_2nd, 2)}%")
    else:
        print("No optimal solution found.")
```

```
        if model.status == GRB.INFEASIBLE:
            print("Computing IIS...")
            model.computeIIS()
            model.write("infeasible.ilp")

    print(f"RESOLUTION TIME: {round(end_time - start_time, 2)} seconds
\n\n\n\n")
```

Appendix C – Heuristic BFD algorithm

```
import time
import random
import json
import numpy as np

def bfd_two_stage_simple_with_costs(items, bins_first, bins_second,
Rjt, Nkt, alfa):
    """
    Two-stage heuristic (Best-Fit Decreasing + cost logic)
    """

    # 1) Order items & bins
    items_sorted = sorted(items, key=lambda x: -x['volume'])
    bins_first_sorted  = sorted(bins_first,  key=lambda b: b['cost'])
    bins_second_sorted = sorted(bins_second, key=lambda b: b['cost'])

    # 2) Initialize bin states
    def init_state(bins_sorted):
        return {
            b['id']: {
                'remaining_volume': b['volume'],
                'remaining_weight': b['weight'],
                'remaining_length': b['length'],
                'type_count': {},
                'items': [],
                'assigned_type': None,
                'cost': b['cost'],
            } for b in bins_sorted
        }

    state_J = init_state(bins_first_sorted)
    state_K = init_state(bins_second_sorted)

    # 3) Adaptive placement (Best-Fit + Cost)
    def try_place_adaptive(item, bins_sorted, state, type_cons,
alpha=0.8):
        best_bin, best_key = None, None
        for b in bins_sorted:
            bid, s = b['id'], state[b['id']]
            if (item['volume'] > s['remaining_volume'] or
                item['weight'] > s['remaining_weight'] or
                item['length'] > s['remaining_length']):
                continue
            t = item['category']
            if s['assigned_type'] is not None and s['assigned_type'] !=
t:
                continue
            if (bid, t) in type_cons and s['type_count'].get(t, 0) >=
```

```
type_cons[(bid, t)]:
                continue

            cost = b['cost']
            total_vol = s.get('total_volume', s['remaining_volume'] +
item['volume'])
            fill_gain = item['volume'] / max(total_vol, 1e-9)
            score = alpha * cost + (1 - alpha) * (1 - fill_gain)
            if best_key is None or score < best_key:
                best_key, best_bin = score, bid

        if best_bin is not None:
            s = state[best_bin]
            s['remaining_volume'] -= item['volume']
            s['remaining_weight'] -= item['weight']
            s['remaining_length'] -= item['length']
            t = item['category']
            s['type_count'][t] = s['type_count'].get(t, 0) + 1
            if s['assigned_type'] is None: s['assigned_type'] = t
            s['items'].append(item['id'])
        return best_bin

    # 4) FIRST STAGE (assign items to J-bins)
    assignments, leftovers = {}, []
    for item in items_sorted:
        bid = try_place_adaptive(item, bins_first_sorted, state_J, Rjt,
alpha=0.8)
        if bid is not None:
            assignments[item['id']] = bid
        else:
            leftovers.append(item)

    # 5) SECOND STAGE (assign leftovers to K-bins)
    if leftovers:
        for item in leftovers:
            bid = try_place_adaptive(item, bins_second_sorted, state_K,
Nkt, alpha=0.8)
            assignments[item['id']] = bid

    # 6) Cost computation
    used_bins_J = [bid for bid, s in state_J.items() if s['items']]
    used_bins_K = [bid for bid, s in state_K.items() if s['items']]

    #cost_map_J = {b['id']: b['cost'] for b in bins_first_sorted}
    cost_map_K = {b['id']: b['cost'] * (1 + alfa/100) for b in
bins_second_sorted}  # surcharge applied

    cost_J = sum(b['cost'] for b in bins_first_sorted)
    cost_K = sum(cost_map_K[bid] for bid in used_bins_K)
    total_cost = cost_J + cost_K
    cost_share_J_pct = 100.0 * cost_J / total_cost if total_cost > 0
else 0.0

    cost_report = {
        'used_bins_J': used_bins_J,
        'used_bins_K': used_bins_K,
        'cost_J': cost_J,
        'cost_K': cost_K,
        'total_cost': total_cost,
        'cost_share_J_pct': cost_share_J_pct
    }
```

```
    vol_map_J = {b['id']: b['volume'] for b in bins_first_sorted}
    vol_map_K = {b['id']: b['volume'] for b in bins_second_sorted}

    # 7) Fill levels
    def build_fill_report(state, vol_map):
        rep = []
        for bid, s in state.items():
            V_tot = vol_map[bid]
            V_used = V_tot - s['remaining_volume']
            fill_pct = (100.0 * V_used / V_tot) if V_tot > 0 else 0.0
            rep.append({
                'bid': bid,
                'fill_pct': fill_pct,
                'volume used': V_used,
                'volume total': V_tot,
            })
        return rep

    fill_report = {
        'J': build_fill_report(state_J, vol_map_J),
        'K': build_fill_report(state_K, vol_map_K),
    }

    return assignments, state_J, state_K, cost_report, fill_report


spread = 'Sp3'
surcharge = 40
n_scen = 25

for rep in range(10):
    filename =
f'instances_json/instance_{spread}_s{surcharge}_n{n_scen}_r{rep}.json'
    print(f"\n=== RUN {rep} | FILE {filename} ===")
    with open(filename, 'r') as f:
        data = json.load(f)

    # Extraction of bins from opened instance
    bins = data['bins']
    scenarios = data['scenarios']
    alfa = data['surcharge']

    # list which will contain the full number of bins. This will be
used from now on
    expanded_bins = []
    counter = 0
    for b in bins:
        bin_copy = b.copy()
        quantity = bin_copy.pop('quantity')  # remove field 'quantity'
        for _ in range(quantity):
            new_bin = bin_copy.copy()
            new_bin['id'] = counter  # add unique id to be able to
track the bin
            expanded_bins.append(new_bin)
            counter += 1

    # Step 1 — compute total volume per scenario
    demands = [sum(item['volume'] for item in scenarios[s]) for s in
scenarios]
    mean_demand = np.mean(demands)
```

```
    q80_demand = np.quantile(demands, 0.8)  # coverage at 80%

    # Step 2 — determine how much capacity to open in first stage
    total_bin_capacity = sum(b['volume'] for b in expanded_bins)
    coverage_ratio = q80_demand / total_bin_capacity
    coverage_ratio = min(1.0, coverage_ratio)

    n_total = len(expanded_bins)
    n_j = max(1, int(coverage_ratio * n_total))  # dynamic 1st stage
size

    shuffled_bins = expanded_bins.copy()
    random.shuffle(shuffled_bins)
    fixed_bins = shuffled_bins[:n_j]
    extra_bins = shuffled_bins[n_j:]

    # Step 3 — choose a random scenario to test heuristic performance
    random_scenario_id = random.choice(list(scenarios.keys()))
    items = scenarios[random_scenario_id]
    for i, item in enumerate(items):
        item["id"] = i
        item.pop("number_items", None)

    # Step 4 — build type constraints
    T = ['S', 'M', 'B']
    Vj = {b['id']: b['volume'] for b in fixed_bins}
    Vk = {b['id']: b['volume'] for b in extra_bins}
    R_jt, N_kt = {}, {}
    dividers = {'S': 10, 'M': 25, 'B': 40}
    for bin_id, volume in Vj.items():
        for type, divider in dividers.items():
            R_jt[(bin_id, type)] = volume // divider
    for bin_id, volume in Vk.items():
        for type, divider in dividers.items():
            N_kt[(bin_id, type)] = volume // divider

    # Step 5 — run heuristic
    start = time.time()
    assignments, state_J, state_K, costs, fill_levels =
bfd_two_stage_simple_with_costs(
        items, fixed_bins, extra_bins, R_jt, N_kt, alfa)
    end = time.time()
    execution_time = end - start

    # Print results
    print("\nAssignments to bins J")
    for bid, s in state_J.items():
        if s['items']:  # solo bin usati
            print(f"Bin {bid}: {s['items']}")
    print("\nAssignments to bins K")
    for bid, s in state_K.items():
        if s['items']:
            print(f"Bin {bid}: {s['items']}")
    print("\nBin J usati:", costs['used_bins_J'], "Costo J:",
costs['cost_J'])
    print("Bin K usati:", costs['used_bins_K'], "Costo K:",
costs['cost_K'])
    print("Costo totale:", costs['total_cost'])
    print("Costo percentuale primo stage:", costs['cost_share_J_pct'])
    print("\nFill levels bins J:")
    counter = 0
```

```
    level = 0
    for f in fill_levels['J']:
        print(f"Bin {f['bid']}: Filling percentage: {f['fill_pct']}")
        level += f['fill_pct']
        counter += 1
    print(f"Average fill level bins J: {level / counter:.2f}%")
    print("\nFill levels bins K:")
    counter = 0
    level = 0
    if costs['used_bins_K']:
        for bins in costs['used_bins_K']:
            for f in fill_levels['K']:
                if f['bid'] == bins:
                    print(f"Bin {f['bid']}: Filling percentage:
{f['fill_pct']}")
                    level += f['fill_pct']
                    counter += 1
                    break
        print(f"Average fill level bins K: {level / counter:.2f}%")
    else:
        print("No bins of set K was used")
    print("\nExecution TIME:", execution_time)
```