Enhancing Middle-Mile Inventory Management Policies Through Simulation and Reinforcement Learning

by

Matthew Robins
B.S., Applied Mathematics - Computer Science
Brown University, 2020

Submitted to the MIT Sloan School of Management and
Operations Research Center
in partial fulfillment of the requirements for the degrees of

Master of Business Administration

and

Master of Science in Operations Research

in conjunction with the Leaders for Global Operations program
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 2024

Author ....
MIT Sloan School of Management and
Operations Research Center
May 18, 2024

Certified by ....
Georgia Perakis, Thesis Supervisor
John C Head III Dean (Interim), MIT Sloan School of Management;
Professor, Operations Management, Operations Research & Statistics

Certified by ....
Vivek Farias, Thesis Supervisor
Patrick J. McGovern (1959) Professor, Operations Management

Accepted by ....
Georgia Perakis
John C Head III Dean (Interim), MIT Sloan School of Management;
Professor, Operations Management, Operations Research & Statistics

Accepted by ....
Maura Herson
Assistant Dean, MBA Program, MIT Sloan School of Management

Enhancing Middle-Mile Inventory Management Policies Through Simulation and Reinforcement Learning

by

Matthew Robins

Submitted to the MIT Sloan School of Management and
Operations Research Center
on May 18, 2024, in partial fulfillment of the
requirements for the degrees of
Master of Business Administration
and
Master of Science in Operations Research

Abstract

This thesis explores approaches for enhancing middle-mile inventory management within the global supply chain of a large footwear and apparel company, referred to as "Atlas". The first part discusses the design and implementation of a high-performance, heuristic system to determine stock transfer order (STO) decisions between Atlas's distribution centers. This system employs a greedy algorithm to match supply to demand while respecting resource constraints. As Atlas's newly procured third-party solution proved insufficient for testing due to slow performance, this work develops an emulator of the production system that achieves a 30x speedup and integrates with Atlas's end-to-end supply chain simulation framework. This emulator enabled Atlas to efficiently test different configurations and decision making rules on historical and theoretical data, providing valuable insights prior to deploying the production system. The second part investigates the potential of reinforcement learning (RL) to augment or replace Atlas's middle-mile decision making. A simplified supply chain environment is modeled as a Markov Decision Process, and an RL agent is trained and benchmarked against optimization-based and heuristic approaches. While the RL policy does not outperform these alternatives in the simplified environment, this work provides a foundation for Atlas to explore RL applications as they scale to more realistic supply chain environments.

Thesis Supervisor: Georgia Perakis
Title: John C Head III Dean (Interim), MIT Sloan School of Management;
Professor, Operations Management, Operations Research & Statistics

Thesis Supervisor: Vivek Farias
Title: Patrick J. McGovern (1959) Professor, Operations Management

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

In the context of global merchandising, large-scale retailers are confronted with the challenge of managing complex multi-echelon supply chains. This complexity arises from the extensive scale and diversity of their operations, coupled with the unpredictability of market demands and the need for efficiency and customer satisfaction. The work done as part of this thesis was carried out at one such retailer, which will be denoted as Atlas throughout, for the sake of anonymity.

One central focus of this challenge is inventory flow management, a critical process in which a retailer monitors and controls its inventory from the manufacturer to its distribution network and finally to the point of sale. The effectiveness of inventory management policies plays a pivotal role in enhancing a retailer's operational efficiency, adaptability to market fluctuations, and overall financial performance. Traditional inventory management methods, typically linear and static in nature, may be insufficient for addressing the high-dimensional or stochastic problems faced by large retailers. Conventional approaches may struggle to incorporate practical business constraints or to remain robust under varying model assumptions.

Consequently, retailers are increasingly turning to advanced analytics and optimization techniques to improve their inventory management strategies, often incorporating elements from stochastic optimization, heuristic algorithms, and deep learning. The decisions regarding the flow of inventory through a supply chain network can be modeled as a Markov Decision Process (MDP). This perspective allows the sequential decision-making problem of inventory flow to be approached using Reinforcement Learning (RL) methods. Recent research in this area has shown potential in applying RL to such problems [16].

This thesis is divided into two parts. The first part discusses the development and implementation of a greedy heuristic decision engine for middle-mile inventory positioning within Atlas's supply chain. The term "middle-mile" refers to the movement of inventory between distribution centers, as opposed to the "first-mile" (factory to distribution center) and "last-mile" (distribution center to customer) stages. This decision engine is specifically designed to align with Atlas's operational constraints and goals, making this part of the thesis a case study in the real-world application of supply chain decision-making tools.

The second part of the thesis explores the use of RL for making middle-mile inventory management decisions in Atlas's North American supply chain. This involves framing the supply chain as an RL problem, building a simplified environment incorporating key elements of the real supply chain network, developing and training a parametrized inventory management policy, and evaluating its performance through simulation. A crucial aspect of this analysis is benchmarking the RL policy against traditional heuristic and model predictive control policies, and conducting ablation studies to understand the impact of model assumptions and environmental changes on model performance. This will lead to an analysis on the potential application of RL in Atlas's operational context, including considerations for future improvements such as better constraint incorporation, enhanced learning efficiency, and scalability.

In summary, this thesis provides an examination of heuristic and RL approaches in the context of Atlas's inventory management. While offering insights into the challenges and potentials of these methods, it also acknowledges the complexities and limitations inherent in applying advanced algorithms to real-world supply chain scenarios. This work contributes to the broader understanding of inventory management in large-scale retail environments and serves as a stepping stone for further exploration and refinement in this field.

1.1 Industry Overview

1.1.1 Footwear and Apparel Market

In the $1.5 trillion USD global footwear and apparel [14], a pronounced shift towards direct-to-consumer (DTC) models, spurred by e-commerce proliferation and the COVID-19 pandemic, has necessitated a fundamental reevaluation of supply chain strategies among leading players such as Nike, Adidas, Lululemon, Zara, and H&M. This transition from centralized distribution to physical outlets towards a model catering to a varied and geographically diverse consumer base introduces complex logistical challenges.

The DTC approach, diverging from uniform bulk shipments to brick-and-mortar stores, demands tailored inventory management capable of handling diverse, individualized consumer orders. This complexity requires advanced forecasting techniques and dynamic replenishment strategies, striking a critical balance between inventory holding costs and the risks of stockouts, which directly impact customer satisfaction and digital marketplace reputation.

Additionally, heightened consumer expectations for rapid and flexible delivery have placed a premium on optimizing "last-mile" logistics, historically the most resource-intensive segment of the supply chain. In response, industry giants are investing in sophisticated inventory systems that integrate data analytics, machine learning, and artificial intelligence. These systems are designed to enhance demand prediction, streamline inventory distribution, and improve supply chain efficiency from production to end-user delivery.

Thus, the shift to DTC models in the footwear and apparel sector, driven by digital integration and evolving consumer behavior, compels a strategic overhaul in inventory management. Retailers are now tasked with developing agile, data-driven approaches to meet the demands of a digitally connected global consumer base, ensuring efficient product delivery that aligns with evolving market expectations and operational goals.

1.1.2 Key Supply Chain Variables for Consideration

In the domain of supply chain management for the global footwear and apparel industry, a rigorous and methodical approach is required to synthesize various key factors. Forecasting demand is a critical yet challenging component, requiring an in-depth analysis of historical sales, market trends, and consumer behavior. This forecasting must be adeptly aligned with supply chain dynamics, encompassing supplier reliability, lead times, and the impact of geopolitical factors, to ensure a consistent and responsive product flow.

Financial considerations, specifically transportation and holding costs, are closely connected with these supply and demand dynamics. Transportation costs, influenced by fluctuating fuel prices, carrier rates, and logistical efficiencies, must be optimized alongside the expenses incurred in warehousing and inventory maintenance. These costs are significant determinants of the overall supply chain budget and require strategic management to achieve a balance that maintains high service levels without excessive expenditure.

Product specificities, such as Global Trade Item Numbers (GTINs) or stock keeping units (SKUs), also play a crucial role in effective supply chain management. These identifiers not only facilitate efficient tracking and inventory management but also assist in tailoring logistics strategies to the unique characteristics of each product, such as size, weight, and perishability. Often times, each SKU is linked to a designated case pack quantity or batch size, dictating that transportation and handling of these items occur in precise, integral multiples. This requirement ensures uniformity and efficiency in the movement of goods throughout the network.

Additionally, capacity constraints within production facilities, warehouses, and transportation networks must be carefully evaluated. Identifying and managing these constraints is essential to prevent bottlenecks and maintain a smooth flow of goods through the supply chain.

In integrating these factors, supply chain optimization in the footwear and apparel industry requires a sophisticated approach that balances operational efficiency, cost management, and responsiveness to market demands. This necessitates a holistic view of the supply chain, considering each element not in isolation but as part of an interconnected system that drives overall business performance.

1.2 Atlas Supply Chain

The focus of this work is on Atlas's North American supply chain. Due to long lead times for bulk orders, long-term planning for far-out seasonal trends, and legacy reasons, it can take at least 6 months from when an order is put in at a manufacturer to when it actually arrives in the United States. Due to this large delay from when units are scheduled for production to when they arrive, the supply chain decisions in this thesis remain focused on positioning the inventory within network to best meet key target metrics of the business. We assume the upstream decisions regarding incoming supply are fixed and outside of our control.

1.2.1 Atlas North American Supply Chain: Terminology and Network

In the digital supply chain of Atlas, three primary types of orders play a role in the movement and management of inventory. These orders dictate the flow of goods through various stages of the supply chain, from manufacturing to final delivery to the customer.

- **PO (Principle Order):** Principle Orders mark the commencement of the inventory route. They specify the products, quantities, and delivery timelines from suppliers or manufacturers. In Atlas's supply chain, POs are significant for tracking the journey of inventory from the Consolidator in Asia to destinations within the North American supply chain, which could be an CDF, CDC, or RSC (all defined below).

- **STO (Stock Transfer Order):** STOs facilitate the internal movement of stock between different Atlas facilities. They are critical for balancing inventory across the network, ensuring that products are redistributed efficiently in response to regional demand and supply dynamics.

- **Fulfillment Orders:** These orders pertain to the final stage of the supply chain, focusing on last-mile customer fulfillment. They are essential for ensuring that the end consumer receives their products in a timely and efficient manner, directly impacting customer satisfaction and service quality.

Alongside these order types, a network of operational hubs and strategic regional points forms the integral framework of Atlas's digital supply chain:

- **Consolidator:** This facility plays a strategic role in Atlas's supply chain, where manufactured goods from various factories across Asia are consolidated. The Consolidator acts as a central hub before these goods are shipped to North America, streamlining the initial stage of the distribution process.

- **CDF (Cross Dock Facility):** The CDF's primary function is the receipt, sorting, and processing of products received from the Consolidator. This facility operates as a sortation hub, organizing goods for distribution within the supply chain. Importantly, the CDF is not a storage facility and does not retain inventory overnight, functioning solely as a transitional point near the cross-dock for inbound shipments. In this sense, the CDF is referred to as a pass-through facility. Additionally, the use of the CDF for incoming inventory is not compulsory; many Principle Orders (POs) are directed to bypass the CDF, with goods shipped directly to regional or central distribution centers.

- **RSC (Regional Service Center):** The RSC functions as a regional warehouse and distribution center within Atlas's supply chain. It is responsible for managing the distribution of products within its designated region. RSCs receive inventory either directly from Principle Orders (POs), the CDF, or the CDC. Unlike the CDF, RSCs are equipped for storage and play a critical role in regional warehousing. They also handle fulfillment orders for their respective regions, with 'in-region' fulfillment to a Demand Location (DL) being a preferred option due to its cost efficiency and shorter delivery times.

- **CDC (Central Distribution Center):** The CDC is at the heart of the supply chain, managing the storage and distribution of products to retail outlets, e-commerce channels, and to the RSCs. It is essential for maintaining inventory levels and ensuring the timely availability of products throughout the network. It has higher storage capacity than the RSCs.

- **DL (Demand Location):** A Demand Location represents a broad geographic region within the United States in Atlas's supply chain framework. Each DL is associated with a specific Regional Service Center (RSC). This association is strategic, aligning the distribution capabilities of the RSC with the demand patterns and logistical requirements of the region it serves. Fulfillment orders dispatched from an RSC to its associated DL are considered 'in-region' and are preferable due to their cost-effectiveness and the efficiency of delivery. This structure ensures that each broad region has a dedicated distribution hub, optimizing the supply chain's responsiveness and agility in meeting regional demand. In practice, Atlas has 3 DLs and corresponding RSCs.

Figure 1-1 illustrates how the Atlas digital supply chain network works. Edges A, B, and C represent possible first-mile PO routes for inventory, and edges D, E, and F represent possible middle-mile STO routes for inventory. The dashed lines to the end consumer represent last-mile fulfillment to an end customer. The focus of this thesis is on the middle-mile STO decisions. Throughout, "nodes" refers to facilities that hold or ship inventory, "edges" or "lanes" refer to potential pathways for inventory to flow from one node to another.

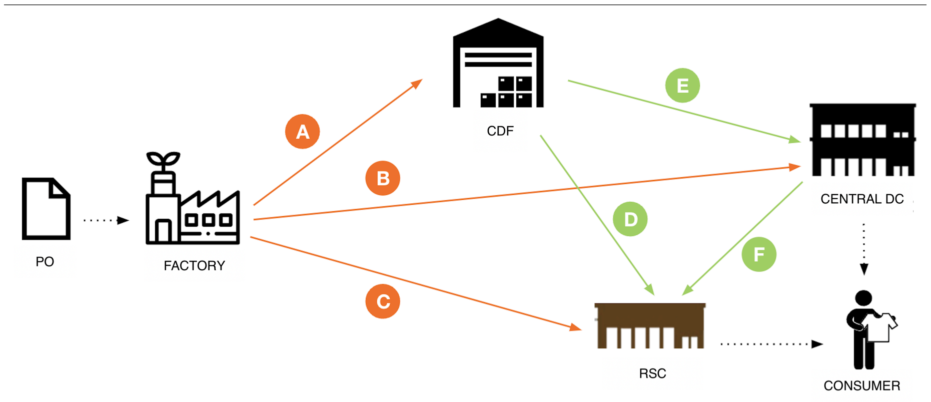


Figure 1-1: High-Level Illustration of Atlas Digital Supply Chain

1.2.2 Constraints and Objectives

Constraints

In optimizing Atlas's supply chain, several constraints must be carefully considered:

- **Lane Capacity Constraints:** The capacity of transportation lanes is a limiting factor, dictating the maximum volume of goods that can be moved on a given day.
- **Inbound and Outbound Processing Capacity:** The capacity of facilities to process inbound and outbound shipments imposes limits on the volume of goods that can be handled each day.
- **Receiving Storage Capacity:** Storage capacity at receiving facilities restricts the amount of inventory that can be held.
- **Full Case Pack Quantity Constraints:** The requirement to transport goods in full case pack quantities introduces limitations in inventory movement, impacting the optimization of shipment sizes and frequencies.
- **Inventory Availability Constraints:** A fundamental constraint is the availability of inventory on hand; the supply chain cannot distribute more inventory than what is physically available.

Objectives

The primary objectives for Atlas in optimizing the supply chain revolve around cost minimization and efficiency:

- **Minimizing Transportation Costs:** While Principle Order (PO) decisions are considered fixed or sunk costs in this analysis, significant potential exists to influence Stock Transfer Order (STO) costs directly. Additionally, effective inventory positioning can indirectly reduce fulfillment costs by optimizing in-region distribution.

- **Maximizing In-Region Fulfillment:** A key performance indicator is the percentage of in-region fulfillment. Higher in-region fulfillment rates correlate with reduced transportation costs and improved customer service.

- **Optimizing Inventory Velocity:** Metrics such as inventory turns are crucial for evaluating the efficiency of the supply chain. Effective inventory management aims to increase the velocity of inventory turnover, balancing the need for product availability against the cost of holding inventory.

1.3 Problem Motivation and Approach

The transformation of Atlas's supply chain was significantly influenced by the rise of digital demand. Historically, supply was predominantly directed to the Central Distribution Center (CDC) for wholesale distribution. However, the integration of Regional Service Centers (RSCs) marked a strategic shift to better cater to digital demand. RSCs, being closer to the customer, facilitated the breakdown of traditionally large wholesale case packs into smaller units suitable for retail distribution.

Initially, Stock Transfer Order (STO) decisions concerning the movement of inventory from the CDC to the RSCs, were predominantly manual, with managers executing these transfers on an ad hoc basis. As Atlas's operations evolved, the company expanded to incorporate three RSCs and transitioned to an automated, linear programming (LP)-based solution for making STO decisions.

The traditional inventory flow from the Consolidator to an RSC followed a serial pattern: **Consolidator** $\rightarrow$ **CDC** $\rightarrow$ **RSC**. The introduction of the Cross Dock Facility (CDF) and the flexibility for Principle Orders (POs) to be shipped to any network node necessitated a revised inventory policy. To address this, Atlas acquired a third-party middle-mile STO decision engine.

The implementation of this decision engine, scheduled for early 2024, has encountered several challenges and open questions, partially addressed through this work:

1. **Lack of Testing Environment:** The third-party provider did not offer a testing sandbox or substantial integration support, leaving Atlas without a means to assess the engine's impact on key operational metrics in a production environment. Furthermore, this product does not integrate with Atlas's comprehensive end-to-end supply chain simulator, which includes both first-mile and last-mile decision aspects.

2. **Complex Configuration:** The third-party decision engine involves numerous parameters affecting STO logic. Identifying optimal settings for these parameters is a challenge, even with a simulation environment.

3. **Runtime Performance Issues:** The third-party decision engine requires up to six hours to generate a single day's STO decisions. This inefficiency poses significant hurdles for integration with Atlas's supply chain simulator; for instance, simulating long periods would have exorbitant run time, impeding the ability to rapidly test configurations and assess impact on performance metrics.

To address these issues, this work proposes the development of a high-performance, exact logical replica of the middle-mile decision engine (the "Emulator"), and its integration into Atlas's end-to-end supply chain simulation framework. The primary contribution of this study lies in the practical implementation of this decision engine, which not only significantly reduced run times but also enabled Atlas to test various configurations. This testing process was instrumental in uncovering and addressing critical unexpected behaviors in the decision engine that, if left undetected, could have led to significant operational challenges in a live production environment.

The design and implementation of this decision engine will constitute the first part of the thesis. The second part will explore how RL might be used to augment or replace Atlas's middle-mile decision making policy in the future. By framing the supply chain as an RL problem, we aim to build a simplified environment that incorporates key elements of the real supply chain network. Within this environment, we will develop and train a parametrized inventory management policy and evaluate its performance through simulation. A crucial aspect of this analysis will be benchmarking the RL policy against traditional heuristic and model predictive control policies, as well as conducting ablation studies to understand the impact of model assumptions and environmental changes on model performance. This will lead to an analysis of the potential application of RL in Atlas's operational context, including considerations for future improvements such as better constraint incorporation, enhanced learning efficiency, and scalability.

Contributions

This thesis aims to provide valuable contributions to the practical application of advanced decision-making techniques within the context of a large-scale retailer's supply chain operations. By developing a high-performance emulator for simulating Atlas's middle-mile supply chain decisions, we achieved a 30x speedup compared to the existing system. This enables efficient testing of various scenarios and configurations, empowering Atlas to make data-driven decisions and optimize their supply chain operations. The development of an internal RL library tailored to Atlas's supply chain environment allows for the exploration of RL-based approaches in enhancing inventory management policies. The library is designed to be benchmarked against traditional heuristic methods and optimization-based approaches, providing a comprehensive evaluation of its performance and suitability.

Through extensive simulations using the middle-mile emulator, we tested various configurations and uncovered critical unexpected behaviors in the decision engine.

This process prevented potential operational challenges that could have arisen in a live production environment, demonstrating the value of thorough testing and validation in real-world supply chain settings. Furthermore, the development and training of a parametrized inventory management policy using RL within a simplified supply chain environment showcase the potential of applying advanced decision-making techniques to complex supply chain problems. The performance of this policy was evaluated through simulation and benchmarked against traditional heuristic and model predictive control policies, providing insights into the relative strengths and weaknesses of these approaches.

Our work also includes an analysis of the potential application of RL in Atlas's operational context, considering factors such as constraint incorporation, learning efficiency, and scalability. This analysis provides a foundation for future improvements and the integration of RL-based approaches into Atlas's supply chain decision-making processes, highlighting the potential benefits and challenges of adopting these techniques in a real-world setting. By presenting a case study of Atlas's supply chain challenges and the application of advanced decision-making techniques, this thesis contributes to the broader understanding of inventory management in large-scale retail environments. This work demonstrates the importance of combining domain expertise with data-driven approaches to optimize supply chain operations and drive business value, serving as a valuable reference for practitioners and researchers in the field.

Chapter 2

Overview of Inventory Management Policies

This chapter begins with a review of inventory management policies, focusing on key dimensions relevant to the field, including deterministic versus stochastic models, supply chain network structure, inventory review frequency, and push versus pull strategies. It explores classic inventory management models along with their underlying assumptions. A discussion of Atlas's own supply chain follows, specifically examining its adaptation to these models and strategies in light of their own practical challenges. This includes an analysis of Atlas's approach to managing the complexities of its multi-echelon network, its use of both deterministic and stochastic forecasting methods, and the review systems within its global distribution framework.

2.1 EOQ

The seminal Economic Order Quantity (EOQ) model, established by Ford W. Harris in 1913, stands as a pivotal foundation in the field of inventory management. This model addresses the formulation of a replenishment strategy for a single item as it traverses through one inventory location, such as a retailer or warehouse, to meet demand.

While section 2.3 of our study delves into more sophisticated models from the literature, which are characterized by relaxed or different assumptions, it is this section that offers a derivation of the EOQ model. This derivation is not only instructive in understanding the underlying mechanics of the EOQ model but also serves as a critical stepping stone in appreciating the nuances and complexity of inventory management more generally.

The model considers the following costs

Cost Variable	**Definition**
K	Fixed cost per order
c	Variable cost per order, incurs c cost per unit
h	Holding cost, incurs h cost per unit per unit time

Table 2.1: List of Costs in EOQ Model

and relies on the assumptions that demand is deterministic and constant, that there is no order lead time, and that no shortages are allowed [6]. Non-integral order quantities are also allowed. Since ordered units instantaneously arrive at the inventory location due to the zero lead time assumption, it would not make sense for any policy to place an order before using up all units of on-hand inventory to fulfill demand; otherwise, unnecessary holding costs would be incurred. Additionally, since demand is constant and stationary, any optimal policy will put in replenishment orders for some fixed amount of units Q. For large Q, the facility will need to put in orders less frequently because it will have more inventory to fulfill demand before depletion. This reduction in the number of orders reduces the fixed order costs accrued but increases the holding costs since there will be more inventory at the facility on average. This trade-off is at the heart of the EOQ model, and the key result is finding an optimal $Q = Q^*$ to minimize total cost. The parametrization of such a policy leads to the classic "sawtooth" pattern that can be seen in Figure 2-1.

By way of example let us consider two EOQ policies, each facing a demand of 600 units per year, both shown in Figure 2-1. The red policy orders 150 units upon inventory depletion whereas the blue policy orders 200 units. Including orders put in at time $t = 0$, observe that the blue policy places 3 orders whereas the red policy places 4 over the first 12 months, incurring costs of $3(K + 200c)$ and $4(K + 150c)$,

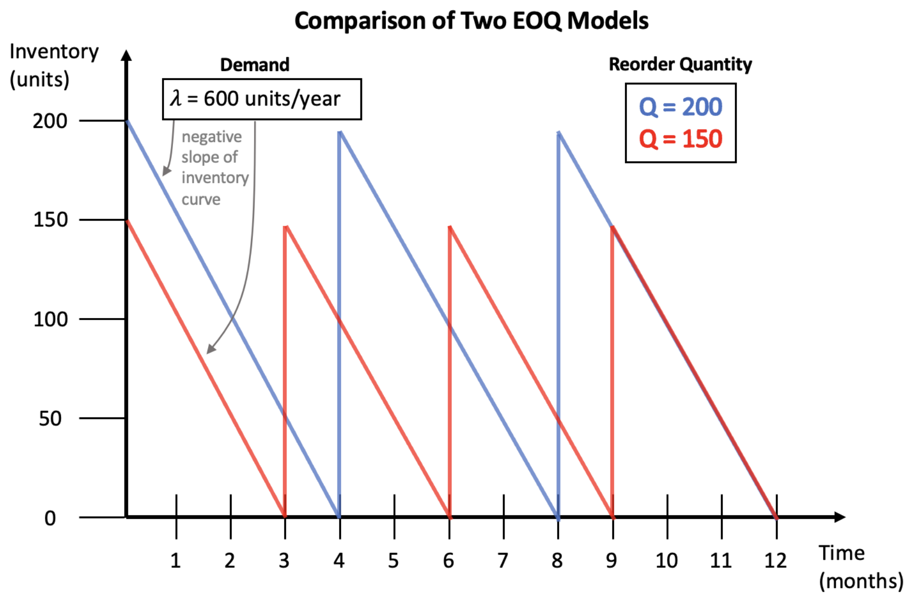


Figure 2-1: Example of Two EOQ Models with Identical Demand

respectively. Additionally, observe that the blue policy has greater average inventory than the red policy over the 12 month period, 100 units versus 75 units, respectively, which can be found by taking the average area under each curve. Thus, the holding costs over the period for the blue policy and red policy are $100h$ and $75h$, respectively. The blue policy achieves lower total cost whenever

$$3(K + 200c) + 100h < 4(K + 150c) + 75h$$
$$\implies 25h < K$$

As we can see, which EOQ policy is preferred in this example depends on the holding cost and fixed order cost, but not the variable cost per order c. As will be seen, this is true in general as solving for the optimal Q^* will only depend on $\lambda, K,$ and h.

For arbitrary Q, to fulfill all demand over one unit of time, this requires placing $\frac{\lambda}{Q}$ orders. Moreover, the total amount of units ordered over one unit of time will equal the total demand λ. The average inventory position over one unit of time is $\frac{Q}{2}$ as the inventory level linearly decreases from Q to 0. Altogether, this gives us a total cost

(TC) function:

$$TC(Q) = \underbrace{c\lambda}_{\text{Variable Transaction Cost}} + \underbrace{\frac{\lambda}{Q}K}_{\text{Fixed Transaction Cost}} + \underbrace{\frac{Q}{2}h}_{\text{Holding Cost}}$$

.

Differentiating and setting to zero gives us

$$Q^* = \sqrt{\frac{2\lambda K}{h}},$$

and by examining the second derivative of $TC(Q)$ at Q^* one would indeed find Q^* is the minimizer of total cost.

Figure 2-2, from Vandeput [15], depicts the general graph of the TC curve and decomposes them into the aggregate transaction and holding cost curves.

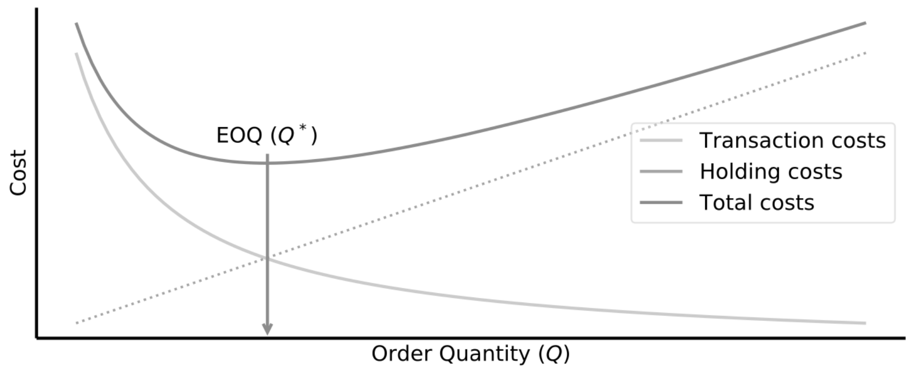


Figure 2-2: Trade-off Between Transaction and Holding Costs in EOQ Policy

While the EOQ model presented here applies to a deterministic one product setting without capacity constraints, many approaches have been taken to weaken or remove these assumptions, as shall be seen over the next few sections.

2.2 Beyond EOQ: Other Classical Single-Stage Inventory Policies

This section highlights other classical single-stage inventory policies in a context different from the deterministic demand assumption of the EOQ model. We follow the discourse in Silver, Pyke, and Peterson [12] which covers four typical policy parametrizations; this is meant to be representative but not exhaustive. These policies account for stochastic demand and allow for nonzero lead times, necessitating a buffer inventory, or safety stock (SS) level, to mitigate uncertainties. The safety stock serves as a safeguard against stockouts caused by unpredictable demand fluctuations or supply delays. Its goal is to balance consistent service levels with the costs of holding extra inventory and the risks of lost sales or reduced customer satisfaction from stock unavailability. Determining the right SS level requires analyzing demand and supply variability, transaction costs, and holding costs. This involves a trade-off similar to the EOQ model: higher SS levels reduce stock-out risks but increase holding costs.

The models also consider the order placement frequency, defined by the parameter R. An R value of 0 implies *continuous review* (orders can be placed anytime), while an R value greater than 0 indicates *periodic review* (orders placed at intervals of R time units).

Additionally, both *net stock* (NS) and *inventory position* (IP) are important concepts in understanding inventory management policies. Net stock is defined as the physical inventory on hand minus any backordered or unmet demand. This is a critical measure as it reflects the actual available inventory, considering both the items physically in the warehouse and any existing commitments to fulfill unmet customer orders. In mathematical terms, net stock can be represented as:

$$\text{Net Stock} = \text{On-hand Inventory} - \text{Backordered Demand}$$

The inventory position, in contrast, is a more comprehensive metric. It represents the anticipated future inventory level, factoring in both the current net stock and any outstanding orders yet to be delivered. Formally, the inventory position is expressed as:

$$\text{Inventory Position} = \text{Net Stock} + \text{Outstanding Orders}$$

2.2.1 1. Order-Point, Order Quantity (s, Q) System

This inventory management system is a continuous review policy that activates when inventory levels drop to or below a predetermined threshold, denoted as s. Upon reaching this critical level, an order of a fixed quantity Q is placed.

In Figure 2-3 below, an (s, Q) policy with parameters $s = 40, Q = 60$ is shown. Note that as soon as the inventory level reaches 40 units, a new order of 60 units is placed, causing the IP to increase to 100. The difference between the the dashed line representing NS, and the solid line representing IP, is the outstanding orders; these two lines converge once the lead time has elapsed and the order has been received.

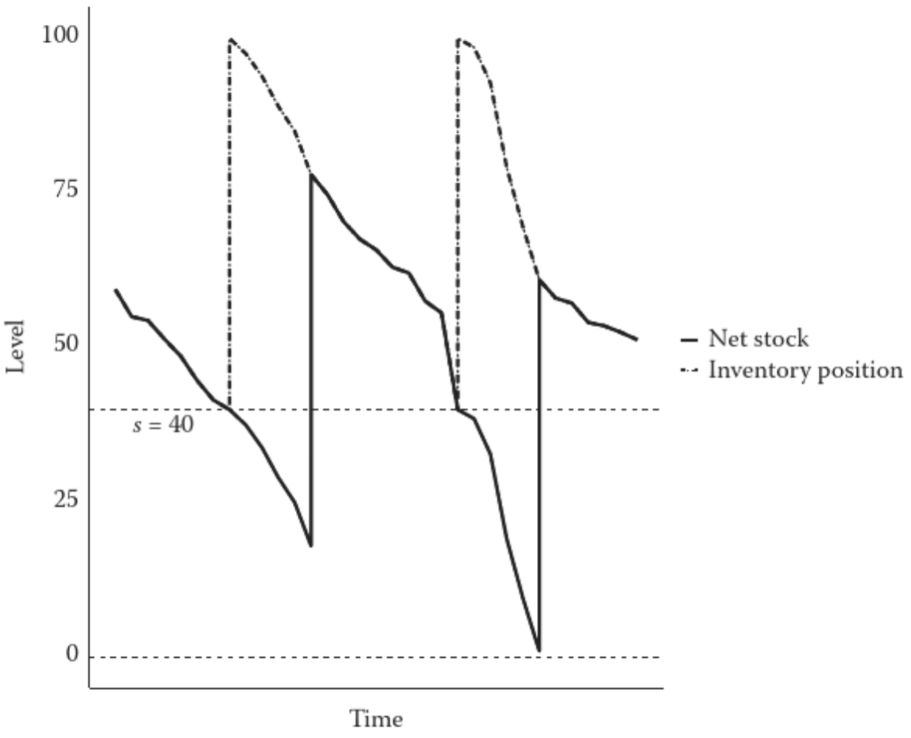


Figure 2-3: Example (s, Q) Policy; from Silver, Pyke, and Peterson [12]

2.2.2 2. Order-Point, Order-Up-to-Level (s, S) System

Similar to the (s, Q) system, the (s, S) system is also a continuous review policy whereby an order is triggered when the inventory level falls to the reorder point s. Unlike the (s, Q) system, the order quantity is not fixed but varies to replenish stock up to a pre-defined upper level S.

In the (s, S) system, the order quantity is dynamically determined based on the current inventory level when it reaches the reorder point. The main objective is to restore the inventory to the upper level S, thus the ordered quantity will be S minus the current inventory level. This flexibility allows the system to adapt to varying demand patterns and inventory levels more effectively than the (s, Q) system, but at the cost of losing standardization. For example, Atlas and many retailers require shipping in full case packs so variable order quantities are not always feasible.

Consider an example where an order comes in that takes the on-hand inventory level below s. In the (s, Q) system, a fixed quantity Q would be ordered regardless of how much below s the inventory falls, so the NS would increase to $s + Q$ upon reorder. However, in the (s, S) system, the order quantity would be calculated as $S -$ current inventory level, so the net stock would increase to S upon reorder.

2.2.3 3. Periodic-Review, Order-Up-to-Level (R, S) System

The (R, S) system involves reviewing inventory at fixed intervals, R, and placing orders to raise the inventory level to a predetermined point, S. This method is particularly suitable in contexts where lead times are relatively predictable and consolidating orders is advantageous. Additionally, setting the target inventory level S necessitates a careful assessment of demand variability, the cost implications of holding excess inventory, and the potential for lost sales due to stock shortages.

In the figure below, the lead-time L is assumed to be fixed, and the reorder is put in every R units of time.

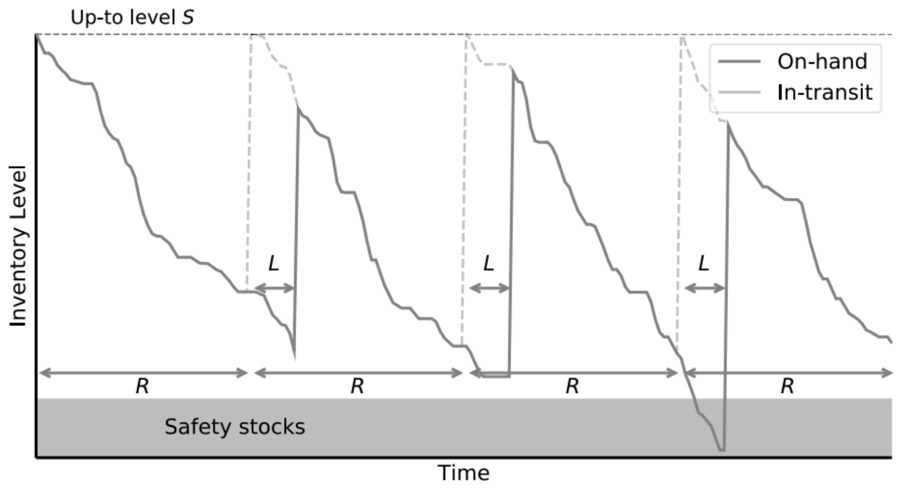


Figure 2-4: Example (R, S) Policy; from Vandeput and Makridakis [15]

2.2.4 4. Hybrid (R, s, S) System

The hybrid (R, s, S) system merges elements of continuous and periodic review strategies. Inventory is assessed at regular intervals R, but orders are placed only if levels fall below a specific point s. The order size adjusts to achieve a desired upper inventory level S. This model provides a versatile approach but practically finding the parameters to optimize this system may be significantly more complex than the other models[12].

2.3 Policy Selection in the General Case

Once a class of inventory models has been chosen, a key question remains of how to best select the policy parameters. Additionally, one might ask if such a policy is provably optimal. The class of policies discussed so far are all single-stage; i.e., there is only one decision point in the system, typically at the most downstream node. For single-stage systems under fairly weak assumptions, the (s, S) system is indeed optimal [9].

In a multi-echelon system, however, decisions need to be made at each stage or echelon, taking into account the inventory levels and demand patterns at the downstream stages. Echelon base-stock policies are a class of policies designed for multi-echelon systems. In these policies, each stage in the network maintains a base-stock level, which is the target inventory level for that stage. The base-stock level at each stage depends on the base-stock levels and the demand at the downstream stages.

When a demand occurs at the most downstream stage, it triggers a sequence of orders upstream. Each stage orders enough to bring its echelon inventory position (the sum of the inventory at that stage and all downstream stages, plus any outstanding orders) up to its base-stock level. The key feature of echelon base-stock policies is that they take into account the inventory levels and demand at all downstream stages when making ordering decisions at each stage. This is in contrast to single-stage policies, which only consider the inventory level and demand at the most downstream stage.

Echelon base-stock policies have been shown to be optimal in certain multi-echelon systems, such as serial systems with certain assumptions on the demand process and cost structure [2], [4]. In the general case, however, base-stock policies need not be optimal, and the optimal policy may be unknown or highly complex [13]. Indeed, Bertsimas and Thiele [1] give a simple example of a two-echelon supply chain network structure whose optimal policy is not base-stock. In more complex networks, heuristic or simulation-based methods may be the only viable option, especially with the introduction of constraints such as batch size requirements, "fair-share" rules for inventory distributions, and network capacity limitations [13] all of which are critical in Atlas's supply chain. It is therefore unsurprising that Atlas chose to build a parameterized heuristic approach to their middle-mile supply chain decision making, whereby simulation efforts could help tune the parameters.

2.4 A-B-C Classifications

Another important consideration in inventory management is the classification of items based on their importance or value. The A-B-C classification, also known as the Pareto principle or the 80/20 rule, is a widely used approach for this purpose. In this classification, items are divided into three categories:

A items: These are the most important items, typically constituting about 20% of the total inventory items but accounting for about 80% of the total value or importance. B items: These are items of intermediate importance, usually making up about 30% of the total inventory items and accounting for about 15% of the total value. C items: These are the least important items, constituting about 50% of the total inventory items but accounting for only about 5% of the total value. The A-B-C classification helps in focusing managerial attention and resources on the most important items. A items require tight control and frequent monitoring, while C items can be managed with simpler, less time-consuming methods. B items fall in between and require a moderate level of control.

Silver, Pyke, and Peterson [12] discuss the A-B-C classification in detail and provide guidelines for its application in inventory management. Their literature review finds that the classification should be based on the annual dollar usage of each item, which is the product of the annual demand and the unit cost, or other criticality measures [5], [3]. It is also recommended to perform periodic review of the classification, as the importance of items may change over time.

In the context of Atlas's supply chain, the A-B-C classification can be a useful tool for prioritizing inventory management efforts. By identifying the most critical items, Atlas can ensure that these items are closely monitored and that their inventory levels are carefully controlled. This can help to prevent stockouts of these critical items, which could have a significant impact on Atlas's operations and customer satisfaction. In the context of middle-mile allocation decisions, Atlas may seek to position its A products in distribution centers closer to its retail customers because of its high confidence in the ability to sell those items. Conversely, less critical or lower volume products should not take up valuable space in facilities close to end consumers, so those may be routed to a cheaper, central storage facility instead.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Design and Implementation of Middle-Mile Decision Engine

This section outlines the design and implementation of Atlas's inventory management policy for middle-mile STO decisions. It begins with the a detailed description of the software architecture, including the data inputs and parameters that get fed into the system, as well as the sequential decision making logic performed through simulation.

3.1 System Design Overview

Atlas's middle-mile STO (Stock Transfer Order) decision engine adopts a heuristic, rules-based approach for inventory management. This system is designed to simulate supply chain dynamics over some time horizon, using forecasted assumptions about demand, capacities, and lead times to sequentially roll out an STO plan. The actual logic of the system were presented as requirements by the business; the focus of this work is in a more generic heuristic implementation.

In some ways, the approach is similar to a model predictive control (MPC) framework where the solution to an optimization problem over some time horizon gives the control actions, all based on a model of the environment. The optimization problem to find the control actions, or STO decisions, that minimize total cost and meet demand requirements over thousands of SKUs during the simulation time horizon may be formulated as an integer program. In practice, the high dimensionality of this problem, coupled with additional (potentially nonlinear) constraints such as fair-share distribution requirements, led Atlas to adopt a heuristic, rules-based decision engine instead. That being said, optimization-based methods may be preferable to the rules-based system required by the business, so building out a benchmark policy based on optimization may be beneficial future work. Nevertheless, Atlas decided to use a heuristic sequential decision making algorithm rather than an optimization algorithm to produce a feasible set of actions to take. The details of this approach are covered in this chapter.

In this work, Atlas's model of its supply chain environment is assumed to be deterministic in all aspects except realized demand. That is, on a given day all lead times, incoming supply, and capacity limitations are assumed to be known and fixed. Point estimates for SKU level demand in each demand location are used in the allocation strategy, but a safety stock parameter controls how much additional inventory to position to ensure adequate service levels.

On each day, the engine evaluates the current state of the supply chain, taking into account the existing inventory levels and capacities, anticipated supply deliveries, full case pack requirements, and the demand forecasts for the immediate future. Using this information and certain input parameters, the system applies heuristic rules to sequentially allocate inventory. As such, the system may be described as a parameterized deterministic lookahead policy for sequential decision making [7]. These decision rules, covered in detail in chapter 4, are designed to strike a balance between various supply chain objectives, such as minimizing stockouts, reducing transportation costs, and ensuring fair distribution across demand lcoations. The decision process is iterative; the system continuously adapts to the daily fluctuations in demand and supply. The day's STO decisions are then executed, translating the theoretical allocation into actual inventory movements across the network. The following day, this decision-making cycle repeats, with the engine incorporating the latest data on demand realization, supply changes, and inventory levels. This approach allows for a responsive adaptation to the market, aligning inventory distribution with real-time operational realities. By eschewing the complexity of solving a full stochastic optimization problem daily, Atlas's heuristic approach provides a pragmatic and fast solution to its middle-mile STO decisions.

3.2 Demand Queue

The system's architecture is predicated on a push-based operational model, wherein a structured demand queue orchestrates the sequential allocation of inventory through simulation. This model is designed to anticipate and meet consumer demand by proactively distributing resources across the network.

The demand queue is a dataset that delineates the anticipated requirements across various dimensions. Each item or row in the queue specifies the number of units of a SKU that are demanded at a DL on a given day of the simulation. Moreover, each row includes an indicator field as to whether the demanded units are to meet immediate customer demand that day or are to meet desired safety stock levels. Unless explicitly stated otherwise, the term "demand" encompasses both types.

Within the theoretical framework of inventory categorization, the system employs a proprietary segmentation strategy, classifying products into distinct segments—$A, B, C, \ldots$ and T (Tail)—based on their respective demand attributes, specifically forecast accuracy and volume. This stratification is akin to the established $A-B-C$ inventory management methodology, wherein A items are characterized by high value but low turnover, B items by moderate value and turnover, and C items by low value but high turnover. The T segment in Atlas's system encapsulates Tail products, which are identified by their low volume and infrequent demand.

These product segments are used to order the demand queue and also serve as inputs for the system's algorithmic logic, which will be expounded upon in subsequent sections. Within this logic, products categorized within the Tail segment are typically allocated to the central distribution center (CDC), a strategy that mitigates the risk of excess inventory within the supply network. In contrast, products in the A segment—those with high volume and forecast confidence—are preferentially routed to regional service centers (RSCs). This targeted approach seeks to position high-confidence closer to consumption points to expedite fulfillment and enhance service levels.

The construction of the demand queue incorporates a "slicing" mechanism that breaks down the forecasted demand for each SKU into smaller equal segments. The demand queue's slicing mechanism ensures equitable inventory distribution by dividing the forecasted demand for each SKU into N equal segments by unit volume, where N is a configurable system parameter, and assigning each slice a number 1 through N. Next, the mechanism sorts all slices across all demand, so slices labeled 1 appear first in the queue, and slices labeled N appear last. This approach prevents inventory monopolization by any single Demand Location (DL) based on demand volume alone.

To give an example, consider a scenario where we have inventory of 100 units of a product and two DLs with demands of 150 units (DL1) and 50 units (DL2). Let us assume the system slices these demands into five segments ($N = 5$). Without slicing, DL1's higher demand might secure all 100 units. However, with slicing, DL1 has segments of 30 units each and DL2 has segments of 10 units each. The system allocates inventory to satisfy each DL's first segment before proceeding to the next, thus DL1 and DL2 would receive 20 and 10 units respectively from the first segment. This process continues until all inventory is allocated, ensuring both DLs receive inventory despite the difference in demand, with neither dominating due to their size alone. In this way, the push-based system maintains approximate service level equilibrium across all nodes of the supply network, adhering to the principles of fairness and strategic fulfillment for Atlas.

Over the course of the simulation, a new demand queue is constructed each day then sequentially fulfilled via certain decision making rules. Unlike many sequential decision making algorithms where actions are made during some policy roll out which affect and update present state, we took a decidedly different approach. As we step through each line item in the demand queue, our actions dictating how demand gets fulfilled may be made retroactively and actually impact past state. If $\{a_1, a_2, \ldots, a_n\}$ are a temporally ordered sequence of decisions made up to some time t, then we may

Table 3.1: Sample Demand Queue

Is Safety Stock?	**Slice**	**Demand Location**	**Product Code**	**Product Division**	**Segment**
N	1	A_DL	1901	B	0.5
N	1	A_DL	1902	A	1.5
N	1	B_DL	1901	B	1
N	1	B_DL	1902	A	2
N	2	A_DL	1901	B	0.5
...	...	...	...	...	...
Y	1	A_DL	1903	T	11.5

make some new decision a' at time $t' \leq t$ to fulfill the next line item in the demand queue so long as the decision sequence $\{a_1, a_2, \ldots, a', \ldots, a_n\}$ remains valid (i.e. does not violate any constraints or consume more resources than available).

Consider the scenario presented by the third line item in Table 3.1, which specifies a demand for 1 unit of product code 1901 at demand location A_DL. Given the operational framework of the system, the fulfillment process must account for the lead time associated with transferring inventory from a source node to the demand location. For illustrative purposes, let us assume a lead time of two days for transfers from the central distribution center (CDC) to the demand location A_DL.

In this context, the system's decision-making algorithm retrospectively evaluates the possibility of initiating a stock transfer order (STO) from the CDC to A_DL two days prior to the current simulation day. This retrospective assessment allows for the dynamic reallocation of resources to meet emergent demand patterns effectively. If, on reviewing the inventory status and transportation capacity two days ago, it is determined that sufficient resources were available and no constraints would be violated by the allocation of this STO, the system then proceeds to integrate decision a'—representing the STO for product code 1901 to A_DL—into the temporal sequence of decisions $\{a_1, a_2, \ldots, a', \ldots, a_n\}$.

This integration is contingent upon maintaining the validity of the entire decision sequence, ensuring that subsequent actions do not result in resource overconsumption or breach operational constraints. The next section covers the abstractions that allow the decision engine to efficiently track and manage these resources and constraints as the decision sequence gets built.

3.3 Class Architecture

This subsection covers a few high-level abstractions used in the decision engine design.

3.3.1 Resource

At the most atomic level, we have a **resource** object. A **resource** is any quantity, expressed in inventory units, that constrains one's ability to move inventory around the network. Each **resource** has a unique identifier, or key. For example, a lane capacity **resource** would have a key including the source node code, destination node code, and date. The number of units associated with this **resource** would reflect the outstanding capacity that Atlas has to STO inventory from the source to destination. The list of **resources** relevant to the decision engine are shown below:

Throughout, a **resource map** refers to a database or map, indexed by the key specified in Table 3.2.

3.3.2 Supply

Supply objects are abstract representations of how demanded units of a SKU in a demand location get fulfilled. Each **supply** can be represented by a collection of **resources** and has a unique supply identifier (**sid**).

The structure of a supply object includes several key components:

- **Unique Identifier (SID):** A composite key formed from relevant attributes, enabling specific identification.
- **Resource Collections:** Categorized into product-independent and product-dependent. The supply object stores the relevant resource keys for lookup.

Name	**Key**	**Description**
Inventory	(SKU, Node, Date)	Tracks inventory units of a SKU at a specific node on a certain date.
Pass-Through Inventory	(SKU, Node, Date)	Tracks inventory units of a SKU at a pass-through node on a certain date. A pass-through node is a node that cannot store inventory overnight, so unlike an `Inventory` resource, a `Pass-Through Inventory` resource cannot get carried over from one day to the next.
Lane Capacity	(Src, Dest, Date)	Represents the capacity of a transportation route between source (Src) and destination (Dest) nodes on a specific date, facilitating the movement of goods.
Inbound Processing Capacity	(Node, Date)	The maximum quantity of goods a node can receive and process for a given day.
Outbound Processing Capacity	(Node, Date)	The maximum quantity of goods a node can dispatch and process for shipment on a given day.
Receiving Node Storage Capacity	(Node, Date)	The maximum quantity of goods a receiving node can store as of a specific date, accounting for space limitations and inventory levels.

Table 3.2: Resources

  - **Product-Independent Resources:** Include capacities and constraints not varying with product specifics, e.g., `lane capacity` and `inbound processing capacity`.
  - **Product-Dependent Resources:** Tied to specific SKUs, such as `inventory` and `pass-through inventory`.

Since the number of products can be very high and the actual product SKUs in the network can vary over time, supply objects employ a dynamic key generation strategy for product-dependent resources. This method involves storing partial keys (e.g., date and node) and dynamically appending the product-specific component at runtime.

- **Excess Inventory**: When a supply object procures or consumes inventory in predetermined quantities (e.g., case packs), but the actual demand requires less than this quantity, the remainder is recorded as excess inventory. The excess inventory lookup table has each key represent a product and its associated value, initialized at 0, indicates the excess quantity of units for that product held by the supply object. For instance, if a supply object uses a full case pack of 12 units to fulfill a demand for 10 units, the remaining 2 units are stored within the excess inventory tracker. If the supply object is called on to fulfill additional units of demand for this product, it should first use these 2 excess units before consuming more of its resources.

The `Supply` object includes two key methods: `available` and `consume`.

Available Method The `available` method calculates the available quantity of a given product by considering both product-independent and product-dependent resources.

Algorithm 1 Pseudocode for `available`

```
 1: procedure AVAILABLE($product$)
 2:     $min\_avail \leftarrow \infty$
 3:     for all ($resource\_type, key$) in resource collections do
 4:         $resource\_map \leftarrow get(resource\_type)$   // Retrieves access to global data store
 5:         if $resource\_type$ is product-dependent then
 6:             $avail \leftarrow resource\_map[key, product]$
 7:         else
 8:             $avail \leftarrow resource\_map[key]$
 9:         end if
10:         if $avail \leq 0$ then
11:             return 0                    // Early stopping improves performance
12:         end if
13:         $min\_avail \leftarrow min(min\_avail, avail)$
14:     end for
15:     return $min\_avail$
16: end procedure
```

Consume Method Decrements the resources used to fulfill quantity of demand using the supply object.

Algorithm 2 Pseudocode for `consume`

```
procedure CONSUME($qty, product$)
    for all ($resource\_type, key$) in resource collections do
        if $resource\_type$ is product-dependent then
            $resource\_map[key, product] -= qty$
        else
            $resource\_map[key] -= qty$
        end if
    end for
end procedure
```

3.3.3 Four Types of Supply

The decision engine incorporates various types of supply objects, each tailored to specific logistics and inventory management scenarios within the network. The four primary types of supply objects are `OHSupply`, `STOSupply`, `CDFDivertSupply`, and `CDFPushSupply`. All inherit the supply base class attributes and methods mentioned previously.

1. OHSupply (On-Hand Supply) The `OHSupply` object represents fulfilling demand with inventory that is immediately available at a node. It is characterized by having inventory on-hand at a specific location (node) and date. This type of supply is crucial for meeting immediate demand without the need for additional transportation or processing, thus minimizing lead times and costs. As we shall see in later sections, the heurstic algorithm will always prefer to use on-hand supply to meet demand before searching through other supply objects. The only resource associated with this supply object is the `inventory` at the fulfilling node.

2. STOSupply (Stock Transfer Order Supply) `STOSupply` objects represent fulfilling demand with inventory that is moved from one node to another within the network, typically from a CDC to an RSC. These supply objects consider the lead time required to transport goods from the `src` node to `rcv` node.

3. CDFDivertSupply (CDF Divert Supply) The `CDFDivertSupply` is a specialized form, or subclass, of `STOSupply` designed for direct transfers to destination nodes from the cross dock facility. Since the CDF is a pass-through facility and cannot store inventory overnight, any inventory units received by the CDF that do not have a specified target destination get automatically "pushed" to the CDC (see `CDFDefaultSupply` below). On the other hand, the `CDFDivertSupply` allows demanded units of inventory to be met by routing arrived units at the CDF to a `rcv` node via STO, overriding the default behavior.

Importantly, this supply object has a `pass-through inventory` resource, which does not carry over its units each day. Roughly speaking, this ensures that if node v demands x units of product P on day t, the demand can be fulfilled by CDF supplies, but only if the CDF receives at least x units of product P on day $t - lt_{\{CDF,v\}}$ where $lt_{\{CDF,v\}}$ is the lead time between the CDF and v. Additionally, this supply object excludes outbound processing capacity considerations to reflect that all goods received by the CDF on a given day must be sent out.

4. CDFDefaultSupply (CDF Default Supply) The `CDFDefaultSupply` is also a specialized form, or subclass, of `STOSupply`. It facilitates the fulfillment of demand at a DL in a two-stage process: first inventory gets pushed to the CDC from the CDF, then gets routed to an RSC. Similar to the `CDFDivertSupply`, this object contains a `pass-through inventory` resource. The key difference, however, is the `CDFDefaultSupply` object represents a multi-leg fulfillment of demand. Therefore, it contains resources associated with the second leg of transportation as well, including source outbound capacity at the CDC, receiving inbound capacity at the RSC, receiving storage at the CDC, and lane capacity.

3.3.4 Deferred Decision Making for CDF Inflow

The architecture of supply objects within our system, notably `CDFDivertSupply` and `CDFDefaultSupply`, draws inspiration from the concept of 'lazy evaluation' in orchestrating the inflow of inventory at the CDF. We strategically defer the decision-making related to inventory processing paths at the CDF. Upon daily arrival, inventory at the CDF is presented with two primary destinational pathways: redirection to a Regional Service Center (RSC) via an STO, or, in the absence of a pre-determined destination, default forwarding to the CDC. The determination of these pathways is intentionally postponed, reliant on a assessment of demand and logistical evaluations conducted at future points within the simulation timeline. Inventory at the CDF exists in a provisional state—simultaneously tagged for CDC transfer while available for RSC redirection—until a conclusive decision is executed in simulation. This provisional state is maintained until either the simulation concludes (in which case the remaining unallocated CDF units get pushed to the CDC) or a specific demand during simulation catalyzes a retrospective allocation to a node.

This methodology facilitates an adaptable recalibration of inventory flows, responsive to the evolving landscape of demand patterns and logistical constraints during simulation. For instance, the decision to allocate inventory units arriving at the CDF on day t to an RSC (v) through an STO is predicated on a retroactive evaluation performed on day $t + lt_{\{CDF,v\}}$, where $lt_{\{CDF,v\}}$ signifies the lead time from the CDF to node v. Similarly, the decision to STO inventory arriving at the CDF to the CDC and then to an RSC is predicated on a retroactive evaluation performed on day $t + lt_{\{CDF,CDC\}} + lt_{\{CDC,v\}}$. In practice, out of the two options, it would always be preferable to directly transfer inventory to the RSC from the CDF as this would bring lower transportation costs (by triangle inequality, loosely speaking). The issue is, because the CDF cannot store inventory overnight, fulfilling demand at node v directly from the CDF is only feasible when demanded units at time t arrive at the CDF *exactly* at $t - lt_{\{CDF,v\}}$. On the other hand, the CDC serves as a storage buffer, so the multi-leg route can feasibly meet demand as long as inventory units arrive at the CDF at any time $t' \leq t - lt_{\{CDF,CDC\}} - lt_{\{CDC,v\}}$.

Note that for RSC (u) and RSC (v) (not necessarily distinct), there is no guarantee that $lt_{\{CDF,CDC\}} + lt_{\{CDC,u\}} > lt_{\{CDF,v\}}$. Consider a scenario whereby a demand line at the DL serviced by u comes earlier than a demand line at the DL serviced by v, for an identical product. If indeed $lt_{\{CDF,CDC\}} + lt_{\{CDC,u\}} \leq lt_{\{CDF,v\}}$ it may be possible that the first demand line is fulfilled via a multi-leg `CDFDefaultSupply`, consuming

CDF inventory, and making it impossible for the second demand line to be met via a direct transfer `CDFDivertSupply`. This example illustrates the heuristic allocation strategy prioritizes meeting demand sequentially in the simulation, and that there can be a tension between meeting earlier demand and minimization transport costs. This retroactive decision-making allows the system to perform inventory distribution by leveraging information and insights that were not available at the original time t.

3.4 Supply Queues

In addition to the demand queue, the system also utilizes supply queues, which together give matchings between demand locations and available supply sources through a coordinated allocation process. These queues can be thought of as priority queues which dictate the sequential order in which supply objects are evaluated and utilized to meet demand at a DL. Each priority queue is dynamically constructed based on a mapping that takes into account the specific DL, the nature of the demand (whether it is for safety stock or actual demand), and the product segment (categorized as A, B, C, ... T for tail).

Supply Priority Mapping The supply priority queue for each demand scenario is mapped as follows:

$$\underbrace{(DL, \text{Demand Type}, \text{Product Segment})}_{\text{Demand Scenario}} \rightarrow \text{Ordered List of Supply Objects}$$

This mapping ensures that for any given demand scenario, there exists a pre-defined, ordered list of supply objects to be sequentially tested for their ability to fulfill the demand. The algorithm traverses through this list, starting with the most preferred supply source and moving to the next option only if the current source cannot fully meet the demand. Ultimately, this is a greedy sequential matching with resource constraints.

Construction of the Supply Priority Queue The construction of the supply priority queue follows a deliberate order, prioritizing the use of on-hand inventory at the servicing RSC (`OHSupply`) as the first option. This preference aligns with the objective to minimize lead times and transportation costs by first utilizing inventory that is immediately available. Should the on-hand inventory at the RSC be insufficient or non-existent, the system then considers other avenues to fulfill demand.

Generally, for each demand scenario, two configurable inputs are used to construct the supply priority queue. The first is a prioritized list of fulfillment nodes for each DL, and the second is a prioritized list of supply objects used by each fulfilling node. For example, assume our demand scenario is characterized by actual demand (not safety stock) in some demand location DL_A for a non-tail product. Let DL_A be serviced by two fulfilling nodes, primarily by RSC$_A$ and secondarily by the CDC. Our inputs may look like the following:

1. **Fulfilling Node Priority List for DL:**
    - First preference: RSC$_A$
    - Second preference: CDC
2. **Supply Object Priority List for each Fulfilling Node:**
    - For RSC$_A$:
        (a) On-Hand Supply at RSC$_A$ (`OHSupply`)
        (b) Inventory transferred from CDC to RSC$_A$ (`STOSupply`)
        (c) Inventory diverted from CDF to RSC$_A$ (`CDFDivertSupply`)
        (d) Inventory pushed from CDF to CDC and then to RSC$_A$ (`CDFDefaultSupply`)
    - For CDC:
        (a) On-Hand Supply at CDC (`OHSupply`)

This example configuration indicates that the system first attempts to fulfill demand in the demand scenario with inventory already available at RSC$_A$. If this is insufficient, it sequentially considers inventory transfers from the CDC to RSC$_A$, directly diverting inventory from CDF to RSC$_A$, or using inventory that was initially pushed from CDF to CDC and then transferred to RSC$_A$. The CDC serves as a last resort, utilizing its on-hand inventory to fulfill the demand directly. This outputted supply queue can be visualized in Figure 3.1 below.

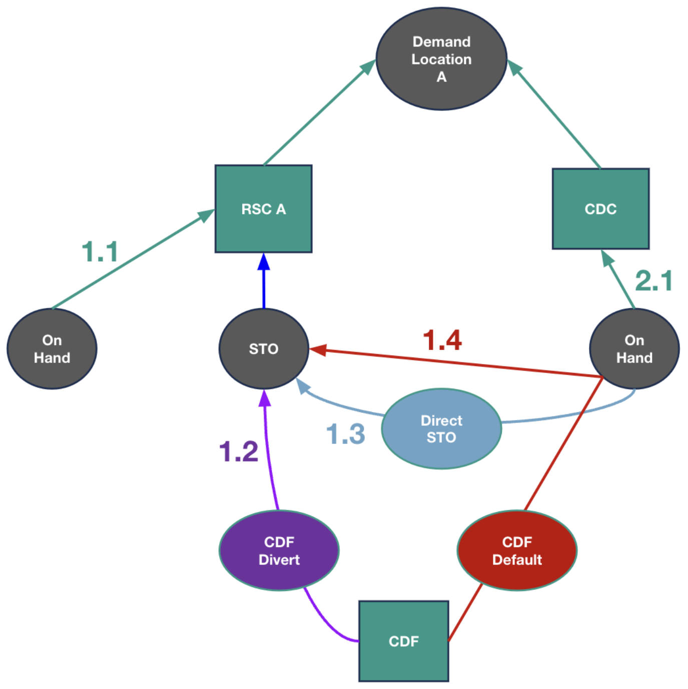


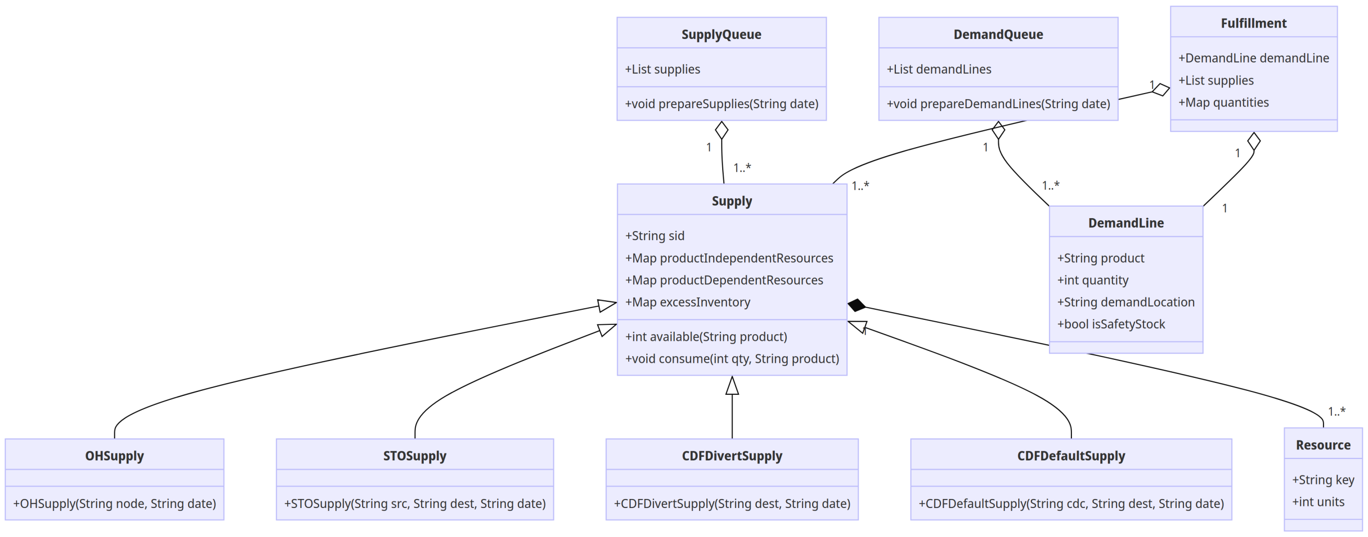
Figure 3-1: Example of Supply Queue

For each demand line with the demand scenario described above, the algorithm will traverse supply options [1.1, 1.2, 1.3, 1.4, 2.1] (as depicted in figure) and consume resources in each until demand is fulfilled or supplies are exhausted. For safety-stock demand scenarios, the fulfilling node will typically only consist of the respective RSC. Unlike in an actual demand scenario where the CDC can serve as a fulfillment node of last resort, demand for safety stock at a DL can only be fulfilled by an RSC. Beyond demand types, different product segments or DLs may have different supply queues. The actual inputs and construction of each supply queue is proprietary but Figure 3-1 provides a high-level illustration of "typical" behavior.

3.4.1 Daily Preparation of Supplies

At the start of each simulation day, a snapshot is taken of the state of all resources at the beginning of that day. This snapshot serves as a global shared data store of resources referenced by all supply objects. Iterating through each demand scenario, we construct the respective supply queues based on the input configurations; however, because the same supply object may appear in the supply queues of multiple demand scenarios, we make sure to pass shallow copies to preserve shared state. This will be necessary to track available resources as the sequential allocation progresses through different demand scenarios at different points in time.

3.5 Summary of Class Architecture



SupplyQueue
+List supplies
+void prepareSupplies(String date)
DemandQueue
+List demandLines
+void prepareDemandLines(String date)
Fulfillment
+DemandLine demandLine
+List supplies
+Map quantities
Supply
+String sid
+Map productIndependentResources
+Map productDependentResources
+Map excessInventory
+int available(String product)
+void consume(int qty, String product)
DemandLine
+String product
+int quantity
+String demandLocation
+bool isSafetyStock
OHSupply
+OHSupply(String node, String date)
STOSupply
+STOSupply(String src, String dest, String date)
CDFDivertSupply
+CDFDivertSupply(String dest, String date)
CDFDefaultSupply
+CDFDefaultSupply(String cdc, String dest, String date)
Resource
+String key
+int units
1
1..*

Figure 3-2: Class Diagram

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Allocation Logic, Matching Supply and Demand

Now that the high-level software design underpinning the supply chain representation has been discussed, we move onto the core middle-mile fulfillment algorithm, which will match supply to demand while respecting all resource constraints. This section covers how the greedy matching algorithm functions, emphasizing the mechanisms through which each demand line is assessed against a prioritized queue of supply options, the nuances of handling excess inventory, and the implications of full case pack logic. The algorithm returns a proposed list of fulfillments each day, where a fulfillment is defined as a mapping of a demand line to a collection of supply objects used to meet said demand, including the quantity contributed by each. This section will also cover how the proposed fulfillments get post-processed after each simulation day and converted into realized fulfillment decisions.

4.1 Middle-Mile Fulfillment Algorithm for Single Demand Line

This section outlines the demand and supply matching logic given a demand line and the supply queue corresponding to the demand scenario. To speed up the compute, any time a demand line cannot be successfully fulfilled by a collection of supply objects, a shortage is logged for the specific (DL, SKU) pair. This allows future demand lines with the same (DL, SKU) pair to perform early stopping and skip the search over the supply queue. By construction, actual customer demand is always prioritized in the demand queue before safety stock demand, and the set of supply objects that can fulfill safety stock demand is always a subset of the supply objects that can fulfill actual demand. Therefore, the set of supply objects able to fulfill for a (DL, SKU) pair can only decrease as we iterate through the demand queue, and indeed the early stopping approach is valid.

Additionally, the matching may take into account full case pack requirements. These requirements specify the integer multiple quantity of a product that must be sent together during an STO. It may be the case that an STO sends more than a demand line requests due to case pack rounding, which leads to excess inventory being transferred, or that a facility has inventory but not enough to send in a full case pack quantity. In the case when excess inventory is sent via an STO to fulfill a demand line, for future demand lines, if applicable, we will always prefer to use the excess sent before iterating through additional supplies. In the matching algorithm, source consumption is differentiated from receiving consumption, where the difference will be logged as excess. For fulfillments done using on-hand supplies, case pack multiple requirements do not apply.

Algorithm 3 Demand Fulfillment Process for Single Demand Line

```
 1: procedure FULFILL DEMAND(demand line, supply queue, shortages)
 2:     Initialize an empty collection of fulfillments, $F$.
 3:     Let quantity demanded from demand line be $Q$.
 4:     if shortage of (DL, SKU) $> 0$ then
 5:         Early stop, exit.
 6:     else
 7:         for each supply in the supply queue do
 8:             $E :=$ supply.excess_inventory(SKU)
 9:             $E_{consumed} :=$ max(0, Q-E)
10:             $Q = Q - E_{consumed}$                ▷ Reduce qty demanded by excess used
11:             $A =$ supply.available(SKU)
12:             Adjust quantities based on case pack sizes:
13:             Let full case pack quantity from demand line be $P$.
14:             $A^{-}_{case\_pack\_multiple} = P * \lfloor A/P \rfloor$        ▷ Round down to nearest case pack
15:             $Q^{+}_{case\_pack\_multiple} = -P * \lfloor -Q/P \rfloor$      ▷ Round up to nearest case pack
16:             $C_{source} = \min\{Q^{+}_{case\_pack\_multiple}, A^{-}_{case\_pack\_multiple}\}$ ▷ Consumed at src
17:             supply.excess_inventory(SKU) $\mathrel{+}= \max\{C_{source} - Q, 0\}$
18:             $C_{receiving} = \min\{C_{source}, Q\}$
19:             if $C_{source} > 0$ or $E_{consumed} > 0$ then
20:                 supply.consume($C_{source}$, SKU)
21:                 $Q = Q - C_{receiving}$        ▷ Reduce qty demanded by consumed at rcv
22:                 Log fulfillment to $F$.
23:             end if
24:             if $Q = 0$ (all demand is fulfilled) then
25:                 Exit the loop.
26:             end if
27:         end for
28:         if $Q > 0$ (not all demand could be fulfilled) then
29:             Increment shortage of (DL, SKU) by remaining unmet demand $Q$.
30:         end if
31:         Return the fulfillments, $F$.
32:     end if
33: end procedure
```

Algorithm 3 gives us the matching logic for a single demand line, and it what follows, we outline the demand fulfillment process for a single simulation day.

Algorithm 4 Demand Fulfillment Process

```
 1: Initialize a mapping for shortages, *shortages*.
 2: Read in data to initialize resources.
 3: for each date in simulation horizon do
 4:     Take snapshot of resources and build supply queues and demand queue.
 5:     for each demand line in the demand queue do
 6:         FulfillDemand(demand line, supply queue, shortages)
 7:         Collect fulfillment details.
 8:     end for
 9:     Update resources based on today's fulfillments.
10: end for
11: *Post-process collected fulfillments to get STO decisions for start day.
```

The final post-process step converts the collection of fulfillments over the entire simulation into the actionable STO decisions to be made at the start date of the simulation. In the production system, these will dictate the real STO decisions on the ground. The post-processing groups similar STO fulfillments together. The final recommended STOs from the algorithm consist of all the simulated STOs with send date equal to the simulation start date, as well as any leftover CDF inventory that will automatically get pushed to the CDC.

4.2 Repack Thresholds

One formal business requirement in the production system, was the introduction of a decision parameter called "repack thresholds." At a high level, repack thresholds are used to determine whether it is viable to proceed with an STO as initially planned or if adjustments are necessary to align with packaging or shipment size constraints and the operational thresholds for repacking at different nodes within the network. These thresholds are especially important in scenarios where the demand for a product falls below the case pack sizes, and due to case pack rounding, may lead to potential inefficiencies in shipping and handling. For example, if there is 0.5 units of predicted demand at a DL, but a case pack is 24 units, the matching algorithm may create an STO of 24 units which indeed satisfies demand, but may leave undesirable excess units at the receiving node that may accrue inventory holding costs. The business requirements stipulated for a post-processing step to the outputted STOs, whereby repack thresholds may modify or cancel these decisions.

The approach considers several scenarios for applying repack thresholds:

1. **Demand Lower Than Repack Threshold and Case Pack Size**: If the quantity demanded of an item is less than both the repack threshold and the case pack size, the system may cancel the STO or adjust the quantity, particularly if the source is an CDF. This decision is based on whether repacking is feasible or economically justifiable at the CDF. If not, such STOs might be redirected to the CDC or adjusted to avoid small, inefficient shipments that do not meet the repack threshold.

2. **Repack Threshold Lower Than Demand But Demand Lower Than Case Pack Size**: In situations where the demand is greater than the repack threshold but still below the case pack size, the algorithm might round down the STO quantity to the nearest repack threshold if repacking is an option at the source. If the source is the CDF and repacking is applied, any excess quantity not meeting the repack threshold may need to be pushed to the CDC due to the CDF's operational constraints on storing goods.

While a more holistic optimization framework taking into account inventory holding costs directly would likely be a more appropriate long term solution, the business considered this heuristic method of modifying STOs as a viable alternative. As will be seen in the following chapter, the simulation framework built to reproduce the production logic will lead to valuable insights into the behavior of repack parameters as well as other configurations of the middle-mile decision making process.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Key Takeaways and Simulation Results

The framework outlined in Chapters 3 and 4 gave Atlas the capabilities to quickly emulate the middle-mile decision making behavior of the production system prior to it going live. Upon completion, we integrated the middle-mile emulator into the company's end-to-end supply chain simulation framework which includes a much more granular last mile decision engine. While the middle-mile emulator seeks to fulfill aggregate demand in each DL during its rollout, the end-to-end simulator has a last mile fulfilment decision engine that fulfills at the individual digital order level. Therefore, in the final end-to-end testing simulations, the middle-mile emulator would still roll out a plan to determine each days' STO decisions, but the customer fulfillment component would much more closely resemble reality. Altogether, the end-to-end system would finally inform the company on key performance metrics under the new middle-mile STO policy, at least in simulation.

5.1 Lesson Learned from Software Procurement

Ideally, Atlas would have been able to test the production middle-mile system directly; however, this was not possible as testing support was not offered by the third-party provider, and we did not have access to the production source code. One key takeaway is the importance of carefully considering how how to mitigate technical risk while transitioning to a new software system, especially one built externally. Since Atlas is a large customer with bespoke requirements, it is understandable that a vendor might not have a readily accessible testing environment to match Atlas's business requirements. Nevertheless, Atlas would have benefited greatly from negotiating these terms upfront to include them as part of the deal. In general, any outsourcing results in some form of dependence, and by not securing an adequate level of support, the company was forced to built its own in-house emulation of the production system, resulting in an inefficient use of resources.

That being said, by starting from the ground up, we were able to build a much faster version of the emulator than the production system, and one that was interoperable with Atlas's end-to-end simulation framework.

5.2 Emulator vs Production System Run Time

System	**Run Time for Single Day's STO Decisions**
Middle-Mile Production System	5-6 hours
Middle-Mile Emulator	5-10 minutes

Table 5.1: Comparison of Run Time between Middle-Mile Production System and Emulator

As seen in Table 5.1, we achieved a 30x speedup over the production system. Even if we were to integrate the production system into Atlas's end-to-end supply chain simulator, it would be much too slow to be useful. When our middle-mile emulator was integrated into the end-to-end simulator, we were able to simulate each day's decisions in about 30 minutes, which includes granular last-mile fulfillment decisions in addition to the middle-mile STO decision making process. Thus, Atlas would be able to run different 6 month simulations on historical data with the new middle-mile logic in under a week, for example. This was fast enough to allow Atlas to test different configurations and measure performance.

It is unclear why we were able to achieve such a performance advantage versus the production system, as we did not have access to the latter. Notably, by continuously identifying the performance bottleneck using profiling tools, we were able to iterate on the emualtor to achieve speed gains. For example, early stopping during supply search, pruning out depleted supplies from the search tree, and moving away from Pandas dataframes, whose lookups are relatively expensive, to simple Python dictionaries, all helped improve performance. As a deterministic sequential decision making problem, very little of the problem could be parallelized. The performance issues boiled down to how to sequentially execute the logical and arithmetic operators as fast as possible, in accordance with the decision making rules. The emulator speed could be improved further if rewritten in a compiled language, as opposed to raw Python, but for the business's time and maintainability requirements, the emulation time achieved was sufficient.

5.3 Using the Emulator to Improve Production Logic

One key example of how the emulator was used to test and improve production logic was an analysis of the repack threshold, as described in section 4.2. A baseline scenario, which did not include any repack logic, was compared to a scenario with repack logic. The performance metrics collected in simulation are summarized in Table 5.2 below.

Metric	**% Change**
RSC Network Utilization	-2.0pp
RSC In-Region Fulfillment	-1.7pp
Total Cost	+0.6%
Last Mile Cost	+0.9%
CO2 (kg)	+1.6%
Split Orders	+4.2%
Ground Units	-0.1%
Air Units	+0.8%
Units Processed	-0.1%
Total Shipments	+0.6%
Multi-order count	-0.1%
Backlog qty over capacity qty	+24.7%

Table 5.2: Percentage Change, With Repack Compared to No-Repack Baseline

We see that with the repack configuration turned on, in-region fulfillment and capacity utilization got worse. Additionally, costs, carbon emissions, and split orders increased. Overall, this analysis was used to identify shortcomings with the repack logic that might have otherwise gone undetected. Based on this analysis, Atlas and the third-party vendor were able to modify the production logic accordingly to avoid these deleterious effects.

In general, Atlas can use the simulation framework to test different configurations of the middle-mile decision engine. Any of the parameters, such as how many slices to use in the demand queue, product segmentation strategy, and more can now be tested in simulation, and the aggregate performance impact can be captured, both on historical and theoretical data.

Chapter 6

Building an RL Environment to Supplement Supply Chain Decision Making

6.1 Problem Formulation

Looking forward, Atlas sought to build out their internal capabilities and knowledge of RL systems to determine how they could be used in supply chain decision making problems. A natural candidate to begin this exploration was to frame the middle-mile STO decision problem within the context of an RL framework. We took a simplified version of the real supply chain dynamics to create an environment and train an agent.

The Atlas Supply Chain Environment is conceptualized as a Markov Decision Process (MDP), formalized by the tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition probabilities, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, all of which will be defined in detail later in this section.

At a high-level, the supply chain network architecture (i.e. which nodes and lanes are present) is configurable in the construction of the environment. Additionally, the environment can be initialized with any number of SKUs, d; it is assumed the SKUs are known and fixed prior to runtime. Each time step in the environment corresponds to a single day. We chose to start with a simplified model depicted in Figure 7-1 below. Throughout, we refer to this model as Benchmark v1.

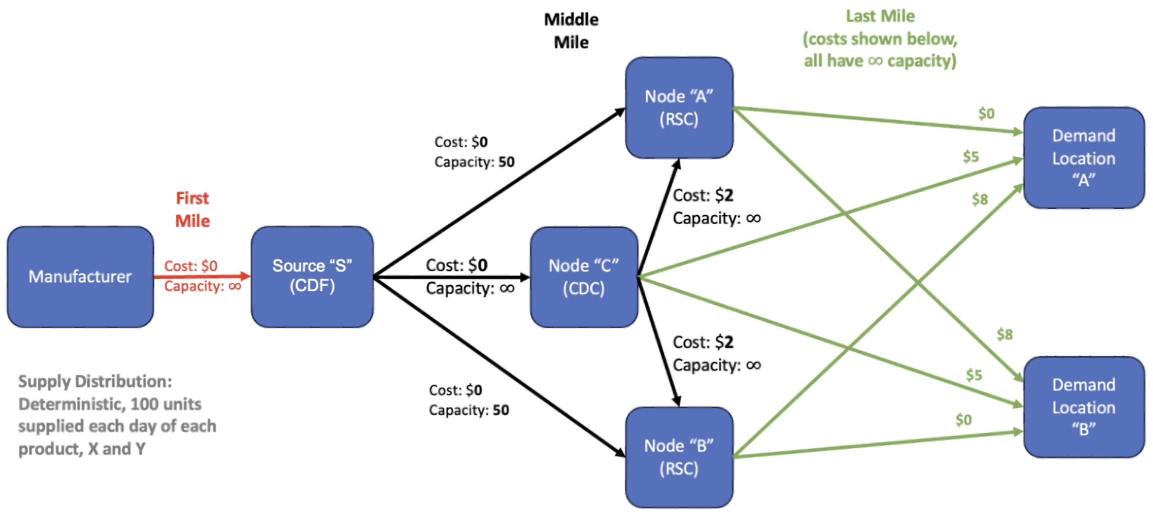


Figure 6-1: Example of Supply Chain Environment

The Benchmark v1 environment for the supply chain model includes the following key assumptions:

- There are two SKUs ($d = 2$), denoted as X and Y.

- The network consists of a single manufacturer, a source node (Cross Dock Facility, CDF), two Regional Service Centers (RSCs), and a Central Distribution Center (CDC).

- The supply from the manufacturer to the CDF is deterministic, with 100 units supplied each day for each product.

- The demand for each SKU at each DL follows a Poisson distribution whose rate parameter follows sinusoidal variations to reflect seasonal demand patterns. Conditioned on a SKU and time, the demand at each DL is identically independently distributed. Overall, the total expected demand, per day, over a one year period for every SKU is equal to the total daily supply of 100 units. Below is an example realization of the two products' aggregate demand over time in a simulation.

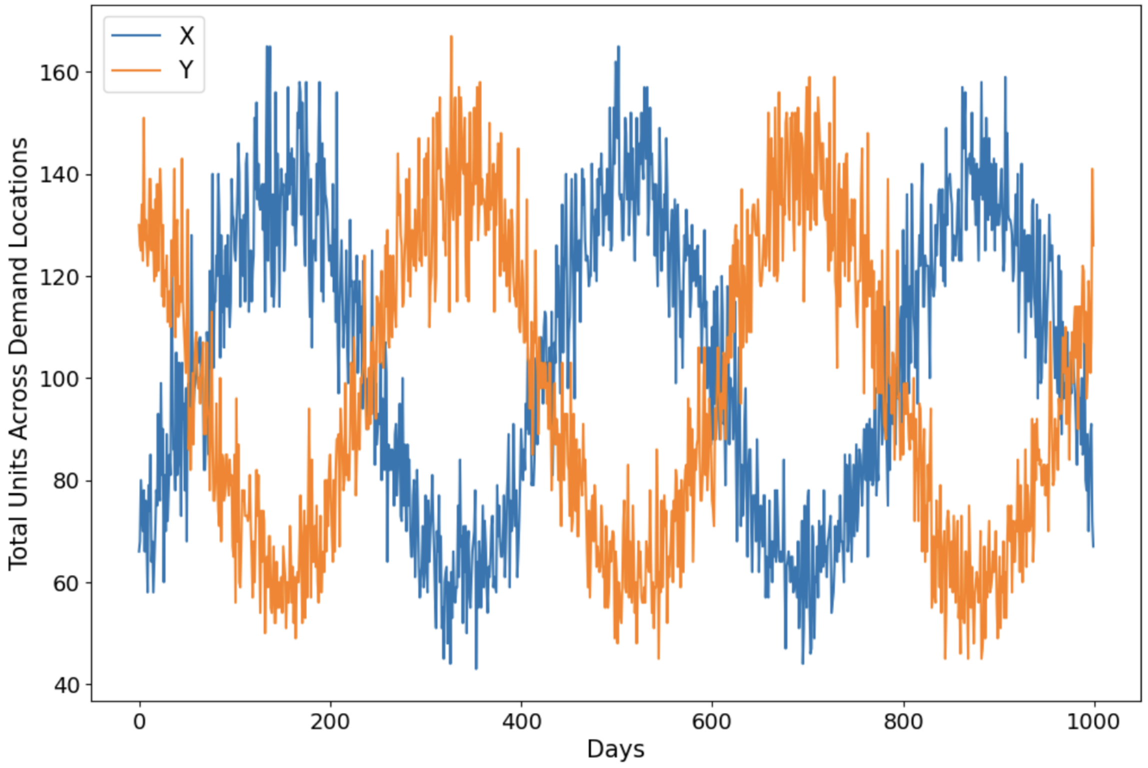


Figure 6-2: Realized Demand in Simulation

- Each lane has an associated cost for shipping per unit, capacity limitations on outbound processing on a given day, and predetermined fixed lead times, as shown in Table 7.1. The lead time of 0 from the manufacturer to the CDF is simply representing the idea that units will instaneously appear at the CDF according to the supply distribution, which in this case is a fixed 100 units. Notably, the 1 day lead time for each other lane represents the idea that an STO triggered on day t will be immediately reflected as inventory at the next time step, on day $t + 1$. However, the environment setup is flexible enough to consider arbitrary lead times, and this could be a path for future exploration.

The costs here are meant to reflect the idea that in-region fulfillment should be the cheapest option, followed by fulfillment from the central distribution center, then followed by cross-region transfers which are most expensive.

Table 6.1: Costs and Fixed Lead Times of Benchmark v1

Lane	Cost per Unit	Capacity	Lead Time (days)
Manufacturer to CDF (First Mile)	\$0	∞	0
CDF to CDC (Middle Mile)	\$0	50	1
CDF to Node "A" (Middle Mile)	\$0	∞	1
CDF to Node "B" (Middle Mile)	\$0	∞	1
Node "C" to Node "A" (Middle Mile)	\$2	∞	1
Node "C" to Node "B" (Middle Mile)	\$2	∞	1
Node "A" to Demand Location "A" (Last Mile)	\$0	∞	1
Node "B" to Demand Location "A" (Last Mile)	\$5	∞	1
Node "C" to Demand Location "A" (Last Mile)	\$8	∞	1
Node "A" to Demand Location "B" (Last Mile)	\$8	∞	1
Node "B" to Demand Location "B" (Last Mile)	\$0	∞	1
Node "C" to Demand Location "B" (Last Mile)	\$5	∞	1

6.2 State Space $\mathcal{S}$

To simplify the model, the real-world case pack and integrality constraints are ignored, and the state space is represented by a continuous domain capturing the inventory levels across the network for each SKU. Let us assume we have d SKUs in the model which are all known and fixed prior to runtime. At a given time t, each node has a d-dimensional state vector where each element represents the number of units of a particular SKU. Similarly, for each lane connecting two nodes u and v with lead time $lt_{u,v} > 1$, there exists a d-dimensional state vector for each s, $1 \leq s < lt_{u,v}$, where each element represents the number of units of a particular SKU s-days along in its transit journey from u to v. Then $\mathcal{S}$ is the concatenation of all such state vectors across node-level and transit-level inventories, which can be thought of as a $k \times d$ matrix for some k. Given the network configuration, the state space $\mathcal{S}$ can be represented as follows, where each row corresponds to an SKU (X or Y), and each column corresponds to the inventory position at the node or in transit location:

$$
\mathcal{S} = \begin{array}{c} \\ \text{Product X} \\ \text{Product Y} \end{array} \begin{array}{cccc} \text{Source (CDF)} & \text{Node A (RSC)} & \text{Node B (RSC)} & \text{Node C (CDC)} \\ \end{array}
$$

$$
\mathcal{S} = \begin{array}{c|cccc} & \text{Source (CDF)} & \text{Node A (RSC)} & \text{Node B (RSC)} & \text{Node C (CDC)} \\ \hline \text{Product X} & S_{CDF,X} & S_{A,X} & S_{B,X} & S_{C,X} \\ \text{Product Y} & S_{CDF,Y} & S_{A,Y} & S_{B,Y} & S_{C,Y} \end{array}
$$

Additionally, $\mathcal{S}$ contains 14 day forecasts of supply for each SKU and of demand for each (SKU, DL) pair. As implemented currently, these forecasts are simply the point estimates of the underlying sampling distributions which are assumed to be known. These forecasts may help the agent better plan for shifting demand dynamics.

In extensions to Benchmark v1, when the environment has lead times greater than 1 day, the columns may also include in-transit units at each day along each lane. As the state space scales linearly with the number of products, future implementations may want to take an approach to the state space that better scales with d. Lastly, we note the inventory at the manufacturer is assumed to be unlimited and thus is not explicitly modeled in the state space.

6.3 Action Space $\mathcal{A}$

The action space $\mathcal{A}$ is a continuous, multi-dimensional space where each action $a \in \mathcal{A}$ dictates the percentage of inventory outflow from a source node (CDF or any node holding inventory) to one or multiple destination nodes (RSCs or CDC). Actions are normalized to ensure that the total percentage of inventory allocated from a source to all destinations sums to 1, maintaining the conservation of mass within the system. Given the CDF's role as a pass-through facility without the capability to store inventory overnight, all inventory at the CDF must be allocated to subsequent nodes within the same time step.

An example of the action space matrix for two products, X and Y, across six possible lanes is illustrated below:

$$\mathcal{A} = \begin{array}{c} \\ \text{Product X} \\ \text{Product Y} \end{array} \begin{array}{cccccc} \text{CDF} \to \text{A} & \text{CDF} \to \text{B} & \text{CDF} \to \text{C} & \text{C} \to \text{A} & \text{C} \to \text{B} & \text{C} \to \text{C (CDC)} \\ \left[\alpha_{X,\text{CDF}\to\text{A}}\right. & \alpha_{X,\text{CDF}\to\text{B}} & \alpha_{X,\text{CDF}\to\text{C}} & \alpha_{X,\text{C}\to\text{A}} & \alpha_{X,\text{C}\to\text{B}} & \left.\alpha_{X,\text{C}\to\text{C}}\right] \\ \left[\alpha_{Y,\text{CDF}\to\text{A}}\right. & \alpha_{Y,\text{CDF}\to\text{B}} & \alpha_{Y,\text{CDF}\to\text{C}} & \alpha_{Y,\text{C}\to\text{A}} & \alpha_{Y,\text{C}\to\text{B}} & \left.\alpha_{Y,\text{C}\to\text{C}}\right] \end{array}$$

Where $\alpha_{SKU,\text{source}\to\text{destination}}$ represents the fraction of inventory for a given SKU dispatched from the source node to the destination node. For example, $\alpha_{X,\text{CDF}\to\text{A}}$ denotes the percentage of product X's inventory moving from the CDF to node "A" (RSC). It is important to note that the sum of the actions across all outgoing lanes from the CDF must equal 1, reflecting the full allocation of inventory from this pass-through node. On the other hand, inventory at node "C" (CDC) can either be distributed to RSCs or remain at the CDC, hence the inclusion of the $\alpha_{SKU,\text{C}\to\text{C}}$ term, which represents the percentage of inventory that remains at the CDC.

6.4 Transition Dynamics $P(s'|s,a)$

The transition dynamics within the environment capture the supply chain's evolution from one state to the next, incorporating both deterministic and stochastic elements. In Benchmark v1, the supply arriving at the CDF is deterministic, modeled as a fixed number of units for each SKU. In contrast, demand at each Demand Location (DL) is stochastic, following a Poisson distribution for each product at each DL.

At each time step t, the following events occur in sequence:

- **Supply Arrivals:** A fixed quantity, specifically 100 units, of each SKU is added to the CDF's inventory, representing incoming supply at end of day $t-1$.

- **In-Transit Advancements:** STOs that were triggered in previous time steps (on $t-1$ or earlier) advance through the lanes with fixed lead times. STOs that reach their destination are added to the respective node's inventory.

- **STO Triggers:** Based on the agent's actions, new STOs are triggered, moving inventory from source nodes to destination nodes or redistributing within the CDC. These are assumed to be set at the start of day t and occur before demand, and the corresponding fulfillment is realized throughout day t.

- **Last Mile Fulfillment:** Lastly, the environment's last mile fulfillment algorithm allocates units from RSCs and the CDC to meet the stochastic demand at the DLs. This algorithm functions independently of the agent's actions and operates in a greedy, sequential manner, as elaborated below.

Last Mile Fulfillment Algorithm The last mile fulfillment is not explicitly modeled within the action space. Instead, after the agent's actions are applied and the in-transit inventory is updated, a greedy sequential fulfillment algorithm is invoked. This algorithm prioritizes fulfilling customer demand at each DL using the following steps:

1. Compile a list of orders for each SKU based on the realized demand at each DL, which in Benchmark v1 is just a sampling from Poisson(t) where t varies seasonally. Aggregate all orders and randomly shuffle them.

2. Sort the available lanes to each DL by cost, in ascending order, to prioritize cheaper fulfillment options.

3. For each order, sequentially fulfill customer demand from the cheapest lane, using available inventory, until all orders are satisfied or inventory is exhausted.

4. If inventory at a node is insufficient to fulfill an order, the algorithm continues to the next best lane option.

5. This process continues until either all orders are fulfilled or there are no more lanes with available inventory to fulfill the remaining orders.

6. Orders that cannot be fulfilled result in missed sales, which can potentially impact the reward negatively depending on the reward design.

6.5 Reward Function $R(s, a)$

The reward function $R(s, a)$ quantifies the immediate cost or benefit incurred by taking action a in state s. In the context of our environment, the reward function is designed to capture the key objectives of the supply chain. Like many RL problems, reward design can have a large influence on agent's behavior, and the reward design in our environment is configurable to prioritize different objectives.

The reward at each time step may take the following into consideration:

- **Transportation Costs:** A cost is incurred for each unit of product shipped between nodes, which is subtracted from the reward. This cost is lane-specific and is proportional to the number of units shipped and the cost per unit of each lane.

- **Penalties for Lane Capacity Violations:** If the triggered STOs exceed the capacity of a lane, an additional penalty is applied, which is proportional to the excess volume shipped and a violation penalty factor. This relaxes the capacity constraints by turning them into costs in the reward function.

- **Sales Revenue:** A positive reward may accrue for each unit of product successfully delivered to a demand location, scaled by a sales reward factor. This factor can be tuned to represent the profit margin or the importance of sales revenue in the overall optimization objective. Note that last mile fulfillment occurs automatically as part of the transition dynamics of the environment and as such we do not need to explicitly promote fulfillment via a reward to see such behavior. Additionally, the incoming supply is exogenous to our STO decisions, so it may be unfair to penalize for missed sales as insufficient inventory under this framework is primarily a result of the random demand coming in above supply, which is outside of the agent's control. In Benchmark v1, we set the sales reward factor to 0 to focus on transportation costs associated with STOs, but we mention such a reward to showcase how this framework might be extended to include different facets of the overall supply chain objective, especially if we were to modify the action space to include first or last mile decisions as well.

For Benchmark v1, our reward on each step may be calculated as:

$$\underbrace{\sum_{l \in L_{\text{Middle-Mile}}} \sum_{p \in P} c_l \times f_{l,p}}_{\text{Middle Mile Costs}} + \underbrace{\sum_{l \in L_{\text{Last-Mile}}} \sum_{p \in P} c_l \times f_{l,p}}_{\text{Last Mile Costs}} + \sum_{l \in L} \phi \times \delta_l$$

where

- c_l : unit cost of transporting on lane l
- $f_{l,p}$: number of units sent of product SKU p through lane l
- ϕ : lane violation penalty per unit over capacity
- δ_l : units over capacity for lane l, 0 when under capacity (including when capacity is infinite).

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Policy Comparison for Supply Chain Decision Making

In this section we outline three classes of policies used to train on the Benchmark v1 environment and extensions.

7.1 PPO Policy

The first class of policies we explore for training on the Benchmark v1 environment is based on Proximal Policy Optimization (PPO) by Schulman et al. [10], a type of policy gradient method for reinforcement learning. PPO has gained popularity for its effectiveness in a wide range of environments and its relative simplicity compared to other algorithms. We employ the standard implementation from the Stable Baselines library, utilizing an MLP (Multi-Layer Perceptron) architecture for the policy network.

PPO operates by optimizing a surrogate objective function, which allows for multiple epochs of minibatch updates while ensuring the policy does not diverge too much from its previous iteration. This is achieved through the use of clipping in the objective function, which penalizes changes to the policy that move the ratio of the new policy probability to the old policy probability too far from 1.

7.1.1 Implementation Details

For the PPO policy, we configure the agent as follows:

- **Neural Network Architecture:** The policy and value functions are represented by separate MLPs, each consisting of two hidden layers with 64 units each. The activation function used is ReLU (Rectified Linear Unit) for hidden layers, with a linear activation for the output layer of the value network and a softmax activation for the policy network's output layer.

- **Learning Rate:** The learning rate is set to 3×10^{-4}, a common choice for PPO implementations that balances the trade-off between convergence speed and stability.

- **Clip Range:** The PPO clip range, which controls the clipping of the policy update ratios, is set to 0.2. This follows the standard practice in PPO implementations to prevent excessive policy updates that could lead to performance degradation.

- **Optimizer:** We use the Adam optimizer for training the neural networks, as it automatically adjusts the learning rate during training, providing a more stable and efficient optimization process.

The full set of default parameters for this implementation can be found in the Stable-Baselines3 documentation [8].

The PPO agent is trained over a series of episodes, each representing a sequence of decisions and transition steps within the Benchmark v1 environment. The goal is to maximize the cumulative reward, which in this context involves minimizing transportation costs and lane violation penalties.

7.2 Model-Based Control (MPC) Policy

The Model-Based Control (MPC) approach formulates the decisions needed to optimize our objective over a planning horizon as an LP. As the supply and demand distributions on each day are assumed to be known, even though the actual quantities are random, we take the expected values of these distributions as our inputs into the LP. We iteratively perform the following steps: solve the LP over some fixed planning horizon, apply the STO decisions given on the first day of the solution, realize actual supply and demand (sampled from our distributions), and apply the transition dynamics of our environment. At each decision step, this method seeks to minimize the expected total cost over the planning horizon. The optimization problem is subject to constraints that ensure actions are feasible with respect to the action space and the state transitions adhere to the supply chain dynamics.

The MPC policy is particularly powerful in an environment like ours, where the dynamics are partially known and predictable, allowing for effective planning. However, the computational complexity of solving the optimization problem at each step can be significant, especially as the planning horizon and the dimensionality of the action space increase (such as by including more SKUs). Below, we write out the full formulation.

Decision Variables

- $\texttt{inv}_{d,n,p}$: Inventory level of product p at node n on day d.

- $f_{d,s,r,p}$: Number of units sent of product p through lane from source node s to receiving node r on day d. Can be thought of as a network flow. The optimized middle-mile flow variables for the first day represent the STO decisions that will get input into the environment.

- $\delta_{d,s,r}$: Units over capacity for lane from source node s to receiving node r on day d, 0 when under capacity. Can be thought of as a slack variable.

- $\texttt{fulfilled}_d$: Total product quantities fulfilled on day d. Auxiliary variable used to simplify the reward formulation.

Parameters and Constants

- $\texttt{init\_inv}_{n,p}$: Initial inventory level of product p at node n. Will be set to same starting inventory as environment.
- $\texttt{supply}_{d,p}$: Forecasted daily supply of product p on day d, determined by taking the expected value under the environment's supply distribution.
- $\texttt{demand}_{d,DL,p}$: Forecasted daily demand of product p on day d, determined by taking the expected value under the environment's demand distribution.
- $c_{s,r}$: Unit cost of transporting goods from source node s to receiving node r.
- $\texttt{lane\_capacity}_{s,r}$: Capacity of the lane from source node s to receiving node r.
- β: Reward for fulfilling a unit of product.
- ϕ: Lane violation penalty per unit over capacity.

Objective Function

The objective is to maximize the total fulfillment reward minus transportation costs and penalties for lane capacity violations.

$$\text{Maximize} \quad \beta \cdot \sum_{d} \texttt{fulfilled}_d - \sum_{d,s,r,p} c_{s,r} \cdot f_{d,s,r,p} - \sum_{d,s,r} \phi \cdot \delta_{d,s,r}$$

Note that this is identical to the objective in the PPO formulation described in [6.5], except for the fulfillment reward. This is because the PPO agent operates directly on the environment where last mile fulfillment is built into the transition dynamic step, and as such, does not need a fulfillment reward term to exhibit this behavior. Conversely, the LP-based approach includes last mile fulfillment as part of the decision variables as it builds out a plan over the horizon timeline, and to encourage fulfillment, we must reward this behavior by choosing a sufficiently large β. Without this reward, the LP-approach would simply not flow any products through the network to avoid all transportation and capacity violation costs.

Constraints

1. **Initial Inventory Constraints**:

$$\forall n, \forall p, \quad \texttt{inv}_{0,n,p} = \texttt{init\_inv}_{n,p}$$

2. **Flow Conservation**: Difference in inventory must equate to inflow minus outflow.

$$\forall n, \forall d, \forall p, \quad \texttt{inv}_{d+1,n,p} - \texttt{inv}_{d,n,p} = \sum_{s} f_{d+1-lt_{s,n},s,n,p} - \sum_{r} f_{d,n,r,p}$$

where $lt_{s,n}$ represents the lead time from source s to node n.

3. **Outflow Limitation**: Cannot send out more than available.

$$\forall n, \forall d, \forall p, \quad \texttt{inv}_{d,n,p} \geq \sum_{r} f_{d,n,r,p}$$

4. **Supply Distribution**: All incoming supply must be fully distributed to nodes, as CDF cannot store inventory.

$$\forall d, \forall p, \quad \texttt{supply}_{d,p} = \sum_{r} f_{d,S,r,p}$$

5. **Fulfillment Calculation**: Auxilliary variable storing total fulfilled units each day.

$$\forall d, \quad \texttt{fulfilled}_{d} = \sum_{DL} \sum_{src} \sum_{p} f_{d,src,DL,p}$$

6. **Regional Demand Fulfillment**: Cannot fulfill more than demand for each DL.

$$\forall DL, \forall d, \forall p, \quad \sum_{src} f_{d,src,DL,p} \leq \texttt{demand}_{d,DL,p}$$

7. **Lane Capacity**: Lower bounds capacity overage. The objective penalizes

capacity overage so any optimal solution will indeed be set to this lower bound.

$$\forall s, \forall r, \forall d, \quad \sum_p f_{d,s,r,p} - \texttt{lane\_capacity}_{s,r} \leq \delta_{d,s,r}$$

$$\forall s, \forall r, \forall d, \delta_{d,s,r} \geq 0$$

7.3 Heuristic Policies

Heuristic policies may rely on simple, rule-based strategies to make decisions within the supply chain environment. Unlike the PPO and MPC policies, which either learn from interaction with the environment or optimize based on predictions, heuristic policies follow predetermined rules designed to capture domain-specific knowledge or intuition about effective strategies. The Benchmark v1 environment is simple enough to create reasonable rules-based approaches for STO decision making, described below:

Table 7.1: Heuristic Policies for STO Decisions

STO	Policy 1	Policy 2	Policy 3
S → A	25%	33%	0%
S → B	25%	33%	0%
S → C	50%	33%	100%
C → A	50%	33%	50%
C → B	50%	33%	50%
C → C	0%	33%	0%

Heuristic policies can be quickly implemented and provide a baseline against which the performance of more complex policies like PPO and MPC can be compared.

7.4 Training and Evaluation

Training of the PPO agent involves iteratively improving its policy by interacting with the environment, collecting trajectories of states, actions, and rewards, and then updating the policy network based on the observed returns. We evaluate the performance of the PPO policy both during and after training by measuring the average cumulative reward over a set of evaluation episodes. This allows us to track the agent's learning progress and assess its ability to make cost-effective supply chain decisions.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 8

Policy Performance and Key Takeaways

The policies mentioned in the previous chapter will each be run on the same Benchmark v1 environment. The results will be presented and discussed. Additionally, this chapter will discuss key findings and potential generalizations of our RL approach.

8.1 Simulation Performance Comparison

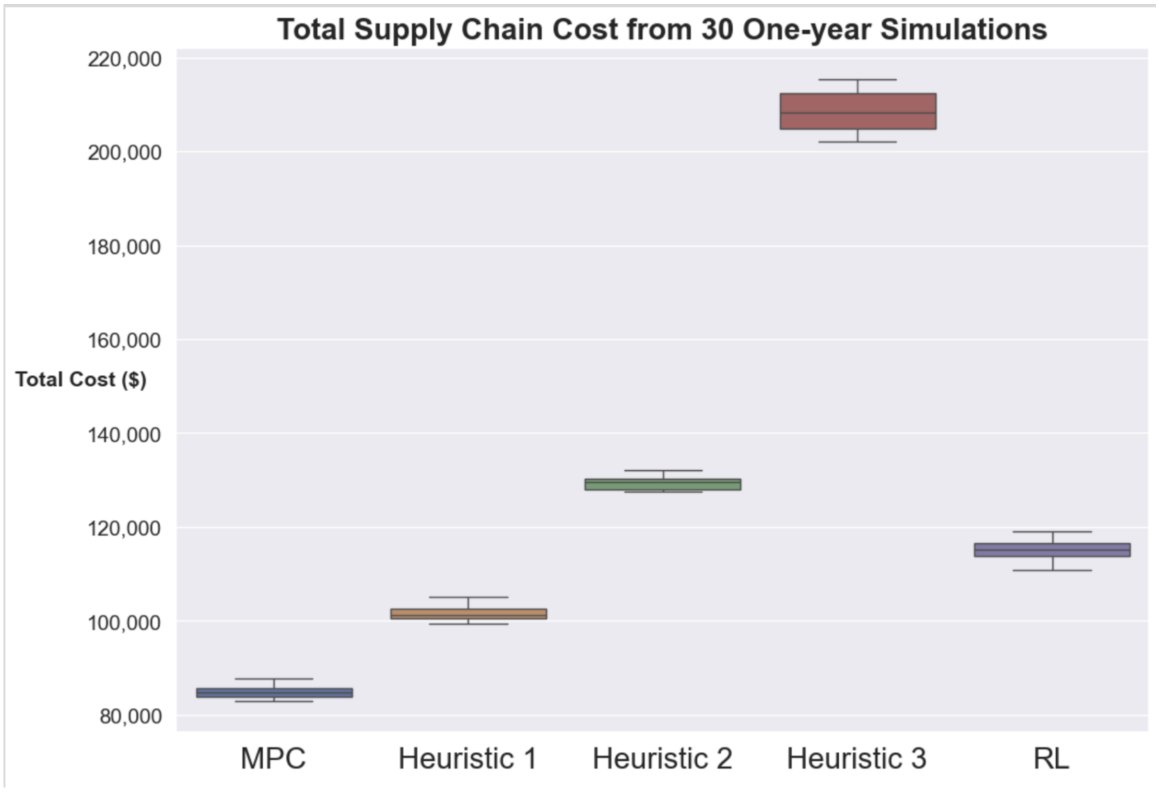
To compare the performance of the PPO RL policy, MPC policy, and three heuristics, as described in Chapter 7, we ran 30 one-year simulations for each policy in our environment. The PPO agent was trained prior to the test simulations on the same environment. While each policy may have different reward functions (or none at all in the case of heuristics), the evaluation metric will be on total cost incurred by transportation and lane violation penalties, as accrued in the actual environment. We can also convert this performance metric to cost on a unit basis by dividing total cost by the number of units supplied into the system. In the Benchmark v1 environment, over 1 year, we expect $2 \times 365 \times 100 = 73,000$ units to enter the supply chain, so we arrive at an approximate unit cost by dividing total simulation cost by this figure. The results of the simulation are summarized in Table 8.1.

Additionally, a box plot of all 30 runs is shown below, demonstrating low variance

Table 8.1: Median Total Cost Across 30 Simulations

	MPC	Heuristic 1	Heuristic 2	Heuristic 3	RL
Median Total Cost	\$84,935	\$101,768	\$129,285	\$208,404	\$115,211
Median Unit Cost (approx.)	\$1.16	\$1.39	\$1.77	\$2.85	\$1.58

in policy performance.



The MPC policy performed best, followed by Heuristic 1, which sends 50% of supply to the CDC and 25% to each CDC. Since the lane capacities from the CDF to each RSC are taken to be 50 units in Benchmark v1, and 200 total units are supplied across both products in expectation each day, it is sensible that about $\frac{50}{200}$=25% of inventory should be diverted to each RSC to minimize transportation costs without violating lane capacity. Ignoring lane capacities, in this environment, it is almost always preferable to bypass the CDC completely and ship directly to RSCs to achieve lower cost in-region fulfillment. The sole, and rare, exception would be when a large deviation above expected demand is realized at one region, depleting its in-region inventory, resulting in fulfillment from the other region, which is more expensive than fulfillment from the

CDC. So in this case, having some safety stock at the CDC may provide an effective buffer and reduce expensive cross-region fulfillment. Nevertheless, for the parameters and distributional assumptions chosen in Benchmark v1 environment, Heuristic 1 seems to perform reasonably well, and most closely matches human intuition.

Unlike the simple heuristics, the MPC policy can take the current inventory state and supply and demand forecasts into account when making STO decisions. As such, it is perhaps unsurprising that this policy outperforms all heuristics. Conversely, the "black-box" PPO policy does not build out a plan over some time horizon or have "intuition" about the right rules to apply, like a good heuristic might. Considering the lack of interpretability and worse performance than the MPC and Heuristic 1 in simulation, Atlas employees rightfully asked whether RL-based methods were viable for their supply chain problems. While the answer to this is unclear and will require additional work at the company, it is important to note that on simple problems with a low amount of uncertainty and simple transition dynamics, RL-based approaches may not fully exploit the known information and underperform very basic policies. For example, consider the game of Tic-Tac-Toe. While RL-approaches could certainly learn an optimal strategy for this game, it is readily clear to anyone who has played, that a rules-based approach is sufficient to play optimally. For a game like chess, however, with much greater complexity, deep reinforcement learning techniques, such as those used by Alpha Zero[11], provide greater performance than any rules-based approach. Point being, while Benchmark v1 is an interesting minimally sufficient environment to resemble the true problem and test different policies, it would be interesting to see how these different approaches scale as the environment gets more realistic and complex.

For example, with thousands of SKUs and dozens of constraints, it is unclear what a good heuristic policy might look like. The LP-based approach may face performance issues if scaled to the full problem, depending on modelling assumptions. Assuming a supply chain network has 10 edges, lead times of 5 days between nodes, 1,000 products, and a 100 day time horizon, this gives us on the order of 5 million decision variables, which may certainly be feasible with modern solvers, but additionally complexities, such as integrality constraints, fair-share distribution requirements, stochastic lead times, or new constraints, may prove less tractable.

8.2 Interpreting RL Policy Behavior

One might be interested to see the emergent behaviors of the PPO agent as it trains. Initially, we hypothesize the large negative penalty term for lane capacity violations dominates and causes the agent to favor sending inventory directly to the CDC which is uncapacitated. However, as the agent continues to learn, it then starts to ship roughly 25% of inbound volume to each RSC, as done in Heuristic 1.

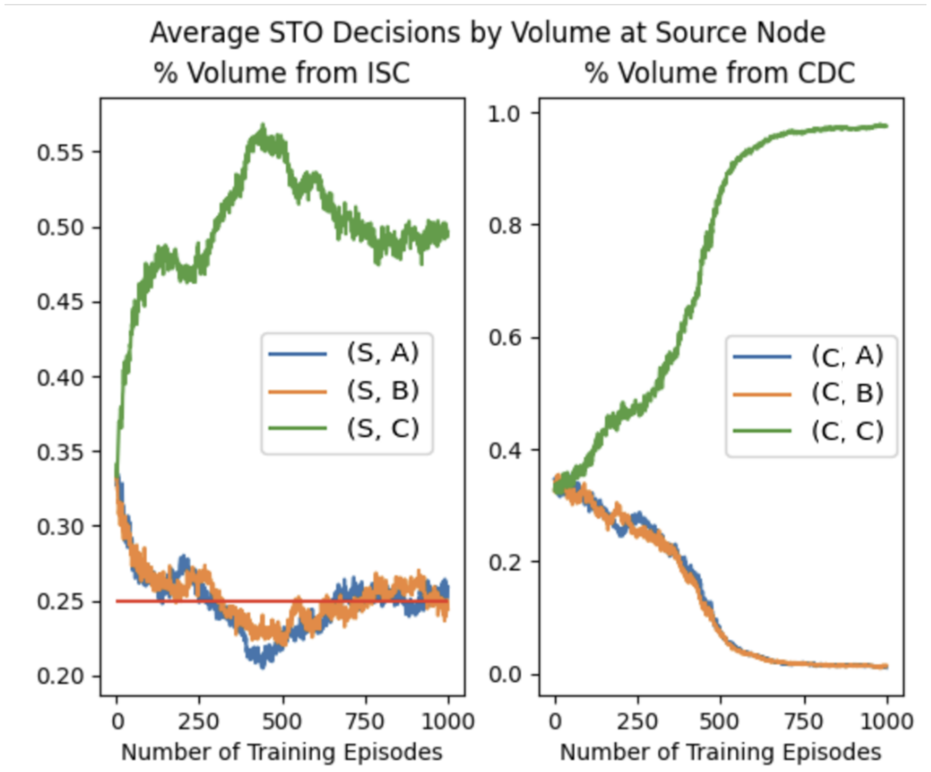


Figure 8-1: STO Decisions

The idea that the agent first minimizes lane penalty cost is reflected in Figure 8.2 below. The three components of cost, middle-mile transport, last-mile transport, and lane violation penalty, are shown, along with aggregate cost.

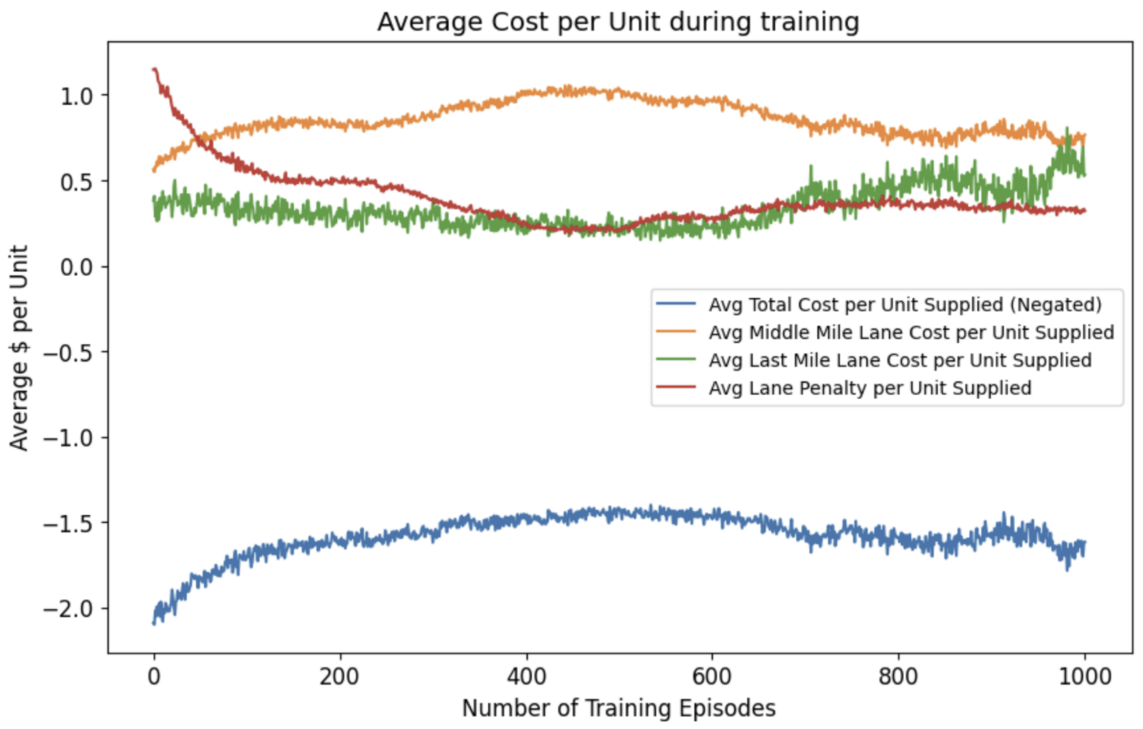


Figure 8-2: Cost Breakdown

Building analytical tools to capture policy decisions throughout training has been a value-add to Atlas as they continue to experiment with this technology. These tools may provide greater interpretability to the agent's decision making. Moreover, the art of reward design becomes more approachable when such tools allow a decision maker to view the behavioral and reward impacts of their reward function.

8.3 Future Work and Considerations

There is still a great deal of work to be done in incorporating constraints into the RL formulation. The simple penalty approach here may not scale to additional constraints, and future work should consider formulations which directly respect resource constraints or dynamically tune cost parameters during training. Moreover, the state and action space scale linearly with the number of products, and for thousands of SKUs, the agent may suffer from a curse of dimensionality. Multi-agent RL with partially shared state of the supply chain may be an interesting avenue to explore. Additionally, the action space chosen here may not be the most appropriate to model

STO decision making; rather than considering the percentage of inventory that should be allocated to each node, perhaps an action space that models unit flow directly or some other parametrization would give better performance.

Going forward, Atlas has a solid foundation to build upon with the environment and agent developed in this work. This baseline provides a valuable starting point for further exploration and refinement of RL-based approaches in the context of their supply chain.To fully leverage the potential of RL, Atlas should consider the following steps:

1. Continual benchmarking: Regularly compare the performance of the RL agent against existing heuristic methods and other optimization techniques. This will help identify areas where the RL approach is providing superior results and where it may need improvement.

2. Iterative refinement: Based on the benchmarking results, iteratively refine the RL model. This may involve adjusting the state and action spaces, reward functions, or model architectures to better capture the nuances of Atlas's supply chain and decision-making processes.

3. Explainable AI: Invest in techniques to make the RL agent's decisions more interpretable and explainable. This will help build trust in the model and allow supply chain managers to gain insights from the agent's learned strategies.

4. Hybrid approaches: Explore hybrid approaches that combine the strengths of RL with traditional optimization methods. For example, RL could be used to learn high-level strategies while optimization techniques handle the detailed allocation decisions.

5. Simulation and stress-testing: Utilize the RL environment to simulate various scenarios and stress-test the agent's performance under different conditions. This can help identify potential weaknesses and improve the robustness of the decision-making process.

6. Gradual deployment: Gradually integrate the RL agent into the decision-making process, starting with low-risk decisions and progressively increasing its responsibilities as it demonstrates reliable performance. By following these steps, Atlas may potentially use RL to uncover novel strategies and emergent behaviors that may lead to improvements in supply chain performance. The insights gained from the RL agent can complement and enhance existing decision-making processes, ultimately driving better outcomes for Atlas and its customers.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 9

Conclusion

The landscape of supply chain management has undergone a significant transformation in recent years, driven by the rapid growth of e-commerce, the increasing complexity of global networks, and the need for agility in the face of market volatility. As a result, large-scale retailers like Atlas are seeking innovative solutions to optimize their inventory management policies and streamline their operations. This thesis has explored the application of advanced decision-making techniques, specifically focusing on the development of a high-performance emulator for middle-mile supply chain decisions and the potential of reinforcement learning (RL) in enhancing inventory management strategies.

The first part of this thesis addressed the challenges encountered by Atlas in implementing a third-party middle-mile Stock Transfer Order (STO) decision engine. The lack of a testing environment, complex configurations, and runtime performance issues prompted the development of a high-performance, exact logical replica of the decision engine. By integrating this emulator into Atlas's end-to-end supply chain simulation framework, we achieved a notable 30x speedup compared to the existing system. This enhancement enables Atlas to efficiently test various configurations, uncover critical unexpected behaviors, and make data-driven decisions to optimize their supply chain operations. The practical implementation of this decision engine not only significantly reduced run times but also allowed Atlas to proactively address potential operational challenges that could have arisen in a live production environment.

The second part of this thesis explored the potential of RL in augmenting or replacing Atlas's middle-mile decision-making policy. By framing the supply chain as an RL problem and building a simplified environment that incorporates key elements of the real supply chain network, we developed and trained a parametrized inventory management policy. The performance of this RL policy was evaluated through simulation and benchmarked against traditional heuristic and model predictive control policies. While the RL policy did not outperform these alternatives in the simplified environment, this work provides a foundation for Atlas to explore RL applications as they scale to more realistic supply chain environments. As Atlas continues to explore the integration of RL-based approaches into their decision-making processes, they will need to address the challenges of incorporating complex constraints, improving learning efficiency, and ensuring scalability to handle the high dimensionality of their supply chain network.

In conclusion, this work aims to provide a valuable case study of the practical application of advanced decision-making techniques in a large-scale retail environment. We hope to contribute to the growing body of knowledge on the practical implementation of advanced decision-making techniques in real-world business settings. As retailers navigate the challenges of an increasingly complex and dynamic supply chain landscape, the insights and methodologies presented in this thesis may serve as a useful reference for practitioners and researchers alike.

Bibliography

[1] Dimitris Bertsimas and Aurélie Thiele. "A Robust Optimization Approach to Inventory Theory". en. In: *Operations Research* 54.1 (Feb. 2006), pp. 150–168. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.1050.0238. URL: https://pubsonline.informs.org/doi/10.1287/opre.1050.0238 (visited on 05/12/2024).

[2] Andrew J. Clark and Herbert Scarf. "Optimal Policies for a Multi-Echelon Inventory Problem". en. In: *Management Science* 6.4 (July 1960), pp. 475–490. ISSN: 0025-1909, 1526-5501. DOI: 10.1287/mnsc.6.4.475. URL: https://pubsonline.informs.org/doi/10.1287/mnsc.6.4.475 (visited on 05/12/2024).

[3] Benito E. Flores and D.Clay Whybark. "Implementing multiple criteria ABC analysis". en. In: *Journal of Operations Management* 7.1-2 (Oct. 1987), pp. 79–85. ISSN: 0272-6963, 1873-1317. DOI: 10.1016/0272-6963(87)90008-8. URL: https://onlinelibrary.wiley.com/doi/10.1016/0272-6963%2887%2990008-8 (visited on 05/13/2024).

[4] Stephen C. Graves. "A Multiechelon Inventory Model with Fixed Replenishment Intervals". en. In: *Management Science* 42.1 (Jan. 1996), pp. 1–18. ISSN: 0025-1909, 1526-5501. DOI: 10.1287/mnsc.42.1.1. URL: https://pubsonline.informs.org/doi/10.1287/mnsc.42.1.1 (visited on 05/12/2024).

[5] James A G Krupp. "Measuring inventory management performance". In: *Production and inventory management journal : the journal of the American Production and Inventory Control Society, Inc.* 35.4 (Dec. 1994). Place: Falls Church, VA : Publisher: The Society, pp. 1–. ISSN: 0897-8336.

[6] Steven Nahmias and Tava Lennon Olsen. *Production and operations analysis: strategy, quality, analytics, application.* eng. Seventh edition. One learns by doing. Long Grove, Illinois: Waveland Press, Inc, 2015. ISBN: 978-1-4786-2306-9.

[7] Warren B. Powell. "Designing Lookahead Policies for Sequential Decision Problems in Transportation and Logistics". In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), pp. 313–327. ISSN: 2687-7813. DOI: 10.1109/OJITS.2022.3148574. URL: https://ieeexplore.ieee.org/document/9702124/ (visited on 05/13/2024).

[8] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[9] Herbert Scarf. *The Optimality of (S,s) Policies in the Dynamic Inventory Problem.* Technical Report 11. Stanford University, Office of Naval Research, Apr. 1959.

[10] John Schulman et al. "Proximal Policy Optimization Algorithms". In: (2017). Publisher: [object Object] Version Number: 2. DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347 (visited on 05/13/2024).

[11] David Silver et al. "Mastering the game of Go without human knowledge". en. In: *Nature* 550.7676 (Oct. 2017), pp. 354–359. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature24270. URL: https://www.nature.com/articles/nature24270 (visited on 05/13/2024).

[12] Edward Silver, David Pyke, and Rein Peterson. "Inventory Management and Production Planning and Scheduling (Third Edition)". In: *Journal of The Operational Research Society - J OPER RES SOC* 52 (Jan. 2001), pp. 845–845.

[13] David Simchi-Levi and Yao Zhao. "Performance Evaluation of Stochastic Multi-Echelon Inventory Systems: A Survey". en. In: *Advances in Operations Research* 2012 (2012), pp. 1–34. ISSN: 1687-9147, 1687-9155. DOI: 10.1155/2012/126254. URL: http://www.hindawi.com/journals/aor/2012/126254/ (visited on 05/12/2024).

[14] *Statista - Apparel & Shoes.* en. URL: https://www.statista.com/markets/415/topic/466/apparel-shoes/ (visited on 05/12/2024).

[15] Nicolas Vandeput and Spyros G. Makridakis. *Data science for supply chain forecasting.* eng. 2nd edition. Business & economics. Berlin Boston: De Gruyter, 2021. ISBN: 978-3-11-067110-0.

[16] Yimo Yan et al. "Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities". In: *Transportation research.* 162 (June 2022). Place: Exeter, England : Publisher: Elsevier Science Ltd, ISSN: 1366-5545.