Production and Operations Management

A Machine Learning Approach to Solve the E-commerce Box-Sizing Problem

Shanthan Kandula
Information Systems, XLRI-Xavier School of Management Jamshedpur, India, shanthank@xlri.ac.in

Debjit Roy (*Corresponding Author)
Operations & Decision Sciences, Indian Institute of Management Ahmedabad, India, debjit@iima.ac.in

Kerem Akartunalı
Department of Management Science, University of Strathclyde, Glasgow, UK, kerem.akartunali@strath.ac.uk

Abstract: E-commerce packages are notorious for their inefficient usage of space. More than one-quarter volume of a typical e-commerce package comprises air and filler material. The inefficient usage of space significantly reduces the transportation and distribution capacity increasing the operational costs. Therefore, designing an optimal set of packaging box sizes is crucial for improving efficiency. We present the first learning-based framework to determine the optimal packaging box sizes. In particular, we propose a three-stage optimization framework that combines unsupervised learning, reinforcement learning, and tree search to design box sizes. The package optimization problem is formulated into a sequential decision-making task called the box-sizing game. A neural network agent is then designed to play the game and learn heuristic rules to solve the problem. In addition, a tree-search operator is developed to improve the performance of the learned networks. When benchmarked with company-based optimization formulation and two alternate optimization models, we find that our ML-based approach can effectively solve large-scale problems within a stipulated time. We evaluated our model on real-world datasets supplied by a large e-commerce platform. The framework is currently adopted by a large e-commerce company across its 28 fulfillment centers, which is estimated to save the company about 7.1 million USD annually. In addition, it is estimated that paper consumption will be reduced by 2080 metric tons and greenhouse gas emissions by 1960 metric tons annually. The presented optimization framework serves as a decision support tool for designing packaging boxes at large e-commerce warehouses.

Key words: machine learning, reinforcement learning, prescriptive analytics, e-commerce, packaging design
History: Received: November 2023; Accepted: August 2024 by Chelliah Sriskandarajah after two revisions.

1 Introduction

E-commerce is a significant channel for retail sales. In 2022, more than two billion people shopped online, generating 5.31 trillion USD in sales worldwide (Cramer 2023). One of the driving forces behind e-commerce growth is the channel's flexibility, which poses significant challenges to e-tailers. Almost all goods sold on e-commerce platforms require physical delivery, where goods are packaged and transported to the end customer. Consequently, online retailing incurs significant fulfillment costs. The primary phases of order fulfillment include picking, packaging, and shipping. Out of these phases, packaging plays a crucial role because of its far-reaching impact on e-commerce operations. It not only protects and identifies

the orders but also influences the transportation and distribution of goods (Azzi et al. 2012). Well-designed packaging can decrease material and shipping costs while increasing customer satisfaction.

In this article, we study packaging decisions to increase volumetric efficiency. One of the critical decisions on the packaging is the selection of optimal box sizes. In general, a small number of boxes, often less than 100 (Leung et al. 2008, Yueyi et al. 2020, Yang and Mu 2020) are used to cover all the product SKUs (stock keeping units) hosted in the warehouse. Since large platforms host thousands to millions of SKUs (Collis et al. 2018), an assortment of 100 or even 1000 box sizes is insufficient to accurately fit all the product SKUs, resulting in empty/void space in the boxes. Empty space is undesirable as it mandates the use of additional protective materials, such as air pillows and bubble wraps, that further increase the cost of packaging. More importantly, boxes with poor space utilization result in inefficient storage and transportation capacity usage. For instance, consider a truck fully loaded with boxes half-filled with the item and the other half with the protective material. In this case, though the truck is loaded fully, half of the capacity is utilized by the protective material and air; therefore, the truck effectively utilizes only half of the total capacity. In fact, 25% to 40% of an e-commerce package comprises air and filler material (Ampuja 2015). Though companies can reduce the amount of empty space by employing a larger assortment of boxes, the increased assortment size complicates procuring, handling, and inventorying, resulting in additional costs. Therefore, a manageable number of well-designed boxes is required to reduce fulfillment costs.

1.1 Problem Context

This study is motivated by a real-world problem faced by one of the largest e-commerce platforms in South Asia. We have collaborated with the company to develop the ML-based optimization framework presented in this paper. This company was established in the late 2000s as an online bookstore. With time, the company expanded into various product categories and currently hosts over 80 categories. Currently, it is one of the largest e-commerce platforms, delivering approximately 60 million shipments every month and employing fewer than 50 boxes. During the initial years of operation, the company procured standard boxes from vendors based on various SKU demands for packaging the orders. As the company grew, the number of SKUs hosted on the platform proliferated, and several new packaging boxes were added to the box assortment. Over time, the size of the box assortment increased, prompting the company to optimize the number of boxes and their dimensions. To this end, all the SKUs that require corrugated boxes for packaging are divided into three clusters based on the volumes. Among these three clusters, the third cluster carried less than 1000 SKUs but was found to be using 22 boxes. Hence, the company developed a mathematical programming formulation (described in Section 5) to optimize the packaging box assortment for this cluster. The newly generated assortment for this cluster had six boxes less than the previous assortment, and the space efficiency improved significantly. After this optimization exercise, the company planned to refresh the whole packaging box assortment for all the SKUs to improve their packaging factor (PF). To achieve

this objective, an optimization exercise for creating an assortment covering all the SKUs was planned. Though their developed mathematical programming approach was suitable in formulation, the size of the problem prohibited them from employing it (as shown in Section 6). The mixed-integer programming (MIP) approach was intractable for covering a few hundred thousand SKUs. To tackle this scalability problem, we propose a new framework in this paper. Furthermore, the framework applies to e-commerce platforms hosting thousands to millions of product SKUs.

1.2 Approaches to Solution

Determining the K optimal box sizes that maximize the overall space utilization is referred to as the box-sizing problem. It is a sparsely studied discrete optimization problem. A few studies (Xu et al. 2008, Shih Jia Lee and Thio 2015, Brinker and Gündüz 2016, Yueyi et al. 2020) formulate mathematical programming (MP) approaches, while others (Leung et al. 2008) propose heuristics. The approaches select K boxes from a set of candidate boxes for small problem instances. However, a set of candidate boxes is not available in our context. Though candidate sets can be developed by considering all possible box dimensions similar to the current practice, given the wide range of sizes, the candidate set results in many thousands of candidate boxes, which is impractical to work with given the NP-Hard complexity of the problem (Brinker and Gündüz 2016). For instance, selecting 30 boxes out of 10000 candidates results in a solution space of order 10^{87}. In contrast to the prior literature, we do not make any assumptions about the availability of candidate boxes and develop a scalable and effective optimization framework for addressing the problem.

Traditionally, NP-Hard problems are solved either by developing exact algorithms or heuristics. Exact algorithms are prohibitively expensive for large problem instances, while heuristics are cheaper to run and produce reasonable solutions without theoretical guarantees. Developing heuristics is time-consuming as it involves problem-specific research and trial-and-error effort. Besides, the performance of heuristics varies from instance to instance. Instead of hand-crafting heuristics for the problem in this study, we propose neural networks that automatically learn heuristics using reinforcement learning. Reinforcement learning (RL) is a machine learning paradigm that allows agents to learn to perform sequential decision-making tasks. RL agents have achieved superhuman performance in games such as Chess, Shogi, and Go (Silver et al. 2018). Recently, approaches for learning heuristics to solve combinatorial optimization problems, such as the Vehicle Routing Problem (Nazari et al. 2018) and Job Shop Problem (Junyoung Park and Park 2021), also found success. Inspired by their success, we propose a reinforcement learning approach for solving the box-sizing problem. In particular, we formulate the box-sizing problem as a novel sequential decision-making task called the box-sizing game. Then, we propose neural network agents to learn to solve the problem.

Our approach works in three stages. First, we treat each SKU as a data point in the three-dimensional space and solve a data-partitioning problem to generate the initial solution. We improve the initial solution

using an improvement heuristic learned in the second stage. Instead of hand-engineering the heuristic, we propose the box-sizing game, a sequential decision-making task, and develop an automated agent that learns the improvement heuristic through gameplay. We use neural network policies to represent automated agents. Neural network policies are neural networks that accept solution states as inputs and generate a probability distribution over possible improvement moves as outputs. At the end of the learning process, the distribution gives higher probabilities to moves that are likely to improve the input solution state. Finally, we integrate the learned policy in tree-search to transform the initial solution into a better final solution. To validate the performance of our optimization framework, we compare its performance with an MP-based approach.

The proposed optimization framework is effective and scalable. It generates high-quality box assortments within a few hours by considering a few hundred thousand SKUs simultaneously. We have presented the framework to our industrial partner, and they are pleased with the approach. The framework is currently deployed, and significant savings are being realized by our collaborator. Besides, the approach is not specific to any platform and can be easily applied to other e-commerce warehouses. The details of implementation and accrued savings are discussed in Section 7.

1.3 Contributions

We contribute to both research and practice. Towards research, we extend the literature on box-sizing problems and machine learning. In contrast to the existing literature on box-sizing problems, our approach caters to large problems typical of large e-commerce platforms that host hundreds of thousands of SKUs. Mainly, we do not make assumptions about the availability of candidate boxes nor generate them explicitly. We demonstrate the scalability and effectiveness of our approach by experimenting with real-world datasets. Furthermore, our approach is a fresh take on optimization problems. We present the first learning-based optimization framework for solving the box-sizing problem and one of the few to demonstrate the application of reinforcement learning in operations management. In addition, we contribute a new application to machine learning. We demonstrate how different paradigms, such as unsupervised learning, reinforcement learning, and tree search, can be combined to solve a real-world problem. The article also answers the recent calls from the AI community to establish reinforcement learning applications in real-world contexts (Chen et al. 2021). Towards practice, we solve an important problem for the e-commerce industry. The presented framework is being used successfully by our industrial partner.

The remainder of this article is organized as follows. In Section 2, we review the relevant literature. Section 3 presents the formal problem description. The optimization framework, the box-sizing game, the neural network architecture, and the learning algorithms are introduced in Section 4. Section 5 describes the company's approach and other alternative benchmarks. The computational experiments and the corresponding results are presented in Section 6. The implementation details, organizational impact, and practice commentary are presented in Section 7. Section 8 concludes the article.

2 Literature Review

In this section, we summarize the literature related to the box-sizing problem and machine learning-assisted optimization. First, we review the available work on box-sizing problems and then summarize the existing machine learning applications in optimization.

2.1 Box-Sizing Problem

The box-sizing problem is a design problem where packaging box sizes that optimize a specific criterion while covering the input set of SKUs are determined. Prior approaches focused on minimizing the unfilled space and the total packaging costs. Broadly, the approaches can be categorized into two streams. One stream of the literature assumes or generates a set of candidate boxes and then formulates the problem to select a few boxes from the available candidates. In contrast, the second stream directly generates the optimal box dimensions without the candidates. In the following paragraphs, we summarize and contrast the approaches.

Leung et al. (2008) were the earliest to consider the box-sizing problem in the first stream of literature. They solve an industrial problem where carton boxes are designed for a towel manufacturer selling three varieties of towels to distributors. Their approach considers possible combinations of towel orders and employs a genetic algorithm to select the optimal set of boxes that maximize space utilization. Xu et al. (2008) also study a business-to-business box-sizing problem where different quantities of SKUs ordered by stores located in a different continent are packaged. Here, items of the same type are packed into inner boxes, and all inner boxes addressed to the same store are shipped in one outer box. Finally, multiple outer boxes belonging to multiple stores are packed in containers for shipping. For generating the candidate inner and outer box sizes, the authors propose a cut-pack-select approach where they cut container dimensions into several candidate outer boxes and outer boxes into several candidate inner boxes. Consequently, they formulate an integer program to select the box sizes that maximize space utilization.

Unlike the previous works, which focused on business-to-business shipping, the following two studies focus on business-to-consumer delivery. Brinker and Gündüz (2016) propose a $p - median$ approach to select box sizes from a set of candidates that maximize space utilization for a small set of virtual orders. In their approach, candidates are generated by considering the minimum and maximum dimensions of SKUs. On the other hand, Yueyi et al. (2020) minimize total packaging costs. They present an MIP formulation to select a small set of boxes from the given candidate boxes. In contrast to these approaches, we study large e-commerce warehouses where hundreds of thousands of product varieties are stored and candidate boxes are unavailable. Though candidate boxes can be generated by considering all possible box dimensions, the wide variety of product sizes results in a substantial number of candidates. Such a large set of candidates targeted to cover several thousands, sometimes millions of product varieties, make Integer Programming

formulations impractical because of their NP-Complete complexity. Our approach neither assumes the availability of candidate boxes nor generates them explicitly through discretization, making it scalable to large e-commerce assortment sizes.

The work proposed by Shih Jia Lee and Thio (2015) is the only attempt that belongs to the second stream of literature. Their approach directly finds the optimal crate lengths using mixed-integer programming without requiring candidate box sizes; however, only crate lengths are determined while keeping other dimensions constant. In contrast, our approach simultaneously determines all three dimensions of the packaging boxes. We consider a three-dimensional optimization problem, while they consider a one-dimensional problem. Besides, their approach considers film rolls of different lengths, while our approach targets e-commerce warehouses that host a wide variety of SKUs. In addition, we develop a learning-based framework for solving the box-sizing problem. Specifically, we combine machine learning approaches: unsupervised learning, and reinforcement learning, with tree search. The proposed framework is a fresh take on the optimization problem and automatically generates instance-specific heuristics. Table EC.1 in the online supplement summarizes the differences between the prior works and our approach.

2.2 Machine Learning Assisted Optimization

Traditionally, large instances of discrete optimization problems are solved using hand-crafted heuristics. Recently, machine learning assisted optimization methods have been developed to tackle them (Bengio et al. 2021). These methodologies employ machine learning algorithms or neural network function approximators to either provide ancillary support to traditional solvers or build full-fledged solvers. For instance, Alvarez et al. (2017) use ExtraTrees to support the solver while tackling a mixed-integer program. Our work is closely related to the stream of literature that focuses on developing full-fledged solvers.

Full-fledged solvers based on learning methods for solving combinatorial optimization problems have shown success in the recent past (Mazyavkina et al. 2021). In these approaches, both supervised, and reinforcement learning methods are employed. Vinyals et al. (2015) were the first to propose pointer networks to solve TSP using a supervised learning method. They use heuristic solutions of TSP (travelling salesman problem) instances to train the networks and then employ them in predicting the solutions for new problem instances. Though their approach generates acceptable solutions, the need for having many solved problem instances makes it less appealing. Also, the pointer network cannot find solutions better than those seen during the training process. In order to solve the problems posed by taking a supervised learning perspective, Bello et al. (2017) propose a reinforcement learning (RL) approach to solve TSP and knapsack problems. Their method does not require solved problem instances. Instead, an artificial agent represented by a neural network learns an optimal policy to generate the solution through trial-and-error experience. Working with an RL paradigm removes the requirement of training with solved problem instances and provides an opportunity to find better solutions than heuristics. Therefore, subsequent approaches have focused primarily on

reinforcement learning. For instance, Dai et al. (2017) solve minimum vertex cover, maximum cut, and TSP problems, while Yang et al. (2022) derive pricing strategies. The proposed artificial agents or neural network policies find better solutions in all these works. In contrast to these prior approaches, we propose a novel neural combinatorial optimization framework to solve the box-sizing problem. Specifically, the framework proposes a sequential decision-making task called the box-sizing game, in which higher game scores translate to lower objective values. We then use reinforcement learning to train neural networks to learn to play the game and score higher, eventually solving the optimization problem. In addition, the framework also includes a tree-search procedure that serves as an improvement operator and increases the neural network performance in optimization. The proposed work extends the scope of problems studied in machine learning assisted optimization literature and develops a framework for solving a vital practice problem. It also answers the recent calls to establish RL applications for solving real-world problems (Chen et al. 2021).

3 Problem Description

We consider e-commerce warehouses that stock a set of carton boxes to package all SKUs hosted in the warehouse. Most e-commerce orders are single-item orders. For instance, the fraction of single-item orders fulfilled from a large e-commerce warehouse is 95.4% (Bansal et al. 2021). Likewise, the fraction of single-item orders at the Brazilian e-commerce platform Olist is 87.5% (Sionek 2018). In our case, the fraction of single-item orders varies between 90% and 95%. Therefore, we limit the analysis to single-item packaging in this article. Once an order is placed in these warehouses, the carton box that offers the most compact fit is used for packaging the SKU during the order dispatch process. Typically, warehouses employ 50 to 100 packaging box types (or sizes) to cover thousands to millions of product SKUs. Since the number of SKUs is much higher than the number of boxes used, there is bound to be empty space in the packaged box. Depending on the volume of space, paper shreds or cardboard sheets are padded to arrest products' movement. The extra packaging material and poor space utilization lead to both high storage costs at intermediate delivery hubs and high distribution costs. By employing a large set of boxes, the empty space, thereby, material and transportation costs, can be reduced; however, procuring, handling, and inventorying a larger set of boxes increases complexity and incurs additional costs. Further, a large box assortment increases the time to select a box for packaging and reduces packaging speed. Hence, a manageable number of well-designed boxes that increase space utilization is required to reduce fulfillment costs. Therefore, the objective of the solution approach is to determine the sizes of K packaging boxes that minimize overall empty space when covering the given set of SKUs. The overall empty space is quantified by a metric called packaging factor (PF), described later in this section. Note that K is a user input, and the approach does not explicitly determine the number of boxes (K) needed. Nevertheless, users can employ the approach for different values of K and choose the best K based on the marginal benefit of increasing the size of the box assortment.

3.1 Notation

In this section, we describe the general notation for the problem. We present the proposed machine learning framework in Section 4 and describe the company's prior approach along with the other mixed-integer programming benchmarks in Section 5. Our aim is to place a set of SKUs $i \in D$ ($|D| = n$), into a set of packaging boxes $j \in P$. Each SKU, i, is represented using four attributes, its length l_i, breadth b_i, height h_i, and its expected demand d_i. Similarly, for each box j, we define its length L_j, breadth B_j, and height H_j. We remark that each of these attributes for a box will be either a decision variable (in case we do not limit sizes of a box to a predefined set of candidate boxes) or a parameter (in case P is a predefined set of candidate boxes). They take the decision variable form in the proposed machine learning framework and the MIP benchmarks where the dimensions of boxes are determined. Also, here $|P| = K$. They take the parameter form in the company's prior approach where a set of K packaging boxes are selected from a given set of candidates P. We also assume that an SKU i can be placed in box j only if $l_i \leq L_j$, $b_i \leq B_j$, $h_i \leq H_j$. In all the approaches considered, we define x_{ij} as a binary decision variable that indicates whether an SKU i is assigned to box j or not. The overall compactness of a set of packaging boxes P covering the given set of SKUs D is measured by a metric called packaging factor (PF). It is defined as follows:

$$PF = \frac{\sum_{i \in D} \sum_{j \in P} d_i L_j B_j H_j x_{ij}}{\sum_{i \in D} d_i l_i b_i h_i} \tag{1}$$

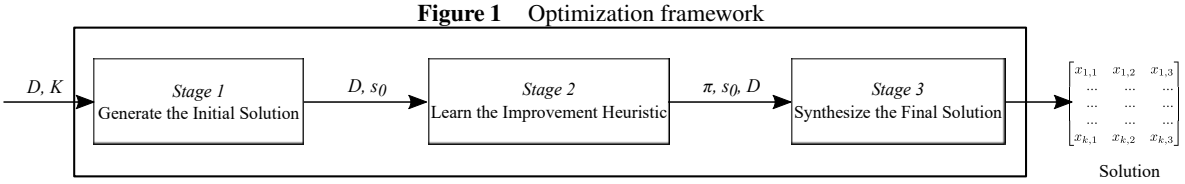
Here, $x_{ij} = 1$ if box j offers a snug fit to the SKU i. Among all the boxes, the box j is considered to offer a snug fit if it results in the lowest empty space when the SKU i is placed inside it. The numerator term computes the total packaging volume required to cover the SKUs specified in the SKU set. Note that the numerator term is the objective function at the optimal solution. The denominator term calculates the sum of volumes of the SKUs (the total volume of SKUs in D). In an ideal case, where every box perfectly fits the SKU item, the volume of the packaging box will be closer to the volume of the SKU, resulting in a unit packaging factor. However, most items cannot have an exact fit and empty space is unavoidable; therefore, packaging box volumes are greater than the SKU volumes, resulting in PF greater than 1. For example, $PF = 2$ implies that the package volume is twice that of the SKU volume: on average, every box is half empty. The proposed methodology aims to determine the dimensions of a set of K packaging boxes that minimize the PF while covering the input set D. We develop a three-stage framework to solve this optimization problem. Section 4 describes the framework. Table EC.2 in the online supplement summarizes the notations used.

4 ML-based Optimization Framework

In this section, we first overview the framework and subsequently describe the individual stages; the technical details of the algorithms are given in the online companion. Figure 1 shows the proposed optimization

framework. The framework expects the SKU set (D) and the required number of boxes (K) as inputs. Firstly, an initial solution (s_0) is generated and then passed to the second stage. In the second stage, a sequential decision-making task called the box-sizing game is formulated using s_0 and D. The game states correspond to box sizes, and the actions correspond to possible transformations. A higher game score translates to a lower value of the objective function. Here, a neural network agent plays the game for a few million steps and learns the optimal playing policy π, which serves as the improvement heuristic. Finally, by employing a tree-search procedure, a solution is synthesized from the initial solution (s_0) using the learned policy π. The procedure implicitly acts as a policy improvement operator and results in better policy decisions while synthesizing the solution. Each stage is thoroughly described in the following sub-sections.

Figure 1 Optimization framework

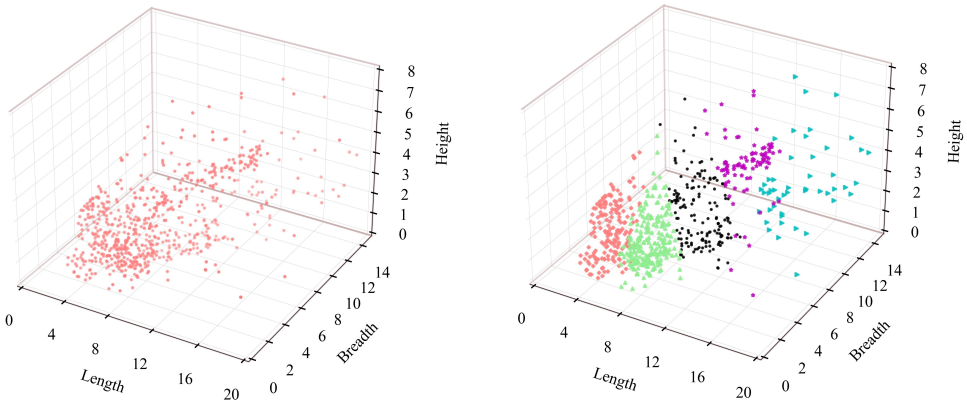


4.1 Stage 1: Generating the Initial Solution

The objective of the first stage is to generate a feasible initial solution that could serve as a starting point for improvement in later stages. The key idea is to look at each SKU from the set D as a data point in the $3D$ space. Figure 2a shows an example of such a representation of SKUs. Each point in the plot corresponds to an SKU. The three dimensions represent the SKU's length, breadth, and height.

Once all SKUs are represented in the $3D$ space, two extremities to design carton boxes are possible. In the first case, for each SKU, a box with the same dimensions can be generated. This design results in an ideal PF by creating n boxes for n SKUs. On the other hand, one large box fitting all the SKUs can be created. This design results in the poorest PF possible. Between these extremities of designing one-box-for-all and one-box-for-every, a healthy alternative is to design a box for a group of SKUs. We follow this idea and generate K-groups or subsets of SKUs by partitioning the created data space. Since a single box is expected to cover all the SKUs belonging to a group, the SKUs must be grouped such that they are as close to each other as possible in terms of Euclidean distance. Generating such groups or partitions of the dataspace is an NP-hard problem (Aggarwal and Reddy 2013); therefore, we employ a greedy heuristic from unsupervised learning, the k-means clustering algorithm.

The k-means algorithm is a simple and widely used partitioning algorithm (Aggarwal and Reddy 2013). It is highly efficient in terms of computational time and memory; this allows our framework to quickly generate the initial solution and spend more time improving it. When compared to other popular clustering methods (such as agglomerative or spectral clustering), it consumes far less computational resources,



(a) SKUs represented in $3D$ data space (b) SKU partitions generated by k-means

Figure 2

making it suitable for the problem sizes we consider. Section 6.5 establishes the scalability of the k-means method. Note that K corresponds to the number of boxes required to cover the SKU set D, while k represents the number of clusters in the k-means clustering. Observe that $k = K$. In our problem context, the k-means algorithm starts by choosing k SKUs randomly as the initial centroids, and then each SKU is assigned to the closest centroid based on Euclidean distance. Once clusters are formed, all centroids are updated, and the SKUs are reassigned; this process of centroid update and SKU reassignment continues for a predefined number of iterations. Finally, partitions of SKUs having similar dimensions are formed.

Once k partitions are formed, boxes that fit each partition can be generated quickly. Let $s = \{(l_r, b_r, h_r)\}_{u=1}^{v}$ be any partition or subset consisting of v SKUs. If $(l_{max}, b_{max}, h_{max})$ be the maximum length, breadth, and height among all SKUs of the respective partition, then a box with the exact dimensions $(l_{max}, b_{max}, h_{max})$ can fit all SKUs belonging to that subset. Following this proposition, K boxes can be generated for k partitions resulting in a K-box assortment. Figure 2b shows an example representation of SKU clusters formed using the k-means algorithm. Here five clusters are formed, which can be translated into five unique box sizes. Algorithm 1 in the online companion summarizes the procedure to generate the initial solution.

4.2 Stage 2: Learning the Improvement Heuristic

In this section, we develop a learning-based approach for improving the initial solution. Improvement heuristics are generally used to transform the initial solution into a better final solution. These are algorithms or transformation rules that are hand-crafted through trial and error, leveraging domain-specific knowledge. However, hand-engineering new algorithms or heuristic rules is laborious; in addition, the performance of the heuristics can vary from instance to instance. To overcome the dependence on human scientists to create these algorithms and generate instance-specific heuristics, we propose a learning-based approach. In our approach, an automated agent learns the improvement heuristic from experience gained through interactions

with a simulation model. Specifically, we translate the box-sizing problem into a sequential decision-making task called the box-sizing game. The game is designed such that high scores in the game correspond to lower values of the minimization objective, PF. Subsequently, an automated agent learns to play the game and scores higher, eventually solving the optimization problem.

In the box-sizing game, all the game states an agent can reach are the K-box assortments themselves. The game always starts with the initial solution, say s_0, as the initial game state. At any time point t, the agent can choose between two types of actions. The first type of action modifies the current game state, a K-box assortment s_t, into another K-box assortment s_{t+1}. Based on the PF of the states s_t and s_{t+1}, the agent receives positive or negative rewards for improving or worsening the solution. The agent can also choose the second type of action, a resignation that submits the current K-box assortment and exits the game. The agent aims to maximize the game score, which is the accumulated reward. Learning to play the game translates into learning the improvement heuristic, and scoring high translates to reducing PF to the minimum.

The box-sizing game we described is a sequential decision-making task; we use the most popular decision-making formalism, the Markov Decision Process (MDP), to model it. MDP is generally specified using a tuple of states, actions, rewards, and transition dynamics. In our box-sizing game, the state space includes all the possible K-box assortments, with each game state corresponding to a solution. Actions are the possible transformations of the game states. By playing a sequence of actions, the agent transforms the initial state into a final solution. Actions are improving or worsening in nature; the quality of action depends on the state from where it is executed. Therefore, we define a reward function that maps the state-action pairs to scalar reward values. The transition dynamics specify the probability of reaching the next state s_{t+1}, from a current state s_t when an action a_t is applied, $P(s_{t+1}|s_t, a_t)$. In the box-sizing game, the actions from a state always result in the same transformations; therefore, the MDP is deterministic, resulting in the unit or zero probabilities. During the gameplay, the next state s_{t+1} is independent of the previous game states and actions when the current state s_t and action a_t are known; therefore, the Markov property holds.

The goal of the agent interacting with the MDP is to learn a policy $\pi_\theta(a|s)$ that maps the states to actions, such that following the actions specified by the policy maximizes the game score. Here, θ represents the policy parameters. Our agent learns this policy using reinforcement learning. The following sections define the MDP components: states, actions, rewards, and the learning algorithm.

4.2.1 State Space

The state of the MDP should capture all the information the agent requires. The whole K-box assortment is treated as a state in our box-sizing game. Since each box is represented by (l, b, h) dimensions, the complete state becomes a $K \times 3$ matrix. The i^{th} row of the matrix specifies the length, breadth, and height of the i^{th} box. To reduce redundancy among the representations, the dimensions of the box are arranged such that $l >= b >= h$. Note that the carton boxes cannot be manufactured with extremely high precision; therefore, the dimensions take discrete values with a step size.

4.2.2 Action Space

The actions of the MDP correspond to the transformations of the game state. We define transforming actions and resigning actions. Transforming actions alter the current K-box assortment into a new assortment while resigning actions submit the current assortment and exit the game. In total, there are $6K+1$ actions numbered from 0 to $6K$. The first $6K$ actions from 0 to $6K-1$ are transforming actions, and the last action, numbered $6K$, is a resignation action. Out of the $6K$ transforming actions, the first $3K$ actions from 0 to $3K-1$ decrement the box dimension by the step size. The subsequent $3K$ actions from $3K$ to $6K-1$ increment the box dimensions by the step size. Figure 3 shows an example of transforming actions. It demonstrates the effect of actions on a 3-box assortment with a step size of 0.5 units. The 3×3 matrices represent the dimensions of the assortment. Columns indicate length, breadth, and height, while rows index individual boxes. Since it is a 3-box assortment, the first $3K$ actions from 0 to 8 decrement the box dimensions by the step size, and the next $3K$ actions from 9 to 17 increment the box dimensions. For instance, action 0 decrements the length of the first box, action 4 decrements the breadth of the second box, and finally, action 8 decrements the height of the last box. Similarly, actions 9, 13, and 17 increment the respective dimensions. Notice that the action numbers correspond to the indexes of dimensions when the matrix is vectorized by stacking rows.

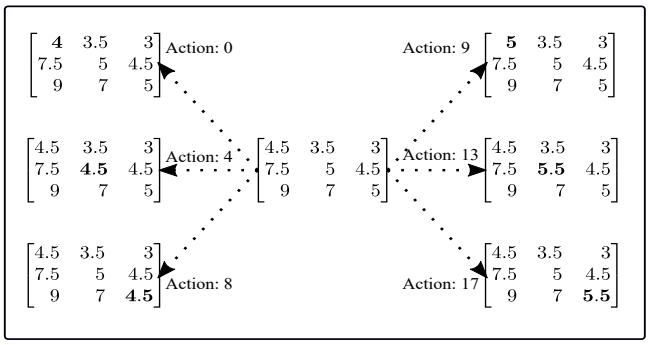


Figure 3 Illustration of the effect of actions on a 3-box assortment

4.2.3 Rewards & Terminal Conditions

$$r(s_t, a_t, s_{t+1}) = \begin{cases} 0, & \text{if resignation action is selected and terminate} \\ -1, & \text{if } s_{t+1} \text{ is infeasible or } PF(s_{t+1}) > PF(s_0), \text{ and terminate} \\ PF(s_t) - PF(s_{t+1}), & \text{if } s_{t+1} \text{ is feasible} \end{cases} \quad (2)$$

The reward function specifies what needs to be achieved. In our box-sizing game, positive, negative, or zero rewards are given for each action, and the agent accumulates rewards during the gameplay. Equation 2 shows the designed reward function. At any point t in the game, the agent can choose to pass the resignation action numbered $6K$, which will submit the K-box assortment represented by s_t, and the game environment returns zero reward. The accumulated rewards or the game score remain unchanged. On the other hand,

three outcomes are possible if the agent chooses to alter the assortment by employing transforming actions. Suppose the resultant state s_{t+1} is an infeasible solution, or the solution quality is worse than the initial solution state s_0. In that case, a negative reward of -1 is awarded to the agent, ending the game. Also, the accumulated rewards are decremented by 1 due to the addition of the negative reward. These conditions encourage the agent to submit the solution and end the game voluntarily instead of exploring poor solutions. Suppose the resultant state s_{t+1} is feasible, then a value proportional to the improvement in solution is awarded as a reward. Since our minimization objective is the packaging factor (PF), we award a reward equal to $PF(s_t) - PF(s_{t+1})$. This expression gives a positive reward for improving the solution quality and a negative reward for decreasing the solution quality.

4.2.4 Learning to Play the Box-sizing Game

Our automated agent uses deep neural networks to make action selections during gameplay. The network consists of 2 to 3 hidden layers; the architecture is shown in Section EC.3 in the online companion, and the exact parameters and the procedure to obtain them are specified in EC.4 and EC.5. The input layer consists of $3K$ units, which takes all the dimensions of the K-box assortment. The output layer consists of $6K+1$ units corresponding to the actions. When a game state s_t is passed to the input layer, the network outputs a probability distribution over actions. The agent samples an action from the output distribution and passes it to the game environment. This neural network serves as a policy in the box-sizing game and assists the agent in selecting actions. However, for the policy to behave optimally, that is, to maximize the score during gameplay, the parameters θ of the neural network must be learned. We adapt proximal policy optimization (PPO) (Schulman et al. 2017), a reinforcement learning algorithm for learning the policy parameters.

The main objective of this stage is to train the neural network to learn a policy that would maximize the game score during gameplay. To this end, we initialize a neural network with random parameters and adjust the parameters using gameplay experience. The box-sizing game starts with the initial solution state s_0 and the game score set to zero. At $t = 0$, the agent inputs s_0 to the neural network, observes the output action probabilities $\pi_\theta(.|s_0)$, samples an action a_0, and then passes it to the environment. The game environment executes the action and returns the next state, s_1, and a reward, r_1. The agent collects the transition example (s_0, a_0, s_1, r_1). This agent environment interaction process continues in the following time steps $t = 1, 2, 3\ldots$, and the agent collects a set of transition examples $\{(s_1, a_1, s_2, r_2), (s_2, a_2, s_3, r_3), (s_3, a_3, s_4, r_4), \ldots\}$ until the game reaches any one of the terminal conditions. This path of state-action pairs from the initial state to the terminal state is known as an episode. We could use this experience collected in one episode to update the policy parameters; however, this would result in high variance updates. Therefore, instead of collecting transition examples from one game episode, we collect many transition examples from a few episodes to reduce the variance. Readers can refer to Sutton and Barto (2018) for a thorough review of variance in updates. Once z number of transition examples are collected using the policy network, we update

the policy parameters using the PPO update (Schulman et al. 2017). PPO is a state-of-the-art reinforcement learning algorithm for yielding optimal neural network policies. It avoids drastic policy changes during subsequent updates by limiting how much the new policy can change from the old policy. This behavior improves training stability and reduces the chance of policy collapse, where the networks suddenly forget the learned behavior. The following equation gives the PPO update:

$$\theta_{t+1} = \arg\max_{\theta} \mathop{\mathbb{E}}_{s,a\sim\pi_{\theta_l}} [L(s,a,\theta_l,\theta)] \tag{3}$$

where $L(s,a,\theta_l,\theta)$ is given by

$$L(s,a,\theta_l,\theta) = \min(q_\theta A^{\pi_{\theta_l}}(s,a), clip(q_\theta, 1-\varepsilon, 1+\varepsilon) A^{\pi_{\theta_l}}(s,a)) \tag{4}$$

$$q_\theta = \frac{\pi_\theta(a|s)}{\pi_{\theta_l}(a|s)} \tag{5}$$

In every iteration during the training process, the PPO update aims to maximize the objective $L(s,a,\theta_l,\theta)$ by finding a new neural network policy parameterized by θ. Here, q_θ measures how probable an action a at state s is under the new policy when compared to the old policy found in the previous iteration (parameterized by θ_l). $A^{\pi_{\theta_l}}(s,a)$ is the advantage estimate of taking action a at state s; it is calculated by accounting for all the subsequent rewards possible (Refer to EC.6 for a description of advantage estimation). To maximize the objective $L(s,a,\theta_l,\theta)$, the PPO update finds a new policy that gives higher probabilities to actions with higher advantage estimates. However, when an action becomes very much probable under the new policy when compared to the old policy, q_θ takes large values, which in turn results in higher objective function values. Large objective function values cause large destructive updates that compromise the optimization goal, resulting in poor policies. The clip function in the objective prevents this effect. In particular, the clip function limits how much the action probabilities can change under the new policy compared to the old policy. To this end, it adjusts the q_θ values between the threshold interval $[1-\varepsilon, 1+\varepsilon]$. For instance, if $\varepsilon = 0.1$, the function becomes $clip(q_\theta, 0.9, 1.1)$ and limits the q_θ values between $[0.9, 1.1]$. In other words, this restricts the new policy to be 10% different than the old policy and prevents drastic policy changes with each update. We refer the interested reader to (Schulman et al. 2017, OpenAI 2018) for a comprehensive review of the PPO update and its components, and to Algorithm 2 in the online companion, which presents the learning procedure for training neural networks.

Note that we employ a state-of-the-art policy-based approach (PPO) with neural networks over a value-based approach for solving the second stage. We adopt this approach because of the inherent advantages of policy-based methods over value-based methods. First, policy-based methods are highly effective in comparison to value-based methods in high-dimensional action and state spaces. In our problem, the actions

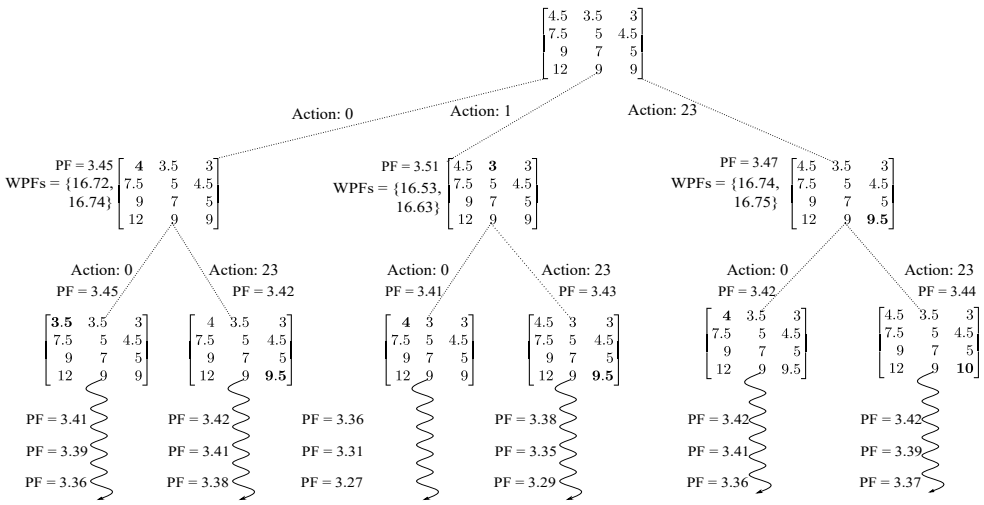


Figure 4 Search tree to improve a 4-box assortment

range in a few hundred, while the state space ranges in the orders of greater than 10^{100}, and therefore, estimating values for each state-action pair with value-based methods is ineffective. Moreover, policy-based methods are known to have stable training and better convergence properties as their action preferences change smoothly over training. In the value-based methods, because exploration/exploitation has to be implemented separately, the action preferences may change erratically, resulting in lower stability during training. For a thorough understanding of the differences between both approaches, refer to Sutton and Barto (2018).

4.3 Stage 3: Searching with the Policy Network

The main objective of this stage is to transform the initial solution generated in the first stage using the neural networks trained in the second stage. Though the trained neural networks can be directly applied on s_0 to generate a solution, the approach presented in this section improves the action decisions made by the network, resulting in a better final solution than directly applying the policy from the second stage. The experiments described in Section 6.2 confirm this fact. To illustrate this method, we set up a search tree with the initial solution s_0 as the root node. Child nodes correspond to the transformations of the parent node. Transforming the solution corresponds to making action selections at each level while traversing the tree depth. Figure 4 shows the search tree corresponding to a 4-box assortment. Every parent node has 24 children corresponding to all the actions. Due to space constraints, only 2-3 children of a parent are shown at each level.

In this method, actions are selected based on consequences. Here, the search procedure employs the trained neural networks to simulate the consequences of actions and then commits to the child nodes at each level. Since the search procedure actively employs the policy and makes the decisions by itself, we call this method: policy-assisted active search (PAAS). While traversing the search tree, suppose we are at a state s,

let the policy suggested action be a_p (say it leads to state s_p); instead of directly selecting it, we ask whether we should choose another action a_x (say it leads to state s_x). It would be wise to select a_x if it results in better solutions in the consequent search path. One way of observing the consequences of an action selection is to evaluate all the descendant nodes exhaustively. However, visiting all nodes is impractical as it leads to a combinatorial explosion. The desired alternative is to check only a small percentage of nodes that offer the best solutions among the descendants. Notice that the neural network policy is trained to find good future states or descendants, that is, to directly select the optimal actions by looking at the state. Therefore, we use the trained policies to simulate the future search trajectories and observe how the objective values transpire in the future if we select an action now.

In summary, the key idea of PAAS is to allow the policy to play out actions from each child node till its resignation and observe the objective values in resultant search trajectories. An optimally trained policy shows the best solutions that can be reached from a child node; accordingly, the child node that could give a better solution can be selected. We define a weighted packaging factor (WPF) metric to quantify a search trajectory's goodness. Suppose $< s_x, s_1, s_2, s_3, ..., s_\tau >$ is a search trajectory with packaging factor values $< PF_x, PF_1, PF_2, PF_3, ..., PF_\tau >$, the WPF is given by

$$WPF = PF_x + \beta PF_1 + \beta^2 PF_2 + ... + \beta^{\tau-1} PF_\tau \tag{6}$$

The WPF scores indicate the average quality of the descendant solutions where β specifies the weights. The search can expect to reach solutions with low objective values by selecting actions that result in low WPF values. High values of $\beta \in [0.9, 1]$ make the search far-sighted and reduce the greedy nature, while low values increase it. Figure 4 illustrates the PAAS search procedure. Suppose the search starts at the root node; the search routine must decide on the child node to proceed forward. Here, PAAS expands all 24 child nodes and simulates the possible descendants from each child node. In the figure, four steps are simulated from each child node two times. The PF values along the dashed lines indicate the objective values of descendants encountered during the simulation. For instance, two paths are simulated from action 0, resulting in $< 3.45, 3.45, 3.41, 3.39, 3.36 >$ and $< 3.45, 3.42, 3.42, 3.41, 3.38 >$. Based on this, two WPFs are calculated: 16.72 and 16.74. Similarly, two trajectories are simulated from all the child nodes, and WPFs are calculated. Observe that the WPF values from Action 1 are the lowest, indicating that the descendants would result in better solutions; therefore, the search procedure selects the child node corresponding to Action 1. This process of repeatedly selecting the child nodes is repeated until a predefined number of iterations, and finally, the best solution observed during the search is returned. Refer to Algorithm 3 in the online companion for the algorithm that summarizes the PAAS procedure.

There is one essential detail to notice here. Though the RL policy learned in stage 2 may appear greedy, it considers long-term consequences. In stage 2, the RL approach is formulated by accounting for all the

consequent rewards using the advantage estimate $A^{\pi_{\theta_l}}(s,a)$ (see Eq. 4). Therefore, an optimal RL policy should account for the long-term behaviors inherently. However, we can only learn an approximation of the global optimal RL policy during training because we observe only a small set of episodes and use first-order optimization methods to learn policy network parameters (which is standard practice to make the learning problem computationally tractable). Therefore, the learned RL policy cannot completely solve myopic behaviors and holds potential for improvement. It is also not greedy because it is trained to make decisions based on the returns of the overall episode or trajectory during training. We use the third stage to improve the decisions made by the RL policy. Observe that the PAAS procedure in stage 3 selects actions based on simulations with respect to the learned policy; this style of action selection is a special case of the policy improvement theorem (Sutton and Barto 2018). It improves the neural network policy decisions during the search, resulting in action selections that are at least as good as the standalone policy. Therefore, the PAAS method can also be viewed as a policy improvement operator for the learned neural network policy. Section 6.2.2 discusses this aspect through experimental results. Section EC.7 in the online supplement illustrates the three stages.

5 Company's Optimization Approach & Alternative Benchmarks

In this section, we briefly present alternative mathematical programming formulations for the problem on hand, including the approach employed by the company. We remark that these MIP formulations attempt to solve the problem with slightly different perspectives, however, this provides the decision makers an assessment of both suitability and applicability in a given setting while considering a breadth of methodologies, rather than being myopic. They also help us to establish the validity and effectiveness of the proposed approach.

Before moving forward with the alternative formulations, we first discuss basic problem assumptions and variables. Each MIP aims to place a given set of SKUs $i \in D$ ($|D| = n$) into a set of packaging boxes $j \in P$, where P may be either a predefined set of finite (and large) number of options, or simply an infinite number of possibilities in a continuous range of dimensions. As discussed in Section 3, each SKU, i, is represented using four attributes: its length l_i, breadth b_i, height h_i, and its expected demand d_i. Similarly, each box j is represented by its length L_j, breadth B_j, and height H_j. Recall that each of these attributes for a box will be either a parameter (in case P is a predefined set of options for packaging sizes) or a decision variable (in case we consider any possible size for a dimension and not only predefined options). We also make the following modeling assumptions/restrictions: (1) an SKU i can be placed in box j only if $l_i \leq L_j$, $b_i \leq B_j$, $h_i \leq H_j$; (2) each SKU should be allocated to only one box; and (3) exactly K boxes should be selected from P. In case of a predefined set of P, we also assume that (4) a set of P packaging box dimensions are available, which may be e.g. obtained through discretization by considering all possible box dimensions. In all the MIP formulations considered, we define x_{ij} as a binary decision variable that indicates whether an SKU i is assigned to box j or not.

5.1 Company's Approach

We describe the mathematical program employed by the company in this section. This formulation is in line with the MIP formulation of Brinker and Gündüz (2016), and it assumes that the set P is a fixed (but rather large) set of candidates for different packaging dimensions, where these dimensions are obtained in our case through discretization by considering all possible box dimensions. Though this approach removes the freedom on choosing box dimensions and also may potentially result in significant computational challenges due to sheer number of candidates one would obtain from discretization, it is often the method of choice in the industry, including industry leaders such as Amazon (O'Neill 2022), where it is applied in a rather crude and approximate fashion (e.g., they use 2-inch intervals to initiate the process and then apply the framework iteratively in an approximate manner, whereas our framework uses 0.5 inch intervals using an exact approach). Our latter approaches introduced in Section 5.2 offers more flexibility in obtaining the box dimensions (length, breadth, and height) from a continuous range instead of pre-determined discrete values. By defining the coefficient c_{ij} as the cost of placing SKU i in box j (i.e., the total volume incurred due to the assignment) and the binary decision variable y_j that indicates whether j^{th} box is selected in the K-box assortment or not, the mathematical program is stated as follows:

$$\min \quad \sum_{i\in D, j\in P} c_{ij}x_{ij} \tag{7}$$

$$\text{s.t.} \quad \sum_{j\in P} x_{ij} = 1 \qquad \forall i \in D \tag{8}$$

$$\sum_{j\in P} y_j = K \tag{9}$$

$$x_{ij} <= y_j \qquad \forall i \in D \;\; \forall j \in P \tag{10}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in D \;\; \forall j \in P \tag{11}$$

$$y_j \in \{0,1\} \qquad \forall j \in P \tag{12}$$

Here, $c_{ij} = d_i L_j B_j H_j$, which computes the package volume required to cover an SKU i having a demand d_i with a box j. However, in cases where the SKU is larger than the box, x_{ij} variables can be simply eliminated through preprocessing to allow only feasible assignments. The objective function aims to minimize the total packaging volume which results in selecting compact packaging boxes to cover the SKU set. Constraint (8) ensures that each SKU is assigned to only one box. Constraint (9) ensures that only K boxes are selected. Constraint (10) ensures that an SKU i will only be assigned to box j if box j is selected in the K-box assortment. In summary, the model selects K number of boxes from P that reduce the packaging volume (or overall empty space) while covering the given set of SKUs, D. The company uses the described model several times by varying K to generate box assortments of different sizes. Once all box assortments are generated, they compare assortments using a measure of compactness, the packaging factor (PF).

5.2 Alternative Benchmark Approaches

We present two more MIP formulations for comparing the proposed approach and establishing its effectiveness. Different than the company's approach presented in Section 5.1, these alternative formulations allow an unlimited choice of box dimensions due to considering continuous ranges of values (rather than a predefined set of candidates) for each box dimension. The first formulation we consider aims, broadly speaking, for a best fit by minimizing the difference between each SKU and box dimension for each assigned pair. More specifically, we define continuous decision variables $\delta^l_{ij}/\delta^b_{ij}/\delta^h_{ij}$ for the difference between the length / breadth / height of the SKU i and box j. Then, this results in the following formulation minimizing 1-norm distance, with expected demands used as weights, and $|P| = K$ due to variable nature of box dimensions.

$$\min \quad \sum_{i\in D, j\in P} d_i(\delta^l_{ij} + \delta^b_{ij} + \delta^h_{ij}) \tag{13}$$

$$\text{s.t.} \quad x_{ij} l_i \le L_j \qquad \forall i \in D, j \in P \tag{14}$$

$$x_{ij} b_i \le B_j \qquad \forall i \in D, j \in P \tag{15}$$

$$x_{ij} h_i \le H_j \qquad \forall i \in D, j \in P \tag{16}$$

$$(L_j - l_i) + (x_{ij} - 1) l_{max} \le \delta^l_{ij} \qquad \forall i \in D, j \in P \tag{17}$$

$$(B_j - b_i) + (x_{ij} - 1) b_{max} \le \delta^b_{ij} \qquad \forall i \in D, j \in P \tag{18}$$

$$(H_j - h_i) + (x_{ij} - 1) h_{max} \le \delta^h_{ij} \qquad \forall i \in D, j \in P \tag{19}$$

$$\sum_{j\in P} x_{ij} = 1 \qquad \forall i \in D \tag{20}$$

$$x_{ij} \in \{0,1\}, L_j, B_j, H_j, \delta^l_{ij}, \delta^b_{ij}, \delta^h_{ij} \ge 0 \qquad i \in D, j \in P \tag{21}$$

We note that constraints (14)-(16) guarantee assumption (1) only for the active (i, j) pairs. Moreover, constraints (17)-(19) are big-M constraints (maximum SKU dimension, i.e., $l_{max}, b_{max}, h_{max}$, as big-M values) that hold as equation for active (i, j) pairs due to the nature of minimization problem. This model is previously investigated by Shih Jia Lee and Thio (2015), albeit for a one-dimensional problem and very small test instances.

The high number of continuous variables for the difference between each dimension of SKU i and box j, along with numerous big-M constraints, may cause computational challenges in the MIP setting, and therefore, we consider next a more crude model that is rather an approximation of the previous model. More specifically, we define only a single continuous variable, Δ, as to capture the worst possible difference between dimensions weighted with the expected demand. This results in the following formulation, which is essentially an ∞-norm problem (if one considers each tuple of one of the three dimensions and SKU/box pairs as a dimension), or equally, a min max problem (minimizing the worst difference.)

$$\min \quad \Delta \tag{22}$$

$$\text{s.t.} \quad x_{ij} l_i \leq L_j \qquad \forall i \in D, j \in P \tag{23}$$

$$x_{ij} b_i \leq B_j \qquad \forall i \in D, j \in P \tag{24}$$

$$x_{ij} h_i \leq H_j \qquad \forall i \in D, j \in P \tag{25}$$

$$d_i \left((L_j - l_i) + (x_{ij} - 1) l_{max} \right) \leq \Delta \qquad \forall i \in D, j \in P \tag{26}$$

$$d_i \left((B_j - b_i) + (x_{ij} - 1) b_{max} \right) \leq \Delta \qquad \forall i \in D, j \in P \tag{27}$$

$$d_i \left((H_j - h_i) + (x_{ij} - 1) h_{max} \right) \leq \Delta \qquad \forall i \in D, j \in P \tag{28}$$

$$\sum_{j \in P} x_{ij} = 1 \qquad \forall i \in D \tag{29}$$

$$x_{ij} \in \{0,1\}, L_j, B_j, H_j, \Delta \geq 0 \qquad i \in D, j \in P \tag{30}$$

We finally remark that these two alternative benchmark formulations do not explicitly minimize the total volume simply to avoid nonlinearity that would stem from variable box dimensions but rather conserve linearity by minimizing each individual box dimension. Further technical discussion on this aspect, along with the company's approach, is presented in the online supplement in Section EC.8.

6 Computational Analyses

This section describes the experimental settings for evaluating the ML-based optimization framework and the corresponding results. As discussed in Section 4, the optimization framework requires two inputs: 1) the four attributes (length, breadth, height, expected demand) of all the SKUs present in the set D and 2) the required number of boxes (K). We employ the dataset supplied by our industrial partner for the experiments. The dataset consists of the length, breadth, and height dimensions of 153,625 SKUs and their expected demand observed during a month. Note that some of the SKUs are fragile and require clearance inside the box to keep protective material such as air pillows or bubble wrap. In the dataset, all the SKU dimensions are adjusted for the clearance values. For instance, say l, b, h are the length, breadth, and height dimensions of an SKU item, and if it requires a cushioning of 0.2-inch on all sides, the SKU dimensions are represented as $l + 0.2, b + 0.2, h + 0.2$ in the dataset. The minimum and maximum clearances required are 0 and 2 inches, respectively, in the dataset. After adjusting for clearances, the final dimensions range between 0.5 and 29.5 inches. We refer to this dataset as D_{real}.

We present two sets of analyses in this article. In the first set, we generate box assortments of various sizes to cover the SKUs in D_{real}. In particular, we generate an initial solution for each K-box assortment using stage 1 and improve them using the proposed approach. We report the training dynamics of the neural network agents and contrast the packaging factor (PF) of solutions generated by each stage. The first

set of experiments helps us enquire whether the proposed box-sizing game enables a neural network to learn an improvement heuristic and shows its effectiveness. In addition, they show the incremental value of each stage of the framework. In the second set of experiments, we compare the proposed approach with the company's prior approach and other alternative MIP benchmarks described in Section 5. To test the company's approach against ours, we create problem instances in line with real-world use cases. In particular, we create 12 datasets consisting of SKUs between 2000 and 7500 by randomly selecting a subset of SKUs from D_{real}: $D11\_2000$, $D12\_2500$, $D13\_3000$, $D14\_3500$, $D15\_4000$, $D15\_4500$, $D16\_5000$, $D18\_5500$, $D18\_6000$, $D20\_6500$, $D21\_7000$, $D22\_7500$. The dataset names signify the range of dimensions and the number of SKUs sampled. For instance, $D12\_2500$ has the details of 2500 SKUs whose dimensions are less than or equal to 12 inches. Refer to Section EC.9 in the online companion for more details on the datasets. To cover these datasets, we generate box assortments of sizes 10, 20, 30, and 40 using both approaches and report PFs, computational times, and the gaps reported by the state-of-the-art MIP solver. The explicit details of the solver, computational configuration, and solution times are reported in Section 6.3. We present the comparison of the proposed approach with alternative benchmarks in Section 6.4. We explore alternative approaches for generating the stage-1 solution in Section 6.5. Furthermore, we compare the performance of the proposed approach against standard meta-heuristics approaches in Section EC.10 of the online companion. We explore alternative ML-based methods to solve the box-sizing problem in Section EC.11. The robustness analysis is provided in Section EC.12.

6.1 Behavior of the Neural Networks

In this section, we first discuss the training dynamics of the neural networks and then contrast the performance between individual stages of the framework. Figure 5 shows the training dynamics of the neural networks while they learned to design the 10, 20, 30, and 40 box assortments by transforming the respective initial solutions. Each subplot corresponds to a unique K-box assortment. During the training process, after every z steps of gameplay, the neural network parameters are updated, and the average game score obtained in the respective game episodes is plotted. The horizontal axis represents the number of steps played, and the vertical axis represents the average score obtained in a game episode. Note that the neural networks can improve, worsen, or quit the game without changing the initial assortment. Accordingly, positive scores are awarded for improvement and negative scores for worsening or reaching infeasible solutions, as discussed in Section 4.2.3. Figure 5 shows that the networks have worsened the solution during the initial training steps, and the environment exited the game with a high negative score. However, with training, the networks learned to play a mix of improving and worsening moves, receiving fewer negative scores. Towards the end of the training process, the neural networks learned to play only improving moves and consistently reached positive rewards without reaching infeasible states. This learning behavior can be seen consistently across four assortments and suggests that the proposed learning task is valid.

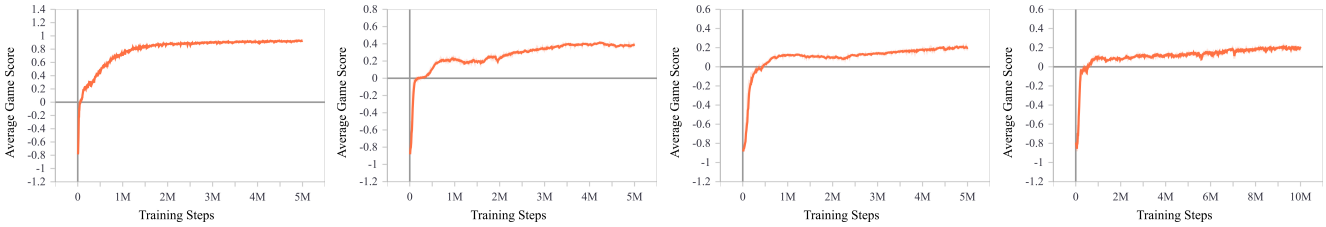


Figure 5 Training dynamics of neural networks while generating: 10, 20, 30, 40 box assortments

6.2 Incremental Value of Each Stage

In this section, we delineate the importance of individual stages. Table 1 reports the PFs of the solutions generated by each stage to cover the SKUs present in the dataset D_{real}. Observe that the solution improves with each stage across all the assortment sizes (ranging from 10 to 100). On average, the improvement from stage 1 to stage 2 is 7%, stage 2 to stage 3 is 9%, and stage 1 to stage 3 is 15%. These figures indicate that each subsequent stage improves the solution significantly and shows their contribution to the overall solution approach. In the following subsections, we discuss the importance of stage-1 & stage-3.

Table 1 Packaging factors (PFs) of the solutions generated by each stage

	Assortment Size									
Stage	10	20	30	40	50	60	70	80	90	100
I	3.963	2.686	2.464	2.289	2.166	2.129	2.076	2.022	1.968	1.958
II	3.006	2.437	2.286	2.175	2.077	2.013	1.984	1.925	1.899	1.863
III	2.930	2.305	2.056	1.959	1.867	1.813	1.777	1.764	1.717	1.699

6.2.1 Importance of Stage 1

The initial solution generated by stage 1 serves as a tight starting point for the neural network policy to improve upon. It helps the network concentrate on every action and ultimately learn the improvement moves. We perform the following study to establish the same. This study assumes that an initial solution from stage-1 is unavailable for stage-2 and stage-3. Nevertheless, since both stages require an initial solution, we create a feasible initial solution for use with these stages. Recall that the solution is represented by a $K \times 3$ matrix, with each row representing rows and columns representing different dimensions. If we initialize all the entries of the solution matrix with a value equal to the maximum possible dimension of an SKU (in the SKU dataset D_{real}), the resultant solution assortment will be feasible and can serve as a starting point for both stages 2 & 3. For instance, if the maximum possible dimension of any SKU in the SKU set is 30 inches, then the initial solution will have K boxes, each with a length, breadth, & height equal to 30 inches. This initial solution is passed to both stages 2 & 3 in the study for creating a K-box assortment. Table 2 shows the PFs of the solutions generated by the proposed framework using the initial solution discussed above (referred to as without stage-1) and with the initial solution generated by stage-1. Observe that there is a

significant difference in the qualities of solutions between both approaches. Using the solution generated by stage-1 appreciably improves the performance across all assortments. The performance difference is due to the nature of the initial solution. When an initial solution, as described in this study, is used, the agent will be presented with a set of K large boxes. In this case, the agent can choose any decrementing action (corresponding to any box), improve the assortment, and reap positive rewards. The box choice and dimension do not matter until an assortment of different sizes is formed. Moreover, the agent should continue selecting decrementing actions for a significant part of the game to arrive at a compact assortment. Therefore, the agent will be biased towards decrementing actions, reducing its learnability. In contrast, the solution generated by stage-1 offers a tight starting point for the agent to improve. The agent should concentrate on the box and the modification step, or the game will terminate with a poorer score.

Table 2 Packaging factors (PFs) of the solutions with & without stage-1

	Assortment Size									
	10	20	30	40	50	60	70	80	90	100
with Stage 1	2.930	2.305	2.056	1.959	1.867	1.813	1.777	1.764	1.717	1.699
without Stage 1	3.544	3.156	3.619	3.112	3.484	3.204	3.373	4.285	3.168	3.350
Difference (in %)	-21	-37	-76	-59	-87	-77	-90	-143	-85	-97

6.2.2 Importance of Stage 3

The stage 3 selects actions by rolling out the policy learned in stage 2; this style of action selection invokes the policy improvement theorem (Sutton and Barto 2018) and improves the quality of policy decisions during the search. Suppose we are at a state s; let the policy suggested action be a_p (say it leads to state s_p); instead of directly selecting it, we ask whether we should choose another action a_x (say it leads to state s_x). It would be wise to select a_x if it results in better solutions in the consequent search path. We use the learned policy to play out from s_x and check whether it can generate better solutions in stage-3. Nevertheless, if no action a_x exists that can generate better solutions than a_p, the stage-3 search selects a_p. Therefore, stage-3 selects an action that is at least as good as the policy-suggested action a_p at every step. This way of searching in stage-3 improves action selections one step at a time and works as a policy improvement operator for the learned policy. As discussed previously, from Table 1, stage-3 improves the solution by 9% on average.

6.3 Comparison of ML-approach with the Company's MIP Approach

In this section, we compare the performance of the proposed ML-based optimization framework (PAAS) with the performance of Gurobi in solving the integer program described in Section 5.1. To this end, we apply both approaches to the 12 SKU datasets described as experimental settings in this section. In particular, we generate 10, 20, 30, & 40 box assortments for covering the datasets. All the experiments are

conducted on a high-performance computer instance with 24 Intel Xeon Gold 6140 cores and 192 GB of RAM. We set the time limit to 20 hours. Table 3 shows the packaging factor values of the box assortments generated by both approaches, run time in hours, the optimality gap reported by the Gurobi solver, and the difference in solution quality between both approaches. The results show that the PAAS approach outperforms the Gurobi solver in terms of run time. Most of the problem instances could not be solved by the Gurobi solver within the time limit of 20 hours, while the PAAS approach takes 1 to 3 hours to solve them. In addition, the computational resources consumed by Gurobi were too high. The solver failed with an out-of-memory error while solving the instance with 7500 SKUs & 10772 box candidates, while the PAAS approach could efficiently work with less than 16GB of RAM. Regarding solution quality, up to a small number of SKUs (4000), the Gurobi solver performed better, albeit with solutions that are, on average, only 0.44% better. On the other hand, as the number of SKUs increases to greater than 4500 SKUs, the solver fails to generate practically usable solutions due to very poor solution quality. The solver reports gaps of about 90% and more. In instances greater than 6500 SKUs, a 100% gap can be seen. On the other hand, the PAAS generates high-quality solutions within the time limit. These experiments show that the company's approach using Gurobi is unsuitable for solving medium to large problem instances. At the same time, the PAAS approach consumes less computational resources and generates high-quality solutions across all instance sizes.

6.4 Comparison with the Alternative Benchmarks

We compare the performance of the proposed approach against the alternative MIP formulations presented in Section 5.2. We employ data sets with smaller numbers of SKUs for experiments, as these approaches are more computationally expensive than the company's approach. We use 50 & 100 SKUs to test the Alternative Formulation-1 (i.e., 1-norm formulation) and 1000 & 5000 SKUs to test the Alternative Formulation-2 (i.e., ∞-norm formulation). The time limit is set to 20 hours. All experiments are conducted on a high-performance computer with 16 Intel Xeon Gold 6140 cores and 128 GB of RAM. Tables 4 and 5 show the packaging factor values of the box assortments generated by our approach and the respective MIP benchmarks as they covered the respective SKUs, the corresponding run time, the optimality gap reported by the Gurobi solver (for MIP benchmarks), and the difference in solution quality between the respective approaches.

The results from both tables show that the proposed PAAS approach consistently outperforms the MIP formulations concerning running time and solution quality. Formulation-1 (see Table 4) struggles to solve even small instances. For instance, it could not generate five boxes for 50 SKUs in the given time limit, while the proposed PAAS approach takes less than an hour to create a 10% better solution. On the other hand, though Formulation-2 could solve a small instance with 1000 SKUs to optimality, it starts to struggle to solve these problems as the box assortment size increases from 10 to 20. One interesting observation

Table 3 Performance comparison between PAAS and Company's Approach from Section 5.1

			Runtime (hours)		PF			
Assortment Size	Number of SKUs	Number of Box Candidates	PAAS	Gurobi	PAAS	Gurobi	Gap (%)	Difference
10	2000	1342	0.94	8.5	2.014	**2.008**	0.0	-0.29%
10	2500	1808	0.94	15.4	2.066	**2.053**	0.0	-0.62%
10	3000	1872	0.97	8.0	2.085	**2.058**	0.0	-1.29%
10	3500	2432	0.97	20.0	2.124	**2.107**	1.2	-0.80%
10	4000	3096	0.98	20.0	2.168	**2.167**	2.3	-0.05%
10	4500	3868	1.00	20.0	**2.233**	21.420	94.2	859.25%
10	5000	4672	1.03	20.0	**2.227**	23.465	94.7	953.66%
10	5500	5668	1.04	20.0	**2.218**	24.180	94.9	990.17%
10	6000	6776	1.05	20.0	**2.254**	25.773	95.1	1043.43%
10	6500	8080	1.06	20.0	**2.299**	28.741	100.0	1150.15%
10	7000	9412	1.07	20.0	**2.320**	30.382	100.0	1209.57%
10	7500	10772	1.10	-	**2.333**	-	-	-
20	2000	1342	1.91	5.2	1.702	**1.696**	0.0	-0.33%
20	2500	1808	1.91	20.0	1.716	**1.712**	0.9	-0.23%
20	3000	1872	1.98	16.7	1.733	**1.711**	0.0	-1.27%
20	3500	2432	2.01	20.0	1.754	**1.737**	1.4	-0.99%
20	4000	3096	2.06	20.0	1.765	**1.760**	1.8	-0.28%
20	4500	3868	2.12	20.0	**1.788**	11.449	92.7	540.32%
20	5000	4672	2.19	20.0	**1.801**	13.422	93.7	645.25%
20	5500	5668	2.26	20.0	**1.783**	13.870	94.1	677.90%
20	6000	6776	2.28	20.0	**1.806**	14.806	100.0	719.82%
20	6500	8080	2.34	20.0	**1.826**	16.531	100.0	805.31%
20	7000	9412	2.35	20.0	**1.875**	17.495	100.0	833.07%
20	7500	10772	2.44	-	**1.849**	-	-	-
30	2000	1342	2.14	11.1	1.589	**1.582**	0.0	-0.43%
30	2500	1808	2.21	20.0	1.591	**1.585**	0.1	-0.38%
30	3000	1872	2.25	20.0	1.588	**1.582**	0.7	-0.38%
30	3500	2432	2.29	20.0	1.597	**1.593**	1.1	-0.25%
30	4000	3096	2.33	20.0	1.619	**1.624**	3.4	0.33%
30	4500	3868	2.47	20.0	**1.627**	23.162	97.2	1323.60%
30	5000	4672	2.53	20.0	**1.641**	24.936	97.4	1419.56%
30	5500	5668	2.59	20.0	**1.634**	26.775	100.0	1538.62%
30	6000	6776	2.63	20.0	**1.649**	29.446	100.0	1685.69%
30	6500	8080	2.75	20.0	**1.665**	32.599	100.0	1857.90%
30	7000	9412	2.78	20.0	**1.674**	35.207	100.0	2003.17%
30	7500	10772	2.86	-	**1.691**	-	-	-
40	2000	1342	2.30	7.4	1.522	**1.519**	0.0	-0.19%
40	2500	1808	2.37	15.3	1.523	**1.520**	0.0	-0.18%
40	3000	1872	2.43	20.0	1.515	**1.512**	0.5	-0.20%
40	3500	2432	2.50	20.0	1.522	**1.519**	0.8	-0.21%
40	4000	3096	2.61	20.0	1.543	**1.532**	1.3	-0.70%
40	4500	3868	2.69	20.0	**1.552**	6.281	89.9	304.70%
40	5000	4672	2.73	20.0	**1.557**	6.342	90.0	307.32%
40	5500	5668	2.88	20.0	**1.547**	6.615	90.5	327.60%
40	6000	6776	2.83	20.0	**1.570**	7.097	100.0	352.04%
40	6500	8080	2.90	20.0	**1.569**	7.961	100.0	407.37%
40	7000	9412	3.02	20.0	**1.574**	8.455	100.0	437.17%
40	7500	10772	3.09	-	**1.586**	-	-	-

Table 4 Performance comparison between PAAS and Alternative MIP Formulation-1 (Section 5.2)

		Runtime (hours)		PF			
Assortment Size	Number of SKUs	PAAS	Gurobi	PAAS	Gurobi	Gap (%)	Difference
3	50	0.75	0.01	**1.417**	1.550	0.0	8.58%
3	100	0.71	2.97	**1.494**	1.607	0.0	7.03%
4	50	1.09	0.02	**1.310**	1.427	0.0	8.20%
4	100	1.08	20	**1.371**	1.465	23.2	6.42%
5	50	0.80	20	**1.238**	1.377	15.2	10.09%
5	100	0.81	20	**1.304**	1.399	43.5	6.79%

Table 5 Performance comparison between PAAS and Alternative MIP Formulation-2 (Section 5.2)

		Runtime (hours)		PF			
Assortment Size	Number of SKUs	PAAS	Gurobi	PAAS	Gurobi	Gap (%)	Difference
10	1000	1.04	0.62	**1.761**	2.034	0.0	13.42%
10	5000	1.32	20.02	**2.205**	2.945	59.8	25.13%
20	1000	2.13	20.00	**1.460**	1.749	7.8	16.52%
20	5000	2.91	20.01	**1.751**	2.330	77.5	24.85%
30	1000	2.43	20.00	**1.342**	1.521	44.1	11.77%
30	5000	3.47	20.01	**1.572**	2.098	71.2	25.07%

regarding Formulation-2 is that the solutions it generated for instances with 5000 SKUs have significantly better quality than the solutions generated by the Company's MIP approach. On the other hand, the PAAS approach solves all the instances in less than 4 hours, with solutions that are more than 10% better. These experiments show that the alternative MIP approaches are not suitable for the problem at hand, and the PAAS approach is superior in running time and solution quality.

6.5 Choice of Stage-1 Method

In this section, we compare the performance of our stage-1 method, k-means, with existing clustering methods that can be employed to solve the box-sizing problem. The experiments in this section show whether the available methods can serve as an alternative to k-means. As discussed in Section 4.1, the box-sizing problem can be transformed into a clustering problem, where various SKUs can be grouped into clusters, and boxes can be designed to cover the SKUs of each cluster. However, only a few clustering methods can be applied to the problem. This is because of two essential requirements. First, we must assign all SKUs to resultant clusters to account for their dimensions during box generation. Second, we must generate a fixed set of clusters to control the number of boxes generated. Therefore, many popular clustering methods, such as affinity propagation, DBSCAN, mean shift, and their derivatives, cannot be used since they do not assign

Table 6 Performance comparison between *k*-means, agglomerative, and spectral clustering methods

		Memory (MB)			PF		
Assortment Size	Number of SKUs	*k*-means	Agglomerative	Spectral	*k*-means	Agglomerative	Spectral
30	1000	0.01	1.35	71.21	1.798	1.884	1.839
30	5000	0.05	190.99	782.48	1.848	1.840	1.959
30	10000	0.39	763.18	3179.29	1.841	1.995	1.940
30	15000	1.15	1716.67	7244.54	1.794	1.861	1.939
30	20000	1.55	3052.22	11933.25	1.767	1.807	1.894
..	..	..	..	..	..	..	..
30	150000	27.00	87900.00	176000.00	2.352	-	-

all SKUs or generate a fixed set of clusters. This leaves us with popular alternatives such as agglomerative and spectral clustering methods. In this section, we compare the performance of our stage-1 algorithm *k*-means with both methods.

In the experiments, we randomly sample a set of SKUs: 5000, 10000, 15000, and 20000, and generate a 30-box assortment using *k*-means, agglomerative & spectral clustering methods. We report the packaging factor values of the box assortments generated by each method, along with the computational memory required to run each of these approaches in Table 6. We employ the standard library (*scikit-learn*) implementations of these approaches. The results from Table 6 show that the solutions generated by *k*-means are either similar or better than those generated by agglomerative and spectral clustering methods. In addition, notice the memory requirements of different approaches. Both agglomerative and spectral clustering methods consume large amounts of memory. They simply do not scale; as the number of SKUs increases, the memory requirements increase quadratically. In fact, we could not run the spectral clustering approach on 150,000 SKUs which is a standard problem size our approach is designed to operate on. For the problem in question, spectral clustering requested 176 GB of RAM; conversely, agglomerative clustering needed 87.9 GB, and *k*-means required 27 MB. Therefore, considering the scalability, *k*-means is the most suitable choice for our stage 1.

7 Organizational Impact

We presented the proposed approach to our industrial partner and shared the software for utilizing the optimization framework. The company received the research very well; all the stakeholders in the company, including the senior leadership, analytics managers, and warehouse managers found the software (framework) and analysis helpful. Currently, the software is deployed at the company. Section 7.1 describes the organizational impact. A senior vice president commented:

This team has come up with a novel solution for packaging optimization. It is a very complex problem at the scale we operate in our industry. At our company, we are always trying to push the boundaries of

innovation, partnering with leading academic institutions helps us with this endeavor. The aim is to build practical applications that can create the right impact in our key performance metrics. The solution has been readily and eagerly adopted by the packaging design team at our company to improve their operations. It has helped them realize substantial benefits in packaging.

A senior director commented: *Packaging is a critical component of e-commerce order fulfillment. It has a direct impact on customer experience and supply chain costs. Because of the huge quantum of SKUs sold through our platform, designing optimal packaging for our products is an extremely challenging problem. Solving this problem streamlines our supply chain starting with the procurement of packaging boxes. Efficient packaging reduces the air shipped. Collaboration with the team has helped us arrive at an innovative and practical solution. This project has a massive impact on how we refine our packaging at regular intervals.*

One analytics manager commented: *This optimization framework shows us an alternative way of solving the packaging problem. The current approach is very compute-heavy and challenging to run. We see your approach as feasible in terms of running. It generates the packaging boxes in one go for all the SKUs quickly; that itself is a pretty good tradeoff. I am fairly comfortable with the idea here; it is quite an elegant solution. One more benefit of this framework is that we can see the boxes changing; we can easily retain, and alter the boxes and run multiple test conditions as required. Also, the implementation is pretty straightforward; based on our initial runs, we do not see any issues in its usage. We are definitely interested in leveraging this for other domains.*

7.1 Implementation Details & Quantified Impact

The distributed software requires two mandatory inputs: the dataset consisting of SKU dimensions and the number of boxes (K) the user wants to use to cover the input set of SKUs. The software then generates K box sizes using the framework described in this paper. Section EC.13 in the online supplement discusses the software. The company used this framework on around 200,000 SKUs and generated box assortments of sizes between 15 and 50. Based on the business analysis, they have chosen the 35-box assortment that has a PF of 1.91 for the considered set of SKUs. Refer to EC.14 in the online companion for the analysis that compares PF vs. the size of assortment (K). The boxes with generated dimensions were then procured from box suppliers and deployed in one fulfillment center. Finally, after extensive pilot studies that accounted for PF improvement, packer ease of use, and durability, the newly designed 35-box assortment is deployed across 28 fulfillment centers. With the ML-based optimization framework, the company refreshed its entire box assortment. The newer assortment is smaller than the previous one and also increased space efficiency. The annual savings with better packaging design are derived from multiple sources (as shared by the company).

1. *Material cost savings:* The newer assortment is more compact than the previous assortment. This resulted in an average weight reduction of 13% per carton (Δ_w). The weight of a carton contributes to 52%

of costs (W_c). On average, a carton costs 0.085 USD (C_c). The company delivers approximately 350 million orders in cartons every year, therefore, yearly savings are: $350 \times 10^6 \times \Delta_w \times W_c \times C_c \approx 2$ million USD.

2. *Transportation cost savings:* The newer assortment reduced the average shipment volume from 383 cubic inches to 264 cubic inches, resulting in a 31% reduction in space utilization (Δ_t). The average shipment transportation costs are 0.045 USD (C_t). The yearly savings for processing 350 million orders are: $350 \times 10^6 \times \Delta_t \times C_t \approx 4.9$ million USD.

3. *Packing time savings:* The newer assortment is smaller than the previous assortment which improved the search and packing efficiency. The packing time is reduced by 2 sec (Δ_p). The warehouse labor cost per second is 0.0003 USD (C_p). The yearly savings for processing 350 million orders are: $350 \times 10^6 \times \Delta_p \times C_p \approx$ 0.2 million USD.

4. *Raw paper savings:* The newer assortment is compact, this reduced the box weight by 5.945 gm on average. For processing 350 million shipments, the total paper savings per year are $350 \times 10^6 \times 5.945gm \approx$ 2080 metric tons.

5. *Emission reductions:* The newer assortment increased space efficiency and, thereby, the loadability of trucks. The number of trucks required to carry the load has decreased. In total, 14 million km of line-haul mileage is saved per year. Each km produces an average of 140 gm CO_2 emissions. The yearly savings are $14 \times 10^6 \times 140gm \approx 1960$ metric ton CO_2 equivalent.

8 Conclusion

This article considers the e-commerce box-sizing problem where a set of packaging boxes that cover the given set of SKUs are to be determined. We take a machine learning perspective to solve the problem. In particular, we propose an optimization framework combining unsupervised learning, reinforcement learning, and tree-search to solve the problem. Specifically, the box sizes are generated in three steps. First, a data partitioning problem is formulated, and the k-means clustering algorithm is used to create an initial solution. Subsequently, a sequential decision-making task called the box-sizing game is designed, and a neural network agent is trained to learn an improvement heuristic in the second step. Finally, the learned neural networks are integrated into a tree-search procedure to synthesize the final solution from the initial solution. We perform computational experiments using real-world data and establish the superiority of the approach.

We make several observations from this study. Firstly, we find that the proposed box-sizing game enables neural network policies to learn the improvement heuristic. The policies learned to take improving moves along with the termination condition consistently across all the assortment sizes on real-world and synthetic datasets. This observation indicates that the proposed box-sizing game is an effective formulation of the optimization problem, and neural networks can indeed be employed to learn the heuristics. Secondly, we observe that the learned heuristics are extremely fast to execute. In our experiments, the learned heuristic

takes milliseconds to generate the solution. This speedy execution is primarily due to avoidance of the objective function computation before selecting transformations. Finally, we observe that the performance of the learned heuristic can be significantly improved by embedding it in a tree-search procedure. The tree-search acts as a policy improvement operator and improves policy decisions, resulting in high-quality solutions. In our experiments across multiple assortment sizes with real-world datasets, the tree-search integrated with network policies consistently showed better results. This observation suggests that the performance of the policies generated through reinforcement learning can be further improved by designing and integrating suitable policy improvement operators.

8.1 Implications to Research & Practice

The study has implications for research. Firstly, it contributes to the operations management literature by proposing a scalable and effective optimization framework for solving the e-commerce box-sizing problem. It introduces a novel perspective of solving the respective optimization problem through machine learning and contrasts the characteristics with standard approaches. Secondly, it contributes to the reinforcement learning literature by demonstrating a real-world application in the domain of warehouse operations. More importantly, it shows the synergies between different machine learning paradigms and the benefits of their integration specifically for solving optimization problems. Section 8.2 discusses how the proposed framework generalizes to other OR problems.

The study informs industrial practice in several ways. First, the optimization framework is scalable and practical, and it can be directly applied to design box sizes for e-commerce warehouses. Note that the framework is not constrained to any specific e-commerce platform and, therefore, can be employed by any company. Our approach is appreciated by the industrial partner, and the implementation results from the field are encouraging. Second, the learned heuristics can replace the standard approaches when combined with improvement operators, reducing the laborious human effort to develop and tune algorithms for specific problem instances.

8.2 Generalization to other Operational Problems

The proposed optimization framework can be generalized to transportation and assignment problems in the OR literature. As described in the article, the framework generates an initial solution in the first stage, learns rules to improve the solution in the second stage, and finally applies the learned rules in the third stage. In the context of transportation and assignment problems, stage-1 can be replaced by heuristics to generate initial solutions for the respective problems. For instance, Vogel's or Hungarian methods can be used in stage-1 based on the problem. In the second stage, actions can be defined in line with the type of solution required. We defined actions that either increment or decrement the entries of a solution by 0.5 inches for the box-sizing problem. Transportation problems can use the same action scheme with a larger or smaller

step size. However, for assignment problems, actions can be defined to enter either 0 or 1 in the solution matrix. The reward function should be carefully modified to consider all constraints and to give positive and negative rewards. Once a policy is generated for these problems, the third stage can be directly applied.

Acknowledgements

This work was supported by the Indo-Dutch Grant [grant number 13 (4)/2018-CC&BT] provided by the Ministry of Electronics & IT (MeitY), India, in collaboration with the Netherlands Organization for Scientific Research (NWO). We also thank the e-commerce company for their support throughout this research.

References

Aggarwal, Charu C., Chandan K. Reddy. 2013. *Data Clustering: Algorithms and Applications*. 1st ed. Chapman & Hall/CRC, Boca Raton.

Alvarez, Alejandro Marcos, Quentin Louveaux, Louis Wehenkel. 2017. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29 (1), 185-195.

Ampuja, Jack. 2015. Optimize packaging and save money. URL `https://www.logisticsmgmt.com/article/dimensional_weight_optimize_packaging`. (Accessed 19 September 2021).

Azzi, A., D. Battini, A. Persona, F. Sgarbossa. 2012. Packaging design: General framework and research agenda. *Packaging Technology and Science*, 25 (8), 435-456.

Bansal, Vishal, Debjit Roy, Jennifer A Pazour. 2021. Performance analysis of batching decisions in waveless order release environments for e-commerce stock-to-picker order fulfillment. *International Transactions in Operational Research*, 28 (4), 1787-1820.

Bello, Irwan, Hieu Pham, Quoc V Le, Mohammad Norouzi, Samy Bengio. 2017. Neural combinatorial optimization with reinforcement learning. *Workshop Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Bengio, Yoshua, Andrea Lodi, Antoine Prouvost. 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290 (2), 405-421.

Brinker, Jan, Halil Ibrahim Gündüz. 2016. Optimization of demand-related packaging sizes using a p-median approach. *The International Journal of Advanced Manufacturing Technology*, 87 (5-8), 2259-2268.

Chen, Minmin, Omer Gottesman, Lihong Li, Yuxi Li, Zongqing Lu, Rupam Mahmood, Niranjani Prasad, Zhiwei Qin, Csaba Szepesvari, Mathew E. Taylor. 2021. Reinforcement Learning for Real Life. URL `https://sites.google.com/view/RL4RealLife`. (Accessed 03 August 2022).

Collis, David, Andy Wu, Rembrand Koning, Huaiyi CiCi Sun. 2018. Walmart Inc. takes on Amazon.com. *Harvard Business School Case*, 718 481-513.

Cramer, Ethan. 2023. Worldwide Ecommerce Forecast 2023. URL `https://www.insiderintelligence.com/content/worldwide-ecommerce-forecast-2023`. (Accessed 25 October 2023).

Dai, Hanjun, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17, Curran Associates Inc., 6351-6361.

Junyoung Park, Sang Hun Kim Youngkook Kim, Jaehyeong Chun, Jinkyoo Park. 2021. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59 (11), 3360-3377.

Leung, S.Y.S., W.K. Wong, P.Y. Mok. 2008. Multiple-objective genetic optimization of the spatial design for packing and distribution carton boxes. *Computers Industrial Engineering*, 54 (4), 889-902.

Mazyavkina, Nina, Sergey Sviridov, Sergei Ivanov, Evgeny Burnaev. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers Operations Research*, 134 105400.

Nazari, Mohammadreza, Afshin Oroojlooy, Martin Takáč, Lawrence V. Snyder. 2018. Reinforcement learning for solving the vehicle routing problem. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18, Curran Associates Inc., Red Hook, NY, USA, 9861–9871.

O'Neill, Sean. 2022. How Amazon learned to cut its cardboard waste. URL `https://www.amazon.science/latest-news/amazon-cardboard-boxes-waste-reduction/`. (Accessed 25 July 2022).

OpenAI. 2018. Proximal Policy Optimization. URL `https://spinningup.openai.com/en/latest/algorithms/ppo.html`. (Accessed 12 August 2022).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. 2017. Proximal policy optimization algorithms. URL `https://arxiv.org/abs/1707.06347`. (Accessed 03 August 2023).

Shih Jia Lee, Loo Hay Lee, Ek Peng Chew, Julius Thio. 2015. A study on crate sizing problems. *International Journal of Production Research*, 53 (11), 3341-3353.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362 (6419), 1140-1144.

Sionek, André (Olist). 2018. Brazilian E-Commerce Public Dataset by Olist. URL `https://www.kaggle.com/dsv/195341`. (Accessed 02 May 2022).

Sutton, Richard S., Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. 2nd ed. A Bradford Book, Cambridge, MA, USA.

Vinyals, Oriol, Meire Fortunato, Navdeep Jaitly. 2015. Pointer networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'15, MIT Press, Cambridge, MA, USA, 2692–2700.

Xu, Jing, Hu Qin, Rendao Shen, Chenghao Shen. 2008. An optimization framework for the box sizing problem. *2008 IEEE International Conference on Service Operations and Logistics, and Informatics*, vol. 2. 2872-2877.

Yang, Cenying, Yihao Feng, Andrew Whinston. 2022. Dynamic Pricing and Information Disclosure for Fresh Produce: An Artificial Intelligence Approach. *Production and Operations Management*, 31 (1), 155-171.

Yang, Guang, Cun (Matthew) Mu. 2020. A machine learning approach to shipping box design. Yaxin Bi, Rahul Bhatia, Supriya Kapoor, eds., *Intelligent Systems and Applications*. Springer International Publishing, Cham, 402-407.

Yueyi, Li, Zhang Xiaodong, Wang Pei. 2020. A cost-minimization model to optimal packaging size in e-commerce context. *Proceedings of the 2019 Annual Meeting on Management Engineering*. AMME 2019, Association for Computing Machinery, New York, NY, USA, 35–41.