Warehouse Operations

by

Mohammadnaser Ansari

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 12th,2020

Keywords: Performance Analysis, WODS, Data Generator

Approved by

Dr. Jeffrey Smith, Department of Industrial & Systems Engineering, Auburn University
Dr. Jorge Valenzuela, Department of Industrial & Systems Engineering, Auburn University
Dr. Fadel Megahed, Farmer School of Business, Miami University
Dr. Levent Yilmaz, Department of Computer Science & Software Engineering, Auburn University

Table of Contents

[1] Ansari, M., and Smith, J. S. (2020). Design, Storage Location Allocation, and Routing: A Comprehensive Performance Analysis. Submitted to the Computers and Industrial Engineering (C&IE)

[2] Ansari, M., and Smith, J. S. (2017). Warehouse Operations Data Structure (WODS): A data structure developed for warehouse operations modeling. Computers and Industrial Engineering, 112. https://doi.org/10.1016/j.cie.2017.08.009

[3] Ansari, M., Smith, J. S., Rasoolian, B. (2020). Hybrid Synthetic Order Data Generator: A Two-Phase Process in Generating Correlated Order Picking Data. To be Submitted to the International Journal of Production Research (IJPR)

List of Figures

List of Tables

Chapter 1

Introduction

Warehouse operations include the administrative and physical processes related to the storage of materials and goods. Although some view a warehouse as only a place to store goods, the processes of storage, consolidation, packing, and shipping provide substantial economic benefits to both the business itself and the customers. As primary objectives of the logistic systems are lowering costs and improving customer satisfaction by reducing overall inventory and decreasing the cycle time, warehouses play a significant role in achieving these goals. By providing the right product or material at the right time through processes such as order consolidation, assembling the order, cross-docking, etc. to minimize the overall cost, warehouses are a significant part of any supply chain. Operating an efficient warehouse can help the business both reduce the overall cost of storing and retrieving of goods as well as increasing the customer satisfaction.

The Council of Supply Chain Management Professionals (CSCMP) estimated that the US spent an overall of $1.4 trillion in logistic cost in the year 2015 which around $427 billion of it is inventory and the cost associated with storing it. Although the CSCMP breaks down the inventory cost into storage (with $141 billion), financial cost (with $158 billion), and other (obsolescence, shrinkage, insurance, etc. with $128 billion), operating an optimized warehouse process can positively affect the cost in all areas. Considering the enormous amount of money spent every year on warehousing operations and related processes, the vast research done in this area is not far from expectations.

Warehouse problems can be divided into three categories [57]:

- Strategic problems that include issues such as the number of warehouses, warehouse sizes, designs, and equipment. These problems have long term effects on the system.
- Tactical problems which consist of workforce, product allocation, and order picking policies subjects. The problems in this category are considered as issues with mid-term effect.

- Operational problems which include decisions on route selection, batch size, and short term workforce cases. The decisions made here have short-term effects.

The issues regarding the strategic level concerns topics that needs to be addressed as the first step in establishing a new warehouse. These decisions have long term impact and require high investments. Works done by Gue et al. [26] and Meller and Gue [42] on various warehouse designs are examples of research, solving the strategic level warehouse problems. Decisions made in this step are either permanent, or extremely expensive to reconsider.

At the tactical level problems, issues such as workforce and product allocation will be discussed, issues that need to be addressed before the warehouse can be operational. The tactical level decisions have mid term impacts. These decisions, although not permanent, cannot be easily undone. For example, consider the issue of product allocation. Although even after the warehouse is up and running, the location of products in the warehouse can be changed, however, it will be expensive and time consuming. If the correct methods have not been selected during the proper time, implementing new method, even if they out-perform the current method, is not always possible. This happens because sometimes the benefits of the new method, will not out-weight the costs related to implementation.

Operational decisions on the other hand, are considered as decisions with short term impacts. Warehouse operations on its basic level consist of receiving Stock Keeping Units (SKUs), storing the SKUs, obtaining orders from customers and pick them, and consolidating the picked items and prepare them for shipping or in other words receiving, storing, picking, and shipping.

Receiving and shipping are the processes that oversee the inflow and outflow of goods and material into the warehouse. Assigning incoming and outgoing trucks to docks and scheduling their loading and unloading are few examples of these processes [23].

Order picking, as the name suggests, is the process of picking the ordered SKUs from their storage location in the warehouse and prepare them for the next step, which is shipping. A picking process consists of [23]:

1. Batching: In which the orders that need to be picked up together are grouped into one picking batch. The goal is to prevent repetitive picks from one location for multiple orders that their picking window overlap.

2. Routing and Sequencing: Deciding the best sequence and route that the picker should take to pick the batched products. The goal here is to minimize the time and distance that the picker needs to travel to obtain all SKUs in the pick list. Solving this problem falls into

Traveling Salesman Problem (TSP) methods which is one of the most researched topics in Operations Research (OR).

3. Sorting: Is happening when multiple orders are batched into one picking trip. The process is to assign the picked SKUs to the order they belong to. The sorting can occur while picking the items (sort-while-pick) or after all the elements are picked (sort-after-pick).

Our studies identified three major gap in the literature related to LAP and warehouse operation processes. In the following sections, we will discuss these shortcomings in more details.

1.1 Comprehensive Performance Analysis

Improving warehouse processes and efficiency has always been the goal of researchers working in this field. Various methods, improving the processes in different levels of decision making has been introduced by researchers. Introducing new methods require extensive performance analysis to test and validate their effectiveness. Chapter 2 will investigate the effects of four main factors in warehouse operations on order picking performance. These factors include the number of cross-aisles, storage location assigning approaches, Location ranking methods, and routing methods. The effect of each factor and their two-way, three-way, and four-way interactions, will be investigated in detail in Chapter 2 by implementing a full-ANOVA. Moreover, sensitivity analysis will be performed to investigate the effect of moving the Input/Output (I/O) point throughout the warehouse. Finally, the best combination of design, SLAP, and routing that will result in the order-picking process's best performance will be suggested. All these comparisons are done using a real order dataset, publicly available by Instacart [34]. Although multiple performance analysis has been done by researchers such as Petersen and Aase [48], to the best of our knowledge, no research has been done that all mentioned factors are analyzed together by using a real order dataset rather than a randomly generated dataset. Our research in this chapter resulted in the following paper, submitted to the International Journal of Production Research (IJPR):

- Mohammadnaser Ansari and Jeffrey S. Smith, "Design, Storage Location Allocation, and Routing: A Comprehensive Performance Analysis," Submitted to the Computers and Industrial Engineering (C&IE)

1.2 Standard Data Structure

Predefined computing paradigms, standards, and data structures are common in many research domains. Such paradigms and generally accepted data structures improve researchers experience,

efficiency, and throughput by reducing the time spent on details such as internal data structures, inter-functional data structures, file formats, etc. and increasing the time available to focus on intellectual works. The idea of having a unified and standard data structure is not new or limited to warehousing. However, the field of warehouse operations lacks such paradigm that unifies the research community on the modeling data structure. We address this gap in research by introducing Warehouse Operations Data Structure (WODS) in Chapter 3, as flexible, expandable, and lean data structure.

Our previous research on data visualization and analysis tools [4, 45] indicated a need for open source tools and materials for researchers working in the simulation field. This research can be considered a continuation of our previous works on developing tools and algorithms that facilitate the research path. Following are our publications related to this research:

- Mohammadnaser Ansari, Ashkan Negahban, Fadel M. Megahed, and Jeffrey S. Smith, "HistoRIA: A New Tool for Simulation input Analysis,". In Proceedings of the Winter Simulation Conference, IEEE (2014): 2702-2713.
- Ashkan Negahban, Mohammadnaser Ansari, and Jeffrey S. Smith, "ADD-MORE: Automated Dynamic Display of Measures of Risk and Error," In Proceedings of the Winter Simulation Conference, IEEE (2016): 977-988.
- Mohammadnaser Ansari and Jeffrey S. Smith, "Warehouse Operations Data Structure (WODS): A Data Structure Developed for Warehouse Operations Modeling." Computers & Industrial Engineering 112 (2017): 11-19.

1.3 Sample Data and Data Generators

Any new method or approach in tackling warehouse operations problems needs to be thoroughly tested, compared and benchmarked in reference to other methods to evaluate its effectiveness. In order to do such thing, researchers not only need a set of real or at least realistic data, but that data needs to be standardized among researchers to return reliable benchmarking results. However, the importance of protecting customer privacy and companies trade secrets resulted in a shortage of real data available to the research community. On the other hand, the complexity of order data and more specifically the correlation of line-items prevented researchers from creating a realistic order data generator that can mimic the real data, while protecting the privacy and confidentiality agreements. In order to address this gap, in Chapter 4 will develop an order data generator algorithm that extracts the essence of an actual order data and generates an arbitrary number of orders with a arbitrary number of SKUs while keeping the characteristics of the actual

data, such as the correlations between SKUs. Our work for this chapter resulted in the following publications:

- Mohammadnaser Ansari, Behnam Rasoolian, and Jeffrey S. Smith, "Synthetic Order Data Generator for Picking Data," International Material Handling Research Colloquium (IMHRC), 2018
- Mohammadnaser Ansari, Behnam Rasoolian, and Jeffrey S. Smith, "Hybrid Synthetic Order Data Generator: A Two-Phase Process in Generating Correlated Order Picking Data," To be submitted to the International Journal of Production Research (IJPR)

1.4 Conclusion

Our research in the field of warehouse operations has not been limited to just one aspect of the research area. In Chapter 2, we first investigated and analyzed an order picking warehouse's performance using various levels of the Strategic, Tactical ,and Operation problems. We selected designs, SKU location-allocation, and routing methods as the factors in the study. Our research indicated that although adding cross-aisles to a warehouse with traditional parallel aisles will increase its efficiency at first, it will have a negative effect in high number of cross-aisles. That is because the extra width that the added cross-aisle will impose on the warehouse, even without considering the extra cost related to building, will result in longer distances. For the first few cross-aisles, this distance's negative effect is less than the positive effect of cross-aisle in adding flexibility in routing. However, as the number of cross-aisles increases, this consequence will eventually outgrow the positive effect.

Another result from our research in Chapter 2 is the investigation in the effect of moving I/O point in the X and Y direction. Our analysis of the effect of moving the I/O point indicated that moving the I/O deeper inside the warehouse will increase the efficiency. On the other hand, moving the I/O point further from the middle of the warehouse to the corners will decrease the performance. A combination of X and Y movement of the I/O point forms a diagonal line. Choosing any point above the line as the I/O point, will result in better performance while choosing any point below the line will result in a worst performance of the order picking warehouse.

In Chapter 3, we developed a data structure as a common framework that can be used by researchers in the field of warehouse operations. Using the WODS will create a common infrastructure for research in this area, which prevents wasteful programming and modeling of the already existing methods in warehousing operations. This data structure and its modular nature

enable researchers and analysts to easily model and test various designs and configurations, and facilitates the process of what-if and sensitivity analysis. With the onset of the COVID-19, the importance of resiliency and planning for inevitable disruptions has once again come to light. This data structure can be used to substantially decrease the time and effort in designing and testing a resilient warehouse. This data structure is also technology agnostic, meaning it is not limited to particular tools or technologies in the warehouse. Automated Storage/ Retrieval Systems (AS/RS), Carousel Systems, AGVs, lift trucks, and other technologies and vehicles can be implemented in the design and operational steps of the coding process, utilizing the modular structure of the WODS. Along with the data structure, a set of starting point tools has been developed to help researchers and facilitate warehouse operations development and modeling. With the support of the research community working in this field, we believe that WODS can become the defacto standard in the data structure to develop warehouse operations and related programming. To this purpose, all codes and data structures have been published on a publicly available and dedicated website. Research community, interested individuals, and companies can access all materials as well as their documentation on http://warehouselayout.org.

In Chapter 4, we developed the Hybrid Synthetic Order Data Generator, a specialized data generator, using a two-phase algorithm to create order data samples. These sample data are used to test, validate, and benchmark any new methods in supply chain and warehousing optimization and modeling. Generated data has always been in use in research related to supply chain and warehousing. However, the generated data are mostly random data, generated based on each SKU's probability and an average and standard deviation of the number of items per order. However, our work in Chapter 4 resulted in a data generator that keeps the probability of SKUs and the average, standard deviation, and the probability distribution of line per item. On top of all the mentioned characteristics, our Hybrid Synthetic Order Data Generator keeps the real order dataset's correlation characteristics. These characteristics are indicators of customer behavior in purchasing products. In other words, it shows which items are frequently bought together. Managers and analysts use these data in making marketing, sales, and promotion decisions. Researchers in the warehouse operation field, specifically, research related to location assigning problems, also use these data to investigate the effect of storing the frequently bought items close to each other.

Our goal in this research is to propose new methods and approaches and pave the research path for other researchers working in this field. We aim to provide researchers with a set of tools and methods that can help them facilitate the processes in designing, developing, and testing new methods and approaches. This will result in faster model development, which will be a great

advantage to researchers and industries in the evolving economy. Various inevitable disruptions might happen, and the onset of COVID-19 showed a lack of preparation for such scenarios. This was proof that such tools and applications are needed in areas such as warehousing, which enables the researchers to develop and model resilient systems that can be easily tested and implemented. We have also tackled the problem of testing and benchmarking data by providing a realistic order data generator that has long been a problem among researchers in the fields of warehousing. Our research can also be the first step for future developments in tools and methods that can be used to facilitate the process of developing and testing warehouse operation models. The flexibility of the developed tools enables researchers and analysts to implement any method, vehicle, technology, and tools in their study and also test and benchmark their performance.

We encourage the research community to utilize the free and available tools to facilitate their warehouse operations research endeavors and provide valuable feedback and improvements. With the community's help in the research related to both WODS and Hybrid Synthetic Order Data Generator, we hope better tools and codes, improved performance, reduced error, and broader implication can be achieved, which will benefit the research community in return.

Chapter 2

Design, Storage Location Allocation, and Routing: A Comprehensive Performance Analysis [1]

Mohammadnaser Ansari, Jeffrey S. Smith

Abstract

With the current disturbance in the supply chain and sudden shift towards e-commerce with the onset and continued impact of COVID-19, warehousing and warehouse efficiency is once again in the spotlight. Decisions on methods and approaches can affect the performance and efficiency of the warehouse. Analyzing the performance of various methods and approaches in warehousing has been a subject of interest for researchers for a long time. However, a comprehensive analysis that incorporates various levels of problems in a warehouse, using real sample data, has not yet been done. This paper investigates the effect of four main factors in warehouse operations on order-picking performance. The number of cross-aisles, storage location assigning approaches, location ranking methods, and routing methods have been examined using a real order data sample. Later, their effect on the performance of order-picker in a traditional parallel aisle warehouse has been investigated. The authors also performed a full-ANOVA and sensitivity analysis to find the best combination of design, SLAP, and routing, resulting in the lowest order-picking time for a set of order data.

Keywords: Warehouse, Routing, Design, SLAP, Performance

2.1 Introduction and Literature Review

A warehouse performance indicator can be anything such as accuracy, capacity, and response time [57, 60]. However, productivity and cost-effectiveness can be named as the essential factors for warehouse management. About 20% of logistics costs are warehouse related. Kearney and Kearney [36] and Rouwenhorst et al. [57] divide the warehouse operations into four main categories: (1)Receiving, (2)Storage, (3)Picking, and (4)Shipping. Out of these main sectors in the

[1] Ansari, M., and Smith, J. S. (2020). Design, Storage Location Allocation, and Routing: A Comprehensive Performance Analysis. Submitted to the Computers and Industrial Engineering (C&IE)

warehouse, order-picking with 55-65% of warehouse operation cost is the most costly operation [12, 58, 14]. Moreover, more than 50% of the time in the order-picking process is related to the picker travel in the warehouse [14, 63]. Considering these, improving the order-picking process and efficiency not only affects the overall warehouse operation cost, it can also have a tangible reduction in the total supply chain cost. Various factors and decisions can affect the order-picking efficiency. These decisions need to be taken at different stages of warehouse construction and operation.

Rouwenhorst et al. [57] categorize all warehouse problems into three levels:

- Strategic problems which include the number of warehouses, warehouse sizes, designs, and equipment.
- Tactical problems which consist of workforce, product allocation, and order-picking policies.
- Operational problems which include decisions on route selection, batch size, and short term workforce.

Various performance comparisons have been made by researchers at strategic, tactical, and operational levels. Here we will review the literature analyzing and comparing warehouse performance on these levels. We will consider the warehouse design from the strategic level, storage location-allocation from the tactical level, and picker routing from the operational level.

Warehouse designs involve determining overall warehouse structure, sizing, layout, equipment, and operational strategies [24]. Researchers tested different combinations and methods in this area. Vaughan [64], using the Traversal and the Optimal routing method, analyzed cross-aisles' effect on order-picking efficiency. He also developed an ANOVA table with warehouse length, number of aisles and cross-aisles, and number of items per order as factors. Caron et al. [9] took an analytical approach to find the optimal picking area design when using the Cube-per-Order Index (COI) SLAP and the Traversal routing method. Using mathematical formulation for calculating distance, he compares the effect of cross-aisle and I/O point location on travel time. Heragu* et al. [28] introduced an analytical formulation for space assignment in a warehouse with a forward and reserve area. Since the formulation only performs for a small number of SKUs, they also provided a heuristic method. Roodbergen and Vis [55] established an analytical formulation for warehouse design based on travel distances from two routing methods, S-Shaped and Largest Gap. Their research is based on a traditional single-block warehouse with multiple aisles and no cross-aisles. They later validate their analytical method using a simulation model.

Storage Location Allocation Problem (SLAP) is the problem of determining storage location for each product in the warehouse. A suitable SLAP method will profoundly impact the performance of the order-picker in the warehouse. Knowing this, researchers compared several SLAP methods' performances. Frazelle [20] evaluated the Dedicated, the Random, and the Class-Based SLAP methods. Xiao and Zheng [67] first introduces mathematical formulation for SLAP and later suggest a heuristic algorithm to solve the SLAP in a warehouse with a known bill of material (BOM).

Routing is one of the most researched topics in warehousing and logistics. Finding the optimal route for a truck with multiple deliveries or a picker with multiple SKUs to pick in a warehouse is an NP-hard problem. Although the solving methods and tools have been improved substantially, still solving such problems highly depends on the number of picks. This unsolvability, among other factors, leads to the introduction of the various heuristic methods. These methods were analyzed, and if possible, their performance has been compared to other heuristic and sometimes the Optimal methods. Petersen [47] performed a full comparison of routing methods, including the Optimal, the Return, and the Midpoint, in a warehouse with the Random SLAP method. De Koster and Van der Poort [13] compared several routing methods in a warehouse model with a decentralized depot point. The modeled warehouse is a single-block, multi-aisle warehouse with two cross-aisles (front and back). Caron et al. [8] analyzed the expected travel distance of S-Shaped (Traversal) and Return routing policies in a two-block warehouse where the COI SLAP method is implemented using mathematical formulations. Petersen and Schmenner [49] performed a complete comparison of various routing methods (the Largest Gap, the Return, the Midpoint, the Traversal, and the Composite) in various location ranking methods (the Diagonal, the Within-Aisle, the Across-Aisle, and the Perimeter). Other factors such as I/O location, Order size, and order skewness were also investigated. Roodbergen and Koster [53] compared the order-picker performance in a warehouse with two and three cross-aisles under various routing methods. Roodbergen and Koster [54] introduced an upgraded routing method (the Combined+) in a warehouse with multiple cross-aisles. Their results show that the higher the number of cross-aisles, the lower the overall picking time. Hwang et al. [33] evaluated the performance of three routing methods, the Return, the Traversal, and the Midpoint. They modeled the travel distance for the mentioned routing methods and compared their performances using various order sizes, SKU popularity skewnesses, and warehouse length to width ratios.

As the literature review illustrates, there is a gap in research when it comes to a full performance comparison. Although studies such as Petersen and Aase [48] performed a semi-full

Study	Cross-aisle	Routing	Storage Allocation	Real Data
Vaughan [64]	✓	✓		
Caron et al. [9]	✓			
Roodbergen and Vis [55]		✓		
Frazelle [20]			✓	
Xiao and Zheng [67]		✓	✓	✓
Petersen [47]		✓		
De Koster and Van der Poort [13]		✓		
Caron et al. [8]	✓	✓		
Petersen and Schmenner [49]		✓		
Roodbergen and Koster [53]	✓	✓		
Roodbergen and Koster [54]	✓	✓		
Hwang et al. [33]		✓		
Petersen and Aase [48]		✓	✓	
This paper	✓	✓	✓	✓

Table 2.1: An overview of warehouse performance analysis literature.

comparison of warehouse performance under various routing and SLAP methods, complete comparisons using all different numbers of cross-aisles, various routing and SLAP methods are yet to be done in this field. Also, experimenting with real order dataset as the input in analyzing performance makes our analysis more reliable and on-par with real-life case studies, a characteristic that is not common in the literature. This paper uses the largest testing dataset with the highest number of SKUs for analyzing various methods in this field. Table 2.1 represents the aspects studied in the literature reviewed in this paper. As shown in Table 2.1, although a large portion of the reviewed papers studied the effect of various routing methods, that is not the case for the cross-aisle counts and the storage allocation methods. Also, it can be seen that a tiny portion of the reviewed literature is using real data to test, validate, or compare different methods.

In this paper, a full performance comparison of various configurations of the number of cross-aisles, SLAP methods, and routing approaches in a warehouse will be developed and thoroughly analyzed. The rest of the paper is as follows. In Section 2.2, we will provide the details about the simulation model developed in order to perform the tests, and in Section 2.3, the results of the simulation model will be analyzed in detail using statistical tools and methods.

2.2 Simulation Model

To perform the comparison, a fully adjustable simulation model has been developed. AnyLogic® was selected as the simulation software because of its versatile warehouse object library and its flexibility in adding features using Java®. For our purposes, simulation software was preferred

S	Matrix of SKUs with size $s \times 4$ (ID, name, frequency, slot ID)
T	Matrix of slots with size $t \times 7$ (ID, aisle, block, row, position, level, SKU ID)
C	Matrix of SKUs and their allocated location ID with size $s \times 2$
A	Matrix of association rules with size $a \times 2$ (itemset, support)
DB	Database of all orders
O	Current order
V	Locations that needs to be visited
M_l	Location ranking method
M_s	SKU ranking method
M_r	Routing method
P_{np}	Number of pickers parameter
P_{ps}	Picker speed parameter
P_{no}	Number of orders for experiment parameter
P_{nc}	Number of cross-aisles
P_{na}	Number of aisles
P_{nr}	Number of rows per aisle
P_{nl}	Number of levels
P_{ww}	Warehouse width
P_{wl}	Warehouse length

Table 2.2: Table of notation.

over programming languages such as the Python®. Although some works in developing warehouse objects and methods libraries have been done in research [2], a simulation software as flexible as AnyLogic® gives us some edge in parameter adjustment, verification, and experimentation. Using the notations in Table 2.2, Algorithm 1 represents the overall processes in a simulation model of an order-picking warehouse. Figure 2.1 illustrates each step, its parameters, and its corresponding results in creating and developing the warehouse simulation model in this paper. Later in this section, we will explain the input data, warehouse creation process, implemented factors and methods, and the experiments in more detail.

2.2.1 Warehouse Creation

The primary modeled warehouse is an 800 by 540 feet, consisting of 20 aisles and two cross-aisles (top and bottom), with 150 locations in each aisle and nine levels for each location. This basic design, changes, as more cross-aisles are added to the warehouse. With the added cross-aisles, warehouse length increases; however, the number of locations per aisle decreases as the aisle length lessens. This is because the total number of locations in the warehouse should remain constant. Table 2.3 lays out the list of simulation model parameters. Figure 2.3(a) illustrates the basic design of the modeled warehouse with the stated parameters.

Algorithm 1: Order-picking algorithm simulated by AnyLogic®.

```
Create warehouse with parameters P_ww, and P_wl
Add racks with parameters P_nc, P_na, P_nr, and P_nl
Read SKU information file and fill the S matrix
Sort T in decreasing order using the selected M_l
Sort S in decreasing order using the selected M_s
for i ⩽ Size_S do
    T_{i,7} = S_{i,1}
    S_{i,4} = T_{i,1}
for P_no do
    Randomly select O from DB
    for i ⩽ Size_O do
        Set V_i = S_{Oi,4}
    Sort V using the selected M_r
    for i ⩽ Size_V do
        Walk to V_i
        Pick SKU C_{V_i,7}
    Retutn to I/O
```

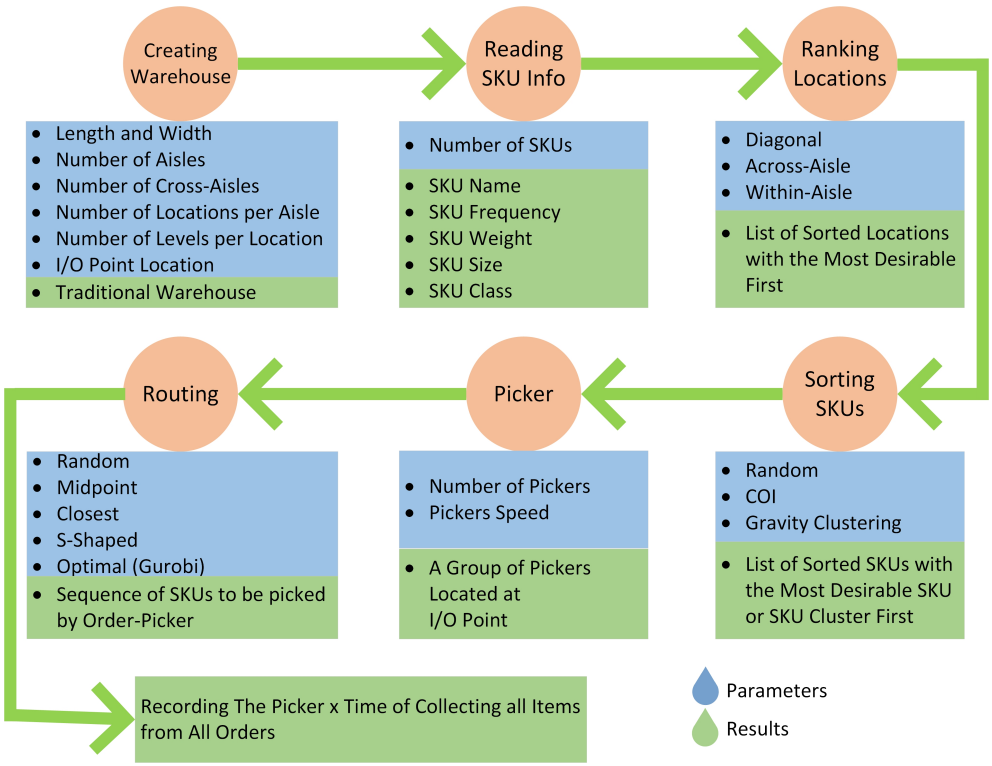


Figure 2.1: The simulation model development flowchart

Number of aisles	20
Number of cross-aisles	2-10
Warehouse length	540-660
Warehouse width	800
Locations in aisle	150-30
Levels in each location	9

Table 2.3: Simulation model parameters.

SKUs	$\approx 50,000$ SKUs
Orders	$\approx 3.2M$ orders
Average lines-per-order	≈ 10
Extracted rules	$\approx 57,000$ Pairs of itemsets
Skewness	20/90

Table 2.4: Instacart retail data

2.2.2 Input Data

Any experiment and performance analysis needs a set of input data that resembles the study's actual events. For our analysis, we used publicly available data from Instacart® [34]. Table 2.4 illustrates Instacart's meta-data. As shown, the orders follow a 20/90 structure, meaning that 20% of SKUs are responsible for 90% of items being ordered. This falls between a 10/90 and 20/80 order structure, which both are considered as highly skewed order data. Figure 2.2 represents the histogram of the number of lines-per-order in the Instacart® order data.

2.2.3 Methods

We implemented various SLAP, location ranking, and routing methods in warehouses with a different number of cross-aisles for our comparison. In this section, we will discuss each varying factor in more detail.

Cross-Aisle

For our performance analysis, we want to analyze cross-aisles' effect on different routing and SLAP methods. For this reason, we will incorporate warehouses with 2 to 10 cross-aisles. Figure 2.3 represents a snapshot of the modeled warehouse. As shown in Figure 2.3(a), we have a minimum of two cross-aisles that act as the front and back corridors.

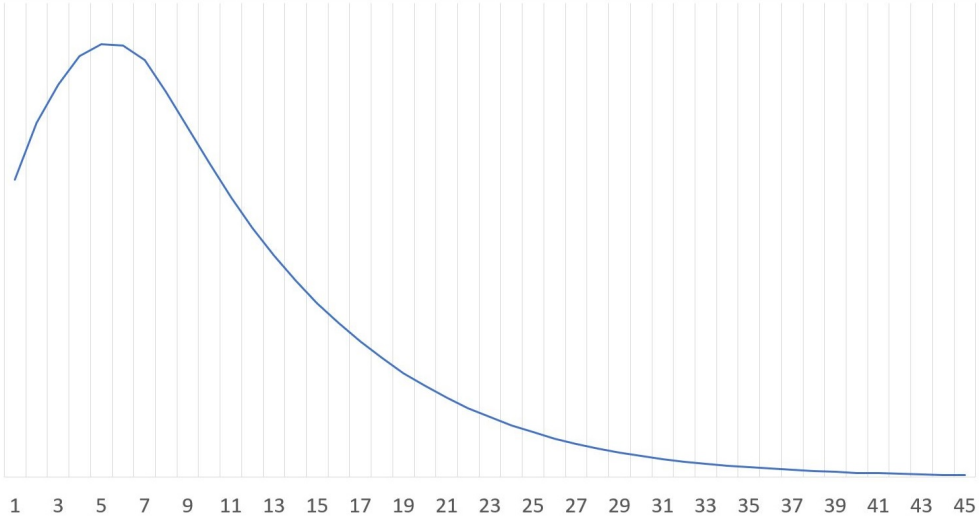
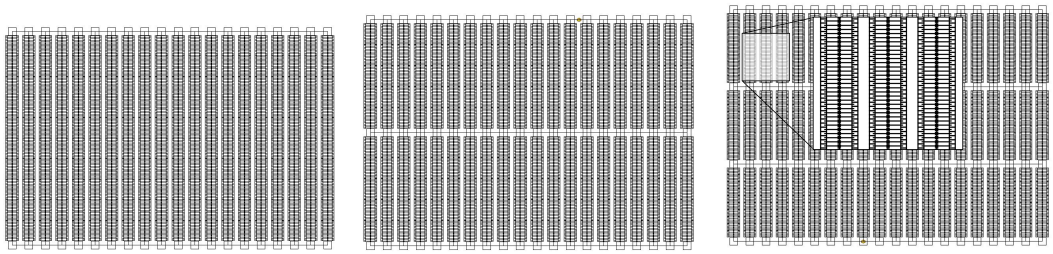


Figure 2.2: Histogram of the number of lines-per-order in Instacart® data



(a) 2 Cross-Aisle (b) 3 Cross-Aisle (c) 4 Cross-Aisle

Figure 2.3: Warehouse Design with Various Number of Cross-Aisles

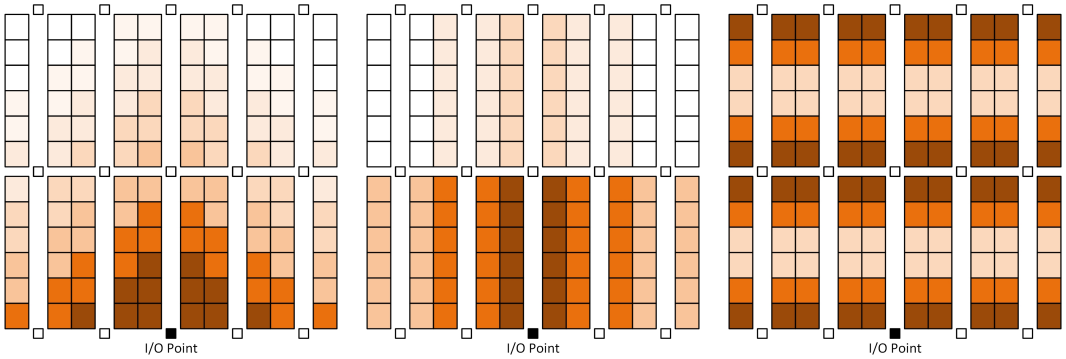
SLAP

SLAP is the process of assigning SKUs to locations in the warehouse. This process can be broken down into three separate parts.

- Ranking Locations is the process of finding and sorting locations based on how appealing they are. The appeal criterion is different for each location ranking method. This criterion is usually a calculated number based on the location distance to a point in the warehouse. Figure 2.4 illustrates various location ranking methods for multi cross-aisle warehouse.
  - Diagonal: Sorting locations based on slot distance to the I/O point (Figure 2.4(a)).
  - Within-Aisle: Sorting locations based on aisle distance to the I/O point (Figure 2.4(b)).
  - Across-Aisle: Sorting locations based on slot distance to the closest cross-aisle (Figure 2.4(c)).
- Prioritizing SKUs is finding and sorting SKUs based on a predefined criterion. Traditionally, SKUs are sorted individually based on the frequency they are being ordered. However, with the use of real data, cluster assigning of the SKUs to locations based on their correlations with other SKUs has arisen. In this paper, we both incorporate COI, as a legacy SKU sorting method, and Gravity Clustering [3] as a correlated approach in sorting.
  - Cube per Order Index (COI): A legacy SKU sorting method introduced by Heskett [29]. In this method, SKUs are sorted based on the frequency that they are picked, per unit of space required.
  - Gravity Clustering: A new method of sorting SKUs based on their frequency and their correlation with other SKUs [3]. In this method, using the gravitational force formula logic, introduced by Sir Isaac Newton, the attraction between the SKUs are calculated. If for the two SKUs i and j, the force of this attraction, $F_{i,j}$, surpasses the predefined threshold value, t, these two SKUs will be clustered together.

$$F_{ij} = \frac{Support_i \times Support_j}{(\frac{N}{Support_{i,j}})^2}$$

    where
    - $Support_i$ is the number of orders containing SKU i
    - $Support_j$ is the number of orders containing SKU j
    - $Support_{i,j}$ is the number of orders containing both SKUs i and j



(a) Diagonal. (b) Within-Aisle. (c) Across-Aisle.

Figure 2.4: Various Location Ranking Methods.

∗ and N is the total number of orders

Algorithm 2 represents the Gravity Clustering method in more detail.

- Assignment is the process of assigning ranked SKUs to their prioritized location counterparts.

Algorithm 2: Gravity Clustering pseudo code.

```
Set A as the association rules with itemsets of size 2 from DB
Sort S in decreasing order based on frequency
while Size_S > 0 do
    Choose the first SKU in S, i
    Create a new cluster with i as the core SKU
    Remove i from S
    for ∀ j ∈ K do
        Calculate F_{i,j} based on A_{[i,j]}, S_{i,3}, and S_{j,3}
        if F_{i,j} > t then
            Add SKU j to current cluster
            Remove j from S
Sort current cluster based on SKU frequency
```

Routing

Various routing methods have been developed to thoroughly analyze the effects of SLAP and cross-aisles on each approach. Five different routing approaches, including the S-Shaped (Traversal), the Midpoint, the Closest, the Optimal, and the Random have been developed. Figure 2.5 illustrates routing methods in a sample warehouse with three cross-aisles.

- In the S-Shaped (also known as Traversal) method, the picker starts from the leftmost aisle which contains an SKU that needs to be picked. The picker goes up from the aisle to

the end of the last block that contains a pick, while picking everything in his way. From that point, one block at a time, if there is a pick in an aisle, the picker will traverse that aisle and picks the SKU on the way. This process continues for every aisle and every block. Figure 2.5(a) represents a sample pick trip using the S-Shaped method.

- The Midpoint routing is similar to S-Shaped. The difference in this method is that the picker will only pick the SKUs that their distance to the picker is less than the distance of the aisle mid-point. In other words, in each block, the picker will first pick the items on the upper half of the block, then the lower half. Figure 2.5(b) illustrates a sample pick trip using the Midpoint routing method.

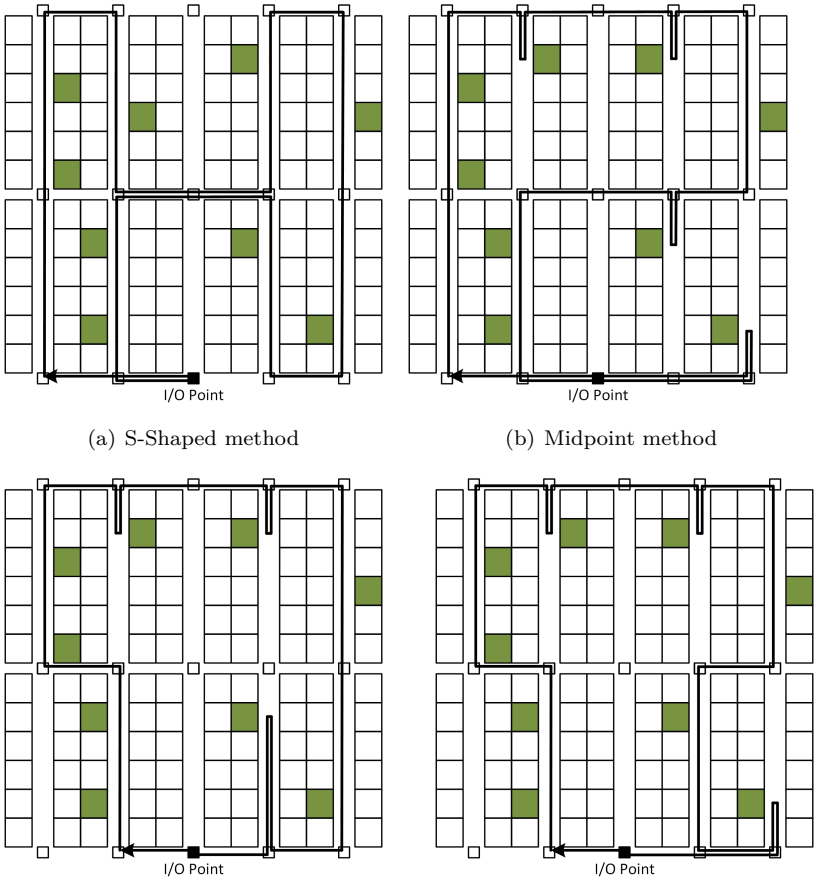
- The Closest SKU routing method, as the name suggests, selects the closest SKU to the picker as the next pick. Figure 2.5(c) represents a sample pick trip using the Closest SKU method.

- The Optimal routing method, using a solver program, finds the optimal route for each pick trip (Figure 2.5(d)). The Optimal routing method in this paper is achieved by the real-time connection of Gurobi® (Version 9.0.0) and AnyLogic®. For each picking trip, a distance matrix based on the target SKUs' locations is calculated and sent to the optimizer. The results of the Gurobi®-solved route will be sent back to simulation software, which will be assigned to the order-picker. Figure 2.6 represents the flow chart of calculating and taking the optimal route for each pick trip (in this experiment, each pick trip is a single order).

2.2.4 Experiment

As seen in Table 2.4, the order data consists of $3.2M$ orders. Because of the enormous number, we incorporated the Bootstrapping method to randomly sample $10,000$ orders from the original set of $3.2M$. Later, to reduce the error accompanying random sampling, we used the Common Random Number (CRN) when testing different simulation configurations [68].

Our main criterion in comparison is the simulation run length, which is the time it takes the order-picker to pick all items of the sampled $10,000$ orders. This means that having multiple order-pickers do not affect the overall $Time \times NumberOfPickers$, which is the response we are interested in. We replicated each simulation configuration five times. This means that each scenario consists of $50,000$ orders, and on average, a staggering number of $500,000$ pickups.



(a) S-Shaped method (b) Midpoint method

(c) Closest SKU method (d) Optimal method

Figure 2.5: Sample pick trip for various routing methods

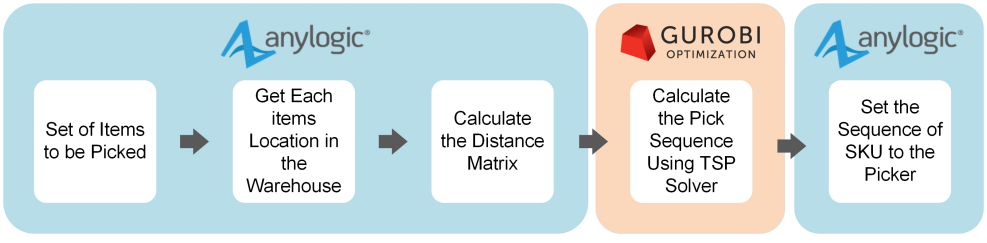


Figure 2.6: Flow chart of finding the optimal routing for each pick trip

Factor	#	Level
Number of Cross-Aisles (CA)	9	2, 3, 4, 5, 6, 7, 8, 9, 10
Routing Methods (RM)	5	Optimal (O), Midpoint (M), S-Shaped (S), Closest SKU (C), Random (R)
Location Ranking Method (LSM)	3	Diagonal (D), Within-Aisle (W), Across-Aisle (A)
SLAP Method (SM)	3	Random (SR), COI (CO), Gravity Clustering (GC)

Table 2.5: Experiment Design Factors and Their Corresponding Levels

2.3 Results

This section, will investigate the effects of various factors in the order-picker performance in a warehouse. Table 2.5 presents the nomenclature for various factors in our study as well as their respective levels used in our experimentation. As shown in Table 2.5, we also studied the Random routing and the Random SLAP. The simulation configuration in which the routing and SLAP are random, and the warehouse has two cross-aisles, gives us the worst-case scenario. We used this worst-case scenario as the baseline that all other configurations can be compared to. In Table 2.6, the improvement percentage over the baseline is presented. Each cell is color-coded, with the significant improvements as Green, moderate improvements as Yellow, and the minuscule improvement percentages (including the baseline with 0% improvement) as Red.

SM / RM / LSM / CA		COI			Gravity			Random
		A	D	W	A	D	W	-
Closest	2	52.3%	66.7%	64.1%	52.3%	66.7%	64.1%	33.5%
	3	36.4%	51.5%	51.3%	36.4%	51.5%	51.3%	45.2%
	4	47.4%	65.5%	65.6%	47.4%	65.5%	65.5%	47.7%
	5	32.0%	49.2%	49.6%	32.0%	49.3%	49.5%	48.3%
	6	40.1%	58.4%	58.7%	40.1%	58.4%	58.7%	48.4%
	7	31.2%	49.9%	50.2%	31.3%	49.9%	50.2%	48.0%
	8	43.0%	63.7%	64.0%	42.9%	63.7%	64.0%	47.4%
	9	39.4%	58.9%	59.2%	39.4%	58.9%	59.2%	46.3%
	10	34.8%	54.4%	54.7%	34.8%	54.4%	54.7%	45.3%
Midpoint	2	34.8%	63.0%	67.0%	34.7%	63.0%	67.0%	28.4%
	3	43.7%	69.4%	70.8%	43.7%	69.4%	70.8%	38.7%
	4	44.8%	71.2%	71.9%	44.8%	71.3%	71.9%	40.9%
	5	45.1%	71.8%	72.1%	45.1%	71.8%	72.1%	41.4%
	6	44.8%	71.9%	72.0%	44.8%	71.9%	72.0%	40.9%
	7	44.2%	71.4%	71.4%	44.2%	71.4%	71.4%	40.2%
	8	43.8%	71.1%	71.3%	43.7%	71.1%	71.2%	39.2%
	9	43.3%	70.8%	70.9%	43.3%	70.8%	70.9%	38.0%
	10	42.8%	70.5%	70.7%	42.7%	70.6%	70.6%	37.5%
Optimal	2	56.9%	71.4%	70.6%	57.1%	71.4%	70.7%	38.6%
	3	55.3%	72.8%	73.0%	55.6%	73.0%	73.2%	48.1%
	4	55.0%	73.6%	73.7%	55.2%	73.8%	74.0%	50.6%
	5	55.0%	73.9%	74.2%	55.2%	74.1%	74.3%	51.3%
	6	54.9%	74.0%	74.2%	55.1%	74.3%	74.5%	51.4%
	7	54.9%	74.1%	74.4%	55.0%	74.2%	74.5%	51.4%
	8	54.7%	73.9%	74.1%	54.8%	74.1%	74.3%	51.0%
	9	54.1%	73.5%	73.7%	54.5%	73.7%	73.9%	49.8%
	10	53.4%	73.0%	73.2%	54.0%	73.5%	73.7%	48.8%
Random	2	7.6%	59.6%	51.9%	7.5%	59.6%	51.9%	0.0%
	3	6.2%	44.6%	43.7%	6.2%	44.6%	43.7%	14.6%

Random	4	13.9%	57.1%	56.6%	13.9%	57.1%	56.6%	15.9%
	5	4.0%	41.8%	42.0%	4.0%	41.9%	42.1%	15.5%
	6	9.9%	50.9%	51.0%	9.8%	50.9%	51.0%	14.3%
	7	2.9%	42.3%	42.6%	2.9%	42.3%	42.5%	13.5%
	8	9.3%	54.8%	54.8%	9.3%	54.8%	54.8%	11.6%
	9	7.1%	50.7%	50.9%	7.1%	50.7%	50.9%	8.7%
	10	4.0%	46.3%	46.5%	4.0%	46.4%	46.5%	7.4%
S-Shaped	2	19.9%	54.6%	66.1%	19.8%	54.5%	66.1%	21.3%
	3	35.0%	66.8%	69.8%	35.0%	66.8%	69.8%	34.3%
	4	38.9%	69.9%	71.1%	38.9%	69.9%	71.7%	37.7%
	5	40.9%	70.8%	71.4%	40.9%	70.8%	71.4%	39.2%
	6	41.4%	71.1%	71.3%	41.5%	71.1%	71.3%	39.3%
	7	41.5%	70.7%	70.8%	41.6%	70.6%	70.8%	38.9%
	8	41.6%	70.5%	70.6%	41.6%	70.5%	70.6%	38.2%
	9	41.5%	70.2%	70.3%	41.6%	70.2%	70.3%	37.0%
	10	41.2%	70.0%	70.1%	41.2%	70.0%	70.1%	36.4%

Table 2.6: Average percentage improvement over baseline

We analyzed the effects of the factors (Table 2.5) and their interactions in Table 2.7, which represents the Full Factorial Analysis of Variance (ANOVA). Using the ANOVA table, we analyze the differences between the various configurations of the factors and their levels. A full factorial ANOVA enables us to not only study the effect and significance of each factor on the overall performance of the warehouse operations, but also analyze the joint effects of all factors and combinations. As shown, all factors and all possible interactions have a p-value of less than 0.001. This means that all factors and interactions of all possible combinations of factors, significantly affect the warehouse operations' overall performance. This shows the importance of studied factors on the performance of order-picking operations.

As it should, the Optimal routing outperforms all other routing methods. Amongst heuristic routing methods (the Closest, the Midpoint, and the S-Shaped), the S-Shaped is the under-performer in all scenarios. However, the best performer varies based on other factors. The SLAP, the Location ranking methods, and the number of cross-aisles, can affect the performance of the routing method and, ultimately, the decision on which method to choose. Based on 63 scenarios

	Factor	DF*	SS**	F-Value	P-Value
Main Effects	CA	8	$5.15e13$	13397.48	0.000†
	RM	4	$1.37e15$	711376.31	0.000†
	LSM	2	$7.86e14$	818389.6	0.000†
	SM	2	$8.36e11$	869908.4	0.000†
Two-way Interactions	CA × RM	32	$6.54e13$	4251.19	0.000†
	CA × LSM	16	$9.86e11$	128.21	0.000†
	CA × SM	16	$4.35e13$	5659.09	0.000†
	RM × LSM	8	$8.45e13$	21992.29	0.000†
	RM × SM	8	$1.06e14$	27659.44	0.000†
	LSM × SM	4	$3.93e14$	204597.4	0.000†
Three-way Interactions	CA × RM × LSM	64	$9.06e12$	294.77	0.000†
	CA × RM × SM	64	$2.99e13$	973.35	0.000†
	CA × LSM × SM	32	$4.23e13$	32.07	0.000†
	RM × LSM × SM	16	$4.23e13$	5498.08	0.000†
Four-way Interactions	CA × RM × LSM × SM	128	$4.53e12$	73.71	0.000†
	Error	1620	$7.78e11$		
	Total	2024	$3.82e15$		

† Significant

* Degree of freedom

** Sum of squares

Table 2.7: Full factorial ANOVA for order-picker total travel time ($\alpha = 0.01$)

that each heuristic routing method is tested in, the Midpoint is superior in 48, and the Closest performs best in 15.

As it was mentioned, Assigning SKUs to a location can be divided into sorting locations and assigning SKUs. Out of the three location ranking methods (Figure 2.4), in all cases, the Across-Aisle approach results in the worst performance because of the I/O location. The I/O point is located in the middle of the lowest cross-aisle, which makes almost all of the locations that are considered desirable in this location ranking method (Figure 2.4(c)), far from the depot. Between the other two location ranking methods, Within-Aisle (Figure 2.4(b)) out-performs Diagonal (Figure 2.4(a)), 39 to 6 out of 45 possible scenarios for the number of cross-aisles and routing methods.

The effect of the number of cross-aisles is different from previous studies [64, 53, 54]. This is because of the extra width that adding cross-aisles will effectuate to the warehouse's design, and consequently, to the distance that the order-picker needs to travel to pick all items. Although the flexibility that extra cross-aisles adds to the routing makes the overall distance shorter for the first few cross-aisles, the negative effect of extra distance negates the initial improvement. Depending on the routing, SLAP, and location ranking method, this shift in performance improvement happens at cross-aisles five to seven. For the Optimal routing method, the Within-Aisle location

ranking method, and the COI SLAP, The best number of cross-aisles in our experiment is five. Adding another cross-aisle and bringing the total number of cross-aisles to six damages the overall performance, and this damage gets worst as the number of cross-aisles increases to higher numbers. The extra added distance, that comes with adding each cross-aisle, also undermines the performance improvement even in the first few cross-aisles. As shown in Table 2.6, in any combination of the SLAP, the Routing, and the Location Ranking method, the improvement with added cross-aisles is not significant specially after the first added cross-aisle.

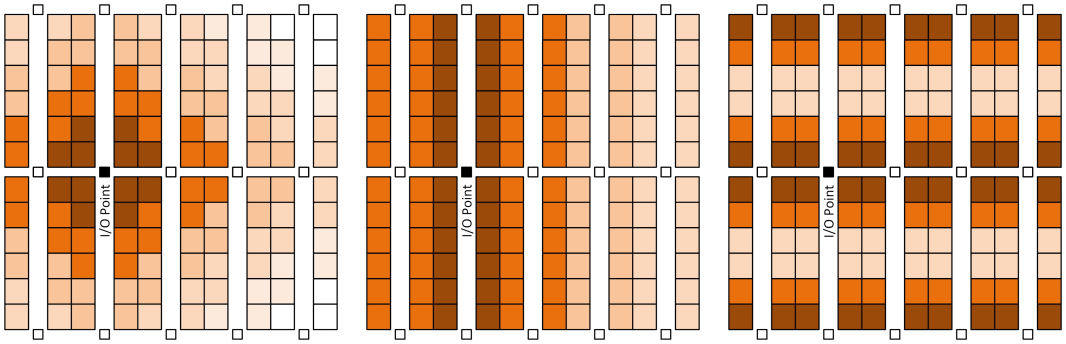
On the other hand, abnormalities can be found in the effect of adding cross-aisles in the Closest and the Random routing method. The performance in these cases fluctuates depending on the number of cross-aisles. The reason behind this comes back to the double-edged sword characteristic of adding cross-aisles. The benefit of adding cross-aisle (more freedom in choosing route) and its side effect (extra added distance), both highly affect the performance. In some cases, the benefit outdoes the side effect and results in performance improvement. In others, the side-effect has a higher effect and causes the overall performance to fluctuate.

Also, as shown, adding the cross-aisles does not significantly improve the results. This is because of the underlying structure of the order data. As it was mentioned in Section 2.2.2, the order data follows a 20/90 pattern. If we analyze the order data deeper, we can see that 70% of orders only consists of 10% of the SKUs. On the other hand, the average lines-per-order is around 10. These mean that in the warehouse of study, for most items that need to be picked up, the order-picker does not have to travel more than a few aisles and cross-aisles further from the I/O point. This reduces the effect of cross-aisle compared to a less-skewed SKU frequency chart, such as an order data with 30/70 or 40/60 structure.

We investigated three various SLAP methods. Although it will result in the best utilization of space in the warehouse [11], the Random SLAP method is the under-performer in our analysis. Between the two other methods, although the differences might be minuscule, the Gravity Clustering performers better than the COI in 75 scenarios out of 135. In the best-case scenario, the Gravity Clustering reduces the simulation time by 0.91%. On average, however, the improvement is 0.20%.

2.3.1 Sensitivity Analysis: Floating I/O Point

As stated by [55], in a traditional warehouse design with not middle cross-aisles (only the two cross-aisle at the top and bottom), it is best if the I/O point is placed in the middle of the lower cross-aisle. This assessment is based on moving the I/O locations from one corner of the warehouse to the other. However, with the various numbers of cross-aisles, the I/O point is not



(a) Sorting Locations based on Slot Distance to Floating I/O Point (Diagonal).

(b) Sorting Location Based on Aisle Distance to Floating I/O Point (Within-Aisle).

(c) Sorting Locations based on Distance to Cross-Aisle (Across-Aisle).

Figure 2.7: Effect of Floating I/O on Various Location Ranking Methods.

limited to linear movements on the lower cross-aisle. For example, by merely putting a conveyor from an intersection of any aisle or cross-aisle to one side of the warehouse, I/O point can take any location in the 2-D plane of the warehouse.

In this section, our goal is to analyze the benefits or drawbacks of moving I/O through various locations in the warehouse. For our analysis, we investigated a warehouse with nineteen aisles and five cross-aisles. We will consider the overall time it takes an order-picker to pick items of $10,000$ orders from the pool of $3.2M$ orders (Table 2.4). Each scenario is replicated three times. Various levels for factors such as the Random, the Closest, and the Optimal for routing factor; the Diagonal, the Within-Aisle, and the Across-Aisle for location ranking methods are investigated. Figure 2.7 presents the changes that floating I/O point will inflict on the location ranking methods. As shown in Figure 2.7(c), changing the I/O point location will not affect the location ranking method logic in the Across-Aisle approach.

The number of levels for the cross-aisle location of the I/O point and aisle location of the I/O point are three and nine, respectively. As shown in Figure 2.8, the area of study for floating I/O is the lower left quarter of the warehouse. Any other location that can be studied for the I/O point in the warehouse, can be reflected upon a point in the selected area. The aisle numbering starts from one as the leftmost aisle and increases to nine as the middle aisle. The same is for the cross-aisle numbering. The lowest cross-aisle is numbered one and increases as we go up to the middle cross-aisle, which is numbered as three.

It should be mentioned that routing methods such as the Midpoint and the S-Shaped are structurally heuristic methods that are started from corner location in the warehouse and moving the I/O points to the middle of the warehouse makes them to not function properly. Also, the

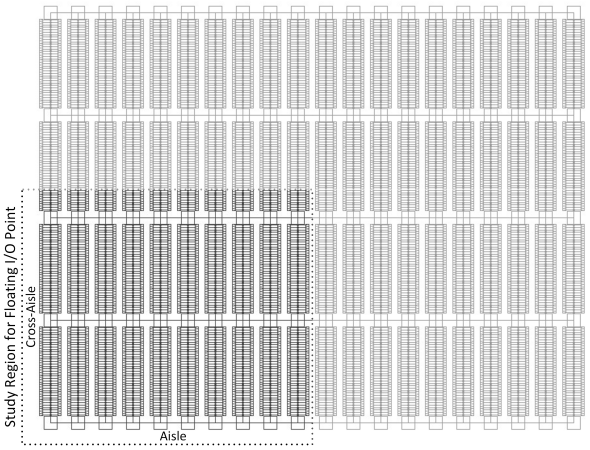


Figure 2.8: The Warehouse Area in Study for Floating I/O Point.

Gravity Clustering SLAP method, as stated in Section 2.3, although statistically different, is not functionally impressive.

Table 2.8 illustrates the ANOVA results for this sensitivity analysis experiment. As shown in Table 2.8, all factors, including routing method, location ranking approach, and the aisle and cross-aisle in which the I/O point is located (I/O A and I/O CA, respectively), are significant.

In Table 2.9, the improvement percentage over baseline for each routing and location ranking method is represented. For each routing method and location ranking approach, the middle-aisle, lower cross-aisle I/O point (Aisle 9 and cross-aisle 1) acts as the baseline value. All values in each routing-location combination are compared to the baseline. As can be seen by taking a closer look into Table 2.9, Moving the I/O point from the middle aisle (aisle 9) closer to the first aisle (aisle 1) decreases the performance of the order-picker in the warehouse. On the other hand, leaving the lower cross-aisle and going up towards the middle cross-aisle increases the modeled warehouse's performance.

As mentioned, moving the I/O point deeper in the warehouse decreases the overall simulation time, or in other words, the time it takes the order-picker to collect all items. The reason for this improvement in performance lies in the strategic location of middle points in the warehouse. Compared to the cases in which the I/O point is located in the lowest cross-aisle, if the I/O point is located somewhere in the middle of the warehouse, the order-picker has access to twice the number of SKUs within the same walking span. This proves that despite the location ranking method, moving the I/O point to higher cross-aisle/aisle intersections will improve the performance.

	Factor	DF*	SS**	F-Value	P-Value
Main Effects	RM	2	2.60e14	995557.34	0.000†
	LSM	2	6.95e14	2663400.71	0.000†
	I/O CA	2	6.77e12	25958.54	0.000†
	I/O A	8	1.42e12	1361.23	0.000†
Two-way Interactions	RM × LSM	4	1.77e11	247685.68	0.000†
	RM × I/O CA	4	1.01e12	1933.74	0.000†
	RM × I/O A	16	4.71e11	225.65	0.000†
	LSM × I/O CA	4	1.77e11	338.56	0.000†
	LSM × I/O A	8	5.31e11	254.54	0.000†
	I/O CA × I/O A	16	3.66e9	1.75	0.035†
Three-way Interactions	RM × LSM × I/O CA	8	8.35e10	80.07	0.000†
	RM × LSM × I/O A	32	1.98e11	47.47	0.000†
	RM × I/O CA × I/O A	32	2.01e9	0.48	0.993†
	LSM × I/O CA × I/O A	32	3.98e9	0.96	0.540†
Four-way Interactions	RM × LSM × I/O CA × I/O A	64	1.38e11	1.66	0.002†
	Error	486	6.34e10		
	Total	728	1.09e15		

† Significant

* Degree of freedom

** Sum of squares

Table 2.8: Full factorial ANOVA for order-picker total travel time with floating I/O point ($\alpha = 0.01$)

Routing	Location	CA \ Aisle	1	2	3	4	5	6	7	8	9
Closest	Across-Aisle	3	-0.06%	1.33%	2.48%	3.32%	4.20%	4.67%	5.23%	5.62%	5.90%
		2	-1.07%	0.25%	1.41%	2.21%	2.88%	3.46%	3.88%	4.35%	4.47%
		1	-5.05%	-3.75%	-2.75%	-1.95%	-1.17%	-0.72%	-0.40%	-0.16%	0.00%
	Diagonal	3	5.54%	7.02%	8.07%	8.72%	9.13%	9.38%	9.48%	9.57%	9.56%
		2	3.73%	5.29%	6.41%	7.06%	7.49%	7.85%	7.94%	8.05%	8.09%
		1	-3.36%	-2.21%	-1.40%	-0.80%	-0.42%	-0.18%	-0.09%	0.01%	0.00%
	Within-Ailse	3	3.36%	4.71%	5.70%	6.19%	6.50%	6.80%	6.94%	6.96%	6.94%
		2	1.82%	3.20%	4.03%	4.66%	5.00%	5.39%	5.55%	5.60%	5.59%
		1	-3.05%	-2.02%	-1.28%	-0.91%	-0.69%	-0.38%	-0.19%	-0.07%	0.00%
Optimal	Across-Aisle	3	1.77%	2.80%	3.51%	3.93%	4.26%	4.47%	4.64%	4.78%	4.81%
		2	0.47%	1.39%	2.15%	2.64%	3.08%	3.40%	3.62%	3.85%	3.79%
		1	-3.87%	-2.70%	-2.02%	-1.26%	-0.76%	-0.28%	-0.25%	-0.07%	0.00%
	Diagonal	3	7.45%	8.33%	8.99%	9.41%	9.52%	9.46%	9.37%	9.26%	9.07%
		2	5.03%	6.03%	6.71%	7.17%	7.23%	7.37%	7.27%	6.96%	6.94%
		1	-1.53%	-0.72%	-0.29%	-0.20%	-0.04%	-0.04%	0.02%	-0.16%	0.00%
	Within-Ailse	3	6.04%	6.87%	7.66%	7.87%	7.92%	8.03%	7.94%	7.78%	7.48%
		2	4.01%	4.88%	5.46%	5.79%	5.83%	6.05%	5.89%	5.72%	5.56%
		1	-0.60%	0.03%	0.42%	0.60%	0.31%	0.38%	0.21%	0.19%	0.00%
Random	Across-Aisle	3	-1.08%	0.43%	1.93%	3.21%	4.43%	5.44%	6.35%	6.90%	7.20%
		2	-2.50%	-1.16%	0.32%	1.61%	2.57%	3.65%	4.30%	4.88%	5.16%
		1	-6.59%	-5.18%	-4.17%	-2.99%	-2.07%	-1.24%	-0.67%	-0.22%	0.00%
	Diagonal	3	9.52%	11.83%	13.14%	13.96%	14.36%	14.55%	14.69%	14.78%	14.75%
		2	7.63%	10.07%	11.59%	12.46%	12.99%	13.32%	13.44%	13.51%	13.55%
		1	-5.75%	-3.35%	-1.93%	-1.09%	-0.62%	-0.30%	-0.11%	-0.06%	0.00%
	Within-Ailse	3	4.42%	6.56%	7.92%	8.59%	8.93%	9.16%	9.28%	9.28%	9.25%
		2	2.87%	5.19%	6.54%	7.32%	7.76%	8.06%	8.15%	8.22%	8.21%
		1	-5.47%	-3.15%	-1.82%	-1.13%	-0.71%	-0.36%	-0.11%	0.04%	0.00%

Table 2.9: Improvement percentage over baseline for floating I/O point

On the other hand, the already better performing Diagonal location ranking method will increase its dominance with a middle cross-aisle I/O point by a variation of the same logic. This is because not only the order-picker has access to more SKUs within the same walking span, the newly accessible SKUs are among the most frequent.

2.4 Conclusions

This paper investigated the effect of various routing, SLAP, location ranking method, and cross-aisle on a manual, multi-item, traditional order-picking warehouse. To do so, a warehouse simulation model has been developed in AnyLogic® experimenting with a sample data of $10,000$ orders from a real dataset pool of size $3.2M$.

Our investigation showed that although adding cross-aisles will increase the order-picker performance, the extra size added to the warehouse because of the cross-aisles width, even without considering the extra cost, adds to the travel time and hence the total order fulfillment time. The best number of cross-aisles, depending on other factors and configurations, is usually between five and seven cross-aisle.

Our analysis of various location ranking methods illustrated that both Diagonal and Within-Aisle are performing well and better than the Across-Aisle method. In the S-Shaped routing method, the Within-Aisle is performing better than the other location ranking methods. Although the Across-Aisle is the worst performer among the three, it performs better in the Mid-point routing method compared to when integrated with other routing methods.

Other than the COI SLAP method, we also investigated the effect of a correlated storage assignment method, Gravity Clustering. Our analysis showed that although Gravity Clustering performs statistically better, the performance improvement is minuscule.

We later investigated the effect of moving the I/O point in both the X and the Y-axis, meaning moving the I/O point to various aisles and cross-aisles. The experiments showed that moving the I/O point deeper in the warehouse increases the order-picking process's performance. At the same time, moving the I/O point closer to the corner decreases the performance.

In this paper, we included most of the well-known and widely used methods in routing and SKU location assigning, and various cross-aisles (within a reasonable and applicable range) in traditional warehouse design. However, there are numerous heuristic routing methods, SLAP, and warehouse design that are not included in this paper. Methods such as Combined and Combined+ in routing [54], Flying-V and Chevron in warehouse design [25, 26, 42], and multiple heuristics correlated storage allocation [37, 7, 6]. As the next steps in the research, these methods need to be compared to other widely used methods, and their interactions with other significant

factors in warehouse performance should be assessed. On the other hand, although a considerable pool of real order data is used for this experiment, the data is limited to one company, Instacart®. Further research can study the effects of various datasets with different characteristics (such as average number of lines-per-order and correlation structure) on an order-picking warehouse's performance.

Chapter 3

Warehouse Operations Data Structure (WODS): A Data Structure Developed For Warehouse Operations Modeling [1]

Mohammadnaser Ansari, Jeffrey S. Smith

Abstract

Generally accepted data structures and predefined paradigms such as geographic information system (GIS) in research domains enhance researchers experience and efficiency by providing defined data structures and file formats. Warehousing and related topics represent one of the most studied areas in supply chain with no predefined standards or data structures when it comes to developing warehouse operation related programs and code. In this paper, we propose WODS, a new data structure that integrates, stores, edits, analyzes, and displays data related to all aspects of warehouse modeling and development. The paper also describes a set of tools to demonstrate the use of WODS which can also play as a starting point that facilitates the process of modeling a warehouse operation program.

Keywords: Standard structure, WODS, Database, Graph

3.1 Introduction

Predefined computing paradigms, standards, and data structures are common in many research domains. Such paradigms and generally accepted data structures improve researchers experience, efficiency, and throughput by reducing the time spent on details such as internal data structures, inter-functional data structures, file formats, etc. and increasing the time available to focus on intellectual works. The idea of having a unified and standard data structure is not new or limited to warehousing. In the field of geography and geology, Tomlinson [62] introduced the term "geographic information system (GIS)" as a general system for storing and displaying geographic information. In the field of mathematics, IBM's format of mathematical programming

[1] Ansari, M., and Smith, J. S. (2017). Warehouse Operations Data Structure (WODS): A data structure developed for warehouse operations modeling. Computers and Industrial Engineering, 112. https://doi.org/10.1016/j.cie.2017.08.009

system (MPS) became the standard after the success of its ASCII LP solver [16]. The format is currently accepted by numerous commercial LP solvers as well as Computational Infrastructure for Operations Research (COIN-OR) open source system. In the filed of operations research, Durillo and Nebro [17] developed jMetal, a Java software framework for solving multi-objective optimization problems. Such data structures are readily available through the referenced standards (whether they are formal standards, de facto standards, or just commonly available and widely used and supported formats). Availability of such structures enabled researchers to focus on the scientific aspect of the research at the same time facilitating to understand the format used by other researchers in the field.

Warehouse operation has been one of the most researched topics in supply chain field (approximately 9,300 warehouse related papers out of 39,900 supply chain papers based on approximation by Web of Science^{TM}). Considering the vast amount of literature in the field and the role that modeling plays in this work, a need for developing a standard data structure emerges. Such data structure should be

- General to encompass all aspects of warehouse operations.
- Flexible to be able to incorporate all available methods and functions as well future approaches in solving any warehousing problem.
- Expandable so that connectivity to other fields of supply chain be possible.
- Lean so that it can be functional with just the basic data necessities.

In this paper, we proposed a general, flexible, expandable, and lean data structure called Warehouse Operations Data Structure (WODS). The WODS is a set of well-defined, related information stores that facilitate the development of formal, unambiguous system descriptions in a given domain. WODS and its constituent components are completely abstracted from any specific picking area layout, order behavior, picking method or implementation. The structure provides ultimate flexibility, allowing the WODS to facilitate the process of modeling any picking layout, regardless of equipment or storage type. This significantly simplifies the development, verification, and sharing of computer models of order picking operations and facilitates the analysis of layouts, resource allocations, and operational strategies. Accompanying the WODS is a set of sample software code as a way of testing the data structure in various scenarios as well as providing examples of one of many ways that the WODS can facilitate the process of programming.

The developed paradigm could have a significant impact on the broader research community. Having a standard data structure for modeling picking warehouses facilitates the sharing of order

picking-related research results, code, and example models and greatly simplifies the independent verification of research results. Further, the sample software tools significantly reduce the amount of time required new researchers need for getting familiar and become productive and ultimately leads to commercial code based on the research results. By providing the source code, models, and examples at a dedicated web site (http://warehouselayout.org) that supports user interaction (comments, general discussion, uploading of code revisions and additional example models), the time required for community acceptance and embrace will be shortened.

In this paper, we first review the current body of literature and its shortcomings in section 3.2. Next, in section 3.3, the WODS, its features, and its merits will be discussed in detail. Later in section 3.4, the data structure is tested by coded methods of solving various warehouse operation problems.

3.2 Literature Review

Literature related to warehouse operations can generally be broken down into four main categories; (1) design; (2) location assignment; (3) sequencing and batching; and (4) picker routing. All of these are well studied in the literature. In this section, we will review some of the papers in warehouse operations, what they have in common, and investigate their missing link.

Warehouse design is one of the topics that attracts a significant research attention. The topic revolves around finding the best warehouse design and layout to reduce picking/stocking time and/or increase storage utilization. It started with optimizing the size [65] and dimensions [18] and later expanded to cover all aspects of layout designs in various storage classes [24] including aisle design. The possibility of the now called traditional designs being replaced by non-traditional designs such as Flying V [26], Chevron, butterfly, and leaf [46] were investigated in this field. The one thing that all of these papers have in common is their representation of the warehouse as a graph which facilitated the design and performance evaluations.

The Location assignment problem (LAP) or slotting is the process of assigning stock keeping units (SKU) to locations in the warehouse. Different algorithms and methods such as cube-per-order index (COI) [8], correlation based storage [67], class based storage [10], and golden zone [50] were introduced and investigated over the years to optimize this process. Like the literature related to warehouse design, the warehouses in these papers are represented by a graph. Moreover, the location assignment problem is based on the design of the warehouse since it is the design that defines a location as good or bad. However, most papers in this area are considering only traditional warehouse designs and they test their methods with the most

popular design for a warehouse (2-by-1, a warehouse with a middle aisle) which itself questions how each method will perform under an alternative design.

Sequencing and batching is the process of assigning the SKUs that should be picked together as a group in one picker trip. The batching process can be as simple as picking all SKUs of an order in one trip or complicated methods such as not picking some families of products with other (e.g. poisonous material with foods), having weight limitations, size limitations, or time constraints [56, 22, 21, 40, 31]. The batching process is highly dependent on other aspects of warehouse operations as well as order behavior and timing. Without a complete warehouse operation model, the methods cannot easily be evaluated in other configurations of a warehouse (different location assignments or warehouse designs).

The picker routing problem is the most studied problem in warehouse operations [23]. The question is to formulate the best route a picker in a warehouse should take in order to pick all assigned SKUs in the shortest time and/or distance possible. The routing problem by itself is a type of traveling salesman problem (TSP) [52]. Many simple and highly used methods including return, traversal, and midpoint [33] as well as heuristic and optimal methods in solving the TSP [13, 53, 13, 38] has been introduced and investigated. The common point in almost all of the papers in this field is the graph representation of the warehouse. However, their methods and solutions cannot be easily evaluated in other configurations of the warehouse.

The vast size of literature in each field shows the enormous amount of time and effort put in this area. However, there is a separation of research in these topics that caused each question to be answered in a different infrastructure. This separation, lack of common data structures, and consequently, difficulty to share the materials of solving and modeling these type of questions resulted in repetition, waste of time, and partial solutions.

3.3 The Proposed Data Structure

In this paper, we developed a data structure specifically designed for the picking warehouse domain, Warehouse Operations Data Structure (WODS). WODS was developed to introduce a common structure for researchers in the field of warehouse operations as a way to simplify the process of storing, retrieving, analyzing, and displaying the data. Furthermore, having such structure facilitates the interaction of various fields of warehouse operation and consequently the knowledge sharing among researchers, much like the role of GIS in the field of geography and geology.

The WODS consists of a set of related information stores and is implemented using a relational database. The basic concept of the WODS is illustrated in Figure 3.1. The implementation

of the WODS consists of the SQL database and the internal graph-based representation (shown with the blue background in the figure). These components are fundamental to advanced model development, yet commonly accepted standards and open source implementations of these standards did not exist in supply chain and warehousing research area. As a result, each research team develops and uses its own implementation based on their own needs. This fragmentation resulted in partial data structures and limitations in code, models, and software tool sharing and limits research reproducibility. These limits progress in the field as researchers are forced to work on the exact same, basic, fundamental components and are unable to leverage other works easily. The proposed WODS goal is to define a standard structure for research in the field of warehouse operations. It should be mentioned that dissemination of the models and tools along with the incorporation of recommendations and work from other researchers in the community is a critical part achieving the goal of this paper. It will only be through the effective dissemination and incorporation of community feedback that the model emerges as a commonly accepted standard.

In this section, we first investigate the WODS itself in more detail in Section 3.3.1 and later the benefit of using a new paradigm will be discussed in Section 3.3.2.

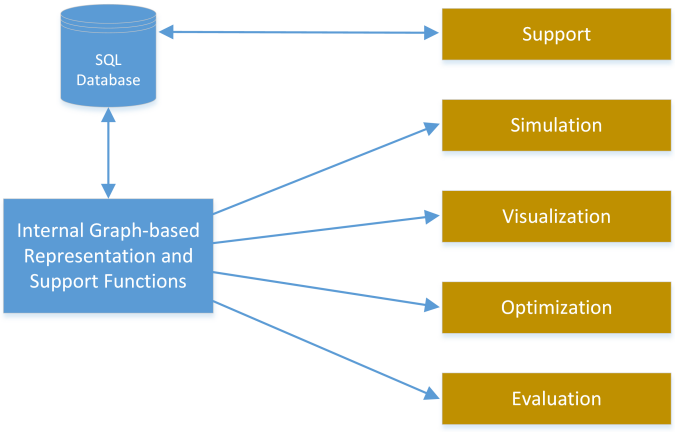


Figure 3.1: Basic concept of the WODS and related tools

3.3.1 Warehouse Operations Data Structure (WODS)

As it was mentioned the goal is to develop an unambiguous description model for picking facilities and areas. The developed WODS includes extensible structures to support early steps of warehouse development (building the actual warehouse structure and designing the aisles) to daily warehouse operations (Assigning stock keeping units to locations, order receiving and batching, picking, etc.). In order to achieve such generality and comprehensiveness in developing

the WODS, we used SQL database software as a widely used and accepted method of information storing. Using SQL results in the ability to store incredibly large amounts of data while not affecting the speed of data storage and retrieval. On the other hand, the WODS is developed with having the goal of independency in mind. In other words, each section of the WODS, while related to other sections, can be managed separately. The benefits of such method of development will be later discussed in section 3.3.2. However, it should be mentioned that WODS is a design for database and its implementation is not limited to SQL and can be implemented in any database management software.

Figure 3.2 shows the entity relationship (ER) diagram for the WODS implemented in MySQL^{TM}. The WODS consists of 6 relational tables. Each part of the picking process is done based on one or more of the tables from the WODS. The structure of the WODS can be broken down into 4 main sections based on their role in the picking system which will be explained in more detail in the following sections. It should be mentioned that the tables can be expanded to add different fields for each entry. The presented tables are merely a starting point for warehouse operations and more complex models might need extra information stored in the tables which this flexibility is already embedded in the WODS. For example, a level definition for each slot can be added to the table and later be used as a measure of elevation and accessibility in some functions (e.g. time it takes to reach to a slot).

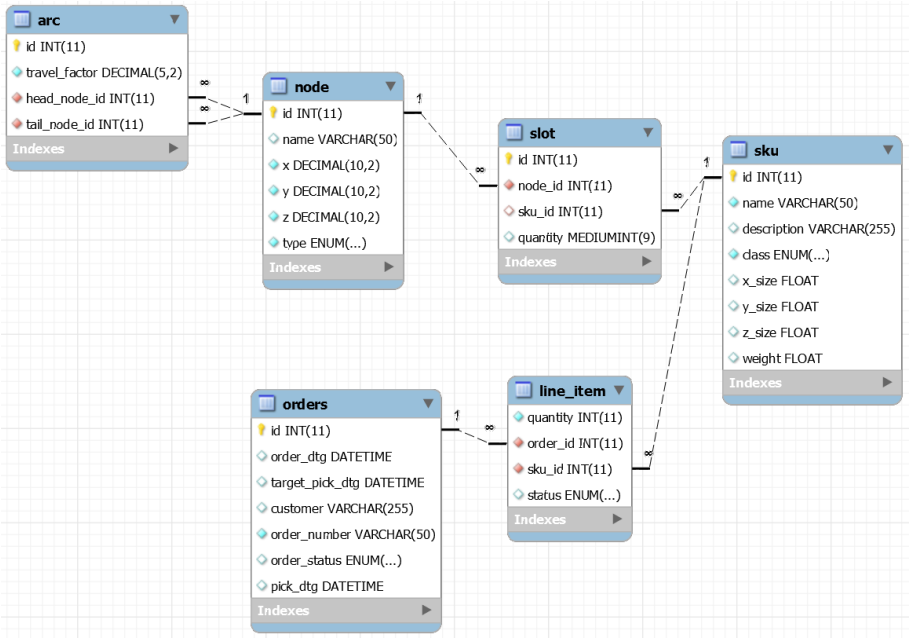


Figure 3.2: ER Diagram

Picker Movement Graph

Picker movement graph (Arcs, Nodes) is a graph style representation of the warehouse based on arc and node tables of the WODS in which the locations that picker might stop in (for picking or dropping purposes) are playing the role of the node and arcs identify the allowable picker movements. Figure 3.3 illustrates a graph representation of the warehouse. Here, the disks represent the nodes, or in other words the physical locations where the worker can/will stop or change direction. Each node has a series of picking locations (slots) assigned to it (illustrated by using the same color as the connected node in figure 3.3) which if the worker wants to pick from those slots, he needs to stop on the appointed node.

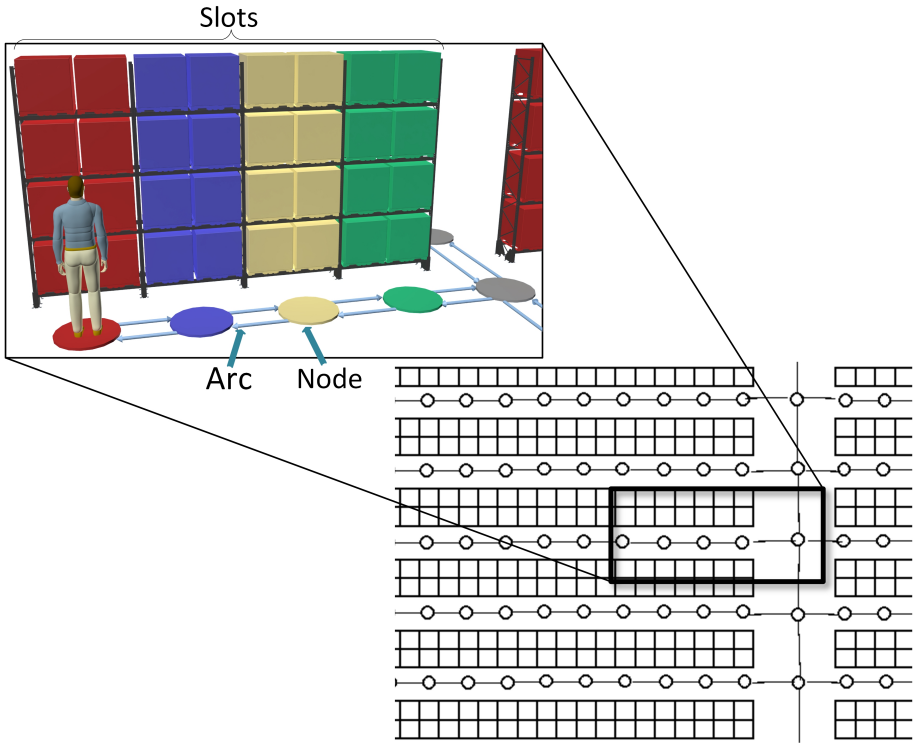


Figure 3.3: Illustration of graph representation approach in modeling a warehouse

Using graph as a method to represent the layout is a commonly used approach, but implementation of such method alongside other warehouse operations as well as its independence from any software platform gives the graph representation method a unique flexibility and comprehensiveness in tackling any design barrier. Examples of multiple warehouse designs (traditional 2-by-1 design and non-traditional Chevron with adjustable angle) done by different software is illustrated in Figure 3.4 in which the graph representation of the WODS is being used.

Node table This table stores the details about each node of the graph. The nodes are represented by their X, Y, and Z value. The location values are relational which means that the 0, 0, 0 point in coordinates can be any point from the warehouse (although this location is commonly assigned either to I/O point or the bottom left corner of the warehouse). As well as its location, each nodes carries the type characteristic which help differentiating between pick nodes (nodes that have slots assigned to them), travel nodes (nodes that are used merely as a travel point), and I/O nodes (nodes in which the products are recived to the warehouse or left the warehouse at them). Other types of nodes can also be defined based on their availability or necessity in modeling the warehouse such as packing station, picker resting area, etc. Name and unique ID of each node can also be specified as an identifier for being able to access each node separately.

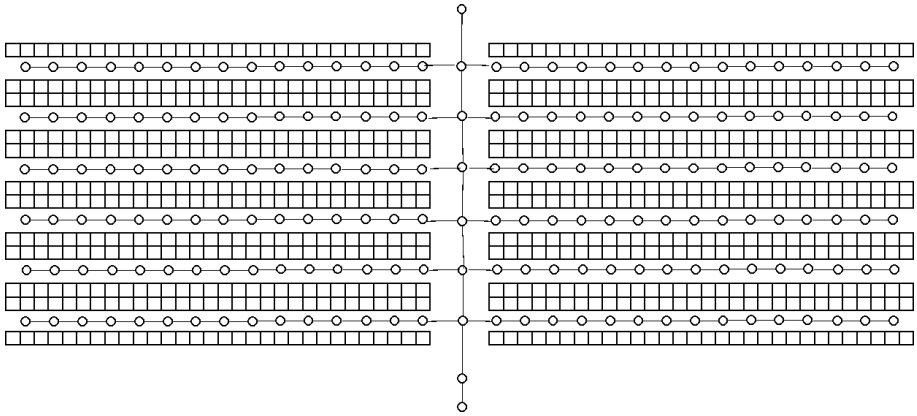
Arc table As its name suggests, the Arc table is responsible for identifying the arcs in the graph representation. Each arc consists of a starting point (tail node) and ending point (head node). This representation method defines each arc as a unidirectional path which prevents confusions about its direction. Also, each arc has a travel factor which is the relative speed that the picker can travel on that specific arc. This feature is most helpful considering that an arc is any allowable movement path for the picker which can mean a simple corridor, a staircase, an elevator, an escalator, etc. and a travel factor differentiates between the speeds that a picker can travel on each of them.

Stock Keeping Units

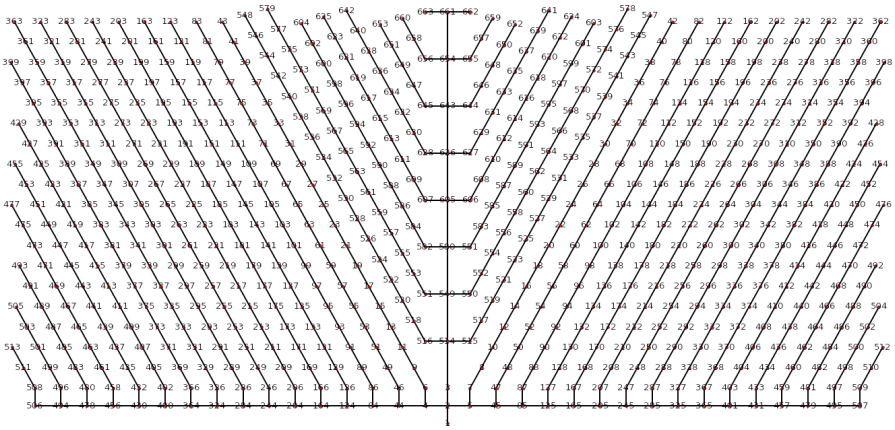
The sku table holds information related to each Stock Keeping Unit (SKU). Physical characteristics such as dimensions (length, width, and height) as well as weight of each SKU are not always a necessity for the WODS to be functional, but in more complicated processes and researches these type of information might be needed which is why a place holder has been assigned to them in this version of WODS. Regardless, as it was mentioned the tables can be expanded not only by the number of rows, but also by the number of columns. Other characteristics can be added to each table and be used in the model for special purposes.

Slots

Locations in the pick area where SKUs are assigned for storage. The SKU ID field links the slot record with the record for the SKU assigned to that slot. Slots are also mapped to nodes in the picker movement graph. A slot being mapped to a picker node indicates that the picker must travel to the node location in order to pick an item from the given slot and this could



(a) Traditional 2-by-1 design (with racks)



(b) Chevron design (graph only)

Figure 3.4: Sample warehouse designs

represent racks, floor stacks or a combination of different physical storage methods/equipment. The slot table also holds information related to the quantity of the SKU it holds in that specific location. The WODS is flexible when it comes to having multiple SKUs in one slot or multiple slots holding the same SKU. This flexibility comes in handy in cases where either the SKU are too small for an assigned slot (e.g. a container with multiple nail types in it) or one type of SKU can be found in different locations through out the warehouse (e.g. having ketchups both in the sauce and frozen food aisles in grocery stores)

Pick Orders

The list of orders that need to be picked is stored in orders and line-item tables. In order to prevent replication of data and for minimization of the data size, the information about the order and the requested items in each one are stored separately.

Orders Holds the information about the order itself. Information such as the customer, order date, requested arrival date, current status of the order(processed, picking, shipping, and shipped), and actual pick date are stored in this table.

Line-Item Stores the data related to each line of the order. A line in order is defined as a request for a specific quantity of a specific SKU. Also, this table stores the information regarding the current status (processed, picking, shipping, and shipped) of that specific line of order (status of order changes if all line items in that order are in the same status)

3.3.2 Benefits

As it was previously mentioned, the WODS can be implemented in any database software (such as MySQL™, Microsoft® Access, Microsoft® SQL Server, etc.), since the design is the crucial part of the WODS, it will not affect the functionality of the data structure. Additionally, breaking down the data storing into multiple relational tables, enables the programmer to work only on one aspect of warehousing rather than the whole process. For example, if we look at warehouse operations as a car, the WODS would be the chassis while each section of it is a part of the car. Now consider a researcher who wants to design an improved version of a car engine. It would be tedious, unnecessary, and wasteful for the researcher to design a whole car from scratch just to be able to test the new engine. This means creating something that already exists and might have been built by an even more specialized team in that field, but was not available for the research community to work on. The WODS plays the same role as the chassis in the example; it can support any programming language as long as it retrieves the raw data from the database and stores the processed version of it back to the database. For example, a Python programmer designs a new warehouse layout, and It stores the information in the database. Another programmer decides to use C# as a programming platform to assign SKUs to locations, so s/he retrieves the data related to the arcs, nodes, and slots from the database, processes them and stores the updated data into the database. And at the end, a researcher uses Java to find the best possible path a picker can take to collect all SKUs and brings them back to the depot point based on the order, warehouse, and slot information it receives from the database. Also, WODS does not confine the user to create or process the data by programming. Tables (all or specific columns of them) can be manually saved to the database while later retrieved by a function. For example, a user can input the data related to design of the warehouse (arcs, nodes, and slots table) manually while later being retrieved by a TSP function. Figure 3.5 illustrates the completion steps the WODS goes through from being

an empty data structure, as a start point, into a comprehensive tool which can be used to store, retrieve, analyze, and display data.

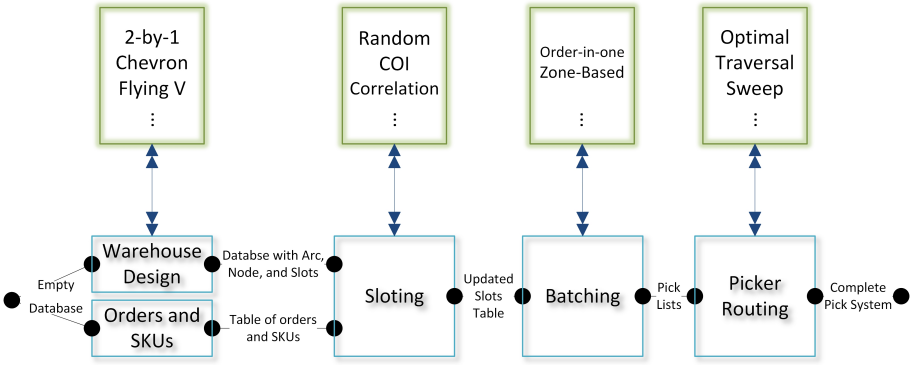


Figure 3.5: Pick system completion steps

Another benefit of using such method is in the field of simulation modeling. Discrete-event simulation has become an indispensable tool for analyzing warehouse and distribution center designs, order picking strategies, and task allocation policies. A scan of the current academic literature on warehouse design, order fulfillment methodologies, and slotting will show that most studies involve simulation with some aspect of the research generally as an evaluation tool to verify the performance of a design and/or operational strategy [44]. However, developing simulation models that incorporate picker movement and individual pick locations can be cumbersome, as the models require their own representation of the picker movement graph. Further, models that compare alternative layouts generally require the graph representation to be implemented for each alternative layout. The process can be significantly simplified by using the WODS to generate portions of the simulation models automatically. Figure 3.6 shows a portion of a Simio^{TM} simulation model where the picker movement graph has been automatically generated using the database and a Simio add-in written in C#. With the add-in implementing the picker movement graph and the custom library object instances modeling the other pick system components, the coding time required to create fully-functional simulation models will be reduced, allowing the researchers to focus more on the experimentation and analysis functions rather than model-building.

The WODS will have a significant impact on the portion of the material handling and logistics community that focuses on computer modeling and analysis of warehouse and distribution system. With the WODS and provided sample code and the user-interactive component

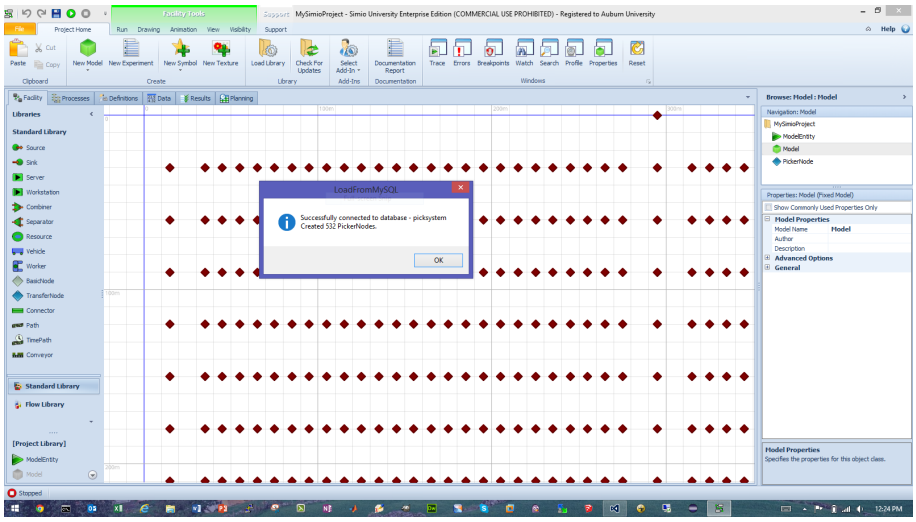


Figure 3.6: Simio add-in to generate the picker movement graph from the database

of http://warehouselayout.org, the work is publicized and the user's feedback will be incorporated. Once critical mass is achieved, the community development process will take over and the resulting WODS and tools will form the basis of the de facto standards that we seek.

3.4 WODS In Action

In this section, we describe a set of sample programs to demonstrate the functionality of the WODS. These programs were designed using known methods and approaches in solving warehouse related problems in designing, location assigning, batching, and picker routing. The purpose is to demonstrate the generality of the structure in being able to tackle all aspects of warehouse operations; its flexibility when adding features and controversial methods; expandability of the structure when being incorporated in other fields of supply chain; and check if it is lean enough to speed up the process of experimentation and data collection. On the other hand, the developed set of sample code can also be used as a set of starting point tools for research community considering that various common methods in different fields of warehouse operations have been coded.

Because it is open source, powerful, and widely common, Python has been used as the main platform for programming the samples. However, Java versions of some of the samples have also been developed. The Python samples, Java samples, and full documentation of the WODS can be found at http://warehouselayout.org website. In this section, a short review of the functions (that the WODS has been successfully tested with) will be presented by reviewing their role and how they work.

3.4.1 Utilizing WODS in Programming Warehouse Operations in Python

The set of sample code simply named Pick System Package is an extension package for facilitating the process of programming a warehouse in Python. The package is an integration of database software (MySQL™ in the current version) with Python module. The package consists of multiple modules, separated based on their role in the pick system. The package does not need installation and importing the Python files into the project will give the user the access to all modules and their functions. Although the functions in the package are newly written, some use existing and commonly available packages. Packages such as NetworkX, Matplotlib, Google Ortools, and Gurobi can be named as some libraries used for some of the functions.

In this section, we will review some of the functions developed in Python with the goal of solving warehouse operation problems by utilizing the WODS. As it was mentioned, the package can also play as a starting point for academic researchers to be able to create, test, or evaluate a specific method in the pick system without the need to code the whole pick system from scratch. The end user can replace the functions in any of the processes with other functions that play a similar role from the package or add a new function with just some minor modification. Figure 3.7 illustrates flow chart of creating a pick system in WODS.

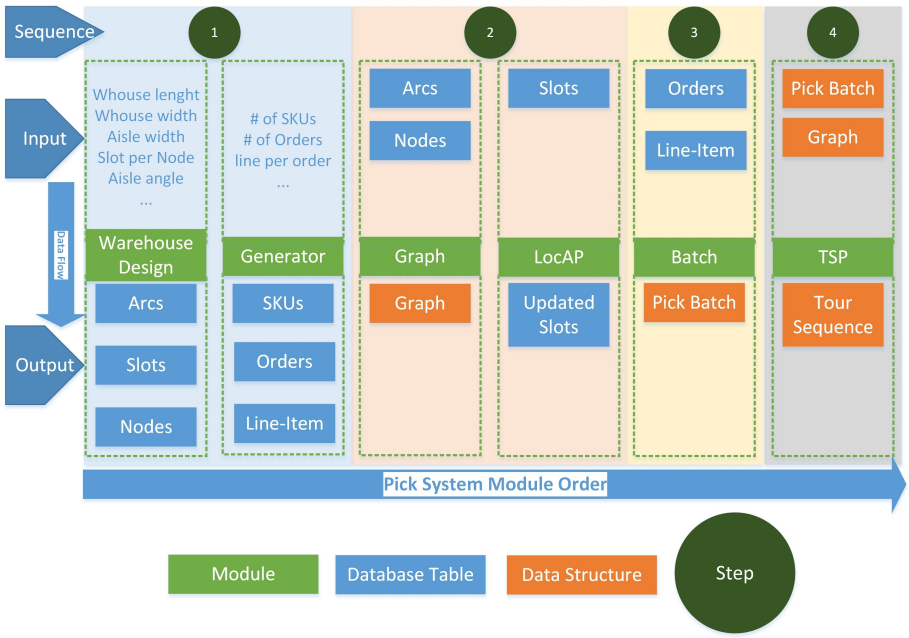


Figure 3.7: Pick System Flow Chart

Figure 3.7 identifies the steps that the user should take in order to create a complete pick system. There are four steps (color coded) that should be passed in the correct sequence. The tasks in each step can be done simultaneously or interchangeably, while the modules to the right

of each module need the output of first module as a starting point. The input/output of each module can be broken down into 4 main categories:

1. Parameters are identified by blue text. These values are user defined and are used in creating the samples and/or designs.

2. Database tables (identified by blue boxes) which represent a table from the SQL database.

3. Data Structures which are kept in memory rather than the database. They can be saved and retrieved, but the the programmer is flexible on which structure needs to be used.

It should be noted that the figures highly simplified the process. Each function in the module is responsible for a part of output and the figure illustrates the cumulative responsibilities of all functions in each module. Further in this section, we will introduce each module and its function based on its sequence superiority. The functions are categorized based on the common research fields and will clarify the ability of WODS in encompassing all aspects of warehouse operation researchs. A more detailed review of each module and a table of all modules and their functions can be found in the package tutorial in http://warehouselayout.org.

Warehouse Design

The warehouse design module (named whousedesign in Python package) is responsible for designing and creating a warehouse based on the specified type. Tables arc, node, and slot in the WODS can be filled with the output of this module. The following functions can be used to create a different type of design based on the user preferences. Alongside the coded functions for this task, we tested the ability of WODS by manually populating the table rather than using computer programming.

- twobyone: The twobyone function creates a two-by-one warehouse based on the given parameters. The two-by-one function is developed to incorporate both traditional and some non-traditional designs such as Chevron. By changing the angle of the aisles to a non-zero value, the design will change from traditional to Chevron.

- draw_whouse: This function visualizes the warehouse based on the exact location of the nodes, arcs and slots. Keep in mind that this function solely draws the nodes, arcs, and slots, not the graph. The output can be used to verify the warehouse design.

Sample Generator

The sample generator module is responsible for creating sample order, SKU, and line-item for testing purposes. Real orders and line-item might have some characteristics such as correlation which are not generated using the current sample generators.

- sku: The sku generator function, creates a list of lists of SKUs in which each list contains a unique id, a name which is generated by adding the "sku" before the id of each sku, a blank description and a class of A for all SKUs.
- order_datebound: The order function creates random order list based on the specified start date, end date and the average number of orders per day.
- order_normal_datebound: The order function creates random order list based on the specified start date, end date, the average number of orders per day, and standard deviation of the normal distribution.
- line_item_fixn: The line-item generator, generates random lines per each order. The number of lines per order and the quantity of the SKU in each line is asked from the user. The generator selects random SKUs from the sku list for each line.

Graph

The graph module of the package uses the available packages in Python designed to develop graph as a tool to create a graph representation of the warehouse. Some of the function in this group need to retrieve data from the database which for that a set of tools for database connection management has been incorporated in the Python package but not presented in this paper. The complete documentation of all functions can be found at http://warehouselayout.org.

- nx_create: Creates a graph using the package NetworkX.
- nx_dijkstra_dm: The distance matrix (DM) in this function is calculated via Dijkstra algorithm (a well-known algorithm in calculating node distances in a graph). The output of this function can be used for TSP calculations.
- nx_draw_graph: This function visualizes the graph as a plot as well as saving the plot in .png format. This function can be used to as a visual aid in creating the graph and the warehouse

Location Assigning Problem (Slotting)

The location assigning problem (named as locap in the Python package) in the warehouse is the process of assigning the SKUs to Slots. The smarter the assignment, the shorter the pick process. The locap module of the package contains functions that assigns SKUs to slots entirely or at least eases the process.

- random: The random function of the locap is creating a random assignment of SKUs to locations.
- coi: This functions uses one of the most famous location assigning methods called Customer Order Index (COI) in assigning the SKUs to Slots in the warehouse. The COI method assigns the most popular SKUs (The one that has been ordered the most) to the best locations in the warehouse (The ones closest to the depot point).

Batch

The batch module of the package is responsible for assigning the line-items that should be picked in one trip into a group. The pick sequencing can be based on the picker capacity, the items relevance (e.g. the chemicals should not be picked in the same tour as groceries in some cases), pick tour limitations, and many more. This module of the package can be used to implement different pick sequencing. Following is the available module in the current version of the package.

- order_in_one_all: Creates pick lists based on the order in which the lines belong to. Each outputted pick sequence contains all the lines in a single order. The output of this function creates a list of list assigning all items to pick batches.
- fixed_batch_size_all: Creates pick lists based on a fixed batch size amount. All batches will be exactly the same size (Except the last batch which assigns all the remaining items in one batch even if the count is smaller than the batch size). The output of this function creates a list of list assigning all items to pick batches.
- sku_to_node_pick: The sku_to_node_pick transforms the SKU pick list (which contains the SKUs that should be picked in each tour) into their corresponding nodes. Each node possesses a physical address in the warehouse and so in the graph representation of the warehouse. The outputs of this function can be used in order to calculate the shortest path that the picker can take to pick all SKUs as well as guiding the picker to the exact location (slot) of the SKU. The function automatically ignores the SKUs that should be picked but there is no stock left for that SKU.

TSP

The Traveling Salesman Problem (TSP) calculates the best order of the nodes that should be visited in a graph to reduce the total traveling time or distance. In this concept, the corresponding node to each slot (which itself was selected based on the SKU that the picker wanted to pick) will play the node in the graph role and the picker will be the traveling salesman. Utilizing such algorithms can benefit us in achieving the best (or in some cases close-to-best) path the picker should take. The following are the functions that can be used to achieve this goal in the current version of the package.

- google_sku: The Google function of TSP module uses a TSP algorithm that google incorporates in google maps website and app to find the shortest path between two or more (in multiple destination routings) locations. It should be mentioned that the method uses a heuristic algorithm which does not guarantee the optimal solution.

- google_node: Has the same functionality of the google_sku function. The only difference is that in this case, the function receives the nodes, and slots that it should visit as input rather than the SKUs that it should pick and list of all slots and the SKUs they are holding.

- google_distance_matrix: Has the same role of the google_sku and google_node function. The only difference is that in this function, rather than using the nx_dijkstra_dm function to calculate the distance matrix, it gets the already calculated distance matrix. This function can be useful when the user wants to use another method rather than the default for calculation the distance matrix.

- gurobi_sku:This function calculates the optimal route that the picker should take. The function uses the Gurobi Python package. The parameters and constraints in this function can be manually changed in order to relax or add some constraints.

- gurobi_node:Has the same functionality of the gurobi_sku function. The only difference is that in this case, the function receives the nodes, and slots that it should visit as input rather than the SKUs that it should pick and list of all slots and the SKUs they are holding.

- gurobi_distance_matrix: Has the same role of the gurobi_sku and gurobi_node function. The only difference is that in this function, rather than using the nx_dijkstra_dm function to calculate the distance matrix, it gets the already calculated distance matrix. This function can be useful when the user wants to use another method rather than the default for calculation the distance matrix.

3.5 Conclusions

In this paper, we proposed a generalized data structure, simply named Warehouse Operations Data Structure (WODS) as a common framework to be used by researchers in warehouse operations field. Using the WODS will create a common infrastructure for research in this area, which prevents wasteful programming and modeling of the already existing methods in warehousing operations. This data structure and its modular nature enable researchers and analysts to easily model and test various designs, facilitating the process of what-if and sensitivity analysis. With the onset of the COVID-19, the importance of resiliency and planning for inevitable disruptions has once again come to light. This data structure can be used to substantially decrease the time and effort in designing and testing a resilient warehouse. This data structure is also technology agnostic, meaning it is not limited to particular tools or technologies in the warehouse. Automated Storage/ Retrieval Systems (AS/RS), Carousel Systems, AGVs, lift trucks, and other technologies and vehicles can be implemented in the design and operational steps of the coding process, while using the modular structure of the WODS.

We believe that by the support of the research community in this area, the WODS can be the de facto structure when it comes to the development of warehouse operations related programmings. We also developed a set of starting point tools that can facilitate the process of modeling a warehouse and save time for the researchers studying this area. All of the proposed WODS configurations as well as all programming code provided by the research community will be placed on the dedicated website for the public and research community to use (http://warehouselayout.org). This research and the proposed WODS are highly dependent on the feedback and support of the research community in order to expand and achieve their full potential. Future research on this topic can be expanding the provided package, developing a standalone tool or website that can build and run any available or newly uploaded code with other parts of the code, and creating a standard set of data that helps in evaluating all methods without bias and with a common ground.

Chapter 4

Hybrid Synthetic Order Data Generator: A Two-Phase Process in Generating Correlated Order Picking Data [1]

Mohammadnaser Ansari, Jeffrey S. Smith, Behnam Rasoolian

Abstracts

Sample data are in high demand in warehouse operations and order-picking research. Researchers in these fields use sample data for testing, validating, and benchmarking purposes. In order-picking research, sample data makes testing new methodologies, such as storage allocation methods, batching and clustering algorithms, and routing approaches, easier and more accessible. Unfortunately, concerns regarding intellectual property and confidentiality agreements between companies and consumers, limit the availability of such data for research purposes. This need for and lack of such data results in researchers using generated order data. However, order data follows a complex structure with complex correlations between items, which makes generating a realistic order data difficult. This paper presents a two-phase approach for generating order data based on a small set of real data. This Hybrid Synthetic Order Data Generator mimics the real order data characteristics and generates a new set with an unlimited number of orders and SKUs. We later investigate, analyze, and compare the proposed data generator's performance to two separate real order datasets.

Keywords: Synthetic Data, Association Rule, Correlation, Data Generator

4.1 Introduction and Literature Review

Huge amounts of data are being generated every second, and the rate of generation is only accelerating with 2.5 quintillions (10^{18}) bytes per day by 2018 estimates [15]. In addition to videos, images, soundtracks, emails, and many more data types, this data includes almost every transactional activity in the retail industry. By mining this transactional data, companies

[1] Ansari, M., Smith, J. S., Rasoolian, B. (2020). Hybrid Synthetic Order Data Generator: A Two-Phase Process in Generating Correlated Order Picking Data. To be Submitted to the International Journal of Production Research (IJPR)

find patterns in their customers' behaviors, keep their business up-to-date and agile, and their marketing on-point and precise. Unfortunately, most of this data is restricted for researchers. Various data privacy and security regulations keep the data away from the public and research community. This restriction is also valid for retail order data, a record of every purchase in a store. On the other hand, such data are highly needed by researchers in warehousing and supply chain fields to test and validate methods and approaches through experimentation.

To propose, test, and validate new methods, simulation models and experiments are developed to summarize the system's current state in a computerized model. Using the simulation model, researchers can then test various scenarios and algorithms without the high cost or the risk of malfunctioning that can happen if the test is done in real life. Like any other scientific method, the simulation environment's methods need to be thoroughly tested and verified before being implemented in real life. Ideally, testing and verification are done using data sampled from the real system. However, as discussed before, data confidentiality restricts or outright prohibits access to real data in many cases.

Researchers use sample data in model development, testing, validation, and benchmarking. This means that numerous research fields require such data. Utilizing Synthetic Data is an approach to solve the problem by providing sample test data not obtained by direct measurement. This method of data generation can help fill the gap caused by the lack of real data. Synthetic data contains all important characteristics of real data without violating any confidentiality rules and agreements. By having real data characteristics, synthetic data can replace real data in testing, validation, and benchmarking of models. Research shows that not only are Synthetic Data a good replacement, in some cases, they even surpass the suitability of real data [41].

Unlike the field of warehousing, other fields were more responsive to the need for sample data. Software and application development are some of the fields that use synthetic data. Data generators are used in application development such as Optical Character Recognition (OCR) software to evaluate and improve the accuracy of the application [30, 35]. Stress testing data warehouses is also another example of generated data being used. A standard approach to evaluating data warehouse performance is by testing the data warehouse's capability to handle a huge size of data [61, 19]. To be able to perform these tests, specialized data generators are developed. Server-side applications and websites are not excluded from vigorous stress testing, and various specialized data generators are created for this purpose. [39, 66]. Also, with the emergence of the Internet of Things, data generators are in higher demand, and to support the research of tools for IoT dataset generation, Anderson et al. [1] developed the implementation results of synthetic IoT data generation framework.

In environmental research, other than the cost of obtaining accurate data, researchers sometimes need data from environmental scenarios that have not actually happened. These data are widely used in what-if scenario analysis and forecasting. Rain-fall data [32] and weather data [51] can be named as some of the synthetic data uses in environmental research. On the other hand, tools and technologies such as sensors play a substantial role in environmental monitoring and seismic activities. Yu et al. [69] investigate the use of synthetic sample data to test the sensor performance in simulated situations.

The goal of any data generator is to create a dataset that is similar to the real data. This similarity depends on the context that the sample data is developed for. In the context of server-side applications, the sample data needs to mimic a user's behavior in the application. In optical character recognition software testing, the sample data is an image of a text. In the context of warehouse operations research, the generated sample order data needs to resemble real customers' behavior in purchasing. This resemblance can be broken down into the following categories:

- Number of lines per order (distribution, average, and standard deviation)
- Probability of each SKU appearances
- Correlations between SKUs (will be discussed in more detail in Section 4.2)

Despite the need, to our knowledge, no synthetic data generator exists in the order picking/warehousing domain that can create data that resembles real data in all categories. On the other hand, having such a data generator can provide researchers with the required data to test new methods and provide evidence on its performance. Such a method can be incorporated in warehouse development packages, such as what is done by Ansari and Smith [2] and simulation software.

Researchers currently use random data generator methods to generate sample order data. However, the resemblance of the sample data generated by the current data generator methods in this field to the real order data is low. This paper we will introduce a new Hybrid Synthetic Order Data Generator (based on the Synthetic Order Data Generator introduced by Ansari et al. [5]) that will help researchers in generating sample data that highly resembles the real order data. The resemblance of the generated order data to real order data will improve the testing and benchmarking of newly introduced methods in the field of warehouse operations and order-picking research.

As mentioned, the method introduced in this paper improves the method introduced by Ansari et al. [5] with the inclusion of performance analysis and comparison. This paper first explains our proposed data generation method and its difference from the current defacto methods in more detail in Section 4.2. Later, the Hybrid Synthetic Order Data Generator's performance analysis will be performed in Section 4.3, presenting the full capabilities of this method.

4.2 Hybrid Synthetic Order Data Generator

Our review of eleven random warehouse operations performance analysis papers indicated that in all of them, the approach to testing new methods utilizes generated data. However, in the ones that go into more detail regarding the testing data, the sample data follows a simple random order generation method. Although the random data keeps the Stock Keeping Units' (SKU) relative frequencies and the average size of orders close to the real dataset, the correlations of SKUs that affect which items should be put together in the same order are entirely ignored. It should be mentioned that this "Correlation" refers to the general definition of the word, meaning association and relation. As the authors, we disagree with using this term because it has a unique and different meaning in mathematics and statistics. However, the word is widely being used by academia and industry and has become the defacto word in referencing the association between the SKUs. This paper will follow the same nomenclature for clarity and unification with other publications in this area.

If we consider the notation in Table 4.1, Algorithm 3 represents the approach widely used by researchers in generating sample order data.

M	Matrix of SKUs with size $m \times 1$
I	Number of items in the real dataset
s_i	Frequency of SKU i (where $i \in M$)
S	Matrix of size $m \times 2$ containing cumulative SKU frequency
a	The average number of items per order
d	The standard deviation of the number of items per order
f	Distribution of the number of items per order
n	Desired number of orders

Table 4.1: Table of notation.

It should be mentioned that Algorithm 3 represents the case that a sample set of order data is being generated from a set of known SKU frequencies and known order distribution. In cases that this information is not available, or in other words, the sample data parameters are not extracted from real data, methods such as ABC are used to estimate the values. ABC is

Algorithm 3: Generating order data based on SKU frequency and number of lines-per-order distribution.

```
1  Fill matrix S by calculating the frequency of each SKU
2  for n do
3      NumberOfItemsPerOrder = f.Inverse( Random(0,1), a, d)
4      TempS = S
5      for NumberOfItemsPerOrder do
6          RandomSelection = Random(0, TempS_last)
7          for m do
8              if (S_x > RandomSelection)&(S_{x-1} < RandomSelection) then
9                  Add SKU x to order
10                 break loop
11         Remove SKU x from TempS
```

fundamentally based on Pareto distribution and logic. This method breaks down SKUs into three main categories:

- Category A represents a small number of SKUs responsible for a large portion of items being ordered. A 20/80 ratio is widely used by researchers, which means that 80% of items that are being ordered are only from 20% of the SKUs.
- Category B is the middle ground between category A and category C. These are the SKUs that are being ordered as if all SKUs have equal chances of selection. 10/10 is an example of the SKU/Item ratio that represents SKUs in category B.
- Category C is the opposite of category A. The SKUs that belong to this group are the ones that are rarely ordered but make up a large portion of total SKUs. For example, a 70/10 ratio for category C means 70% of SKUs are only responsible for 10% of order lines.

Researchers argue that category B is unnecessary, enabling ABC to be represented by numbers from category A, such as 20/90, 20/80, and 30/70 [64, 48, 33].

As can be seen in Algorithm 3, The SKUs assigned to the same order are randomly selected from the list of SKUs based on their frequency. This means that the SKU selection process is independent of the order's current state. This means that having a specific SKU already in the order does not affect the selection probability of the next SKU/SKUs. To understand the problematic effect of SKUs being independent, we first need to talk about the SKU correlation.

Consider three sample SKUs in a basic grocery store: Doughnut, Muffin, and Coffee. Doughnut and muffin are negatively correlated. This means that choosing one reduces the chance of the other one being chosen. On the other hand, both doughnut and muffin have positive correlations with coffee. If someone has coffee in her/his order, the chance of muffin or doughnut being selected in that order increases. This is also true if the first item in the basket is either

doughnut or muffin, as both will increase the chance of coffee to be put in the cart. If these correlations are ignored in data generation, although all SKUs' frequency will remain the same, the orders do not resemble the real order data. These correlations between SKUs are mined by various Association Rule Mining tools and are utilized by managers in making marketing, sales, and promotional decisions. On the other hand, researchers are exploiting these correlations in improving Storage Location Assigning Problem (SLAP) methods in warehouses and retail stores [37, 70].

These correlations between SKUs are extracted using the Association Rule Mining methods. To explain the Association Rule Mining methods, we first need to explain a few terms:

- Item: Each line of an order or each asked SKU of an order.
- Itemset: Is a set of items or SKUs.
- Rule Support: The proportion of orders that contain specific SKU X and SKU Y. $Support(X \rightarrow Y) = Support(X + Y)$ the probability of SKU X and Y appearing in the same order where X and Y are two distinct SKUs.

Every rule illustrates a correlation between two or more products with calculated support. The rules are not limited to two-way correlations. Three-way correlation such as X,Y, and Z are also considered and are calculated as follows:

- $Support(X, Y \rightarrow Z) = Support(X, Y + Z)$ the probability of SKU X,Y, and Z appearing in the same order where X,Y, and Z are three distinct SKUs. The same concept is applied to all other possible combinations of SKUs with any size.

Association Rule Mining is the process of extracting these rules from the order dataset. One of the most famous algorithms in extracting association rules is the Apriori. However, the Apriori algorithm is relatively slow, and new algorithms with improved processes we introduced over time. All these algorithms are open source, and various versions of them can be found online. This paper used the Frequent Pattern Growth (FP-Growth) [27] algorithm in mining the association rules. This algorithm is considerably faster than the Apriori algorithm, which can have a big effect on the overall data generation process time because of the study's data size.

This paper proposes a two-phase hybrid synthetic order data generator based on the method introduced by Ansari et al. [5]. The data generation process is based on the mined association rules. By extracting the association rules from a real order dataset and using a unique algorithm, a new set of data is generated. Using our method will keep the association rules of the newly

generated data close to the original dataset. This means that the correlations between the SKUs in the generated order data will be preserved.

Because of the complexity of the real order data and the time-consuming process of extracting all association rules, the rule mining process is limited to significant SKUs. This means that we only mine for the items and itemsets that their support value is higher than a predefined threshold value. Mining only for the itemsets' association rules that their support is higher than the predefined threshold value will result in some itemsets' association rules to be ignored. However, it should be mentioned that these lost itemsets' association rules are insignificant (rules that the probability of their appearance is lower than the predefined mining threshold value). This means that even preserving them in the data generation process will be unnecessary since they are too small to be considered or affect any research and managerial decision-making processes.

Our method combines an upgraded Synthetic Generator (introduced by Ansari et al. [5]) and a Random Generator explained in Algorithm 3. Our method in generating data utilizes the Pipe technique. In this technique, information is passed through one process to the other. This means that the output of one process will act as the input to the next process. Our method consists of two steps. In the first step or Phase-I, we will generate the lines and correlations that belong to category A SKUs. As mentioned, category A SKUs are a small portion of SKUs that make up a large portion of the lines. This means that although the total number of extracted SKUs that the generated order data is based on is a small portion, they are responsible for the most significant part of lines and correlations in the order dataset. The next step or Phase-II will incorporate the more significant portion of SKUs that are responsible for a smaller part of the order dataset lines. Phase-I and Phase-II are called Core and Detailing, respectively

- The Core generation is the process of extracting the association rules with a predefined threshold and generating a set of order data that preserves the relative frequency of all significant SKUs and their correlation structures. Because of the predefined threshold value, the process of Association Rule Mining will result in insignificant SKUs to be ignored. Although only affect the insignificant SKUs, this data loss does change the final generated data's overall structure. The final result of Phase-I will have the correlation aspect of the original dataset; however, it will have fewer SKUs and a smaller average order size.

- The Detailing phase will do the final touches in generating the order data. The data loss that will occur in Phase-I will be compensated in this phase. This means adding the non-significant SKUS to the orders. This will bring the average item per order and the total number of SKUs to the goal level. The final result will be a set of generated order data

in which the average, standard deviation, and correlation structure of the real dataset are maintained.

Figure 4.1 illustrates the overall structure of the generated data. In this figure, the dots represent the SKUs and the lines illustrate the correlation structure between the SKUs. The closer the dots to the circle's center, the higher the probability of the SKU appearance. As the SKUs get farther away from the center, their probability of appearance decreases. As for the correlation structure, if two SKUs are connected to each other with a line, this means that those two SKUs are correlated. Also, the figure is designed in a way that the area represents the total number of items in the order dataset.

Figure 4.1 represents the Hybrid Synthetic Order Data Generator processes. Figure 4.1(a) illustrates the core generation process. As can be seen, a few numbers of SKUs that are generated in this phase are responsible for a large portion of lines. Also, as explained before, the correlation structure (the lines) between the SKUs are preserved in the generated data. Figure 4.1(b) demonstrates the generated data in the Detailing step. Although the number of SKUs in this phase is more than the Core, they only make up a small portion of the generated lines. Also, as can be seen, there is no correlation structure in the generated data in the Detailing step. The combination of both Core generation (as illustrated in Figure 4.1(a)) and Detailing (as presented in Figure 4.1(b)) will result in the complete Hybrid Synthetic Order Data Generator, as can be seen in Figure 4.1(c).

Figure 4.2 compares the real data to the generated data using the Random generator and the data generated by the Hybrid Synthetic Order Data Generator. As can be seen in the figure, the Hybrid Synthetic Order Data Generator (Figure 4.2(b)) method mimics the correlation structure of the real data (Figure 4.2(c)) for the majority of lines and adds the Detailing features from the ABC generation method (Figure 4.2(a)). The Hybrid Synthetic Order Data Generator keeps the majority of correlations between SKUs and the order structure from the original dataset intact. However, the Random generator (Figure 4.2(a)), although keeps the SKU frequencies and the order size structure the same, it has no correlation structure between the SKUs.

Algorithms 4 and 5 represent the Core and Detailing phases processes, using the notation from Table 4.2.

Algorithm 4 will be responsible for creating n orders, as required. However, the number of items-per-order is not yet correct, and some of the generated orders will have empty lines. With Algorithm 5, Phase-I issues, including the average and standard deviation of item-per-order value, will be resolved. The final outcome will resemble the real dataset in all the aspects mentioned in Section 4.1. As will be discussed later in Sections 4.3.3 and 4.3.4, with the correct

M	Matrix of SKUs with size $m \times 1$
I	Number of items in real dataset
F	Matrix of SKU frequency $\forall SKU \in M$
a	Average number of items per order
d	Standard deviation of number of items per order
n	Desired number of orders
s_i	Relative support value of i, where $i \subset M$
S	Association rule matrix with relative support value
O	Order matrix
PDF	Probability distribution function

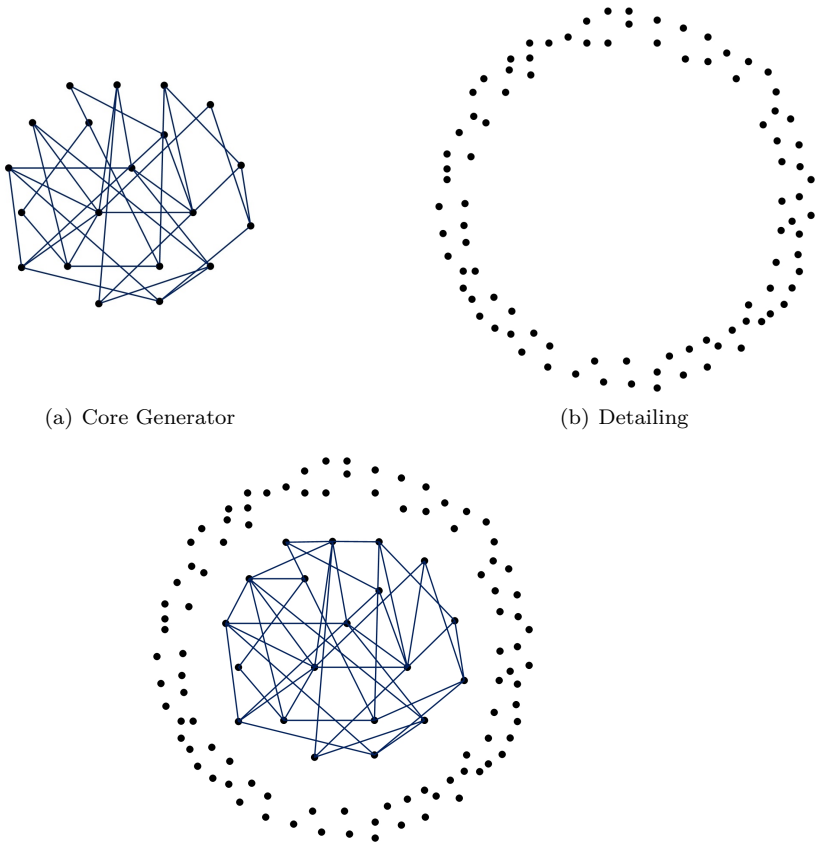
Table 4.2: Table of notation.

Algorithm 4: Simplified Core generation process.

```
1  Create Goal.S as n × S
2  while All Goal.S rows > 0 do
3      Select the row i from Goal.S where s_{i} < s_{j} (∀ j ∈ S)
4      if Times ThresholdValue adjusted for j < 5 then
5          try:
6              Add i to O
7              Adjust Goal.S value for that itemset by subtracting 1
8              Adjust Goal.S for all j where j ⊂ i
9          catch Subset Value Less Than Adjustment:
10             Add ThresholdValue to j
11             Return to line 4
12     else
13         Next i
```

Algorithm 5: Simplified Detailing process.

```
1  Create Transition.PDF
2  for Each SKU i Not In S do
3      for F_i do
4          Select TargetOrderSize by Random Sampling of Transition.PDF
5          Select Order o from O_{TargetOrderSize} (∀ o ∈ O_{TargetOrderSize})
6          Add SKU i to o
7          Update Transition.PDF
```



(a) Core Generator

(b) Detailing

(c) Core Generator and Detailing

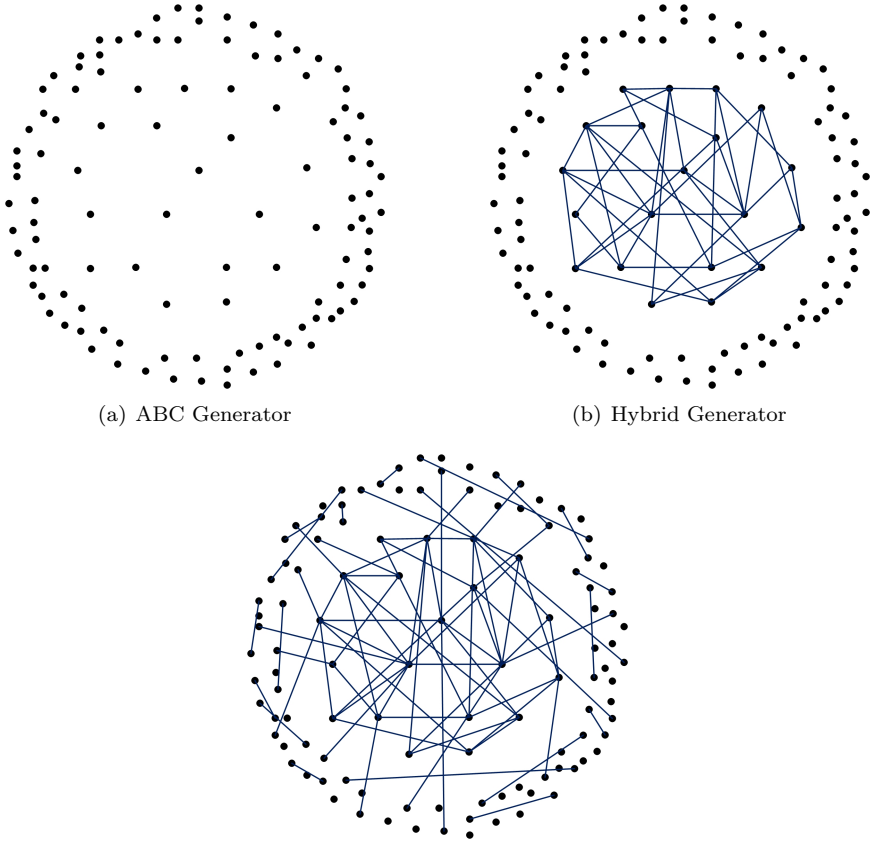
Figure 4.1: Two-Phase Hybrid Synthetic Order Data Generator.

adjustments to the process, the Core phase can generate any number of orders and SKUs based on an order dataset with a limited number of orders and SKUs.

As mentioned before, the core step, which preserves the correlation structure, is limited to only significant SKUs. The level by which significance is evaluated, the threshold value, can be adjusted to encompass more and even all SKUs. However, doing so requires extensive rule extraction, which is a time-consuming and computationally complicated process. Also, the inevitable error might be more significant than some association rules with low support value, which makes the overall results unreliable. On the other hand, keeping all rules and correlations will not affect the Market Basket Analysis, Correlated Storage Location Assignment, and other related research because of the low probability and nominal correlation values [3].

4.3 Results

For testing our algorithm, we used the Fruithut order data publicly available by Mitchell [43]. The data is a set of sales tickets consisting of the order number, item name, quantity, and category. Having the item names and the category of products they belong to is a crucial set of



(a) ABC Generator

(b) Hybrid Generator

(c) Real Data Structure

Figure 4.2: Correlation structure of various data generators.

information that plays a significant role in generating data with an arbitrary number of SKUs (as will be discussed in detail in Section 4.3.4). Table 4.3 represents our preliminary analysis of the Fruithut order data structure.

In this section, we will test our proposed algorithm using the order data mentioned above. In Section 4.3.1, the processes which are taken to recreate the Core of the generated data will be discussed and analyzed in more detail. In Section 4.3.2, the Detailing step processes will be performed on the generated data, and the results will be compared to the original dataset. In Section 4.3.3, the proposed methods' performance will be tested in generating an arbitrary number of orders. Furthermore, in Section 4.3.4, we will discuss the processes taken for generating an arbitrary number of orders and an arbitrary number of SKUs. In Section 4.3.5, the method will be tested with a widely available and in-use set of order data for benchmarking purposes.

Number of Orders	181,971
Number of Items	659,222
Average Line per Order	3.62
Number of SKUs	1,267
Number of SKU Categories	44

Table 4.3: Fruithut order data preliminary analysis.

Itemset	Support	Relative Support
[13]	7,475	0.0410
[158]	1,382	0.0076
[1,27]	2,244	0.0123
[13,75]	1,054	0.0058
[25,27,75]	973	0.0053

Table 4.4: Sample of 281 extracted rules from the Fruithut data.

4.3.1 Phase-I

The first step in recreating the Core is to extract the association rules from the original dataset. The most crucial factor in extracting rules is the Rule Threshold. This value is the line between significant and insignificant rules. Any rule that has a support value more than the predefined rule threshold will be considered as significant. This also means that if a rule's support value does not reach the predefined threshold, that association rule will be considered insignificant. Although having a lower rule threshold will result in a higher number of correlation structures between the SKUs being replicated, this process is time-consuming, and considering the error of replicating association rules, smaller support values might result in a dataset with inaccurate association rule supports. The correlations between SKUs are essential and worthy of analysis in various research fields, as long as they are significant and the items in the rules are ordered frequently.

Considering these arguments for assigning threshold value, we chose the rule threshold as 0.005 or 0.5%. This means that we consider an SKU a Core SKU or an itemset's association rule a significant rule, as long as their relative support value is higher than 0.005 or, in other words, the SKU or the itemset appears in at least 910, or 0.5% of the orders. Using the mentioned rule threshold, we extracted the real (original) order dataset's association rules. This resulted in 281 rules, from which 132 of them are itemsets of size two or higher. Table 4.4 represents a few sample lines from the extracted association rules.

Using the mentioned association rules, we created an order dataset of size 181,971, the exact number of orders from the original dataset, to compare the generated Core's results with

Rule	Original Support	Generated Support	Difference	% Difference
[1]	0.041814	0.041809	0.000005	0.01%
[14]	0.042765	0.042760	0.000005	0.01%
[51]	0.009600	0.009595	0.000005	0.06%
[54]	0.005638	0.006199	0.000561	9.94%
[69]	0.014590	0.016047	0.001456	9.98%
[126]	0.019679	0.019673	0.000005	0.03%
[127]	0.010864	0.011947	0.001083	9.96%
[12, 27]	0.005407	0.003995	0.001412	26.12%
[16, 27]	0.008501	0.008496	0.000005	0.06%
[23, 43]	0.006166	0.006693	0.000528	8.56%
[23, 75]	0.008952	0.008485	0.000467	5.22%
[27, 316]	0.005215	0.005556	0.000341	6.53%
[66, 75]	0.007023	0.007023	0.000000	0.00%
[75, 108]	0.009315	0.009309	0.000005	0.06%
[50, 114]	0.006847	0.006545	0.000302	4.41%
[27, 31, 75]	0.005655	0.003632	0.002022	35.76%
[25, 27, 75]	0.005347	0.006265	0.000918	17.16%

Table 4.5: Comparing rule supports in the original dataset versus generated dataset for few randomly selected rules.

the original data. Table 4.5 represents a few randomly selected rules from the original dataset's association rules and their corresponding support value from the generated dataset. Analyzing all extracted rules and comparing them to the original dataset's rules shows that, on average, there is a 4.62% difference between the support value of the two sets of extracted rules. As mentioned before in Section 4.2, this deviation is unavoidable and necessary. On the other hand, however, because of lost SKUs in the process of extracting association rules, the average number of lines per order in the generated data is lower than the original. Our original order dataset has an average of 3.62 lines per order, while the generated set of data has only 3.10. The standard deviation of the original dataset is 3.13, while for the Phase-I generated data, the value is 1.71. The average number of lines in the generated data has a 17% error from the original dataset. This error is 55% in the standard deviation. Although we expect some error and variation in average and standard deviation of the number of lines per order between the original data and the generated data, these numbers are considerably high.

So far, the generated data has a small percentage of difference in the support values of the association rules and a relatively high percentage of error in average and standard deviation of lines per order to the original dataset. In Phase-II, as will be discussed in Section 4.3.2, we will reduce the error of the average and the standard deviation of the number of lines per order.

	Original	Phase-I Generated	Phase-II Generated
Average	3.62	3.10	3.68
Standard Deviation	3.13	1.71	3.62
Max Item per Line	36	13	36
ABC Structure	10/90	10/90	10/90

Table 4.6: Comparing the results of generated data in Phase-I and Phase-II to the original dataset.

4.3.2 Phase-II

In this section, the results of Phase-I will be put into the Detailing process in which the items that did not make the cut to be considered as significant in the extracted association rules, will be appended to the generated data. In this process, not only will we add the missing SKUs from Phase-I, but we will also be doing so while we are fine-tuning the line-per-order probability distribution function. To do so, we start with the SKUs that were not frequent enough to be mined for Phase-I. In the next step, a Transition probability distribution function (PDF) of item-per-order will be created by subtracting the current PDF from the goal PDF for each order size. SKUs are added to a random order of size X, in which the X is selected by random sampling of the Transition PDF. By adding an item to an order of size X, the order will change the size to $X+1$, which changes the current PDF. Because of the changed current PDF, the Transition PDF will be updated each time an SKU is added to the generated dataset. This process continues until all missing SKUs are added to the generated order.

Although in this method, the goal is to make the current PDF as similar as possible to the goal PDF, because of the errors associated with Phase-I processes, there will be no guarantee that both PDFs will be the same. Table 4.6 illustrates the comparison results of the original dataset versus the generated datasets in Phase-I and Phase-II.

As can be seen in Table 4.6, Phase-II of the proposed Hybrid Synthetic Order Data Generator, as expected, will add the final tuning of the features to the correlated Core, generated in Phase-I. It should be mentioned that Phase-II will not add any significant rule to the generated dataset. The reason is that the SKUs added in Phase-II are all considered insignificant by the predefined threshold standards. However, for a rule to be considered significant, all of its subsets should be significant as well. If an SKU has a probability of appearance lower than the predefined threshold, neither the SKU nor any itemset it belongs to, cannot have a support value that surpasses the threshold value.

Phase-I and Phase-II processes' cumulative actions will result in a dataset that mimics all aspects of characteristics from the real order dataset. The final generated data not only satisfies

the defacto standards in order data generation (such as distribution, average, and standard deviation of line-per-order), it also keeps the essential part of the correlation structures between the SKUs. Although losing some itemsets and SKU correlations are unavoidable, the research related to association rule mining and market basket analysis is mainly focused on the critical and influential SKUs and their links with other influential SKUs.

4.3.3 Modifying Number of Orders

The data generated in Sections 4.3.1 and 4.3.2 are intentionally generated with the same number of orders as the original dataset. This was to illustrate and perform a side-by-side comparison of the original data and the generated data results using Hybrid Synthetic Order Data Generator in Phase-I and Phase-II, as can be seen in Table 4.6. However, the proposed method has no limitation on the number of orders generated without losing the data characteristics. Table 4.7 represents the comparison of the extracted rules from the original dataset versus the extracted rules from a generated dataset with twice the number of orders (363,942) and a generated dataset with half the number of orders (90,986) compared to the original dataset order count (181,971). As can be seen, there is no significant difference.

Having a set of data with half the number of orders of the original is simply like random sampling half of the generated data if the generated data follows the same number of orders as the original. This means that the relative support of the rules will remain within a small range of error. The same happens when generating a dataset with double the number of orders compared to the original. In this case, the process is like generating two sets of order data with the same size as the original and appending one to the other.

Table 4.8 illustrates the structural differences between generated datasets of various sizes to the original dataset. The same logic that was explained before can be applied here as well. Generating a lower number of orders than the number of orders in the original dataset is merely a sampling, and generating a higher number of orders, is appending smaller ones together. This means that the average, the standard deviation, the maximum item-per-order, and the ABC structure, will have insignificant differences compared to the original dataset.

4.3.4 Modifying Number of SKUs

So far, all generated data are using the original data as the source for the number of SKUs. Our original order data, Fruithut [43], consists of 181,971 orders and 1,267 SKUs. The final results of the generated data, although with an arbitrary order size, were also based on 1,267 SKUs. However, in this section, we propose a categorization method that enables us to create

Rule \ Data Size	1/2× Original	Original	2× Original
[24]	0.03961	0.03962	0.03961
[39]	0.01073	0.01074	0.01127
[67]	0.01213	0.01104	0.01159
[68]	0.02427	0.02427	0.02670
[69]	0.01458	0.01459	0.01532
[106]	0.03756	0.03757	0.03756
[118]	0.00764	0.00632	0.00698
[150]	0.00665	0.00550	0.00665
[226]	0.00907	0.00750	0.00824
[14, 50]	0.00399	0.00666	0.00400
[22, 27]	0.00920	0.00920	0.00920
[25, 50]	0.00399	0.00825	0.00400
[27, 59]	0.01266	0.01126	0.01258
[27, 263]	0.00399	0.01893	0.00400
[43, 50]	0.00399	0.00729	0.00400
[75, 212]	0.00399	0.00828	0.00400
[25, 27, 75]	0.00745	0.00535	0.00664

Table 4.7: Comparing the rule supports in the original dataset versus the gnerated data with half the number of orders and the generated data with double the number of orders.

Data Size	1/2× Original	Original	2× Original
Average	3.67	3.62	3.67
Standard Deviation	3.53	3.13	3.61
Max Item per Line	36	36	36
ABC Structure	10/90	10/90	10/90

Table 4.8: Comparing the results of the generated data in cases of half and double the number of orders compared to the original dataset.

an arbitrary number of SKUs. In this method, we first abstract the SKU data to a higher level, which means that we only keep the category that the product belongs to instead of each product name and ID. This modification means that, for example, Granny Smith Apples, Gala Apples, Envy Apples, Honeycrisp Apples, and Fuji Apples, which all are various types of the same fruit, will be replaced by their category name, Apple. The rules will then be extracted based on the modified order data. Later on, Phase-I and Phase-II will generate a new set of order data in which the items, instead of being a full product name (e.g., Fuji Apples), will only be a category name (e.g., Apples).

Furthermore, after generating the target number of orders, category names will be replaced by the products in that category based on the probability distribution of that category's items. It is in this step that instead of the original number of SKUs in that category, a modified number of SKUs can replace the category in the generated order data. This means that instead of the original probability distribution for each category of products, a new probability distribution will be used to implement the new added SKUs. Figure 4.3 is the flow chart Detailing each step in generating an arbitrary number of SKUs.

In generating an arbitrary number of SKUs, few things need to be addressed:

- This rule abstraction is not without the loss of some data. Although the generated data will have a correlation structure, the correlation is only between the categories and not the SKUs. This change in correlation structure means that, for example, a correlation between Romaine Lettuce and Ranch Dressing will be replaced by a correlation between their categories, Lettuce and Dressing. These lost correlations are not retrievable because the replacement items are not guaranteed to be the same items from the original set of data (as they are randomly selected from a set of SKUs using a probability distribution function). On the other hand, the loss of data, in general, will not significantly affect the purpose for which we try to keep the correlations intact. Items of the same category are stored together, whether they are correlated or not. For example, in a grocery store, all apples are kept in the same vicinity. So, although we will not be sure which exact SKUs are correlated, we know which categories of SKUs are correlated, and will be beneficial to keep them close to each other.

- This method assumes that the added SKUs that were not in the original dataset will belong to a category already in the stock. This means that the newly added product can be an Apple, Spinach, or Lettuce, but if we never had a product in the Tropical Fruit category, we cannot add now. The reason is that we need to have some information about how a

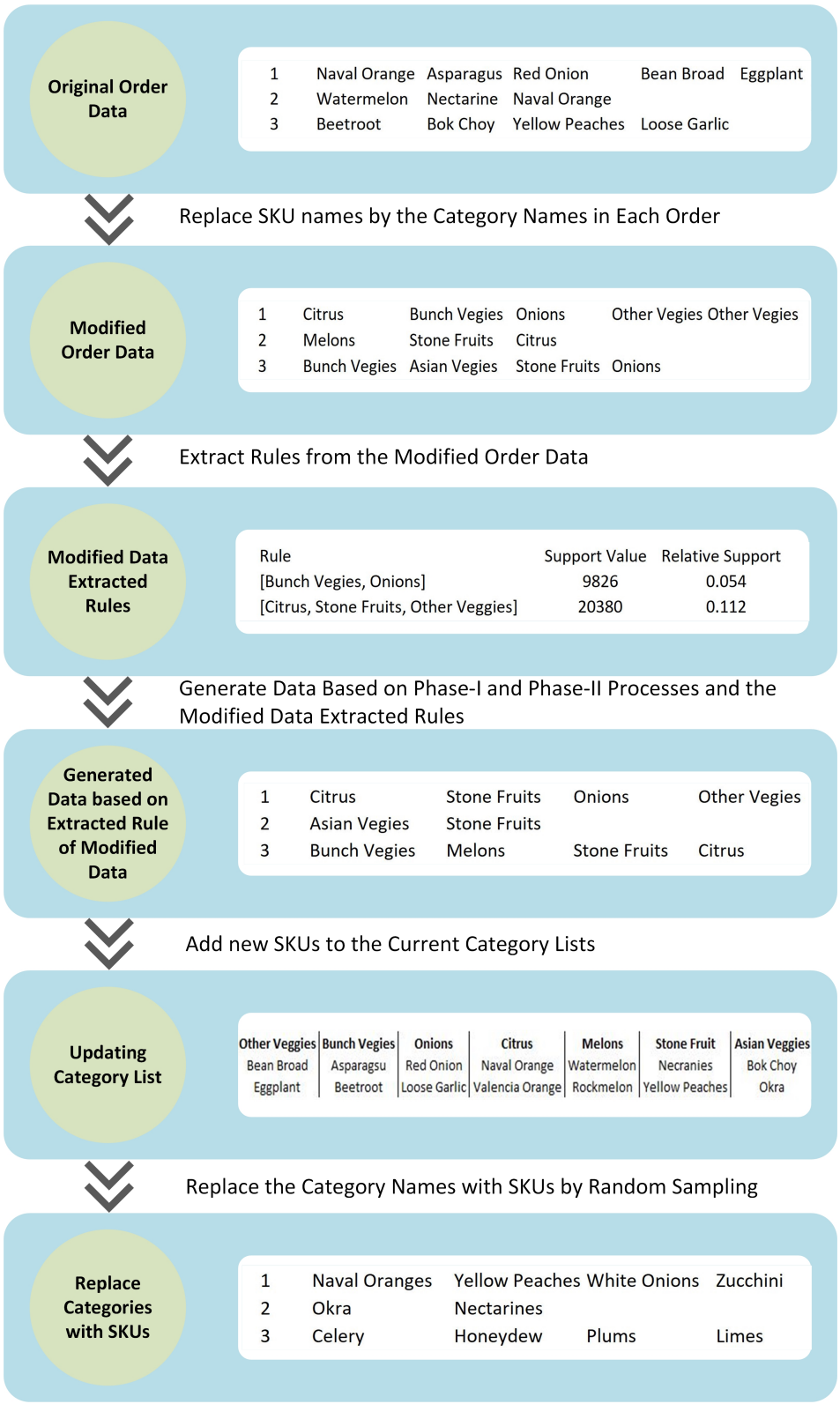


Figure 4.3: Flowchart of generating order data with arbitrary number of SKUs.

Itemset	Support	Relative Support
[1]	32,910	0.1808
[15]	33,883	0.1846
[1,18]	11,032	0.0606
[4,15]	12,683	0.0697
[18,19]	11,727	0.0644

Table 4.9: Sample extracted rules from category representation of the original data.

category of products is correlated to other categories, and for a category of products that were not in the original dataset, no such information exists.

- Other than the correlations of each item being lost in the rule extraction and data generation steps, SKUs from the same category in the same order will be ignored. Although the chances of two products from the same category being in the same order are low, it is not impossible. In almost 10% of orders, two or more of the products in the same order, are from the same category. An example can be seen in Figure 4.3. SKUs Bean Broad and Eggplant both belong to the same order and the same category, Other Veggies. When SKUs are abstracted to the category names, having multiple replications of the same category will not affect the association rule value. This means that it is up to Phase-II to make up for the lost items by assigning a probability function to each category type, which randomly decides how many SKUs should replace that category in the reverting (from category to SKU) step.

As mentioned before, in Table 4.3, our input data consists of 1,267 SKUs from 43 different categories. Table 4.9 illustrates a few lines of the extracted rules from the modified order dataset. Each item number represents a category of products instead of a single SKU.

Comparisons between the extracted rules from the original dataset and generated order data with a modified number of SKUs will be pointless, considering that the items are randomly assigned to their places. However, suppose we revert the generated data to its categories and compare the extracted association rules with that of the original order dataset counterpart. In that case, we will get support values within a reasonable margin of error from the original (such as one seen in Table 4.5).

As mentioned, the abstraction and the loss of data, although not significantly, still make the generated order dataset differ from the original dataset. Table 4.10 represents the original dataset's comparison results versus the generated dataset from Section 4.3.2 and the generated dataset with an arbitrary number of SKUs.

Data Type	Original Dataset	Generated from SKU Rule Extraction	Generated from Categoty Rule Extraction
Average	3.62	3.68	3.32
Standard Deviation	3.13	3.62	2.81
Max Item per Line	36	36	35
ABC Structure	10/90	10/90	10/90

Table 4.10: Comparing the results of the original dataset, generated data from extracted SKU rules, and generated data from extracted category rules.

4.3.5 Benchmark

The real data we have used so far, Fruithut order data, is a publicly available dataset provided by Mitchell [43]. However, in the fields related to market basket analysis and association rule mining's test and development, the datasets in use are usually smaller sets of order data with fewer SKUs. Our research indicated that in many cases, as a sample of order data, baskets1ntrans.xlsx [59] is being used. Also, multiple institutions and colleges use this data as the example dataset in teaching market basket analysis and association rule mining. Our goal in this section is to provide another set of data that we can use to test and analyze the performance of the Hybrid Synthetic Order Data Generator and set a widely used, relatively small, and readily available set of data as a benchmark. This benchmark can be later used by researchers to compare various order data generator methods and algorithms. Having a standard set of data will pave the way in comparing methods, as the results can be compared with no transformation or scaling. Table 4.11 represents the preliminary analysis of baskets1ntrans.xlsx (will be referred to as Basket1 in this paper) order dataset.

Number of Orders	939
Number of Items	2,800
Average Line per Order	2.98
Number of SKUs	11

Table 4.11: Basket1 order data preliminary analysis.

We used the extracted rule from the Basket1 dataset and generated 93,900 orders (100 times the original dataset). Table 4.12 compares the extracted rule from the original dataset to the rules from the generated dataset. Table 4.13 represents and compares the metadata of the generated data in Phase-I and Phase-II to the original dataset.

As can be seen in Table 4.13, unlike the Fruithut data, Phase-I and Phase-II data in the generated dataset based on Basket1 follow precisely the same data structure. This change in

Rule	Original Support	Generated Support	Difference
[1]	0.1948	0.1948	0.0000
[5]	0.3344	0.3099	0.0245
[11]	0.2066	0.1885	0.0181
[1, 4]	0.0479	0.0479	0.0000
[8, 10]	0.0479	0.0437	0.0043
[1, 2, 10]	0.0138	0.0117	0.0021
[2, 7, 11]	0.0202	0.0170	0.0032
[2, 3, 6, 10]	0.0128	0.0117	0.0011
[4, 5, 7, 9]	0.0106	0.0106	0.0000
[3, 4, 6, 8, 10]	0.0106	0.0117	0.0011

Table 4.12: Comparing rule supports in the original dataset versus the generated dataset for a few randomly selected rules.

	Original	Phase-I Generated	Phase-II Generated
Average	2.98	3.10	3.10
Standard Deviation	1.53	1.51	1.51
Max Item per Line	8	11	11
ABC Structure	50/50	50/50	50/50

Table 4.13: Comparing the results of the generated data in Phase-I and Phase-II to the original dataset.

characteristics is because of the following differences in SKU adding processes in the generation of the two datasets.

1. The ABC structure of the data is following a 50/50 pattern, which means all SKUs have an almost equal frequency of appearance in the dataset.

2. The extracted association rules included all SKUs, which means no SKU is missing in the generated dataset that needs to be added by the Detailing step.

Because of the highly correlated SKUs in the Basket1 dataset, all SKUs have been considered as significant and were added to the generated data via the Core generation process. This resulted in the Detailing step to play no part in generating this dataset.

4.4 Conclusion

Realistic sample data for testing, comparing, and benchmarking new methods and applications has always been needed in the supply chain and warehousing fields. For a long time, randomly generated data has been in use by researchers to compare their proposed methods to the previously available approaches. Although these random data generators create data that look and even in some cases are similar to the actual data, they lack the behavioral aspect of costumers

in ordering products. In other words, the random generators will not be able to replicate the correlation structure of the order data. In this paper, we introduced the Hybrid Synthetic Order Data Generator. This data generator incorporates a two-phase method in generating an order dataset with an arbitrary number of orders and SKUs while keeping the significant correlations between the SKUs intact. This approach has been tested and resulted in remarkable similarity in correlation structure and other characteristics between the generated dataset and the real dataset. Later, the ability of the proposed Hybrid Synthetic Order Data Generator in generating an arbitrary number of orders and an arbitrary number of SKUs while keeping the Core correlation structure of SKUs has been tested.

This paper is, by the author's knowledge, the first study in generating realistic sample order data for warehouse-related research and operations. This makes this research area relatively untouched by academia. Further research in this field is required to test, validate, and improve the proficiency and performance of the proposed method. These research can focus on keeping a more substantial portion of the correlation and association rule mining, improving the data generator's overall performance, and enhancing the fine-tuning and Detailing in Phase-II. On the other hand, future research can expand this order data generator and develop methods and algorithms that on top of the current features, can control the generated data's correlation structure.

References

[1] Jason W Anderson, KE Kennedy, Linh B Ngo, Andre Luckow, and Amy W Apon. Synthetic data generation for the internet of things. In 2014 IEEE International Conference on Big Data (Big Data), pages 171–176. IEEE, 2014.

[2] Mohammadnaser Ansari and Jeffrey S Smith. Warehouse operations data structure (wods): A data structure developed for warehouse operations modeling. Computers & Industrial Engineering, 112:11–19, 2017.

[3] Mohammadnaser Ansari and Jeffrey S Smith. Gravity clustering: A correlated storage location assignment problem approach. In 2020 Winter Simulation Conference (WSC). IEEE, 2020.

[4] Mohammadnaser Ansari, Ashkan Negahban, Fadel M Megahed, and Jeffrey S Smith. Historia: A new tool for simulation input analysis. In Proceedings of the Winter Simulation Conference 2014, pages 2702–2713. IEEE, 2014.

[5] Mohammadnaser Ansari, Behnam Rasoolian, and Jeffrey S Smith. Synthetic order data generator for picking data. In Proceedings of 15th IMHRC, Savannah, Georgia, USA, 2018.

[6] F Bindi, R Manzini, A Pareschi, and A Regattieri. Similarity coefficients and clustering techniques for the correlated assignment problem in warehousing systems. In Proceedings of 19th International Conference on Production Research, 2007.

[7] Filippo Bindi, Riccardo Manzini, Arrigo Pareschi, and Alberto Regattieri. Similarity-based storage allocation rules in an order picking system: an application to the food service industry. International Journal of Logistics: Research and Applications, 12(4):233–247, 2009.

[8] Franco Caron, Gino Marchet, and Alessandro Perego. Routing policies and COI-based storage policies in picker-to-part systems. International Journal of Production Research, 36 (3):713–732, 1998.

[9] Franco Caron, Gino Marchet, and Alessandro Perego. Optimal layout in low-level picker-to-part systems. International Journal of Production Research, 38(1):101–117, 2000.

[10] Felix TS Chan and Hing Kai Chan. Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of class-based storage. Expert Systems with Applications, 38(3):2686–2700, 2011.

[11] Gilles Cormier and Eldon A Gunn. A review of warehouse models. European journal of operational research, 58(1):3–13, 1992.

[12] John Joseph Coyle, Edward J Bardi, C John Langley, et al. The management of business logistics, volume 6. West publishing company St Paul, MN, 1996.

[13] René De Koster and Edo Van der Poort. Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions. IIE transactions, 30(5):469–480, 1998.

[14] René De Koster, Tho Le-Duc, and Kees Jan Roodbergen. Design and control of warehouse order picking: A literature review. European Journal of Operational Research, 182(2): 481–501, 2007.

[15] Domo. Data never sleeps 5.0. https://www.domo.com/learn/data-never-sleeps-5?aid=ogsm072517_1&sf100871281=1, 2018 (accessed May 10$^{\text{th}}$, 2020).

[16] J Druckerman, D Silverman, and K Viaropulos. IBM optimization subroutine library, guide and reference, release 2. Document Number SC23-0519-2, IBM, Kingston, NY, 1991.

[17] Juan J Durillo and Antonio J Nebro. jmetal: A java framework for multi-objective optimization. Advances in Engineering Software, 42(10):760–771, 2011.

[18] Richard L Francis. On some problems of rectangular warehouse design and layout. Journal of Industrial Engineering, 18(10):595, 1967.

[19] Michael Frank, Meikel Poess, and Tilmann Rabl. Efficient update data generation for dbms benchmarks. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, pages 169–180. ACM, 2012.

[20] EH Frazelle. Stock location assignment and order batching productivity. Georgia Institute of Technology, Atlanta, Georgia, 1990.

[21] AJRM Gademann, Jeroen P Van Den Berg, and Hassan H Van Der Hoff. An order batching algorithm for wave picking in a parallel-aisle warehouse. IIE transactions, 33(5):385–398, 2001.

[22] Noud Gademann and Steef Velde. Order batching to minimize total travel time in a parallel-aisle warehouse. IIE transactions, 37(1):63–75, 2005.

[23] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. Research on warehouse operation: A comprehensive review. European journal of operational research, 177(1):1–21, 2007.

[24] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. Research on warehouse design and performance evaluation: A comprehensive review. European Journal of Operational Research, 203(3):539–549, 2010.

[25] Kevin R Gue and Russell D Meller. Aisle configurations for unit-load warehouses. IIE Transactions, 41(3):171–182, 2009.

[26] Kevin R Gue, Goran Ivanović, and Russell D Meller. A unit-load warehouse with multiple pickup and deposit points and non-traditional aisles. Transportation Research Part E: Logistics and Transportation Review, 48(4):795–806, 2012.

[27] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. ACM sigmod record, 29(2):1–12, 2000.

[28] Sunderesh S Heragu*, L Du, Ronald J Mantel, and Peter C Schuur. Mathematical model for warehouse design and product allocation. International Journal of Production Research, 43(2):327–338, 2005.

[29] James L Heskett. Cube-per-order index-a key to warehouse stock location. Transportation and distribution Management, 3(1):27–31, 1963.

[30] Tin Kam Ho and Henry S Baird. Evaluation of ocr accuracy using synthetic data. In Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval, pages 413–422. Citeseer, 1995.

[31] Ying-Chin Ho, Teng-Sheng Su, and Zhi-Bin Shi. Order-batching methods for an order-picking warehouse with two cross aisles. Computers & Industrial Engineering, 55(2):321–347, 2008.

[32] TV Hromadka. A rainfall-runoff probabilistic simulation program: 1. synthetic data generation. Environmental software, 11(4):235–242, 1996.

[33] H Hwang, YH Oh, and YK Lee. An evaluation of routing policies for order-picking operations in low-level picker-to-part system. International Journal of Production Research, 42 (18):3873–3889, 2004.

[34] Instacart. The instacart online grocery shopping dataset. https://www.instacart.com/datasets/grocery-shopping-2017, 2017 (accessed March 2nd, 2020).

[35] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. arXiv preprint arXiv:1406.2227, 2014.

[36] E Kearney and A Kearney. Excellence in logistics 2004. European Logistics Association, Brussels, 2004.

[37] Byung Soo Kim and Jeffrey S Smith. Slotting methodology using correlated improvement for a zone-based carton picking distribution system. Computers & Industrial Engineering, 62(1):286–295, 2012.

[38] Kap Hwan Kim and Ki Young Kim. An optimal routing algorithm for a transfer crane in port container terminals. Transportation Science, 33(1):17–33, 1999.

[39] Diwakar Krishnamurthy, Jerome A Rolia, and Shikharesh Majumdar. A synthetic workload generation technique for stress testing session-based systems. IEEE Transactions on Software Engineering, 32(11):868–882, 2006.

[40] Tho Le-Duc and Rene MBM de Koster. Travel time estimation and order batching in a 2-block warehouse. European Journal of Operational Research, 176(1):374–388, 2007.

[41] Emilie Lundin, Håkan Kvarnström, and Erland Jonsson. A synthetic fraud data generation methodology. In International Conference on Information and Communications Security, pages 265–277. Springer, 2002.

[42] Russell D Meller and Kevin R Gue. The application of new aisle designs for unit-load warehouses. In NSF Engineering Research and Innovation Conference, pages 1–8. Citeseer, 2009.

[43] D Mitchell. Fruithut order data. https://data.world/digitalbias/operationsanalysis/workspace/file?filename=fruithut_data_ordered_csv_file_1_1.csv, 2016 (accessed May 20th, 2020).

[44] Ashkan Negahban and Jeffrey S Smith. Simulation for manufacturing system design and operation: Literature review and analysis. Journal of Manufacturing Systems, 33(2):241–261, 2014.

[45] Ashkan Negahban, Mohammadnaser Ansari, and Jeffrey S Smith. Add-more: automated dynamic display of measures of risk and error. In 2016 Winter Simulation Conference (WSC), pages 977–988. IEEE, 2016.

[46] Ömer Öztürkoğlu, Kevin R Gue, and Russell D Meller. Optimal unit-load warehouse designs for single-command operations. IIE Transactions, 44(6):459–475, 2012.

[47] Charles G Petersen. An evaluation of order picking routeing policies. International Journal of Operations & Production Management, 17(11):1098–1111, 1997.

[48] Charles G Petersen and Gerald Aase. A comparison of picking, storage, and routing policies in manual order picking. International Journal of Production Economics, 92(1):11–19, 2004.

[49] Charles G Petersen and Roger W Schmenner. An evaluation of routing and volume-based storage policies in an order picking operation. Decision Sciences, 30(2):481–501, 1999.

[50] Charles G Petersen, Charles Siu, and Daniel R Heiser. Improving order picking performance utilizing slotting and golden zone storage. International Journal of Operations & Production Management, 25(10):997–1012, 2005.

[51] Budong Qian, Reinder De Jong, Jingyi Yang, Hong Wang, and Sam Gameda. Comparing simulated crop yields with observed and synthetic weather data. Agricultural and forest meteorology, 151(12):1781–1791, 2011.

[52] H Donald Ratliff and Arnon S Rosenthal. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. Operations Research, 31(3):507–521, 1983.

[53] Kees Jan Roodbergen and RenÉde Koster. Routing methods for warehouses with multiple cross aisles. International Journal of Production Research, 39(9):1865–1883, 2001.

[54] Kees Jan Roodbergen and RenÉde Koster. Routing methods for warehouses with multiple cross aisles. International Journal of Production Research, 39(9):1865–1883, 2001.

[55] Kees Jan Roodbergen and Iris FA Vis. A model for warehouse layout. IIE transactions, 38 (10):799–811, 2006.

[56] MB Rosenwein. A comparison of heuristics for the problem of batching orders for warehouse selection. International Journal of Production Research, 34(3):657–664, 1996.

[57] Bart Rouwenhorst, B Reuter, V Stockrahm, Geert-Jan van Houtum, RJ Mantel, and Willem HM Zijm. Warehouse design and control: Framework and literature review. European journal of operational research, 122(3):515–533, 2000.

[58] Robert A Ruben and F Robert Jacobs. Batch construction heuristics and storage assignment strategies for walk/ride and pick systems. Management Science, 45(4):575–596, 1999.

[59] Ramesh Sharda, Dursun Delen, and Efraim Turban. Business intelligence: a managerial perspective on analytics. Prentice Hall Press, 2013.

[60] Francielly Hedler Staudt, Gülgün Alpan, Maria Di Mascolo, and Carlos M Taboada Rodriguez. Warehouse performance measurement: a literature review. International Journal of Production Research, 53(18):5524–5544, 2015.

[61] Y Tay. Data generation for application-specific benchmarking. VLDB, Challenges and Visions, 2011.

[62] Roger F Tomlinson. A geographic information system for regional planning. In GA Stewart,(ed.: Symposium on Land Evaluation, Commonwealth Scientific and Industrial Research Organization, MacMillan of Australia., Melbourne, 1968.

[63] James A Tompkins, John A White, Yavuz A Bozer, and Jose Mario Azaña Tanchoco. Facilities planning. John Wiley & Sons, 2010.

[64] TS Vaughan. The effect of warehouse cross aisles on order picking efficiency. International Journal of Production Research, 37(4):881–897, 1999.

[65] John A White and Richard L Francis. Normative models for some warehouse sizing problems. AIIE Transactions, 3(3):185–190, 1971.

[66] Mark A Whiting, Jereme Haack, and Carrie Varley. Creating realistic, scenario-based synthetic data for test and evaluation of information analytics software. In Proceedings of the 2008 Workshop on BEyond time and errors: novel evaLuation methods for Information Visualization, pages 1–9, 2008.

[67] Jian Xiao and Li Zheng. A correlated storage location assignment problem in a single-block-multi-aisles warehouse considering bom information. International Journal of Production Research, 48(5):1321–1338, 2010.

[68] Wei-Ning Yang and Barry L Nelson. Using common random numbers and control variates in multiple-comparison procedures. Operations Research, 39(4):583–591, 1991.

[69] Yan Yu, Deepak Ganesan, Lewis Girod, Deborah Estrin, and Ramesh Govindan. Synthetic data generation to support irregular sampling in sensor networks. GeoSensor Networks, 1 (4):211–234, 2003.

[70] Ren-Qian Zhang, Meng Wang, and Xing Pan. New model of the storage location assignment problem considering demand correlation pattern. Computers & Industrial Engineering, 129: 210–219, 2019.