Klö.iı8

**Koç University**

**College of Engineering**

**INDR 491 Engineering Design Final Project Report**

**Spring 2020**

***Allocation of Orders to the Sorter***

***at Trendyol’s Warehouse***

**Information about the team:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **ID** | **Email** | **Phone** |
| 1 : Berhan Karabay | 0053728 | bkarabay15@ku.edu.tr | 05069506794 |
| 2 : Can Taşal | 0053726 | ctasal15@ku.edu.tr | 05349440351 |
| 3 : Caner Kayalı | 0053759 | ckayali15@ku.edu.tr | 05309386464 |
| 4 : Ceylin Kolan | 0053933 | ckolan15@ku.edu.tr | 05396521214 |

**Project Supervisor: Ahmet Çınar**

**Academic Supervisor:**

|  |  |
| --- | --- |
| **Name** | **Signature** |
| Sibel Salman |  |

Table of Contents

[1. Brief System Description 3](#_Toc41659913)

[2. System Analysis 3](#_Toc41659914)

[2.1. Problem Definition 3](#_Toc41659915)

[2.2. Objective and Scope of the Project 4](#_Toc41659916)

[2.3. Analysis and Interpretation of the Data 4](#_Toc41659917)

[3. Literature Review and Sector Analysis 4](#_Toc41659918)

[3.1. Literature Review 4](#_Toc41659919)

[3.2. Sector Analysis 6](#_Toc41659920)

[4. System Design 6](#_Toc41659921)

[4.1. System Architecture 6](#_Toc41659922)

[4.2. Model Description 8](#_Toc41659923)

[4.3. Solution Methodologies and Validation 11](#_Toc41659924)

[4.4. Consideration of Alternatives 12](#_Toc41659925)

[4.5. Scenario / What-If Analysis 13](#_Toc41659926)

[5. Implementation 15](#_Toc41659927)

[6. References 16](#_Toc41659928)

[7. Appendix 17](#_Toc41659929)

# Brief System Description

Trendyol was founded in 2010 by Demet Mutlu. It is the fastest growing e-commerce company in Turkey and the MENA region that serves over 25 million customers and employs over 2000 people. Their vision is to make fashion affordable and accessible in Turkey. Their mission is to provide services for their customers to shop whatever they want, whenever they want which includes more than 100 domestic and foreign brands, including their own brand Trendyolmilla.

Technology is the key element in the services of Trendyol. They have announced an investment and strategic partnership with Alibaba Group in 2018 in order to improve themselves and become a global e-commerce company. Trendyol is the market leader with 20 million textile visitors monthly and 75 million products sold per year. It is the most downloaded mobile commerce application with 15 million downloads. One of their strengths is the variety of the products they sell which has more than 1500 product categories. In Turkey, 3 out of 4 women who purchased online have shopped on Trendyol. Trendyol has two Research Development centers in Istanbul which ensures Trendyol to improve continuously.

Trendyol is growing rapidly by including new brands and new categories and constantly changing itself to satisfy their customers. Recently, Trendyol has founded its own cargo company. Trendyol has 3 warehouses in the Istanbul region and the scope of this project is going to involve the one located in Gebze. Its service begins with a customer’s online purchase and continues with the delivery of the product on the stated date with highest customer satisfaction. Trendyol provides free return of products and values the customer opinion mostly which is observed by the comment window of each product sold on the website.

The warehouse system of Trendyol is well structured and is concerned with their sales channels called stock through and flow through. Another sales channel is called marketplace but is outside the scope of this project since it is not concerned with the warehouses. Stock through products are stored in the warehouses for certain periods of time before being sent to customers, whereas the flow through products are not stored and kept in the warehouse as inventory. Contracted sellers of products send the ordered products to the Trendyol warehouse and upon reception, the products are initially divided into two categories. If an order includes only one product, they are separated from the others. Additionally, all products that have been ordered as part of a multi item order are sent to the sorter for further processes until they are shipped to the customers finally.

# System Analysis

## Problem Definition

The current system at Trendyol was initially studied on February 7, 2020 during a visit to the warehouse. With help from Trendyol employees including the supervisor, the group had a chance to have an overview and a visualization of the system.

The project is related to products that are in the flow-through category. These can be described as the ones that are sent to the warehouse and then sent to customers from there. The intermediate procedures include the sorting of the products. The material flow is designed such that all products that are included in orders with multiple items have to go through the sorter at the warehouse. The sorter has several chutes which help the items to be allocated into different batches.

Additionally, the chutes can accept more than one batch, provided that they belong to the same boutique. In other words, if a certain batch is assigned to a chute, then other batches from that boutique must also be assigned to that chute. After the completion of the sorting process, meaning that the products in a certain batch have been assigned to a single chute, they are then directed to the packaging area. As the project is concerned only with the sorting process at the warehouse, further procedures such as packaging are beyond our interest.

The sorter includes a limited number of chutes resulting in inefficiency when the number of batches exceeds the number of chutes. In order to separate batches and transfer them to packaging, a single batch should be assigned to a single chute. However, it will not be possible if the number of batches exceeds the number of chutes. Although increasing the number of chutes seems to be a solution, high costs of machines and variability in number of batches through boutiques is eliminating this option. Therefore, the scope of this problem is sorting operation starting with gathering all batches in sorter until transferring them to packaging.

## Objective and Scope of the Project

The main objective of this project is to construct a sorting system model where the batches will be separated and transferred to packaging area with minimum number of stages for each boutique. When number of batches exceeds number of chutes, it is not possible to assign single batch to a chute meaning that the process will include multiple stages. The system should be designed to allocate batches to chutes so that each batch is processed as little as possible.

With this project, total time of processing each boutique will be minimized, followed by a decrease in the time that takes for the products to be delivered to customers. This will result as an increase in customer satisfaction and retention.

## Analysis and Interpretation of the Data

The group obtained a sample data set of boutiques and their batches and respective item quantity per batch from Trendyol. The data included 3 data types and therefore had 3 columns named as Boutique ID, Batch ID and Number of Items in each batch. The sample data included 20 different boutiques numbered from 420 to 440. Batch ID’s were named as ISxxxxxx. The data contains 2000 different batches implying that on average a boutique contains approximately 100 batches. In total, there are 113,147 items in the given data set which implies there are 56.57 items per batch. The most significant data is the number of batches per boutique. Since the group was provided that there are 66 chutes in the sorter the threshold for each boutique is 66 batches.

If a number of batches of a boutique exceeds 66 than it must enter to a multi-stage allocation. There are 12 boutiques having more than 66 batches while 8 boutiques have less number of batches than the chute number. Boutique 428 has 199 batches which is the maximum amount while Boutique 438 has the least with 24.

Since the data is straightforward, the group was not able to conduct a detailed analysis of the data other than what is stated above. The data can be directly imported into the constructed model in the source code. In doing so, it is important that the data set is sorted in a sense that boutique ID numbers start from the lowest and increase until the last defined boutique ID number. This is required for not encountering any errors while reading the data from a csv file format.

# Literature Review and Sector Analysis

## Literature Review

The state-of-the-art regarding this project and research about it has not been straight forward. Since the problem is very complex and unique due to its constraints about the boutiques, batches and chute assignments and its multi-stage nature; finding very similar research papers including mathematical models was a challenge. Due to this situation, the group believes that formulation of a model for this specific project is going to require rather more creative thinking. However, for literature review purposes, it was still decided to expand our knowledge about similar problems such as assignment, scheduling and dynamic programming problems.

The job shop scheduling problem is described as follows. Given are a set of jobs and a set of machines. Each machine can handle at most one job at a time. Each job consists of a chain of operations, each of which needs to be processed during an uninterrupted time period of a given length on a given machine. The purpose is to find a schedule, that is, an allocation of the operations to time intervals on the machines, that has minimum length.

In this model, instead of jobs there are batches and for machines there are chutes. The main objective is to find an optimal solution to the assignment of batches to chutes.  In the system, a batch is processed and transferred to the packaging area if all of its order falls into the same chute. However, the system differs with job shop scheduling in terms of number of batches assigned to a chute. In this system, more than one batch can be assigned to a chute initially and then they will be put to the sorter for the upcoming cycle until each batch is allocated into a different chute.  According to the job shop scheduling there has to be a single batch assigned to each chute which is not the case for this system. Therefore, this system cannot be explained and solved as a basic job shop scheduling problem.

Assignment problems and some of its real-life cases are also investigated. In assignment problems, n-objects should be assigned to m-other objects in a way that optimal solution is achieved. Even though it seems to be similar with Trendyol project, as batches should be assigned to chutes, there are major differences that should be consider before. Trendyol problem cannot be solved with one step assignment when number of batches increases. In addition, there are no constraint regarding the number of batches that can be assigned to each chute, meaning multiple batches can be assigned to a chute. In that case, process would not be completed and batches will return to sorter, causing multiple stages to handle. In conclusion, this specific problem requires a completely different way of thinking and management.

Dynamic programming solves the problems by forming objective function, using pre-determined stages and defining states for each stage. It starts with the final stage and recursively going back to first stage of the problem in order to find the optimal solution. The scope of the project is to minimize the number of cycles each batch is assigned to chutes. It concerns with the sorter and assigning batches to chutes in the warehouse system of Trendyol. However, the puzzling part of the problem of Trendyol is that the number of stages are not pre-determined like usual dynamic programming.

Considering the number of available chutes as stages, the available chute number is dependent on the decision that is taken before the stage but since the usage time of chutes have different time scales for each batch this creates unknown number of stages and a complicated system that the usage of dynamic programming is not enough to find an optimal solution.

McFarlane et al. (2015) study interventionist routing algorithm for optimizing the dynamic order-picking routes. This algorithm performs both static and heuristic dynamic order-picking routing algorithms. Static Order-Picking (SOP) includes the formation of batches to prepare static pick-lists. The similar system is used by Trendyol when they prepare job lists for batches which is a time-consuming process. On the other hand, Dynamic Order-Picking (DOP) enables changes of a pick-list during the cycle. This method is not used by Trendyol but can be used by the group as an idea while minimizing the number of cycles each batch assigned to a chute.

Boysen et al. (2018) explain the current trends of warehouse systems in the e-commerce era. They examine different methods used by Amazon Europe and fashion retailer Zalondo such as mixed-shelves storage. The relevant part of this research to the problem of Trendyol is that how they combine batching and zoning with mixed-shelves storage. Batching system is used by Trendyol which allows orders which has multiple items to combine together in order to get into the sorter and assigned to the chutes. Batching allows the pick density per tour to increase and creates more efficient picking process. Formation of batches are emphasized in the article since it is a crucial point for e-commerce retailers. Dynamic order process is suggested since it allows flexible processes that enables urgent orders to deliver on time. Even though this is not used by Trendyol, the articles that the group researched so far show that this will be the new trend for warehouse systems who deal with huge amounts of orders which consists of small orders, large assortment, tight delivery schedules and varying workloads.

## Sector Analysis

Trendyol is in the e-commerce sector. According to study conducted by Nielson in 2018, Turkey has an internet penetration rate of 72%, which indicates 4% of Turkish consumers shop online daily. In addition, 46% of Turkish consumers order services and products online few times in a month. The most preferred product categories are clothes and shoes which consists of 68% of consumer who shop online. In 2018, Turkey had an ecommerce sector worth 5.7 billion euros and a 38% growth rate compared to 2017. The Turkish ecommerce sector is dominated by the firms Gitti Gidiyor, Hepsiburada, Araba.com and Trendyol.

Another study conducted by Deloitte and Informatics Industry Association (TÜBİSAD) estimates ecommerce market size has an average annual growth of 33% between 2014-2018 in Turkey. In addition, they found out that the average annual growth rates of online legal betting, travel, multi-channel retail and pure online retail are 28%, 36%, 31% and 34% respectively. In 2018, ecommerce market size is worth 59.9 billion TRY at total distributed as non-retail market value of 28.4 billion TRY and retail market value of 31.5 billion TRY. It was 42.2 billion TRY in 2017. This indicates that ecommerce sector has a growth rate of 42% between 2017 and 2018.The more detailed information about the e-commerce market size in Turkey between 2014 and 2018 can be seen in the Figure 1.

The article written by Tuğba Bafra and published by Association of International Forwarding and Logistics Service Providers (Utikad) in 2018 states that “Turkey having been estimated to be the second fastest growing country in B2C e-commerce in 2017.”

As a result, it can be concluded that Trendyol is in a sector that grows fast which indicates that there will be more opportunities in the future. Therefore, Trendyol should enhance itself in order to stay in the market and stay as one of the dominant firms in Turkey.

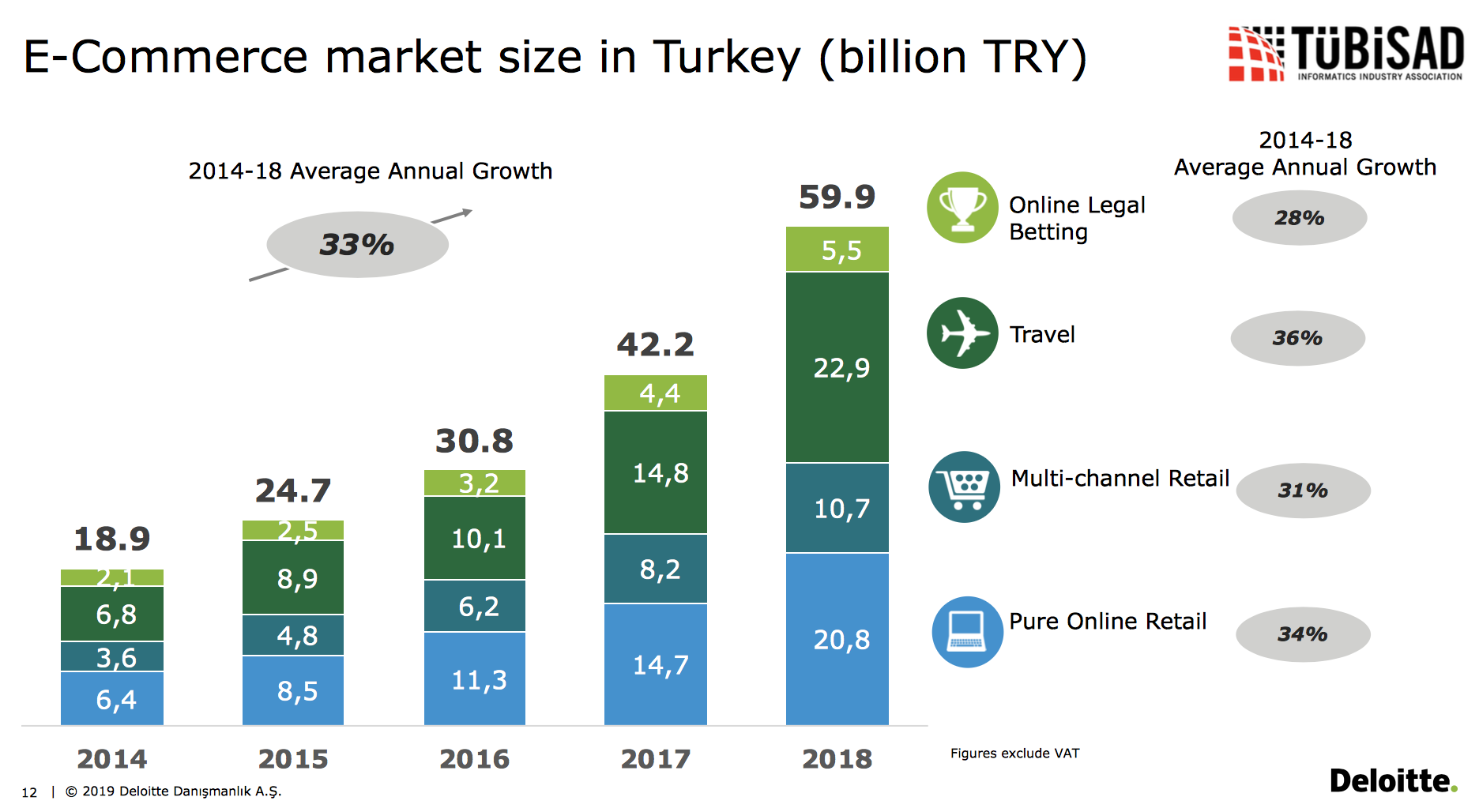


Figure 1: E-Commerce Market Size in Turkey (2014-2018)

# System Design

## System Architecture

The group has worked on the allocation of orders to the sorter at Trendyol’s Warehouse. The system consists of the boutiques, the batches within the boutiques, items in each batch and the chutes in the sorter. Boutiques are the sales channels Trendyol opens periodically based on various occasions such as a special event, or a collection of items from various brands that sell the same kind of products. Every boutique has one or more batches and batches include many items. The sorter at Trendyol’s Warehouse has a total of 66 chutes. Within the system being considered, the chutes do not have a physical capacity regarding the number of items being assigned to them. This fact was told to the group by the project supervisor and other Trendyol employees during the field visit. As assigned items are allocated to the vast baskets that contain the sorted items, the baskets might fill up. However, the group was told that this is manually taken care of by simply changing the baskets right before them being filled with the assigned items, if necessary. Additionally, Trendyol defines a virtual capacity for each chute which is included in the mathematical model. This chute capacity denotes the maximum number of allowed assignments of batches to a certain chute.

The inputs of the system can be classified as boutiques, batches, item lists of batches and the number of chutes available in the sorter. The model will yield the optimal number of stages for the system and allow for variations in the input data set, the number of chutes, the chute capacity and the parametrization constant denoted by alpha. The outputs of the model are the number of items assigned and single batch assignments. The following summary diagram shows inputs and outputs of the system.

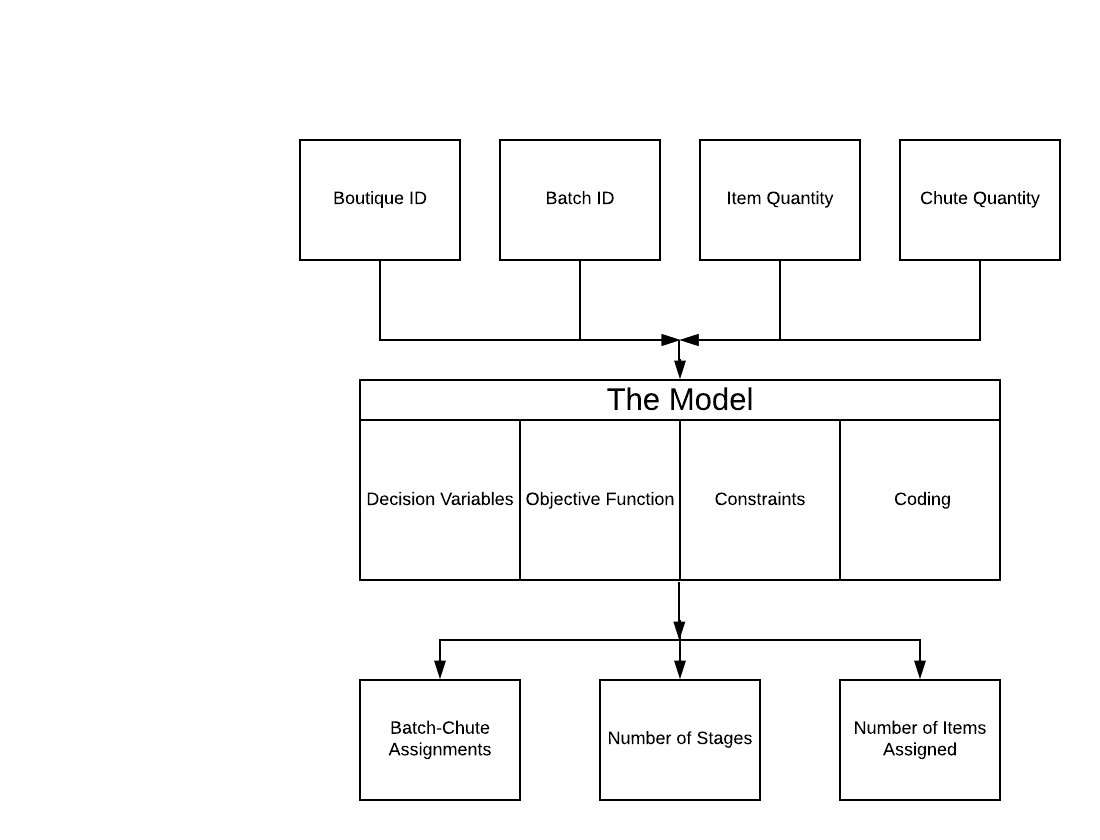


Figure 2: Inputs and Outputs of the System

In order to understand the overall process of the system and the model that is being optimized, a logical system diagram is prepared. Initially, a batch comes to the assignment point on the sorter. Then it is checked whether there is any chute without an assignment in that stage. If there is, then the batch is assigned to one of the available chutes or the batch is assigned to a chute which was assigned to another batch from the same boutique. If there is not any chute without an assignment, then it is checked if there are any chutes that have received an assignment of a batch from the same boutique. If the answer is yes, the batch is assigned to a chute which was assigned to another batch from the same boutique. If the answer is no, then the batch is not assigned and it remains on the sorter.

The developed mathematical model considered all of these facts and turns them into mathematical expressions and equations, which will be explained in more detail in following sections. A logical system diagram that summarizes the detailed explanation of the system above can be found below in Figure 3.

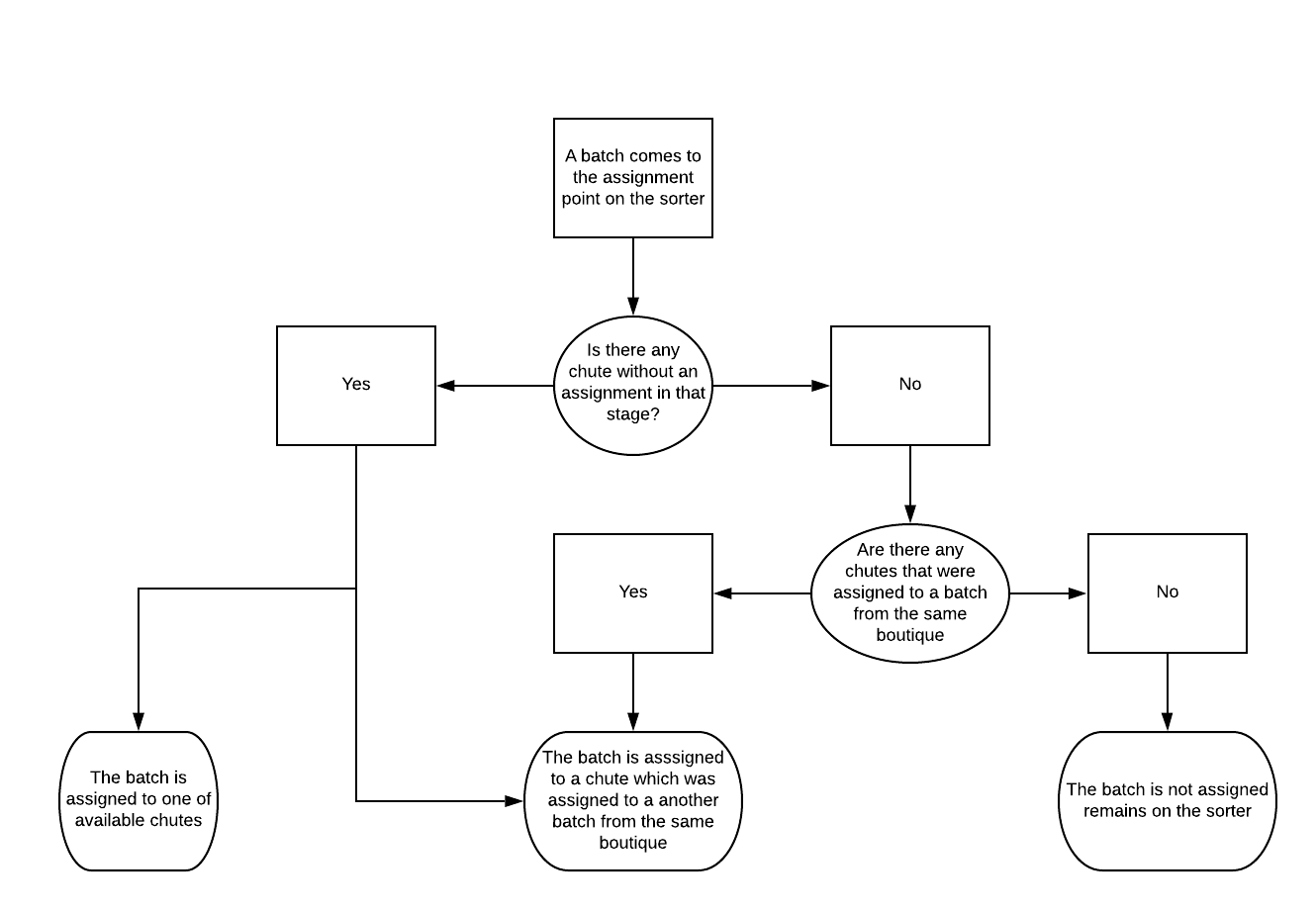


Figure 3: A Logical System Diagram

## Model Description

The system was examined carefully to be converted into a mathematical mixed integer programming model. The model is required to maximize the efficiency of the sorter so that the number of stages would be minimized. In order to decrease the number of stages, the system should release as many batches as possible at each stage. So, each stage will act like a single-stage model. This is why the model is considered to be single-stage. After implementing single stage model in a Java model, it is concluded that converting it to real case multi-stage is possible with repetition of this single stage model. The code is re-constructed based on this idea using the model provided below. The model is constructed in such a way that the model runs in a loop until all input batches have been allocated according to the rules of the real-world situation and the corresponding constraints of the model.

In the system, the only way to observe a release is to have a single assigned batch to a chute. So, our objective is to maximize single assignments to chutes. Furthermore, the total number of items being assigned should also be considered to select batches with maximum number of items. The importance of these objectives is kept flexible in order to be able to evaluate alternative options. The objective is defined accordingly, which maximizes the number of single batch assignments to the chutes and the total number of items in all assigned batches. On the other hand, constraints in the system and alternatives to apply them in our model is observed.

When the objective function and constraints are defined, modelling is started with selecting sets, parameters and decision variables. Inspired by the data set received from Trendyol, the created model includes following inputs: boutiques, batches and chutes.

For each input, a set is created starting with 1 and until the total number of inputs. Here, it is noteworthy that the J value that denotes the number of batches in a boutique is actually determined by the boutiques since it is unnecessary to create variables that are not going to be used. Since the number of required variables and constraints in the model is almost always too much, it was decided to create the variable sets in the coding process in such a way that the total number of batches J is defined as the maximum number of batches in the boutique with the maximum number of batches.

**Sets**

Set of Boutiques: {1,2, …, I}

Set of Batches: {1,2, ..., J}

Set of Chutes: {1,2, …, K}

Parameters are chosen to be; number of items in each batch, number of batches in each boutique, number of possible batch assignments to chutes and weight of the objectives. As it can be seen in following steps, defining these parameters as inputs allows us to create simple formulation for the objective function and constraints.

**Parameters**

: the number of items in boutique i’s batch j

: the number of batches in boutique i

: parametrization constant used as weight of the objectives

: chute capacity - indicates the allowed number of possible batch assignments to a certain chute

A boutique can either be assigned or not. Binary variables X are defined to observe whether a batch from a certain boutique is assigned to a certain chute. The number of batches assigned to a certain chute is defined as variable Y so that identifying and tracking single batch assignments when Y is equal to one is possible. Binary variables Z are created to be used in the objective function, so that the chutes receiving a single batch assignment will be equal to one and zero otherwise. In order to count the total number of batches assigned from each boutique, T variables are created. Finally, s and t dummy variables are created to be used in the constraints. The dummy t variables will prove to be very useful in updating the parameters after completion of each stage.

**Decision variables:**

: binary, equals to 1 if boutique i’s batch j is assigned to chute k

: total number of batches assigned to chute k

: binary, indicator variable equal to 1 if Yk = 1 and 0 otherwise

: total number of batches assigned from boutique i

: binary dummy variable for the if-then constraint, si,k equals 1 if no batch was assigned from boutiques other than i to chute k

: binary dummy variable for the either-or constraint concerned with setting Ti variables either to 0 or bi

The objective function is set to maximization of the total number of single batch assignments in all chutes and the total number of items in all assigned batches of boutiques. Decision variables , and parameters are used for formulation of this function. Since these two objectives are not equally important and depend on preferences of Trendyol employees, parameters and are defined so that the weights of these objectives could be set according to preferences in the function.

**Objective function:**

The constraints in the system are applied to the model using decision variables and parameters explained previously. Furthermore, relationships between the variables are created below.

**Constraints:**

Relationship between and can be seen below. The decision variables are connected so that summation of the X variables will give the total number of batches assigned for each chute.

Relationship between and can be seen below. The decision variables are connected to enforce the condition that if Y is equal to one, Z must be one. Otherwise, Z must be zero.

Relationship between and can be seen below. The decision variables are connected so that summation of the X variables will give the total number of batches assigned from boutique i.

A batch can only be assigned to one chute.

All assigned batches to a certain chute should be from same boutique. In other words; if a batch is assigned to a certain chute from a boutique, there cannot be any assignments from another boutique to that chute. Dummy variables are created for applying the if-then condition in this constraint.

If a batch is assigned from a certain boutique, then all other batches in that boutique should be assigned. In other words, number of assigned batches from a certain boutique can be either zero, or the total number of batches in that boutique. Dummy variables are created for applying the either-or constraint.

**Range constraints:**

The model provides an optimal solution for a single stage. Moreover, it is observed that behavior of all stages is exactly same where objectives and constraints stay constant. The only variation between the stages are sets defined for boutiques, batches and chutes. As a result, with simple adjustments, this single-stage model is converted to represent the whole system by taking it into a loop and updating the required input parameters after completion of each stage.

## Solution Methodologies and Validation

Initially single-stage mixed integer programming model was programmed using Java Eclipse and its *IloConcert* *Technology* and *IloCplex* libraries. *cplex.jar* should be used as a referencing library. The source code with a *.java* expansion was designed so that it could formulate and solve single stage models of the allocation of orders to the sorter problem. Additionally, each line of the data set containing of the inputs BoutiqueID, BatchID and numOfItems is stored in a created class called *Triplet* as Integer, String and Integer values, respectively. This way, each single batch can be identified with its String value that corresponds to its real-life identifier such as ISxxxxx.

After the initial code was validated with accurate results with a fairly small data set during the progress report stage of the project, the focus was later transformed into solving it for multi-stage cases. It is already mentioned that construction of the code will be based on the repetition of the single stage. So, the Java code should be taken into a loop where all sets and variables are redefined at each stage based on the status of batches at the end, and then the code runs for the next stage with redefined sets, until all batches are released from the system.

The Java code is examined carefully in order to identify the optimal way to implement the following method. The inputs are needed to be defined outside of the loop. After all sets and parameters are defined, a loop is created processing all variables, objectives and constraints in it. *Main* function is used to define these parameters, and *SorterAllocation* runs in it. *SorterAllocation* is programmed as a different method, where all remaining codes exists in the loop. The advantage that the *SorterAllocation* function brings is the following. It takes four inputs. These are the dataset, the number of chutes, the chute capacity and the parametrization constant. It is fairly easy to update these parameters by hand and quickly observe results of the model.

At the end of each stage, status of batches are checked whether they entered into the system and if so, whether they left the system. Based on these status, all sets are regenerated. So, when the code runs for the next stage, it will check whether all batches left the system. If so, the loop will end and if not, it will run for the next stage with the new sets. In the appendix of this report, the source code can be found.

To validate the model, a simple and small-scale system was considered. A total of 3 boutiques and 9 batches were distributed into 4 chutes with chute capacity determined as 2. Information about boutiques, batches and items can be found below. It is believed that validating results from such a small data set is in fact necessary to be able to easily visualize the resulting assignments.

For simplicity purposes, the objective term weight parameter was initially set to 0.5 to give the objectives an equal amount of importance.

The model ran without an error and the console output was examined thoroughly to see whether the constraints were constructed correctly. During the progress report phase of this project, the group had already included a validation of the results of the model with a smaller data set. Here, findings from that version of the data set are given once again. These results are for a small data set with the first boutique (10,9,8,7); second (6,5) and third (4,11,12). Each entry represents the number of items in the corresponding batch.

The source code is designed in such a way that almost all variable solution values from the model can be seen in the console output if the corresponding command lines are commented in.

As a result, the model decided on the following assignments: Boutique 1’s batch 1 was assigned to the 3rd chute, whereas the remaining 2nd, 3rd and 4th batches were assigned to the 4th chute. Both batches of the 2nd boutique were assigned to the 2nd chute. Finally, the single batch included in the 3rd boutique was assigned to the 1st chute. The results were satisfactory of all of the constraints that were specified. Additionally, values of other variables such as , , , and also satisfied their constraints and relationships with the assignment variables of . For example, the values that are supposed to tell the total number of batches received by the chutes turned out as = 1, = 2, = 1, = 3. These were consistent with the summations of the variables as expected. Furthermore, the variables needed to be equal to 1 only if their accompanying variables were equal to 1 and these were also satisfied.

As a result, the model was validated as one that can solve a small scale and single stage version as expected. It was also noted that further altercations and maybe simplifications could be needed for rather larger scale problems with lots of boutiques, batches and chutes as the number of variables and constraints grew.

## Consideration of Alternatives

In the beginning of modeling, two possible objective functions are considered. Since the aim of the system is to optimize the sorter allocation system by minimizing the number of stages, the first consideration was to minimize the number of stages. Therefore, defining number of stages as a variable and building a multi-stage system accordingly has been considered. However, complexity of this model was a significant downside and the group has decided to solve the problem by another perspective. The objective function is defined to maximize number of single assignments in each stage, which also enabled the group to work on the single-stage model in the beginning. As a result of all evaluations, a model that maximizes the number of single assignments is preferred considering its simplicity.

In the constructed model, objective function has two objectives and therefore it can be considered as multi-objective function. First part of the objective function is set to maximize the total number of single batch assignments in all chutes and second part of the objective function is to maximize the total number of items in all assigned batches of boutiques. The coefficients of two objectives are determined by parameter and , which enabled the group to investigate what effect will alpha have when alternative weights are given to these two objectives and this will be used in What-If Analysis. In addition, the capacity of the chutes also defined as parameter so that the group can adjust it to obtain the most efficient model by changing the chute capacity parameter, which will be used in What-If Analysis as well.

Two possible options have been considered while deciding on how to convert the single-stage model into multi-stage. First option was to include stage numbers to the variables of the current model and adjust constraints accordingly so that maximization of the single assignments will be provided for all stages.

The second option is to use the current mathematical model and in order to solve for multi-stage, repeat the single stage model by redefining the sets at the beginning of each stage. The group has decided to use the second option and prepared a Java code.

## Scenario / What-if Analysis

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Batch / Boutique ID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 10 | 9 | 8 | 7 | 10 | 9 | 8 | 7 |  |  |
| 2 | 6 | 5 | 10 | 9 | 8 | 7 | 10 | 9 | 8 | 7 |
| 3 | 4 | 11 | 12 |  |  |  |  |  |  |  |
| 4 | 10 | 9 | 8 | 7 |  |  |  |  |  |  |
| 5 | 6 | 5 | 10 | 9 | 8 | 7 |  |  |  |  |
| 6 | 4 | 11 | 12 | 10 | 9 | 8 | 7 |  |  |  |
| 7 | 10 | 9 | 8 | 7 | 10 | 9 | 8 | 7 | 10 | 9 |
| 8 | 6 | 5 |  |  |  |  |  |  |  |  |
| 9 | 4 | 11 | 12 |  |  |  |  |  |  |  |

Table 1: Table representation of created sample data set in the final model

The above table shows the number of items in certain boutique – batch combinations. The entries in the table are the number of items in boutique i’s batch j. A total of 9 boutiques and 53 batches were distributed into 10 chutes with chute capacity determined as 7. Information about boutiques, batches and items can be found above. Excel and csv file versions of this dataset was included in the submission of the java project code. Scenario analysis was conducted using this data set by altering input parameters. The complexity of the model and the way that the source code defines the number of variables and constraints proved to be a limiting factor here. Therefore, the group decided to conduct the analysis with this data.

In order to conduct What-if Analysis, the weight of the objectives and the chute capacity are defined as parameters of the function that runs the model in a multi-stage loop, as explained before. This way, outcomes of different alternatives can be observed to derive optimal value for the parameters and the possible number of stages achievable. With the sets of boutiques, batches and items from the validation, the alternatives are investigated. Tables below show the number of stages for different values of alpha and chute capacity, when all other variables kept constant.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| What-if Analysis | | | | | |
| Chute Capacity =7 | | | | | |
| α | 0.2 | 0.5 | 0.7 | 0.9 | 0.99 |
| Number of stages | 12 | 11 | 12 | 11 | 7 |

Table 2: What-If Analysis: The number of stages with respect to various alpha values

Considering the objective function, determines importance of having single assigned chutes. On the other hand, is for the total number of items in the system. Since a single assignment is the required condition for a batch to leave the system, the number of stages is expected to be lower when increases. It can be seen from the table that the minimum level for number of stages is obtained when is 0.99. However, it should be noted that the relationship between and the number of stages is not perfectly linear. A high value for does not provide lower number of stages all the time. This indicates that optimization should be done very carefully where alternative values for is examined.

Sensitivity of the model to changes in is related to the number of chutes. In objective function, is the coefficient of variable which is created for each chute. So, number of ‘s created in the objective function will be related to the number of chutes. On the other hand, is the coefficient of which is created for every batch. This is why the number of variables multiplied by in the objective function is not related to the number of chutes in the system. As a result, our model will be more sensitive to changes in parameter when chute number increases

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| What-if Analysis | | | | | |
| α=0.5 | | | | | |
| Chute Capacity | 7 | 10 | 15 |  |  |
| Number of stages | 11 | 9 | 9 |  |  |

Table 3: What-If Analysis: The number of stages with respect to various chute capacity values

Another analysis has been conducted with various chute capacities. Since there are no physical constrains on chute capacities, it is used to optimize the system. This is why the code is run for different capacity values and outcomes are investigated. With the current input dataset, it is seen that increasing capacity to 10 batches enabled model to be more efficient where number of stages is decreased to 9. However, further increases in capacity is not required as the outcome does not change. Analysis showed that this model also can be used to determine optimal parameters to increase efficiency.

The Java code was not able to run with higher number of chutes, batches or other parameters that would increase number of variables in the system. It is believed that this model is scalable, and therefore can be used with larger datasets. However, when the number of boutiques, batches, items or chutes is increased, it is required to create all corresponding variables and the number of variables and constraints that are needed to be stored for solving models increases significantly. So, increasing parameters such as the number of sets or the number of chutes causes system to increase variable number geometrically. These significantly high number of variables are not supported by free versions of the *cplex* libraries used in the java code. This is why the group has encountered the following error when the number of boutiques, batches or chutes was set significantly higher.

A screenshot of a cell phone

Description automatically generated

Figure 4: Console output error screenshot that shows size limit is exceeded

As a result, these errors are probably caused by lack of resources of the group and it can be solved by acquiring complete versions of the libraries used in the java code. This way it would be scalable and can be solved for larger datasets. Additionally, it would perhaps be possible to change the intermediate steps of the code that holds vast amounts of system variables in objects such as *arrays*, *lists* or *hashmaps*. The group believes that the code can later on be developed into further versions that reduces the time and storage complexity. It should once again be noted that after all, problems occurred due to having access to only a free version of the *cplex* libraries.

# Implementation

The project started with the visit to the Trendyol warehouse on February 7th, 2020. The group had the chance to observe the project system. Then the system data was obtained from the supervisor. Then the data was analyzed and followed by what can be called a brainstorming session regarding the modeling of the system. The solution approach was determined as first considering the model as a single stage one and then adapting it for a multi-stage case. Therefore, a mathematical model was constructed for the single stage case and a java code was coded for it during the progress report period.

Along with the progress report, the mathematical model and the java code were the deliverables realized during April. These were followed by the adaptation of the single stage model for the real life multi-stage system. The model was modified as well as the respective java code. Also, a video presentation was prepared for the project, which was another realized deliverable. Along with this report, the group concludes the project with the final mathematical model and its source code as the final deliverables realized for this project.

At the end, the group has a mathematical model that considers the multi-stage case within the java source code. The appendix of this report produces a copy of the source code but it should once again be noted that the *cplex.jar* library was used as a referencing library in running it. The proposed algorithm finds an optimal allocation of the batches in the input data set to the chutes. The final report will be presented to the client. However, the group had to perform analysis according to smaller data sets and therefore, it is believed that the model may be inadequate for the real-life data of the client. It is advised to the client to test the proposed model and integrate it into the existing current system in order to test its efficiency. Modifications may be needed according to the performance of the proposed model on the real system. Even if some modifications may be needed, this is a milestone for the client since there isn’t any model for the system and a manual allocation is used currently.

The proposed model can be modified and completely be adapted to their system and they can start to use it for their operations. The proposed algorithm finds an optimal batch-chute allocation that minimizes the number of cycles for each boutique. This algorithm will help sort batches with less number of cycles in a shorter time. Therefore, the sorting process will be conducted faster in a more efficient way. This will decrease the cost of sorting and delivery time to customers. A decrease in overall cost may and should lead to increasing profits for the client. The client can save time as well as earn more profit. Also, the increase of efficiency and speed of the operations may lead to an increase of boutique number and Trendyol can offer brand new boutiques and products to its customers. They can expand their operations with this method. It is believed that further improvements to the source code by client employees might create sufficient and efficient results in the allocation of orders to the sorter at the warehouses of Trendyol.

Overall, the group believes that this project, along with its proposed model for allocation of orders to the sorter, the findings from analysis of different scenarios and other deliverables can become vital guiding elements for the client’s future operations. This may lead to significant amounts of increase in their efficiency and profit.

# References

Bafra, Tuğba. “ E-Commerce In Turkey Potential For Development And Logistics Processes.” *Utikad*, UTİKAD Association of International Forwarding and Logistics Service Providers, Feb. 2019, www.utikad.org.tr/images/HizmetRapor/potentialfordevelopmentofecommerceandeexportinturkeyandlogisticsprocesses-23983.pdf.

Boysen, Nils, et al. “Warehousing in the e-Commerce Era: A Survey.” *ResearchGate*, European Journal of Operational Research, Aug. 2018, [www.researchgate.net/publication/327180275\_Warehousing\_in\_the\_e-commerce\_era\_A\_survey/link/5b8edf6a92851c6b7ec02849/download](http://www.researchgate.net/publication/327180275_Warehousing_in_the_e-commerce_era_A_survey/link/5b8edf6a92851c6b7ec02849/download).

“E-Commerce in Turkey 2018 Market Size.” *TÜBİSAD Information Industry Association*, Deloitte, Apr. 2018, [www.tubisad.org.tr/en/images/pdf/tubisad\_2019\_ecommerce\_eng.pdf](http://www.tubisad.org.tr/en/images/pdf/tubisad_2019_ecommerce_eng.pdf).

“Ecommerce in Turkey.” *Ecommerce News*, June 2019, ecommercenews.eu/ecommerce-in-europe/ecommerce-turkey/.

Lenstra J.K. (1992) Job Shop Scheduling. In: Akgül M., Hamacher H.W., Tüfekçi S. (eds) Combinatorial Optimization. NATO ASI Series (Series F: Computer and Systems Sciences), vol 82. Springer, Berlin, Heidelberg

Lu, Wenrong & Mcfarlane, Duncan & Giannikas, Vaggelis & Zhang, Quan. (2015). An algorithm for dynamic order-picking in warehouse operations. European Journal of Operational Research. -. 10.1016/j.ejor.2015.06.074.

# Appendix

// May 29,2020 Indr491 Project

// Trendyol Allocation of Orders to the Sorter Programming Model

// prepared by Caner Kayalı

// external cplex.jar library referencing is required to run

package indr491Project;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;

import java.util.List;

import ilog.concert.IloException;

import ilog.concert.IloIntVar;

import ilog.concert.IloLinearNumExpr;

import ilog.cplex.IloCplex;

public class TrendyolFinalModel {

public static void main(String[] args) throws IOException, IloException {

// reading input data from csv format, converting it into Triplet format

// string argument should be changed according to csv file path

List<Triplet> inputList = *fromCSVtoTripletList*("/Users/canerkayali/Desktop/Indr491FinalSubmission/Data/data.csv");

// number of chutes in the system

int K = 10;

// number of maximum allowed batch assignments to a chute

int chuteCap = 7;

// parametrization constant for objective function pieces. alpha acts as the

// importance of single batch assignmets while 1-alpha acts as the importance of

// total number of item assignments.

double alpha = 0.5;

// running the model until all boutiques and batches are processed as single

// assignments

*SorterAllocation*(inputList, K, chuteCap, alpha);

}

// function for the model being solved step by step

private static void SorterAllocation(List<Triplet> inputList, int K, int chuteCap, double alpha)

throws IloException {

// this HashMap stores boutiqueID as key and a list of integers representing

// number of items in that boutique as value

HashMap<Integer, List<Integer>> boutiqueIDNumOfItems = new HashMap<Integer, List<Integer>>();

// store the initial boutiqueID from inputList

int startingBoutiqueID = inputList.get(0).getboutiqueID();

// store the last boutiqueID from inputList

int endingBoutiqueID = inputList.get(inputList.size() - 1).getboutiqueID();

// starting from first boutiqueID until the last, putting elements into the

// HashMap boutiqueIDNumOfItems

while (startingBoutiqueID != endingBoutiqueID + 1) {

List<Integer> tempList = new ArrayList<Integer>();

for (int i = 0; i < inputList.size(); i++) {

if (inputList.get(i).getboutiqueID() == startingBoutiqueID) {

tempList.add(inputList.get(i).getnumOfItems());

}

}

boutiqueIDNumOfItems.put(startingBoutiqueID, tempList);

startingBoutiqueID++;

}

// initializing numberOfItems: each integer array represents the number of items

// in the batches of the boutique of that index

ArrayList<int[]> numberOfItems = new ArrayList<int[]>();

// looping through values of the HashMap to add them to ArrayList<int[]>

// numberOfItems

for (List<Integer> myList : boutiqueIDNumOfItems.values()) {

int[] intarr = new int[myList.size()];

for (int i = 0; i < myList.size(); i++) {

intarr[i] = myList.get(i);

}

numberOfItems.add(intarr);

}

// creating an array list of integers to represent boutique sizes at each index.

// The number of integers at each index of numberOfItems gives that value.

ArrayList<Integer> numberOfBatches = new ArrayList<Integer>();

// calculating the total number of items within the system as condition of stage

// looping. Storing this as sum As long as there are items to process the

// overall while loop continues

int sum = 0;

for (int i = 0; i < numberOfItems.size(); i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

sum += numberOfItems.get(i)[j];

}

}

// initializing the stage number

int stage = 0;

// the model runs through many stages as long as there still remains batches and

// items to be processed. this is checked by calculating the sum value after

// each stage.

while (sum > 0) {

// incrementing stage number and printing it

stage++;

System.***out***.println("Stage " + stage + " beginning! ");

System.***out***.println();

// each entry in numberOfBatches represents the total number of batches in a

// certain boutique

for (int[] intarr : numberOfItems) {

numberOfBatches.add(intarr.length);

}

// writing lines for the input data for the specific stage number

System.***out***.println("Solving model for the following: ");

System.***out***.println();

System.***out***.println("numberOfBatches: ");

System.***out***.println(numberOfBatches);

System.***out***.println();

System.***out***.println("numberOfItems: ");

for (int[] intarr : numberOfItems) {

System.***out***.print("[ ");

for (int j : intarr) {

System.***out***.print(j);

System.***out***.print(" , ");

}

System.***out***.println("] ");

}

System.***out***.println();

System.***out***.println();

// extract I and J to represent total number of boutiques and batches,

// respectively.

// size of numberOfBatches gives total number of boutiques.

// J selected as maximum element in numberOfBatches as only so many variables

// representing batches are needed regardless of boutique number.

int I = numberOfBatches.size();

int J = Collections.*max*(numberOfBatches);

///// create the variable sets

IloIntVar[][][] X = new IloIntVar[I][J][K];

IloIntVar[] T = new IloIntVar[I];

IloIntVar[] Y = new IloIntVar[K];

IloIntVar[] Z = new IloIntVar[K];

///// create the dummy variable sets

IloIntVar[] t = new IloIntVar[I];

IloIntVar[][] s = new IloIntVar[I][K];

// the model tends to have slight calculation errors. These extra arrays will

// hold true solution values of exact boolean and integer values instead of

// double values since these are required in update procedure after each stage

int[][][] XBool = new int[I][J][K];

int[] YInt = new int[K];

int[] ZBool = new int[K];

int[] tInt = new int[I];

///// create arrays for holding the constraint equations

IloLinearNumExpr[] XY\_Relationship = new IloLinearNumExpr[K];

IloLinearNumExpr[] XT\_Relationship = new IloLinearNumExpr[I];

IloLinearNumExpr[] YZ\_Relationship\_1 = new IloLinearNumExpr[K];

IloLinearNumExpr[] YZ\_Relationship\_2 = new IloLinearNumExpr[K];

IloLinearNumExpr[][] batchAssignedToOneChute = new IloLinearNumExpr[I][J];

IloLinearNumExpr[][] batchAssignedFromSameBoutique\_1 = new IloLinearNumExpr[I][K];

IloLinearNumExpr[][] batchAssignedFromSameBoutique\_2 = new IloLinearNumExpr[I][K];

IloLinearNumExpr[] T\_Either\_Or\_1 = new IloLinearNumExpr[I];

IloLinearNumExpr[] T\_Either\_Or\_2 = new IloLinearNumExpr[I];

// build the model

IloCplex model = new IloCplex();

// DECISION VARIABLES

// build binary X

// X\_i, j, k is equal to 1 if boutique i's batch j is assigned to chute k

for (int i = 0; i < I; i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

for (int k = 0; k < K; k++) {

X[i][j][k] = model.boolVar("X\_" + (i + 1) + "," + (j + 1) + "," + (k + 1));

}

}

}

// build nonnegative integers Y and binary Z

// Y\_k represents the total number of batches assigned to chute k

// Z\_k is equal to 1 if Y\_k is equal to 1, meaning that a single assignment was

// made to chute k

for (int k = 0; k < K; k++) {

Y[k] = model.intVar(0, chuteCap, "Y\_" + (k + 1));

Z[k] = model.boolVar("Z\_" + (k + 1));

}

// build nonnegative integers T and dummy binary t

// T\_i represents the total assignments made from boutique i

// t\_i equals 1 if T\_i is equal to numberOfBatches[i]

// t\_i equals 0 if T\_i is equal to 0

for (int i = 0; i < I; i++) {

T[i] = model.intVar(0, numberOfBatches.get(i), "T\_" + (i + 1));

t[i] = model.boolVar("t\_" + (i + 1));

}

// build dummy binary s

// s\_i,k equals 1 if no batch was assigned from boutiques other than i to chute

// k

// s\_i,k equals 0 if no batch was assigned from boutique i to chute k

for (int i = 0; i < I; i++) {

for (int k = 0; k < K; k++) {

s[i][k] = model.boolVar("s\_" + (i + 1) + "," + (k + 1));

}

}

/////////////////

// OBJECTIVE FUNCTION

// build the objective function

IloLinearNumExpr objTerms = model.linearNumExpr();

// objective function part1 : sum of z values (single assignments)

for (int k = 0; k < K; k++) {

objTerms.addTerm(alpha, Z[k]);

}

// objective function part2 : sum of number of items assigned

for (int i = 0; i < I; i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

for (int k = 0; k < K; k++) {

objTerms.addTerm((1 - alpha) \* numberOfItems.get(i)[j], X[i][j][k]);

}

}

}

// set objective function to be maximized

model.addMaximize(objTerms);

///////////////////////////

// CONSTRAINTS

// construct x-y relationship constraints

for (int k = 0; k < K; k++) {

XY\_Relationship[k] = model.linearNumExpr();

for (int i = 0; i < I; i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

XY\_Relationship[k].addTerm(1, X[i][j][k]);

}

}

XY\_Relationship[k].addTerm(-1, Y[k]);

model.addEq(XY\_Relationship[k], 0);

}

// construct x-t relationship constraints

for (int i = 0; i < I; i++) {

XT\_Relationship[i] = model.linearNumExpr();

for (int j = 0; j < numberOfItems.get(i).length; j++) {

for (int k = 0; k < K; k++) {

XT\_Relationship[i].addTerm(1, X[i][j][k]);

}

}

XT\_Relationship[i].addTerm(-1, T[i]);

model.addEq(XT\_Relationship[i], 0);

}

// create bigM as a large enough value for bigM type constraints

int bigM = 1000;

// construct y-z relationship\_1 and y-z relationship\_2 constraints

for (int k = 0; k < K; k++) {

YZ\_Relationship\_1[k] = model.linearNumExpr();

YZ\_Relationship\_1[k].addTerm(1, Z[k]);

YZ\_Relationship\_1[k].addTerm(-1, Y[k]);

model.addLe(YZ\_Relationship\_1[k], 0);

YZ\_Relationship\_2[k] = model.linearNumExpr();

YZ\_Relationship\_2[k].addTerm(1, Y[k]);

YZ\_Relationship\_2[k].addTerm(bigM, Z[k]);

model.addLe(YZ\_Relationship\_2[k], bigM + 1);

}

// construct batchAssignedToOneChute constraints

for (int i = 0; i < I; i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

batchAssignedToOneChute[i][j] = model.linearNumExpr();

for (int k = 0; k < K; k++) {

batchAssignedToOneChute[i][j].addTerm(1, X[i][j][k]);

}

model.addLe(batchAssignedToOneChute[i][j], 1);

}

}

// construct batchAssignedFromSameBoutique\_1 constraints

for (int i = 0; i < I; i++) {

for (int k = 0; k < K; k++) {

batchAssignedFromSameBoutique\_1[i][k] = model.linearNumExpr();

for (int j = 0; j < numberOfItems.get(i).length; j++) {

batchAssignedFromSameBoutique\_1[i][k].addTerm(1, X[i][j][k]);

}

batchAssignedFromSameBoutique\_1[i][k].addTerm(-bigM, s[i][k]);

model.addLe(batchAssignedFromSameBoutique\_1[i][k], 0);

}

}

// construct batchAssignedFromSameBoutique\_2 constraints

for (int i = 0; i < I; i++) {

for (int k = 0; k < K; k++) {

batchAssignedFromSameBoutique\_2[i][k] = model.linearNumExpr();

for (int ii = 0; ii < I; ii++) {

if (ii != i) {

for (int j = 0; j < numberOfItems.get(ii).length; j++) {

batchAssignedFromSameBoutique\_2[i][k].addTerm(1, X[ii][j][k]);

}

}

}

batchAssignedFromSameBoutique\_2[i][k].addTerm(bigM, s[i][k]);

model.addLe(batchAssignedFromSameBoutique\_2[i][k], bigM);

}

}

// construct T\_Either\_Or\_1 and T\_Either\_Or\_2 constraints

for (int i = 0; i < I; i++) {

T\_Either\_Or\_1[i] = model.linearNumExpr();

T\_Either\_Or\_1[i].addTerm(1, T[i]);

T\_Either\_Or\_1[i].addTerm(-bigM, t[i]);

model.addLe(T\_Either\_Or\_1[i], 0);

T\_Either\_Or\_2[i] = model.linearNumExpr();

T\_Either\_Or\_2[i].addTerm(-1, T[i]);

T\_Either\_Or\_2[i].addTerm(bigM, t[i]);

model.addLe(T\_Either\_Or\_2[i], bigM - numberOfBatches.get(i));

}

// this avoids unnecessary console output lines during calculations

model.setOut(null);

// checking whether the model is solved. All following commands depends on this

// condition

if (model.solve()) {

System.***out***.println("Stage " + stage + " model solved! ");

// print the objective function in open form and its optimal solution value

System.***out***.println("Optimal Solution Found!");

System.***out***.println();

System.***out***.println("Objective Function: " + objTerms);

double solVal = model.getObjValue();

System.***out***.println();

System.***out***.println("Optimal Solution: " + solVal);

System.***out***.println();

// // print the constraints: comment in for enabling console output

//

// System.out.println(" ----- Constraints ----- ");

// System.out.println();

// System.out.println();

//

// // XY\_Relationship constraints

// System.out.println(" --- XY\_Relationship --- ");

// System.out.println();

// for (int k = 0; k < K; k++) {

// System.out.println(XY\_Relationship[k]);

// }

// System.out.println();

//

// // XT\_Relationship constraints

// System.out.println(" --- XT\_Relationship --- ");

// System.out.println();

// for (int i = 0; i < I; i++) {

// System.out.println(XT\_Relationship[i]);

// }

// System.out.println();

//

// // YZ\_Relationship\_1 constraints

// System.out.println(" --- YZ\_Relationship\_1 --- ");

// System.out.println();

// for (int k = 0; k < K; k++) {

// System.out.println(YZ\_Relationship\_1[k]);

// }

// System.out.println();

//

// // YZ\_Relationship\_2 constraints

// System.out.println(" --- YZ\_Relationship\_2 --- ");

// System.out.println();

// for (int k = 0; k < K; k++) {

// System.out.println(YZ\_Relationship\_2[k]);

// }

// System.out.println();

//

// // batchAssignedToOneChute constraints

// System.out.println(" --- batchAssignedToOneChute --- ");

// System.out.println();

// for (int i = 0; i < I; i++) {

// for (int j = 0; j < numberOfItems.get(i).length; j++) {

// System.out.println(batchAssignedToOneChute[i][j]);

// }

// System.out.println();

// }

// System.out.println();

//

// // batchAssignedFromSameBoutique\_1 constraints

// System.out.println(" --- batchAssignedFromSameBoutique\_1 --- ");

// System.out.println();

// for (int i = 0; i < I; i++) {

// for (int k = 0; k < K; k++) {

// System.out.println(batchAssignedFromSameBoutique\_1[i][k]);

// }

// System.out.println();

// }

// System.out.println();

//

// // batchAssignedFromSameBoutique\_2 constraints

// System.out.println(" --- batchAssignedFromSameBoutique\_2 --- ");

// System.out.println();

// for (int i = 0; i < I; i++) {

// for (int k = 0; k < K; k++) {

// System.out.println(batchAssignedFromSameBoutique\_2[i][k]);

// }

// System.out.println();

// }

// System.out.println();

//

// // T\_Either\_Or\_1 constraints

// System.out.println(" --- T\_Either\_Or\_1 --- ");

// System.out.println();

// for (int i = 0; i < I; i++) {

// System.out.println(T\_Either\_Or\_1[i]);

// }

// System.out.println();

//

// // T\_Either\_Or\_2 constraints

// System.out.println(" --- T\_Either\_Or\_2 --- ");

// System.out.println();

// for (int i = 0; i < I; i++) {

// System.out.println(T\_Either\_Or\_2[i]);

// }

// System.out.println();

System.***out***.println();

System.***out***.println();

// print the variables and store variable solution values into required integer

// formats instead of double solution values from the model. Comment in printing

// command lines when wanting to see model output values

System.***out***.println(" ----- Variables -----");

System.***out***.println();

System.***out***.println(" --- Assignments in stage " + stage + " --- ");

// print X\_i,j,k values and convert solution values into true integers (only 0-1

// in this case)

// XBool array prevents holding unnecessary values such as 1.0000024 or

// 0.00000024 by forcing them into 1 and 0, respectively.

System.***out***.println(" X\_i,j,k : Boutique i's batch j assigned to chute k ");

System.***out***.println();

for (int i = 0; i < I; i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

for (int k = 0; k < K; k++) {

XBool[i][j][k] = (int) model.getValue(X[i][j][k]);

if (XBool[i][j][k] == 1) { // comment in for printing only assigned values of X variables

System.***out***.print(X[i][j][k]);

System.***out***.println();

// System.out.print(X[i][j][k] + " = ");

// System.out.println(model.getValue(X[i][j][k]));

}

}

}

System.***out***.println();

}

System.***out***.println();

// print Y\_k values and convert solution values into true integers

for (int k = 0; k < K; k++) {

YInt[k] = (int) model.getValue(Y[k]);

// System.out.print(Y[k] + " = ");

// System.out.println(YInt[k]);

}

System.***out***.println();

System.***out***.println(" Z\_k : Chute k has received single assignment if Z\_k = 1 ");

// print Z\_k values

// ZBool array prevents holding unnecessary values such as 1.0000024 or

// 0.00000024 by forcing them into 1 and 0, respectively.

for (int k = 0; k < K; k++) {

ZBool[k] = (int) model.getValue(Z[k]);

System.***out***.print(Z[k] + " = ");

System.***out***.println(ZBool[k]);

}

System.***out***.println();

// Print t\_i values

for (int i = 0; i < I; i++) {

tInt[i] = (int) model.getValue(t[i]);

// System.out.print(t[i] + " : ");

// System.out.println(tInt[i]);

}

System.***out***.println();

} else {

System.***out***.println("Model Could NOT Be Solved!");

System.*exit*(0);

}

//////////////////////

// by this point a stage is completed

// updating model inputs

// batches that belong to used boutiques but have not been assigned as a single

// assignment are redefined as a new boutique

System.***out***.println("----Updating parameters----- ");

System.***out***.println();

// printing numberOfItems before update

System.***out***.println("numberOfItems before update: ");

for (int[] intarr : numberOfItems) {

System.***out***.print("[ ");

for (int j : intarr) {

System.***out***.print(j);

System.***out***.print(" , ");

}

System.***out***.println("] ");

}

System.***out***.println();

// cloning numberOfItems in tempItems

ArrayList<int[]> tempItems = new ArrayList<int[]>();

for (int i = 0; i < I; i++) {

tempItems.add(numberOfItems.get(i));

}

// clearing numberOfItems and numberOfBatches to be filled once again later on

numberOfItems.clear();

numberOfBatches.clear();

// adding previously unused boutiques into numberOfItems

for (int i = 0; i < I; i++) {

if (tInt[i] == 0) { // this condition ensures that the i'th boutique was not used

numberOfItems.add(tempItems.get(i));

}

}

// print numberOfItems after inserting previously unused boutiques

System.***out***.println();

System.***out***.println("numberOfItems after inserting previously unused boutiques: ");

for (int[] intarr : numberOfItems) {

System.***out***.print("[ ");

for (int i : intarr) {

System.***out***.print(i);

System.***out***.print(" , ");

}

System.***out***.println("] ");

}

System.***out***.println();

// creating a new array list of integers to hold batches that will be defined as

// a new boutique. The condition is that these batches were not assigned as

// single assignment to the chutes.

ArrayList<Integer> newBoutiqueList = new ArrayList<Integer>();

for (int i = 0; i < I; i++) {

if (tInt[i] == 1) { // this condition ensures that the i'th boutique was used

for (int j = 0; j < tempItems.get(i).length; j++) {

for (int k = 0; k < K; k++) {

if (ZBool[k] == 0 && XBool[i][j][k] == 1) {

// this condition ensures that boutique i's batch j was assigned to chute k

// but was not a single assignment

newBoutiqueList.add(tempItems.get(i)[j]);

}

}

}

}

}

// converting this array list of integers into an integer array to match its

// type with numberOfItems

int[] newBoutiqueArray = new int[newBoutiqueList.size()];

for (int i = 0; i < newBoutiqueArray.length; i++) {

newBoutiqueArray[i] = newBoutiqueList.get(i).intValue();

}

// the newly defined integer array is added to numberOfItems as a new boutique

numberOfItems.add(newBoutiqueArray);

// printing numberOfItems after complete update

System.***out***.println();

System.***out***.println(

"numberOfItems after complete update after stage " + stage + " with new boutique definitions: ");

for (int[] intarr : numberOfItems) {

System.***out***.print("[ ");

for (int i : intarr) {

System.***out***.print(i);

System.***out***.print(" , ");

}

System.***out***.println("] ");

}

System.***out***.println();

// calculating the total sum of items to be processed as in the beginning to be

// used as looping condition in the while loop

sum = 0;

for (int i = 0; i < numberOfItems.size(); i++) {

for (int j = 0; j < numberOfItems.get(i).length; j++) {

sum += numberOfItems.get(i)[j];

}

}

// stage completed here after all required updates, while loop continues from

// the top

}

// after exitting the while loop, allocation is completed

System.***out***.println();

System.***out***.println();

// printing the total number of stages spent within the system

System.***out***.println("Allocation of orders completed after " + stage + " stages!");

}

// this method changes csv file formats into a list containing triplets

// (triplets written as separate class)

// 0: boutiqueID - 1: batchID - 2: numOfItems

public static List<Triplet> fromCSVtoTripletList(String pathName) throws IOException {

String tempLine = "";

List<Triplet> returnList = new ArrayList<Triplet>();

BufferedReader br = new BufferedReader(new FileReader(pathName));

// writing a readLine command and not using it to ignore first line of titles in

// csv file

String headerLine = br.readLine();

// read through csv file

while ((tempLine = br.readLine()) != null) {

String[] eachLine = tempLine.split(",");

// storing first value of boutiqueID as an integer, second value of batchID as a

// String, third value of NumOfItems as an integer

Triplet tempTriplet = new Triplet(Integer.*parseInt*(eachLine[0]), eachLine[1],

Integer.*parseInt*(eachLine[2]));

returnList.add(tempTriplet);

}

return returnList;