Deep Reinforcement Learning for the Stochastic Joint Replenishment Problem with Non-Zero Lead Time

Jonathan Nicklin

University of Twente

Faculty of Behavioural Management and Social Sciences

Industrial Engineering and Management

05-06-2025 – Enschede, Netherlands

UNIVERSITY OF TWENTE.

Master thesis Industrial Engineering and Management

Deep Reinforcement Learning for the Stochastic Joint Replenishment Problem with Non-Zero Lead Time

Author:
J.M. Nicklin (Jonathan)
j.m.nicklin@utwente.nl

University of Twente
Drienerlolaan 5
7522NB Enschede

Supervisor Ahold Delhaize
P. Meints (Pieter)

Supervisors University of Twente
Prof. Dr. Ir. M.R.K. Mes (Martijn)
Dr. D. Prak (Dennis)

Preface

Dear reader,

At the precipice of a new chapter, I find myself reflecting on whether my past self would feel accomplished by all that I have achieved during my academic journey. Over the course of the five or so years living in the Netherlands, I've had my fair share of experiences which have transformed me into the person I find myself as today, and I feel confident that the version of me who started university would be filled with excitement at the prospect of his upcoming adventure.

Remarkably, my experience is only brought to life by the people I have crossed paths with along the way. Firstly, I would like to thank Martijn and Dennis, my university supervisors on this project, for their commitment to providing constructive comments and enjoyable conversation. While not directly named as part of this project, I would also like to thank Fabian – who beyond treating me as a friend – indisputably facilitated the implementation of the ideas presented in this work and would always generously offer time at a moment's notice.

Further, I would like to appreciate those at Ahold Delhaize who have made me feel welcome from the get-go. Importantly, as my manager and company supervisor, I would like to thank Pieter for not only facilitating this opportunity but also for his style of management that has provided me with the autonomy and support to bring the best out of me. Further, I would like to thank Elles and Ivar for their vested interest in my work, enabling me to share and test ideas with their keen eye for detail and domain understanding.

Lastly and certainly not least, I would like to thank my friends and family for their integral support, and I am privileged to have these people by my side when entering this new stage of my life. For all those who have come and gone, I thank you too.

Jonathan

Zaandam, the Netherlands

Management Summary

This paper addresses the Stochastic Joint Replenishment Problem (SJRP) with non-zero lead times using a real-life business case in collaboration with Ahold Delhaize Inbound Logistics (ADIL). The existing system used within ADIL, while it is functional for high-volume products, is limited in efficiency, particularly when ordering from Asian suppliers with long lead times. As a result, these inefficiencies have led to underutilised container capacity, increased costs, excessive use of human resource and a reduction in service levels.

As a consequence, this research explores the use of Deep Reinforcement Learning (DRL) to develop a neural network-based ordering policy. By leveraging Deep Controlled Learning (DCL), an approach is outlined to autonomously generate a cost-effective ordering solution that balances transportation, holding and backordering costs in determining how to fill a single shipping container.

In testing the approach, a small-scale 4-item problem and a larger 47-item problem were used. In the small setting, DCL significantly outperformed a traditional rule-based policy (can-ordering) and another DRL approach (PPO), which has been shown to generate competitive solutions in similar, simplified problems. In doing so, the long-horizon evaluation showed DCL to achieve higher fill rate, lower periodic cost, and better container utilisation. Further, it demonstrated its ability to adapt to a change in demand levels seen between training and test datasets. In the larger setting, however, the computational cost of generating solutions led DCL to fail in producing similar levels of performance.

Managerially, the findings suggest that neural network-based policies hold promise for improving replenishment practices within organisations. While a glimpse of its potential is evident from the small-scale setting, the approach is infeasible for use across a wider assortment of products in its current form. In terms of ADIL, this implies that while DRL may be effective in generating orders for a selection of its assortment, broader adoption would require adaptations which suit the ordering of more than a single unit of transport per period, as well as investigation into computational efficiency improvements.

Future work should focus on computational effectiveness, as well as expanding the research to less specialised inventory control problems of non-dominant transportation costs.

Contents

1 Introduction

Efficiently managing the flow of goods is at the core of any retail business. At a high level, its logistics involve balancing the perceived cost of unfulfilled demand with the costs associated with meeting demand. While the former can be expressed in terms of profit, often this can be misleading when retailers are aspiring towards a brand image associated, for example, with a high service level. In addition, the price associated with providing a product in-store or online to a consumer is variable in nature due to holding and transportation costs.

Unfortunately, the complexity does not end there. When the lead time of an item extends beyond the promised delivery period, demand variations are anticipated through the use of inventory. Furthermore, if the lead time of an item is inconsistent, its stochasticity should also be considered. The aforementioned often form a basis for research in specific domains such as seasonality (Ehrenthal et al., 2014; Riezebos & Zhu, 2019), capacity constraints (Bretthauer et al., 2006), and perishable items (De Moor et al., 2022; Kara & Dogan, 2017) to name a few.

Given its intricacies, therefore, decades of research effort has been ramified into finding (near) optimal solutions to a wide range of implementations. Each solution orientation not only aims to enhance efficiency and performance, but also works towards enabling a practical, real-world application. Thus far, inventory management research has focused on generating interpretable rule-based policies that allow businesses to set clear parameters which trigger defined actions. Initially, these inventory systems were optimised at an item level, before being extended to managing inventory at a (sub-)vendor level via joint replenishment encapsulating spreading the order setup costs over multiple units. While greatly increasing problem complexity, joint replenishment promises the amelioration of reduced inventory levels and consequential total spend.

Following the advancement of computational capability, inventory management research has recently shifted its focus towards leveraging algorithmic and artificial intelligence (AI). This paper looks to build on these works in developing a renewed deep reinforcement learning (DRL) approach to solve a joint replenishment problem under stochastic demand and non-zero lead time durations. Though this method can also facilitate stochastic lead time durations, this element was excluded with respect to the specific problem application. Outcomes of this research suggest that the method can perform well in a small instance but struggles when scaled to larger settings. Acknowledging this, it shows further promise for DRL to tackle intractable inventory management problems for which no existing solution is available.

1.1 Research Structure

In this section, the form of the research including milestones are mentioned. Explicitly, Section 1.1.1 describes the prescribed problem as the subject of this research. Section 1.1.2 formally defines the goal of the research and Section 1.1.3 outlines the various research questions to be answered to reach the research goal. Finally, Section 1.1.4 describes the scope of the research and Section 1.1.5 works through the research approach.

1.1.1 Prescribed problem

Allowing employees to know what to order, order suggestions serve to relieve a large portion of analytical burden from supply planners. Leveraging information on lead time duration, stock

levels, sales and demand forecasting – its goal is to determine when and how much to order to satisfy demand. Generally, order suggestions therefore are an integral part of enterprise resource planning systems.

In spite of these promises, however, ADIL employee testimonies reveal that while the order suggestion tooling is useful, it also has its downfalls. The main issue derives from particular inefficiencies exacerbated by slow-moving items. These slow-moving items tend to be associated with Asian suppliers that ship loose-loaded containers over a long lead time duration. As shipment costs are dominant, it is often a challenge to best fill the containers at each ordering opportunity.

Currently, supply planners will receive an order suggestion once an item reaches a reorder point. If this item is indeed at risk of a stockout, the planner sets out to fill the container as best as possible given the upcoming demand forecast and item volume. Depending on the quality of order suggestion, this task can prove to be arduous.

Table 1 demonstrates an example order suggestion, together with its pallet configuration and expected number of stock weeks pre- and post-replenishment. Noticeably, as the expected stock weeks at replenishment is negative, the order suggestion is generated due to item 5 being at risk of stockout after the lead time duration. To help fill up a container, other items near their replenishment are added to the order. Based on the order amounts, the expected stock weeks after replenishment range between 11 and 89 weeks. In doing so, the order adds between 2 and 52 weeks of stock of each particular item.

Order Suggestion Example

Item #	Order Suggestion	# Collo per Pallet	Stock Weeks at Replenishment	Stock Weeks Post Replenishment
Item 1	0	648	16	-
Item 2	0	180	13	-
Item 3	2160	432	4	11
Item 4	5400	2700	16	35
Item 5	3120	640	-1	15
Item 6	576	480	15	21
Item 7	576	576	11	13
Item 8	720	240	15	25
Item 9	0	384	11	-
Item 10	1728	576	6	11
Item 11	2160	432	9	17
Item 12	1200	1200	57	89
Item 13	1152	432	10	20
Item 14	3024	432	13	63

Table 1: Example of an Order Suggestion

Dissecting the related information further, it becomes apparent that more items nearing their respective replenishment point were not included within the suggested order. Given the dominating costs of transport, company policy dictates that 40ft high cube containers are always used. After calculating, it shows that about 46% of its capacity is used when completely trusting the prescribed order amounts shown in Table 1. This unutilised container volume, while could be used for the unincluded items, could also be used to add more in-demand items to reduce order periodicity and consequent costs. Ultimately, this exemplifies the ineffectiveness of the current system, which renders the suggestions as merely useful for replenishment trigger points and not the associated order quantities. Accordingly, this has led system users to analyse the validity of its suggestions by separately generating revised order amounts.

Management, therefore, is interested in developing a new system to prompt effective item ordering suggestions. The goal of the project is not only to explore methods to effectively lower costs but also to induce dependability through high service levels.

1.1.2 Research Goal

In accordance with the highlighted problem, the goal of the research is to derive an alternate order suggestion system. Explicitly, the formal research goal is:

Develop and implement a state-of-the-art method to generate order suggestions which is suitable for the Ahold Delhaize Inbound Logistics department.

1.1.3 Research Questions

Fed by the research goal, overarching research questions can be formulated to strategically and thoroughly work towards its resolution. Each research and subsequent sub-research question breaks down the process into steps.

1. What is the current state at Ahold Delhaize Inbound Logistics?
    a. What are the key responsibilities of the Ahold Delhaize Inbound Logistics department?
    b. How is the department structured?
    c. What is the current state of ordering at Ahold Delhaize Inbound Logistics?

2. What efforts have been made towards similar topics in literature?
    a. What joint replenishment strategies have been explored in academic literature?
    b. What are their respective benefits and drawbacks?
    c. Which replenishment methods are most relevant to the specific challenges in Ahold Delhaize's inbound logistics?

3. How should the problem be modelled and compared?
    a. What is the chosen approach?
    b. Which assumptions or considerations are necessary from an implementation perspective?

   c. What are some benchmarks that can be used to understand the performance of the developed approach?

4. What results can be expected from the chosen approach and benchmarks?
   a. What metrics are suitable to judge the performance of different inventory control methods?
   b. How does the proposed model compare to alternative solutions in terms of efficiency and cost-effectiveness?
   c. What practical benefits and challenges arise from implementing the chosen model within a real-world setting?
   d. How does this approach improve on existing academic literature, and what should be done to further build on it?

1.1.4 Scope

Due to potential internal gains, the research will focus on gains to be found from Asian supplier ordering. This set of suppliers is characterised by loading cases full of consumer units into a container. These items are subsequently shipped via sea freight to finally arrive at ADIL warehouses located in the Netherlands.

1.1.5 Research Approach

In the remainder of the work, each of the research questions are sequentially answered. Chapter 2 outlines the company in detail to understand the setting in which to operate. It highlights specific characteristics and challenges which are necessary to acknowledge to determine an appropriate solution. Further, Chapter 3 discusses relevant developments in scientific literature to generate an understanding of the approaches made in similar topics – concluding with a mention of any relevant literature. Chapter 4 subsequently derives the methodology chosen to approach the specific issue. In conjunction, benchmark solutions are described. Reporting the results, Chapter 5 presents analytical research findings before Chapter 6 concludes and discusses the main findings prior to highlighting practitioner and academic recommendations.

2 Problem Context

Conducting this work at a host company, the following section outlines the setup within the department, as well as highlighting its specific requirements. Accordingly, Section 2.1 introduces the company as a whole, before Section 2.2 introduces the department in concern. Section 2.3 summarises the key findings with respect to the goal of the research.

2.1 Company Introduction

Handed over by his father in 1887, Albert Heijn took over a small shop situated in Oostzaan, the Netherlands. Within three decades, his brand had grown to about 75 locations. Continuing the trend of expansion and innovation, the Albert Heijn banner became the first retail company to go public in 1948, and the first grocery store in the Netherlands to offer self-service which allowed consumers to collect their own goods from shelves in 1952. During the 1970s, its growth ensured it became the largest supermarket chain within the Netherlands, and its domestic expansion turned international with the acquisition of businesses in Spain and the United States. Before the turn of the century, more businesses in Latin America, Central Europe and Asia were taken over.

Following an internal crisis in 2003, the business was forced to remodel and divest away from many of the aforementioned investments, focusing on its most profitable ventures in Europe and the United States. Under a strategy named “Reshaping Retail” which founded itself with the principles of increasing customer loyalty, increasing variety, expanding geographic reach, simplicity, responsible retail and people - the company was once more able to regain its footing. Its resurgence eventually led to the acquisition of Belgian supermarket conglomerate Delhaize in 2016 to form the organization known to this day as Ahold Delhaize (AD).

2.2 Department Introduction

Serving as an importer of a selection of foreign articles including wines, international cuisines and other non-food items, Ahold Delhaize Inbound Logistics (ADIL) is a department within AD which maintains the role of purchasing, receiving and distributing foreign goods. As an importer, ADIL inherently deals with international vendors. Together with these suppliers, a total of circa 300,000,000 items per year are sent to one of five AD warehouses located within the Netherlands. A large portion of this flow of goods originates from nearby Italy, France and Spain (Figure 1), while a significant portion of intercontinental traffic comes from Chile, Argentina, South Africa and China (Appendix 1).

Figure 1: Volume of European Trade Units

Using these products, ADIL supplies a range of retail chains across Europe (Table 2). Based in the Netherlands, however, its primary customer is the popular supermarket chain Albert Heijn (AH). Through its number of stores and cultural staple status, AH has been able to amass a market-leading majority in domestic supermarket business. Predominantly to reap the benefits of economies of scale, ADIL also supplies other Dutch retail chains as well as additional subsidiary chains located in Belgium, Czech Republic, Greece, Romania and Serbia.

ADIL Customer Banners

Name	Establishment Year	Country	Number of Stores
Albert	1991	Czech Republic	340
Albert Heijn	1887	the Netherlands	1268
Alfa Beta	1939	Greece	585
Bol.com	1999	the Netherlands, Belgium	-
Delhaize	1867	Belgium	818
Etos	1919	the Netherlands	523
Gall & Gall	1884	the Netherlands	628
Maxi	2000	Serbia	529
Mega	1995	Romania	977

Table 2: ADIL Customer Banners

Given the complexity of the network, ADIL deploys an array of departments whose operations involve product and supplier management, demand management, order management, and logistics. Providing the culture of continuous improvement, the department is looking to further

build and innovate on its efforts to more effectively serve the end consumer with competitive pricing while maintaining a reliable service.

In the following, context attributing to the company is outlined in detail. Section 2.2.1 discusses product and supplier management, followed by demand management in Section 2.2.2. Thereafter, Section 2.2.3 goes into the topic of order management, and 2.2.4 describes logistics processes.

2.2.1 Product and Supplier Management

Product management concerns setting up and maintaining products within the ADIL range in accordance with close communication with each of its retail banners. Specifically, when a subsidiary retailer is interested in introducing a foreign product into their assortment, this is communicated to the ADIL product management team along with indications of introduction week; number of carrying stores; and item rotation to establish temporal demand information. In doing so, these figures will be forwarded onto the demand and ordering teams to ensure that there is not only enough stock to initially fill the shelves, but also that a buffer stock is built such that the banner can fulfil the initial demand created by the introduction of the product.

Beyond this, another key task of product management is to ensure that there are enough items to fulfil the increased demand induced by product promotions. As well as steady communication with the banners, the team maintains internal demand forecasts. In the case that there is a projected insufficiency of items to fulfil demand, product management will help orchestrate any amendments to the promotion – such as reducing the attractiveness of the sale by placing the product in a less prominent position within the store or promotion catalogue or exchanging the promotions of two different products in line with stock availability.

On the other hand, supplier management is involved with managing suppliers tasked with delivering products. This is especially important for the own-brand products of each of the banners, of which different suppliers can be asked to deliver the different size variants of the same item. Once a supplier is chosen, the team helps guide the vendor in setting up digital compatibility with AD systems through providing information on its cooperation and products. This can include any regulatory documents indicating a certification or license. Similarly to product management, supply management also monitor and follow up on any delays in the shipment process to understand the underlying reasons for the delay.

2.2.2 Demand Management

Determining predictions of upcoming sales, demand management combines all necessary and available information together. By doing so, this ultimately allows the ordering team to have a greater confidence in understanding what should be ordered and when. Generally, forecasts within ADIL are formed from differing types of banner demand information. Its main retail banner offers demand data directly from the cash register, while other retail stores will share the number of units retrieved from warehouses as an indication of demand. In addition, ADIL have widely been granted access to inventory levels of its partners.

Deriving informative demand indications is predominantly done using time series forecasting methods such as exponential smoothing or moving average. As a rule of thumb, the extent to which demand is smoothed depends on the product type. Within the ADIL range, items are

distinguished between five categories: WINE, FOOD, NEARFOOD, NONFOOD, and SPIRITS (Table 3). A fundamental use of this classification is to determine effective smoothing parameters to more effectively act on variations in demand. For example, an averaging period of 12 weeks is chosen for items within the WINE category, which is a significantly shorter horizon than the 52 weeks of slow movers typically associated with the NONFOOD category. Adaptations to this policy are manually made on a case-by-case basis.

Item Categories

Category	Description*	Examples
WINE	-	Red Wine, White Wine, Rosé Wine, Sparkling Wine, Dessert Wine
FOOD	Products that when consumed provide nutritional support for the body.	Pasta, Chewing Gum, Chocolate, Canned Tomato, Stock Cubes
NEARFOOD	Products which interact with the body for its intended use, but are not meant for consumption.	Shampoo, Diapers, Toothpaste, Laundry Detergent, Dish Soap
NONFOOD	Products that are not meant for consumption but serve various practical, decorative, or recreational purposes within a household.	Cutting Board, Corkscrew, Vase, Stuffed Toys, Decoration
SPIRITS	-	Vodka, Whiskey, Rum, Gin, Tequila

*internal definitions to distinguish between item categories

Table 3: Internal Item Categories

Furthermore, to ensure the continued accuracy of information, a separation is made between promotional and non-promotional demand. During periods when an item is on sale, retail banners will typically experience an increase of around two to ten factors depending on aspects such as its prominence in stores or catalogue, or extent of discount. Considering how demand of the item largely tends towards pre-promotional levels thereafter, it is non-sensical to include inflated demand in future estimates.

After forming pre-emptive demand information with the aid of external forecasting software, it is forwarded onto the enterprise resource planning (ERP) system. Beyond feeding into digital tooling, these forecasts are made available for order management to provide basis to strategic decisions and order amounts.

2.2.3 Order Management

Following on from forecasting, order management is the process by which items are bought through a vendor to serve customer demand. Classically, inventory problems are defined by the goal of reducing the average level of stock on hand while minimising all relevant costs - subject to constraints. Aside from blanket orders, this process within the department is instigated primarily by order suggestions. Depending on the item, order indications originate either within the ERP system or an external supply chain management (SCM) tool. Due to its inherent complexity, these platforms are largely considered to be black-box solutions.

Once it is determined that an order of an item should be placed, planners assemble purchase orders at a vendor level. As roughly 80% of ADIL's suppliers are responsible for more than one product within the range, differing items can be consolidated into a singular order to save on total

ordering costs. Considering the combination of relatively high transportation costs and the sheer scale of operations, ADIL has a strong preference towards shipping with either full truck loads (FTL) or full container loads (FCL). When ordering small durable items from Asia for example, there may not be enough upcoming demand to fill a singular container. In this case, ADIL can either decide to ship an unfilled container, fill up the container with more items, or consolidate items with another company. Due to how the latter takes a period of about two to three weeks to process, the general solution is to load the container with more items.

In addition to establishing reordering policies, companies generally require a safety net to ensure the number of backorders is kept to a minimum in the form of safety stocks. Occasionally, demand will be greater than expected, or vendors will be unable to timely deliver on purchase orders for reasons such as material shortages, production bottlenecks, or quality certification issues. Congruently, operating with international partners, criminal operations, natural disasters, and regional conflict are also topics of heightened concern for ADIL. Currently, ADIL operates inventory with fixed safety stocks based on a given number of weeks of demand.

Further adding to the complexity, seasonal events should be kept in mind when ordering from specific vendors. An example of this is the seasonal shutdown of factories in Italy. Being the most important origin with respect to volume, Italian produce is of utmost importance for operations. In the leadup to the event of a shutdown, therefore, ADIL ensures to order surplus produce to cover demand for an extended period of time.

2.2.4 Logistics

To ensure the steady flow of goods between locations, ADIL leverages strategic partnerships with logistics specialists. These third-party logistics (3PL) providers are well integrated within its ecosystem, contributing to order suggestions as well as warehousing. Depending on the item that is to be held, it is sent to one of five warehouses (Table 4) through predetermined trade lanes. In addition to bringing the items from the supplier, logistics companies are also responsible for transporting the goods towards each customer. Besides regulating that shipments are on time and the finances are in check, this process is managed externally.

ADIL Warehouses	
Location	Inventory Type
North Brabant	Non-Food
Gelderland	Food, European Wine, Oil
North Holland	Diapers
Flevoland	Overseas Wine
North Brabant	Electronics

Table 4: ADIL Warehouses

In order to account for costs associated with the transportation and storage of goods, ADIL deploy a system which accordingly inflates the unit cost of an item. By doing so, this allows a

comprehensive overview on spending, as well as giving the ability to offer all-inclusive prices for its retail banners.

2.3 Summary

In setting out the requirements for an ordering system, each element of ADIL's department should be considered. From a product and supplier management perspective, an important and relevant aspect of their work derives from dealing with promotion and introduction periods of items. Next demand management importantly consider the forecasts of each item as an input into ordering decisions - which can be subject to the aforementioned promotion and introduction periods of items as well as the type of item. The order management team focuses on the task of placing orders at vendors, most of which are suppliers of multiple carried products. When using container shipments, there is a large preference towards filled containers due to its costs and simplicity with respect to consolidation. Ordered shipments are transported using the aid of 3PL providers.

Developing a solution to the perceived problem, therefore, should be mindful of each of the roles of the stakeholders.

3 Literature Review

Contextualising the chosen problem with prior research, Section 3.1 details historically used rule-based policy approaches. Section 3.2 introduces (deep) reinforcement learning in general prior to outlining its specific applications within the inventory management field in Section 3.3. Lastly, Section 3.4 discusses the relevant literature used as a foundation for this research.

3.1 Rule-Based Policies

Balancing holding and ordering costs, the classical problem of optimising item inventories has long concerned academics. Generically, the dilemma consists of finding an approach to ensure an item reaches a pre-defined service level according to some demand distribution at minimal cost. In addition, these problems have developed to include mechanisms to reflect real-life scenarios such as lost sales, lead time variability, and multi-echelon systems.

Instead of considering each item in isolation, coordination of multiple items from a single vendor enables a larger number of total items to be ordered at once. Consequently, this leads to reduced transportation costs and reduced environmental impact. Initiating this process has popularly been achieved through the use of can-order, periodic review or aggregate inventory policies. First introduced by Balintfy (1964), can-order policies (s_i, c_i, S_i) dictate that an item i is replenished until stock level S_i once it reaches reorder point s_i. All other items $i \neq j$ from the same vendor are also replenished until S_j if their respective inventory levels have reached or fallen below a can-order level c_j. The primary motivation towards can-order policies is founded by the economic benefit of sharing ordering costs for items close to their reorder point outweighing that of the increased holding costs.

Alternatively, Atkins & Iyogun (1988) proposed periodic review policies (R, s_i, S_i) in which inventory levels are observed after a review period R to determine the items to have fallen below or reached a reorder point s_i. These items are consequently replenished until S_i. Principally, periodic review policies wait for demand to accrue over time before consolidating individual items requiring replenishment. This simplicity has led to its favourability within the industry. Comparative studies between can-order policies reveal that periodic review policies outperform when setup costs are relatively high (van Eijs, 1994) but are disadvantageous when vendors supply a larger number of items (Melchoirs, 2002; Schultz & Johansen, 1999) or if demand variations are high (Johansen & Melchoirs, 2003).

Combining the can-ordering with periodic review, Johansen and Melchoirs (2003) develop a compensation method to leverage the advantage of each respective approach. Testing with irregular demand patterns, the research showed cost reductions of up to 15% when demand was irregular. While the study investigates stochastic demand patterns using probability mass functions, no extension is available to accomadate variable lead time duration or capacity constraints.

Another paradigm of rule-based joint replenishment problems, proposed by Renberg and Planche (1967), consider aggregate ordering as a trigger. As the founding concept, the (Q, S) policy continuously reviews the inventory level of items to observe the point at which the depleted stock equals the modal transport capacity. Advantageous when transportation costs are high, aggregate order triggering pushes use of FTLs. Pantumsinchai (1992) compliments this idea by

proposing an optimal control parameter calculation under the assumption that demand follows a Poisson demand process. Evidentially, the work shows that while aggregate order triggering is advantageous when ordering costs are high and holding costs are low, there is a tendency to to incur frequent shortages. Due to how this is caused by the aggregate inventory level failing to trigger an order, Viswanathan (1997) evidence effective performance of (Q, S) policies when all items have identical cost and demand parameters.

Provided the aforementioned limitations, various adaptations to this method to hybridise the reorder triggers or alter the order amounts. Pertaining to the former, aggregate triggers are combined with can-ordering through (Q, s, S) policies (Gürbüz et al., 2007) and with periodic review for (R, S, Q) policies (Özkaya et al., 2006). Other variations include $Q(s, S)$ policies, which use aggregate order triggers to reorder items when inventory level falls below s, raising it to level S (Nielsen & Larsen, 2004; Viswanathan, 1997); (s, Q) policies, which fill the transport capacity to equalise the time until the next replenishment (Tanrikulu et al., 2009); $(Q, S|T)$ policies, which periodically review inventory and ship only if the order quantity reaches Q (Cachon, 2001); and (U, S) policies use the accumulation of a Poisson demand process to trigger an order if the volume exceeds transport capacity (Li & Schmidt, 2019).

The main issue with the implementation of most of these methods, however, is determining optimal values for each control parameter. Similarly to the single item setting, joint replenishment problems are modelled using states and state transitions in a Markov chain. The increased complexity stems from the exponential state space expansion caused by each item inventory level combination, as well as the associated exponential action space expansion required by joint replenishment. Commonly referred to as the curse of dimensionality, this renders problems of large action and state spaces mathematically intractable for instances with more than a handful of items (Creemers & Boute, 2022).

Acknowledging this, Silver (1974) proposed an n-item decomposition to tackle joint replenishment using n independent problems with discounted ordering costs for including other items within an order. Critics of this method argue that while ordering is optimal for the triggering item, it may not be for articles added to the order (Zheng, 1994). In addition, as this technique has readily been used to determine can-order policy parameters, negative sentiment towards its efficiency has been subsequently ill-attributed (van Eijs, 1994).

As an alternative, genetic algorithms (GA) or differential evolution (DE) have been evidenced to iteratively learn improved rule-based policies. More specifically, GAs digitise the concept of natural selection to effectively learn control parameters. For example, Hong and Kim (2009) derive an unbiased estimator to compute the exact cost in polynomial time applied within a GA to determine a well-performing inventory control policy. On the other hand, DE algorithms break down complex problems into discrete steps, enabling performance analysis through successive approximations. A novel DE approach to joint replenishment developed by Wang, He, et al. (2012) was used in a setting of two power companies to generate near-optimal results. Wang, Dun, et al. (2012) later develop an approach to joint replenishment and delivery which combine GAs and DEs into a hybrid differential evolution algorithm (HDE). Applied to an instance of stochastic demand, HDE is shown to be a suitable solution candidate to the intersection of two NP-hard problems.

Other heuristic approaches have been leveraged when there are no existing exact solution methodologies. Notably, Kaspi and Rosenblatt (1991) developed RAND, which has been since modified to suit different applications in solving the joint replenishment problem. Comparing the use of RAND and GA, Khouja et al. (2000) find that GAs found improved solutions to the joint replenishment problem in comparison to RAND in addition to being easier to implement. More recently, Visentin et al. (2023) built on existing research to propose a promising stochastic dynamic programming heuristic which generates near-optimal (R, s, S) solutions with a considerably reduced computation time.

More recently, however, Creemers and Boute (2022) proposed an exact evaluation approach to solve can-order policies in a continuous time setting by sufficiently reducing the state space through embedded Markov chain modelling that leverages (compound) Poisson demand processes to rework an existing near-optimal solution. Importantly, the method can incorporate backlog, lost sales, and non-zero lead times adaptations without increasing state space dimensions. Authors still note; however, the model is still constrained to suit a smaller number of items.

3.2 (Deep) Reinforcement Learning

Emerging as an alternative solution methodology, machine learning in its many facets has diffused into a wide range of domains. Reinforcement learning, a subset of machine learning, is an agent-based sequential technique which interacts with an environment to maximise some cumulative reward (Sutton & Barto, 2005). More comprehensively, in each time step the system records the current state s_t of the environment before the agent suggests an action a_t to take and the consequential reward R_{t+1} of the action is observed (Figure 2). Cyclically the agent continues to learn appropriate state-dependent actions $\pi(a_t|s_t)$ leading to improved long-term rewards.

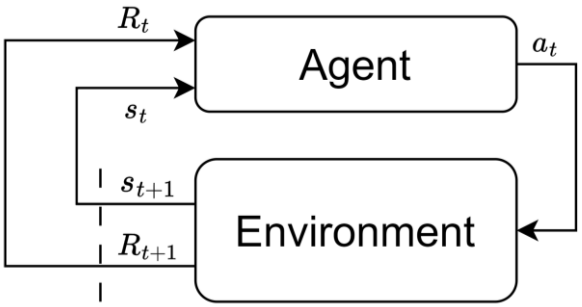


Figure 2: Agent-Environment Interaction

Reinforcement learning can be classified into model-based or model-free methods (Li, 2018) which distinguish between accessibility of agents to transition and reward functions, referred to as a model. Model-based reinforcement learning, therefore, leverages a dynamics model of the environment to predict state transitions and rewards to enable informed action decisions (Demizu et al., 2023). Incongruously, model-free approaches are suitable for instances involving complexities such as high stochasticity, long-term planning horizons, partial observability, and imperfect information (Rolf, et al., 2023). Notably, compared to model-based, model-free methods require more computational effort to learn an effective policy (Sutton & Barto, 2005).

In instances pertaining large state and action spaces, however, reinforcement learning falters in computational effectiveness. Subsequently, DRL has emerged as a revolutionary technique to

derive an action for a given state instead of leveraging static action value tables for storage and updates (Mohamadi et al., 2023). DRL uses artificial neural networks to emulate neural links such that an array of inputs can be mapped to an array of output values $\boldsymbol{y} = \phi(\boldsymbol{x})$ (Figure 3). Between the input and output layers, a series of hidden layers process the data by means of weighted neural connections and neural biases. At each neuron, besides those contained within the input layer, an activation function determines if and to what extent the neuron should be activated. Once activated, this signal is sequentially forwarded onto the next layer in the neural network. At the consequential output layer, desired outputs are presented through classification or regression.

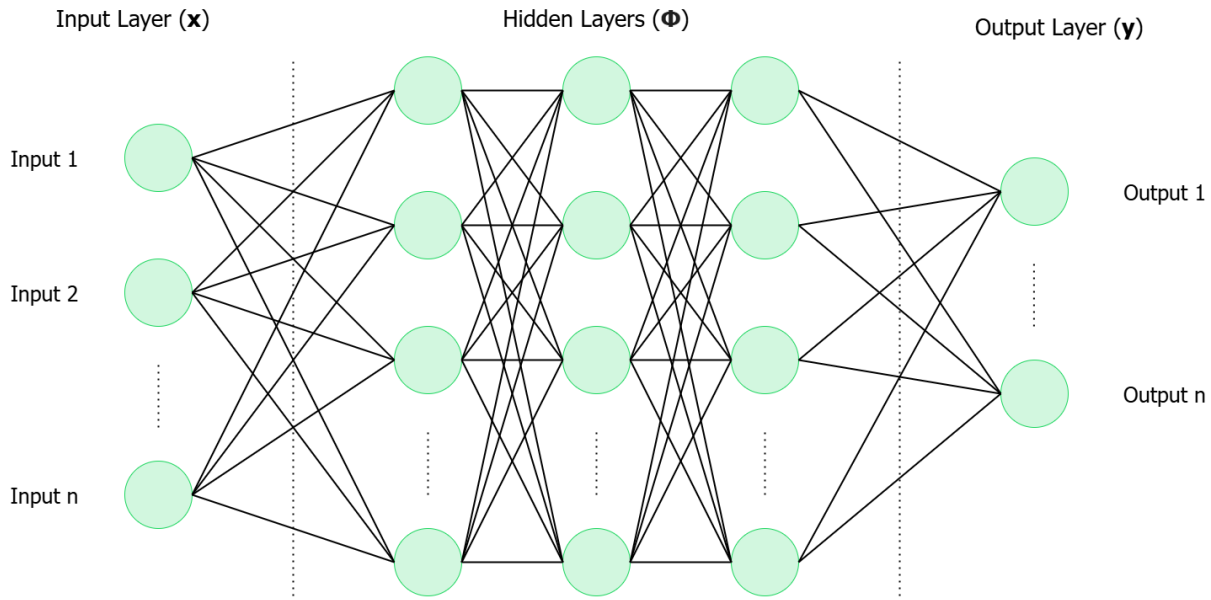


Figure 3: Artificial Neural Network Construction

3.3 Deep Reinforcement Learning within Inventory Management

As of recent, model-free (D)RL has been developed as an alternative method in various complex fields, including inventory management (Boute et al., 2021). In search of an optimal inventory policy, two predominant algorithmic approaches are available. Firstly, value-based algorithms generally derive a parameterised approximator $Q_\theta(s,a)$ to map states s to an apropos action a. The resulting policy π^* is subsequently approximated by the optimal action-value function $Q^*(s,a)$, which considers the utility of taking each action for a given state (Equation 1). Inherently, each update utilises data collected at any point during the training (Rolf, et al., 2023).

$$a(s) = \arg\max_a Q_\theta(s,a) \quad (1)$$

Alternately, policy optimisation algorithms determine an optimal parameterised policy $\pi_\theta(a|s)$, where each parameter θ is typically optimised with respect to an expected return $J(\pi_\theta)$ using gradient ascent. Characteristically, policy optimisation algorithms only consider the most recent policy when updating (Rolf, et al., 2023). To a lesser extent, Bayesian and evolutionary approaches have been evidenced within inventory management literature to date.

In the remainder of the chapter, specific value-based and policy optimisation deep reinforcement learning methods used within inventory management literature are introduced in Sections 3.3.1

and 3.3.2 respectively. In addition, Section 3.3.3 presents DRL training variance reduction methods used within inventory management literature.

3.3.1 Value-Based Algorithms

By far the most common value-based algorithms used in DRL inventory management literature include Q-learning, Deep Q-Networks (DQN), and State-Action-Reward-State-Action (SARSA). As the most established technique, Q-learning sets out to determine the maximal expected value for state-action combinations. Suitable for limited discrete state and action spaces, Q-learning agents generate a tabular record of Q-values $Q(s, a)$ to represent the total expected future reward for taking an action a in state s. Notably, Chaharsooghi et al. (2008) used this approach in a popular toy problem to derive suitable integrated ordering policies to minimise total inventory costs. Furthermore, Meisheri, et al. (2021) leveraged Q-learning for a multi-product, multi-period environment with uncertain demand to determine a concurrent inventory management framework.

Extending the Q-learning approach, DQN was developed to apply artificial neural networks and experience relay techniques (Mnih, et al., 2013). The latter refers to the concept of introducing a memory buffer of the past experiences of an agent to randomly sample such that correlation between consecutive decisions is reduced and training stability increases. Zwaida et al. (2021) applied DQN to automatically trigger restocking decisions to tackle supply chain disruptions such as manufacturing problems, demand issues, and raw material shortages. Oroojlooyjadid et al. (2022) also developed a DQN approach to optimise the inventory of the well-known beer game problem.

Lastly, SARSA is an adaptation of the tabular Q-learning approach by which actions are taken with respect to a policy to learn Q-values (Rummery & Niranjan, 1994). SARSA has been used within inventory control literature to tackle instances of non-stationary state-based demand influenced by promotional periods and seasonality (Yin et al., 2022).

3.3.2 Policy Optimisation Algorithms

Utilised to a lesser extent, policy optimisation algorithms such as policy gradient, A2C/A3C, PPO and SMART approaches are still prevalent in inventory management literature. In addition, an approach suited towards solving inventory-oriented problems, coined deep controlled learning (DCL), has been proposed. In general, policy gradient approaches increase the certainty of selecting actions associated with higher rewards until an optimal policy is obtained. Specifically, the algorithm incrementally updates policy parameters θ using stochastic descent according to policy performance $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$ in which α represents the learning rate.

Similar to a traditional actor-critic method, A2C/A3C methods simultaneously train the actor π_s and critic $V_\pi(s)$ (Sutton et al., 1999) but do so through the use of artificial neural networks and non-linear function approximators. Despite the different names, A2C and A3C methods are mathematically identical but are executed using a GPU or CPU respectively (Rolf, et al., 2023). In a comparative study, A3C was trained to match the performance of state-of-the-art heuristics and approximate dynamic programming. Results suggested that minimal adaptations would be necessary to transfer this method into a real-life company setting (Gijsbrechts et al., 2022).

The most popular of the policy optimisation algorithms, Proximal Policy Optimisation (PPO) again leverages an actor-critic method to both allow the actor to map observations to actions and the critic to give an expectation of rewards (Schulmann et al., 2017). Relevantly, Hachaïchi et al. (2020) used PPO to train an RL agent to place optimal orders, considering stochastic lead times. In addition, Vanvuchelen et al. (2020) applied PPO to a capacitated joint replenishment setting with zero lead times to outperform existing methods.

As an extension of the temporal difference algorithm, SMART is applicable to problems not defined comprehensively by Markov chains (Das et al., 1999). Previously, this method has been used to determine an integrated inventory management policy which optimised the performance over the complete supply chain (Giannoccaro & Pontrandolfo, 2002).

Finally, DCL algorithmically casts reinforcement learning as a classification problem to iteratively improve policies by evaluating each action for a given state under multiple exogenous scenarios to propose an action which reduces expected costs over a trajectory (Temizöz et al., 2025). While the aforementioned generic methodologies have mixed performance in different classifications of high-stochasticity inventory problems (De Moor et al., 2021; Gijsbrechts et al., 2022), DCL has shown promise in consistently outperforming state-of-the-art heuristics in achieving a low optimality gap (Gijsbrechts et al., 2022). Since its recent induction, DCL has also been used in capacitated lot-sizing (van Hezewijk, Dellaert & van Jaarsveld, 2024), and dual sourcing (Akkerman et al., 2024) inventory problem settings.

3.3.3 Training Variance Reduction

A common issue among DRL inventory management literature, training stability enables algorithms to smoothly taper to more reliable prescriptive advice. A series of approaches have been presented in the form of common random numbers (CRN), sequential halving (SH) and reward shaping.

Decreasing variance while improving the accuracy of each suggested action-state pairing, CRN is a technique involving evaluating each action under the same exogenous scenarios (Temizöz et al., 2025). Implemented within DCL, it ensures alternate actions for a given state a fair comparison in which training variance is reduced.

Additionally utilised in DCL, SH is a bandit algorithm which efficiently assigns exogenous scenarios with more promising actions (Temizöz et al., 2025). As semantically inferred, SH iteratively discards the least promising half of actions to focus on more auspicious decisions. In doing so, this facilitates an increase in training stability for high stochasticity environments.

Furthermore, reward shaping is another technique that has been proposed in inventory management literature. Typically used as a tool for sparse reward signals, inventory management literature also suggests that reward shaping can be leveraged to tackle sample inefficiency, high computational cost, and training stability associated with DRL (De Moor et al., 2021). At a high level, reward shaping enables the agent to learn from existing domain knowledge based on the deviation between the chosen and desired action taken. Figure 4 depicts how the additional teacher role interacts with the traditional agent-environment interaction.

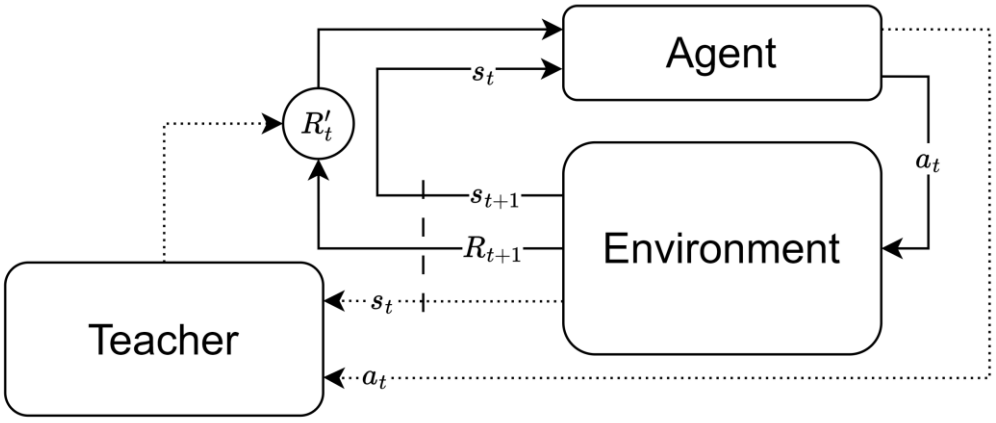


Figure 4: Reward Shaping Agent-Environment Interaction

Anticipatedly, adding a teacher element within neural network training is to provide guidance through altering intercepted reward signals depending on the perceived quality of action. While evidence suggests a trustworthy, fast, and more stable convergence towards an optimal policy, improper use of reward shaping has the ability to divert the model from the optimal outcome (De Moor et al., 2021).

3.4 Relevant literature

Although research on joint replenishment inventory policies is plentiful, there seems to be no work which addresses the specific problem at hand, let alone the chosen solution approach. The closest parallel work is that of Vanvuchelen et al. (2020) which describes a stochastic joint replenishment problem with zero lead times and a stepwise ordering cost function to represent transportation volume constraints. While the method provides a near-optimal solution to drastically outperform a carefully chosen heuristic and rule-based solution, it importantly fails to include non-zero lead time durations which while adding complexity are more reflective of a real-life replenishment process. Rightly, Vanvuchelen et al. (2020) point out how the effectiveness of a policy derives from its ability to fill truck- or container loads when transport costs are dominant. Through forcing the model towards full containers, it is able to achieve results but in doing so debilitates its potential to work in more generic problem instances.

Once more, while there is a significant amount of literature on rule-based inventory solutions, only one such joint replenishment policy with the goal of filling containers was found. Presented by Li and Schmidt (2019) the (U, S) policy waited for demand following a Poisson process to accrue to the volume of a container before it is triggered. Although this is an effective way of filling containers, it fails in practice as it uses a Poisson distribution to model demand, assuming uniform demand sizes instead of the demand magnitude variability induced by reality. In that light, it would entail that if multiple units of an item were ordered within an instance, it could overfill the volume of a single container. In doing so, the ordering policy would dictate either that an additional container would be ordered in a single period, or the order is shortshipped to the detriment of cost and service level respectively.

More generically, as an adaptation of the base stock policy tailored towards joint replenishment, traditional can-order policies as presented by Balintfy (1964) still remain the most common heuristic in practice for most problems. Due to the combinational complexity of defining a can-

order policy, no scalable short-form solution is available despite recent efforts by Creemers and Boute (2022). In addition, can-order policies fail to directly include capacities within its formulation, while also limited by its ability to only observe inventory levels.

Recent developments in approaching inventory management, presented by Temizöz et al. (2025), evidence renewed promise at effective solving of new or existing problems in the form of DCL to learn neural network policies. Though successful within different inventory management domains, it has yet to be applied to the joint replenishment problem.

As such, the continuation of this work highlights an alternate and renewed approach to achieving a cost-effective joint replenishment policy which includes stochastic demand and non-zero lead times. In doing so, it proposes a technique to allow autonomous sequential decision-making in choosing how to fill a single unit of transport based on a larger number of input variables.

4 Methodology

In accordance with the literature presented, the most promising avenue of approach is chosen to be DCL. In its ability to handle high levels of stochasticity, it has been evidenced to outperform state-of-the-art heuristic solutions in other inventory problems. In part, this is due to how the technique leverages CRN and SH to reduce training variance. The formulated model will be introduced in Section 4.1, before discussing the benchmark policies in Section 4.2.

4.1 Model Formulation

To achieve an effective implementation, stylistic choices are necessary to reduce the problem into something permissively computational. The first decision of which pertains to how ordering is determined on a pallet level. Beyond simplicity, this lot-sizing approach has a multitude of benefits. Firstly, adding at least a pallet of an item encapsulates a realistic estimate of the minimum order quantity (MOQ), especially useful in scenarios where data regarding actual supplier MOQs is limited in quality. Secondly, full pallets or partial pallets still incur the same warehousing costs. Procedurally, when a new pallet is introduced into a warehouse, partial pallets of the same item are often not combined for efficiency to facilitate a first-in-first-out (FIFO) inventory policy. By restricting ordering decisions to an integer number of pallets, it also seeks to more effectively use warehouse space. Lastly, lot-sizing reduces the possible discrete actions for implementations operating at a large scale. While alternate methods to reducing this large scale exist; the culmination of the aforementioned lend itself to the reduction of action space through a positive integer number of pallets.

Further, it was chosen that a maximum of one container would be ordered per period. Noting that the problem setting requires orders to be placed only a handful of times per year, it is intuitive that a suitable solution should be restricted to order at most one container per period. Also, despite simplifying the problem into pallet denominations, complexity arrives in how the volume of pallets differs with respect to each item. In combination with how containers are loose loaded, the filling of a container resembles a bin-packing problem. For simplicity, an industry-accepted value for container capacity is used such that no further items would fit if the combined volume of ordered items exceeds it.

In dealing with the large action space, an approach is chosen to decompose the action into two sub-actions: which item to order and how much of that same item to order. After deciding on which and how much of an item to order, this process repeats until either there are no more items to add to an order, the model decides not to add another item to the order, or there is no remaining capacity within the container. Doing so allows for a step-by-step approach to filling a container, rather than offering all combinations of how to fill a container at once.

In further detail, Section 4.1.1 documents the variables and parameters. The respective state and action spaces are outlined in Sections 4.1.2 and 4.1.3, and the overall transition dynamics of the MDP are presented in Section 4.1.4. Section 4.1.5 provides the reward function used to express the cost induced per period to ultimately guide the model to decide actions with improved long-term rewards. Lastly, Section 4.1.6 describes the process of DCL as the approach used as the focus of this research.

4.1.1 Variables and Parameters

t :	Time period
$i \in K$:	Set of items supplied by a given vendor
$I_{i,t} \in \mathbb{Z}$:	Inventory level of item i in period t
$\{q_{i,t} \in \mathbb{Z}^+ \forall i \in K\} \in Q_t$:	Number of pallets of item i scheduled to arrive in period t
$\bar{C} \in \mathbb{R}^+$:	Maximal available cubage within a container
$z_t \in \{0,1\}$:	Binary indicator for if a container was ordered in period t
$\tau \in \mathbb{R}^+$:	Used container capacity
$\mu_{i,t}^L \in \mathbb{R}^+$:	Forecasted demand of item i in period t over lead time L
$\sigma_{i,t}^L \in \mathbb{R}^+$:	Forecasted standard demand deviation of item i in period t over lead time L
$d_{i,t} \in \mathbb{Z}^+$:	Observed demand of item i in period t
$v_i \in \mathbb{R}^+$:	Pallet volume of item i
$b_i \in \mathbb{R}^+$:	Backordering cost per pallet of item i
$\mathrm{o} \in \mathbb{R}^+$:	Ordering cost per container
$h \in \mathbb{R}^+$:	Holding cost per pallet
$L \in \mathbb{Z}^+$:	Deterministic lead time duration
$T \in \mathbb{Z}^+$:	Experiment time horizon (non-stationary demand)
$\gamma \in [0,1]$:	Discount factor
$\alpha \in [0,1]$:	Forecast smoothing factor
$\beta \in [0,1]$:	Forecast standard deviation smoothing factor

4.1.2 State Space

An effective state space completely describes the environment using a minimal number of variables with the interest of computation time in mind. In the proposed item state space (Equation 2) variables are updated at different points in time. Firstly, state variables which update only when entering a new period include inventory level of item i in period t ($I_{i,t}$), forecasted lead time demand of item i in period t ($\mu_{i,t}^L$), and the forecasted standard demand deviation of item i in period t during lead time ($\sigma_{i,t}^L$). The remaining state variable representing the number of pallets of every item i scheduled to arrive in each period within the lead time (Q_t) is updated after each sub-decision.

$$s_\tau = \left\{I_{i,t}, \mu_{i,t}^L, \sigma_{i,t}^L, Q_t\right\} \forall \mathrm{i} \in \mathrm{K} \quad (2)$$

In updating the forecast and standard deviation of each item at each time step, simple exponential smoothing (SES) was used (Equations 3 and 4). Extrapolating to consider demand over lead time, these values were multiplied by a lead time coefficient to respectively reflect the mean and standard deviation of demand over the lead time (Equations 5 and 6). While any suitable forecasting approach can be used, SES was selected for its widespread application and flexibility.

$$\mu_{i,t} = \alpha d_{i,t} + (1-\alpha)\left(\mu_{i,t-1}^L \Big/ L\right) \quad (3)$$

$$\sigma_{i,t}^2 = \beta\left(d_{i,t} - \left({}^{\mu_{i,t-1}^L}\!/_{L}\right)\right)^2 + (1-\beta)\left({}^{\sigma_{i,t}^2}\!/_{\sqrt{L}}\right) \quad (4)$$

$$\mu_{i,t}^L = L * \mu_{i,t} \quad (5)$$

$$\sigma_{i,t}^L = \sqrt{\sqrt{L} * \sigma_{i,t}^2} \quad (6)$$

The complete state space, as a representation of the entire environment, contains a collection of the item state spaces as well as other important variable information (Equation 7). Contextually, the only other item-unspecific variables to alter with time is the used container capacity τ, and remaining time horizon $T - t$ in the case of non-stationary demand.

$$s_\tau = \left\{\left(I_{i,t}, \mu_{i,t}^L, \sigma_{i,t}^L, Q_t \ \forall \mathrm{i} \in \mathrm{K}\right), \tau, T - t\right\} \quad (7)$$

4.1.3 Action Space

Due to permissible discrete actions exponentially increasing as a function of the number of items in a joint replenishment problem, determining a suitable approach to the action space is of paramount importance. Instead of considering each of the possible actions at once, the proposed formulation breaks down the combinatorial action space into two iterative and separate decision opportunities at an item level. The first-stage decision A_τ^1 considers which, if any, of the items is to be ordered based on the provided state space (Equation 8). In this instance, any $(p_1, p_2, \dots, p_{|K|})$ or no (further) items can be chosen (p_0). The second-stage decision A_τ^2 determines the order quantity q of the item previously suggested in the first-stage decision, once more based on the same available state space (Equation 9). Due to the spatial capacity constraint, the model only tolerates the number of pallets of item i that can fit into the remaining space within the container. For this reason, a permissible item quantity upper bound is expressed using container capacity $\bar{C}$, remaining capacity τ and pallet volume v_i. This process iteratively continues for all items until the model dictates that no other item can or will be ordered in the current period.

For implementation purposes, first- and second-stage actions are represented in combination according to Equation 10.

$$A_\tau^1 = a_t^1 \in \left\{p_0, p_1, p_2, \dots, p_{|K|}\right\} \quad (8)$$

$$A_\tau^2 = a_t^2 \in \{q_1, \dots, \lfloor(\bar{C} - \tau)/v_i\rfloor\} \quad (9)$$

$$A_\tau = \{A_\tau^1, A_\tau^2\} \quad (10)$$

4.1.4 MDP Transition Dynamics

Figure 5 describes the transition dynamics of the proposed MDP model. Descriptively, each period t is broken down into a maximum of $|K| + 1$ sub-decisions. Each sub-decision is also split into the two aforementioned stages. If the first-stage decision $A_\tau^1 \neq p_0$ is chosen, indicating the agent has decided to order an item i, the model checks if the remaining spatial capacity $\bar{C} - \tau$ is less than the volume of a full pallet of the nominated item v_i. Given no capacity violation, a subsequent second-stage decision then determines how many complete pallets of item i should be added to the remaining container space. Thereafter, state variables τ and $q_{i,t+L}$ are updated

before choosing the next item to add to the order. Order quantity decisions are not revisited and no more of this item can be added thereafter.

If at any stage during period t the first-stage decision $A_\tau^1 = p_0$ or the volume of the chosen item i (v_i) exceeds the remaining capacity $\bar{C} - \tau$, the model enters a new period after item inventories are updated with incoming $q_{i,t}$ and outgoing $d_{i,t}$ stock. Prior to initialising the variables for a new period $t + 1$, forecasting parameters are updated, and the value of the reward function is calculated. Noticeably, this transition diagram is akin to a capacitated periodic can-order formulation in which altering the duration of time period t can influence the extent to which the review period is continuous.

This formulation also differs from other approaches to similar problems in that it does not simplify by enforcing full shipments. In that light, while applied to a specific instance, it would have the ability to expand to more generic settings of joint replenishment.

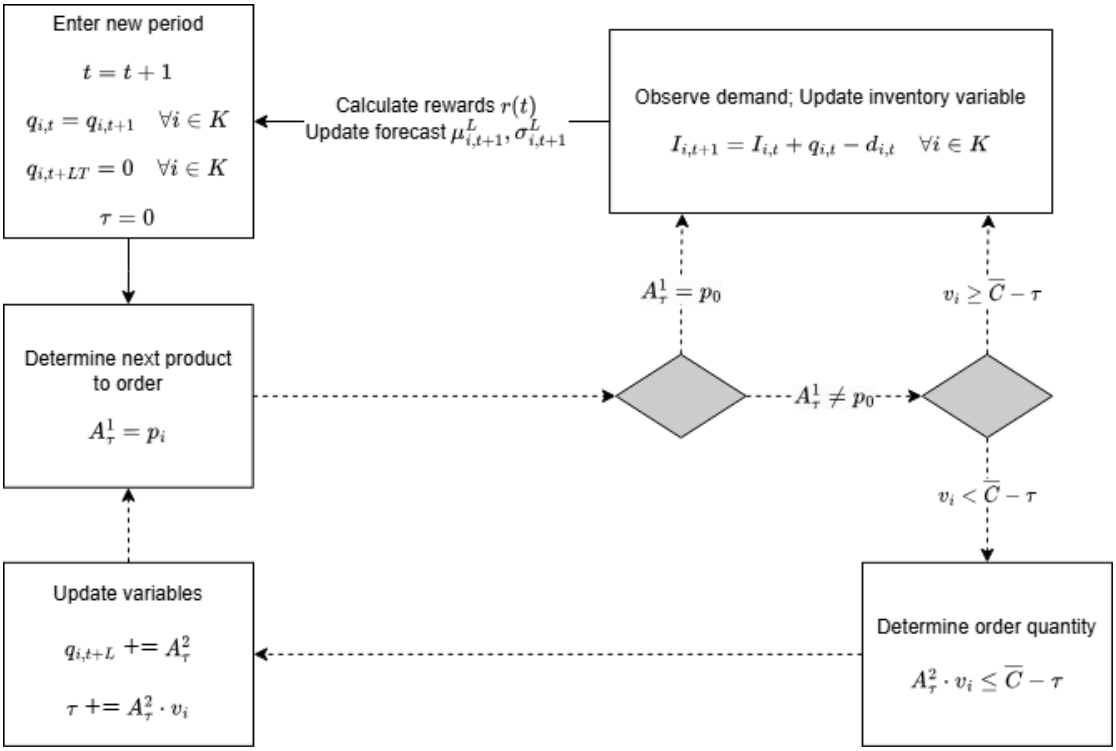


Figure 5: MDP Transition Dynamics

4.1.5 Reward Function

Allowing both the model and end user to understand the quality of decisions made, a cost function is defined to quantify holding, backordering, and ordering costs. As shown by Equation 11, each variable is multiplied by a defined cost coefficient before being summed. Effective decision policies π_t should minimise the (discounted) sum of taking an action a_t in state s_t over a sufficiently long horizon T (Equation 12).

$$c(t) = \sum_{i \in K} \left(h_i \cdot [I_{i,t}]^+ + b_i \cdot [I_{i,t}]^- + s \cdot z_t \right) \quad (11)$$

$$C_t^{\pi_t}(s_t) = \sum_{j=0}^{T} \gamma^j \, c_{t+j}\big(s_{t+j}, a_{t+j}\big) \quad (12)$$

4.1.6 Deep Controlled Learning (DCL)

At its core, DCL is founded by the use of Classification-Based Policy Iteration (CBPI), treating policy learning as a classification task instead of leveraging policy gradients. In doing so, the approach simulates state-action combinations to understand the most favourable action for a given state. Later, these favourable actions are treated as labels using a form of supervised learning to generate a neural network policy which mimics this mapping. Suitable for stochastic and complex inventory problems, it avoids estimating noisy cost-to-go functions by leveraging a simulated best action for a given state.

In detail, the process of DCL begins with defining an initial (heuristic) policy. For exploration purposes, as in this case, no-pre-defined logic can also be used in favour of a random policy. Using the chosen policy, the algorithm leverages hyperparameters including the number of states to sample N, number of exogenous samples M, rollout horizon H, warm-up period L, and number of generations n.

For each of the N samples contained within a generation, the dataset is initialised using a warm-up trajectory of L periods in order to reach realistic, frequently visited states representative of the current policy. For each state s, all feasible discrete actions $a \in A$ are evaluated over a rollout horizon H using the same set of M exogenous inputs (e.g. demand). Aiding a fair comparison, CRN is used to ensure the same random inputs are applied across all samples. Following, SH is used to sequentially eliminate half of the actions pertaining to the worst-performing state-action combinations before a single simulation-based best action $\hat{\pi}^+(s)$ is determined through the lowest estimated cost. Each of the best state and action pairings are collated into a dataset K, later to be used as inputs to supervised learning (Equation 13).

$$K = \big\{\big(s_k, \hat{\pi}^+(s_k)\big)\big\}_{k=1}^{N} \quad (13)$$

Lastly, a neural network classifier is trained to map states to actions to a neural network policy π_θ to mimic the simulation-based action selection process (Equation 14).

$$\pi_\theta(s) = \mathrm{argmax}_a NN_\theta(s)[a] \quad (14)$$

This process is repeated for n generations, where each newly trained policy π_θ is used to generate the subsequent dataset before a resulting in a final trained neural network policy π_n.

4.2 Benchmark Policies

To understand the relative performance of the proposed model, it will be compared against other solution implementations. As discussed in Section 3.4, there are no directly comparable works, though parallels can be drawn. The first of the comparisons pertain to a related state-of-the-art solution developed by Vanvuchelen et al. (2020), namely generating a neural network policy using PPO trained on a cost function. While this approach simplified the situation through a zero-lead time assumption, it is still possible to test the technique with a more complex scenario.

In addition to comparing against PPO, and to gain perception on the overall benefit of using neural network policies, a simulation-based GA approach to tuning rule-based control parameters is used. Due to how the (U, S) policy relies on demand arriving on a one-by-one basis, the more generic can-order approach will be used as a benchmark.

Accordingly, Section 4.2.1 outlines the genetic algorithm used to generate can-order policies before Section 4.2.2 discusses how PPO will be used as an alternate DRL method to generate a neural network policy.

4.2.1 Genetic Algorithm

Genetic algorithms are a parallelisable and adaptable approach, useful in a range of combinatorial optimisation problems, for efficiently searching optimal configurations. To apply a GA to generate a rule-based inventory policy, individuals can be represented as an array of tuples, in which tuples refer to a set of control parameters of an item e.g.

$$\text{individual} \rightarrow [[s_1, c_1, S_1], [s_2, c_2, S_2], \dots, [s_{|K|}, c_{|K|}, S_{|K|}]]$$

After an initial population composed of a series or randomly generated individuals is obtained, the fitness of each are evaluated. In this case, the fitness of an individual is determined by its performance over a simulation where each of the tuples represents an ordering policy of an item. Individuals that outperform a large portion of the populus are chosen to reproduce offspring in crossover, whereby elements of each parent create a range of possible values to be chosen.

Stimulating exploration, mutation is included such that offspring genes are randomly altered with a small probability. These mutated individuals are then, along with the unmutated offspring and parents, chosen to form a new population. Including the parents once more within the new population, referred to as elitism, ensures that best-performing individuals are not lost during crossover or mutation to stabilise and accelerate algorithmic performance. This is cyclically performed over a number of generations until a stopping criterion is met, such as a maximum number of generations or no further improvement over a number of cycles.

4.2.2 Proximal Policy Optimisation (PPO)

Similarly to DCL, PPO falls within the class of DRL algorithms. Beyond being considered the state-of-the-art technique in parallel problems, PPO outperforms other widespread DRL approaches through its ability to stably and efficiently update policies by constraining updates within a trust region, thus balancing exploration and exploitation without requiring complex second-order optimisation techniques.

More elaborately, PPO begins by collecting batches of experience through running the current policy π_θ in the environment. Each batch of experience, containing multiple trajectories, collects a sequence of transitions that include at a minimum a state (s_t), action (a_t), and reward (r_t) at each time step. Thereafter, the return and advantage estimate is computed for each period within a trajectory. Often notated R_t, the return is typically calculated as the cumulative discounted sum of future rewards. Improving learning efficiency, an estimate of advantage (A_t) on the other hand measures the difference between the utility of performing an action compared to the expected value under the current policy $V_\phi(s_t)$. In doing so, this aids the algorithm to gauge its ability to

perform better or worse than average and understand how much to weight updates to improve its performance.

Next, PPO calculates the probability ratio between updated (π_θ) and previous ($\pi_{\theta_{old}}$) policies for each action taken during data collection (Equation 15), thus measuring the extent to which the updated policy deviates from the previous policy in terms of assigning probability to the same action in a given state. When $r_t(\theta)$ significantly deviates from 1, it indicates that the updated policy significantly prefers an alternate action than before, hinting at the potential for large, unstable policy updates.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \qquad (15)$$

Avoiding instability, PPO optimises a clipped surrogate objective function in which ϵ defines a trust region around the previous policy to within the bounds of $[1-\epsilon, 1+\epsilon]$ (Equation 16). By use of clipping, smaller incremental updates are used instead of purely exploiting the objective $r_t(\theta)$.

$$\mathcal{L}_{clip}(\theta) = \mathrm{E_t}[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)] \qquad (16)$$

Parallel, the value function network is trained to minimise the coefficient-scaled square error between predicted and observed returns (Equation 17). Acting as the critic, it is trained to provide estimates of expected returns for performing an action.

$$\mathcal{L}_{Value\ function}(\phi) = \beta E_t\left[\left(V_\phi(s_t) - R_t\right)^2\right] \qquad (17)$$

In addition, an entropy term can be included to lessen premature convergence to deterministic policies by rewarding policies that contain an element of randomness in action distribution (Equation 18). Combining all elements together, Equation 19 is optimised through stochastic gradient descent.

$$\mathcal{L}_{entropy}(\theta) = E_t\left[\mathcal{H}[\pi_\theta(\cdot\,|s_t)]\right] \qquad (18)$$

$$\mathcal{L}(\theta, \phi) = \mathcal{L}_{clip}(\theta) + c_1\mathcal{L}_{Value\ function}(\phi) + c_2\mathcal{L}_{entropy}(\theta) \qquad (19)$$

Each update is performed over a series of epochs through mini-batches drawn from a collected batch of experiences to improve sample efficiency and generalisability. Once updated, the new policy is used to collect another batch, in which the process repeats until convergence.

5 Results

Through generating policies using each of the approaches and comparing the performance of each over a horizon, the effectiveness of each method can be evaluated. In pursuit of comparison, each type of policy is developed using an identical cost function and input parameter values described in Section 5.1. Sections 5.2 and 5.3 outline the training process of the respective rule-based and DRL policy methods. Further, the performance of the different techniques is compared by using a small 4-item setting in Section 5.4 before moving towards a 47-item setting in Section 5.5 to test the scalability of the model.

5.1 Parameters and Key Performance Indicators (KPIs)

Through defining the problem setting through parameters, each of the techniques derive outcomes based on the same information. Table 5 lists all cost, forecasting, capacity and lead time data. As alluded to, the ordering costs per container far outweigh the overall relative costs of holding and backordering. In addition, initial forecasted demand, standard deviation, and smoothing parameter are listed exclusively for the use of the DRL methods which include an internal forecast within the input array. Both DRL and rule-based methods will observe stationary compound Poisson demand induced by the order rates and demand distribution shown. Restricting the permissible actions of the DRL methods, a maximum number of items per order is listed. By increasing this value, the possible action space is increased and hence the computation time to evaluate is also increased.

Obtaining solutions using the parameters, the relative performance of each ordering policy will be gauged using a set of key performance indicators (KPIs). The first of these comprise of the associated costs of an ordering system under backordering, namely backorder costs, holding costs, and ordering costs. In more detail, backordering costs are defined as the average expenditure caused by items being out of stock at the time of demand. As such, backorder costs for each item are incurred for each week an item cannot be delivered from stock. Assigning an expenditure to negative inventory levels promotes a cost-driven system to avoid stockouts to achieve a high service level. Further, holding costs are considered the financial penalty for holding a unit of inventory on stock per time period. By assigning costs to the use of space, this simulates the real-world costs of warehousing the goods and avoids the policy from hoarding too much inventory at one time. The last relevant cost – ordering costs – is an estimate of necessary costs to transport a container of items from the vendor to the warehouse. While there are multiple sizes of shipping container, with the knowledge that ordering costs are dominant the largest container will be used to estimate costs.

Parameter Values (4-item)		
Parameter	Description	Value
Backorder Costs	Cost of not fulfilling (backlogged) demand in a period	100
Holding Costs	Cost of holding an item in inventory per time period	1.9
Ordering Costs	Cost of ordering a container	5000
Lead Time	Number of periods between ordering and receiving items	20
Initial Forecast	Array containing demand forecast over lead time duration per item	[12.53, 14.24, 7.39, 2.42]
Initial Standard Deviation	Array containing standard deviation of demand over lead time duration per item	[4.20, 6.47, 1.63, 0.48]
Order Rate	Array containing the probability of demand in a given period	[0.379, 0.333, 0.318, 0.121]
Volume	Array containing the cubic dimension per item	[1.22, 1.14, 1.42, 1.00]
Smoothing Parameter	Used for forecasting, value determining the importance of new demand observations	0.05
Discount Factor	Value that determines the importance of past cost observations	0.99
Capacity	Liquid capacity of a shipping container (m^3)	65
Max. Number of Pallets	Maximum discrete number of pallets of an item to be assigned to a container	80
Demand Distribution	Two-dimensional array containing the probability of order magnitudes per item	[0.56, 0.28, 0.12, 0.04, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
		[0.41, 0.23, 0.18, 0.18, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
		[0.76, 0.24, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
		[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Table 5: Parameter Values for Training Instance (4-item)

In addition to cost metrics, other KPIs such as container fill rate, periodicity, fill rate, and items per order are valuable. Container fill rate measures the average portion of liquid container capacity used. While it would not be feasible to pack a container such that there is no wasted space, there is an industry-accepted rule-of-thumb used to guide ordering decisions. For the largest container size, the liquid volume is understood to be $65m^3$. Next, periodicity measures the proportion of periods in which an order is placed. A lower periodicity would entail that orders are placed less frequently at the vendor. Further, fill rate is defined as the proportion of customer demand directly fulfilled once ordered. If an item has been backordered, this would incur a one-time loss in service. Finally, items per order is a measure of the average number of different items added to an order – serving as an indicator to the range of items filled into a container at each replenishment opportunity.

5.2 Rule-Based Policy Training Process

In training a competitive policy, a simulation-based GA was used to optimise a string of item policies in the form of tuples. Explicitly, for each generation, a population of 200 policies were simulated over a 20-year horizon. After the cost of each were evaluated, the best performing 10% are considered viable parents to generate offspring policies. An elitist aspect maintained that these parent policies were included within the following generation. In addition to ensure that alternate policies were explored, a 20% chance that offspring policies mutated is also implemented. Over a number of generations, this should entail that a well-performing array of tuples containing a joint ordering policy is found. Figure 6 shows an example of such a learning process, plotted with average cost per period and fill rate percentage on the y-axes. As expected, due to backordering costs, the seeming synergistic optimisation hints at how optimising overall cost reflects optimising for fill rate.

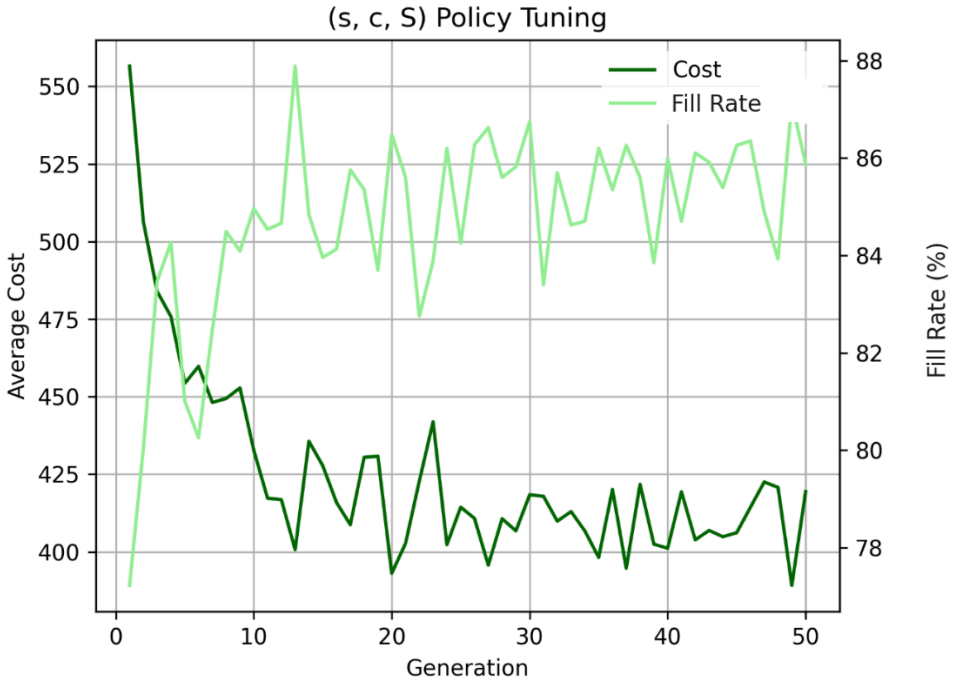


Figure 6: Rule-Based Policy Tuning using Genetic Algorithm Example

Importantly unlike the problem description, can-ordering is unable to directly encapsulate the capacity constraint per period. Consequently, limitations to the parameter values were put in place on a problem-to-problem basis to ensure not more than one container was ordered at a time. In absence of these constraints, it was observed that the algorithm would optimise to order multiple containers within a period, likely to create a trade-off between holding cost and its ability to effectively fill containers.

5.3 DRL Policy Training

To generate the neural network policies via DRL, the MDP developed in Section 4.1.4 is modelled within DynaPlex (Akkerman et. al., 2023). Beyond being designed in C++team to facilitate DCL, DynaPlex has the ability to run different learning approaches such as PPO through its compatibility with Python to leverage libraries such as Tianshou. Describing further, Section 5.3.1 goes over how the PPO approach was trained before Section 5.3.2 talks about the training used to implement DCL.

5.3.1 PPO Training

PPO has many tuneable aspects to facilitate an optimal neural network convergence. While less hyperparameter-dependent than other DRL techniques, the performance of PPO still heavily relies on tinkering with hyperparameter values. Importantly, the inputted values restrict that the magnitude of policy updates by defining a tolerance through clipping, to instigate a greater stability in training. Moreover, the addition of entropy encourages exploration and helps the algorithm not get stuck in local optima.

Further, learning rate and learning rate scheduling respectively sets and gradually adjusts the learning rate over time for improved convergence. Batch size refers to the number of training samples used per iteration and a replay buffer stores past experiences encountered by the agent

to improve learning efficiency. Reward normalisation adjusts the scale of rewards and implementing a discount factor balances the importance of immediate and future rewards. Each of these elements contribute towards training over a number of epochs.

Although PPO can parallelise experience collection across environments, its training speed remains limited by the sequential nature of its policy updates – hence reduces the benefits of using multiple CPU cores. As such, the training process can take place on a personal computer. Table 6 outlines the hyperparameters used in obtaining results, based on setting used by Vanvuchelen et al. (2020).

PPO Hyperparameter Values (4-item)	
Hyperparameter	Value
Nuumber of Layers	3
Hidden Layer Width	[128,128,128]
Loss Cipping	0.2
Number of Epochs	100,000
Replay Buffer Size	1,040
Batch Size	1,040
Entropy Coefficient	0.1
Learning Rate	0.001

Table 6: PPO Hyperparameter Settings

5.3.2 DCL Training

As an alternative way to train a neural network using DRL, DCL has fewer, less delicate hyperparameters to adjust. Firstly, the number of samples N refers to the number of states sampled during the training process. Sampling a greater number of states yields a more effective evaluation of the policy. Further, the number of exogenous samples M defines the number of different scenarios considered for each state-action pair. Increasing the number of exogenous samples prompts more robust policies which can adapt to changes in the environment. Horizon length H dictates the number of time steps each policy is tested over to better understand the long-term consequences of each action. In addition, a warm-up period L can be defined to aid variance associated with the examples at the beginning of a sample horizon before the environment stabilises.

Increasing each hyperparameter enhances the ability of the algorithm to learn an efficient policy, albeit at the cost of greater computational resources. While extensive computational power is necessary, much of it may be parallelised over multiple CPU cores to lessen the time duration of training. As such, to provide learned policies in reasonable time, a cluster was used in the training of DCL over 192 cores using the hyperparameters found in Table 7.

DCL Hyperparameter Values (4-item)	
Hyperparameter	Value
Nuumber of Layers	3
Hidden Layer Width	[128,128,128]
Number of Samples	50,000
Number of Exogenous Samples	2,000
Rollout Horizon Length	500
Warm-up Period	0
Generations	10

Table 7: DCL Hyperparameter Settings

The plot of Figure 7 shows how the policy converges to a lower cost and higher fill rate similar to that of the rule-based GA algorithm. Noticeably, it seems that until the fourth generation there was a struggle to generate an effective policy. Beyond the fifth generation, however, the policy asymptotes after having further explored better approaches to the problem. This converging shape evidences how extensive computational effort is required to develop an improved decision.

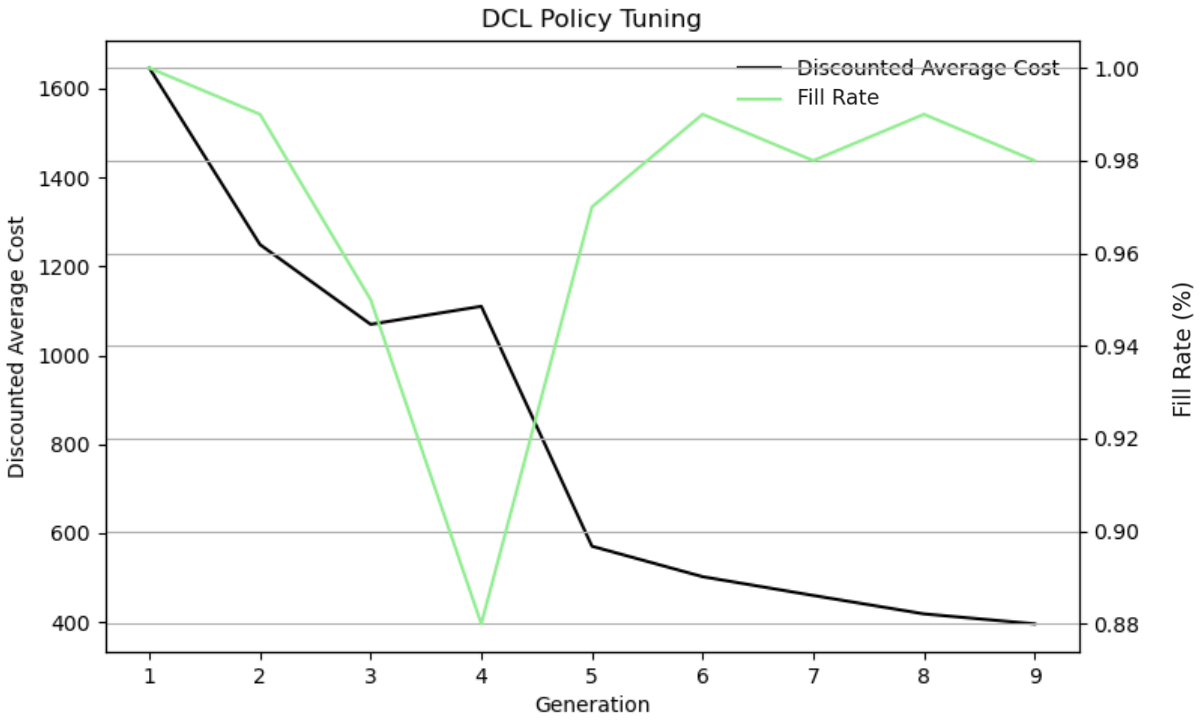


Figure 7: DCL Policy Tuning using DynaPlex

5.4 Performance Results (4-item)

In discerning the differences in performance between the proposed methods, this section primarily generates plots to interpret the differences in decision-making before comparing the policy performance over long-run simulations of both training and test data. As such, Section 5.4.1 firstly presents the performance of the ordering policies using both training and test data. Next, inventory level and inventory position plots of a given item over the same period are shown to study the differences in the sawtooth diagrams in Section 5.4.2. Subsequently, a set of heatmaps are generated to compare the replenishment trigger decisions for each of the methods in Section 5.4.3.

5.4.1 Performance Results

In obtaining an accurate measure of each KPI, the policies are tested using the same demand pattern over a 200-year horizon. Additionally, to understanding performance on training data (Table 8), the use of each policy on test data is assessed (Table 9). Given how the training data comprises of demand throughout the years 2021-2023, the model will be trained on demand influenced by COVID-19. By comparing with data since the start of 2024, an impression on how the policy is able to encapsulate its environment as well as extend to others will be judged.

Training Performance Results (4-item)			
	Can-Order Policy	PPO Policy	DCL Policy
Backorder costs	182	0	12
Holding costs	63	18,504	180
Ordering costs	406	5,000	201
Total costs	651	23,504	393
Container fill rate	49.1%	7.4%	99.1%
Periodicity	0.081	1.000	0.040
Service level	82.4%	100.0%	97.9%
Items per Order	2.21	4.00	3.83

Table 8: Performance Results for Training Data (4-item)

Observing the results, it is evident from a cost perspective that DCL outperforms both can-ordering and PPO policies. Firstly, compared to can-ordering, the DCL policy has a greatly decreased backorder and ordering costs, while holding roughly three times the amount of average inventory. When judged against the PPO policy, DCL obtains marginally higher backorder costs while vastly outperforming on holding and ordering costs.

Importantly, it is also apparent that policies pertaining to lower backorder costs also evidence higher fill rates, suggesting that attributing a high backordering penalty allows the model to train towards systems that serve customers directly from inventory. In an extreme case, such as that presented by PPO, it can prompt policies to converge to a solution which excessively avoids backorder costs through aggressive inventory levels. Looking into it further, it becomes apparent

that the PPO policy trains to a local optimum of ordering the least number of items each period – not opting to forego the chance of ordering a container each period. Consequently, this explains the 7.4% container fill rate, 1.000 periodicity and 4.00 items per order. Due to a combination of low demand and high ordering costs, this leads to a vastly underperforming policy.

Turning attention back to the remaining policies, the results suggest that the can-order policy decides to ship frequent less-than-container loads (LCL) while DCL optimises to fill containers at each replenishment opportunity and order half as often. This goes to exemplify the inverse relationship between container fill rate and periodicity. Intuitively this makes sense as double the amount of half-full containers will be necessary to fulfil the same demand through full container shipments. Given that ordering is considered the dominating cost, this explains the improved overall cost performance of DCL compared to can-ordering. Finally, comparing the average number of items added to an order is lower using can-ordering as opposed to DCL. Given a comparable container fill rate this would imply that less of each item is added to each order, generating a knock-on effect of lowering overall stock levels.

Shifting focus towards test data, it becomes apparent that the average demand of each item decreases across the board by between 2.2-32.4% post COVID-19. In this case, it can be expected that static policies which do not adapt to the environment continue to order based on the heightened demand of the training data. Evidently, this is what happens with the can-order policy as its order periodicity decreases while fill rate increases. In addition, an almost 20% reduction in overall costs, mostly due to decreased backorder costs and ordering costs, is observed.

Test Performance Results (4-item)			
	Can-Order Policy	PPO Policy	DCL Policy
Backorder costs	115	0	14
Holding costs	72	21,106	169
Ordering costs	335	5,000	175
Total costs	522	26,106	358
Container fill rate	52.0%	7.4%	99.2%
Periodicity	0.067	1.000	0.035
Service level	86.6%	100.0%	97.6%
Items per Order	2.25	4.00	3.76

Table 9: Performance Results for Test Data (4-item)

Comparing these results to DCL, curiously despite the reduction in demand the fill rate is shown to marginally decrease, while a slight improvement in container fill rate and decrease in order periodicity is observed. These results hint towards the ability of the policy to adapt to changes in demand patterns to maintain an effective performance. Backed up by a circa 8% reduction in overall costs between the training and test scenarios, suggesting its ability to adapt without retraining.

Both can-order and DCL policies continue to order a similar number of items per order between training and test datasets. Looking into the data further, the can-order approach appears to equally distribute the probability of ordering each item, while DCL seems to prefer items with a lower stock keeping unit (SKU) number (Figure 8). While this may be favourable for items which are ordered according to a priority classification, when no such distinction is made between items this mechanism is undesired. Alternatively, the difference shown by item 4 can perhaps be also purposeful and rather explained by its significantly lesser average demand.

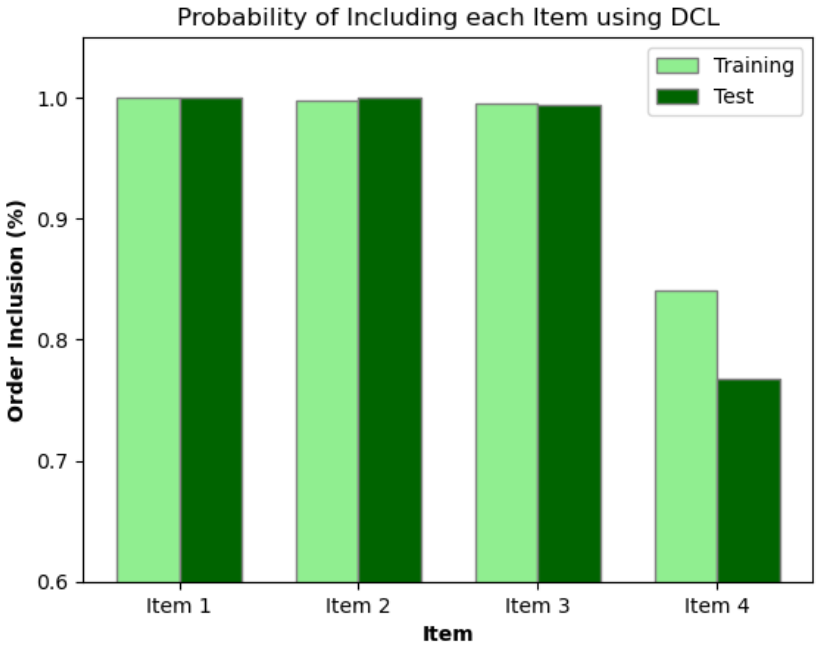


Figure 8: Proportion of Orders containing each Item using DCL (4-item)

Finally, once again, the PPO policy exhibits the same characteristic of ordering a and thus vastly underperforms. Due to this, the technique will be excluded from further analysis within the 4-item setting.

5.4.2 Inventory Plots

In gathering intuition and insight into the differences between the inventory policies, the inventory level and inventory position can be plotted over time. In the diagrams, inventory level increases once an order arrives at the warehouse and inventory position increases once an order is placed at the supplier. Both inventory level and inventory position decrease due to sampling from a demand distribution. Figures 9 and 10 show the inventory plots of item 1 for can-order and DCL policies respectively, both exerpted from the same two-year period.

Reading into the ordering decisions of the can-ordering plot, the replenishment policy seems quite rhythmic as the number of periods between replenishment orders are quite equal and regular. In addition, it is noticable how the plot of inventory position is elevated from that of inventory level. This suggests that there commonly seems to be an in-transit order at any moment im time. Beyond this, the inventory level seems to regularly dip below zero; indicating how there is a short out-of-stock period in most replenishment cycles which will induce backordering as previously shown.

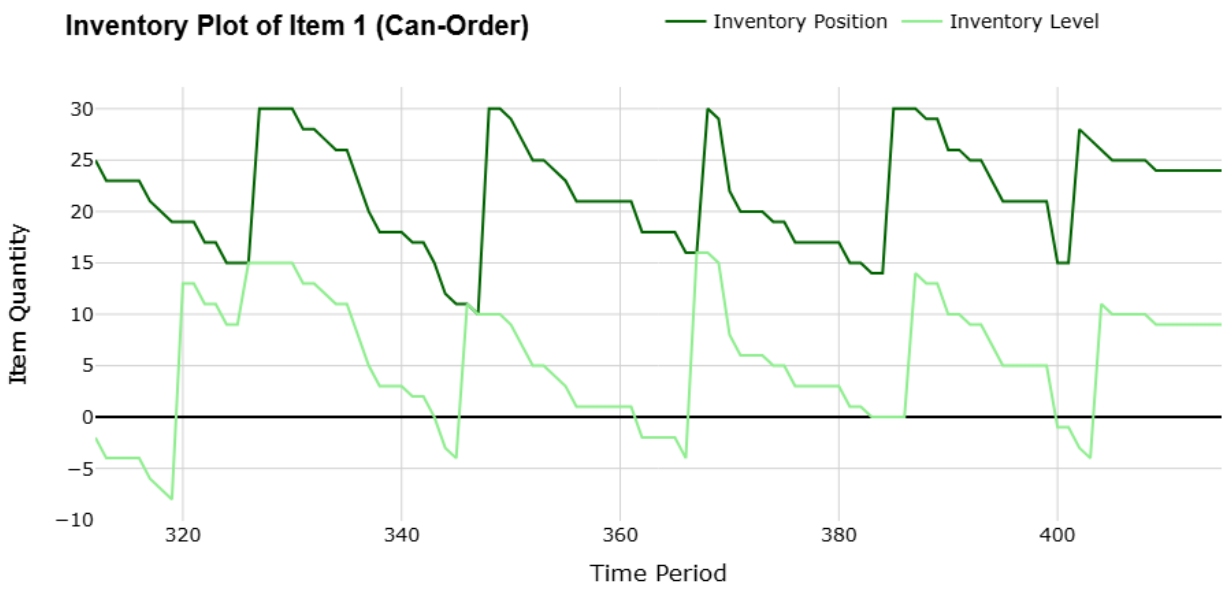


Figure 9: Can-Order Replenishment Policy Inventory Plot

Turning attention to the DCL inventory plot shown in Figure 10, it shows that the inventory level is on average higher and never falls below zero. In addition, there are extended periods in which the lines of inventory level and position merge, indicating that in juxtaposition to the can-order policy, there is not always an order in transit. Similarly, there also seems to be an irregularity between replenishments. Towards the beginning of the plot, it shows there to be a period of no order for about half a year, before ordering twice within the next half year period. In fact, in the latter period it shows how there are two in transit inventories at a single point in time. Interestingly, the plot also shows order magnitudes which differentiate between orders. Combining this with the knowledge of a high container fill rate, it becomes evident that the DCL policy does not learn a single way to effectively use the capacity of a container but also adapts to the environment to act appropriately.

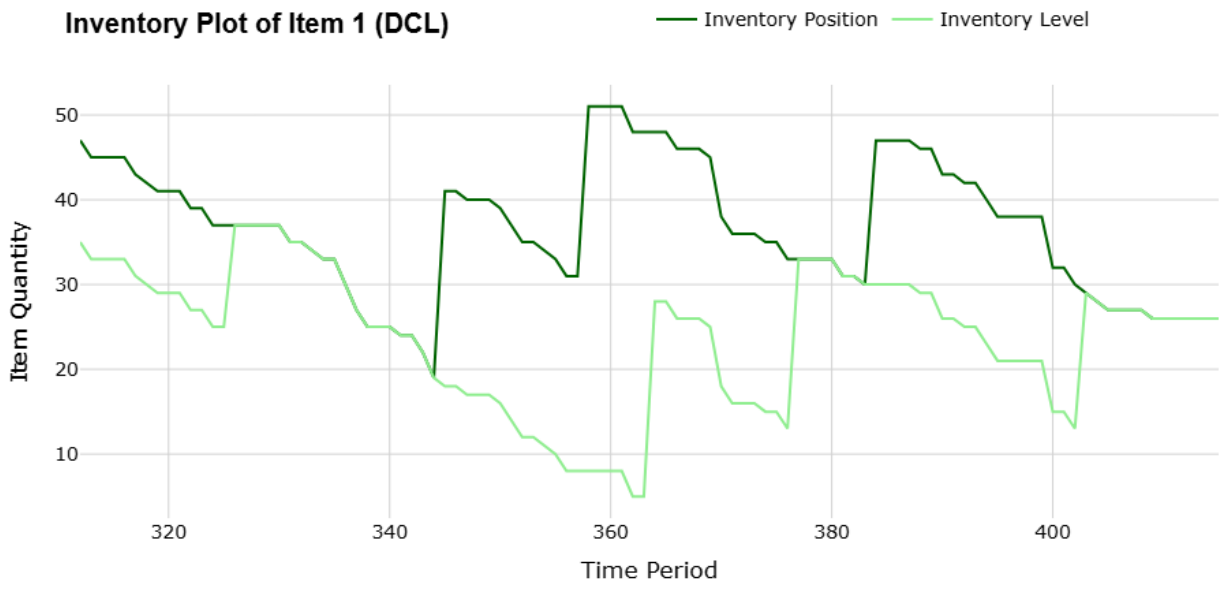


Figure 10: DCL Replenishment Policy Inventory Plot

5.4.3 Heatmaps

Another way to gain insight into the decisions of a neural network policy is to generate heatmaps. Visualising data through regions of colour, in this case heatmaps can be used to show the scenarios in which orders are triggered. Figure 11 depicts how the inventory position of item 1 influences the trigger of an order at various inventory positions of other items. Each of the items not considered within the diagrams is set to an average value. Notably, can-ordering and DCL policies interpret inventory differently. Can-ordering optimises directly based on inventory position, whereas DCL is provided its two components: inventory level and in-transit inventory. Acknowledging this discrepancy, setting the in-transit inventory to zero equates inventory level and position to enable a fair comparison of order triggers.

As expected, the can-order triggers are very rectangular, exhibiting a very distinct L-shape region in which an order should be made. This is due to how an order is triggered if either of the item inventory positions are below a reorder point. As item 2 and item 3 have the same reorder points, it shows that their respective can-order trigger plots are identical. Overall, this predictable shape aids its interpretability – allowing a clear insight into when an item should be ordered only with the knowledge of inventory position for each item.

Moving towards DCL, it becomes apparent that the order triggers loosely resemble that of the can-order policy yet involve an element of interaction between the respective inventory positions. This influence, shown by the slight curve towards the origin better accommodates for the situation in which both items are nearing replenishment. It also appears that there is a larger area of green using the DCL policy, hinting towards how orders are placed at higher inventory positions. Linking back to the results in Tables 8 and 9, it suggests a reason as to why DCL obtained a high fill rate with higher inventory costs. Interestingly, unlike the can-order triggers, the inventory position trigger changes depending on the item it is being compared against.

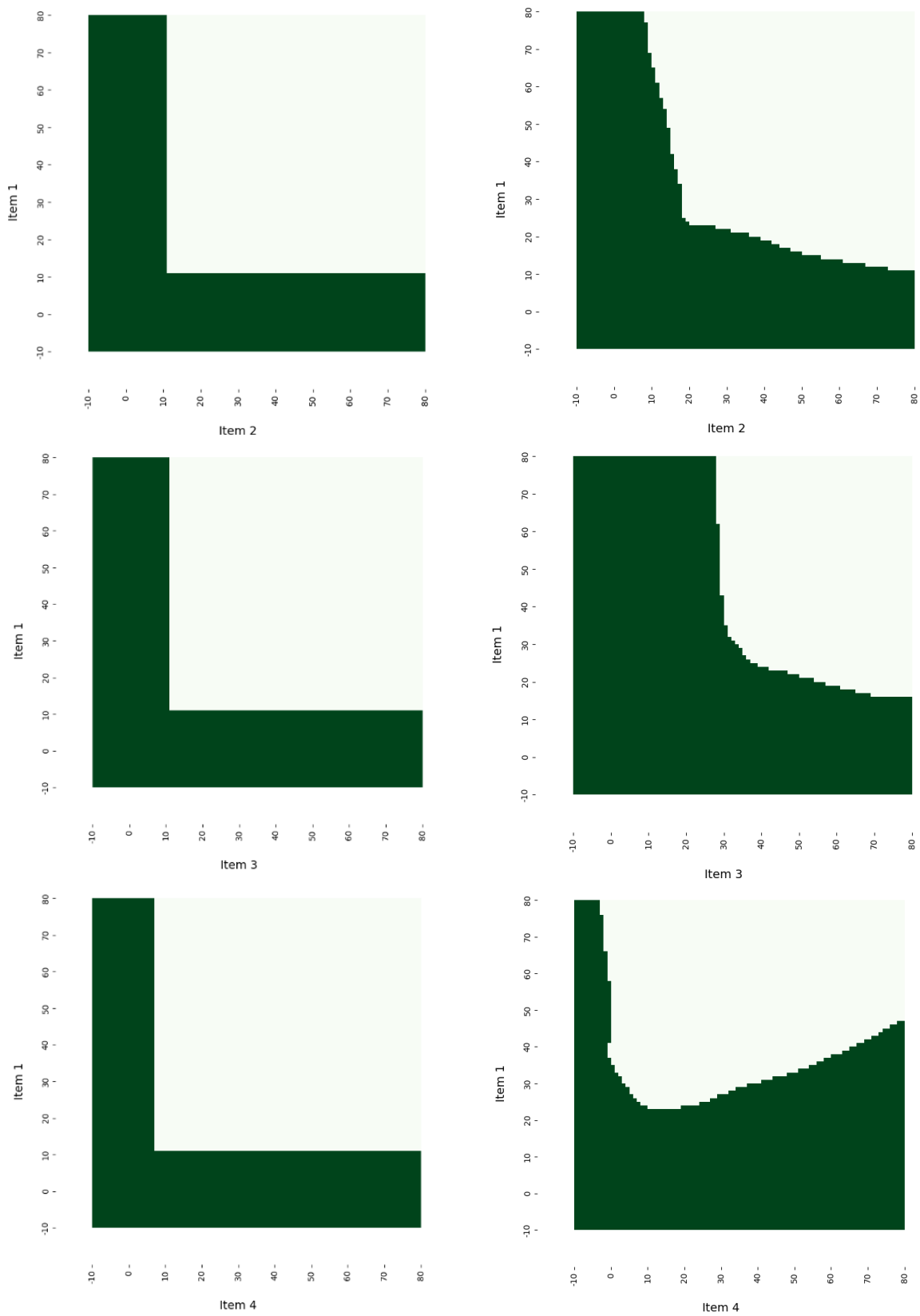


Figure 11: Item Order Triggers using Can-Order Policy (Left) and DCL Policy (Right)

5.5 Performance Results (47-item)

Having obtained successful small-scale test results, in order to see if the model scales it should be tested with a larger problem setting. As such, transitioning to a case with an increased number of items requires redefining the respective demand and volumes. In doing so, however, the complexity of the problem increases dramatically due to the state and action space expansion. Subsequently, this gives no guarantee that a suitable solution will be found using a reasonable amount of compute.

Firstly, the can-order policy was observed to be circa three times slower in training when using the same hyperparameters used in the 4-item problem. Notably, however, despite appropriately restricting the policy parameters, the GA failed to produce a solution which abided by the capacity constraint. Training results suggested that an average of 4.28 containers were included in a single order, exemplifying how the can-ordering adapts through its inability to effectively fill a single container by spreading this out between a greater number of containers and vastly inflating inventory levels. Relaxing the capacity constraint, the can-ordering policy would be able to achieve between a 97.5-98.5% fill rate (Figure 12). While also having a mean 89.9% container fill rate and average periodic cost of around 2,600, it is infeasible within the problem setting and thus is excluded from further results.

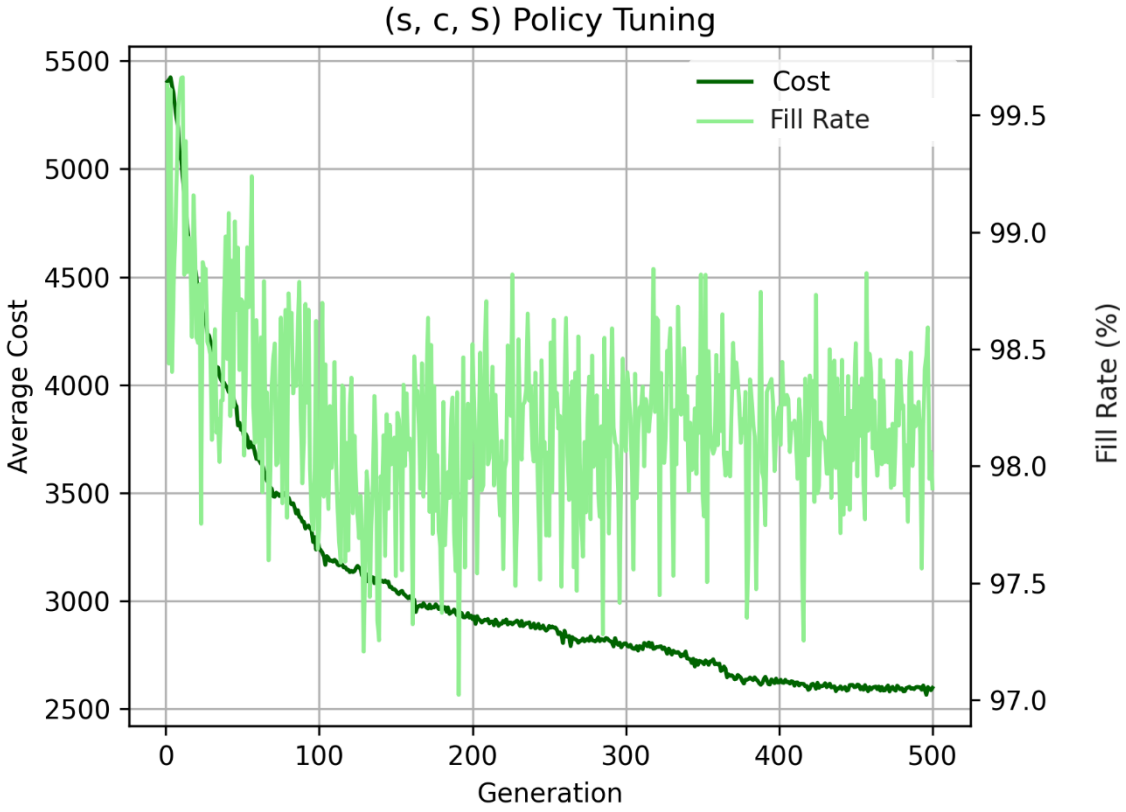


Figure 12: Progression of GA in Tuning 47-item Policy

In training PPO and DCL, adaptations were made to the architecture of the neural network from three layers of 128 nodes to 512. Due to the large expansion of the input array, maintaining the same structure would result in information being lost when being processed by the neural network. Otherwise, training PPO utilised the identical hyperparameters and there was a marginal

increase in computation time. On the other hand, moving towards an increased number of items caused the training time of DCL to sharply increase. Due to this, the number of samples N is selectively reduced from 50,000 to 30,000; and the number of exogenous samples M was decreased from 2000 to 1000. In addition to these reductions, the number of CPU cores used increased from 192 to 960, and the allocated time was increased from 48 hours to 120. Despite these accommodations, the model failed to train completely beyond the third generation. Nevertheless, the PPO and DCL policies were once more tested on training (Table 10) and test data (Table 11) once more to evaluate their respective performance.

Training Performance Results (47-item)			
	Can-Order Policy	PPO Policy	DCL Policy
Backorder costs	-	76.959	3,120,981
Holding costs	-	369.727	818.565
Ordering costs	-	5,000	5,000
Total cost	-	451.686	3,944,546
Container fill rate	-	67.8%	99.5%
Periodicity	-	1.000	1.000
Service level	-	77.8%	34.1%
Items per Order	-	47.00	6.18

Table 10: Performance Results for Training Data (47-item)

Test Performance Results (47-item)			
	Can-Order Policy	PPO Policy	DCL Policy
Backorder costs	-	106.522	4,313,501
Holding costs	-	348.672	821.044
Ordering costs	-	5,000	5,000
Total cost	-	460.194	5,139,545
Container fill rate	-	67.8%	99.5%
Periodicity	-	1.000	1.000
Service level	-	81.7%	28.1%
Items per Order	-	47.00	6.05

Table 11: Performance Results for Test Data (47-item)

Evidently, both PPO and DCL policies were unable to define appropriate actions for the enlarged problem. Once more, PPO optimised to order a single pallet of each item in each time period. As demand for most items is less than the amount being replenished, holding costs are dominant.

Interestingly, however, the policy optimises to not order more items for those exhibiting a greater average demand per period than one. Resulting is an accumulation of backorder costs, not present in the 4-item setting, for uncatered backordered demand.

Starkly different to before, DCL is seen to be vastly outperformed by PPO in both overall costs and fill rate. In further juxtaposition, DCL is also observed to order each period without fail. Given the low fill rate and extreme backorder costs, this would suggest that only a select number of items were ever ordered. Further analysis would suggest only six of the products are consistently ordered, while others are rarely ordered, if at all (Appendix 2). As such, there seems to not be a pattern of preferentially ordering items with a lower SKU number as previously hypothesised. Instead, it indicates that the difference in order inclusion shown in Figure 8 is a result of the lesser demand of the last item. Positively, it is evident that the algorithm is still able to fill containers effectively, though given that the policy dictates there to be an order each period, this would result in a large surplus of items, reflected in the holding costs.

6 Conclusion and Discussion

In describing and summarising the key points of this work, Section 6.1 succinctly highlights its findings. Section 6.2 discusses poignant parts of the research, mostly linking to its approach, which point out its limitations. Section 6.3 continues by outlining the overarching implications of the research for the business before Section 6.4 sets out possible directions for future work.

6.1 Conclusion

Generating policies to control inventory have long been discussed in academic literature, but its complexity often requires customised approaches to generate an apt solution. Further, practitioners contain a vested interest in developing renewed ways to streamline ordering systems. Existing efforts have focused on a simplified version of reality which most notably exclude lead time durations in developing state-of-the-art solutions. This practical oversight hinders the ability of the models being utilised in a real-life setting.

Consequently, this work focuses on bridging the gap between toy-like problems presented in literature and reality to propose an adaptable approach to solving the stochastic joint replenishment problem with non-zero lead time and capacity constraints. When applied to a specialised small problem instance, the model dominates other approaches across all but one of the metrics. In addition, switching between training and test demand data further evidenced the capability of the model to adapt to new situations without the need for retraining. Expanding to a larger problem, the solution was limited by its computational efficiency in generating an acceptable policy. Accordingly, this highlights the opportunity for further work in refining or adapting the model to improve its ability to efficiently generate solutions that can directly be applied to real-world logistics systems.

6.2 Discussion

In facilitating the application of DCL to a real-world inventory application, various assumptions or decisions were made. The first of which pertains to how empirical demand distributions were used to define the magnitude of demand for an underlying compound Poisson process. The choice for this approach was primarily driven by irregularities of demand found within the data. Each of these abnormalities, though could be driven by actual demand, more likely are a result of stockout occasions. Within retail, if there is a stockout of items across stores there is a process of recovery demand by which items are ordered to fill store shelves rather than cover for accumulated demand. While the effects of this can be mitigated by satisfying almost all demand within reasonable time, in reality this mechanism will continue to persist. As such, traditional theoretical distributions fail to encapsulate this properly. In any case, sampling from empirical or theoretical distributions typically assumes that samples are independent and identically distributed (i.i.d.). Acknowledging reason for the handful of inflated demand instances, it can be argued that after observing recovery demand, a similar magnitude of demand would be unexpected – ultimately violating the assumption. As the proposed method, however, includes an element of internal forecasting, the negative effects of this are considered to be minimal.

In addition to how the demand is modelled, there are other more complicated variants to demand that were investigated. The types of products investigated were in part chosen due to their stability. This means that the items would not undergo seasonal effects or promotions, both of

which are prevalent within selling items in retail. Had either been introduced, it would be expected that the model would have failed to learn an effective way to accommodate given the current setup.

Another important aspect of the approach assumes that there is infinite capacity at the assigned warehouse. While there would be ample warehouse capacity in assigning additional pallet bays within a warehouse to slow-moving items, the same sentiment would likely not extend to faster-moving goods. Applying this approach to fast moving items would ideally require a fewer number of periods contained within the lead time to enable a more flexible ordering system. In addition, it is assumed that each of the vendors will always deliver items within a set deterministic lead time. Not only could this be violated for instances of delay, but there are also seasonal closures of some factories that have not been considered.

On the machine learning side of things, DCL uses a random sample of states to test the effectiveness of actions. Due to the vast state space, primarily contributed to by extensive in-transit inventory information, there are many states that were likely tested which would never become reality. During the training, there was little undertaken to facilitate training on more probable states within the state space. For example, testing a policy on a random initial state when there is in-transit inventory for each time step within the lead time would not be a true representative when there are at most one or two deliveries in transit at any point in time. Being a cost-driven approach, this misguided impression of reality could ill-assign positive or negative impressions to policies which receive an unlucky initial state. While this problem was attempted to be mitigated using long testing horizons, other mechanisms such as an induced warm-up period would enable the state to stabilise before collecting any cost inferences. The effects of either altering training duration or warm-up periods were not tested in the current work.

Finally, neural networks remain to be a black-box solution – requiring unrelenting trust from the user in implementation. While the initial experiments showed promise, adoption of the approach would require careful observation. Alternately, this work provides support for the decomposition of item order triggers and filling containers. In doing so, it would render the agent generated responsible for only filling containers, instead of also deciding on which items to order. This would allow businesses a tangible understanding of part of the order triggers, leaving the job of the agent to effectively fill containers on request.

6.3 Recommendations

With regard to implementation within ADIL, it seems too early to recommend the direct implementation of the method developed within this work. While it is evidenced to provide impressive performance for one vendor, it is unable to show the same level of performance to other suppliers in a similar category. Furthermore, due to the scale that the department operates on, the amount of goods ordered from vendors regularly exceed the assumed one container per period capacity applied. This would entail that different systems would be necessary to serve for different supply regions, adding unnecessary complication.

Further, a useful aspect which remained unincorporated is how ADIL have downstream transparency into subsidiary inventory levels. With this information, it is more easily understood when a replenishment order will be necessary, instead of just using its own inventory level and in-transit inventories as indicators. Modelling this into an MDP that could be analysed would lead to

a greatly increased complexity, thus should either be excluded from the generation of planning agents or other methods should be sought out.

6.4 Future research

Importantly, this work treads a step further in generating agents which are able to overtake the role of the supply planner. With the potential to free up and consequently divert human resources to other activities, further effort towards the practical feasibility of this approach seems beneficial. One complication halting its immediate implementation is how some complexities of real-life are yet to be incorporated. An example prevalent in retail, the effects of testing the influence of promotions are not yet studied. During promotional activity, there is a sharp and otherwise unexplainable increase in demand. While the current model includes a forecast within its features, the impact of changing this input variable would be key to understating if the current approach is sufficient.

Provided how the proposed approach did not have any design elements or tricks to ensure it performed well in a specialised setting. Beyond there being no existing approach studied in literature which offers a comparable baseline to this problem, doing so would inherently limit the applicability to a wider scope of inventory problems. Hence, it would be interesting to see how this method performs in simpler, more generic replenishment problems. In doing so, it would be possible to observe its performance against (other) rule-based policies in the proper settings they were designed for.

In addition, it would be interesting to gauge the pooling effect of including a different number of item subsets for a given vendor. While logically it may be more advantageous to consider more items in an order, it could cause to be more complex than it is worth with respect to implementation. In that light, observing the effects of using item groupings could pave way for a greater feasibility for widespread implementation.

Further, the method in its current form proves to be quite computationally expensive. This opens up many different avenues of improvement. Firstly, the model can be decomposed into the activities best performed by a neural network, such as filling a container appropriately, and those that are perhaps supplementary. Transitioning to an order trigger approach akin to rule-based methods as a parameter when training a neural network could relieve the burden of deciding what to order and when. This would improve the interpretability for business professionals who would use this system.

Finally, effort should be placed in understanding effective parameters to train decent DCL solutions. DCL is inherently quite computationally expensive, which hinders its ability for widespread adoption. Other approaches to circumvent the consistently high computational expense could be to divert large amounts of resources to generate teacher policies using DCL, later to use reward shaping in combination with other less intense DRL techniques such as PPO to efficiently retrain and adapt existing knowledge to renewed scenarios.

7 References

Akkerman, F., Begnardi, L., Lo Bianco, R., Temizoz, T., Mes, M., & van Jaarsveld, W. (2023). DynaPlex [Computer software]. GitHub. https://github.com/DynaPlex/DynaPlex

Akkerman, F., Knofius, N., Matthieu, V. D. H., & Mes, M. (2024). Solving Dual Sourcing Problems with Supply Mode Dependent Failure Rates. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2410.03887

Atkins, D. R., & Iyogun, P. O. (1988). Periodic versus "Can-Order" policies for Coordinated Multi-Item inventory systems. Management Science, 34(6), 791–796. https://doi.org/10.1287/mnsc.34.6.791

Balintfy, J. L. (1964). On a basic class of Multi-Item inventory problems. Management Science, 10(2), 287–297. https://doi.org/10.1287/mnsc.10.2.287

Boute, R. N., Gijsbrechts, J., Van Jaarsveld, W., & Vanvuchelen, N. (2021). Deep reinforcement learning for inventory control: A roadmap. European Journal of Operational Research, 298(2), 401–412. https://doi.org/10.1016/j.ejor.2021.07.016

Bretthauer, K. M., Shetty, B., Syam, S., & Vokurka, R. J. (2006). Production and inventory management under multiple resource constraints. Mathematical and Computer Modelling, 44(1–2), 85–95. https://doi.org/10.1016/j.mcm.2005.12.009

Cachon, G. (2001). Managing a retailer's shelf space, inventory, and transportation. Manufacturing & Service Operations Management, 3(3), 211–229. https://doi.org/10.1287/msom.3.3.211.9893

Chaharsooghi, S. K., Heydari, J., & Zegordi, S. H. (2008). A reinforcement learning model for supply chain ordering management: An application to the beer game. Decision Support Systems, 45(4), 949–959. https://doi.org/10.1016/j.dss.2008.03.007

Creemers, S., & Boute, R. (2022). The joint replenishment problem: Optimal policy and exact evaluation method. European Journal of Operational Research, 302(3), 1175–1188. https://doi.org/10.1016/j.ejor.2022.02.005

Das, T. K., Gosavi, A., Mahadevan, S., & Marchalleck, N. (1999). Solving Semi-Markov decision problems using average reward reinforcement learning. Management Science, 45(4), 560–574. https://doi.org/10.1287/mnsc.45.4.560

De Moor, B. J., Gijsbrechts, J., & Boute, R. N. (2021). Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. European Journal of Operational Research, 301(2), 535–545. https://doi.org/10.1016/j.ejor.2021.10.045

Demizu, T., Fukazawa, Y., & Morita, H. (2023). Inventory management of new products in retailers using model-based deep reinforcement learning. Expert Systems With Applications, 229, 120256. https://doi.org/10.1016/j.eswa.2023.120256

Ehrenthal, J., Honhon, D., & Van Woensel, T. (2014). Demand seasonality in retail inventory management. European Journal of Operational Research, 238(2), 527–539. https://doi.org/10.1016/j.ejor.2014.03.030

Giannoccaro, I., & Pontrandolfo, P. (2002). Inventory management in supply chains: a reinforcement learning approach. International Journal of Production Economics, 78(2), 153–161. https://doi.org/10.1016/s0925-5273(00)00156-0

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., & Zhang, D. J. (2022). Can deep reinforcement learning improve inventory Management? performance on lost sales, Dual-Sourcing, and Multi-Echelon problems. Manufacturing & Service Operations Management, 24(3), 1349–1368. https://doi.org/10.1287/msom.2021.1064

Gürbüz, M. Ç., Moinzadeh, K., & Zhou, Y. (2007). Coordinated replenishment strategies in Inventory/Distribution Systems. Management Science, 53(2), 293–307. https://doi.org/10.1287/mnsc.1060.0627

Hachaïchi, Y., Chemingui, Y., & Affes, M. (2020). A Policy Gradient Based Reinforcement Learning Method for Supply Chain Management (pp. 135–140). https://doi.org/10.1109/ic_aset49463.2020.9318258

Hong, S., & Kim, Y. (2009). A genetic algorithm for joint replenishment based on the exact inventory cost. Computers & Operations Research, 36(1), 167–175. https://doi.org/10.1016/j.cor.2007.08.006

Johansen, S. G., & Melchiors, P. (2003). Can-order policy for the periodic-review joint replenishment problem. Journal of the Operational Research Society, 54(3), 283–290. https://doi.org/10.1057/palgrave.jors.2601499

Kara, A., & Dogan, I. (2017). Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. Expert Systems With Applications, 91, 150–158. https://doi.org/10.1016/j.eswa.2017.08.046

Kaspi, M., & Rosenblatt, M. J. (1991). On the economic ordering quantity for jointly replenished items. International Journal of Production Research, 29(1), 107–114. https://doi.org/10.1080/00207549108930051

Khouja, M., Michalewicz, Z., & Satoskar, S. S. (2000). A comparison between genetic algorithms and the RAND method for solving the joint replenishment problem. Production Planning & Control, 11(6), 556–564. https://doi.org/10.1080/095372800414115

Li, L., & Schmidt, C. P. (2019). A Stochastic Joint Replenishment Problem with Dissimilar Items. Decision Sciences, 51(5), 1159–1201. https://doi.org/10.1111/deci.12380

Li, Y. (2018). Deep reinforcement learning. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1810.06339

Meisheri, H., Sultana, N. N., Baranwal, M., Baniwal, V., Nath, S., Verma, S., Ravindran, B., & Khadilkar, H. (2021). Scalable multi-product inventory control with lead time constraints using reinforcement learning. Neural Computing and Applications, 34(3), 1735–1757. https://doi.org/10.1007/s00521-021-06129-w

Melchiors, P. (2002). Calculating can-order policies for the joint replenishment problem by the compensation approach. European Journal of Operational Research, 141(3), 587–595. https://doi.org/10.1016/s0377-2217(01)00304-6

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1312.5602

Mohamadi, N., Niaki, S. T. A., Taher, M., & Shavandi, A. (2023). An application of deep reinforcement learning and vendor-managed inventory in perishable supply chain management. Engineering Applications of Artificial Intelligence, 127, 107403. https://doi.org/10.1016/j.engappai.2023.107403

Nielsen, C., & Larsen, C. (2004). An analytical study of the Q(s,S) policy applied to the joint replenishment problem. European Journal of Operational Research, 163(3), 721–732. https://doi.org/10.1016/j.ejor.2004.02.003

Oroojlooyjadid, A., Nazari, M., Snyder, L. V., & Takáč, M. (2021). A deep Q-Network for the beer game: deep reinforcement learning for inventory optimization. Manufacturing & Service Operations Management, 24(1), 285–304. https://doi.org/10.1287/msom.2020.0939

Özkaya, B. Y., Gürler, Ü., & Berk, E. (2006). The stochastic joint replenishment problem: A new policy, analysis, and insights. Naval Research Logistics (NRL), 53(6), 525–546. https://doi.org/10.1002/nav.20147

Riezebos, J., & Zhu, S. X. (2019). Inventory control with seasonality of lead times. Omega, 92, 102162. https://doi.org/10.1016/j.omega.2019.102162

Rolf, B., Jackson, I., Müller, M., Lang, S., Reggelin, T., & Ivanov, D. (2022). A review on reinforcement learning algorithms and applications in supply chain management. International Journal of Production Research, 61(20), 7151–7179. https://doi.org/10.1080/00207543.2022.2140221

Rummery, G. A. (1994). On-line Q-learning using Connectionist systems. CTIT Technical Reports Series. https://ci.nii.ac.jp/naid/10013482118

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy optimization Algorithms. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1707.06347

Schultz, H., & Johansen, S. G. (1999). Can-order policies for coordinated inventory replenishment with Erlang distributed times between ordering. European Journal of Operational Research, 113(1), 30–41. https://doi.org/10.1016/s0377-2217(97)00431-1

Silver, E. A. (1974). A control system for coordinated inventory replenishment. International Journal of Production Research, 12(6), 647–671. https://doi.org/10.1080/00207547408919583

Sutton, R., & Barto, A. (2005). Reinforcement Learning: An Introduction. IEEE Transactions on Neural Networks, 16(1), 285–286. https://doi.org/10.1109/tnn.2004.842673

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. Neural Information Processing Systems, 12, 1057–1063. http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf

Tanrikulu, M. M., Şen, A., & Alp, O. (2009). A joint replenishment policy with individual control and constant size orders. International Journal of Production Research, 48(14), 4253–4271. https://doi.org/10.1080/00207540802662904

Temizöz, T., Imdahl, C., Dijkman, R., Lamghari-Idrissi, D., & Van Jaarsveld, W. (2025). Deep Controlled Learning for inventory control. European Journal of Operational Research. https://doi.org/10.1016/j.ejor.2025.01.026

van Eijs, M. (1994). On the determination of the control parameters of the optimal can-order policy. *Zeitschrift Für Operations-Research, 39(3), 289–304. https://doi.org/10.1007/bf01435459*

van Hezewijk, L., Dellaert, N., & Jaarsveld, W. (2024). Scalable Deep Reinforcement Learning in the Non-Stationary Capacitated Lot Sizing Problem. https://dx.doi.org/10.2139/ssrn.4846298

Vanvuchelen, N., Gijsbrechts, J., & Boute, R. (2020). Use of proximal policy optimization for the joint replenishment problem. Computers in Industry, 119, 103239. https://doi.org/10.1016/j.compind.2020.103239

Visentin, A., Prestwich, S., Rossi, R., & Tarim, S. A. (2023). Stochastic dynamic programming heuristic for the (R,s,S) policy parameters computation. Computers & Operations Research, 158, 106289. https://doi.org/10.1016/j.cor.2023.106289

Viswanathan, S. (1997). Note. Periodic Review (s, S) Policies for Joint Replenishment Inventory Systems. Management Science, 43(10), 1447–1454. https://doi.org/10.1287/mnsc.43.10.1447

Wang, L., Dun, C., Bi, W., & Zeng, Y. (2012). An effective and efficient differential evolution algorithm for the integrated stochastic joint replenishment and delivery model. Knowledge-Based Systems, 36, 104–114. https://doi.org/10.1016/j.knosys.2012.06.007

Wang, L., He, J., Wu, D., & Zeng, Y. (2012). A novel differential evolution algorithm for joint replenishment problem under interdependence and its application. International Journal of Production Economics, 135(1), 190–198. https://doi.org/10.1016/j.ijpe.2011.06.015

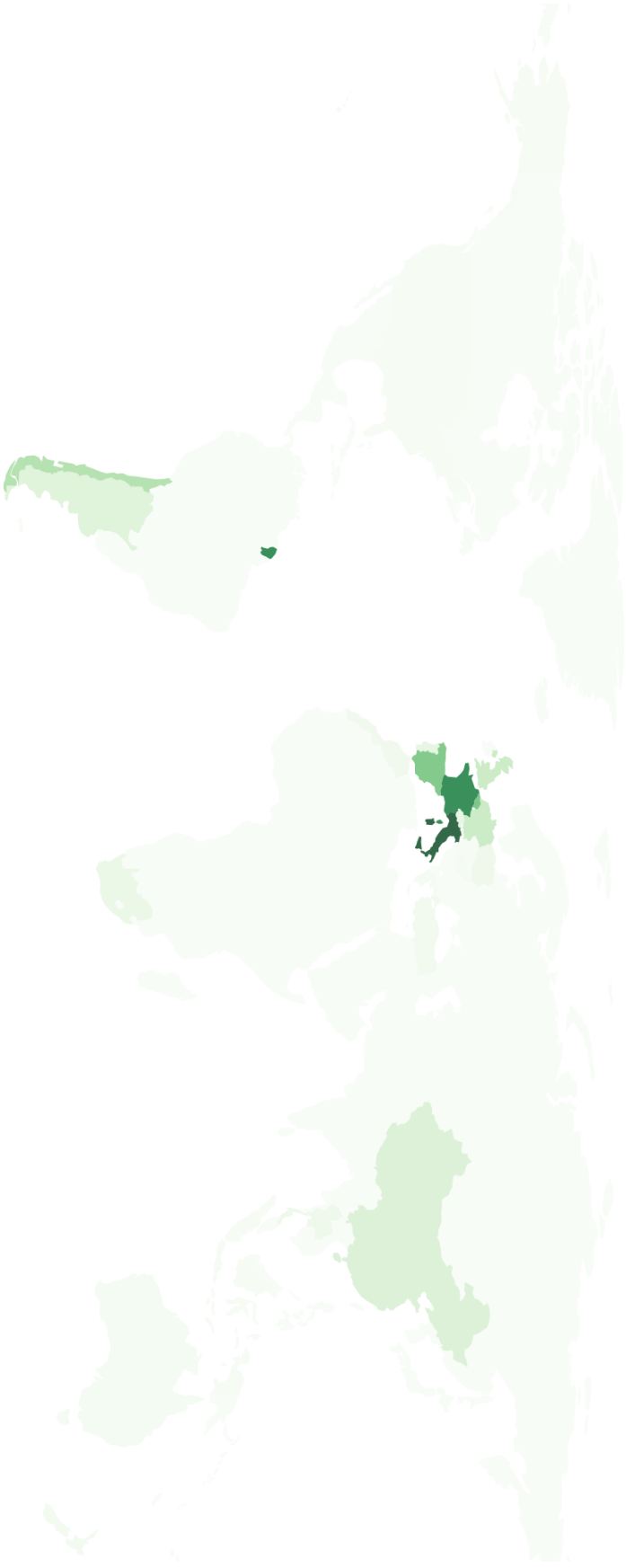
Yin, H., Jiang, Q., Ruan, J., & Zhang, C. (2022). A reinforcement learning method for inventory control under state-based stochastic demand. 2022 5th World Conference on Mechanical Engineering and Intelligent Manufacturing (WCMEIM), 527–532. https://doi.org/10.1109/wcmeim56910.2022.10021568

Zheng, Y. (1994). Optimal Control Policy for Stochastic Inventory Systems with Markovian Discount Opportunities. Operations Research, 42(4), 721–738. https://doi.org/10.1287/opre.42.4.721

Zwaida, T. A., Pham, C., & Beauregard, Y. (2021). Optimization of inventory management to prevent drug shortages in the hospital supply chain. Applied Sciences, 11(6), 2726. https://doi.org/10.3390/app11062726

8 Appendices

8.1.1 Appendix 1: Global Choropleth Map of Total Item Volume



8.1.2 Appendix 2: Probability of Including each Item using DCL (47-item)