Proceedings of the 6th European Conference on Industrial Engineering and Operations Management
Lisbon, Portugal, July 18-20, 2023

E-Commerce Optimal Packaging Project

Alara Kumru, Kerem Tuna, Berfin Kondakcı, Efe Doğruer,
Şevval Berksöz and Onat Gürcanok
Faculty of Engineering,
Department of Industrial Engineering,
Ozyegin University,
Turkey

Abstract

The subject of our senior design project is optimized product packaging. Companies may generally have trouble packing goods of various types, sizes, and volumes. The amount of waste produced by using cardboard and the precise arrangement and dimensions of the boxes to be used in order packaging are the two most significant issues related to the packaging issue. Random packaging, the use of parcels, and the absence of a set packaging sequence or rule result in wasteful cardboard usage, transportation space consumption, and financial loss because each parcel utilized has a cost. The boundaries and restrictions of the challenge were established after an analysis of potential packaging problems faced by the company. Before modeling, there has been literature research. Since we can use restricted amount of different sized boxes the main purpose of the model is to minimize the total volume with providing the information about which order should be placed on which box and in which order. Since the real data set is so large, the heuristic box packing algorithm is developed and its output data is used to decide the packaging orders into boxes.

Keywords
Volume optimization, Logistics industry, Order packaging, Box packing algorithm, Two stage model.

1. Introduction
Xxx Logistics is a mixed logistics company that earns the most of its revenue in warehousing and road transportation. It provides freight transportation for warehouses and online purchases from top Turkish businesses. During our trip to their facility, they informed us about not having an e-commerce order packing system.

Each brand's sections of the warehouse are separated. Each item is assigned a code and placed on the shelves in that order. When the order is received, the officer collects the items and sends them to be wrapped. The items in the order are sorted to fit into one of the 10 boxes that the packing officer deems suitable. As a result, more cardboard is needed, and the package occupies more room in the conveyed vehicle because this does not demonstrate the ideal sequence and outcome.

In this project, we prepared the orders as rectangular prisms and we determined that according to the rule of having the minimum total volume. We applied the model we did for it, for small sized problems beside the big ones. You can see the detailed explanation of them in the following chapters. In this report, the volume of the boxes problem of Xxx Logistic will be discussed and solved.

The problem is analyzed in detail, input data, iterative improvement, greedy resizing, box resizing, order transfer and optimization techniques are planned and a heuristics model based on the optimized box packing algorithm is developed and coded in Java. Detailed explanation and comparison about the results are made in the following chapters.

2. Literature review
Most of the work that was accomplished during this term was focused on improving the model that was constructed during the previous term and reducing the complexity that were observed. Because of this, the articles "A New Mathematical Model for a 3D Container Packing Problem", (Ocloo et al. 2020) and "A Global Optimization Method for Packing Problems", (Hu et al. 2002) which we have investigated in depth throughout the first term and presented in our first report played an extremely important role in the creation of our first step linear model in the chapter 3.4.1. To create the constraints (1-13) and the constraints (17-19), we have benefitted from these articles. During this time,

we worked on improving our model, which was formed based on the papers that we read and researched, with the goal of simplifying the process and developing new approaches. In the span of this period, some of the steps that we took as part of our strategy included simplifying the model into just two stages, constructing a nonlinear model, and, as a concluding step, designing our very own algorithm. The study includes extensive discussion of each of these aspects in full detail.

3. Methods

3.1 Problem Definition

The company we are working with requested that we design a complicated system for this job. This complex method is a developed way to solve the company's current issue.
By using the minimum desi which means the volumetric weight carried by a cargo with its package, and taking into account transportation conditions like the type of vehicle, size, and how it will be placed in accordance with the nature of the products in the order prepared specifically for each customer, shipping parcels and packaging are planned in a way that minimizes waste.

First, at the very beginning of our project, we created a linear model and proceeded through this model. Later, we converted this model we created into a two-stage model. We solved the first stage and did not encounter any problems. We solved the first stage of our model to be linear. For the second stage, we made the model nonlinear and solved it to be nonlinear.

This model and solution we created has been a model and solution developed to solve an existing problem within the company that Xxx communicated to us. Our aim in creating this model was to solve the packaging problem that Xxx conveyed to us. This solution has emerged in order to choose and use the types and sizes of the boxes used to place the ordered products in the most optimal way. Our algorithm that we created to solve this problem is a heuristic algorithm.

3.2 Problem Formulation

There are several assumptions that generate the first and second steps of the model. We considered the items in each order and the order as a rectangular prism, and each item will be placed so that its edges are parallel to one of the order's x, y, or z axes. In accordance with the order dimensions obtained at the completion of the first step, the orders assumed to be this rectangular prism will be placed in one of the boxes in a manner that minimizes the total volume.

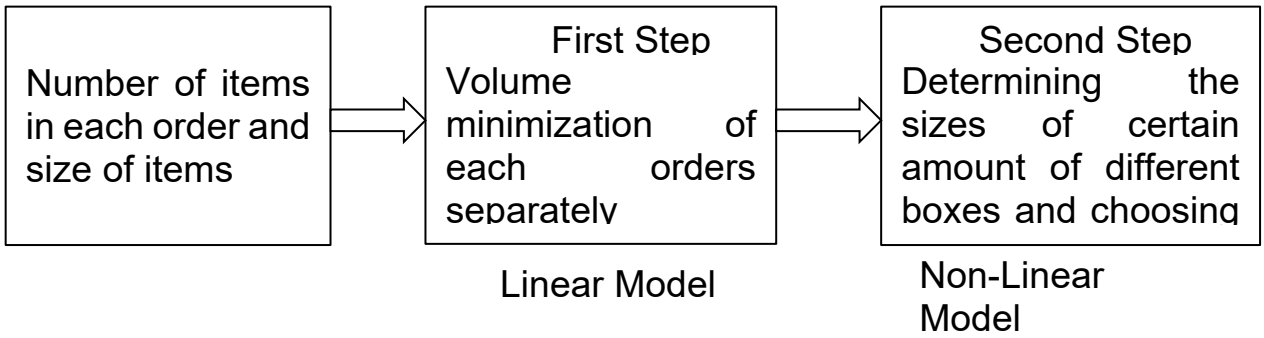


Figure 1. Steps of the Problem Formulation

It is assumed that all the objects in each order and all of the orders are rectangular prisms. Each order includes a list of the items' sizes and quantities. Only six orientations can be used to arrange the objects so that their edges are parallel to the x, y, and z axes. The first step of the problem involved minimizing the volume of each order separately under the assumption that each order was made up of rectangular prisms, taking into consideration the quantity and sizes of

all the items in each order. Our goal in the second step is to size a specific number of boxes in order to reduce the total volume, and we select one of these boxes for each order in order to fit the order inside of it. This is done using the order sizes that were received from the first stage.

Instead of defining variables for the six possible rotations, we sorted the edges of each order and gave them the names large, medium, and small in order to reduce the complexity of our second step model. This is because, based on our assumption, we have declared that there are only six possible rotations in which each edge is parallel to the edges of boxes, in order to be able to place the order into a box, there are only six such rotations. The box's medium edge should be smaller than the box's medium edge, and the order's large edge should be smaller than the box's large edge. The box's small edge should be smaller than the small edge.

Finally, as linear and mixed integer nonlinear models were unable to provide the efficiency we desired, we built an heuristic algorithm in order to speed up the solution and address more complex problems.

Our algorithm is a greedy, local search-based algorithm that iteratively improves the packing of orders into boxes by discovering various combinations of box sizes and switching orders between boxes. This process is known as iteratively improving the packing of orders into boxes. By taking use of this strategy, the algorithm can locate near-optimal solutions for large-scale problems, even in situations where conventional optimization models might be constrained by time

3.3 Choice of modeling approach and its justification

The three-dimensional box packing model is a problem in mathematical optimization. The model aims to reduce the total volume needed to pack a collection of rectangular products.

The model in the first step is commonly expressed as a mixed-integer linear program (MILP), a type of optimization problem where the objective function and constraints are linear and some of the variables are constrained to integer values. The integer variables in the first step of the problem correspond to the orientations of items which are assumed to be rectangular prisms in six different rotations in which their edges are parallel to the x,y and z axes and placement of these items according to each other.

A variety of solution methods, such as branch and bound, cutting planes, column generation, heuristics, and metaheuristics, can be used to solve the MILP. The method chosen is determined by the size and complexity of the issue at hand, the required level of solution quality, and the available computational resources. We chose to use the Gurobi package of Java to solve the first step of our problem. In the second step of our problem we used a model which is expressed as a mixed-integer nonlinear program in which there exist nonlinear constraints or nonlinear objective functions and integer variables. The integer variables in our second step model correspond to the chosen box type for each of the orders separately. To solve this model we have tried Gams Studio, Neos online solvers and Excel Solver. However, due to the complexity of this model, these solvers were able to solve only smaller sizes of problems than we needed. So, we have designed an algorithm to solve the second step of our problem. We chose a heuristic,

local search-based approach for our algorithm due to its numerous advantages over conventional optimization models:

• Scalability: Heuristic algorithms are more scalable than exact algorithms, allowing them to manage larger problem sizes that pose computational challenges for exact methods.

• Efficiency: Native search techniques enable efficient exploration of the search space, making them suitable for real-world applications with limited computational resources.

• Flexibility: The proposed algorithm can be adapted to a variety of industrial environments, optimizing resource utilization and contributing to more efficient and sustainable supply chain practices.

• Computational efficiency: Heuristic algorithms typically demand less computation than exact methods, resulting in faster solutions.

However, heuristics are not without their drawbacks:

• Optimality: Heuristic algorithms do not guarantee the discovery of the optimal global solution. Instead, they frequently generate near-optimal solutions that may be close to optimal.

• Sensitivity: The performance of heuristics is affected by the selection of heuristic parameters and initial conditions, which can impact the quality of the final solution.

The algorithm uses a combination of greedy search, local search, and iterative improvement to pack orders into boxes and find a near-optimal solution. Although it cannot always find the global optimum, it can explore the search space efficiently and often produce good packaging solutions.

A heuristic algorithm is a methodical way to make decisions that don't only rely on exact arithmetic calculations but also on experience and intuition. This strategy makes it possible to address issues more quickly and can deliver highly accurate solutions, especially for complex issues where other approaches might be challenging.

3.4 Model Representation

We initially created a two-step solution to the problem. These two phases are represented by two distinct models. The high computational complexity of a single-stage model that we generated in the first term was the primary reason for its two-stage design, as stated in the initial report. Listed below are the two models and their respective explanations.

3.4.1 First Step:

Linear Model

M: Very large number
MS: Maximum edge length

N : Number of items in order

Indices

1. i: Index of Item {1,...,N} {N = Number of items}
2. k: Index of Item {1,...,N} {N = Number of items}
3. u: Index for Size {1,...,MS} {MS = Maximum edge length}

Parameters

1. p_i: Length of item i
2. q_i: Width of item i
3. r_i: Height of item i

Decision Variables

1. x_i: x coordinates of Left-Bottom-Behind of item i
2. y_i: y coordinates of Left-Bottom-Behind of item i
3. z_i: z coordinates of Left-Bottom-Behind of item i
4. a_{ik}={1, if item i is on the left side of item k
   0, otherwise
5. b_{ik}={1, if item i is on the right side of item k
   0, otherwise
6. c_{ik}={1, if item i is on the behind of item k
   0, otherwise
7. d_{ik}={1, if item i is on the front side of item k
   0, otherwise
8. e_{ik}={1, if item i is under the item k
   0, otherwise
9. f_{ik}={1, if item i is on the item k
   0, otherwise
10. l_{xi}={1, if length of item i is parallel to x-axis
    0, otherwise
11. l_{yi}={1, if length of item i is parallel to y-axis
    0, otherwise
12. l_{zi}={1, if length of item i is parallel to z-axis

0, otherwise

13. w_{xi}={1, if width of item i is parallel to x-axis
0, otherwise
14. w_{yi}={1, if width of item i is parallel to y-axis
0, otherwise
15. w_{zi}={1, if width of item i is parallel to z-axis
0, otherwise
16. h_{xi}={1, if height of item i is parallel to x-axis
0, otherwise
17. h_{yi}={1, if height of item i is parallel to y-axis
0, otherwise
18. h_{zi}={1, if height of item i is parallel to z-axis
0, otherwise
19. $B1_u$: Continuous (length is equal to its own value and width is equal to index u)
20. $S1_u$: Continuous (continuous variable summation of $B1_u$* u, and its index u is equal to the height)
21. $HMax_u$={1, if height is equals to u
0, otherwise
22. $WMax_u$={1, if width is equals to u
0, otherwise
23. $LMax_u$={1, if length is equals to u
0, otherwise

Mathematical Model

$$OBJ=\text{minimum } \sum_{u=1}^{MS} S1_u \, u$$

Subject to

$x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \leq x_k + (1-a_{ik})M \quad , \quad \forall \; i,k \in N \quad$ **[1]**

$x_k + p_k l_{xi} + q_k l_{xk} + r_k h_{xk} \leq x_i + (1-b_{ik})M \quad , \quad \forall \; i,k \in N \quad$ **[2]**

$y_i + p_i l_{yi} + q_i w_{yi} + r_i h_{yi} \leq y_k + (1-c_{ik})M \quad , \quad \forall \; i,k \in N \quad$ **[3]**

$y_k + p_k l_{yk} + q_k w_{yk} + r_k h_{yk} \leq y_i + (1-d_{ik})M \quad , \quad \forall \; i,k \in N \quad$ **[4]**

$z_i + p_i l_{zi} + q_i w_{zi} + r_i h_{zi} \leq z_k + (1-e_{ik})M \quad , \quad \forall \; i,k \in N \quad$ **[5]**

$z_k + p_k l_{zk} + q_k w_{zk} + r_k h_{zk} \leq z_i + (1-f_{ik})M \quad , \quad \forall \; i,k \in N \quad$ **[6]**

$a_{ik} + b_{ik} + c_{ik} + d_{ik} + e_{ik} + f_{ik} \geq 1 \quad , \quad \forall \; i,k \in N \quad$ **[7]**

all items in an order should be at least in left tor right or back or front or below or above of any other item)

$l_{xi} + w_{xi} + h_{xi} = 1 \quad , \quad \forall \; i \in N \quad$ **[8]**

(at least one edge should be parallel to x axis)

$l_{yi} + w_{yi} + h_{yi} = 1 \quad , \quad \forall \; i \in N \quad$ **[9]**

$l_{zi} + w_{zi} + h_{zi} = 1 \quad , \quad \forall \; i \in N \quad$ **[10]**

$l_{xi} + l_{yi} + l_{zi} = 1 \ , \ \forall \ i \in N$ **[11]**

(length should be parallel to one of the axis)

$w_{xi} + w_{yi} + w_{zi} = 1 \ , \ \forall \ i \in N$ **[12]**

$h_{xi} + h_{yi} + h_{zi} = 1 \ , \ \forall \ i \in N$ **[13]**

$\sum^{MS}_{u=1} Wmax_u = 1$ **[14]**

(if width of the order is u then, $Wmax_u$ is 1)

$\sum^{MS}_{u=1} Hmax_u = 1$ **[15]**

$\sum^{MS}_{u=1} Lmax_u = 1$ **[16]**

$x_i + p_i l_{xi} + q_i w_{xi} + r_i h_{xi} \leq \sum^{MS}_{u=1} Lmax_u u \ , \ \forall \ i \in N$ **[17]**

$y_i + p_i l_{yi} + q_i w_{yi} + r_i h_{yi} \leq \sum^{MS}_{u=1} Wmax_u u \ , \ \forall \ i \in N$ **[18]**

$z_i + p_i l_{zi} + q_i w_{zi} + r_i h_{zi} \leq \sum^{MS}_{u=1} Hmax_u u \ , \ \forall \ i \in N$ **[19]**

$\sum^{MS}_{u=1} \ B1_u \geq \sum^{MS}_{u=1} Lmax_u u$ **[20]**

$B1_u \leq MWmax_u \ , \ \forall \ u \in MS$ **[21]**

$\sum^{MS}_{u=1} S1_u \geq \sum^{MS}_{u=1} \ B1_u \ u$ **[22]**

$S1_u \leq MHmax_u \ , \ \forall \ u \in MS$ **[23]**

$S_u \geq 0 \ , B1_u \geq 0 \ , \ \forall \ u \in MS$ **[24]**

$x_i \ , y_i \ , z_i \geq 0 \ , \ \forall \ i \in N$ **[25]**

Detailed Explanation of <u>First Step:</u>

Our first model's objective is to determine the dimensions of the rectangular prism with the minimum volume that each order can fit into individually. Each order consists of a certain amount of items, all of which we consider to be rectangular prisms, and we arrange them in such a way as to fit into the minimum rectangular prism that the order will fit into. Finally, we consider the rectangular prism with the minimum volume that will turn out, as the order itself. In this way, the dimensions of the order are formed as a rectangular prism, minimizing the volume.

The order dimensions that end up being in the shape of a rectangular prism are used as inputs for the order dimensions that will be used in the second stage. You can see in section 5.3.2 for further information regarding the second stage.

Constraints (1-6) indicate the positions of different items compared with each other. The end point of an item which is closer to the origin(left back bottom corner) has smaller coordinates than the starting point of the other item in order to eliminate the overlapping problem. (left, right, back, front, below, above)

Constraint (7) indicates that all items must be placed at least one of the positions which are right, left, behind, front, below or above any other item respectively.

Constraints (8-10) implies that for each axis, there is one edge of item i which is parallel to that axis.

Constraints (11-13) implies that for each edge of item i, there is one axis which is parallel to that edge.

Constraints (14-16) "u" index of corresponding decision variable is equal to the corresponding edge if that variable is equal to 1.

Constraints (17-19) ensure that the end point of each item is less than total length, width and height.

Constraint (20-21) indicates that the value of the decision variable $B1_u$ which has an index u that is equal to the width is equal to the length.

Constraint (22-23) indicates that the decision variable $S1_u$ which has an index u which is equal to the height is equal to the summation of $B1_u$ times u, which is equal to the length times width. (x-y surface).

Constraints (24-25) Non-negativity constraints for continuous variables

3.4.2 Second Step:

Non linear model:

M: very large number
N: number of orders
Nb: number of box types

Indices
4. i: Index of Order
5. j: Index of Box

Parameters
4. l_i: Largest edge of order i
5. m_i: Medium edge of order i
6. s_i: Smallest edge of order i

Decision Variables
24. L_j: Largest edge of box j
25. M_j: Medium edge of box j
26. S_j: Smallest edge of box j
27. o_{ij}={1, if order i is in box j
    0, otherwise

Mathematical Model

$$OBJ=minimum(\sum^{N}i=1 * \sum^{Nb}j=1 * o_{ij} * L_j * M_j * S_j)$$

Subject to

$L_j \geq l_i - M(1-o_{ij})$ $\forall$ i $\in$ [1,N] , $\forall$ j $\in$ [1,Nb] [1]

$M_j \geq m_i - M(1-o_{ij})$ $\forall$ i $\in$ [1,N] , $\forall$ j $\in$ [1,Nb] [2]

$S_j \geq s_i - M(1-o_{ij})$ $\forall$ i $\in$ [1,N] , $\forall$ j $\in$ [1,Nb] [3]

$\sum^{Nb}_{j=1} o_{ij} = 1$ $\forall$ i $\in$ [1,N] [4]

Detailed Explanation of Second Step:
The need in this model is fitting the results of the first step, which are order sizes, in a restricted amount of different sized boxes. That is why this non-linear model is written to determine 10 different box sizes. In the first step, we determined rectangular prisms and this non-linear model helps us to put those rectangular prisms in 10 different sized boxes and the important part in this is to determine the box sizes in a way that the total volume will be the minimum.

In the model, orders and boxes are always rectangular prisms and orders are placed inside the box with each side parallel to the edge of the box. With this way, the need of defining binary variables for rotations happened and it increased the complexity. That is why we named each order's edges as large, medium and small and ordered them, instead naming them as length, height and width. According to our first assumption, each side of the order is thought out to be parallel to one of the x,y and z axes of the box. To fit this order into the box, it is required that the large side, medium side and the short side of the order should be less than or equal to the large side, medium side and short side of the box consecutively. You can see this in the parameters. Constraint 1-2-3 represents the rule for an order to fit in a box in which the large edge of the order must be smaller or equal than the large edge of the box. The same rule is applied for the medium and the small edge too. The first three constraints show that. The fourth constraint shows that one order can be chosen for only one box. Which means that there is only one j for each i that is why the sum of the j's must be equal to 1.

As a result of this model, we have determined a certain number of different box sizes in order to take up the minimum volume of total orders. However, non-linear model did not work on Neos and Excel Solver for big sizes because the complexity of them are high. That is why we proposed an algorithm for this problem. We used the non-linear model in Neos and Excel to solve small sized problems and also we used the same small sized problems in the algorithm to see the difference between them and to find the optimality gap. It also provided us with the performance of the algorithm. You can see the detailed explanation of the algorithm in the algorithm part and see the results of it in the following chapter.

3.4.3 Heuristic Algorithm

Our heuristic solution algorithm for the BPP involves the following components:

- Inputs: A list of orders, each with three dimensions (largest edge, medium edge, and shortest edge), and a list of potential box dimensions.

- Outputs: An optimized packing solution, represented by a list of boxes with their respective dimensions and packed orders.

Our algorithm is implemented using the Java programming language and consists of four main classes: Main, Environment, Box, and Order. The following sections provide a detailed description of each class, focusing on their functionality and interactions in the context of the algorithm.

The algorithm is based on a greedy, local search approach that iteratively improves the packing solution by resizing and redistributing orders between box pairs. It makes use of four primary classes (Main, Environment, Box, and Order) to represent and manipulate the packing solution.

- Main Class: The entry point of the program. It reads data from three text files (large.txt, medium.txt, and short.txt) and initializes three lists (largelist, mediumlist, and smalllist) with the respective values from the files. It then creates an initial environment, finds the best packing solution using the bestSol method, updates the dimensions of the boxes in the environment, and finally computes and displays various statistics (number of orders in each box, dimensions of each box, sum of over-allocated space, and total volume).

- Environment Class: Represents the packing environment, containing a list of boxes and their associated orders. This class contains methods for calculating total volume, updating the packing solution, and applying the local search-based improvement procedure.

- Box Class: Represents a single box with its dimensions, volume, and a list of orders. This class contains methods for setting box dimensions, updating the volume, and adjusting the box size to optimize the packing solution.

- Order Class: Represents a single order with its dimensions. This class has a constructor that initializes the order with the given dimensions.

Main Class Functionalities:
The Main class serves as the entry point of the program. It reads data from three text files (large.txt, medium.txt, and short.txt) and initializes three lists (largelist, mediumlist, and smalllist) containing the respective values from the files. It then generates the range of the size values with reducing and sorting the lists of each of the three size categories (large, medium, and small).

The core functionality of the program involves creating an initial environment, finding the best packing solution using the bestSol method, which is working until the isImprovable attribute of environment become false, updating the dimensions of each two box pairs within the environment, and finally, after computing and displaying various statistics such as the number of orders in each box, dimensions of each box, and total volume it chooses best improvements among them and updates the environment accordingly and re-run the itself with the updated environment.

Environment Class Functionalities:

The Environment class represents the packing environment, which contains the list of boxes and their associated orders. The following methods are implemented within this class:

- Environment(ArrayList<Box> list): Constructor that initializes the environment with the given list of boxes and calculates the total volume of packed orders.
- changeImprovable(): Changes the is Improvable attribute to false.
- updateVol(): Updates the total volume of the packed orders in the environment.
- changeBox(Box b1, Box b2): Transfers orders from one box (b2) to another (b1) if the orders in box b2 can be fit into box b1.
- improve(int b1, int b2): Attempts to improve the packing solution by resizing and redistributing orders between the box pairs b1 and b2. It iteratively explores different box size combinations and selects the one that minimizes the combined volume occupied by the orders in both boxes.

Box Class Functionalities:

The Box class represents a single box with its dimensions, volume, and a list of orders. The following methods are implemented within this class:

- Box(double l, double m, double s): Constructor that initializes the box with the given dimensions and calculates its volume.
- setEdges(double l, double m, double s): Sets the dimensions of the box and updates its volume.
- updateVol(): Updates the volume of the box based on its dimensions.
- decreaseSizeBox(): Adjusts the box dimensions according to the remaining orders with decreasing the sizes if it’s more than needed.

Order Class Functionalities:
The Order class represents a single order with its dimensions. The following methods are implemented within this class:
- Order(double l, double m, double s): Constructor that initializes the order with the given dimensions.

Our algorithm is designed to optimize the packing of orders into boxes and uses multiple methods to achieve its purpose. Let's examine the step-by-step explanation of the algorithm's methodologies.

The algorithm begins by populating the environment with an inventory of boxes and their respective positions. It also specifies the possible box resizing dimensions. These dimensions are separated for corresponding edges, which are largest edge, medium edge, shortest edge.

The algorithm then enters a phase of iterative improvement. The algorithm attempts to resize and redistribute orders between box pairs to find potential improvements to the present packing solution. For each box pair, the algorithm attempts to resize the smaller box and transfer orders from the larger box to the smaller box, evaluating the packaging volume that results. This idea is came from creating an improving direction with increasing the size of smaller box to transfer the orders from larger box. Thus, each increase in the edge sizes of smaller box increases the total volume as much as the number of orders that were chosen to place in that box times the amount of increase in the volume of that box type. However, since we have increased the volume of smaller box, more amount of orders become able to transfer to that box from the larger box. So, because of transferring the orders to a smaller box, total volume decreases as much as number of orders which were transferred times the difference between the volume of larger box and the updated volume of smaller box. So that, if the increase in the volume of the smaller box times the number of orders that are placed in that box is less than the the number of orders which are transferred from larger box to smaller box times difference between the volume of the larger box and smaller box, then the total volume is decreased.

In the algorithm, the expansion of the smaller box is performed greedily. The algorithm cycles through every conceivable combination of larger dimensions up to the size of the largest box. The algorithm selects the optimal combination based on the updated volume in both categories.

After resizing the smaller box, the algorithm transfers orders between boxes according to their dimensions. If each order from the larger box can fit into the smaller resized box, the order is transferred and the larger box is emptied. This operation is carried out by the changeBox() method from the Environment class.

After transferring the orders, the algorithm uses the decreaseSizeBox() method to reduce the dimensions of the larger box to the smallest size necessary to accommodate the remaining orders.

The algorithm stores the packaging volumes in a TreeMap and chooses the pair of containers with the smallest combined volume. It then revises the box inventory to include these enhanced boxes and recalculates the total volume of packing.

This procedure is repeated for each pair of boxes in the environment until no further packing solution enhancements are possible.

The algorithm optimizes the packaging of orders in boxes by iteratively examining potential improvements, resizing the boxes, transferring orders between boxes, and recalculating the total packing volume. The algorithm is anticipated to substantially improve the packaging and shipping process's efficiency and effectiveness, resulting in cost savings and increased customer satisfaction.

Pseudocode of our heuristic algorithm;

1. Read input data from files: large.txt, medium.txt, short.txt
2. Create empty lists for large, medium, and small orders
3. Loop through each file and populate the corresponding order list with data from the file
4. Sort the orders in each list by size
5. If there will be N boxes with different sizes, create a list of boxes, with the first N-1 boxes initialized with size 0, and the last box having sizes equal to the upper bound of each of the edge categories(largest,medium,small).
6. Create an initial environment object with the box list
7. Loop until the environment is not improvable: a. For each pair of boxes, create a new environment object with the boxes swapped b. Try to improve the new environment by moving orders between boxes with adjusting box sizes and chooses the new sizes of that pair of boxes which creates minimum volume with that box pairs and the orders inside of these boxes c. Add the new environment to a map, with its volume as the key
8. Set the environment to the one with the smallest volume in the map
9. Update the volumes of the boxes in the environment.

10. If the total volume isn't different from the beginning it sets isImprovable attribute of environment to false, loop stops and returns the latest environment.
11. If the total volume has decreased, it continues to re-run the bestSol method with the updated environment until isImprovable becomes false.
12. Output the total volume and details of each box, including the number of orders, largest edge, medium edge, and smallest edge

4. Results

In order to reach the solution we needed to search, learn and use various of new softwires and programs which will support us solving the linear model, non-linear model and algorithm.

4.1 Excel Solver for Second Step: Non-Linear Model

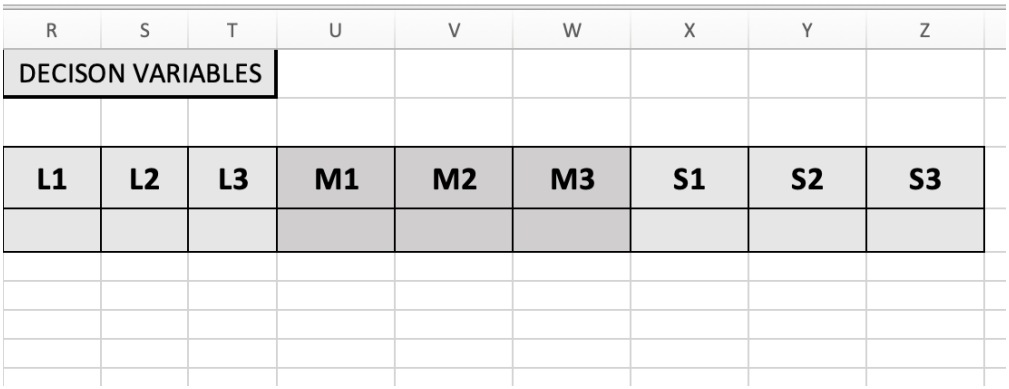
In order to solve the model we developed, shown in chapter 3.3.2., in excel solver, it was written in the excel sheet in the following format, respectively:

1. The table with the index of 10 order - 3 boxes was prepared for the o_{ij} decision variable, which indicates which order will be in the box. Then the totals for each order row for constraints in the model are added to the Sum column.

M	N	O	P	Q
	Oij			
SUMS	İ-J	1	2	3
3	1	1	1	1
3	2	1	1	1
3	3	1	1	1
3	4	1	1	1
3	5	1	1	1
3	6	1	1	1
3	7	1	1	1
3	8	1	1	1
3	9	1	1	1
3	10	1	1	1

Figure 2. Excel Solver Screenshot "o_{ij}" Decision Variable

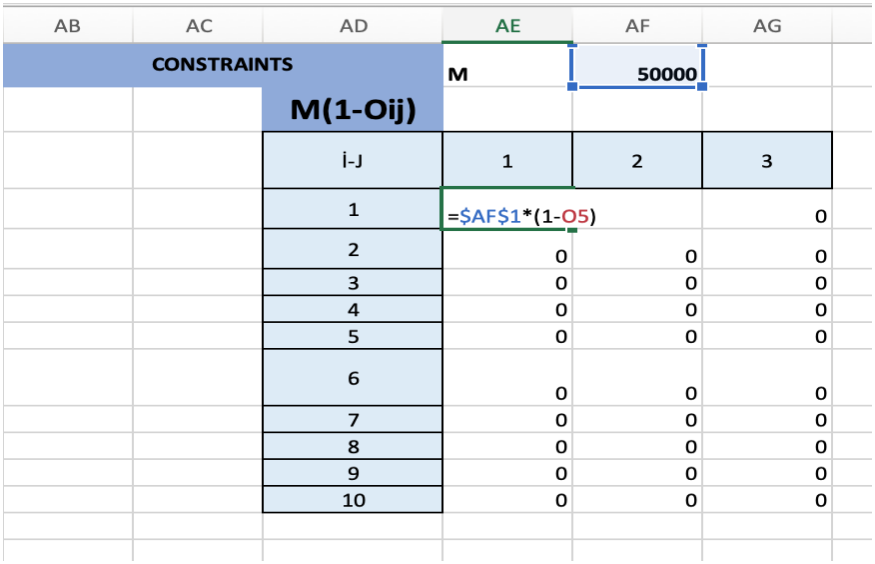
2. Decision variables representing the dimensions of the boxes L_j, M_j, S_j were added empty to fill after the solution.



R	S	T	U	V	W	X	Y	Z
DECISON VARIABLES								
L1	**L2**	**L3**	**M1**	**M2**	**M3**	**S1**	**S2**	**S3**

Figure 3. Excel Solver Screenshot "L_j, M_j, S_j" Decision Variables

3. $M(1-o_{ij})$ values were created by using the previously written o_{ij} variables using constraints ... which compares order sizes with box sizes.



AB	AC	AD	AE	AF	AG
	CONSTRAINTS		**M**	**50000**	
		M(1-Oij)			
		İ-J	1	2	3
		1	=$AF$1*(1-O5)		0
		2	0	0	0
		3	0	0	0
		4	0	0	0
		5	0	0	0
		6	0	0	0
		7	0	0	0
		8	0	0	0
		9	0	0	0
		10	0	0	0

Figure 4. Excel Solver Screenshot $M(1-o_{ij})$ for Constraints

4. The order lengths are pasted into the gray table. The order length and the previously created variables L_j, M_j, S_j and $M(1-o_{ij})$ were used to use the constraint for each length to be used in the excel solver. $L_j - l_i + M(1-o_{ij}) \geq 0$, $M_j - m_i + M(1-o_{ij}) \geq 0$, $S_j - s_i + M(1-o_{ij}) \geq 0$.

AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW
Lİ-li+M(1-Oij)					**Mİ-mi+M(1-Oij)**					**Sİ-si+M(1-Oij)**			
Largest Edge	**L1**	**L2**	**L3**		**Medium Edge**	**M1**	**M2**	**M3**		**Smallest Edge**	**S1**	**S2**	**S3**
20	=$R$4-AJ4+AE4	-20	-20		6	-6	-6	-6		5	-5	-5	-5
18	-18	-18	-18		9	-9	-9	-9		4	-4	-4	-4
21	-21	-21	-21		19	-19	-19	-19		18	-18	-18	-18
34	-34	-34	-34		12	-12	-12	-12		5	-5	-5	-5
20	-20	-20	-20		13	-13	-13	-13		5	-5	-5	-5
20	-20	-20	-20		6	-6	-6	-6		5	-5	-5	-5
33	-33	-33	-33		12	-12	-12	-12		3	-3	-3	-3
26	-26	-26	-26		16	-16	-16	-16		5	-5	-5	-5
20	-20	-20	-20		6	-6	-6	-6		5	-5	-5	-5
37	-37	-37	-37		27	-27	-27	-27		2	-2	-2	-2

Figure 5. Excel Solver Screenshot Order and Box Sizes comparison Constraints

5. Finally, the objective function was created in three stages. First, 3 volumes were obtained by multiplying the box sizes. Then, the volumes obtained were multiplied by the o_{ij} variables on a box basis to calculate the volumes occupied by the order. Lastly all of them are summed to obtain the objective function.

OBJECTIVE FUNCTION	
Box Volume Lj * Mj * Sj	
Box1	=R4*U4*X4
Box2	0
Box3	0
Objective oij * Lj * Mj * Sj	
J1	0
J2	0
J3	0
	0

Figure 6. Excel Solver Screenshot Box Volumes

OBJECTIVE FUNCTION	
Box Volume Lj * Mj * Sj	
Box1	0
Box2	0
Box3	0
Objective oij * Lj * Mj * Sj	
J1	0
J2	0
J3	0
	=SUM(B10:B12)

Figure 7. Excel Solver Screenshot Total Volume

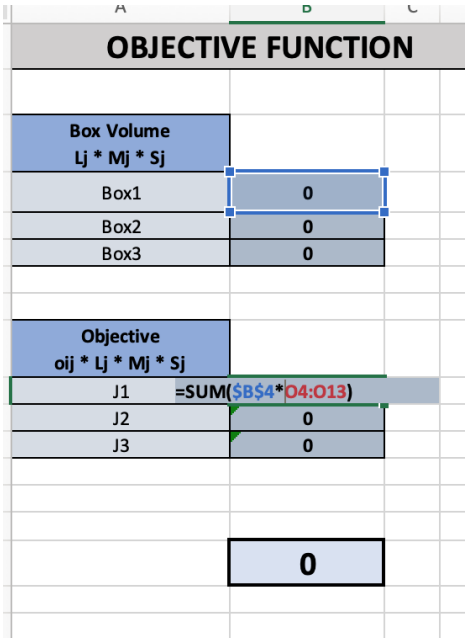


Figure 8. Excel Solver Screenshot Total Volume of Orders Chosen for Specific Box Types

6. The model was written manually for the Excel solver screen. First of all, the cell where we wrote the objective function is selected for the 'Set Objective' cell. Then the o_{ij} and L_j, M_j, S_j variables are selected for the 'By Changing Variable Cells' cell. Lastly, the $L_j - l_i + M(1-o_{ij}) \geq 0$, $M_j - m_i + M(1-o_{ij}) \geq 0$, $S_j - s_i + M(1-o_{ij}) \geq 0$ constraints mentioned in the third step were added and corrected to be greater than zero; the Sum column mentioned in first step is set to 1; o_{ij} variables are set to binary.

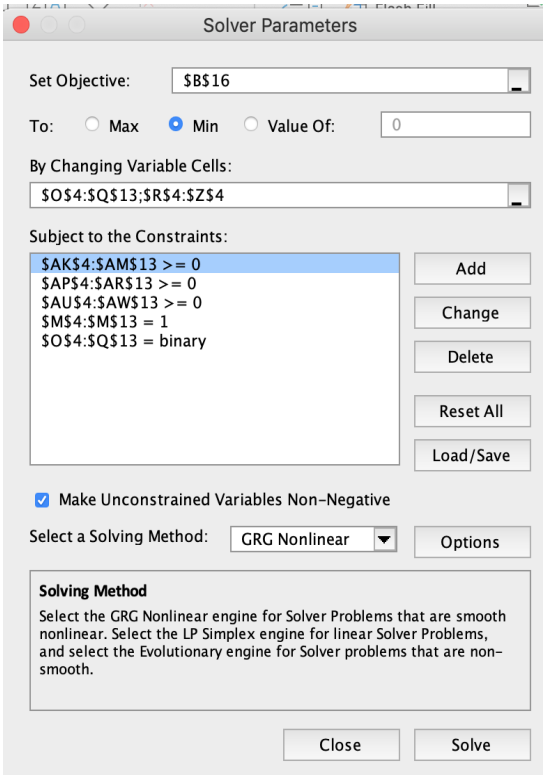


Figure 9. Excel Solver Screenshot for Solving the Model

7. Since the model we developed is a complex problem such as Mixed-integer non-linear programming (MINLP), we adjusted the Constraint Precision and Integer Optimality values to approach the optimal result in Excel Soler. We aimed to meet Constraints more precisely by lowering the Constraint Precision. Likewise, we increased the optimal sensitivity by lowering the Integer Optimality value and aimed to find solutions closer to the optimal. However, lowering both values increased the solution times individually.

Proof:
Firstly, when Constraint Precision: 0.1 Integer Optimality: 1 shown in Figure 10 is taken, In Figure 11 it is written that it solved in 184789436,354 seconds, but according to our manual measurement, it gave the answer in about 130 seconds. Objective function value is 1.01627597064138E-12, which corresponds to 0.000000000010162759706 and the given values were resolved in a short time. However, it was concluded that it did not give the correct result.

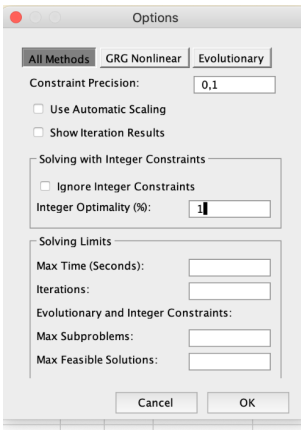
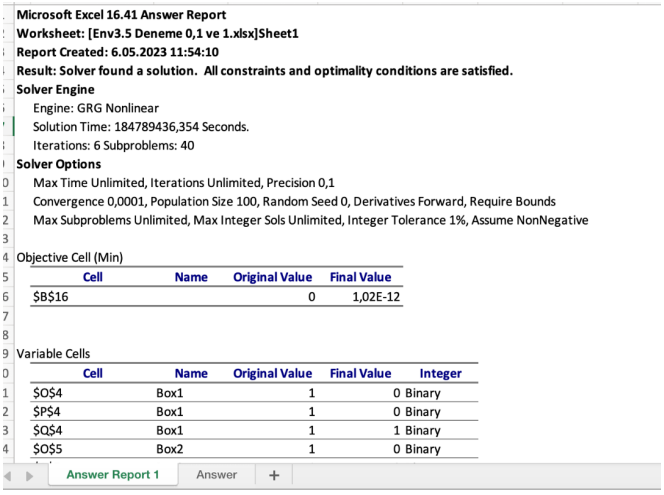


Figure 10. Excel Solver Screenshot Integer Optimality and Precision Values



Microsoft Excel 16.41 Answer Report
Worksheet: [Env3.5 Deneme 0,1 ve 1.xlsx]Sheet1
Report Created: 6.05.2023 11:54:10
Result: Solver found a solution. All constraints and optimality conditions are satisfied.
Solver Engine
Engine: GRG Nonlinear
Solution Time: 184789436,354 Seconds.
Iterations: 6 Subproblems: 40
Solver Options
Max Time Unlimited, Iterations Unlimited, Precision 0,1
Convergence 0,0001, Population Size 100, Random Seed 0, Derivatives Forward, Require Bounds
Max Subproblems Unlimited, Max Integer Sols Unlimited, Integer Tolerance 1%, Assume NonNegative

Objective Cell (Min)

Cell	Name	Original Value	Final Value
$B$16		0	1,02E-12

Variable Cells

Cell	Name	Original Value	Final Value	Integer
$O$4	Box1	1	0	Binary
$P$4	Box1	1	0	Binary
$Q$4	Box1	1	1	Binary
$O$5	Box2	1	0	Binary

Figure 11. Excel Solver Screenshot Solver Results and Performance

Then, to increase the affordability percentage of the constraints and to reach the optimal result, the values were adjusted as Constraint Precision: 0.0001 Integer Optimality: 0.001 as shown in the Figure 12 and solved again. According to our manual measurement, Solver solved the model in about 633 seconds with the new values, but found the objective function value of 13617.3672542409. It has been concluded that with smaller constraint precision and integer optimality values, it gives more optimal answers in a longer time.

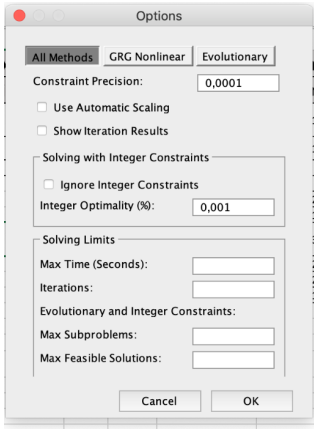


Figure 12. Excel Solver Screenshot Lower Integer Optimality and Precision Values

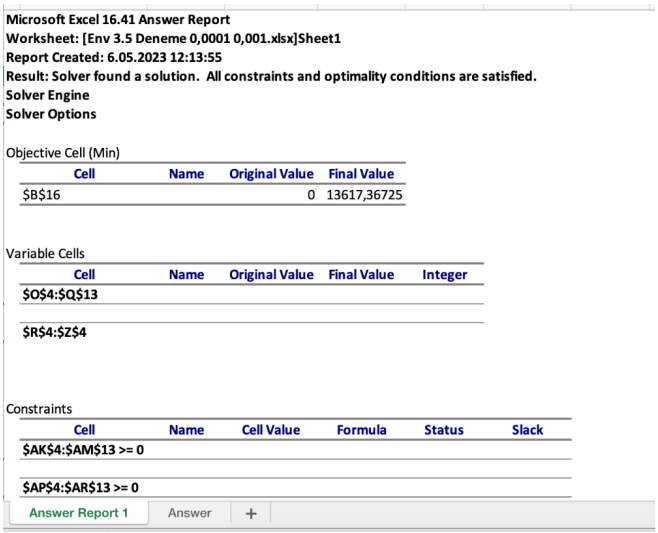


Figure 13. Excel Solver Screenshot Solver Results and Performance with Lower Integer Optimality and Precision Values

4.1.2 Excel Solver Results

Constraint Precision 0,0001 and Integer Optimality 0,001

Chapter 4.1 constraint precision 0.0001 and integer optimality 0.0001 in step 7 were run on 10 selected data sets, but it was concluded that it did not get optimal results every time.
The proof that Environment 0 selected among them does not give the optimal result is as follows:

According to the result in Figure 14, as you can see from the O_{ij} table, the value of O_{73} is given as 1, that is, the 7th Order is in the 3rd box. However, the longest side of the 7th Order is 33, while the longest side of the 3rd Box is 30. This shows that the 7th Order does not fit in the 3rd Box. The result leads to the conclusion that the result of excel Solver is not optimal and constraints are not met.

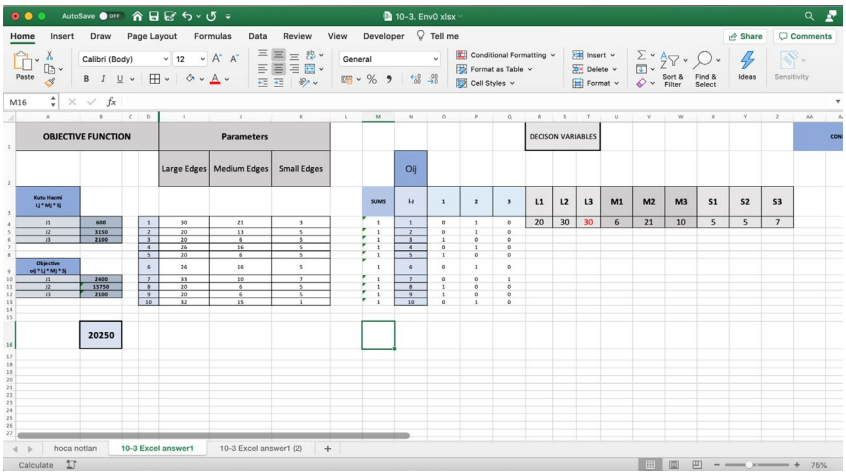


Figure 14. Excel Solver Screenshot a Part of the Solution of the Solver

The 5 randomly selected environment results tested are shared in the Table 1:

Table 1. Excel Solver Results of Different Problem Instances

Environments	Excel Solver Results
Env0	20250
Env1	37678
Env2	13782
Env3	15955
Env4	29385

As you can see, the objective function results of the excel solver do not meet the constraints while 5 Environments run. For example, in Environment 2, the smallest length of the 3rd Box is 0.

As a result, the values obtained with Constraint Precision 0,0001 and Integer Optimality 0,001 values did not give the optimal answer, so the values were changed. New results were obtained by arranging 15 different environments, including the 5 environments solved above, as Constraint Precision 0,000001 and Integer Optimality 0,0001 which is shown in Figure 15 .

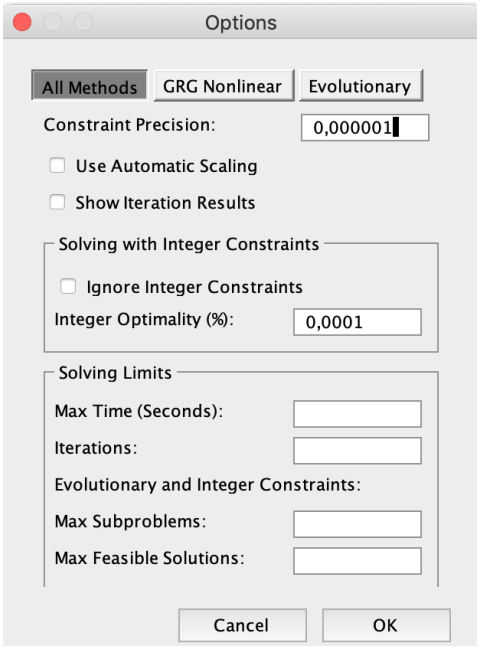


Figure 15. Excel Solver Screenshot Lower Integer Optimality and Precision Values 2

Arranged Excel Solver results will be explained in Final Result Section by comparing them with Java and Neos results.

4.2 Neos and Gams for Second Step: Non-Linear Model

To solve the non-linear model, Gams Studio was first downloaded to our computer and an academic license was obtained to use the program. However, since the license cannot solve small size data, we decided to use Neos' online solvers. We uploaded the Gams code file we have to the Baron solver of Neos, which solves the mixed integer non-linear program, and obtained the answers from Neos. However, here too, the big problems could not be solved because it exceeded the time limit while solving it. Only small size problems could be solved from Neos. We also solved the same size problems from Excel Solver and compared the results. Then, we developed and used a heuristic algorithm, since the problem at hand was larger in size.

```
Solution     = 102029.999999998  found at node 78
Best possible = 102019.79802
Absolute gap  = 10.2019799977279  optca = 1E-9
Relative gap  = 9.9990002918045E-5  optcr = 0.0001


---- EQU e7

    LOWER    LEVEL    UPPER   MARGINAL

b1    .     36.000    +INF      .
b2    .     21.000    +INF      .
b3    .     48.000    +INF      .

---- EQU e8

    LOWER    LEVEL    UPPER   MARGINAL

b1    .     28.000    +INF      .
b2    .     19.000    +INF      .
b3    .     40.000    +INF      .

---- EQU e9

    LOWER    LEVEL    UPPER   MARGINAL

b1    .      7.000    +INF      .
b2    .     18.000    +INF      .
b3    .     20.000    +INF      .
```

Figure 16. Neos Solver Result Screenshot

```
- Matrix    [min, max] : [ 1.000E+00, 1.000E+10] - Zero values observed as well
* The model exceeds the demo license limits for nonlinear models of more than 1000 rows or
  As an academic user, you can submit large models to the NEOS service for free (see https:
  Alternatively, feel free to contact sales@gams.com to discuss your options
* Status: Terminated due to a licensing error
*         License file: C:\Users\Kerem\Documents\GAMS\gamslice.txt
*         Inspect listing file for more information.
- Job new1.gms Stop 03/17/23 16:57:34 elapsed 0:00:00.027
```

Figure 17. Gams Solver Screenshot

4.3 Java for Algorithm

To solve the algorithm, we needed a coding environment that we were familiar with before. For this, we preferred to use Java. We developed a heuristic algorithm in Java and used this algorithm for large size problems. Although it gives much faster results than other solvers, some problems in small size have deviations from the optimal result. In our report, we took the average of the optimality gaps we detected in the algorithm and added them.

You can see the output of the Algorithm with 68 orders and 10 boxes which are obtained from the data given by Xxx Logistics.

Table 2. Algorithm Solution of 68 Orders 10 Boxes

Ordercount: 7 Largest Edge: 37.0 Medium Edge: 28.0 Shortest Edge: 3.0 Box ID: b0
Ordercount: 10 Largest Edge: 20.0 Medium Edge: 17.0 Shortest Edge: 8.0 Box ID: b1
Ordercount: 1 Largest Edge: 40.0 Medium Edge: 35.0 Shortest Edge: 1.0 Box ID: b2
Ordercount: 5 Largest Edge: 38.0 Medium Edge: 28.0 Shortest Edge: 5.0 Box ID: b3
Ordercount: 1 Largest Edge: 39.0 Medium Edge: 30.0 Shortest Edge: 13.0 Box ID: b4
Ordercount: 21 Largest Edge: 20.0 Medium Edge: 9.0 Shortest Edge: 5.0 Box ID: b5
Ordercount: 1 Largest Edge: 21.0 Medium Edge: 19.0 Shortest Edge: 18.0 Box ID: b6
Ordercount: 17 Largest Edge: 35.0 Medium Edge: 17.0 Shortest Edge: 5.0 Box ID: b7
Ordercount: 4 Largest Edge: 35.0 Medium Edge: 27.0 Shortest Edge: 7.0 Box ID: b8
Ordercount: 1 Largest Edge: 48.0 Medium Edge: 40.0 Shortest Edge: 20.0 Box ID: b9

233683.0 Total Volume

```
Execution time:2.631304
7 37.0 28.0 3.0 b0
10 20.0 17.0 8.0 b1
1 40.0 35.0 1.0 b2
5 38.0 28.0 5.0 b3
1 39.0 30.0 13.0 b4
21 20.0 9.0 5.0 b5
1 21.0 19.0 18.0 b6
17 35.0 17.0 5.0 b7
4 35.0 27.0 7.0 b8
1 48.0 40.0 20.0 b9
233683.0 total volume
```

Figure 18. Algorithm Result with 68 Orders 10 Boxes

In addition, below you can see the run-time values of the different size of randomly generated problems. In our mixed-integer nonlinear model, problem sizes are depending on the amount of orders and the number of available boxes with different sizes. In a sample problem with N-orders and k-box types, our model consists of N*k binary variables and 3*k continuous variables.

- Execution time: 673.1554326 seconds 1000 orders 100 boxes (problem size of 100000 binary variables and 300 continuous variables)
- Execution time: 352.4037074 seconds 1000 orders 68 boxes (problem size of 68000 binary variables and 204 continuous variables)
- Execution time: 55.1165279 seconds 1000 orders 30 boxes (problem size of 30000 binary variables and 90 continuous variables)
- Execution time: 22.3729194 seconds 1000 orders 20 boxes (problem size of 20000 binary variables and 60 continuous variables)
- Execution time: 12.811688 seconds 1000 orders 15 boxes (solved 3 times and the mean is written) (problem size of 15000 binary variables and 45 continuous variables)
- Execution time: 6.79053567 seconds 1000 orders 10 boxes (solved 3 times and the mean is written) (problem size of 10000 binary variables and 30 continuous variables)
- Execution time: 8.15922283 seconds 200 orders 20 boxes (solved 3 times and the mean is written) (problem size of 4000 binary variables and 60 continuous variables)
- Execution time: 3.96086227 seconds 200 orders 15 boxes (solved 3 times and the mean is written) (problem size of 3000 binary variables and 45 continuous variables)
- Execution time: 1.81056907 seconds 200 orders 10 boxes (solved 3 times and the mean is written) (problem size of 2000 binary variables and 30 continuous variables)
- Execution time: 1.33370097 seconds 100 orders 10 boxes (solved 3 times and the mean is written) (problem size of 1000 binary variables and 30 continuous variables)
- Execution time: 246.3202423 seconds 68 orders 68 boxes (problem size of 4624
- binary variables and 204 continuous variables)
- Execution time: 26.2029533 seconds 68 orders 30 boxes (problem size of 2040
- binary variables and 90 continuous variables)
- Execution time: 7.0039728 seconds 68 orders 20 boxes (problem size of 1360

- binary variables and 60 continuous variables)
- Execution time: 1.1705193 seconds 68 orders 10 boxes (problem size of 680
- binary variables and 30 continuous variables)
- Execution time: 0.4228647 seconds 68 orders 5 boxes (problem size of 300
- binary variables and 15 continuous variables)
- Execution time: 0.3488401 seconds 34 orders 10 boxes (problem size of 340
- binary variables and 30 continuous variables)
- Execution time: 0.1090495 seconds 34 orders 5 boxes (problem size of 170
- binary variables and 15 continuous variables)
- Execution time: 0.0746078 seconds 17 orders 5 boxes (problem size of 85
- binary variables and 15 continuous variables)
- Execution time: 0.0291601 seconds 17 orders 3 boxes (problem size of 51
- binary variables and 9 continuous variables)
- Execution time:0.0288277 seconds 15 orders 3 boxes (problem size of 45
- binary variables and 9 continuous variables)
- Execution time: 0.0187489 seconds 10 orders 3 boxes (problem size of 30
- binary variables and 9 continuous variables)
- Execution time: 0.0386097 seconds 20 orders 3 boxes (problem size of 60
- binary variables and 9 continuous variables)
- Execution time: 0.0581197 seconds 20 orders 4 boxes (problem size of 80
- binary variables and 12 continuous variables)
- Execution time: 0.0954053 seconds 20 orders 5 boxes (problem size of 100
- binary variables and 15 continuous variables)
- Execution time: 0.0697145 seconds 15 orders 5 boxes (problem size of 75
- binary variables and 15 continuous variables)
- Execution time: 0.0483631 seconds 15 orders 4 boxes (problem size of 60
- binary variables and 12 continuous variables)
- Execution time: 0.0326716 seconds 10 orders 4 boxes (problem size of 40
- binary variables and 12 continuous variables)
- Execution time: 0.1048102 seconds 30 orders 5 boxes (problem size of 150
- binary variables and 15 continuous variables)
- Execution time: 0.0537614 seconds 10 orders 5 boxes (problem size of 50
- binary variables and 15 continuous variables)
- Execution time: 0.0744879 seconds 30 orders 4 boxes (problem size of 120
- binary variables and 12 continuous variables)
- Execution time: 0.0458658 seconds 30 orders 3 boxes (problem size of 90
- binary variables and 9 continuous variables)

4.4 Gurobi for First Step: Linear Model

To solve the algorithm, we needed a coding environment that we were familiar with before. Since this model is a linear model, we preferred to use the Gurobi library to solve the model. By solving the first stage of the problem from Gurobi, we determined the dimensions of the rectangular prisms with the minimum volume that they can enter separately for each order. In our second step, we used our results in the form of rectangular prisms as the measurements of the orders, which we assumed to be in the form of the same rectangular prism.

4.5 Final Result

Comparison of the Java, Neos and Excel Solver:

The results of Java, Neos and Excel Solver are shown in Table 3 as follows :

Table 3. Result Comparison of Problems with 10 Orders 3 Boxes

	Java Results	**Neos Solution Results**	**Excel Solver Results (Taking the right ones)**
Data 1	20902	20901,9466	20902
Data 2	30940	30940	**39582**
Data 3	40970	40970	**48690**
Data 4	37869	37869	37869
Data 5	31590	30029.9398	**34438**
Data 6	71139	71139	**72840**
Data 7	25294	25294	**26096**
Data 8	30099,999999998166	26412	**26676**
Data 9	33642	33642	**38192**

Data 10	28905	28518	**30458**
Data 11	57266	54420	**66692**,72
Data 12	27625	27624.94	**27625**
Data 13	27980	27979,9251	27980
Data 14	27625	27624,94	27625
Data 15	14598	14598	**15108**

The 15 Environments which we run in Java, Neos, and Excel Solver gave answers that are not exactly optimal. We see from the results that Excel Solver in Table 3 gave the same or greater than Java and Neos results.

Since it is not certain that we achieved an optimal result in all 3 Solver methods we used, we wrote the sub-optimality according to the Neos or Excel value that gave the lowest result. So the real optimality gap will be greater than or equal to the optimality gap that we have found with the formula below. The formula is shown in Figure 19 as follows:

$$\frac{(Java\ Result - Min(Excel\ Result,\ Neos\ Result)}{Min(Excel\ Result,\ Neos\ Result)}$$

Figure 19. Optimality Gap Calculation of Algorithm

According to formula shown in Figure 19 , The comparison results are shown in Table 4 as follows:

Table 4. Optimality Gaps

	Comparison
Data 1	0,0%
Data 2	0,0%
Data 3	0,0%
Data 4	0,0%
Data 5	5,2%
Data 6	0,0%
Data 7	0,0%
Data 8	14,0%
Data 9	0,0%
Data 10	1,4%
Data 11	5,2%
Data 12	0,0%
Data 13	0,0%
Data 14	0,0%
Data 15	0,0%

4.6 Verification and Validation

To guarantee the reliability and accuracy of a mathematical model, verification and validation are essential processes that must be taken. The equations of the model have been included into the system in a manner that is accurate and correct. The assumptions that are used in the model are rational and appropriate for the problem that is being modeled. It is reliable and gives outcomes that are never inconsistent. We tried to simplify things by linearizing the model, and once we did so, we were able to solve the model using the data we had. While this was happening, we tested it out on a few smaller orders and found that it produced accurate results.
To ensure the accuracy and robustness of our algorithm, we conducted multiple experiments with various input datasets, including both synthetic and real-world problems with varying order counts and bin sizes. These tests were designed to validate the algorithm's ability to manage a variety of problem sizes and characteristics, as well as its packaging and computational efficiency.
Using NEOS Solver and Excel Solver, the results of the algorithm were compared to those of a two-stage mixed integer nonlinear model. Particularly for 10-3 dimensional (30 binary variables, 9 continuous variables) problem examples, our algorithm provided exact solutions that matched both NEOS Solver and Excel Solver's results. This demonstrates that our algorithm is effective at locating optimal solutions for these specific problem scales.

For other problem sizes, our algorithm's performance exhibited a relatively narrow optimality range, with the difference between the obtained solutions and those of NEOS Solver and Excel Solver usually very close. Based on our 15 sample problems which contains 10 orders and 3 box we have obtained 11 solutions with approximately %0,0001 or less optimality gap, 1 solution with %1.4 optimality gap, 2 solutions with %5.2 optimality gap and one solution with %14 optimality gap and it can also solves problems as large as a mixed-integer nonlinear model that contains 100000 binary variables and 100 continuous variables. This demonstrates that

our algorithm can usually produce near-optimal solutions for a wide range of problem instances while maintaining a reasonable level of computational efficiency.

*In addition, our algorithm proved to be an accurate method for one-dimensional and two-dimensional problems and provided optimal solutions for examples of such problems. This demonstrates the adaptability and dependability of our algorithm when dealing with different dimensions of the Box Packing Problem.
Overall, the verification and validation process demonstrated that our algorithm effectively reduces the total number of crates used for order packaging while maintaining efficient search space exploration. Our algorithm contributes to more efficient and environmentally responsible supply chain practices by minimizing the total volume of cartons used.

5. Implementation

There is no requirement for additional data or resources for deployment. We have enough information that our organization has provided to us. We can choose from orders of the cardboard type. For each order, a new sort of cardboard will be used. Because there are just a few different kinds of cardboard, we built this model to determine if the products in an order will fit in the cardboard that will be utilized in that particular order. The ideal answer will then be obtained when we run this model.
After the greedy algorithm we developed, we created a solution that will overcome the problem we want to solve. Our solution can be easily adapted to daily life and gives visible results. As we mentioned earlier, our main focus in this semester is to simplify the model we developed in IE401 and reduce computational complexity. In this direction, we created a two-stage model in which we developed our own algorithm to satisfy the needs of the company. Xxx company can run the two-stage method we have developed by putting the data it wants into it. In the first stage, the rectangular box in which the items will be located is minimized for the second stage and turns it into an input. The company can solve the problem by reaching the optimal number of boxes specified at the end of the second stage. Our solution will ensure that the problem reaches a conclusion without requiring any resources when the necessary data is provided.
Our method computes the results with the expected speed and ease in accordance with the targeted and planned situation. The problem that may be encountered during the application of this method is that in the presence of very large and complex data, it may deviate from the optimal result. In smaller sizes, we observed that the optimality gap was approximately 1.72% on average when we selected 10 random orders and ran them in 3 boxes.

6. Conclusion

6.1 Summary of the Work Done

As previously indicated, the project's main focus was on issues with suitable packaging for online sales. A literature review, investigation, and in-depth handling and analysis of the problem were all done throughout the problem's solution phase. According to the principle of having the least total volume, we produced the orders for this project as rectangular prisms. Along with the larger challenges, we also used the model we developed for it to smaller issues. Input data, iterative improvement, greedy resizing, box resizing, order transfer, and optimization techniques are planned, and a heuristics model built on top of the optimized box packing algorithm is created and programmed in Java. The problem is thoroughly evaluated.

6.2 Conclusion

To determine the packaging orders into boxes, we designed the optimum box packing algorithm and used its output data. We employed iterative improvement, greedy search, and local search to create the optimized box packing algorithm. The goal of performing these searches is to identify the order packaging into boxes that comes the closest to being ideal. We sought to reduce the overall volume of orders that would be placed in a constrained number of boxes of various sizes. To tackle the issue, we examined the supplied data, and we programmed the algorithm in Java. To sum up, after collecting all the necessary data and information from the company and visiting the company, the problems were identified and made organized and resolved by using the necessary assumptions and creating the necessary algorithm.

References

https://neos-server.org/neos/solvers/index.html
https://github.com/kt980610/MinimumVolumeBoxes (Source code of algorithm)

A Global Optimization Method for Packing Problems

Hu, N.-Z., Tsai, J.-F., & Li, H.-L., A global optimization method for packing problems. *Journal of the Chinese Institute of Industrial Engineers*, *19*(1), 75–82, 2002. https://doi.org/10.1080/10170660209509185

A New Mathematical Model for a 3D Container Packing Problem

Ocloo, V. E., Fügenschuh, A. R., & Pamen, O. M. (2020). *ANEWMATHEMATICALMODELFORA3D containerpackingproblem - core*. Retrieved November 26, 2022, from https://core.ac.uk/download/pdf/287791673.pdf

Biographies:

Kerem Tuna: Senior year undergraduate industrial engineering student at Ozyegin University.

Onat Gürcanok: Senior year undergraduate industrial engineering student at Ozyegin University.

Şevval Berksöz: Recent graduate from Ozyegin University industrial engineering undergraduate program.

Efe Doğruer: Senior year undergraduate industrial engineering student at Ozyegin University.

Alara Kumru: Recent graduate from Ozyegin University industrial engineering undergraduate program.

Berfin Kondakcı: Recent graduate from Ozyegin University industrial engineering undergraduate program.