Development and Implementation of an Inventory Location Optimization Algorithm for Improved Distribution Center Picking Rates

by

Dov Hochsztein

Bachelor of Science in Mechanical and Aerospace Engineering
Rutgers School of Engineering, Rutgers University, 2016

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING IN ADVANCED MANUFACTURING AND DESIGN
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SEPTEMBER 2018

Signature of Author..... Signature redacted

Dov Hochsztein
Department of Mechanical Engineering
August 10, 2018

Certified by ..... Signature redacted .....

David Hardt
Ralph E. and Eloise F. Cross Professor in Mechanical Engineering
Thesis Advisor

Accepted by ..... Signature redacted .....

Rohan Abeyaratne
Quentin Berg Professor of Mechanics, Mechanical Engineering
Chairman, Committee for Graduate Students

This page was intentionally left blank.

Development and Implementation of an Inventory Location Optimization Algorithm for Improved Distribution Center Picking Rates

by

Dov Hochsztein

Submitted to the Department of Mechanical Engineering on August 10, 2018 in Partial Fulfillment of the requirements for the Degree of Master of Engineering in Advanced Manufacturing and Design

ABSTRACT

A strategic roadmap was developed for Waters Corporation detailing four central themes and twelve project proposals to direct future projects and innovations at the company over the next 3-10 years. Specific attention was paid to the growing need at Waters to shift processes from manual to automatic and to convert data collection and usage techniques from analog to digital. Pilots of two projects, Warehouse RFID-based Dock Door System and Inventory Relocation, were developed in conjunction with the preparation of the roadmap in order to provide proofs of concept and technology and to serve as reference points for future projects. The Inventory Relocation Pilot Project is discussed in this thesis.

An automatic inventory relocation tool was developed to recommend product locations within Waters' Global Distribution Center (GDC). The tool was configured to allow full region reorganization, relocation of a specified number of stock keeping units, and recommendation of put-away locations for incoming shipments. The pilot was constrained to the high bay area at GDC due to the amount of available data for that region, but was designed to be easily extendable to include other regions in GDC and other Waters distribution centers.

Simulations were run that showed greater than 40% reductions in the objective function that correspond to approximately 5% less time spent picking orders by material handlers, given the amount of available supporting data.

Thesis Supervisor: David Hardt

Title: Ralph E. and Eloise F. Cross Professor in Mechanical Engineering

This page was intentionally left blank.

ACKNOWLEDGMENT

I would like to take the opportunity to extend thanks to those who helped make this possible:

To Gabriel Kelly and Lichung Pan for their constant guidance and availability at Waters Corporation in seeing this project through to completion.

To Kathleen Wright, Ryan Cashman, Kwaku Manu-Tawiah and the team at GDC for opening up their home to us and helping us with anything we needed.

To everyone at Waters Corporation for their assistance throughout this project, including Dan Welch and Jim McPherson for their support in making the collaboration between MIT and Waters possible.

To my teammates, Rushil Batra and Akshay Harlalka for their collaboration, ingenuity, and endless discussions.

To the rest of the M.Eng. crew for a year I will never forget.

To our advisor, David Hardt for keeping us on track and reminding us to be selfish.

To Jose Pacheco for welcoming us to the M.Eng. Program and leading us for the past 12 months.

To my parents and siblings for supporting me in everything I do.

To my wife, Talia, for keeping me sane all year long.

This page was intentionally left blank.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

The purpose of this project is to find opportunities for process improvement within various business verticals under the umbrella of digitalization and automation. The team was expected to find opportunities, develop and present them, and research and implement a pilot project on one or more of them.

The focus of this thesis is the implementation of an Inventory Relocation Tool capable of recommending new storage layouts for a distribution center based on product data.

1.2 MOTIVATION

The motivation for an Inventory Relocation Tool stems from a lack of regarding the stock in distribution centers. Furthermore, due to the daily need to visit different stock storing locations in a distribution center to fulfil orders, optimizing the choice of storage locations is crucially important. Suboptimal storage layouts can lead to many wasted man-hours of traveling, searching, and maneuvering associated with handling material. It is therefore valuable to develop a tool that automatically calculates and recommends location improvements to distribution center managers.

1.3 OBJECTIVE

The objective for Inventory Relocation is to develop a tool that automatically recommends new storage layouts based on the current layout, product data, and storage location data, which increases the efficiency of the warehouse by shortening the amount of time associated with visiting the storage locations.

1.4 TASK DIVISION

1.4.1 Corporate Roadmap

The corporate roadmap was developed as a group. The team deliberated on fundamental themes and considered multiple projects in order to construct a corporate plan for future projects.

Individual projects were developed by individual members of the team and those proposals appear separately in each thesis.

1.4.2 Pilot Projects

The team developed and implemented two pilot projects, a Warehouse RFID-based Dock Door System and an optimal Inventory Relocation problem. With two projects and three members, the team took a dedicated/floater approach toward allocating its resources. The author of this thesis participated and contributed during the general planning and research phase of the RFID project, but was otherwise dedicated exclusively to the Inventory Relocation project. Likewise, A. Harlalka participated in the ideation and planning phase of the Inventory Relocation project, but was otherwise dedicated exclusively to the RFID Project. On the other hand, R. Batra was engaged directly on both projects and conducted research, analysis, and testing in service of both objectives. This hybrid approach between direct collaboration and task division was crucial to the success of both projects within the time constraints of the shortened project timeline.

1.5 RELATED WORK

Lots of research has been conducted in the area of warehouse optimization. While most of the works that exist do not apply directly to this problem, many are related, and they informed some of the development and formulations of the methods in this thesis.

In terms of the optimal location for a distribution center, the so called "Uncapacitated Warehouse Location Problem" (UWLP) has been studied extensively. It is well known that the general UWLP is NP-hard, meaning that solutions are non-deterministic and can be verified in polynomial time [1], and various attempts at solving it heuristically have been made [2].

Warehouse optimization has also been studied. It has been suggested that for automated warehouse systems with grid-like storage location arrays, genetic algorithms can be used to optimally place items [3]. Approaches have also been studied to consider class-based storage [4] and consolidation [5]. There are also discussions of using linear programming techniques [6]. Optimal methods for order picking have been studied as well [7].

1.6 THESIS STRUCTURE

The remainder of this thesis is divided into three sections. The first section, containing the second and third chapters introduce Waters Corporation and discuss the Proposed Strategic Roadmap. The next section, containing the fourth, fifth, and sixth chapters, discusses the implementation of the Inventory Relocation Project, including the mathematical formulations, technical implementations, and interfaces for existing enterprise software. The final section, containing the final three chapters, discusses the results of the Inventory Relocation Project as well as the project conclusions and future work.

CHAPTER 2

BACKGROUND

This section introduces Waters Corporation, the host of this thesis project and the primary subject of the corporate and technical analyses. It details the history of the partnership between Waters and MIT. The chapter also introduces Waters' manufacturing business unit, its Massachusetts based distribution center, and the main elements of its technology and product lines.

2.1 WATERS CORPORATION: CORPORATE PROFILE

Founded in 1958 by James Waters, Waters Corporation is a $2.2 billion publicly traded corporation headquartered in Milford, Massachusetts. It is a global leader in complementary analytical technologies like mass spectroscopy, liquid chromatography, rheometry and microcalorimetry with presence in over 125 countries [11], [12].

The company boasts of an advanced design division, sprawling manufacturing facilities and service network focused on ultra-performance liquid chromatography (UPLC), high-performance liquid chromatography (HPLC), chromatography columns and chemistry products, mass spectrometry (MS) systems, thermal analysis and rheometry instruments. Waters employs over 7200 people worldwide. The company has segregated its operations into two separate divisions: Waters Division and TA Instruments [12].

The Waters Division is responsible for design, development and production of UPLC, HPLC and MS instruments and consumables which are used in a broad range of industries to measure the chemical, physical and biological composition of materials. Their products are primarily used by pharmaceutical, life science, biochemical, industrial, academic and government organizations. Meanwhile, TA instruments is responsible for the thermal analysis, rheometry and microcalorimetry instruments which are used for determining the applicability of various chemical substances used in industrial, consumer goods and health care products [12].

2.2 WATERS – MIT ALLIANCE

Incorporated in 2013, Waters – MIT Alliance has led to multiple projects in areas ranging from Research & Development to Manufacturing to Operations to Supply Chain. Every

academic year, a team of student consultants from the Master of Engineering in Advanced Manufacturing & Design program at MIT come in to augment various teams at Waters realize their strategic goals and at times, help them set a vision for the future.

The projects have resulted in patent-worthy scientific implementations and have made it possible to bring fresh innovative ideas into the scientific legacy existing at Waters. The company benefits from the out-of-the-box thinking and an outsider's perspective to their challenges. In addition to the student team, a faculty member provides the necessary subject matter expertise required to take these ideas forward. Meanwhile, the students develop systemic thinking by getting a holistic view of the business and gaining unparalleled industry experience.

2.3 ADVANCED MANUFACTURING CENTER

Located in Milford, MA, the Waters Advanced Manufacturing Center is a state-of-the-art manufacturing facility which houses the Advanced Instrument Assembly & Accessory Kitting Operations area, a Class 10000 clean room and the Global Machining Center [10].

The Advanced Instrument Assembly & Accessory Kitting Operations area is responsible for assembling the standalone kits and the HPLC instruments. The 8,500 sq. ft. clean room produces optics, micro valves and critical parts for Waters products [10].

The Global Machining Center is responsible for manufacturing various components like columns and valves which are critical to the performance of Waters instruments for distribution worldwide. Around 85% of manufacturing is outsourced to contract manufacturers; however, high- precision components and parts for New Product Development (NPD) are manufactured in this facility. The process flow in the factory is primarily job-shop in nature with different areas corresponding to turning and milling operations. The valve cell and the column area are exceptions for the existent flow is product based [10].

2.4 GLOBAL DISTRIBUTION CENTER (GDC)

Waters' worldwide distribution is centralized in three primary distribution centers across the world, the Asian Distribution Center (ADC) in Singapore, the European Distribution Center (EDC) in the Netherlands, and the Global Distribution Center (GDC) in Franklin, Massachusetts. GDC handles most of Water's outbound shipments, sending over 1500 lines per day both directly to customers and to other distribution centers as well.

In October 2017, Waters moved its Global Distribution Center into its current home in Franklin, MA. GDC is currently at a facility with 56,000 square feet of space outfitted with various types of storage, ranging from high bay areas, to standard shelving, to refrigerator and freezer locations in order to maintain over 11,000 distinct stock keeping units in over 20,000 separate storage locations. GDC has two separate loading docks and employs the traditional "flow-through" warehouse model. While GDC has dedicated staff for receiving, international and domestic picking, and international and domestic packing, many of its experienced material handlers are cross trained to enable flexibility when confronting different order patterns and daily, quarterly, and seasonal trends.

2.5 WATERS PRODUCTS

This section introduces the products which form the most significant part of product line-up for Waters.

2.5.1 High Pressure Liquid Chromatography Instruments

High Liquid Chromatography (LC) is a technique to separate a substance/material into its constituents by using a pressurized liquid solvent. The most significant markets for LC instruments are the life sciences and pharmaceutical industries which require the technique to identify new drugs, test the purity of certain pharmaceuticals and diagnose diseases. Waters manufactures highly customizable LC instruments for the end user. There are various configurations and degrees of automation which the customer can choose from, starting with component configured systems for research and teaching applications to fully automatic systems which can provide much higher throughputs in industrial settings [13].

In 2004, Waters introduced a novel technology called the Ultra-performance Liquid Chromatography (UPLC). This technology enabled the use of very small but uniform sized particles as a packing material for its columns. These packed columns when used with the Acquity UPLC system helped the customers separate the materials more reliably and achieve the separations in a faster time frame. These systems are Waters' current workhorses in the LC instrument segment [13].

2.5.2 **Mass Spectroscopy Instruments**

Mass Spectroscopy (MS) primarily allows users to identify unknown compounds/materials by measuring the weights of the molecules that get converted into ions during the process [13]. In September 1997, Waters transformed itself from a minor player in the MS instrument industry to a leader by acquiring MicroMass Ltd. based in Manchester, England. MS is frequently used with other analytical techniques like LC or gas chromatography for drug testing, nutritional safety testing and environmental testing procedures [13].

2.5.3 **Consumables**

For LC, the primary consumables are columns, which are essentially steel tubes packed with the separation media [13]. These columns have a much shorter life cycle than instruments and are typically replaced at regular intervals. The packing material for the columns is typically made from Silica or Polymeric resins. During the liquid chromatography process, the sample is introduced into the column at a high pressure and its subsequent interaction with powder initiates the separation[13].

CHAPTER 3

STRATEGIC ROADMAP

This section details the strategic roadmap that was researched and developed during the exploration phase of the 2018 Waters - MIT Alliance project. While the complete roadmap was submitted to Waters in a unified document, the proposed projects are listed below to help understand the significance of the roadmap, themes and some of the ideas.

3.1 THE NEED

The major markets served by Waters Corporation include pharmaceutical, food and materials. However, the majority of revenue (over 50%) is derived from the pharmaceutical industry slated to grow at 6% for the next few years [14]. This requires the company to ramp up production volumes amidst increasing cost pressures from the market to continue to beat benchmark indices. The scenario in the industry is changing rapidly owing to the high-growth markets in Asia, lingering fears of protectionism and the economic conditions in Europe. Waters requires a strategic overhaul to provide them the necessary edge and make their value chain agile enough to respond to any event effectively.

Traditionally, Waters has led the industry owing to its high-quality instrument portfolio that has resonated with the customers spanning from the pharmaceutical industry to the food industry. The competition is now catching up by providing innovative designs with a shorter turnaround time by incorporating feedback from detailed customer studies. For instance, some of the competitors have focused more on enhancing the overall user interface (UI) of their instruments by inclusion of smooth touch-displays whereas most of Waters instruments have an LCD screen controlled by buttons. With the passage of time, the customer needs have evolved towards UI being a major buying factor in addition to the all-round performance. The design teams should scout for such drifts in customer requirements and work towards adapting the next generation products to the same.

Industry experts cite the development of Artificial Intelligence as the game-changer in this decade. A plethora of applications of the technology has altered the way the businesses operate. The ever-increasing complexity of the supply chain has long warranted a systemic

overhaul to prune inefficiencies and work towards productivity improvements. It is recommended that the supply chain of the future be quick to adapt and pivot according to the mood of the market. The latest technological revolution can aid in better capture and utilization of data for effective decision making. However, these technologies are only as good as data input for analysis which makes it all the more important to streamline data collection procedures and use technology there as well. For instance, the use of Automatic Identifications (Auto-ID) technologies in conjunction with machine learning and data science can unearth hidden inefficiencies in the supply chain by improving visibility of product and information flow.

The first set of checkpoints in transition to the organization of the future would be to revisit the business processes and explore use cases of latest technological trends. This should be followed by a detailed technical feasibility as well as economic viability study ensuring that the suggested alterations are in line with the strategic goals. The MIT team during the exploratory phase interacted with different teams at Waters to take the first step in rethinking the way business is conducted and laid the strategic roadmap presented in this chapter. The following sections discuss about the importance of certain themes and how they fit into the roadmap and subsequently touches upon each idea in detail.

3.2 OUTLOOK

The purpose of this roadmap is to document the findings of the exploration phase in a way that identifies and directs future projects at Waters Corporation. After considering the observations and points of view of various managers, the MIT team weighed the different goals and pain points to find an overall direction for the next three to ten years.

The team highlighted four overarching themes that encompass the direction that Waters seems to want to move toward as well as twelve specific high-value projects that fit into those themes and address some of Waters' most crucial needs. Many of these projects are pilot projects envisioned to bring new approaches, thought processes, and technology into Waters' field of vision in order to drive and inform other possibilities for growth and improvement.

As this roadmap is designed to illuminate future projects, it can potentially serve as a menu for teams of interns or MIT M.Eng. teams to select projects. Guided by the leadership team at Waters, two of the projects, Inventory Management using RFID and Inventory Relocation were selected as pilots for this year's team.

3.3 THEMES

The strategic roadmap is centered around four central themes: Optimization, Automation, Digitalization, and Standardization. These have been identified as key areas for improvement across all business areas.

3.3.1 Optimization

Perhaps the simplest objective is for Waters to optimize aspects of its processes. For a very long time, Waters has relied on its industry leading products and quality to ensure consistent consumption, profitability, and growth. As most of Waters' projects sell at extremely high margins, it has been very easy for the company to sit comfortably atop the market and sustain its high level of success.

However, a recent shift in the mindset of Waters' management is in favor of working to improve process flows and efficiencies to increase product quality and reduce direct and indirect costs. It is extremely important for Waters to optimize whatever it can so that it is better equipped to keep pace with its competitors in this age of continuously evolving technology and markets.

3.3.2 Digitalization

Large corporations like Waters face challenges in sufficiently collecting, storing, maintaining, and utilizing the massive amount of data that describes its day-to-day processes. Business specific data is crucial for decision makers to carve out the competitive strategy and streamline operations. Managers across all of Waters' business units need to be able to monitor the activity and availability of their resources, machines, and materials, and the task of handling all of these concerns becomes increasingly difficult as the company grows.

On one hand, this theme involves an effort to collect more digital data from various parts of the company. And it is equally important that the data collected using conventional methods be ideal for digital processing and use. On the other hand, Waters is already collecting tremendous amounts of information in many of its business units. Effort is necessary to find uses for data that is already collected and continue to find ways to make the information profitable.

3.3.3 Automation

Currently, Waters' experiences tremendous waste through manual tasks. From repetitive physical tasks like box erection to tedious computer tasks like data entry and manual analytics, thousands of man-hours are spent doing tasks that don't necessarily require the attention of human staff members. While it is not the intention of the corporation to downsize its staff and lay off its workers, Waters still has room to grow by enabling its workers to spend more time on value-adding activities. Machines have the power to augment the ability of humans and allow them to perform more effectively.

3.3.4 Standardization

The final theme is focused on unifying aspects of Waters' business. As with many corporations, the larger Waters' grows, the more difficult it becomes to ensure that output results of work in many business areas is sufficiently homogeneous.

It was decided that effective standardization would be ensured if it's included in the upstream processes like product design. Teams responsible for design are often so pressed by their release deadlines that individual parts of the teams end up diverging in direction from other parts. Without time to realign the entire teams, the resulting work ends up with aspects from many different perspectives that do not have common features. Having processes, protocols, and component designs that do not share any portions results in increased work for staff and customers who have to deal with those protocols, as well as larger families of parts and documents to support the wide variety of entities. It is therefore proposed that making concentrated, team-wide initiatives to introduce strategic standardizations to homogenize the output of the business units will bring great value to the corporation.

3.4 PROJECT PROPOSALS

In service of the roadmap and its themes, twelve project proposals were prepared, four of which appear in this thesis, with the rest distributed between Batra [8] and Harlalka [9]. The MIT team brainstormed multiple ideas across the central themes and met with the key stakeholders in different departments to learn about their problems in more detail. It is believed that streamlining the existing business processes would be the first step in this roadmap.

Under the Optimization theme, four main ideas were considered: inventory relocation within DC, distribution network redesign, redefining shelf life to reduce scrap and determining optimal lot sizes. The team believes that an algorithm for optimal location of the inventory in warehouse could help GDC reduce its costs. Similarly, after learning that Waters' current distribution network involves significant back and forth shipping, the team believes that there could be an opportunity to optimize the current network, especially given the fact that freight costs Waters $40 million dollars each year. A meeting with the demand planning team resulted in the generation of the other project ideas based on their recommendations. One of their suggestions involved evaluating the potential of lengthening the expiration periods for certain products while the other sought to determine the optimal lot sizing structure by analyzing customer behavior and balancing trade off with manufacturing costs. It is believed that these ideas can be augmented by better quality data that can be analyzed to unearth inefficiencies.

Digitalization of some of their processes would prove essential for getting access to high fidelity data. This theme encompassed four main ideas: data-driven manufacturing, digitalization pilots for manufacturing and assembly, inventory tracking using RFID, and IoT-based HPLC system architecture. During the MIT team's interview with the manufacturing team, it was learned that many machines in the shop floor were "MTConnect ready" referring to the common data protocol shared by these machine tools. By leveraging the capability of MTConnect, the team believes that a dashboard representing the real time activity on the shop floor could be created for the shop manager, thus saving him time to inspect shop operations physically. During the team's visit to the shop floor, it was also learned that certain valuable shop floor metrics were being calculated manually. The team suggested shifting to digital methods of measurement and recommended the Digital Factory Kit offered by Tulip as a possibility. The visit to GDC, where the team observed the painstaking process of scanning individual items, convinced them of the potential of using RFID for inventory tracking. Similarly, learning about the intricacies in capturing consumer data led the team to propose the idea of introducing an IoT Based HPLC System Architecture. In addition to better data, it's required to augment the workforce with necessary machines to improve productivity.

To ensure that humans work only on value-adding activities, certain parts of the processes need to be automated. Within this broad theme of Automation, the team proposed two ideas: box erection technologies and assistive technologies for material handlers in GDC. The

team discovered that the lack of communication between TrackWise & SAP necessitated manual checks for inventory levels of raw material for LC columns. Automating the raw material as part of the larger theme of automation could help shorten the effective lead time. In a similar manner, automated box erection technology could help reduce the time required for the material handler to package and ship orders. It's also known that these improvements can only be sustained in the long term if the processes are standardized.

Within the Standardization theme, two main ideas were proposed: design standardization and modular design of HPLC system architecture. The design standardization idea focused on reducing inconsistency in product architecture that lead to greater assembly and service times. Meanwhile, the modular design aspect focused on creating a common rack architecture for LC equipment that would allow for independent disassembly of the individual modules.

The Roadmap is shown in Figure 1:

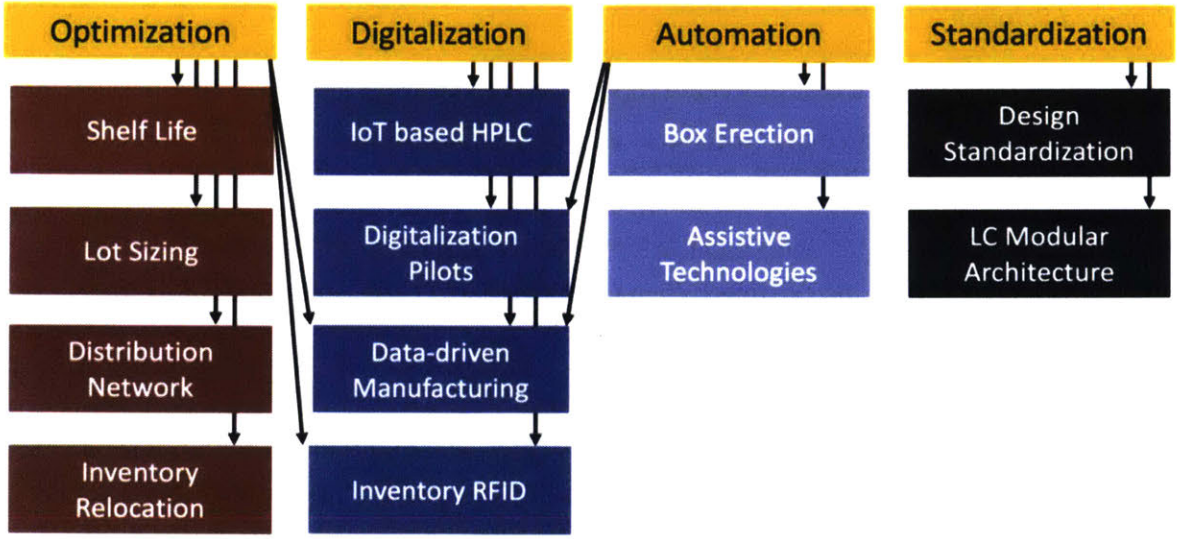

Figure 1 – Corporate Roadmap

3.4.1 **Distribution Network Redesign**

Current State

Waters currently spends approximately $40 million annually on freight. Many of the intricacies of Waters' supply and distribution networks are motivated by various customs, tariffs, and taxes associated with conducting business in all of the relevant countries. Additionally, some of the channels and nodes in the distribution network exist due to historical decisions that may

not be applicable today. And it is only because of the inertia of the shipping process and associated real estate and hardware that these channels and nodes still exist.

It is currently the case that products are being shipped in their completed form from the manufacturing plant to a distribution center and then to a customer. For example, Waters customers in the United Kingdom interested in a particular product will end up receiving a product that originated in a plant in the UK, but was sent first to Wexford, Ireland before returning back to the UK.

There are currently many such inefficiencies in Waters' distribution network and while it would be costly to make changes, the possibilities and benefits ought to be explored fully. Such an undertaking would not fit the scope or timeline of the M.Eng. program and MIT – Waters Alliance, but it still remains an integral part of the future planning for Waters at the corporate level.

Proposed Solution

In general, network redesign endeavors tend to be expansive and affect all aspects of the business. As such, it is extremely crucial that care be taken in determining which aspects of Waters' network can be changed in which absolutely cannot. For the aspects that can be changed, it is important to obtain estimates associated with various options that reflect how costly it would be to alter the existing system (be it overhead costs, capital costs, downtime costs, or inventory costs).

These estimates would provide an outer framework for a network redesign project – indicating which shipping steps can be changed, and at what cost. From there, tools and techniques for optimization of distribution networks can be used to find shipping channels and nodes that can be reconfigured or eliminated entirely.

Next Steps

As this project involves searching for a solution that could potentially restructure large portions of the corporation, it is essential that Waters create a taskforce that it trusts in order to conduct this project. Assigning such a project to a group of interns or graduate students without consultation of higher level management would be a waste of time, as any solution such a team could come up with would then need to be validated by Waters executives. It would be extremely easy (and perhaps prudent) for an executive to shy away from implementing such an

extensive change without first reexploring it. Therefore, it is recommended that Waters only task such a project to a team of external consultants if it constrains the problem so that the team can provide suggestion without having to implement the change. Alternatively, an internal taskforce of Waters personnel skilled in distribution network studies would be ideal for approaching this problem. This project has a potential to bring great value to Waters, but it is crucial that Waters is ready for such a change before investing its resources in tackling this problem.

3.4.2 Lot Sizing Optimization for Lot-Controlled Consumables

Current State

Waters sells many consumable components that are lot-controlled, meaning that the lot number is relevant in differentiating parts (as opposed to traditional parts that are identical as long as they have the same part number). Specifically, columns that contain particular powder chemistries are controlled because the manufacturing process that produces the powder guarantees that all columns made in the same batch have sufficiently identical chemical compositions, but it does not guarantee that columns from different batches have sufficiently identical compositions. Many of Waters' customers are required to perform validations on these columns that are specific to the lot before conducting experiments using Waters instruments and columns and that validation would need to be repeated if the customer were to conduct the experiments with columns from another lot. As such, Waters has the practice of differentiating column lots for the purpose of these customers, and is capable of filling orders that have lot specific requests.

At present, lot specific ordering has caused Waters problems from both overstocking and understocking perspectives. As customers continue to request columns from the lots as their previous orders, it often happens that orders are impossible to satisfy as the necessary products are completely out of stock, and producing new columns that are identical is impossible to accomplish. On the flipside, certain lots that are purchased by smaller customers with less demand end up scrapped because the expiration date comes before many of the units can be sold.

Waters has also made some unwritten purchasing agreements with customers who requested specific lots be reserved for their use only. However, as there were no binding legal contracts, Waters was not protected from the eventuality that these customers would not buy enough units from the lots and Waters has had to scrap many units on account of its generosity.

Both the manufacturing department and the order fulfillment are affected by the choice of lot sizing. For extremely large lot sizes, the cost to the manufacturing department associated with delaying other production will be very high. Conversely, tiny lot sizes introduce very high setup time to production time ratios that also tax the manufacturing department. Likewise, overly large lot sizes often cause product scrapping, while overly small lot sizes cause missed orders. As both concerns are mitigated by a "middle of the road" approach, the lot sizing is currently set by the production schedule and designed to fit approximately with the demand. This gives the status quo "medium" lot sizing. Unfortunately, there are still a lot of orders that are missed as well units that are scrapped.

Past work [17] has shown that in a manufacturing system, effective lot scheduling and sizing can be used to increase efficiency using an EQQ model originally developed by Harris [18].

Proposed Solution

The recommended optimization problem is to select a periodic schedule of lot sizes (that can still change with demand) that minimizes the costs incurred by manufacturing scheduling, missed orders, and unit scrapping. It is hypothesized that it is not necessarily most effective to have all lots be the same size as the flexibility of having some larger lots and some smaller can enable Waters to more efficiently serve its customers.

Procedures can be put in place when a customer wants a new lot to match the customer with an appropriately sized lot considering the particular customers purchasing behavior regarding lot specific ordering. Customers with the propensity to order many columns in the same lot can be 'assigned' to larger lots while smaller customers with little to no evidence of needing to repeat their orders can be fulfilled from lots that don't have quite as many units remaining.

An analysis tool can be developed to recommend ideal lot sizing that can, in conjunction with customer lot assigning practices, optimize the lot sizing to reduce costs. This tool would need to analyze the current situation and customer behavior to interpret the lot sizing possibilities. Development of a tool that takes the current situation as an input and returns a solution is extremely important so that the instead of solving the problem once, it can be solved

over and over as circumstances change. Likewise, the methods, assumptions, and parameters can be adjusted in subsequent trials so that the optimization can be improved.

Next Steps

To move forward with such a project, it is important for the scope to be well defined. While it is easy for a project planner to list all of the concerns that need to be considered in an ideal solution, every constraint has the potential to cause the optimization problem to grow with complexity. Great care needs to be exercised when determining and monetizing the costs and benefits that arise from different lot schedules so that the resulting problem is feasible to solve. It can be beneficial for some simplifying assumptions to be made during the first-pass attempt at solving this problem (i.e. assuming homogeneity in the manufacturing plant to simplify changeover cost calculation or assuming that all products have the same shelf life to simplify the scrapping considerations).

This project is well suited to be tackled by a team with a background in data analytics and machine learning. While this project would be an excellent candidate to give to a team of interns or graduate students, it would be essential for continued success of such a procedure that permanent data-oriented resources be involved. This will require multiple iterations and refinement that can only be done with time, so dedicating personnel to furthering this project would be a responsible course of action.

3.4.3 **Box Erection Assistive Technologies**

Current State

Waters currently has many business areas that require manual assembly of corrugated cardboard boxes. Particularly in the distribution centers, where outbound orders require consolidation of multiple products into shipping boxes, material handling staff members are spending large amounts of time estimating the ideal box size and assembling it. While these tasks are relatively quick when done once, the amount of time spent doing them is actually quite great when considering how often they have to be done.

Proposed Solution

There are automated technologies in existence both for box size determination (estimation based on solving the rectangular prism packing problem) and box erection. Commercially software can be explored in order to find a solution that will automatically recommend box sizes and packing layouts to material handling staff. Reducing the planning time and wasted time due to incorrect box selection can significantly improve the packing rate at GDC. Likewise, mechanical automation for box erection and bottom sealing can be explored to find an ideal solution. Shortening the packing time by preparing boxes automatically for packing can also significantly improve the packing time. There are possibilities that allow for erection of various box sizes without any changeover. The former employer of the author of this thesis is an example of an Original Equipment Manufacturer that specializes in automated packaging solutions (information has been provided to Waters but is omitted from this document for privacy).

Next Steps

Much of this project will be a technology exploration study and cost/benefit analysis. As solutions to these problems exist and are available commercially, the primary goal would be to evaluate existing solutions in order find the most appropriate solution for GDC and then to validate the selection for a favorable return on investment.

3.4.4 Inventory Relocation within DC

Current State

The layouts of Water's distribution centers were determined by planned, one-time analysis and continue to be altered by day-by-day observations and a fair amount of "tribal knowledge." In Franklin, MA in particular, where GDC has made its home since October 2017, the current layout was determined before the move in. Shelving, high bay areas and specialty storage locations (refrigeration, freezer, HAZMAT, etc.) were set up to efficiently handle material receiving and picking. The basic model used is the flow-through model, where the two loading docks are completely divorced from each other. One dock, by the front of the building, handles receiving exclusively, while the other dock, in the rear of the building, handles outbound shipping. One basic principle of inventory location – placing the fastest moving products in the

most efficient locations – was followed, thus the shelving areas are closer to the shipping area while the high bay storage area, which contains larger less frequently ordered parts, is farther from shipping. While the location of the storage hardware (the shelves and high bays themselves) would be extremely costly and non-trivial to move, the storage locations of individual parts is much simpler to relocate.

The management at GDC is interested in gaining visibility about the efficiency of the current layout and to learn easily of any alterations that could be made at any given moment that would increase the average picking rate.

Proposed Solution

Like many large corporations, Waters collects data on many of its day-to-day operations and stores records of many kinds of transactions. Much of this data is stored in its corporate enterprise software, SAP™. As part of the digitalization theme of this roadmap, it is important to Waters that it starts utilizing the data it collects to improve its business. Here lies an opportunity: to study the shipping patterns, picking patterns, and receiving patterns, to determine a way to optimize the layout at the distribution centers. Such a technique can be automated, so that distribution center managers can have a software tool that can be consulted and will recommend material relocation endeavors and initial material put-away locations.

Next Steps

The pilot for this project has been developed and is discussed in detail in the remainder of this thesis. The pilot considers high bay storage locations storage locations in GDC and primarily deals with the SKUs located in that area at the time of this study. It includes some measurement of storage locations and SKUs as well as a preliminary location cost assignment analysis.

Success of this project hinges on careful implementation and ramp-up followed by additional data collection, reevaluation, refinement, and ultimately, extension beyond the scope of the initial pilot.

CHAPTER 4

INVENTORY RELOCATION PROBLEM

This chapter details the problem, approach, and solution for automated inventory relocation recommendations. It describes the problem statement and task division between members of the team and documents the development of the objective function, constraints, and placement algorithm step.

4.1 PROJECT OVERVIEW AND PROBLEM STATEMENT

GDC shifted from its location in the Milford campus to a stand-alone warehouse in Franklin, MA around a year ago. However, the storage location for various SKUs was kept unchanged even after the drastic changes in the overall layout for the 56,000 sq. ft. space. Given the huge area, delivering over 1,500 lines every day requires the pickers to walk miles across the facility.

There have been no attempts to optimize the locations of various SKUs within the DC before or after the shift. After researching this problem, it became clear that while the GDC staff do their best to make sure to place incoming material intelligently, computationally driven recommendations would improve on their decisions and save them the time spent making those decisions. Three key areas were highlighted as ways an automatic software tool could be used by distribution center managers and personnel to improve productivity:

1. **Total Relocation**: A relocation of all SKUs currently in a particular section of the warehouse (or the entire warehouse at once). This goal involves creating a report that will list the new location of every SKU and estimate the resulting improvement to the warehouse's picking speed.
2. **Replace X**: A tool that allows the manager to request recommendations for inventory relocation procedures that involve relocation of x SKUs within a particular section. This goal is intended to give control over how extensive a particular relocation effort should be, as it will often be the case that a manager desires to improve the picking speed but does not have sufficient time and resources to relocate every SKU in the section. This too

will create a report listing the new layout and the estimated improvement, as well as a list of actual moves required.

3. **Recommended Put-Away**: A tool that recommends storage locations for every SKU present in an incoming shipment.

With all three goals, it is crucial that they be implemented as recommendations, so that each particular SKU suggestion can be considered by a human and then ignored if it is believed that there is a reason to disregard the suggestion. Especially at the initial implementation of this system, when there will be many instances of information available to the software tool being incomplete, many of the recommendations will need to be recognized by a human as not meaningful (i.e. if the system has no information on the size of a SKU and then recommends placing it in a location where it clearly will not fit, a human familiar with the materials needs to be able to disregard the recommendation).

As it is necessary to test an inventory relocation system to properly determine its benefits, it was decided that the implementation cannot involve excessive spending. For this reason, the algorithm was to be developed from scratch and tailored directly to GDC. By doing so, any costs associated with purchasing and/or applying commercial solutions (should any exist) to GDC could be avoided. As such, the need to validate the inventory relocation in terms of return on investment can be avoided as the implementation requires no capital expenditure.

In addition to the requirement that the algorithm produce relevant and mathematically sound solutions, it is also necessary for it to interface easily with existing Waters systems and people. Specifically, it must be able to grab whatever information it needs from the databases, and the output results need to be readable and manipulatable by staff members that have basic backgrounds in using computers (text files, emails, Microsoft Excel tables) but not in coding environments.

The process flow for picking operations appears in Figure 2:

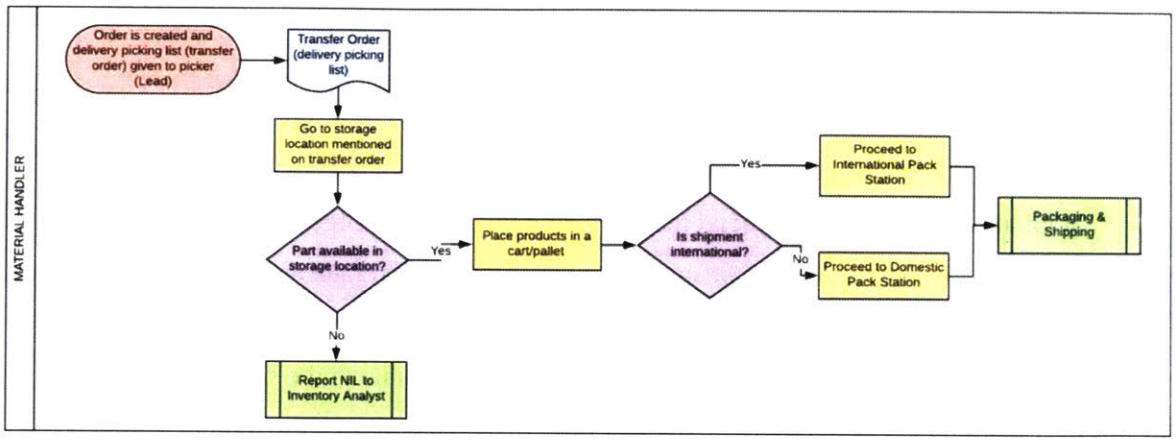

Figure 2 – Process Flow Diagram: Picking [8]

Traversing the aisles accounts for roughly 40% of the time spent in the picking, packing, and shipping process at GDC [8], so it would be extremely beneficial if that time could be shortened.

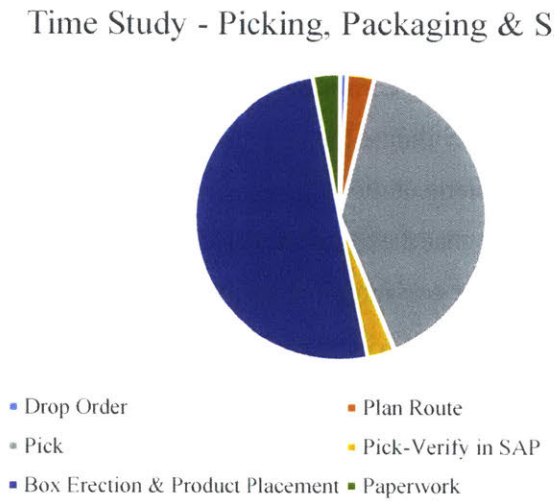

Figure 3 – Step-wise distribution of time for Picking, Packaging & Shipping [8]

4.2 TASK DIVISION

This portion of the "DC of the Future" pilot was handled primarily by the author of this thesis and by R. Batra. While this thesis should be considered the primary reference for the implemented inventory relocation project, multiple aspects of the analysis were conducted in collaboration with R Batra. Much of the necessary data collection, assessment, management, and

the location cost assignment analysis for the high bay area of GDC as well as a full cost savings analysis were done by R. Batra [8].

4.3 OBJECTIVE FUNCTION

As inventory relocation is inherently an optimization problem, it needs to be described by an objective function. At its most basic level, the goal for inventory location is to find a layout that minimizes the cost incurred by the warehouse associated with the layout. This is accomplished by finding an objective function that corresponds most directly to labor costs associated with the layout. It need not be the case that the actual labor cost be equal to or even proportional to the objective function. It is sufficient for the objective function to be related to the actual cost in that it increases/decreases whenever the actual cost increases/decreases. This way, it follows that minimizing the objective function will serve to minimize the actual cost as well with decent certainty.

Let L be the set of all storage locations and S the set of all SKUs. Let the contents of a particular storage location $l_j \in L$ be represented by the following set indicating the existence of SKU $s_i \in S$ in location l_j with quantity $q_{ij} > 0$:

$$l_j = \{(s_i, q_{ij})\} \tag{1}$$

Let a particular total layout $\mathcal{L}$ be defined as a set of all triples (s_i, q_{ij}, l_j) such that SKU s_i exists in location l_j with quantity $q_{ij} > 0$:

$$\mathcal{L} = \{(s_i, q_{ij}, l_j) \mid \exists((s_i, q_{ij}) \in l_j)\} \tag{2}$$

Then the objective function, J_0, can be represented as a sum of the costs associated with picking, receiving, and overhead associated with a particular layout $\mathcal{L}$.

$$J_0(\mathcal{L}) = C_{pick}(\mathcal{L}) + C_{rec}(\mathcal{L}) + C_{overhead}(\mathcal{L}) \tag{3}$$

In general, the primary concern was with costs that change from layout to layout, so efforts were made to determine portions of the cost that do not vary when the storage layout is changed. These simplifications, as well as some others, are discussed in the next section. The general approach with these formulations was to start with the most general and exact forms of the equations and then make simplifying assumptions to model the cost in a way that is feasible to calculate.

4.3.1 Cost Functions

For a particular horizon H (set to six months in the pilot project), all tasks require a particular number of man-seconds to be completed. Thus, the cost functions $C_{pick}, C_{rec}, C_{overhead}$, represent the amount of time spent on the task on account of the layout.

Overhead costs were estimated to remain relatively invariant across different layouts. I.e.,

$$\frac{\partial C_{overhead}(\mathcal{L})}{\partial \mathcal{L}} \approx 0 \tag{4}$$

Likewise, costs associated with receiving were estimated to vary much less than costs associated with picking from layout to layout, as the average number of like units put away in a received shipment was found to be much greater than the average number picked at once. Because of this, the amount of time spent traveling from location to location was much less important when more units were received at once. I.e.,

$$C_{rec}(\mathcal{L}) \ll C_{pick}(\mathcal{L}); \frac{\partial C_{rec}(\mathcal{L})}{\partial \mathcal{L}} \ll \frac{\partial C_{pick}(\mathcal{L})}{\partial \mathcal{L}} \tag{5}$$

Because of this situation, it was determined that the most important cost for which to optimize the layout is the picking time. The objective function was therefore reduced to:

$$J(\mathcal{L}) = C_{pick}(\mathcal{L}) \tag{6}$$

At its core, the cost associated with picking items is simply the amount of time pickers spend walking from location to location in order to pick items during a particular horizon H on account of the current layout $\mathcal{L}$. So, for all instances of traveling from a to b, the total cost associated with the layout is:

$$C_{pick}(\mathcal{L}) = \left[\sum d(a,b)\right](\mathcal{L}) \tag{7}$$

Where $d(a,b)$ is the cost associated with traveling the distance from a to b. Unfortunately, determining how this function behaves for different layouts proved to be very difficult. Pickers take a variety of different approaches to order picking (single order picking, batch picking, wave picking) and build in additional efficiencies and inefficiencies to their tasks. Ultimately, it was decided to define the total cost for this optimization by determining individual

cost amounts associated with the choice of a particular SKU being placed in a particular location. Thus, the cost function was rewritten as:

$$C_{pick}(\mathcal{L}) = \sum_{(s_i, q_{ij}, l_j) \in \mathcal{L}} c(s_i, q_{ij}, l_j) \tag{8}$$

Where the individual cost function $c(s_i, q_{ij}, l_j)$ is defined as some function of the physical volume of the SKU v_i, the quantity of the SKU q_{ij}, the frequency of the SKU f_i (number of times expected to be ordered, regardless of quantity), the expected transit time contribution from the particular SKU/location pairing $dT(s_i, l_j)$:

$$c(s_i, q_{ij}, l_j) = \mathrm{F}\left(v_i, q_{ij}, f_i, dT(s_i, l_j)\right) \tag{9}$$

The behavior of this function was studied by observing the day to day operations at GDC. Ultimately, a few key discoveries were made about the different factors:

1. The cost associated with a unit being in a location did not seem to be affected much by the number of like units present, thus the cost function was assumed to be linearly proportional to the number of units.
2. It was also not much affected by the sizing, as pickers were virtually always picking up all units of a necessary SKU at once and increases in picking time were small compared to transit time (it will eventually be shown that the location and size will be related, so the extra picking time of large, palleted SKUs will be captured by the location).
3. In terms of frequency, it was estimated that the cost function was also linearly proportional to the expected number of trips (i.e. the expected demand divided by the average number of units ordered in a line).
4. The transit time contribution appeared to be affected by a variety of factors, including the distance from the shipping area of the location, the method of picking (which depended on the type of material and whether or not it was palleted), the distance from other identical SKUs (to facilitate the primary-overflow storage strategy), the distance from other different SKUs (depending on the cross correlations or those SKUs).

Because of these observations and calculations, the individual cost function was refined as:

$$c(s_i, q_{ij}, l_j) = q_{ij} \cdot f_i \cdot dT(s_i, l_j) \tag{10}$$

Or, when considering individual units:

$$c(s_i, l_j) = f_i \cdot dT(s_i, l_j) \tag{11}$$

The transit time contribution $dT(s_i, l_j)$ was considered a function of the SKU independent location cost LC_j (the time required to travel to the location from the shipping area – see Batra [8]), the positive contribution from being near other identical SKUs due to the primary-overflow storage strategy a_{ij}, and the positive contribution from being near other SKUs with high cross correlations) b_{ij}:

$$dT(s_i, l_j) = F(LC_j, a_{ij}, b_{ij}) \tag{12}$$

$$a_{ij} = \sum_{l_k | s_i \in l_k} \alpha(d(l_j, l_k), q_{ij}, q_{ik}) \tag{13}$$

$$b_{ij} = \sum_{s_{i'}} \sum_{l_k | s_{i'} \in l_k} \beta(d(l_j, l_k), q_{ij}, q_{i'k}, corr(s_i, s_{i'})) \tag{14}$$

Where the term $\alpha(d(l_j - l_k), q_{ij}, q_{ik})$ represents the positive value associated with SKU s_i being located in both locations l_j and l_k based on the added efficiency of using one location as primary storage and the other location as overflow storage (a strategy typically employed at GDC for fast-moving SKUs). The term $\beta(d(l_j - l_k), q_{i'j}, q_{i'k}, corr(s_i, s_{i'}))$ represents the positive value associated with SKUs s_i and $s_{i'}$ being location in locations l_j and l_k with a given cross-correlation $corr(s_i, s_{i'})$ between the two SKUs that represents the likelihood of the two SKUs being ordered together.

Ultimately, the effect functions α and β were very difficult to model, and the amount of time required to collect data for the distances $d(l_j, l_k)$ would be prohibitively great – if the amount of work required to map distance costs from n storage locations to shipping as done in Batra [8] was $\sim O(n)$, then the amount of work to map distances for every possible pair of the n storage locations would be $\sim O(n^2)$. Therefore, it was decided to simplify the transit time contribution further to be:

$$dT(s_i, l_j) = LC_j \prod \eta_r \tag{15}$$

Where each η_r is a procedural efficiency or inefficiency that scales the location costing. For example, two of these efficiencies are batching and routing where the fact that multiple lines are picked at once and the fact that a route is planned by a human enable the walking time from shipping to the storage areas to be shared across multiple lines. The important assumption that these factors are invariant regardless of the layout is justified by considering average/expected behavior for the order compositions and thus assuming that the practice of batching will have, on average, approximately the same savings for all layouts.

These simplifications were also advantageous for constructing the algorithm, as with this formulation, the location cost becomes independent of the SKUs themselves.

Thus, the objective function was reduced to:

$$J(\mathcal{L}) = \prod \eta_r \sum_{(s_i, q_{ij}, l_j) \in \mathcal{L}} q_{ij} \cdot f_i \cdot LC_j \tag{16}$$

And as the efficiencies term was assumed to be multiplicative and invariant with different layouts, it can be removed to form the simple objective function:

$$J(\mathcal{L}) = \sum_{(s_i, q_{ij}, l_j) \in \mathcal{L}} q_{ij} \cdot f_i \cdot LC_j \tag{17}$$

4.3.2 **Constraints**

Thus far, the possible layouts $\mathcal{L}$ that can be considered are limited only by which SKUs are known to be in the warehouse. Without further limitation, SKUs can be placed in locations to the point where the total volume of the SKUs exceeds the allowable volume in the storage location which cannot be tolerated. Thus, a constraint on possible layouts was formulated:

$$(\forall l_j \in L): \quad \sum_{(s_i, q_{ij}, l_j) \in \mathcal{L}} q_{ij} \cdot v_i \quad \leq \quad v_j \cdot PF_j \tag{18}$$

PF_j is the allowable packing fraction associated with location l_j and v_j is the total true volume in that location. Setting the packing fraction to values less than 1 allows the volume occupied to be constrained further than the total true volume as filling the full volume of a storage location entirely is horribly inefficient as it is very difficult it to fill the location and pick material from it.

Additionally, GDC currently has a policy in place to avoid picking errors whereby lot-controlled SKUs of the same part number but different lots are not permitted to be in the same location.

Thus, the solution desired is the layout $\mathcal{L}$ which minimizes the objective function defined in Equation 17 subject to these conditions.

4.4 PLACEMENT ALGORITHM

The basic premise understood as common sense by everyone in GDC is to place the most popular items in the most prime locations. What was unknown is exactly how to define what are the most popular items and what are the most prime locations.

The algorithm that was developed is a greedy algorithm that simply places the "best" SKUs in the "best" locations and repeats until all SKUs are placed. The constraints on this algorithm and its optimality are discussed in the next sections. The process flow of the algorithm follows the following pseudocode representation (note that this pseudocode is simplified and contains procedures in English that are intuitive but not necessarily formal functions – the full code appears in the Appendix):

Algorithm 1 – Basic Placement Algorithm

Function: Placement Algorithm (*Currently Placed, Locations, SKUs to Place*)

Input: List of locations of already placed stock, Total number to place of each SKU
Output: Final Layout

```
1     Order: Locations BY best to worst
2     Order: SKUs to Place BY highest popularity to lowest
3     For i in (SKUs to Place):
4           Best SKU = i
5           For j in Locations:
6                 If Best SKU FITS in j:
7                       Place: Best SKU in j (as many as will fit)
8                 end
9           end
10    end
```

The algorithm is also represented by the flow chart depicted in Figure 4:

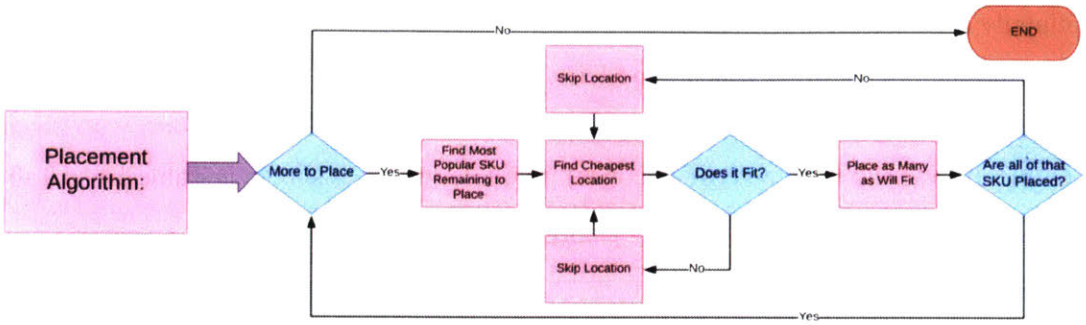

Figure 4 – Placement Algorithm Flow Chart Representation

The main advantages of this algorithm are its simplicity and versatility. It assumes one key premise – at any given moment, there is a current *best* placement to be made, i.e. a pair (SKU and location) that is most advantageous to be made at the current moment (hence the use of a greedy algorithm). This algorithm was also chosen as it allows a variety of different applications ranging from a total relocation (i.e. setting *Currently Placed* to be empty and placing all the SKUs) to a relocate X (define *Currently Placed* as the starting layout and then remove the X SKUs that have the greatest cost and place them again) to a shipment received routine (i.e. setting *Currently Placed* to the starting layout and placing the contents of the shipment).

The only is missing piece in the function definition is how to define "best" for locations (which appears in line 1) and "highest popularity" for SKUSs (which appears in line 2). This proved to be a bit difficult.

The best locations were considered to be the locations that were closest to the shipping area and thus, most accessible to pickers. Analysis was conducted to develop a method to determine SKU-independent location costs for each storage location in the warehouse and is described in detail in Batra [8].

As the frequency of picking of individual items is defined previously as number of expected trips per unit time, it was first considered that the popularity of the products be defined directly as the frequency. In fact, asking most people at GDC to give their gut feeling produced this definition. However, what this definition fails to consider is the size of the different SKUs

and the following example, with just two SKUs and two locations, shows how this criterion fails to optimally organize the warehouse:

Consider a scenario in which two SKUs, with given data on size (in cubic inches), quantity, and frequency (number of trips per hour), are to be placed into locations, with given size (cubic inches) and location cost (seconds). The information is shown in Table 1 and Table 2:

Table 1 – Example 1 SKUs

	SKU 1	SKU 2
Size (in^3)	1	2
Quantity	10	10
Frequency (trips/hr)	3	4

Table 2 – Example 1 Locations

	Location 1	Location 2
Size (in^3)	12	18
Location Cost (s)	5	6

Following the criterion defined above, the most popular item, SKU 2 will be placed into the better location, Location 1 as much as can fit (6 units) and the remaining units of each SKU will be placed in Location 2. This procedure yields the following results:

Table 3 – Example 1 Results Following Pure, Frequency-Based Placement

	Location 1	Location 2
Number of SKU 1	0	10
Number of SKU 2	6	4
Cost SKU 1	0	180
Cost SKU 2	120	96
Volume SKU 1	0	10
Volume SKU 2	12	8
Total Cost	396	

However, if SKU 1 was considered the more important one to place and it therefore was placed into the better location, Location 1, the following results would be obtained:

Table 4 – Example 1 Results Without Following Pure, Frequency-Based Placement

	Location 1	Location 2
Number of SKU 1	10	0
Number of SKU 2	1	9
Cost SKU 1	150	0
Cost SKU 2	20	216
Volume SKU 1	10	0
Volume SKU 2	2	18
Total Cost	386	

Thus, it is unlikely that considering only the frequency of a product without considering its size is optimal.

Because of this discovery, including the size in determining the most crucial SKU was considered. To accomplish this, the utility of a SKU u_i was introduced and defined in the following way:

$$u_i = f_i \cdot \mu(v_i) \tag{19}$$

Where $\mu(v_i)$ is some monotonically decreasing function of the volume of the SKU. This way, SKUs are considered best when they are visited frequently while also occupying little space. The simplest functions that can be used are the basic reciprocal function, $\mu(v_i) = 1/v_i$, and a difference function, $\mu(v_i) = V - v_i$ for some $V\ s.t.\ (\forall i)\ V > v_i$. Multiple possible functions were considered, but ultimately, it was decided to use the reciprocal in order to relate the utilization in a useful, understandable way – "frequency per unit volume." In other words, a SKU with a frequency of 50 trips per month and a volume of 10 in^3 will be considered to have a *utility* of 5 trips/month/in^3. This function was also chosen due to its absolute optimality in a relaxed case where SKU volumes can be split among different places (a formal proof appears in the next section). The function thus becomes:

$$u_i = \frac{f_i}{v_i} \tag{20}$$

It is important to note that with this utility function used in the algorithm, the example shown in Tables 1 and 2 is solved correctly as SKU 1 and SKU 2 have utilities calculated as 3, and 2 respectively, and thus SKU 1 is prioritized.

4.4.1 Algorithm Optimality

The optimality of this algorithm hinges on the definition of the objective function. As the objective function has been defined in Equation 17, it is clear that an optimal layout must exist, and it can be shown that the chosen algorithm does a sufficient job approximating it.

Specifically, a layout $\mathcal{L}^*$ is sought that minimizes the total layout cost in the objective function. Obviously, a brute force algorithm that checks every layout is out of the question, so the algorithm in the previous section was selected. The only question is, how close to the global optimum $\mathcal{L}^*$ can one expect the algorithm's output layout to be?

In fact, for the "continuous" version of this problem (i.e. if it were possible to split SKUs up into infinitely tiny pieces and place fragments of them in any location), it can be shown conclusively that the algorithm presented here in fact generates the optimal layout $\mathcal{L}^*$. Consider the following proof:

Let Δv be an infinitesimally small volume unit and assume that each SKU is quantized in volume as $v_i = n_i \cdot \Delta v$ where n_i is a natural number. The portion of the cost associated with the placement of SKU s_i is assumed to split evenly according to where its volume is placed (i.e. $\frac{1}{n_i}$ of the cost is borne for each volume element of size $\cdot \Delta v$). Thus, the cost associated with placing a portion the size of Δv of SKU s_i in location l_i then given by:

$$c(s_i, l_j) = \frac{f_i \cdot LC_j}{n_i} \tag{21}$$

Let the utility of each SKU be defined as:

$$u_i = \frac{f_i}{n_i} \tag{22}$$

This means the cost can be rewritten as:

$$c(s_i, l_j) = u_i \cdot LC_j \tag{23}$$

Now if the total volume of all SKUs in the warehouse v_{total} is given by $v_{total} = N \cdot \Delta v$, then there are N differential volumes to be placed among the locations and M storage spaces of size Δv to be used by the SKUs. If all SKUs fit in the warehouse, then it must be the case that $M \geq N$. Assume that the $M - N$ storage locations with the highest costs are disregarded entirely

as they are not necessary. Thus, the remaining N locations can be listed by location cost in the multiset of length N:

$$LC_{all} = \{LC_{j_1}, LC_{j_2} \dots\} \tag{24}$$

Where each differential storage space has a location cost LC_{j_k} associated with the location l_{j_k} it is in.

The utilities of the N differential volumes can be listed in the multiset of length N:

$$u_{all} = \{u_{i_1}, u_{i_2} \dots\} \tag{25}$$

Where each differential volume has a utility u_{i_k} associated with the SKU s_{i_k} it is part of.

Thus, the total layout cost $J(\mathcal{L})$ of a layout is simply the sum of the pairwise products of the elements of LC_{all} and u_{all}, paired by where each differential SKU volume is placed.

By the Rearrangement Theorem [16], the minimal sum of pairwise products of two multisets is achieved by ordering one set with the smallest element first and the other set with the largest element first and pairing them in that order (i.e. the smallest element of one set with the largest of the other and so on). Therefore, the minimal value for the total layout cost $J(\mathcal{L})$ is achieved the same way – by placing the costliest SKU volume elements in the cheapest locations in order.

Thus, the placement algorithm that defines utility this way in fact generates the global optimal solution $\mathcal{L}^*$ for the scenario where SKU volumes are continuous and can be split into multiple locations.

In the real, "discrete" problem, SKUs cannot be split, so, when the algorithm has a SKU of a certain volume and a few most optimal storage locations with not enough space left, it is forced to skip the most optimal storage locations in favor of the next best storage location which also has room. Obviously, the total cost will increase as the space is less perfectly utilized, but the optimality of the method is still excellent (though not perfect). When storage locations are not close to full, the discreteness of the size of the SKUs will not matter much, and thus, any inaccuracies the algorithm would make in these situations can be considered "edge effects" that, when looking at the whole picture, are relatively negligible (storage locations can store anywhere from 10-10,000 individual units, so the amount of times the filling up of a location will actually matter is very few). Only extremely large SKUs encountered later on in the algorithm have the

potential to increase the cost by being placed into worse positions, but as there are a wide range of storage location costs, sizes, and SKU sizes, it is likely that these outliers will not have much of an effect.

The following example highlights a scenario in which this algorithm makes inaccuracies. One SKU is much larger than the others and one location is much costlier than the others, and this prevents optimal placement:

Table 5 – Example 2 SKUs

	SKU 1	SKU 2	SKU 3
Size (in^3)	1	2	10
Quantity	10	10	2
Frequency (trips/hr)	3	4	29

Table 6 – Example 2 Locations

	Location 1	Location 2	Location 3
Size (in^3)	12	18	20
Location Cost (s)	5	6	10

Following the algorithm, the locations are ranked 1, 2, 3 (due to their respective costs 5, 6, 10) and the SKUs are ranked 1, 3, 2 (due to their respective utilities, 3, 2.9 ,2). Thus, it fills the space in the following way:

Table 7 – Example 2 Results Following Placement Algorithm

	Location 1	Location 2	Location 3
Number of SKU 1	10	0	0
Number of SKU 2	1	4	5
Number of SKU 3	0	1	1
Cost SKU 1	150	0	0
Cost SKU 2	20	96	200
Cost SKU 3	0	174	290
Volume SKU 1	10	0	0
Volume SKU 2	2	8	10
Volume SKU 3	0	10	10
Total Cost	930		

It does so by first placing SKU 1 into Location 1 because it is best and then having to place the two large, popular SKU 3 units one in each of Locations 2, and 3, and then placing

SKU 2 into the remaining space. However, inspection of this method shows that the difference in utility between SKU 1 and SKU 3 is very small while the difference between SKU 3 and SKU 2 is large. So, utilizing good space for SKU 3 versus SKU 2 is likely more crucial than utilizing good space for SKU 1 as opposed to SKU 2. It can be seen that the algorithm blindly places the best SKU (SKU 1) without looking at the relationship between SKU 2 and SKU 3. Once it does so, there isn't enough room for the large SKU to fit both units into Location 2 and one ends up in the terrible Location 3. This would suggest that perhaps placing the two larger SKU 3s first (despite its lower utility than SKU 1) in the two better locations before placing SKU 1 could be beneficial.

In fact, in this particular example (where the numbers have been chosen carefully to illustrate this point), the discrete sizes of the SKUs and locations forces the algorithm to make suboptimal decisions. The following results are not generated by the algorithm but are better:

Table 8 – Example 2 Results Without Following Placement Algorithm

	Location 1	Location 2	Location 3
Number of SKU 1	2	8	0
Number of SKU 2	0	0	10
Number of SKU 3	1	1	0
Cost SKU 1	30	144	0
Cost SKU 2	0	0	400
Cost SKU 3	145	174	0
Volume SKU 1	2	8	0
Volume SKU 2	0	0	20
Volume SKU 3	10	10	0
Total Cost	893		

As it can be seen, this algorithm fails to notice the importance of looking at multiple SKUs at once, and, for a large scale, this can make the results inaccurate if the numbers are as inconsistent as they are in this example.

4.4.2 Summary of Necessary Assumptions

In order to complete the calculations in the previous sections, simplifying assumptions were made in order to make the algorithm feasible, efficient, and useful. Ultimately, improvements to the algorithm can be made by solving the objective function, constraints, or

algorithm steps by removing the assumptions. The assumptions are included here for easy reference.

Location Cost Assignment Independent of SKU

The most fundamental assumption is that the location cost any storage location is not contingent on what SKUs are there (or what SKUs are in other relevant places). This means items taken from storage with a forklift are treated the same as those taken off by hand. It also means that the effects of the primary-overflow storage strategy and SKU cross-correlations cannot possible be considered by the algorithm. While this seems prohibitive it is important to note that some of the results that will come from this lack of fidelity are results that are desired anyway. For example, as the higher bay areas were given costs associated with accessing them via a reach truck, it technically overestimates the cost of placing a smaller item up there, but ultimately, placing the smaller items at ground level is desired anyway, so such a lack of distinction can certainly be tolerated. As for the primary-overflow storage strategy, it is easy for a human to intervene and ignore results suggesting moving the storage portion down to a theoretically better location.

Removing this assumption is necessary to effectively consider most of the additional effects that are highlighted in this section.

Actual Transit Cost Proportional to Objective Function

A big assumption made in Equation 15 is that the efficiencies added by various strategies will continue to add efficiency for any layout. Obviously, this is not truly guaranteed, but the best that can be done without considering routing or batch ordering is to assume that on average, this is true.

Cost Function for SKU in Location is Linearly Proportional to Quantity

An assumption made to get from Equation 9 to Equation 10 was that the contribution of having $2x$ units in a location is double that of having x units of the same SKU in that location. The justification for this is that the number of units drives the number of trips. It is however known that there are efficiencies at the overhead level, the receiving level, and the inventory counting level, to group like SKUs where possible. Due to the discreteness of the different utility

values, the algorithm will always attempt to put as many like SKUs in the same place anyway, but it does not account for the added benefit in any way.

Many More Units Received at Once than Picked at once for all SKUs

A big assumption made in Equation 5 is contingent on the fact that all SKUs arrive to GDC in much larger quantities than those in which they are picked. Specifically, it requires that:

$$(\forall i)\ \frac{(avg_{received})_i}{(avg_{picked})_i} \approx 0 \tag{26}$$

Where $(avg_{received})_i$ and $(avg_{picked})_i$ represent the average number of units received and picked, respectively, for each SKU s_i. Obviously, this is not truly guaranteed, but the best that can be done without considering the routing or batch ordering is to assume that on average, this is true.

Sets of Location Costs and SKU Utilities are Consistently Spaced, More Space in GDC than SKUs

For the optimality of the algorithm, it is necessary to assume that the different costs for locations, while not truly a continuous set of values, is approximately evenly spaced without any large jumps. It is likewise necessary to assume this for the set of location costs and SKU utilities. The choice of utility function in Equation 20 was made to order the SKUs by their frequency per unit volume, but this is only optimal when the differences between adjacently ranked location costs and SKU utilities are approximately consistent.

The assumption that the numbers are consistent is therefore necessary for the algorithm to generate optimal or approximately optimal results. Note it is also the case that the SKUs fit exactly into the locations in Example 2, and thus, with the extra space in GDC, it is reasonable to assume behavior like this is unlikely to occur.

CHAPTER 5

INVENTORY RELOCATION IMPLEMENTATION

This chapter details the actual implementation of the relocation algorithm. This includes the Python code, the strategy for storage of constitutive data and downloading of variable data, the tools developed for the user interface, and designed flexibilities to facilitate future improvements.

5.1 PYTHON 3.6 ENVIRONMENT

Early in the development process, it was decided that Python would be used to write the code for the algorithm. In order to select the coding language, multiple factors were considered:

1. Familiarity by the team
2. Ease of sharing and collaborating
3. Cost to install
4. Interface with data files currently used by Waters (currently, Excel spreadsheets downloaded from SAP)

With these considerations in mind, Python emerged as the obvious choice due to its modular design, free installation, and intuitive data file interfacing possibilities.

Python is widely used in many industries and many Integrated Development Environments (IDEs) are devoted solely to development with Python. For this pilot, the Spyder IDE (part of the Anaconda package) was used for development purposes, and Waters staff have downloaded it in order to see a similar interface. It should be noted that any Python IDE is sufficient to run the developed scripts provided the computer running them has installed the right version of Python and the necessary libraries.

The Spyder IDE interface is shown in Figure 5:

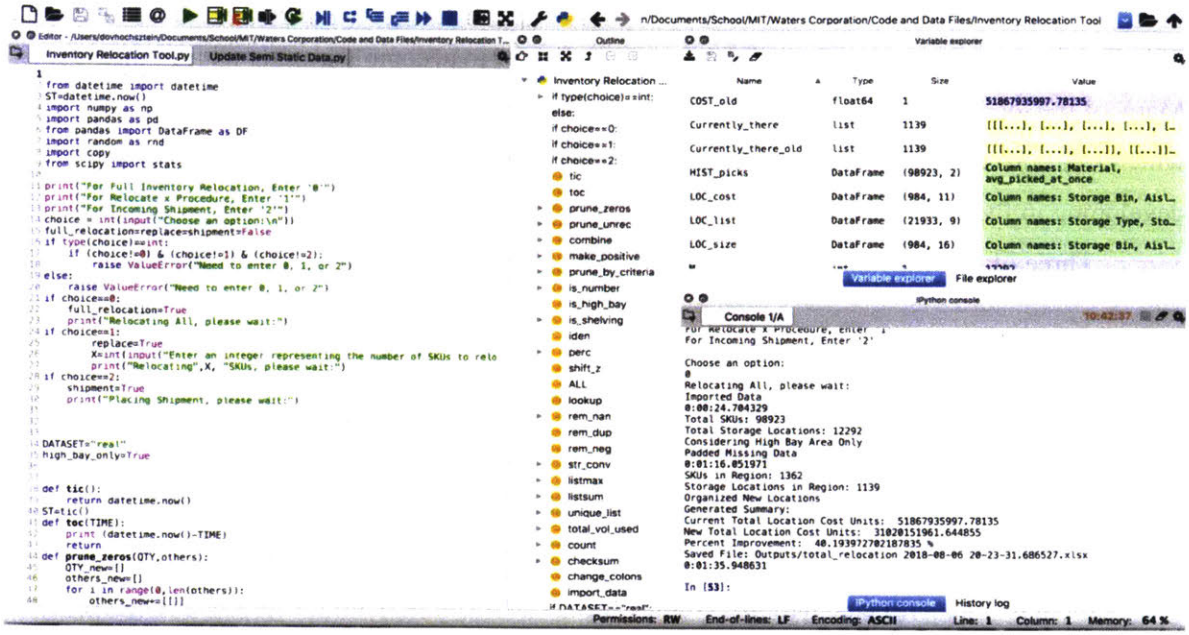

Figure 5 – Spyder Development Interface

5.1.1 **Open Source Development**

The first important property of Python that makes it ideal for this project is its modular, open source platform that enables the use of content created by developers from around the world. The modules are basic plug-and-play add-ons that enrich the capabilities of the already versatile language.

As this implementation is designed to be a pilot, the flexibility Python offers by supporting multiple types of programming paradigms is crucial to enable programmers with various skills and styles to be able to interpret the design intent and further develop the pilot.

5.1.2 **Fully Supported Libraries**

In addition to the benefits of open source development, Python boasts a wide breadth of developed code modules that the Python community has termed "Libraries." Many are packaged with the basic Python module and do not require further downloads. However, even the native libraries need to be imported in a script before running them. Python uses this technique to save memory, so that running a script will require storage of only the libraries it needs.

Python's basic modules are extremely well supported and many pages of documentation are available online on the commands within. Likewise, for most other modules, developers

provide documentation to support their releases. In addition to official documentation, hundreds of online forums exist where users post questions on how to accomplish goals with Python code. Many of the top answers even provide time studies and run time order calculations to compare which solutions are the most efficient under various circumstances. For these reasons, developing in Python is made increasingly easier as the community and discussions grow.

Three of the most important libraries used for this project are described in detail:

numpy

Numpy is one of the most widely used Python libraries as it allows the use of the array object type, that does not otherwise exist in Python. One of Python's most fundamental structures for storing multiple elements is a list, which can be nested to form an array of values. However, a list or list of lists cannot function like a vector or matrix in order to do vectorized (i.e. element by element) calculations. Numpy arrays are extremely useful for linear algebra and are helpful for engineers learning Python who are familiar with MATLAB as the numpy array is very similar to the default MATLAB object type.

pandas

One of the most important features of the pandas library is its native "DataFrame" object type. The DataFrame type is essentially an Excel spreadsheet format with rows, columns, and row and column names. It can be manipulated and sorted like Excel spreadsheets and it is trivial to go back and forth between the DataFrame object and list or array objects. It was a crucial to this project to use DataFrame objects to enable seamless integration back and forth between Excel data files and the Python script as they enabled easy importing and exporting of data. The pandas library commands are also highly optimized and as this application requires handling some Excel spreadsheets with >200,000 rows, the efficiency is also crucial.

os

The os library was instrumental in enabling the script to access and rename other files in the directory. Much of the work in exporting and storing data files involved the ability to find a file with a particular name and archive it with a new name before generating a new version of it. Thus, the os.rename command was very useful.

5.2 MICROSOFT EXCEL DATASHEETS

Waters currently uses SAP as its corporate enterprise software. Embedded within the SAP system is a large data repository which can be probed via queries and reports by credentialed users whose authorization is deemed sufficient for the relevant information. The SAP software allows reports to be easily exported into Excel format. Additionally, columns can be individually requested when the reports are generated and the column names are determined based on the names of the desired fields. This consistency is extremely advantageous when attempting to build software applications on top of the data.

At the start of this project, various data files were provided by Waters staff to the team in order for the team to learn how the data is stored and what it looks like. This data was processed and analyzed in order to determine how best to use and store the data that is to be used by the inventory relocation algorithm. The script was built around grabbing data from these files, using the column names as defining features to ensure that the correct data would be captured regardless of order the columns were selected. Eventually, the program was developed so that it would operate on the available data files in its working directory, and create other data files and outputs as required.

Ultimately it was determined that data of four specific categories would be stored in folders (folder names are indicated parenthetically) in the working directory of the script:

1. Static Data ("Static") includes all data that does not generally change at all with the passage of time.
2. Semi-Static Data ("Semi Static") includes data that changes very little with time.
3. Dynamic Data ("SAP Data") includes all data that needs to be downloaded freshly from SAP with every use of the software tool.
4. Output Data ("Outputs") includes all published reports made by the software tool. These are labeled and timestamped to indicate what they represent and when they were generated.

These four categories were identified and separated so that humans using the software would be able to know where to find various datasheets and be able to change or collect the ones they need.

A directory diagram of the datasheets is shown in Figure 6:

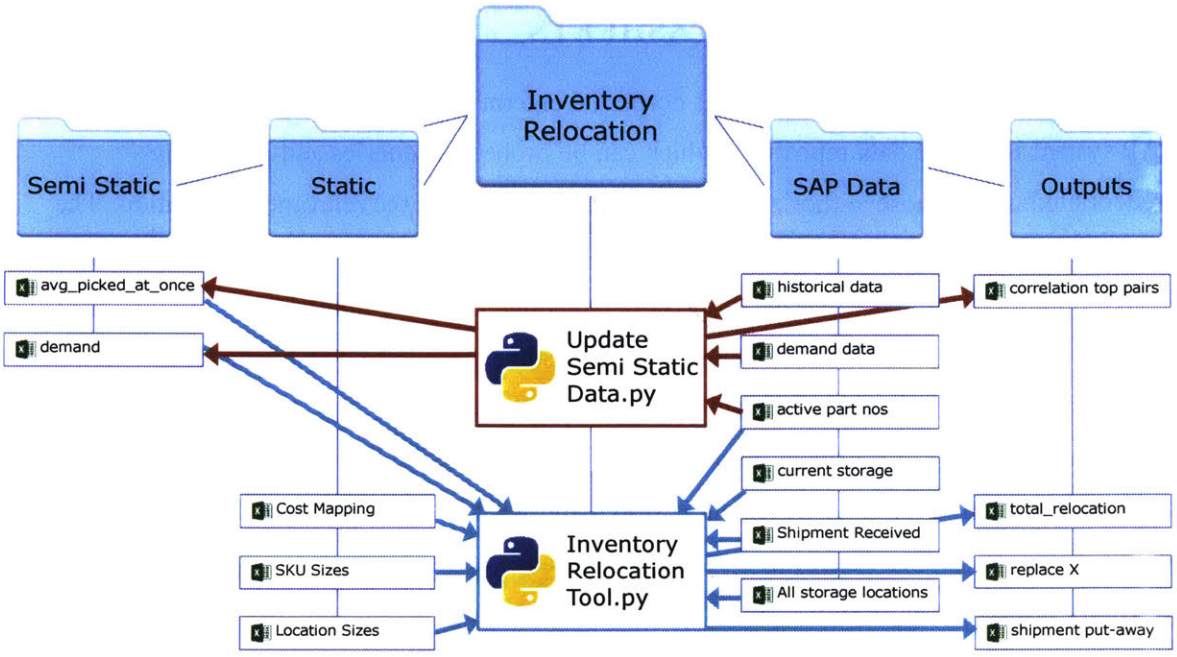

Figure 6 – Inventory Relocation Tool Environment Overview

5.2.1 Static Data

This folder was designed to contain all generally unchanging information about the SKUs and storage locations in GDC. Specifically, it contains the dimensional SKU sizing data, the dimensional storage location sizing data, and the location cost assignment data.

"SKU sizes.xlsx"

This datasheet contains measurements of SKUs stored in GDC. For the pilot project which dealt with SKUs located exclusively in the high bay areas, this is limited to the ~1,000 SKUs found there as opposed to the ~10,000 total SKUs in GDC. The sparsity of measured data on SKUs in other regions is one of the primary reasons that the high bay area was selected for the pilot project. And as it is, only ~400 of the SKUs there actually have measurements populated, and it is suspected that some of the entries contain errors. The purpose of this spreadsheet is to list all SKUs with known dimensions so that the placement algorithm can make well informed choices. The algorithm was designed to infer missing measurement values, but the results will increase in robustness as more data is collected.

The data is listed as length, width, and height of SKUs (assuming a rectangular prism geometry) as well as a number per box entry to account for SKUs that are stored in boxes. It was determined that the most efficient way to estimate the space that one SKU would occupy would be the size of the box divided by the number of units generally found in one box. Therefore, the final column of this spreadsheet represents the estimated spatial volume that a single unit of a SKU occupies. It is important to note that as some SKUs are nonstandard in shape or packaging, the software requires only the volume column, so new entries can be added by simply entering the estimated volume per unit without using the formula involving the length, width, height, and number per box.

"Location Sizes.xlsx"

This datasheet contains measurements of storage locations in GDC. For the pilot project, this is again limited to the ~1,000 locations there as opposed to the ~20,000 total locations GDC. Another reason for selection of the high bay area as the pilot is the lower variability of shelving storage location sizes in the high bay region. The purpose of this spreadsheet is to list all storage locations with known dimensions so that the placement algorithm can make well informed choices. The algorithm can again infer missing values, but here too, the results will increase in robustness as more data is collected

The location data is listed as length, width, and height as well as a packing fraction to account for the fact that occupying 100% of the space in a storage, while being efficient space utilization in a technical sense, is not an effective way to make material accessible. The packing fraction was estimated as 70% for the high bay area but it is anticipated that this value might be higher for shelving, and can be manually adjusted for any location by editing the spreadsheet. Changes in bay heights can also be adjusted in the spreadsheet (and values can be allowed to vary from one location to another). Ultimately, selection of the packing fraction parameter is a tradeoff between the packing efficiency and the accessibility of SKUs in the location. The ability to access SKUs is strongly determined by how much of the physical space is filled. The sensitivity of the output results to this parameter is relatively low, as shown in Table 9:

Table 9 – Sensitivity Analysis of Packing Fraction

Packing Fraction	Relative Resulting Objective Function
50%	+0.82%
60%	+0.82%
70%	Baseline
80%	-0.43%
90%	-0.65%

Thus, it is reasonable to choose the packing fraction considering the accessibility within the locations and allow the results to be driven by that decision.

"Cost Mapping.xlsx"

This datasheet contains calculations of the location costs associated with each storage location. Values are calculated formulaically based on the shelf, bay, and level, designations using the analysis described in Batra [8].

5.2.2 Semi-Static Data

This folder contains up to date data files that are generated periodically from other data as well as archives of older versions of these files. The main purpose of keeping these files semi-static is that the runtime for the necessary calculations was much greater for these files than for the main algorithm operations (~40 versus of ~1.5 minutes) and it was decided that the change in data over even a period as long as a week was so negligible that it hardly paid to require this analysis to be performed every time the tool is consulted. Thus, the optimal procedure is to run the script that generates the semi-static data periodically and setting it as a background task if possible. Every time the main script is run, it simply uses the newest versions of the semi-static data files.

"demand.xlsx"

This file indicates the demand for every SKU in GDC over a particular horizon (for the pilot it has been set at six months, but this can be changed easily). The demand is gathered from two possible sources:

1. The demand forecast published by the planning group for the next six months (or another period if the horizon is changed)

2. The archive of all lines picked for the previous six months (or another period if the horizon is changed)

The dual sourcing for demand estimates was chosen so that the demand estimates would be most complete, as neither the forecast nor the historical data necessarily reflects the popularity of every SKU. The planning group does not forecast every single SKU, and the historical data can have no entries for a SKU (either by some coincidence or for a newer SKU) or be generally an inaccurate sample. Thus, an attempt to combine the information was made.

The first step in considering the forecast data was to determine a scaling factor that reflects the ratio between number of units *sold* from GDC to number of units *shipped* from GDC as the latter includes units shipped to other internal Waters distribution centers. The actual quantity that is of interest is the number of units shipped from GDC as even units sent to other Waters distribution centers need to picked off the shelves. Thus, after analyzing the SKUs for which both a forecast for the next six months existed as well as sufficient data from the previous six months, it was determined that this ratio is approximately 60%. With this figure, all forecasts were scaled up by a factor of ~1.66 to reflect how they would translate to number of units picked off the shelves.

With the two options for demand, the following scheme was followed to get demand values for all SKUs:

1. If neither the forecast nor the previous six-month history contained values for a particular SKU, a demand of 0 was used (ultimately 0.001 was used to facilitate subsequent calculations).
2. If the forecast had a value but the history did not, the forecast value was used.
3. If the history had a value but the forecast did not, the history value was used.
4. If both contained values, a weighted average of the two (currently set at 60%-40% in favor of the historical demand, but this can be changed) was used.

As the situation where both values were available was relatively uncommon, the results were determined to be relatively insensitive to the weighting factor.

"ave_picked_at_once.xlsx"

Since the demand simply conveys how many units are expected to be picked but not how many times a picker would need to pick an item from the location, the average number of units

picked at a time was calculated for each SKU from the historical pick data. This would allow a "trip demand" to be determined simply by dividing the demand by the average line quantity (or equivalently, counting the number of times a SKU appeared as a line item, regardless of the line quantity). For SKUs that were never picked in the previous six months, a value of 2000 picked at once was set to automatically make the trip demand very low to reflect the low popularity (since these products were never picked, it is likely that the demand will have been set to 0.001 anyway, so this precaution is a redundancy in those cases).

5.2.3 Dynamic Data

This folder is the target location for all data downloaded from SAP. These files are required to be downloaded from SAP before running the scripts so that the critical information is completely up to date.

"active part nos.xlsx"

This lists all active Waters part numbers so that running the script can match other data files against this master list. It serves as a compatibility check to make sure none of the SKUs listed in the other data files are unknown to SAP. It can be downloaded from SAP by running 16 (Plant: US10, Storage Location: F101) and specifying the column "Material".

"current storage.xlsx"

This spreadsheet is effectively a snapshot-in-time of the GDC layout and lists where every SKU is at the present moment. The script uses this to generate the present state from which it starts its analysis as well as the benchmark for cost calculations. It can be downloaded from SAP by running WC in SQ01 and specifying the columns "Stor.bin", "Material", and "Batch".

"All storage locations.xlsx"

This lists all storage locations in GDC. Like the SKU list, it is included to check compatibility of other location-related data files by ensuring that none list unknown locations. It can be downloaded from SAP by running LX03 and specifying the column "Storage Bin".

"Shipment Received.xlsx"

This spreadsheet is what would be populated in the event of an incoming shipment that needs to be placed using the algorithm. It can be created manually (or downloaded from SAP by running a report designed for this application) and requires the columns "Material", "Quantity", and "Batch".

"historical data.xlsx"

This spreadsheet is a complete log of all shipped orders for the previous 6 months. It is used to calculate the data that is populated into the average picked at once and demand spreadsheets (semi-static data) and thus, needs to be updated from SAP only when the script for those datasets is run. It can be downloaded from SAP by running GDC_0001 in SQ01 and specifying the columns "Delivery", "Target Qty", "Material", "Material Description", and "Batch".

"demand data.xlsx"

This spreadsheet is a complete list of all forecast data available for the next six months for all SKUs at GDC. It too is used to generate the semi-static data so it needs to be updated only when semi-static data is being updated. It can be downloaded from SAP by running APO – LISTCUBE (specify US10) and specifying the columns "APO Location / Material", "Fiscal year / Period", and "Forecast for Release - Final".

5.2.4 **Output Data**

This folder contains output results from running the main algorithm script or update script.

"total_relocation YYYY-MM-DD HH-MM-SS....xlsx"

This spreadsheet is the output when a full inventory relocation is selected when running the main relocation script. It lists the old and new layouts in separate sheets as well as a list of necessary moves to switch from the old layout to the new layout, which is ordered by default by the incremental location cost associated with moving it from one location to the other. It includes a summary sheet with a total location cost estimation and changes from the old layout to the new one.

"replace X YYYY-MM-DD HH-MM-SS....xlsx"

This spreadsheet is the output when "relocate X" is selected when running the main relocation script. It too lists old and new layouts, an ordered list of moves, and a summary sheet.

"shipment put-away YYYY-MM-DD HH-MM-SS....xlsx"

This spreadsheet is the output when "incoming shipment" is selected when running the main relocation script. It simply provides a list of the new layout as well as a shorter list of the locations of the newly placed items. There is also a summary sheet detailing the increase in cost with the new items.

"correlation top pairs YYYY-MM-DD HH-MM-SS....xlsx"

This spreadsheet is an output when the Update Semi-Static Data script is run. For this pilot, as the location costing was determined independently of the SKUs, it was not yet possible to incorporate the results of the SKU cross correlation matrix into the algorithm. The results however, were still calculated and are populated into this spreadsheet when the script is run. As the matrix contains more than a million values, only the top 100 correlations are listed in this output.

5.3 DEVELOPED SCRIPTS

At the completion of this project, two scripts, "Update Semi-Static Data.py" and "Inventory Relocation Tool.py" were developed to address the needs of reorganizing GDC to allow for more efficient order picking. The first is intended to be run periodically, to ensure that the semi-static data is up to date, while the second is the tool to be used to generate quick results for all inventory location questions (be it a shipment put-away, a full relocation, or a relocate X procedure).

5.3.1 Update Semi-Static Data

This script is best run as a background task when possible as the most recent time studies indicate that it takes on roughly 40 minutes to complete the script when running on a 2017 MacBook Pro (16GB RAM, 3.3 GHz Intel Core i5 Processor). This can be sped up by using smaller datasets (i.e. using a horizon of fewer than 6 months for the demand forecast and the historical pick data), but this is not recommended as accuracy will be lost.

The purpose of this script is to take raw data from the dynamic data folder (namely, historical data.xlsx and demand.xlsx) and process them into the two semi-static data files (demand.xlsx and avg_picked_at_once.xlsx) which can be used much more quickly by the main script. Additionally, this script calculates the SKU correlation matrix which is intended as a way to further improve the algorithm when SKU-dependent cost assignment can be introduced. As of the time of the release of this pilot, where location cost assignment is constrained to be independent of SKU, it was simply requested that the greatest elements of this matrix be output so that humans could make informed decisions.

When this script is run, it is designed to find the most recent version of the semi-static data, archive it with its timestamp and then generate new files without timestamps. The Inventory Relocation Tool script always uses the versions of the semi-static data files that are not timestamped (i.e. the newest versions).

The flow of this script is shown in Figure 7:

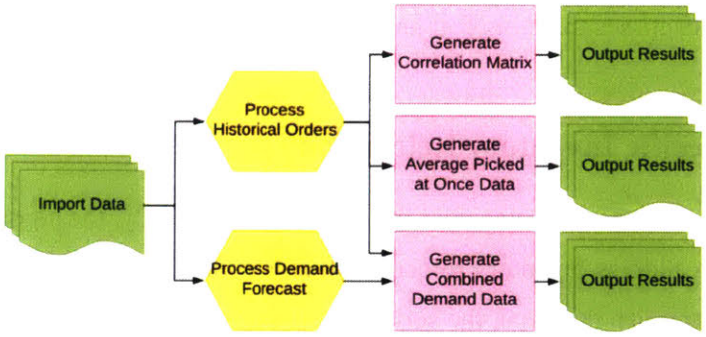

Figure 7 – Update Semi-Static Data Script Flow

5.3.2 **Inventory Relocation Tool**

This script is the main tool for inventory relocation. The current design packages all of the tools in one script and prompts the user at the very beginning to select one. The pilot was developed with a simple "Enter 0, 1, or 2" style user interface with clear directions. The purpose of this script is to take data from all of the necessary data files (static, semi-static, and dynamic) and, combined with the user's selection, generate results that can be used immediately to make storage decisions.

Time studies conducted generated an approximate 1.5-minute runtime for this tool using the same test conditions as the previous script.

When this script is run, it generates an output report that appears in the "Outputs" folder that is appropriately named and timestamped.

The flow of this script is shown in Figure 8:

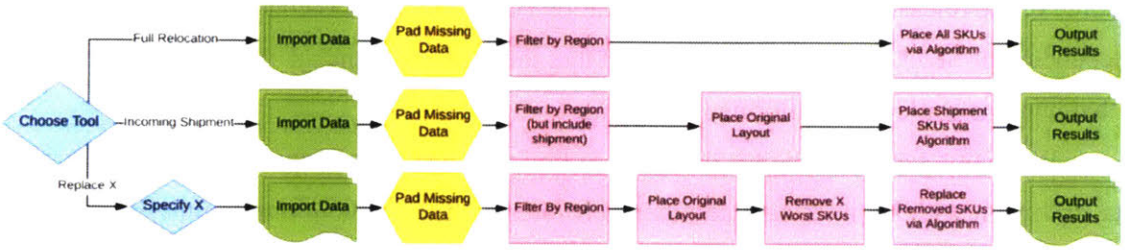

Figure 8 – Inventory Relocation Tool Script Flow

5.4 IMPLEMENTED TOOLS

This section discusses the tools that have been implement for the pilot of the inventory relocation algorithm.

5.4.1 Section Specific Inventory Relocation

As the pilot was selected to consider the high bay region only, the algorithm tool necessarily needed to address inventory relocation that is confined to certain storage regions within GDC. To accomplish this goal, the script was designed to admit a criterion to filter the GDC locations. With the release of the pilot, the script contains an "is_high_bay" criterion function that will return "True" for all locations with an aisle value in the set {39, 40, 41, 42, 43, 44} as those correspond to the high bay area. A Boolean object called "high_bay_only" is set automatically to "True" for the pilot version. Subsequent versions can include an "is_shelving" criterion function and a "shelving_only" or other Booleans that limit the scope of a relocation to include particular regions as well. For now, the full inventory relocation, relocate x, and shipment put-away can only function in the high bay region.

5.4.2 Cost of Situation Estimation

One of the basic functions implemented in the Inventory Relocation Tool script is a total cost function that calculates the total SKU location costs summed across all SKUs under

consideration (i.e. the objective function). This tool is used to calculate the present state and compare it to a proposed new state.

Additionally, there is a cost function calculated on an individual basis which is used when listing the change in cost associated with a particular parts' location-to-location move.

5.4.3 Full Inventory Relocation

This procedure can be executed by running the Inventory Relocation Tool script and selecting the option for full inventory relocation. The main purpose of this feature is to reorganize the entirety of GDC or a specific section within. It is still up to a human to decide whether to execute the moves recommended and it is important to filter out poor suggestions based on incorrect data (i.e. padded data that gives an unreasonably high or unreasonably low value for the size or location cost of a particular SKU or storage location) as well as additional considerations for storage locations (i.e. part cross correlations, primary-overflow storage strategy, routing considerations) that the algorithm is currently blind to.

As discussed previously, this tool will increase in accuracy and generate more meaningful results as the supporting data is collected and refined.

5.4.4 Replace X Relocation

This procedure can be executed by running the Inventory Relocation Tool script and selecting the option for Relocate X. Instead of placing all SKUs into the region from scratch, it takes the current situation as being already placed and then removes the X SKUs that contribute most offensively to the total cost of the situation and places them again using the placement algorithm.

Like the full inventory relocation procedure, this will benefit from increased data availability.

5.4.5 Recommended Put-Away

This procedure can be executed by running the Inventory Relocation Tool script and selecting the option for an incoming shipment. It takes the current situation as being already placed and uses the placement algorithm to recommend storage locations.

This tool will benefit a lot from increased data availability both in terms of accuracy and in terms of accessibility to the entire warehouse. As the pilot is limited to the high bay area, this

tool will only be able to select high bay storage locations for every part in the shipment, and will require humans to ignore the recommendations for parts that clearly belong in shelving areas. It will only be able to recommend put-away locations across the entire facility if the restriction for "high bay only" is lifted, and it is not recommended to do so until a critical volume of supporting data (SKU sizes, location sizes, and location cost assignment) is collected for the other regions.

5.5 FLEXIBILITY FOR FUTURE IMPROVEMENTS

This section describes the flexibilities included in the pilot implementation that were designed to enable the inventory relocation tool to be refined and improved more easily.

5.5.1 Handling of Missing Data

As it has been mentioned before, the algorithm was designed to "pad" the missing data when it does not have anything to use. This technique is common in data science applications and is necessary when steps of the algorithm require calculation or comparison using all data values within a set, and thus missing values will prevent the algorithm from proceeding. Both the static and semi-static data contain constitutive information about specific SKUs or specific locations, but none of these datasets are guaranteed to contain data on every SKU or location (in fact, none of them are even remotely close to complete). The following techniques were applied to infer the missing data points:

1. For demand and average picked at once data, it was straightforward to pad the missing data as the complete order history was being used. A SKU missing from this list indicated that it was not picked at all in the previous 6 months (or another horizon). Thus, it is valid to set the demand levels to extremely low values (and conversely, the average picked at once levels to extremely high values) which indicates that part popularity is exceptionally low.
2. For location volumes, the padding was designed to use the average value of all locations in the region plus a tiny, randomly generated, noise term in order to make them distinct from each other (it was useful when developing the algorithm to have storage location sizes that varied) for any unknown location.
3. Likewise, for location movement costs, any missing value was padded with the average of all locations in the region plus a noise term (again, distinct location costs were advantageous for verifying the algorithm was behaving properly).

4. For SKU sizes, it was determined that simply using an average of all SKUs under consideration was insufficient to pad the missing as SKUs tended to vary in size much more greatly than storage locations (some SKUs are large instruments, while others can be plastic bags containing just one mechanical fastener). Very poor results were obtained when attempting to use this approximation as the smallest of SKUs were always taken from their locations and relocated among other locations because the algorithm thought they were hundreds of times larger than they were. The volume usage calculated for the present state exceeded 5000% in some of the locations. Thus, it was determined that a much more accurate approach would be to take advantage of the negative correlation between SKU quantity in stock and SKU size. While it is far from exact, as plenty of products of the same size can exist in GDC in many different quantities, it is still the case that a large product is more likely to be more scarce in the warehouse. Large products tend to be more expensive, and expensive products tend to be bought in smaller quantities. Because of this basic relationship, the inverse percentile of the current SKU quantity was used to pad missing SKU sizes. For example, if a particular SKU was missing its size, but it currently had 200 units in stock at GDC (which is the 80^{th} percentile for SKUs currently in GDC), the size was set to be the size of the 20^{th} percentile SKU in the list of known sizes.

Ultimately, it is of the utmost importance to get complete data on the dimensions of SKU's and storage locations. It is clear that increasing the base of supporting data will increase the robustness of the tool.

However, for the purpose of real life implementation, it was decided that the tool needs to be designed to give results even when data is incomplete. It was well understood by managers at GDC that a tool that gives excellent results for complete data and increasingly worse results with less and less data would be far superior to a tool that gives excellent results for complete data and cannot process incomplete data at all. Additionally, when extending the tool to new situations, it would work immediately with just a few data points (rather than crashing) even if the results would be virtually meaningless.

5.5.2 New Storage Areas in GDC

While the pilot project was implemented with efforts focused primarily on the high bay area, the existence of the rest of GDC was considered as well, as the team knew that the next step would be to extend the work to the rest of GDC. Instead of truncating all datasets to included only SKUs in the high bay area, the tools were designed to filter them, so that extending to other areas of GDC simply requires the removal/replacement of the filter and collection of the necessary supporting data. As such, it is recommended that as soon as the tool is performing at a satisfactory level for the high bay area, the following steps can be taken to extend to other areas (given that it contains most of the SKUs and storage locations, shelving is a good candidate to go next):

1. Add rows to the SKU sizing datafile for as many shelving SKUs as possible
2. Add rows to the location sizing datafile for as many shelving locations as possible
    a. It is recommended that the padding procedure be altered so that it pads shelving locations sizes by region (i.e. setting any unknown location in the high bays to be the average of all high bay locations and likewise for shelving as opposed to using the global average for all)
3. Repeat the location cost assignment procedure for the shelving region following the scheme outlined in Batra [8].
4. Augment the code in the Inventory Relocation Tool to include necessary criterion functions for the new region of interest and adjust/create the necessary Boolean arguments. Additionally, add steps to the user input section to further separate the possible tools between GDC wide relocation and placements versus region specific relocation and placement

5.5.3 Extension to ADC and EDC

In addition to designing the tool with the intention that it be applicable to all of GDC, it was also designed to be able to be scaled to Waters' other primary distribution centers in Singapore and the Netherlands. It was primarily for this reason that the entire active waters part numbers data sheet be required – as that file contains a list of roughly 100,000 active SKUs while GDC contains only roughly 10,000 of them. Additionally, the list of GDC storage locations was used so that when this project is scaled to other facilities, a new list could simply

be substituted in. The demand data could easily be switched from GDC to the new warehouse as well.

Ultimately, the procedure for scaleup to another distribution center will likely be most appropriate once it is functioning well in the entirety of GDC. At that point the steps for extension would be similar to extending from one region of GDC to another with a few caveats:

1. Location sizing, labeling, and the resulting filtering criteria (i.e. that GDC aisles 39-44 are high bay storage – equivalent conditions would need to be developed) would need to be redone from scratch for the new facility
2. Location cost assignment would need to be redone completely for the new facility following the procedures in Batra [8] and incorporating the intricacies of that facility's standard operation procedures.
3. Most likely, a separate script would need to be made for use at the new facility. Because of this, it would be prudent to institute a periodic version control syncing exercise so that improvements and changes made by teams in one facility could be included in the script used by the other facility. This would help to homogenize the two scripts. Alternatively, the script could be separated so that there is a unified module that functions the same way for every facility (i.e. the relocation algorithm functions) and write separate script modules to contain the logic that is warehouse specific.

CHAPTER 6

SAP INTEGRATION

This chapter details the steps taken to integrate the inventory relocation project with Waters' primary enterprise software, SAP, and its data. It highlights the different options that were researched and considered and discusses the method selected for the pilot as well as the plan for the future.

6.1 OPTIONS

For integrating the inventory relocation with SAP, it is necessary that the dynamic data (in the folder labeled "SAP Data") be obtained from the SAP databases prior to running the inventory relocation tool, as that data needs to be completely up to date. Additionally, it is desired (but not absolutely required) that output results be available on SAP once they are generated.

To accomplish this, three primary avenues were identified. Each has benefits and drawbacks and these options were presented before Waters executives to consider the possibilities. The options are:

1. Offline usage of SAP data – This requires requesting data from SAP and requiring the data files to be placed into the "SAP Data" folder
2. Automatic running of the script through SAP – this involves allowing SAP to call the external Python script in the form of a query or report and automatically feeding it the data it needs
3. Reimplementing the script in a Native SAP developer tool

6.1.1 Offline

The simplest option for use of the algorithm is to run it the way it was run during its development. To accomplish this, a user would execute the necessary SAP reports with the necessary columns included and download them all in Excel format. They would all need to be placed into the "SAP Data" folder with precisely the correct filenames. At this point, the scripts could run in their offline location and generate all of the output files. These files could be manually uploaded, printed, or sent to whoever needs them.

The downsides of this method are the limitations of the local output generation and the requirement for the dedicated file location and precise file naming. It is advantageous in that of the Python script would never have access to everything on SAP, and thus there would be no issues of corrupted files or security.

6.1.2 Automatic

The more preferred option is to automatically run the script using SAP. The team researched the possibilities and resources within Waters' Business Technology (BT) organization confirmed that it is possible and relatively simple for SAP to call external software applications. To accomplish this, the scripts and the relevant directories would need to be stored in a Waters network location accessible by the SAP system. From there, a procedure would need to be created within SAP that automatically ran the necessary reports and saved them to the file location with the correct filenames. It would then call the main script and output the results. The script for updating the semi-static data could be set to run via another SAP procedure which is set to run in the background at regular intervals. This solution was determined to be most ideal as the results can be obtained directly within the SAP system and the integration would be truly seamless.

The only drawback of such a solution is the series of bureaucratic hurdles associated with validating Python as permitted third-party software for Waters SAP system. As the SAP system is used across the entire company in various ways, it would be catastrophic if third-party software were to introduce security problems. As such, Waters has channels in place to validate software for integration with SAP, but that process is nontrivial. Ultimately it was determined that this process is most likely worth the effort given the benefits of direct integration.

6.1.3 Reimplementation

Another alternative is to reimplement the tools within native SAP HANA, SAP's native development platform. The code in the Python scripts would need to be interpreted and then reimplemented on SAP's platform. This would to solve the problems of interfacing external software, as well enable direct integration with the SAP system. In fact, it would not require much work to have output results available directly through SAP due to the nativity of the system. Unfortunately, this possibility is not recommended as it would require translation of the system between coding languages. Such an endeavor is always time consuming and it can never

be guaranteed that all of the functionalities available in one language are available in another. And even functionalities that are available are not necessarily equivalently efficient.

6.2 CURRENT SOLUTION

Unfortunately, the availability of resources within the BT group at Waters was not favorable for implementing the automatic solutions within the timelines of this pilot. Additionally, while SAP does provide an interface environment for using Python, Waters' team did not have it installed. For these reasons, the decision was made to implement the offline solution. The tools and necessary data files are to be stored locally on a PC at GDC.

6.3 PLAN FOR MOVING FORWARD

While the offline solution was selected for the implementation of this pilot, the goal of integrating the tools directly with SAP was not abandoned. The plan for integration involves initial implementation of the offline solution while designating the direct integration as an "enhancement request" within the BT organization, to be completed at a later date when BT resources are more available. Specifically, the plan requires:

1. Copying the SAP info set to protect data from changes that could break the scripts (i.e. changing field names, etc.)
2. Initiating SAP Procedure Integration (SAP PI) to create files that are automatically dropped into locations.
3. Generating reports that can automatically call the Python scripts

CHAPTER 7

RESULTS AND DISCUSSION

This chapter details the results of the implementation and discusses the expectations for performance, and time savings.

7.1 IMPLEMENTATION ON HIGH BAY AREAS

As discussed previously, this pilot project dealt exclusively with the high bay areas. The high bay area at GDC includes approximately 1,000 storage locations and 1,000 distinct SKUs. The script was completed and outfitted with sufficient data to relocate items in the high bay area with reasonable accuracy. Results were discussed with managers at GDC who have excellent knowledge of the Waters products and they were satisfied with the performance. The high bay area was selected due to the presence of extremely large and unpopular SKUs and due to the largest amount of existing supporting data. The tool will be used in the high bay area on a trial basis, with humans overseeing the recommendations and overriding suggestions that are known to be erroneous. Replacing smaller number of SKUs at a time will allow the staff to easily gage the performance of the script and observe the time savings.

7.2 RELOCATION SIMULATION

For the purpose of analysis, simulations were run using statically downloaded versions of the dynamic, SAP data, obtained in July 2018. The script was run with this data using the full relocation tool so that results could be observed. The most obvious results were SKUs that were poorly placed due to the system overestimating or underestimating their sizes (for ones without size data). These errors were immediately obvious to GDC staff members who were familiar with different Waters SKUs. However, there were a fairly large number of SKUs that were replaced into better locations – both popular SKUs getting more accessible locations and unpopular SKUs being relegated to the least accessible locations. These represented very positive results, as it made suggestions that were validated by knowledgeable staff members as good ideas were not previously considered.

The summary section of the simulation revealed a very stark result. According to the analysis, the objective function had dropped by 40.6% from the old layout to the new layout.

This indicated that the total of all of the SKU-location costing values was nearly cut in half. While it is important not to jump to conclusions and get convinced that the picking time will truly be halved, this is still a very encouraging result.

7.2.1 Time Savings Analysis

One of the primary questions to address about the time savings is how a reduction in the objective function translates to time savings in real life. The objective function simply sums the amount of time it would take over six months to remove items off the shelves at their expected demand rate, one typical trip at a time (i.e. removing the average order quantity each time). Obviously, the actual amount of time spent picking is much less than this as that the total value of the objective function translates to millions of days over the six-month period. These numbers are inflated most due to including the quantity in a location as part of the cost, considering the cost as if only one SKU were picked at a time, and measuring the picking time based on worst case scenarios.

Theoretically, if the objective function was simply a linear multiple of the time spent picking, it would follow that the savings of over 50% are in fact valid, as the ratio of the new objective function to the old objective function would hold. However, there are multiple reasons to assume that this is not the case:

1. The additional efficiencies from the primary-overflow storage strategy, SKU cross-correlations are not considered. These are ways in which pickers increase their picking speed for a given layout, and, new layouts that don't consider have the possibility to ruin the efficiencies (in the case of not considering primary-overflow storage) or simply not improve them (in the case of cross correlations). Because of these, the overall time savings will be less than the savings in the objective function
2. Perhaps even more important is the effect of batch picking, as staff are likely to pick multiple SKUs at once, even when presented with one-line orders. Because of this, they pick an average of 15 lines per trip, (as estimated by material handlers). The savings is significantly shortened for the 15-SKU route compared to the sum of the savings for each SKU separately. To illustrate this, a simulation was conducted.

If one considers SKUs of the same size and popularity, then a >50% reduction in the objective function corresponds approximately to a 50% decrease in the average expected

distance of SKUs from the shipping area. However, if 15 SKUs are picked at once, the total savings is significantly lower, especially as part of the path is common anyway.

To model this behavior, a simulation was constructed that selects points randomly in $\mathbb{R}^3$ that lie in the cubic region between (0,0,0) and (10,10,10). The origin point (0,0,0) represents the entrance to the high bay area, and the cubic region represents three dimensions of location variability within the high bay area. As distances between storage locations were unknown, this simulation simply used the Euclidean distances between the points.

To model the initial state, 15 points were generated with a uniform distribution across the region, thus the value 5 was the mean value for each of the x, y, and z coordinates and $5\sqrt{3}$ the mean distance from the origin. The Eulerian path between the 16 points (including the entrance node at the origin) was determined using the Christofides algorithm (which gives a 3/2 approximation to the best possible route) [15].

The final state was modeled using the same procedure, except instead of uniformly distributed points, the points were selected in a manner by which each coordinate was selected from a trapezoidal distribution in the interval [0,10] but with a mean set to be $5(1 - imp)$ where imp corresponds to the savings in the objective function (i.e., for the 56.6% savings found in one simulation, the mean was set at 2.17 instead of 5). The simulation was run 1,000 times in each scenario and the path lengths were compared. The following results were obtained:

Table 10 – Time Savings Analysis Simulation Results

Average Picked at Once	Percent Improvement in Objective Function	Percent Improvement in Path Length
10	53%	28%
10	40%	25%
15	53%	25%
15	40%	23%

This analysis is obviously very idealized as it considered a nice Euclidean space in $\mathbb{R}^3$. It does demonstrate the principle that improvements in the average distance of all SKUs needed will result in a significantly *smaller* improvement in the path length.

Furthermore, it can be assumed that the actual savings in real time might be even less than those in this idealized simulation. This is because the actual distances from storage locations to storage locations are greater than the length of the line connecting the locations. For example,

to travel from a level five storage bay to another level five storage bay, one would have to descend to the ground level, maneuver the vehicle to the new location and then ascend back up to level five. Based on the distance of the two level five storage locations that are farthest from each other, a safety factor of 2 was established to account for this situation, as the distances in this more complex space lead to less savings in the whole route.

Considering the other limitations due to errors from insufficient data, another safety factor of 2 was introduced (i.e. the estimate was made conservatively on account of the errors made due insufficient data). It was determined empirically that approximately half of the recommendations made in the simulation where known to be poor or impossible suggestions.

Combining these, the actual time savings was estimated to be approximately **1/8th** of that of the objective function savings (the ratio of the path length improvement to the objective function improvement in Table 10 is approximately 1:2, combined with two additional safety factors of 2). Thus, the 40% savings in objective function is projected to correspond to approximately 5% savings in actual time.

CHAPTER 8

CONCLUSIONS

This chapter summarizes both aspects of this thesis: the Strategic Roadmap and the Inventory Relocation Project.

8.1 STRATEGIC ROADMAP

The roadmap represents the work done during the first 3-4 weeks of the project timeline. During this time, the team met with various stakeholders in Waters' ranks to determine the areas that needed the most attention. Key themes were identified to provide direction for improvements at Waters over the next decade. The results were presented to Waters executives multiple times and project descriptions were prepared for each idea. These project descriptions appear in CHAPTER 3 of this thesis, Batra [8] and Harlalka [9] and have also been submitted as a unified corporate document to Waters management. It is anticipated that this roadmap will be used to help direct future teams in the continuing relationship between Waters and MIT.

8.2 INVENTORY RELOCATION PROJECT

The team developed an algorithm that relocates products in the high bay area of Waters' Global Distribution Center, despite incomplete supporting data. Tools were developed to relocate entire regions, replace a selected number of stock keeping units, or to put away a received shipment. The tools were developed in Python, but managers can utilize the tools without any prior experience with Python code. The system was designed so that improvements to the data can be entered directly to the datasheets without having to alter the code.

The tools were designed to be flexible, so that extensions to the rest of GDC and to the other DCs as well as improvements can be implemented easily and without unnecessary complications.

Testing and simulations showed a possible 40% decrease to the objective function for the high bay area (with sizing data for 400 out of 1,000 SKUs) when the entire area was reorganized. Analysis and estimation showed this to correspond to an approximate 5% reduction in actual picking time (though this result is intended to be checked and verified in real time as relocations

are executed). It is recommended that Waters first implement the relocation system in the high bay area on a trial basis, to assess its performance and work out kinks.

Additionally, collecting more supporting data was shown to be of critical importance. In fact, many personnel at GDC and across Waters agree that compiling a complete list of SKU dimensions will bring positive value to Waters in other ways (e.g. estimating freight costs), so it is not difficult to justify spending resources collecting them.

The next step would be to scale up to the rest of GDC, followed by the other DCs.

CHAPTER 9

FUTURE WORK

This chapter discusses the future direction for the work contained in this thesis. It details both extensions and improvements that can be made for the Inventory Relocation Project.

9.1 PROJECT EXTENSIONS

There section documents the many ways to extend the work done in the Inventory Relocation Project. They first three sections represent a scaleup plan and are presented in the order of recommended implementation.

9.1.1 Extend to Rest of GDC

As the pilot described in this thesis considers only the high bay area in GDC, the first step would be to extend the pilot to the rest of GDC. This would involve collecting data and updating the script as described in section 5.5.2 and extending the pilot to every other area in GDC. This would allow staff at GDC to consider inventory relocations with a bigger picture in mind. It would enable relocation choices that actually involve moving items stored in shelving areas to high bay areas and vice versa. It would also enable the recommended put-away tool to actually access the entire warehouse and recommend locations throughout the warehouse. The performance would continue to improve as more data would be collected and eventually the entire warehouse could operate on automated location recommendations.

9.1.2 Interface Recommended Shipment Locations with RFID Technology

One of the important aspects of technology interfacing is the automatic use of the recommended put-away function upon receiving a shipment. This can be done the moment a shipment is dragged through the dock door portal that is implemented as part of the RFID pilot as described in Harlalka [9]. Work would need to be done to establish the interface between the recommended location script and the SAP transactions associated with receiving so that a list of storage locations would be generated automatically.

9.1.3 Extend to EDC and ADC

After extending to the entirety GDC, the inventory relocation pilot could be extended to Waters' other major distribution centers, EDC and ADC. The procedure would be similar to the full GDC scaleup, involving collection of data and cost mapping analyses as described in section 5.5.3 . Eventually, Waters can operate all of its main distribution centers on automated location recommendations.

9.1.4 Recommended Relocation During Picking/Receiving

Another possible improvement is to generate a procedure that would automatically recommend location changes to pickers that could executed during their picking routes. As a large portion of the cost of accessing certain storage location (especially those in high bays) is associated with the use and stowing of vehicles, additional efficiencies can be gained by recommending SKU relocations to be done by material handlers already engaged in a task involving those vehicles. For example, one approach would be to store SKU relocations in a queue and recommend them to material handlers any time their current tasks take them into a location/vehicle that makes executing that relocation sufficiently easy. This would render the actual execution of relocation tasks more efficient as material handlers could accomplish them while completing other tasks instead of requiring a block of time to work on inventory relocation.

9.1.5 Use Code Skeleton to Inform Other Optimization Problems

Another extension of this project is to use some of the developed background procedures as starting points for other optimization problems. Many projects Waters can undertake, both those researched by the 2018 MIT team and others, involve downloading SAP data from various places, processing it, and calculating output results for the purpose of improving internal processes. Some of the framework and initialization code developed in this project is potentially partially or completely reusable for these purposes.

9.2 ALGORITHM IMPROVEMENT

This section details ways the placement algorithm can be improved in ways beyond simply collecting more supporting data. It should still be noted that collecting additional data is

still the simplest and most effective way to improve the accuracy of the algorithm, so efforts to obtain this data should be prioritized ahead of attempting to improve the algorithm.

9.2.1 Relax Constraint of SKU Specific Location Costing

The most important assumption to reconsider is the assumption that all SKUs have the same cost in a particular location regardless of size, quantity, or SKU type (whether or not it has pallet handles). This assumption was necessary to limit the scope of the location cost assignments by requiring only measurements from locations to the shipping area ($\sim O(n)$ instead of $\sim O(n^2)$). It was additionally crucial in allowing the simple algorithm step defined in section 4.4 to be optimal. However, relaxing this constraint would enable the algorithm to increase in accuracy in multiple ways if the resulting more complex optimization problem were to be solved. It would enable the algorithm to consider location costing that varies based on SKU instead of requiring the additional assumptions made in that analysis. It would also allow improvements described in the next few sections.

9.2.2 Refinement of Cost Analysis

The first way to improve the accuracy of the algorithm is to improve the calculation of the cost assignments. Simplifying assumptions were made in order to make the task of calculating the cost assignments feasible to be accomplished in the timeframe of this project. Some ways to improve the calculation methods are described in Batra [8].

9.2.3 Refinement of Objective Function

Likewise, the objective function itself can be refined to improve the accuracy of the algorithm. The development of the objective function in section 4.3 employs a variety of assumptions to generate a simple objective function. Some of these can be relaxed in order to get a slightly more robust solution. For example, the objective can be expanded to include the cost incurred by the receiving team. The objective function can also be altered to include the positive effects of primary-overflow storage and SKU cross correlations. Work would need to be done to analyze the tradeoff between the additional accuracy and the added complexity

9.2.4 Improve Placement Algorithm

One of the main ways to improve the system is by improving the algorithm itself. The proof presented in section 4.4.1 suggests that the algorithm is truly optimal in the case where volumes are continuous and SKUs can be cut into pieces. However, the edge effects described can be studied at a more in-depth level to determine a more effective algorithm that perhaps looks at more than one SKU at a time.

Additionally, if the objective function is altered to include receiving costs, primary-overflow storage, and SKU cross correlations, the algorithm would need to be updated to handle these as well.

9.2.5 Include Routing Algorithm

In the current version of the algorithm, the assumption is made that the actual cost would be optimized by optimizing the objective function regardless of the routes taken. However, if the full n x n set of inter-location costs were to be determined, the effects of the routing could be properly considered. The Christofides algorithm [15] mentioned in section 7.2.1 could be used to find the actual path cost for a set of SKUs and probabilistic simulations could be run to model the different order possibilities. Analysis would need to be conducted to ensure that the additional runtime effects introduced with this improvement are worthwhile.

9.3 INTERFACE IMPROVEMENTS

This section discusses improvements that can be made on the front end of the system to make the tools more accessible and user-friendly.

9.3.1 Develop SAP-Python GUI

As described in section 5.3.2 , the pilot version of the inventory relocation tool was implemented with a simple "Enter 0, 1, or 2" style interface. This can be improved to include a slightly more visual interface. Additionally, the usability of this system would increase if a GUI were to be built directly into Waters' SAP system (i.e. the necessary tool choices could be made in the SAP environment instead of in a Python IDE).

9.3.2 **Develop Platform for Algorithm Refinement**

In the pilot version, the entirety of the algorithm exists in one script that handles everything from data downloading, to actual location calculations, to results outputting. The difficulty associated with implementing improvements to the algorithm are partially because the entire procedure is packaged into one script. This difficulty could be lessened by developing modules for the various parts of the script so that each could be improved separately without requiring as much debugging, full system control, and version control.

REFERENCES

[1] Krarup, J., Pruzan, P.M., 1983, "The simple plant location problem: Survey and synthesis," *European Journal of Operational Research,* Vol. 12, pp 36- 81.

[2] Kratica, J., Filipovic, V., Tosic, D., 1996, "Solving the Uncapacitated Warehouse Location Problem by SGA with Add-Heuristic" *XV ECPD International Conference on Material Handling and Warehousing*, University of Belgrade, Belgrade, Serbia

[3] Qin, G., Li, J., Jiang, N., Li, Q., Wang, L., 2013, "Warehouse Optimization Model Based on Genetic Algorithm," *Mathematical Problems in Engineering Volume 2013*, Article ID 619029, 6 pages

[4] Larson, T., March, H., and Kusiak, A., 1997, "A heuristic approach to warehouse layout with class-based storage," IIE Transactions. V29, April 1997 pp 337-349.

[5] Nguyen, S, 2008, "Algorithmic Approach to Warehouse Consolidation and Optimization, M.S. Thesis, California Polytechnic State University, San Luis Obispo, CA

[6] Tompkins, J., White, J., Bozer, Y., Frazelle, E., Tanchoco, J., and Trevino, J., 1996, Facilities Planning 2nd Edition, United States: John Wiley & Sons, Inc.

[7] Horvat, M., 2012, "An Approach to Order Picking Optimization in Warehouses," Diploma Thesis, University of Ljulblana, Ljulblana, Slovenia

[8] Batra, R., 2018, "Operational Analysis of an Inventory Location Optimization Algorithm and RFID Implementation in a Distribution Center" M.Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[9] Harlalka, A., 2018, "Design and Implementation of an RFID-based Dock Door System at a Distribution Center" M.Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[10] Chandar, A. S., 2014, "Optimizing Lot Sizes and Establishing Supermarkets in a MultiPart, Limited-Capacity Manufacturing System," M.Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[11] Waters Corporation, 2018 "Corporate History" [Online], Waters Corporation, Available: http://www.waters.com/waters/en_US/Corporate-History/nav.htm?locale=en_US&cid=134614712 [Accessed: 20-Jul-2018].

[12] Waters Corporation, 2018 "About Waters Corporation" [Online], Waters Corporation, Available: http://www.waters.com/waters/en_US/About-Waters-Corporation-/nav.htm?cid=134614448&locale=en_US [Accessed: 20-Jul-2018].

[13] Waters Annual Report, 2017

[14] 2017, "Waters Corporation - Investor Day Presentation" [Online]. Available: https://waterscorporation.gcs-web.com/static-files/0c8217e9-0c24-446b-8349-c8cdab7cadda. [Accessed: 03-Jul-2018]

[15] Christofides, N., 1976, "Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388", Graduate School of Industrial Administration, Carnegie Mellon University

[16] Hardy, G.H.; Littlewood, J.E.; Pólya, G.; 1952, Inequalities, Cambridge Mathematical Library (2. ed.), Cambridge: Cambridge University Press, Section 10.2, Theorem 368

[17] Puszko, G, 2014, "Efficient Scheduling to Reduce Setup Times and Increase Utilization in a Multiple - Part Manufacturing System" M.Eng. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

[18] Harris, F. W., 1990 [Reprint from 1913], "How Many Parts to Make at Once," Operations Research (Informs), 38 (6), pp. 947 – 950.

APPENDIX – PYTHON CODE

Update Semi-Static Data.py:

```
## IMPORT LIBRARIES
from datetime import datetime
startTime=datetime.now()
import numpy as np
import pandas as pd
from pandas import DataFrame as DF
import copy
import os
import os.path


#toggle between "real" and "fake"
DATASET="real"
get_correlation=True

## Preliminaries
def tic():
    return datetime.now()
ST=tic()
def toc(TIME):
    print (datetime.now()-TIME)
    return
def lookup(ID_list,ID): #use this to help deal with missing data?
    return ID_list.index(ID)
def make_positive(old):
    new=[]
    for i in range(len(old)):
        if old[i]>0:
            new+=[old[i]]
        else:
            new+=[0.0001]
    return new
def make_zero(old):
    new=[]
    for i in range(len(old)):
        if old[i]!=0.0001:
            new+=[old[i]]
        else:
            new+=[0]
    return new
def rem_nan(old,EMPTY=False): # remove 'nan' from a list
    if EMPTY==False:
        new=[x for x in old if str(x) != 'nan']
    else:
        new=[]
        for i in range(len(old)):
            if str(old[i])=='nan':
                new+=['']
            else:
                new+=[old[i]]
    return new
def rem_dup(old): # remove duplicates from a list
    new=[x for x in set(old)]
    return new
```

```
55.  def rem_neg(old): #changes negative quantities to 0
56.      new=(np.array(old).clip(min=0)).tolist()
57.      return new
58.  def str_conv(old):
59.      new=[]
60.      for i in range(0,len(old)):
61.          if isinstance(old[i],str):
62.              new+=[old[i]]
63.          elif isinstance(old[i],list):
64.              new+=[old[i]]
65.          else:
66.              new+=[str(int(old[i]))]
67.      return new
68.  def change_colons(text): # for windows, colons cannot be used in filenames
69.      return text.replace(':','-')
70.
71.
72.
73.  ## OPEN DATAFILES
74.
75.  fake=["Fake/Fake SKU List.xlsx",
76.        "Fake/Fake SKU Demand.xlsx",
77.        "Fake/Fake Historical Pick Data.xlsx"]
78.  real=["SAP Data/active part nos.xlsx",
79.        "SAP Data/demand data.xlsx",
80.        "SAP Data/historical data.xlsx",]
81.
82.  def import_data(data):
83.      SKU_list = pd.read_excel(data[0])
84.      SKU_demand = pd.read_excel(data[1])
85.      HIST_picks = pd.read_excel(data[2])
86.      return SKU_list,SKU_demand,HIST_picks
87.
88.
89.  #(SKU_list,SKU_size,SKU_qty,SKU_demand,LOC_list,LOC_size,LOC_cost,HIST_picks,current_SKU_
     LOCS,shipment_received)=import_fake()
90.  if DATASET=="real":
91.      (SKU_list,SKU_demand,HIST_picks)=import_data(real)
92.  if DATASET=="fake":
93.      (SKU_list,SKU_demand,HIST_picks)=import_data(fake)
94.
95.  ## sort constitutive data to match the same order
96.
97.  SKU_list = DF.sort_values(SKU_list,by='Material')
98.  SKU_demand = DF.sort_values(SKU_demand,by='APO Location / Material')
99.  HIST_picks = DF.sort_values(HIST_picks,by='Material')
100.
101.
102. ## GRAB INFO FROM COLUMNS
103.
104. demand00=rem_nan(SKU_demand['Forecast for Release - FINAL'].tolist())
105.
106.
107.
108. # SKU data
109.
110. mtls=str_conv(rem_nan(SKU_list['Material'].tolist()))
111. #mtls2=str_conv(rem_nan(SKU_qty['Material'].tolist()))
112. mtls3=str_conv(rem_nan(SKU_demand['APO Location / Material'].tolist()))
113. mtls3=[a[4:] for a in mtls3]
114. mtls4=str_conv(rem_nan(HIST_picks['Material'].tolist()))
```

```
all_mtls=[mtls,mtls3,mtls4] #update when changing datasets

print('Imported Data')
toc(ST)


N=len(SKU_list) # Number of SKUs - master is based on the SKU list spreadsheet

print("Total SKUs:",N)

desc=str_conv(rem_nan(SKU_list['Material Description'].tolist()))
def get_desc(mtl):
    return desc[lookup(mtls,mtl)]


mtls3_short=[]
demand0=[]
for i in range(len(mtls3)):
    if mtls3_short.count(mtls3[i])==0:
        mtls3_short+=[mtls3[i]]
        demand0+=[demand00[i]]
    else:
        demand0[mtls3_short.index(mtls3[i])]+=demand00[i]


demand=[]
# pad missing data
for i in range(0,N):
    if mtls3_short.count(mtls[i])==1:
        demand+=[demand0[lookup(mtls3_short,mtls[i])]]
    else:
#         demand+=[np.mean(demand0)+rnd.random()]
        demand+=[0]

print("Padded Missing Data")

GDC_demand_factor=0.6 # gives the ratio between typical demand forecast for GDC and actua
l shipping (accounts for shipments that don't go to customers)

num_months_forecast=len(set(SKU_demand["Fiscal year / period"]))
num_months_hist_data = 6 #requires 6 months of data, need to change this otherwise

#get demand data proportional to same time period as old data
demand=(np.array(demand)/GDC_demand_factor*num_months_forecast/num_months_hist_data).toli
st()


# Historical Data

ord_mtls=str_conv(rem_nan(HIST_picks['Material']))
if DATASET=="fake":
    ords=rem_nan(HIST_picks['Order'])
    ord_qty=rem_nan(HIST_picks['Quantity'])
if DATASET=="real":
    ords=rem_nan(HIST_picks['Delivery'])
    ord_qty=rem_nan(HIST_picks['Target Qty'])

orders = rem_dup(ords) #without duplicates
ord_mtls_short=rem_dup(ord_mtls)

## (NOT CURRENTLY USED)
```

```
174.
175. ## Appear on same order Matrix
176. #gives multiple version of matrix to choose from
177.
178. n=len(ord_mtls_short)
179.
180. a=tic()
181. NUM_ords=len(orders)
182. NUM_lines=len(ord_mtls)
183. ord_mtls_inds=np.zeros(NUM_lines).tolist()
184. for i in range(0,NUM_lines):
185.     ord_mtls_inds[i]=ord_mtls_short.index(ord_mtls[i])
186. toc(a)
187.
188. print("Processing",NUM_ords, "Orders with",NUM_lines,"Lines")
189.
190. ORDS=np.array(ords)
191.
192. if get_correlation:
193.
194.     correlation=np.zeros((n,n))
195.     for i in range(0,NUM_ords):
196.         #indices = [k for k, x in enumerate(ords) if x == orders[i]]
197.         indices = np.where(ORDS == orders[i])[0]
198.         for j in range(0,len(indices)):
199.             for k in range(0,len(indices)):
200.                 correlation[ord_mtls_inds[indices[j]],ord_mtls_inds[indices[k]]]+=1 #i,j
    entry gives number of times i,j were ordered together (diagonal entries are just number o
    f times ordered)
201.
202.
203.     corr1=correlation/NUM_ords #gives fraction of orders
204.     corr2=correlation/NUM_lines #gives fraction of lines
205.     toc(a)
206.     correlation_norm=np.zeros((n,n)) #normalizes wrt popularity of each product
207.     a=tic()
208.     for i in range(0,n):
209.         for j in range(0,n):
210.             if (correlation[i,i]+correlation[j,j]-correlation[i,j])==0:
211.                 correlation_norm[i,j]=0
212.             else:
213.                 correlation_norm[i,j]=correlation[i,j]/(correlation[i,i]+correlation[j,j]
    -
    correlation[i,j]) #number of times pair is ordered divided by the number of times either
    is ordered
214.     toc(a)
215.     interaction_only=copy.deepcopy(correlation)
216.     for i in range(0,n):
217.         interaction_only[i,i]=0
218.     interaction_only_norm=np.zeros((n,n))
219.     for i in range(0,n):
220.         for j in range(0,n):
221.             if i!=j:
222.                 interaction_only_norm[i,j]=correlation_norm[i,j]
223.
224.     print("Generated Correlation Matrices")
225.     toc(ST)
226.
227. def list_correlation(interaction_only,ITER=100):
228.     C=copy.deepcopy(interaction_only)
229.     highest=[]
```

```
    for i in range(0,ITER):
        max_corr=np.max(np.ndarray.flatten(C))
        max_inds=np.where(C==max_corr)
        mtl1=ord_mtls_short[max_inds[0][0]]
        mtl2=ord_mtls_short[max_inds[1][0]]
        highest+=[[mtl1,get_desc(mtl1),mtl2,get_desc(mtl2),max_corr]]
        C[max_inds[0][0],max_inds[1][0]]=C[max_inds[1][0],max_inds[0][0]]=0
    return highest


A=list_correlation(interaction_only,ITER=100)

print("Listed Correlation Top Pairs")
toc(ST)

## Number of times ordered

times_ordered=np.zeros(N)
for i in range(0,NUM_ords):
    indices = [k for k, x in enumerate(ords) if x == orders[i]]
    for j in range(0,len(indices)):
        times_ordered[ord_mtls_inds[indices[j]]]+=1

## Average number picked at once

total_picked=np.zeros(N)
for i in range(0,NUM_lines):
    total_picked[ord_mtls_inds[i]]+=ord_qty[i]
#avg_picked_at_once=total_picked/times_ordered
avg_picked_at_once=np.zeros(N)
for i in range(N):
    if times_ordered[i]==0:
        avg_picked_at_once[i]=2000
    else:
        avg_picked_at_once[i]=total_picked[i]/times_ordered[i]

print("Generated avg_picked_at_once Data")


def soft_mean(a,b,weight=.5):
    if a<=0:
        mean= b
    elif b<=0:
        mean= a
    else:
        mean= a*weight+b*(1-weight)
    if mean<=0:
        return 0
    else: return mean

demand_final=[]
for i in range(0,N):
    demand_final+=[soft_mean(total_picked[i],demand[i],weight=0.6)] #weight of 0.7 means
70% weighting on last six months, 30% on forecast
demand_final=make_positive(demand_final)

print("Generated Combined Demand Data")


if os.path.isfile('Semi Static/avg_picked_at_once.xlsx'):
```

```
289.      old_date=pd.read_excel('Semi Static/avg_picked_at_once.xlsx',sheet_name='DATE')['DATE
    '][0].replace(':','-')
290.      new_name='Semi Static/Backup avg_picked_at_once '+old_date+'.xlsx'
291.      os.rename('Semi Static/avg_picked_at_once.xlsx',new_name)
292.      print("Archived:", new_name)
293.  df=pd.DataFrame(np.array([mtls,avg_picked_at_once]).T)
294.  df.columns = ['Material','avg_picked_at_once']
295.  meta= pd.DataFrame([[str(datetime.now())]])
296.  meta.columns = ['DATE']
297.
298.  writer = pd.ExcelWriter('Semi Static/avg_picked_at_once.xlsx')
299.  df.to_excel(writer,'Sheet 1',index=False)
300.  meta.to_excel(writer,'DATE',index=False)
301.  meta.columns = ['DATE']
302.  writer.save() #SAVE TO EXCEL
303.
304.
305.  print("Saved File: Semi Static/avg_picked_at_once.xlsx")
306.
307.  if os.path.isfile('Semi Static/demand.xlsx'):
308.      old_date2=pd.read_excel('Semi Static/demand.xlsx',sheet_name='DATE')['DATE'][0].repla
    ce(':','-')
309.      new_name2='Semi Static/Backup demand '+old_date2+'.xlsx'
310.      os.rename('Semi Static/demand.xlsx',new_name2)
311.      print("Archived:", new_name2)
312.  df2=pd.DataFrame(np.array([mtls,demand_final]).T)
313.  df2.columns = ['Material','Demand']
314.  #meta= pd.DataFrame([[str(datetime.now())]])
315.  #meta.columns = ['DATE']
316.
317.  writer = pd.ExcelWriter('Semi Static/demand.xlsx')
318.  df2.to_excel(writer,'Sheet 1',index=False)
319.  meta.to_excel(writer,'DATE',index=False)
320.  meta.columns = ['DATE']
321.  writer.save() #SAVE TO EXCEL
322.
323.  print("Saved File: Semi Static/demand.xlsx")
324.
325.  filename=change_colons('Outputs/correlation top pairs '+str(datetime.now())+'.xlsx')
326.  writer = pd.ExcelWriter(filename)
327.  export=DF(A)
328.  export.columns = ['Material 1','Description 1','Material 2','Description 2','Number of Ti
    mes Bought Together']
329.  export.to_excel(writer,'List of Moves',index=False)
330.  writer.save() #SAVE TO EXCEL
331.
332.  print("Saved File:",filename)
333.  toc(ST)
```

Inventory Relocation Tool.py:

```
## IMPORT LIBRARIES
from datetime import datetime
ST=datetime.now()
import numpy as np
import pandas as pd
from pandas import DataFrame as DF
import random as rnd
import copy
from scipy import stats

print("For Full Inventory Relocation, Enter '0'")
print("For A [ Procedure, Enter '1'")
print("For Incoming Shipment, Enter '2'")
choice = int(input("Choose an option:\n"))
full_relocation=replace=shipment=False
if type(choice)==int:
    if (choice!=0) & (choice!=1) & (choice!=2):
        raise ValueError("Need to enter 0, 1, or 2")
else:
    raise ValueError("Need to enter 0, 1, or 2")
if choice==0:
    full_relocation=True
    print("Relocating All, please wait:")
if choice==1:
        replace=True
        X=int(input("Enter an integer representing the number of SKUs to relocate:\n"))
        print("Relocating",X, "SKUs, please wait:")
if choice==2:
    shipment=True
    print("Placing Shipment, please wait:")


#toggle between "real" and "fake"
DATASET="real"
high_bay_only=True

## Preliminaries
def tic():
    return datetime.now()
ST=tic()
def toc(TIME):
    print (datetime.now()-TIME)
    return
def prune_zeros(QTY,others): #list form
    QTY_new=[]
    others_new=[]
    for i in range(0,len(others)):
        others_new+=[[]]
    for i in range(0,len(QTY)):
        if QTY[i]!=0:
            QTY_new+=[QTY[i]]
            for j in range(0,len(others)):
                others_new[j]+=[others[j][i]]
    N=len(QTY_new)
    return QTY_new,others_new,N

def prune_unrec(MASTER,LIST,others):
    others_new=[]
```

```
    LIST_new=[]
    for i in range(0,len(others)):
        others_new+=[[]]
    for i in range(0,len(LIST)):
        if MASTER.count(LIST[i])>0:
            LIST_new+=[LIST[i]]
            for j in range(0,len(others)):
                others_new[j]+=[others[j][i]]
    return LIST_new,others_new
def combine(LIST,QTY):
    LIST_new=[]
    QTY_new=[]
    for i in range(len(LIST)):
        if LIST_new.count(LIST[i])==0:
            LIST_new+=[LIST[i]]
            QTY_new+=[QTY[i]]
        else:
            ind=lookup(LIST_new,LIST[i])
            QTY_new[ind]+=QTY[i]
    return LIST_new,QTY_new
def make_positive(old):
    new=[]
    for i in range(len(old)):
        if old[i]>0:
            new+=[old[i]]
        else:
            new+=[0.0001]
    return new
def prune_by_criteria(criteria,LIST,others):
    others_new=[]
    LIST_new=[]
    for i in range(0,len(others)):
        others_new+=[[]]
    for i in range(0,len(LIST)):
        if criteria(LIST[i]):
            LIST_new+=[LIST[i]]
            for j in range(0,len(others)):
                others_new[j]+=[others[j][i]]
    return LIST_new,others_new
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False
def is_high_bay(location):
    a=location[0:2]
    return (a=='39')|(a=='40')|(a=='41')|(a=='42')|(a=='43')|(a=='44')

def is_shelving(location):
    a=location[0:2]
    if is_number(a):
        return (int(a)>=1) & (int(a)<=20)
    else:
        return False
def iden(x):
    return x
def perc(x):
    p=[]
    for i in range(len(x)):
        p+=[stats.percentileofscore(x,x[i])]
```

```
120.     return np.array(p)
121. def shift_z(x):
122.     p=[]
123. #    for i in range(len(x)):
124. #        p+=[stats.zscore(x,x[i])+10]
125.     p=stats.zscore(x)+3
126.     return np.array(p)
127. def ALL(x):
128.     return True
129. def lookup(ID_list,ID): #use this to help deal with missing data?
130.     return ID_list.index(ID)
131.
132. def rem_nan(old,EMPTY=False): # remove 'nan' from a list
133.     if EMPTY==False:
134.         new=[x for x in old if str(x) != 'nan']
135.     else:
136.         new=[]
137.         for i in range(len(old)):
138.             if str(old[i])=='nan':
139.                 new+=['']
140.             else:
141.                 new+=[old[i]]
142.     return new
143. def rem_dup(old): # remove duplicates from a list
144.     new=[x for x in set(old)]
145.     return new
146. def rem_neg(old): #changes negative quantities to 0
147.     new=(np.array(old).clip(min=0)).tolist()
148.     return new
149. def str_conv(old):
150.     new=[]
151.     for i in range(0,len(old)):
152.         if isinstance(old[i],str):
153.             new+=[old[i]]
154.         elif isinstance(old[i],list):
155.             new+=[old[i]]
156.         else:
157.             new+=[str(int(old[i]))]
158.     return new
159.
160. def listmax(List): #for two level lists
161.     Max=-100000000
162.     for i in range(0,len(List)):
163.         for j in range(0,len(List[i])):
164.             element=List[i][j]
165.             if element>Max:
166.                 Max=element+0
167.                 I=i+0
168.                 J=j+0
169.     return Max,I,J
170. def listsum(List): #for two level lists
171.     Sum=0
172.     for i in range(0,len(List)):
173.         for j in range(0,len(List[i])):
174.             Sum+=List[i][j]
175.     return Sum
176. def unique_list(old):
177.     new=[]
178.     for i in range(len(old)):
179.         if new.count(old[i])==0:
180.             new+=[old[i]]
```

```
181.     return new
182. def total_vol_used(currently_there):
183.     vol=0
184.     for i in range(len(currently_there)):
185.         for j in range(len(currently_there[i])):
186.             vol+=currently_there[i][j][3]
187.     return vol
188. def count(currently_there):
189.     COUNT=0
190.     for i in range(len(currently_there)):
191.         for j in range(len(currently_there[i])):
192.             COUNT+=1
193.     return COUNT
194. def checksum(currently_there):
195.     Sum=0
196.     for i in range(len(currently_there)):
197.         for j in range(len(currently_there[i])):
198.             Sum+=(i)/(5+i)*(currently_there[i][j][1]+currently_there[i][j][2])
199.     return Sum
200. def change_colons(text): # for windows, colons cannot be used in filenames
201.     return text.replace(':','-')
202.
203. ## OPEN DATAFILES
204.
205. fake=["Fake/Fake SKU List.xlsx",
206.       "Fake/Fake SKU Sizes.xlsx",
207.       "Fake/Fake SKU Quantities.xlsx", #Eliminated, can use for new region?
208.       "Fake/Fake SKU Demand.xlsx",
209.       "Fake/Fake Storage Location List.xlsx",
210.       "Fake/Fake Storage Location Sizes.xlsx",
211.       "Fake/Fake Storage Location Costs.xlsx",
212.       "Fake/Fake Historical Pick Data.xlsx",
213.       "Fake/Fake SKU Current Locations.xlsx",
214.       "Fake/Fake Shipment Received.xlsx"]
215. real=["SAP Data/active part nos.xlsx",
216.       "Static/SKU sizes.xlsx",
217.       "Fake/Fake SKU Quantities.xlsx", #Eliminated
218.       #"Fake/Fake SKU Demand.xlsx",
219.       "Semi Static/demand.xlsx",
220.       "SAP Data/All storage locations.xlsx",
221.       "Static/Location Sizes.xlsx",
222.       "Static/Cost Mapping.xlsx",
223.       #"Fake/Fake Historical Pick Data.xlsx",
224.       #"historical data.xlsx",
225.       "Semi Static/avg_picked_at_once.xlsx",
226.       "SAP Data/current storage.xlsx",
227.       "SAP Data/Shipment Received.xlsx"]
228.
229. SAP_names=["active part nos.xlsx",
230.       "",
231.       "",
232.       "Fake SKU Demand.xlsx",
233.       "All storage locations.xlsx",
234.       "",
235.       "",
236.       "Fake Historical Pick Data.xlsx",
237.       "current storage.xlsx",
238.       "Fake Shipment Received.xlsx"]
239.
240. def import_data(data):
241.     SKU_list = pd.read_excel(data[0])
```

```
    SKU_size = pd.read_excel(data[1])
    #SKU_qty = pd.read_excel(data[2]) # Eliminated
    SKU_demand = pd.read_excel(data[3])
    LOC_list = pd.read_excel(data[4])
    LOC_size = pd.read_excel(data[5])
    LOC_cost = pd.read_excel(data[6])
    HIST_picks = pd.read_excel(data[7])
    current_SKU_LOCS = pd.read_excel(data[8])
    shipment_received = pd.read_excel(data[9])
    return SKU_list,SKU_size,SKU_demand,LOC_list,LOC_size,LOC_cost,HIST_picks,current_SKU_LOCS,shipment_received


#(SKU_list,SKU_size,SKU_qty,SKU_demand,LOC_list,LOC_size,LOC_cost,HIST_picks,current_SKU_LOCS,shipment_received)=import_fake()
if DATASET=="real":
    (SKU_list,SKU_size,SKU_demand,LOC_list,LOC_size,LOC_cost,HIST_picks,current_SKU_LOCS,shipment_received)=import_data(real)
if DATASET=="fake":
    (SKU_list,SKU_size,SKU_demand,LOC_list,LOC_size,LOC_cost,HIST_picks,current_SKU_LOCS,shipment_received)=import_data(fake)

print('Imported Data')
toc(ST)

## sort constitutive data to match the same order

SKU_list = DF.sort_values(SKU_list,by='Material')
#SKU_size = DF.sort_values(SKU_size,by='Material') #not necessary anymore
#SKU_qty = DF.sort_values(SKU_qty,by='Material') #not necessary anymore
SKU_demand = DF.sort_values(SKU_demand,by='Material')

if DATASET=="real":
    LOC_list = DF.sort_values(LOC_list,by='Storage Bin')
    LOC_cost = DF.sort_values(LOC_cost,by='Storage Bin')
if DATASET=="fake":
    LOC_list = DF.sort_values(LOC_list,by='Location')
    LOC_cost = DF.sort_values(LOC_cost,by='Location')
#LOC_size = DF.sort_values(LOC_size,by='Location')

HIST_picks = DF.sort_values(HIST_picks,by='Material')

## GRAB INFO FROM COLUMNS


# SKU data

mtls=str_conv(rem_nan(SKU_list['Material'].tolist()))
mtls1=str_conv(rem_nan(SKU_size['Material'].tolist()))
#mtls2=str_conv(rem_nan(SKU_qty['Material'].tolist()))
mtls3=str_conv(rem_nan(SKU_demand['Material'].tolist()))
mtls4=str_conv(rem_nan(HIST_picks['Material'].tolist()))
all_mtls=[mtls,mtls1#,mtls2
          ,mtls3,mtls4] #update when changing datasets
SKU_vols0=make_positive(rem_nan(SKU_size['Volume'].tolist()))
#num_total0=rem_nan(SKU_qty['Quantity'].tolist())
if DATASET=="real":
    demand0=rem_nan(SKU_demand['Demand'].tolist())
if DATASET=="fake":
    demand0=rem_nan(SKU_demand['Quantity'].tolist())

```

```
N=len(SKU_list) # Number of SKUs - master is based on the SKU list spreadsheet

print("Total SKUs:",N)

# Locations Data

if DATASET=="fake":
    locs=rem_dup(str_conv(rem_nan(LOC_list['Location'].tolist())))
    locs1=str_conv(rem_nan(LOC_size['Location'].tolist()))
    locs2=str_conv(rem_nan(LOC_cost['Location'].tolist()))
if DATASET=="real":
    locs=rem_dup(str_conv(rem_nan(LOC_list['Storage Bin'].tolist())))
    locs1=str_conv(rem_nan(LOC_size['Storage Bin'].tolist()))
    locs2=str_conv(rem_nan(LOC_cost['Storage Bin'].tolist()))

all_locs=[locs,locs1,locs2]

loc_vols0=rem_nan(LOC_size['Volume'].tolist())
costs0=rem_nan(LOC_cost['Cost'].tolist())

M=len(locs) #Number of storage locations. Need to confirm they match in data files
for i in range(1,len(all_locs)):
    M0=len(set(locs+all_locs[i]))
    if M0>N:
        raise ValueError("LOCS's Exist in Dataset "+str(i)+" that aren't in the Master List")

print("Total Storage Locations:",M)

# pad missing location data

loc_vols=[]
costs=[]

for i in range(0,M):
    if locs1.count(locs[i])==1:
        loc_vols+=[loc_vols0[lookup(locs1,locs[i])]]
    else:
        loc_vols+=[np.mean(loc_vols0)+rnd.random()]
for i in range(0,M):
    if locs2.count(locs[i])==1:
        costs+=[costs0[lookup(locs2,locs[i])]]
    else:
        costs+=[np.mean(costs0)+rnd.random()/100]



# Current Location Data

curr_mtls=str_conv(current_SKU_LOCS['Material'].tolist())
if DATASET=="real":
    curr_locs=current_SKU_LOCS['Stor.bin'].tolist()
    curr_qty=rem_neg(current_SKU_LOCS['Total stock'].tolist())
    curr_batch=str_conv(rem_nan(current_SKU_LOCS['Batch'].tolist(),EMPTY=True))
if DATASET=="fake":
    curr_locs=current_SKU_LOCS['Stor.bin'].tolist()
    curr_qty=rem_neg(current_SKU_LOCS['Total stock'].tolist())

def mtls_with_batches(mtls,batchs):
    mtls_new=[]
```

```
    for i in range(len(mtls)):
        mtls_new+=[[mtls[i],batchs[i]]]
    return mtls_new


curr_mtls_all=copy.deepcopy(curr_mtls)
curr_qty_all=copy.deepcopy(curr_qty)
curr_mtls_all,curr_qty_all=combine(curr_mtls_all,curr_qty_all)
#Removes entries with unknown location

(curr_locs,[curr_mtls,curr_qty,curr_batch])=prune_unrec(locs,curr_locs,[curr_mtls,curr_qty,curr_batch])


if high_bay_only:
    print("Considering High Bay Area Only")
    #Remove entries with location not in high bay
    (curr_locs,[curr_mtls,curr_qty,curr_batch])=prune_by_criteria(is_high_bay,curr_locs,[curr_mtls,curr_qty,curr_batch])
    #Remove locations not in high bay
    (locs,[costs,loc_vols])=prune_by_criteria(is_high_bay,locs,[costs,loc_vols])

curr_mtls_with_batchs=mtls_with_batches(curr_mtls,curr_batch)

m=len(locs)
# pad missing SKU data

SKU_vols=[]
num_total=[]
demand=[]


for i in range(0,N):
    if mtls1.count(mtls[i])==1:
        SKU_vols+=[SKU_vols0[lookup(mtls1,mtls[i])]]
    else:
            if curr_mtls_all.count(mtls[i])>=1:
                qty_total=curr_qty_all[lookup(curr_mtls_all,mtls[i])]
                PC=(100-stats.percentileofscore(curr_qty_all,qty_total))*0.9
                SKU_vols+=[np.percentile(SKU_vols0,PC)+rnd.random()]
            else:
                SKU_vols+=[np.percentile(SKU_vols0,10)+rnd.random()]

demand=demand0
demand=make_positive(demand)
print('Padded Missing Data')
toc(ST)


# Shipment Data
if shipment:
    ship_mtls=shipment_received['Material'].tolist()
    ship_qtys=shipment_received['Quantity'].tolist()
    ship_batch=rem_nan(shipment_received['Batch'],EMPTY=True)


avg_picked_at_once=np.array(HIST_picks["avg_picked_at_once"])
trip_demand=demand/avg_picked_at_once # we set the trip demand as expected number of trips per unit time

```

```
417.  ## INITIALIZATIONS
418.
419.  def initialize(n=N,m=M):
420.      num_placed=np.zeros(n).tolist()
421.      currently_there=[] #currently_there format [mtl, qty, vlm per, total vlm, batch]
422.      for i in range(0,m):
423.          currently_there+=[[]]
424.      vol_remaining=copy.deepcopy(loc_vols)
425.      return num_placed,currently_there,vol_remaining
426.
427.  if shipment:
428.      num_SKU_ship=len(ship_mtls)
429.      curr_mtls+=ship_mtls
430.      curr_batch+=ship_batch
431.      curr_qty+=np.zeros(num_SKU_ship).tolist()
432.      curr_locs+=np.zeros(num_SKU_ship).tolist()
433.
434.  MTLS=unique_list(mtls_with_batches(curr_mtls,curr_batch))
435.
436.  def current_locations_to_total(curr_mtls,curr_locs,curr_qty,curr_batch,MTLS): #defines to
       place by the current locations
437.      n=len(MTLS)
438.      num_total=np.zeros(n)
439.      for i in range(0,len(curr_mtls)):
440.          ind=lookup(MTLS,[curr_mtls[i],curr_batch[i]])
441.          num_total[ind]+=curr_qty[i]
442.      return num_total,n
443.
444.
445.  ## Place/Remove a SKU command
446.
447.  def batch_ind(SKU_ind,batch):  #get
448.      if MTLS.count([new_mtls[SKU_ind],batch])==1:
449.          return lookup(MTLS,[new_mtls[SKU_ind],batch])
450.      else:
451.          raise ValueError("Can't get batch index")
452.  def sku_ind(BATCH_ind):
453.      if new_mtls.count(MTLS[BATCH_ind][0])>=1:
454.          SKU_ind=lookup(new_mtls,MTLS[BATCH_ind][0])
455.          batch=MTLS[BATCH_ind][1]
456.          return SKU_ind,batch
457.      else:
458.          raise ValueError("Can't get SKU index")
459.
460.  def N_to_n(data):
461.      data_new=[]
462.      for i in range(n):
463.          data_new+=[data[lookup(mtls,MTLS[i][0])]]
464.      return data_new
465.
466.  def place(sku_vols,num_total,num_placed,loc_vols,costs,currently_there,vol_remaining,SKU_
     BATCH_ind,LOC_ind,QTY):
467.      SKU_ind,batch=sku_ind(SKU_BATCH_ind)
468.      num_placed[SKU_BATCH_ind]+=QTY
469.      currently_there[LOC_ind]+=[[MTLS[SKU_BATCH_ind][0],QTY,sku_vols[SKU_BATCH_ind],QTY*sk
     u_vols[SKU_BATCH_ind],MTLS[SKU_BATCH_ind][1]]]
470.      vol_remaining[LOC_ind]-=QTY*sku_vols[SKU_BATCH_ind]
471.      return num_placed,currently_there,vol_remaining
472.
473.  def remove(sku_vols,num_total,num_placed,loc_vols,costs,currently_there,vol_remaining,SKU
     _BATCH_ind,LOC_ind,QTY,CLEAR=True,):
```

```
    SKU_ind,batch=sku_ind(SKU_BATCH_ind)
    location_contents=currently_there[LOC_ind]
    mtls_there=[[a[0],a[4]] for a in location_contents]
    if mtls_there.count(MTLS[SKU_BATCH_ind])==1:
        in_loc_ind=lookup(mtls_there,MTLS[SKU_BATCH_ind])
        if location_contents[in_loc_ind][1]>=QTY:
            location_contents[in_loc_ind][1]-=QTY
            location_contents[in_loc_ind][3]=location_contents[in_loc_ind][2]*location_co
ntents[in_loc_ind][1]
            vol_remaining[LOC_ind]+=QTY*location_contents[in_loc_ind][2]
            num_placed[SKU_BATCH_ind]-=QTY
            if (CLEAR==True)&(location_contents[in_loc_ind][1]==0):
                location_contents.remove(location_contents[in_loc_ind])
        else:
            raise ValueError('NOT ENOUGH THERE TO REMOVE')
    if mtls_there.count(MTLS[SKU_BATCH_ind])==0:
        raise ValueError('NOT THERE AT ALL')
    if mtls_there.count(MTLS[SKU_BATCH_ind])>1:
        raise ValueError('MULTIPLE IN SAME LOCATION')
    return num_placed,currently_there,vol_remaining

##PLACE FROM CURRENT SKU LOCATIONS
num_total,n=current_locations_to_total(curr_mtls,curr_locs,curr_qty,curr_batch,MTLS)
(num_placed,currently_there,vol_remaining)=initialize(n)

SKU_vols_short=N_to_n(SKU_vols)
trip_demand_short=N_to_n(trip_demand)
new_mtls=N_to_n(mtls)

def place_current(curr_mtls,curr_locs,curr_qty,curr_batch,UNREC=True):
    #Reinitialize
    (num_placed,currently_there,vol_remaining)=initialize(n,m)
    curr_mtls_inds=np.zeros(len(curr_mtls)).tolist()
    curr_mtls_inds=[]
    curr_locs_inds=np.zeros(len(curr_locs)).tolist()
    curr_locs_inds=[]
    for i in range(0,len(curr_mtls)):
        #if mtls.count(curr_mtls[i])>=1:
        if locs.count(curr_locs[i])>=1:
            curr_mtls_inds+=[MTLS.index([curr_mtls[i],curr_batch[i]])]
# ============================================================================
#     for i in range(0,len(curr_locs)):
#         if locs.count(curr_locs[i])>=1:
# ============================================================================
            curr_locs_inds+=[locs.index(curr_locs[i])]
        elif UNREC: #if set to True, it will place SKU's in unrecognized locations
            curr_mtls_inds+=[MTLS.index([curr_mtls[i],curr_batch[i]])]
            curr_locs_inds+=[0] #PLACE SKU's in unrecognized location in first location
    for i in range(0,len(curr_mtls_inds)):
        (num_placed,currently_there,vol_remaining)=place(SKU_vols_short,num_total,num_pla
ced,loc_vols,costs,currently_there,vol_remaining,curr_mtls_inds[i],curr_locs_inds[i],curr
_qty[i])
    return num_placed,currently_there,vol_remaining

def combine_like(currently_there):
    C=copy.deepcopy(currently_there)
    mtls_there=[]
    num_like=[]
    for i in range(0,len(C)):
        num_like+=[[]]
        mtls_there=[[a[0],a[4]] for a in C[i]]
```

```
        for j in range(0,len(C[i])):
            num_like[i]+=[mtls_there.count(mtls_there[j])]
    for i in range(0,len(C)):
        mtls_there=[[a[0],a[4]] for a in C[i]]
        to_remove=[]
        mtl_dealt_with=[]
        for j in range(0,len(C[i])):
            if (num_like[i][j]>1) & (mtl_dealt_with.count([C[i][j][0],C[i][j][4]])==0):
                indices = [k for k, x in enumerate(mtls_there) if x == mtls_there[j]]
                qtys = [C[i][k][1] for k in indices]
                C[i][indices[0]][1]=sum(qtys)
                mtl_dealt_with+=[[C[i][j][0],C[i][j][4]]]
                for k in range(1,len(indices)):
                    to_remove+=[C[i][indices[k]]]
        for k in range(0,len(to_remove)):
            C[i].remove(to_remove[k])
    return C


num_placed_old,currently_there_old,vol_remaining_old=place_current(curr_mtls,curr_locs,curr_qty,curr_batch)
currently_there_old=combine_like(currently_there_old)

if shipment:
    del curr_qty[-num_SKU_ship:]
    curr_qty+=ship_qtys
    num_total,n=current_locations_to_total(curr_mtls,curr_locs,curr_qty,curr_batch,MTLS)

    (num_placed,currently_there,vol_remaining)=initialize(n)


print("SKUs in Region:",n)
print("Storage Locations in Region:",m)



def clear_zeros(currently_there):
    C=copy.deepcopy(currently_there)
    return C

## COST OF SITUATION

def part_location_cost(SKU_ind,LOC_ind,costs,trip_dem,sku_vols):
    return costs[LOC_ind]*trip_dem[SKU_ind]
def part_location_cost_util(SKU_ind,LOC_ind,costs,trip_dem,sku_vols):
    return costs[LOC_ind]*trip_dem[SKU_ind]/sku_vols[SKU_ind]
def current_location_costs(currently_there,costs,trip_dem,sku_vols,func):
    current_costs=[]
    current_costs_total=[]
    for i in range(0,len(currently_there)):
        current_costs+=[[]]
        current_costs_total+=[[]]
        for j in range(0,len(currently_there[i])):
            QTY=currently_there[i][j][1]
            current_costs[i]+=[[]]
            current_costs_total[i]+=[[]]
            #per part
            current_costs[i][j]=func(lookup(MTLS,[currently_there[i][j][0],currently_there[i][j][4]]),i,costs,trip_dem,sku_vols)
```

```
            #total
            current_costs_total[i][j]=QTY*func(lookup(MTLS,[currently_there[i][j][0],curr
ently_there[i][j][4]]),i,costs,trip_dem,sku_vols)
    return current_costs,current_costs_total



def total_vol(volumes,qtys=[]):
    if qtys==[]:
        qtys=np.ones(len(volumes))
    total=0
    for i in range(len(volumes)):
        total+=volumes[i]*qtys[i]
    return total
TOTAL_STORAGE_VOLUME=total_vol(loc_vols)
curr_qty_long=[]
for i in range(N):
    if curr_mtls.count(mtls[i])==1:
        curr_qty_long+=[curr_qty[lookup(curr_mtls,mtls[i])]]
    else:
        curr_qty_long+=[0]
TOTAL_SKU_VOLUME=total_vol(SKU_vols,curr_qty_long)
BIGGEST_LOCATION_VOLUME=max(loc_vols)
BIGGEST_SKU_VOLUME = max(SKU_vols_short)

util=iden(np.array(trip_demand_short))/iden(np.array(SKU_vols_short))

def placement_alg(num_total,num_placed,currently_there,vol_remaining,trip_dem,costs,sku_v
ols,loc_vols,ITER=1000000,a=iden,b=iden,SEP_LIKE=True):
    m=len(currently_there)
    num_to_place=(np.array(num_total)-np.array(num_placed))
    still_to_place_inds=[i for i, x in enumerate(num_to_place>0) if x]
    util=a(np.array(trip_dem))/b(np.array(sku_vols))+np.random.rand(len(trip_dem))/100000
00
    cost_order=np.argsort(costs)
    for i in range(0,ITER): #should terminate sooner than this
        rem_util=[util[k] for k in still_to_place_inds]
        if rem_util==[]:
            break
    #    max_util_ind=np.argmax([util[i] for i in still_to_place_inds])
        max_util_ind=util.tolist().index(max(rem_util)) #gives SKU to place
        #print(max_util_ind)
        for j in range(0,m):
            #print(num_to_place[max_util_ind])
            if num_to_place[max_util_ind]==0:
                break
            other_mtls=[a[0] for a in currently_there[cost_order[j]]]
            other_mtls_with_batch=[[a[0],a[4]] for a in currently_there[cost_order[j]]]
            if SEP_LIKE & (other_mtls.count(new_mtls[max_util_ind])>=1) & (other_mtls_wit
h_batch.count(MTLS[max_util_ind])==0): #skips that location if another SKU with the same
P/N but different batch is present
                continue
            can_fit=min(np.floor(vol_remaining[cost_order[j]]/sku_vols[max_util_ind]),num
_to_place[max_util_ind])
            if can_fit>0: #if there is remaining to be placed and any fit
                (num_placed,currently_there,vol_remaining)=place(sku_vols,num_total,num_p
laced,loc_vols,costs,currently_there,vol_remaining,max_util_ind,cost_order[j],can_fit)
                num_to_place=(np.array(num_total)-np.array(num_placed))
                still_to_place_inds=[k for k, x in enumerate(num_to_place>0) if x]
                util=(np.array(trip_dem)*np.array(num_to_place)/np.array(sku_vols))
    return num_placed,currently_there,vol_remaining,i,j,rem_util,still_to_place_inds
```

```
644.
645. # Remove worst utilized item (all quantity in that location as default)
646. def remove_worst(num_placed,currently_there,vol_remaining,trip_demand,costs,sku_vols,loc_
     vols,ALL=True):
647.     if checksum(currently_there)==0:
648.         raise ValueError('NOTHING TO REMOVE')
649.     worst=listmax(current_location_costs(currently_there,costs,trip_demand,sku_vols,part_
     location_cost_util)[0])
650.     SKU_ind=lookup(MTLS,[currently_there[worst[1]][worst[2]][0],currently_there[worst[1]]
     [worst[2]][4]])
651.     LOC_ind=worst[1]
652.     if ALL==True:
653.         QTY=currently_there[worst[1]][worst[2]][1]
654.     else:
655.         QTY=1
656.     (num_placed,currently_there,vol_remaining)=remove(sku_vols,num_total,num_placed,loc_v
     ols,costs,currently_there,vol_remaining,SKU_ind,LOC_ind,QTY)
657.     return num_placed,currently_there,vol_remaining
658. #(num_placed,currently_there,vol_remaining)=remove_worst(num_placed,currently_there,vol_r
     emaining,trip_demand_short,costs,SKU_vols_short,loc_vols)
659.
660. # Remove n worst utilized items
661. def remove_x(num_placed,currently_there,vol_remaining,trip_demand,costs,sku_vols,loc_vols
     ,x):
662.     for i in range(0,x):
663.         (num_placed,currently_there,vol_remaining)=remove_worst(num_placed,currently_ther
     e,vol_remaining,trip_demand,costs,sku_vols,loc_vols)
664.     return num_placed,currently_there,vol_remaining
665. #(num_placed,currently_there,vol_remaining)=remove_x(num_placed,currently_there,vol_remai
     ning,trip_demand_short,costs,SKU_vols_short,loc_vols,1)
666.
667. def replace_x(num_placed,currently_there,vol_remaining,trip_demand,costs,sku_vols,loc_vol
     s,x):
668.     num_total=copy.deepcopy(num_placed)
669.     (num_placed,currently_there,vol_remaining)=remove_x(num_placed,currently_there,vol_re
     maining,trip_demand,costs,sku_vols,loc_vols,x)
670.     ITER=10000000
671.     (num_placed,currently_there,vol_remaining,a,b,c,d)=placement_alg(num_total,num_placed
     ,currently_there,vol_remaining,trip_demand,costs,sku_vols,loc_vols,ITER,iden,iden)
672.     return num_placed,currently_there,vol_remaining
673.
674. 5-.72/2
675.
676. ## Make shipment add to quantities
677.
678. def add_shipment(ship_mtls,ship_qtys,ship_batch,MTLS,num_total):
679.     for i in range(len(ship_mtls)):
680.         num_total[lookup(MTLS,[ship_mtls[i],ship_batch[i]])]+=ship_qtys[i]
681.     return num_total
682. #num_total=add_shipment(ship_mtls,ship_qtys,mtls,num_total)
683.
684. desc=str_conv(rem_nan(SKU_list['Material Description'].tolist()))
685. def get_desc(mtl):
686.     return desc[lookup(mtls,mtl)]
687.
688.
689. def list_moves(currently_there_old,currently_there,SKU_vols,num_total,num_placed,loc_vols
     ,costs,vol_remaining,num_placed_old,vol_remaining_old):
690.     moves=[]
691.     OLD=copy.deepcopy(currently_there_old)
692.     NEW=copy.deepcopy(currently_there)
```

```
    for i in range(len(OLD)): #stuff in same place
        for j in range(len(OLD[i])):
            if [[a[0],a[4]] for a in NEW[i]].count([OLD[i][j][0],OLD[i][j][4]])==1:
                l=[[a[0],a[4]] for a in NEW[i]].index([OLD[i][j][0],OLD[i][j][4]])
                if (OLD[i][j][1]>0) & (NEW[i][l][1]>0):
                    num_moved=min(OLD[i][j][1],NEW[i][l][1])
                    OLD[i][j][1]-=num_moved
                    NEW[i][l][1]-=num_moved
    for i in range(len(OLD)): #stuff that moved
        for j in range(len(OLD[i])):
            for k in range(len(NEW)):
                for l in range(len(NEW[k])):
                    if (OLD[i][j][0]==NEW[k][l][0]) & (OLD[i][j][4]==NEW[k][l][4]) & (OLD
[i][j][1]>0) & (NEW[k][l][1]>0):
                        num_moved=min(OLD[i][j][1],NEW[k][l][1])
                        mtl_moved=lookup(MTLS,[OLD[i][j][0],OLD[i][j][4]])
                        if i!=k:
                            #assume cheaper location is positive cost drop:
                            cost_drop=num_moved*(part_location_cost(mtl_moved,i,costs,tri
p_demand_short,SKU_vols_short)-
part_location_cost(mtl_moved,k,costs,trip_demand_short,SKU_vols_short))
                            moves+=[[locs[i],locs[k],MTLS[mtl_moved][0],MTLS[mtl_moved][1
],get_desc(MTLS[mtl_moved][0]),num_moved,cost_drop]]
                        OLD[i][j][1]-=num_moved
                        NEW[k][l][1]-=num_moved
    if shipment:
        moves=[]
        for i in range(len(NEW)):
            for j in range(len(NEW[i])):
                if NEW[i][j][1]>0:
                    cost_drop=NEW[i][j][1]*part_location_cost(lookup(MTLS,[NEW[i][j][0],N
EW[i][j][4]]),i,costs,trip_demand_short,SKU_vols_short)
                    moves+=[['shipment',locs[i],NEW[i][j][0],NEW[i][j][4],get_desc(NEW[i]
[j][0]),NEW[i][j][1],cost_drop]]
    return moves,NEW,OLD,i,j,k,l

def currently_there_export(currently_there):
    output=[]
    for i in range(0,len(currently_there)):
        for j in range(0,len(currently_there[i])):
            output+=[[locs[i],currently_there[i][j][0],currently_there[i][j][4],get_desc(
currently_there[i][j][0]),currently_there[i][j][1],util[lookup(MTLS,[currently_there[i][j
][0],currently_there[i][j][4]])]]]
    df=pd.DataFrame(output)
    df.columns = ['Location', 'Material','Batch Number','Description','Quantity','Utility
']
    return df

def moves_export(moves):
    output=pd.DataFrame(moves)
    df=pd.DataFrame(output)
    if shipment:
        df.columns = ['Origin','Destination', 'Material','Batch Number','Description','Qu
antity','Cost Increase']
    else:
        df.columns = ['Origin','Destination', 'Material','Batch Number','Description','Qu
antity','Cost Reduction']
    return df

def total_cost(currently_there):
```

```
742.     return listsum(current_location_costs(currently_there,costs,trip_demand_short,SKU_vol
     s_short,part_location_cost)[1])
743.
744. def one(x):
745.     return np.ones(len(x))
746.
747. def get_qtys(currently_there):
748.     qtys=[]
749.     for i in range(0,len(currently_there)):
750.         qtys+=[[]]
751.         for j in range(0,len(currently_there[i])):
752.             qtys[i]+=[currently_there[i][j][1]]
753.     return qtys
754.
755. #num_placed_old,currently_there_old,vol_remaining_old=place_current(curr_mtls,curr_locs,c
     urr_qty,curr_batch)
756. #currently_there_old=combine_like(currently_there_old)
757.
758. if full_relocation:
759.     (num_placed,currently_there,vol_remaining)=initialize(n,m)
760.     if DATASET=="real":
761.         ITER=1000000
762.     if DATASET=="fake":
763.         ITER=1000000
764.     (num_placed,currently_there,vol_remaining,i,j,rem_util,still_to_place_inds)=placement
     _alg(num_total,num_placed,currently_there,vol_remaining,trip_demand_short,costs,SKU_vols_
     short,loc_vols,ITER,iden,iden)
765.
766. if replace:
767.     num_placed,currently_there,vol_remaining=place_current(curr_mtls,curr_locs,curr_qty,c
     urr_batch)
768.     currently_there=combine_like(currently_there)
769.     num_placed,currently_there,vol_remaining=replace_x(num_placed,currently_there,vol_rem
     aining,trip_demand_short,costs,SKU_vols_short,loc_vols,X)
770.
771. if shipment:
772.     num_placed,currently_there,vol_remaining=place_current(curr_mtls,curr_locs,curr_qty,c
     urr_batch,UNREC=False)
773.     currently_there=combine_like(currently_there)
774.     if DATASET=="real":
775.         ITER=1000000
776.     if DATASET=="fake":
777.         ITER=1000000
778.     (num_placed,currently_there,vol_remaining,i,j,rem_util,still_to_place_inds)=placement
     _alg(num_total,num_placed,currently_there,vol_remaining,trip_demand_short,costs,SKU_vols_
     short,loc_vols,ITER,iden,iden)
779.
780. print("Organized New Locations")
781.
782. (moves,Currently_there,Currently_there_old,i,j,k,l)=list_moves(combine_like(currently_the
     re_old),currently_there,SKU_vols,num_total,num_placed,loc_vols,costs,vol_remaining,num_pl
     aced_old,vol_remaining_old)
783.
784. print("Generated Summary:")
785.
786. COST_old=total_cost(currently_there_old)
787. print("Current Total Location Cost Units: ",COST_old)
788. COST=total_cost(currently_there)
789. print("New Total Location Cost Units: ",COST)
790. if shipment:
791.     summ=-(COST_old-COST)
```

```
792.     print("Added Cost:", summ)
793. else:
794.     summ=(COST_old-COST)/COST_old*100
795.     print("Percent Improvement: ",summ,"%")
796.
797. if shipment:
798.     summary=pd.DataFrame([["Current Total Location Cost Units:",COST_old],["New Total Loc
     ation Cost Units:",COST],["Added Cost:",summ]])
799.     summary.columns = ['','']
800. else:
801.     summary=pd.DataFrame([["Current Total Location Cost Units:",COST_old],["New Total Loc
     ation Cost Units:",COST],["Percent Improvement:",summ,"%"]])
802.     summary.columns = ['','','']
803.
804. if shipment:
805.     filename=change_colons('Outputs/shipment put-away '+str(datetime.now())+'.xlsx')
806. if full_relocation:
807.     filename=change_colons('Outputs/total_relocation '+str(datetime.now())+'.xlsx')
808. if replace:
809.     filename=change_colons('Outputs/replace X '+str(datetime.now())+'.xlsx')
810.
811. writer = pd.ExcelWriter(filename)
812. if shipment:
813.     export=moves_export(moves)
814.     export.to_excel(writer,'Put-away Locations',index=False)
815. else:
816.     export=DF.sort_values(moves_export(moves),by='Cost Reduction',ascending=False)
817.     export.to_excel(writer,'List of Moves',index=False)
818.     currently_there_export(currently_there).to_excel(writer,'New Layout',index=False)
819.     currently_there_export(currently_there_old).to_excel(writer,'Old Layout',index=False)
820. summary.to_excel(writer,"Summary",header=False,index=False)
821. writer.save() #SAVE TO EXCEL
822. print("Saved File: "+filename)
823.
824. toc(ST)
```