

## E-Companion

This document contains the supplementary material for the manuscript titled “A Machine Learning Approach to Solve the E-commerce Box-Sizing Problem.” The contents provide a complementary view when read with the original manuscript.

### EC.1 Comparison of the prior approaches

**Table EC.1** Comparison of the prior approaches

Criterion	Approaches					
	Leung et al. (2008)	Xu et al. (2008)	Shih Jia Lee and Thio (2015)	Brinker and Gündüz (2016)	Yueyi et al. (2020)	Our Approach
Context	B2B	B2B	B2B	B2C	B2C	B2C
No. of Dimensions	3	3	1	3	3	3
Requires Candidates	Yes	Yes	No	Yes	Yes	No
Number of Candidates Considered	10	15	Not Applicable	2040	10	Not Applicable
Number of SKUs Considered	3	500	80	4,443	500	200,000
Optimization Criterion	Space Utilization	Space Utilization	Space Utilization	Space Utilization	Total Costs	Space Utilization
Optimization Framework	Genetic Algorithm	Mathematical Programming	Mathematical Programming	Mathematical Programming	Mathematical Programming	Machine Learning

## EC.2 Notations

**Table EC.2** Notations table

Notation	Meaning
$n$	Number of SKUs.
$D$	Set of SKUs.
$(l_i, b_i, h_i, d_i)$	A tuple representing the length, breadth, height, and expected demand of $i^{th}$ SKU.
$P$	Set of packaging boxes.
$(L_j, B_j, H_j)$	A tuple representing the length, breadth, height of the $j^{th}$ packaging box.
$K$	Number of Boxes ( $ P $ ).
$x_{ij}$	Binary decision variable that indicates whether SKU $i$ is assigned to box $j$ .
$s_0$	A set of $K$ packaging boxes represented as a matrix. Generated by the first stage as initial solution.
$\pi, \pi_\theta, \pi_\theta(a s)$	Neural Network model or policy parameterized by $\theta$ .
$k$	Number of partitions of the SKU data space ( $= K$ )
$t$	Time step in the agent game interaction process.
$s_t$	A set of $K$ packaging boxes represented as a matrix, generated at the $t_{th}$ time step.
$a_t$	Action taken by the agent at time step $t$ .
$P(s_{t+1} s_t, a_t)$	Probability of reaching the next state, $s_{t+1}$ from the current state $s_t$ when the agent takes action $a_t$ .
$r_{t+1} = r(s_t, a_t, s_{t+1})$	Reward function that assigns a scalar value for generating $s_{t+1}$ from $s_t$ , by taking action $a_t$ .
$PF(s_t)$	Packaging factor of a solution $s_t$ .
$\epsilon$	Clipping value for PPO.
$A^{\pi_\theta}(s, a)$	Advantage estimate of taking action $a$ from $s$ from state $s$ under the policy $\pi_\theta$ .
$\beta$	User-defined parameter to specify weightage in the PAAS method.
$c_{ij}$	Cost of placing SKU $i$ in box $j$ .
$y_j$	Indicator variable specifying whether box $j$ is selected.
$\delta_{ij}^l, \delta_j^b, \delta_{ij}^h$	Difference between length, breadth, height of the SKU $i$ and box $j$

## EC.3 Neural network architecture

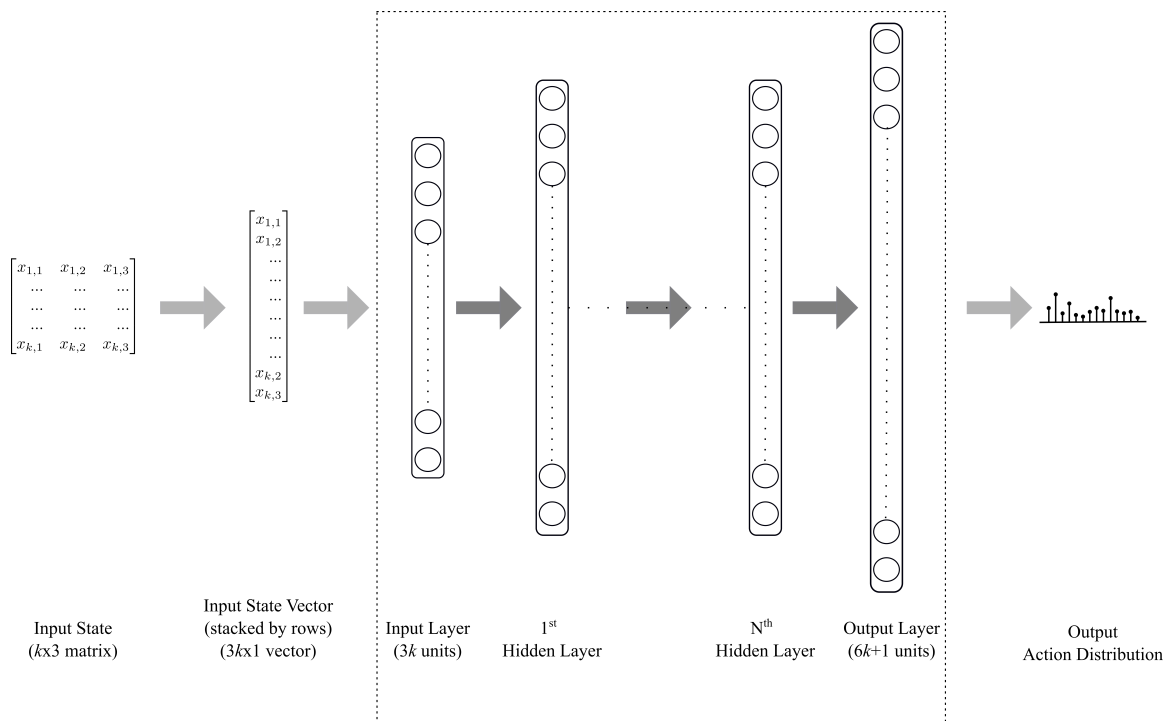


Figure EC.1 Policy Network Architecture

## EC.4 Algorithms

---

**Algorithm 1:** Algorithm to generate an initial solution

---

**Input :**  $K, D$

**Output:**  $P$

$P \leftarrow \{\};$

Represent each SKU as a point in 3D space with *length*, *breadth*, & *height* dimensions ;

Apply  $k$ -means algorithm to generate  $K$  partitions ;

*/\* for each partition, find the maximal SKU \*/;*

**for**  $i=1,2,3\dots K$  **do**

$(l_{max}, b_{max}, h_{max}) \leftarrow ((\max(\{l_u\}_{u=1}^v)), (\max(\{b_u\}_{u=1}^v)), (\max(\{h_u\}_{u=1}^v)))$  ;

$P \leftarrow P \cup (l_{max}, b_{max}, h_{max})$

**end**

---

**Algorithm 2:** Learning algorithm for training the neural networks**Input** :  $\theta_0, \phi_0, z, \epsilon, total\_steps$ **Output:**  $\theta_{l+1}, \phi_{l+1}$  $step\_count \leftarrow 0; B \leftarrow 0; l \leftarrow 0;$ **while**  $step\_count < total\_steps$  **do**    reset game environment and read the initial game state  $s$  ;    **for**  $step = 1, 2, 3, \dots, z$  **do**        sample action  $a$  from the distribution  $\pi_{\theta_l}(a|s)$  ;        pass action  $a$  to the environment, observe the next state  $s'$  and reward  $r$  ;        push  $(s, a, r, s')$  into  $B$ , increment  $step\_count$  ;        **if** *game has ended* **then**            reset game environment and read the initial game state  $s$  ;        **else**             $s \leftarrow s'$  ;        **end**    **end**    retrieve rewards and calculate advantage estimates from  $B$  ;    calculate  $\theta_{l+1}$  maximizing the PPO objective via Adam ;    calculate  $\phi_{l+1}$  minimizing value network MSE loss via Adam ;     $l \leftarrow l + 1$  ;    clear  $B$  ;**end****Algorithm 3:** Policy assisted active search algorithm (PAAS)**Input** :  $s_0, \pi_{\theta_{l+1}}, \beta, n\_rollouts, n\_steps, n\_iter$ **Output:**  $s^*$  $s \leftarrow s_0; f \leftarrow PF(s_0)$  ; $s^* \leftarrow s; f^* \leftarrow f$  ;**for**  $i=1, 2, \dots, n\_iter$  **do**    calculate  $WPF$ s from all child nodes using  $\beta, \pi_{\theta_{l+1}}$ , for  $n\_payouts$  with  $n\_steps$  ;    observe the child node with the lowest  $WPF$ , say  $s'$  ;    **if**  $PF(s') < f^*$  **then**         $s^* \leftarrow s'; f^* \leftarrow PF(s')$  ;    **end**     $s' \leftarrow s$ **end**

## EC.5 Parameters for training the neural networks

The experiments are conducted on an HPC instance with 24 (Intel Xeon 6140) cores and 192GB RAM. Note that the training time majorly depends on the generation of samples from the designed RL environment. Our environment can generate 500 to 2500 samples per second, and our training requires 5 million to 10 million samples based on the number of SKUs and assortment size. This translates to a training time of 34 minutes on a 2000 SKU, 10-box assortment problem. On the other hand, the training time reaches a little less than 9 hours on 150K SKU, 40-box assortment problem. We also found that the training times are roughly the same on the M1 architecture MacBook Pro laptops. Table EC.3 shows the set of parameters that we found ideal for all the problem and SKU sizes for training the neural networks. In the PAAS method, we use 20 rollouts while looking 50 steps ahead with  $\beta = 0.99$ ,  $n\_iter = 3000$ .

**Table EC.3** Parameters

Assortment size ( $K$ )	Parameters	Policy Network	Value Network
10	$\gamma = 1, lr = 10^{-4}, z = 4096, bs = 512,$ $\epsilon = 0.05, activation = \tanh,$ $total\_steps = 5 \times 10^6$	30, 72, 72, 61	30, 72, 72, 1
20	$\gamma = 1, lr = 10^{-4}, z = 6144, bs = 512,$ $\epsilon = 0.05, activation = \tanh,$ $total\_steps = 5 \times 10^6$	60, 120, 120, 121	60, 120, 120, 1
30	$\gamma = 1, lr = 10^{-4}, z = 8192, bs = 512,$ $\epsilon = 0.05, activation = \tanh,$ $total\_steps = 5 \times 10^6$	90, 196, 144, 108, 181	90, 196, 144, 108, 1
40	$\gamma = 1, lr = 10^{-4}, z = 10240, bs = 512,$ $\epsilon = 0.045, activation = \tanh,$ $total\_steps = 10 \times 10^6$	120, 272, 232, 196, 241	120, 272, 232, 196, 1

## EC.6 Advantage Estimation

The clipped PPO objective is scaled by  $A^{\pi_{\theta_l}}(s, a)$  in Equation 4. More precisely,  $A^{\pi_{\theta_l}}(s_t, a_t)$  is the advantage estimate of taking action  $a_t$  at state  $s_t$ . It is calculated using the Equation (EC.1).

$$A^{\pi_{\theta_l}}(s_t, a_t) = r(s_t, a_t, s_{t+1}) + \gamma V^{\pi_{\theta_l}}(s_{t+1}) - V^{\pi_{\theta_l}}(s_t) \quad (\text{EC.1})$$

Here,  $r(s_t, a_t, s_{t+1})$  is the reward observed from the environment when action  $a_t$  is taken from  $s_t$ . These values are observed during the agent environment interaction process. The discount factor,  $\gamma$ , adjusts the influence of future rewards on current estimates. It takes a value between 0 to 1. A lower value makes the estimate short-sighted as it reduces weight on future rewards, while a larger value makes the estimates far-sighted as they account for future rewards that can arise from the next state  $s_{t+1}$ .  $V^{\pi_{\theta_l}}(s_t)$  is the expected

return (cumulative sum of rewards) that can be yielded by following the policy  $\pi_{\theta_t}$  from  $s_t$ . Similarly,  $V^{\pi_{\theta_t}}(s_{t+1})$  is the expected return from the next state,  $s_{t+1}$ . A neural network with a similar architecture as the policy network (EC.3) is used to estimate the values. As shown in Algorithm-2 (EC.4), the network is trained along with the policy network to reduce the mean squared loss between predicted values and value targets. The value targets are computed using the expression:  $r(s_t, a_t, s_{t+1}) + \gamma V^{\pi_{\theta_t}}(s_{t+1})$ . For further details, the reader can refer to Schulman et al. (2017).

## EC.7 Illustration of the three-stage framework

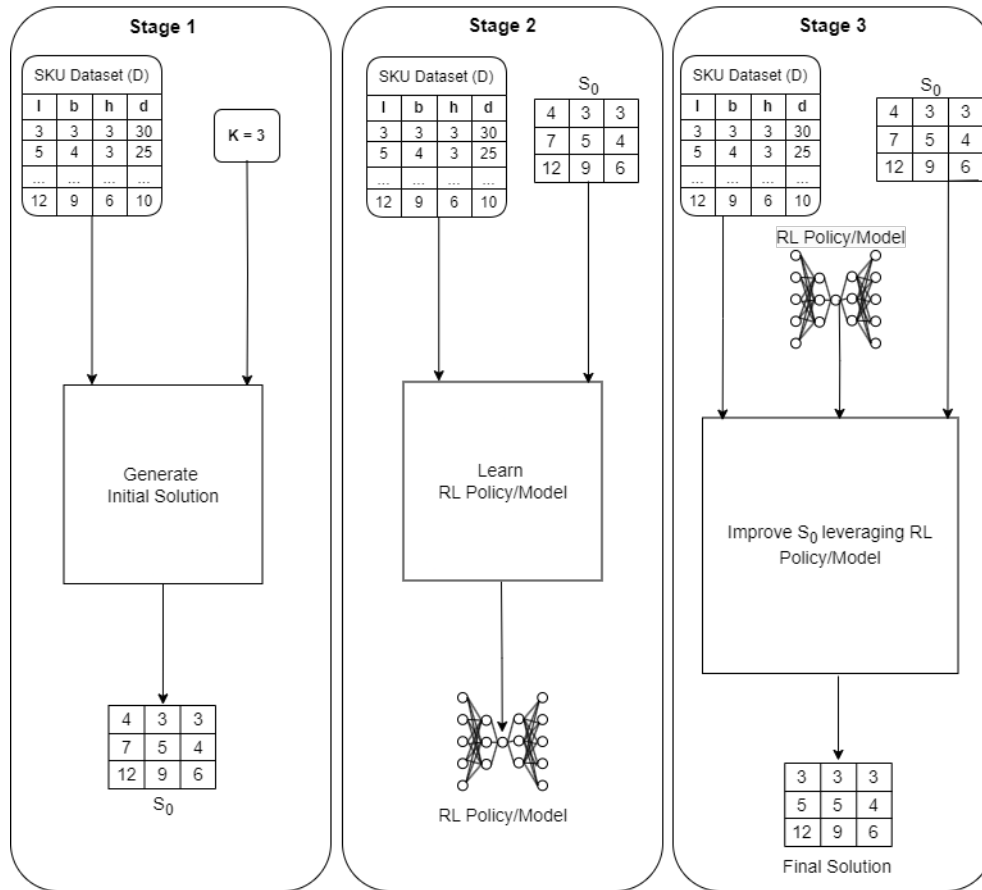
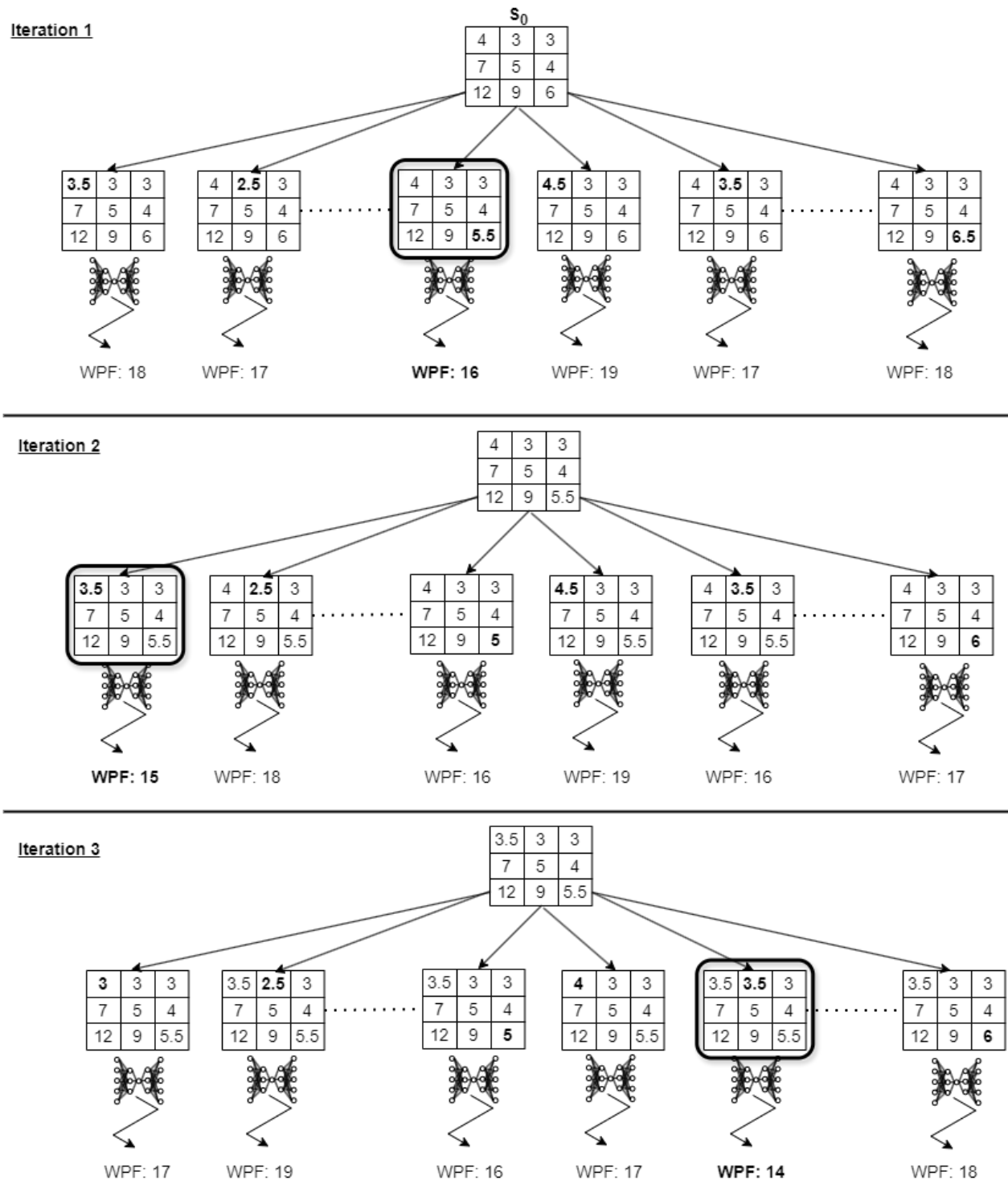


Figure EC.2 Three Stage Illustration

In this section, we illustrate the three-stage optimization framework. Figure EC.2 shows the inputs and outputs of the respective stages. The inputs to stage 1 are the SKU dataset,  $D$ , and the required number of boxes,  $K$ . As shown in the figure, the SKU dataset consists of the length, breadth, height, and demand of the SKUs that must be covered. Using these inputs, the stage-1 generates an initial solution  $s_0$ . Since  $K = 3$  in the illustrated figure, we can observe a  $3 \times 3$  matrix representing the initial solution. The initial solution,  $s_0$ , and the SKU dataset are passed to stage 2 for learning an improvement policy through RL. The output of the



**Figure EC.3** Iterative demonstration

second stage is a neural network policy that can improve the generated initial solution  $s_0$ . We can directly apply the learned policy and generate a final solution; however, by searching as described in stage 3, the performance of the final solution can be improved significantly. This is because stage 3 serves as a policy

improvement operator and improves the learned policy, thereby generating a better final solution. The stage-3 inputs are the SKU dataset  $D$ , initial solution  $s_0$ , and the learned RL policy  $\pi$ . Using these inputs, stage 3 generates a final solution  $s^*$ . Figure EC.3 illustrates a few iterations showing how  $s_0$  is transformed into  $s^*$  in stage 3. In the first iteration, the procedure starts the search at  $s_0$ , expands all the child nodes, and, from each child node, employs the neural network policy to simulate the best possible future solutions. Based on the PFs of the future solutions, WPF scores are assigned to the child nodes in consideration. Once all child nodes are assigned with WPF scores, the child node with the lowest WPF is selected. In the next iteration, the search is again started from the selected child node. This process is repeated until the computational budget is exhausted and the child node that has the lowest PF encountered in the search path is returned as the final solution.

## EC.8 Technical Discussion of Formulations

In this section, we briefly discuss some technical aspects of the MIP formulations presented in Sections 5.1 and 5.2, including how they relate to the problem we are attempting to solve. We omitted this discussion from the main body of the paper in order to avoid disruption of the flow and practical focus of the paper, even though this may be of interest to some readers.

Firstly, we discuss the MIP used by the company as presented in Section 5.1. We observe that the size of the set  $P$  primarily depends on the discretization interval used (which is under our control) and the maximum and minimum dimensions for length, breadth and height of the SKUs. Let  $l_{min}, b_{min}, h_{min}$  be the minimum dimensions for SKUs (in a similar fashion to  $l_{max}, b_{max}, h_{max}$ ) and  $\epsilon > 0$  be the size of the discretization interval used, then the size of  $P$  is simply

$$\frac{l_{max} - l_{min}}{\epsilon} \frac{b_{max} - b_{min}}{\epsilon} \frac{h_{max} - h_{min}}{\epsilon}$$

If we let  $\epsilon > 0$  to be finite but sufficiently small, one may argue that this is the ultimate problem to be solved, with the disadvantage of  $|P|$  becoming extremely large (but still finite). We refer to this problem as the *ideal problem*, denoted by *IDE*. In order to have a manageable problem to solve (notwithstanding the simple practicality of packaging sizes not being able to be very finely cut), the company's approach would dictate the use of a discretization interval, say  $\delta$ , which is significantly larger than  $\epsilon$ , hence only providing an approximation to the ideal problem (but with a much more manageable size of  $P$ ). We note the following result for this approximation, which we refer to as the *company's problem*, denoted by *COM*.

PROPOSITION EC.1. *Let  $OPT$  be the value of the optimal solution for the ideal problem. Then, the company's problem provides a solution with a value at most*

$$OPT \times \left( 1 + \delta \left( \frac{1}{h_{min}} + \frac{1}{b_{min}} + \frac{1}{l_{min}} \right) + \delta^2 \left( \frac{1}{b_{min}h_{min}} + \frac{1}{l_{min}h_{min}} + \frac{1}{l_{min}b_{min}} \right) + \frac{\delta^3}{l_{min}b_{min}h_{min}} \right)$$



We note that this approximation ratio can be adjusted by the user-controlled  $\delta$  as needed. For example, even if  $\delta$  is set to a rather high value of at most half of any of the minimum dimensions, this provides a guarantee of 2.375 times  $OPT$ . Next, we present the proof for this result.

PROOF. Using conventional notation, let  $*$  denote an optimal solution. Then, let  $(L_{ij}^*, B_{ij}^*, H_{ij}^*)$  indicate the optimal length, breadth and height for active  $(i, j)$  pairs (i.e.,  $x_{ij}^* = 1$ ) in the optimal solution of IDE. Since COM uses a less granular grid with intervals of  $\delta$ , at the worst case, there will be a feasible solution available at  $(L_{ij}^* + \delta, B_{ij}^* + \delta, H_{ij}^* + \delta)$  for each SKU  $i$ . It is easy to observe that

$$OPT = \sum_{i \in D} d_i L_{ij}^* B_{ij}^* H_{ij}^*$$

On the other hand, the objective function value of COM, denoted by  $\overline{OPT}$ , will be at worst:

$$\begin{aligned} \overline{OPT} &\leq \sum_{i \in D} d_i (L_{ij}^* + \delta)(B_{ij}^* + \delta)(H_{ij}^* + \delta) \\ &= OPT + \sum_{i \in D} d_i (\delta(L_{ij}^* B_{ij}^* + L_{ij}^* H_{ij}^* + B_{ij}^* H_{ij}^*) + \delta^2(L_{ij}^* + B_{ij}^* + H_{ij}^*) + \delta^3) \\ &= OPT + \sum_{i \in D} d_i L_{ij}^* B_{ij}^* H_{ij}^* \left( \delta \left( \frac{1}{H_{ij}^*} + \frac{1}{B_{ij}^*} + \frac{1}{L_{ij}^*} \right) + \delta^2 \left( \frac{1}{B_{ij}^* H_{ij}^*} + \frac{1}{L_{ij}^* H_{ij}^*} + \frac{1}{L_{ij}^* B_{ij}^*} \right) + \frac{\delta^3}{L_{ij}^* B_{ij}^* H_{ij}^*} \right) \\ &\leq OPT + OPT \left( \delta \left( \frac{1}{h_{\min}} + \frac{1}{b_{\min}} + \frac{1}{l_{\min}} \right) + \delta^2 \left( \frac{1}{b_{\min} h_{\min}} + \frac{1}{l_{\min} h_{\min}} + \frac{1}{l_{\min} b_{\min}} \right) + \frac{\delta^3}{l_{\min} b_{\min} h_{\min}} \right) \end{aligned}$$

The first equation is simply polynomial expansion, the second equation rewrites the expression, and the final inequality follows the simple fact that each of the dimensions is bigger than or equal to the minimum dimensions.  $\square$

Finally, it is also important to remark that IDE and COM are pure binary integer programs, and their objective functions explicitly minimize  $PF$ , which is a key decision criteria for the company, despite not considering a continuous range of values for box dimensions. As briefly discussed earlier in the paper, there are various practical strategies such as preprocessing that enable an effective discretization and hence computational efficiency.

Next, we briefly discuss the alternative MIP formulations of Section 5.2, particularly focusing on the first formulation based on using 1-norm, which we will refer to as  $ALT1$ . Despite considering a continuous range of values for box dimensions,  $ALT1$  does not explicitly minimize  $PF$  unlike IDE or COM, as this would require a highly nonlinear objective function due to the variable nature of box dimensions. We first remark the following rather trivial result.

COROLLARY EC.1.  $ALT1$  is equivalent to solving the following problem, which we refer to as  $\overline{ALT1}$ :

$$\min \left\{ \sum_{i \in D, j \in P} d_i (L_j + B_j + H_j) x_{ij} \mid (14) - (16), (20) \right\}$$

$\overline{ALT1}$  can be easily obtained by substituting equations of the form  $\delta_{ij}^l = (L_j - l_i)x_{ij}$  (and likewise for breadth and height) to ALT1 in order to drop the constraints (26)-(29), and then by removing the constant part of the objective function. The proof of this result is straightforward, as one can easily check that a solution valid for one problem is also valid for the other, and vice versa. Finally, we also note that despite being nonlinear, this problem can be easily linearized due to its bilinear nature, simply by defining a new variable  $Z_{ij}$  to replace  $(L_j + B_j + H_j)x_{ij}$  in the objective function, and by adding new constraints of the form  $Z_{ij} \geq (L_j + B_j + H_j) + (x_{ij} - 1)(l_{max} + b_{max} + h_{max})$  to the problem.

We observe that if one changes the objective function of  $\overline{ALT1}$  to the nonlinear expression  $\sum_{i \in D, j \in P} d_i(L_j B_j H_j)x_{ij}$ , the resulting MINLP will not only explicitly minimize  $PF$  (and hence be an ideal problem itself), but will also have the closest relationship to the IDE as  $\delta$  approaches zero. However, in computational terms, this MINLP will be extremely challenging to solve, even for very small problem sizes. Therefore, the linear problem considered in ALT1 (or  $\overline{ALT1}$ ) will provide practically useful alternatives. Finally, we remark that the second alternative MIP formulation of Section 5.2, which is based on using inf-norm, does not provide any useful theoretical bound or guarantee for ALT1, despite having the same solution space as ALT1. However, as observed in computational results in Section 6.4, this formulation is significantly easier to solve than ALT1 and provides competitive solutions in practice.

## EC.9 Problem instance details

**Table EC.4** Problem instance specifications

Dataset	Number of SKUs	Maximum Length	Maximum Breadth	Maximum Height	Number of Box Candidates
D11_2000	2000	11.5	11.0	10.2	1342
D12_2500	2500	12.5	12.0	11.9	1808
D13_3000	3000	13.0	13.0	12.2	1872
D14_3500	3500	14.0	13.4	12.6	2432
D15_4000	4000	15.0	14.5	13.6	3096
D15_4500	4500	15.5	15.4	14.2	3868
D16_5000	5000	16.5	15.9	15.8	4672
D18_5500	5500	17.5	16.9	15.8	5668
D18_6000	6000	18.5	18.0	15.8	6776
D20_6500	6500	20.0	19.5	16.0	8080
D21_7000	7000	21.0	20.3	16.0	9412
D22_7500	7500	21.5	20.9	16.0	10772

Table EC.4 shows the datasets used in the experiments discussed in Section 6. Each row identifies a unique demand dataset and presents the specific details. Here, the column “number of SKUs” shows the number of SKUs present in the respective dataset. “Maximum Length” reports the maximum length of all SKUs present in the dataset. “Maximum Breadth” & “Maximum Height” show similar dimensional values. “Number of Box Candidates” reports the total number of box candidates that can cover the SKUs present in the dataset.

## EC.10 Comparison with Meta-heuristics

We compare the performance of our approach with two standard meta-heuristic approaches: tabu search (TS) and simulated annealing (SA). Both algorithms iteratively search the local neighborhood until an optimal solution is found. TS exhaustively searches the whole neighborhood while SA tests a single neighbor randomly. Refer to Chopard and Tomassini (2018) for a review. These methods cannot directly generate the box sizes, we use them to improve our stage-1 solution. As discussed in stage 3 of the optimization framework, the improvement problem can be set up as a tree-search, where the initial solution (from stage 1) becomes the root node, and child nodes represent the possible transformations. Once such a tree is formed, improving the solution translates to selecting a child node at each level while traversing the tree depth. We choose the child node based on the respective heuristic rules in the meta-heuristic approaches. In TS, the best child node not present in the tabu list is selected, while SA randomly picks a child node and selects if it improves the solution. It can also select a child that worsens the solution with a small probability based on the metropolis rule. Please refer to Chopard and Tomassini (2018) for a thorough understanding. The packaging factors of child nodes are used as fitness or objective function values. At every level, the child node selected in the previous iteration serves as a parent node, and the selection process is repeated till the termination condition is reached. We terminate both approaches after they stop improving the solution for 1000 iterations. The best solution encountered in the search path is returned as the final solution.

**Table EC.5** Packaging factors (PFs) of the solutions generated by meta-heuristic approaches

Assortment Size	10	20	30	40	50	60	70	80	90	100
Tabu Search	2.968	2.358	2.163	2.028	1.947	1.925	1.882	1.830	1.793	1.769
Simulated Annealing	3.391	2.645	2.461	2.289	2.168	2.081	2.057	1.995	1.933	1.894
PAAS	2.930	2.305	2.056	1.959	1.867	1.813	1.777	1.764	1.717	1.699

In our experiments, we generate box assortments of various sizes (ranging from 10 to 100) to cover the SKUs present in  $D_{real}$  using the two meta-heuristics and the proposed approach. All the methods start from the same initial solution generated by the stage-1 algorithm for generating respective assortments. Table EC.5 presents the packaging factors of box assortments generated by each approach. The proposed approach (PAAS) performs better than the meta-heuristic approaches. On average, the solutions generated by PAAS are 4.1% better than those generated by tabu search and 15.1% better than those generated by simulated annealing. This is because the RL policy is far-sighted in that it selects child nodes that can result in better future solutions, while the meta-heuristics do not have a global view of the search tree and rely on the current child nodes alone. Among the meta-heuristic approaches, tabu search performs better than the simulated annealing approach by 10.6% on average. This is because the SA method randomly selects a solution and can accept worsening moves. Given the vast solution space, once it accepts a few worsening

moves, improving the solution with random moves becomes challenging. On the other hand, the tabu search evaluates all child nodes and stays near the best solutions in every iteration, resulting in a significantly better final solution.

## EC.11 Comparison with ML-based methods

In this section, we compare the performance of the proposed framework against existing ML-based methods. In particular, we leverage the dimensionality reduction methods from unsupervised learning. Dimensionality reduction methods such as PCA (principal component analysis) compress the data from a high-dimensional to a low-dimensional space by retaining relevant information and discarding redundant information. Specifically, they combine or transform a given set of dimensions to generate a smaller set of dimensions. For instance, consider a dataset that contains  $m$  records and  $d$  dimensions; dimensionality reduction methods transform this dataset into a new dataset with  $m$  records and  $d'$  dimensions. Here,  $d' < d$ . These methods try to preserve the structure and variability of the original dataset with a smaller set of dimensions in the transformed dataset. We leverage this compression property to generate box dimensions.

We consider three dimensionality reduction methods for this exercise: principal component analysis (PCA), locally linear embedding (LLE), and t-distributed stochastic neighbor embedding (TSNE). We have chosen these methods for their superior performance on many ML tasks. PCA is known to perform well on natural datasets (Van Der Maaten et al. 2009); however, LLE is known to outperform PCA in certain applications (Bartenhagen et al. 2010). TSNE is a relatively recent method known for handling complex data very well (Van der Maaten and Hinton 2008). Other methods, such as Isomap and Kernel PCA, could not be included since they are memory intensive and require hundreds of gigabytes of memory for modeling SKUs  $\geq 100,000$ .

For our box sizing problem, recall that we have  $n$  SKUs in the dataset, and each SKU  $i$  can be represented as a tuple  $(l_i, b_i, h_i)$  denoting its length, breadth, and height. Therefore, we have three dimensions describing the SKU dataset. We transform the three dimensions into one by applying a dimensionality reduction method. The resultant dimension takes a continuous range of values and captures variability in the features (dimensions) of the SKUs, taking similar values for SKUs of similar sizes. Suppose we require  $K$  boxes to cover the  $n$  SKUs; we group the SKUs along the generated single dimension into  $K$  groups and generate a box size that can fit all the SKUs in a respective group (see EC.4 Algorithm 1). For breaking the SKUs into groups, we employ the Jenks natural breaks (JNB) algorithm. JNB breaks a continuous range of numbers into groups by minimizing within-group variance and maximizing between-group variance. In this manner,  $K$  boxes can be created for  $K$  groups; thereby, a  $K$ -box assortment can be produced.

In the experiments, we generate box assortments of various sizes (ranging from 10 to 100) to cover the SKUs present in the real-world dataset  $D_{real}$  using three dimensionality reduction methods and the proposed framework. We employ the procedure described earlier in this section to generate the box dimensions with

dimensionality reduction methods. Though the described procedure is applicable once three box dimensions are compressed into one dimension, the compression procedure varies between different approaches. For instance, in the case of PCA, the SKU dataset is mean-centered, and a covariance matrix of the mean-centered dataset is computed, eigenvalues and eigenvectors of the same matrix are generated, and finally, a single dimension along the eigenvector with maximum variance is generated. For more procedural details about other dimensionality reduction methods, readers can refer to Van der Maaten and Hinton (2008) and Van Der Maaten et al. (2009).

**Table EC.6** Packaging factors (PFs) of the solutions generated by ML methods

Assortment Size	10	20	30	40	50	60	70	80	90	100
PCA	3.749	3.289	3.127	3.041	2.998	2.951	2.941	2.935	2.891	2.895
TSNE	4.084	3.161	3.147	2.751	2.573	2.413	2.276	2.232	2.271	2.083
LLE	10.201	6.212	4.095	5.294	6.539	7.122	4.003	6.613	3.596	3.653
PAAS	2.930	2.305	2.056	1.959	1.867	1.813	1.777	1.764	1.717	1.699

We report the packaging factor values of the box assortments generated by the respective methods in Table EC.6. From the table, we can observe that the proposed framework outperforms alternative ML-based methods. The proposed approach, PAAS, is better than PCA, TSNE, and LLE by 57%, 35%, and 185%, respectively. Therefore, the proposed approach is superior to the existing ML-based methods.

## EC.12 Robustness Analysis

In this section, we explore the robustness of the solutions generated by our approach. In the first examination, we explore whether the generated solution is robust to changes in the SKU demand. To this end, we alter the demand of all the SKUs present in the original demand dataset by up to 20% and analyze how the packaging factors vary with changes in demand. The second examination explores whether the generated solution can cover a new set of SKUs added to the warehouse. To this end, we randomly generate new SKU dimensions and analyze how many of the generated SKUs are covered or uncovered by the generated assortment. Finally, we check whether the generated solutions can accommodate additional cushioning requirements.

### EC.12.1 Demand Variation Analysis

In our approach, we generated the solution for covering the expected demand of the SKUs; however, in real-world settings, the actual demand deviates from the expected value. Therefore, we examine the effect of deviations in demand on the volumetric efficiency of the generated assortment. To this end, we perturb the expected demand values of the SKUs from the real-world dataset from 5% to 20% and check the packaging factors when the perturbed demand is covered by the assortments generated for the unperturbed demand values. This analysis examines how robust the solutions generated will be for future demand variations.

**Table EC.7** Packaging factors for various demand variations

Assortment Size $K$	Demand Variation (in %)				
	0	5	10	15	20
10	2.9489	2.9471	2.9480	2.9548	2.9608
20	2.2693	2.2688	2.2691	2.2744	2.2750
30	2.0440	2.0435	2.0443	2.0482	2.0471
40	1.9116	1.9113	1.9120	1.9153	1.9145

Table EC.7 reports the packaging factors for various demand variations for each  $K$ -box assortment. From the table, we make two observations. First, the packaging factors change negligibly (by less than 1%) across all the demand variations. This pattern can be consistently seen across all assortments and signifies that the generated solution is robust to the demand changes. Secondly, variations in packaging factors of larger assortments are relatively less compared to that of smaller assortments. As the number of boxes increases, the assortment becomes more specific to the cluster of SKUs, and therefore, their demand slightly affects the packaging factor.

### EC.12.2 Coverage Analysis

In real-world operations, new SKUs are constantly introduced, and these new SKUs must be covered by the box assortments already in use at the time of introduction. This analysis tests whether the generated solutions are robust enough to cover new SKUs whose dimensions are not considered during the solution generation process. To this end, we sample a new set of dimensions from a normal distribution of dimensions parameterized by the mean and standard deviation of dimensions from the real-world dataset. In this analysis, we sampled 100,000 dimensions, out of which 82270 dimensions are non-negative and adhere to maximum and minimum dimension requirements. The generated dimensions act as proxies for the new SKUs. We then test whether the generated solutions cover them.

**Table EC.8** Coverage values of synthetic SKUs

Assortment size ( $K$ )	10	20	30	40
<b>Total number of new SKUs</b>	82270	82270	82270	82270
<b>Number of new SKUs covered</b>	82224	82211	82198	82198
<b>Covered percentage</b>	99.944%	99.928%	99.912%	99.912%
<b>Number of new SKUs uncovered</b>	46	59	72	72
<b>Uncovered Percentage</b>	0.056%	0.072%	0.088%	0.088%

Table EC.8 shows the coverage values of generated SKUs by each  $K$ -box assortment. The table shows that the generated assortments cover more than 99.9% of the generated SKUs. We also observe that as the assortment size increases, the number of uncovered SKUs increases. The larger assortments are specific to

the SKUs clusters, while smaller assortments are general, where every box tries to fit many dimensions. Nonetheless, all the assortment sizes cover more than 99.9% of the new SKUs. This shows that the generated assortments are robust to introducing new SKUs, and managers need not spend effort procuring new box sizes as new SKUs are added.

### EC.12.3 Robustness Analysis for High-value Items with Extra Cushioning

E-commerce warehouses host a variety of products, and some items may require extra cushioning for protection. The use of additional cushioning increases the effective product dimensions. For instance, if an item has  $l, b, h$  as dimensions, adding extra cushioning  $e$  will make its effective dimensions  $l + e, b + e, h + e$ . In such cases, since the dimensions of the items increase, finding a suitable box for packing the product may become difficult. Therefore, in this section, we test whether the box assortments generated by our approach can accommodate items whose dimensions are significantly modified due to extra cushioning. To this end, we randomly select 10,000 items from the real-world dataset and check whether the generated box assortments can accommodate them if an extra cushioning value  $e$  is added. We consider  $e$  values between 0.5 and 2.0 inches.

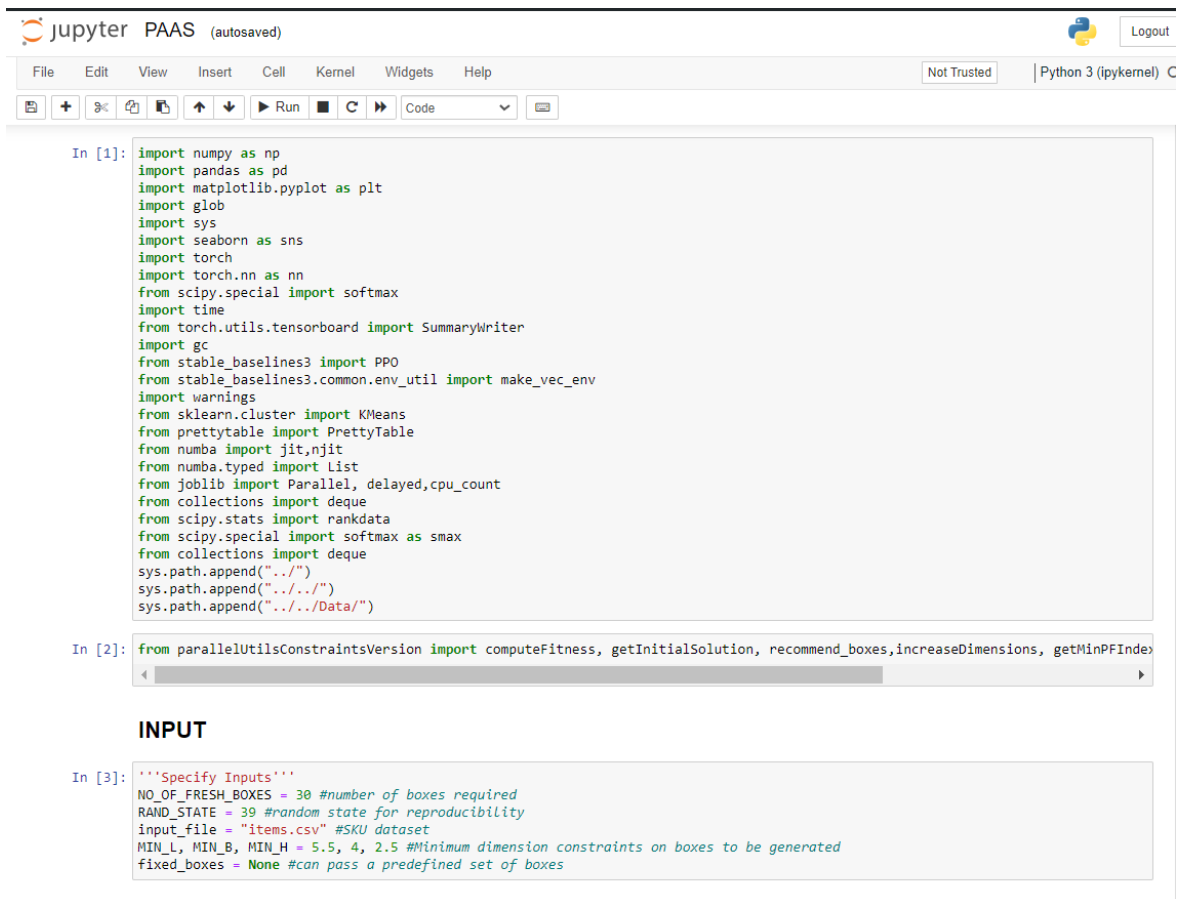
**Table EC.9** Coverage values of SKUs requiring extra protection

Assortment Size, $K$	Cushioning value, $e$ (in inches)			
	0.5	1.0	1.5	2.0
10	99.79	99.61	99.23	98.57
20	99.71	99.29	98.86	98.41
30	99.56	99.11	98.75	98.38
40	99.56	99.11	98.75	98.38

Table EC.9 shows the percentage of items that can be accommodated or covered by a  $K$  box assortment when a cushioning value  $e$  is added to the product. Observe that more than 99% of the items can be covered by all the assortments if a cushioning value of  $\leq 1.0$  inch is used. Similarly, more than 98% of the items can be covered if a cushioning value between 1 and 2 inches is used. This shows that the generated assortments are robust to extra cushioning requirements. Nevertheless, the cushioning requirements of items can be passed to the optimization framework during the assortment creation phase itself. For instance, say  $l, b, h$  are the length, breadth, and height dimensions of an SKU item, and if it requires a cushioning of 0.5-inch on all sides, the SKU dimensions can be adjusted as  $l + 0.5, b + 0.5, h + 0.5$  in the dataset. This adjustment takes care of the cushioning/protection requirements of items and helps the optimization framework generate accurate box dimensions, resulting in 100% coverage. Our industry collaborator follows the same procedure.

## EC.13 Optimization Software

Figure EC.4 shows the implementation of the proposed optimization framework. The SKU dataset, which consists of dimensions and demand, is read as an input file (`items.csv`). The number of boxes required to cover the SKUs can be entered into the variable `NO_OF_FRESH_BOXES`. To ensure that the generated boxes comply with the minimum length, breadth, and height requirements, `MIN_L`, `MIN_B`, `MIN_H` variables are made available. If the user wants to fix a predefined set of boxes, which should be a part of the final assortment, they can be passed as a numpy matrix to `fixed_boxes`. Figure EC.5 shows the output of the program. It reports the running time, the packaging factor of the generated box assortment, and finally, the box assortment itself.



The screenshot shows a Jupyter PAAS interface with the following code in three input cells:

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import glob
import sys
import seaborn as sns
import torch
import torch.nn as nn
from scipy.special import softmax
import time
from torch.utils.tensorboard import SummaryWriter
import gc
from stable_baselines3 import PPO
from stable_baselines3.common.env_util import make_vec_env
import warnings
from sklearn.cluster import KMeans
from prettytable import PrettyTable
from numba import jit, njit
from numba.typed import List
from joblib import Parallel, delayed, cpu_count
from collections import deque
from scipy.stats import rankdata
from scipy.special import softmax as smax
from collections import deque
sys.path.append("../")
sys.path.append("../..")
sys.path.append("../Data/")
```

```
In [2]: from parallelUtilsConstraintsVersion import computeFitness, getInitialSolution, recommend_boxes, increaseDimensions, getMinPFIndex
```

**INPUT**

```
In [3]: '''Specify Inputs'''
NO_OF_FRESH_BOXES = 30 #number of boxes required
RAND_STATE = 39 #random state for reproducibility
input_file = "items.csv" #SKU dataset
MIN_L, MIN_B, MIN_H = 5.5, 4, 2.5 #Minimum dimension constraints on boxes to be generated
fixed_boxes = None #can pass a predefined set of boxes
```

Figure EC.4 Program Input



-----  
 Started Policy Search: 1 / 1  
 Search Completed in: 26.348041407267253 minutes  
 Best Solution: 1.705353344508886

**OUTPUT**

```
In [22]: print(table)
```

SKUs	Boxes	Packaging Factor	CPU Time(sec)
8442	30	1.705353344508886	1580.889546491002

```
In [23]: print(sols)
```

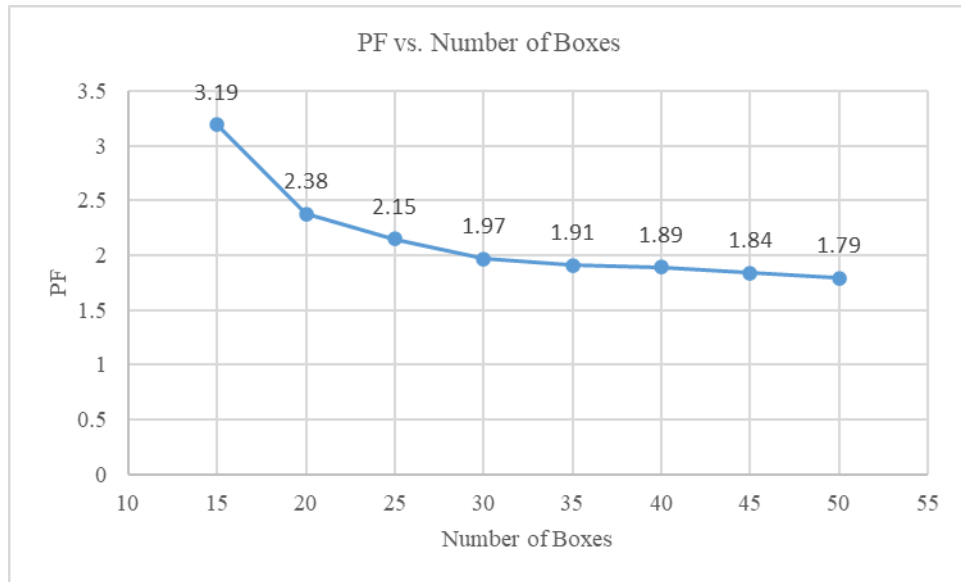
```
[array([[ 8. ,  4.5,  2.5],
        [ 5.5,  4. ,  3.5],
        [ 7.5,  4. ,  5. ],
        [ 9. ,  5.5,  2.5],
        [ 9.5,  7. ,  2.5],
        [11.5,  9. ,  2.5],
        [13. ,  4.5,  3.5],
        [ 8. ,  5.5,  6.5],
        [10.5,  8. ,  4. ],
        [13. ,  9.5,  5. ],
        [14. , 12.5,  3. ],
        [15.5,  4.5,  5.5],
        [29.5, 10.5,  2.5],
        [11. ,  8.5,  8. ],
        [14.5, 11. ,  5.5],
        [14. ,  9. ,  7.5],
        [12.5, 10. ,  9.5],
        [17. , 10.5,  8.5],
        [17. , 15. ,  5. ],
        [20. , 13. ,  6.5],
        [23. , 16. ,  7. ],
        [15. , 11.5, 11. ]])]
```

Figure EC.5 Program Output

## EC.14 Choice of assortment size

As the assortment size increases, the packaging factor (PF) decreases. However, the marginal contribution to the reduction in PF becomes low after a point. For instance, consider the following plot that shows how packaging factors vary with the increase in the number of boxes. These results are obtained when 200,000 SKUs are covered using the assortments of respective sizes.

In plot EC.6, observe that as the number of boxes increases from 35 to 40, the incremental benefit in PF becomes less. On the other hand, operating with a large number of boxes increases the cost per unit packaging, increases packing time due to increased search time, and increases other stocking complexities. Therefore, it is preferable to choose a smaller but optimal assortment. The company has chosen 35 boxes based on the assortment size that they can handle.



**Figure EC.6** PF vs Assortment Size

---

## References

- Bartenhagen, Christoph, Hans-Ulrich Klein, Christian Ruckert, Xiaoyi Jiang, Martin Dugas. 2010. Comparative study of unsupervised dimension reduction techniques for the visualization of microarray gene expression data. *BMC bioinformatics*, 11 (1), 567.
- 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.
- Chopard, Bastien, Marco Tomassini. 2018. *An introduction to metaheuristics for optimization*. 1st ed. Natural computing series, Springer International Publishing, Cham, Switzerland.
- 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.
- 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.
- Van der Maaten, Laurens, Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9 (86), 2579-2605.
- Van Der Maaten, Laurens, Eric O Postma, H Jaap van den Herik, et al. 2009. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10 (66-71), 13.
- 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.
- 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.