Deep Reinforcement Learning to solve the Tactical Production-Inventory Planning Problem in Serial Multi-Echelon Supply Chains

by

Tjum van Dijck (SNR: 2015232)

A thesis submitted in partial fulfillment of the requirements for the degree of
Master in Business Analytics and Operations Research

Tilburg School of Economics and Management
Tilburg University

Supervised by: dr. ir. M.P.M. Hendriks

March 10, 2023
Tilburg

Acknowledgements

Throughout the journey of the past months that resulted in this thesis, I have received much support and assistance from many great people. For this, I am very grateful and would like to thank you.

First of all, I would like to thank Maarten Hendriks and Tijn Fleuren from ASML for their amazing guidance over the past months. Our daily check-ins made me feel part of the team and kept me motivated from the start to finish. Not only were they fun, but I also learned a lot from the discussions that we had during the check-ins. Thank you for sharing the same enthusiasm for the project, which contributed to my choice to further explore the possibilities of this research.

Secondly, I also would like to thank Willem van Jaarsveld and Tarkan Temizoz from the TU/e for all their assistance throughout the whole period. I always looked forward to our meetings in which you shared your nearly unlimited knowledge in this research area and helped me overcome difficulties in understanding the methods and algorithm. I truly appreciate that I could always approach you with my questions.

Furthermore, I would like to take this opportunity to thank Yasemin Merzifonluoglu for reviewing this thesis as a second reader. I really appreciate you spending your time evaluating this work.

Last but definitely not least, I would like to thank my friends and family for supporting me and providing me with all the necessary distractions. Thank you for all the dinners, drinks, activities, and everything else we have done together over the past months. In particular, I want to thank Juul for the journey we had throughout this thesis. It motivated me and brought joy to have a close friend going through the same process and difficulties of writing a thesis. Lastly, I would like to thank my girlfriend Martine for all your love and endless support during these months.

Abstract

This thesis addresses the tactical production-inventory planning problem in serial multi-echelon supply chains with finite production capacity, lead times, and uncertain demand. As this is a complex sequential decision-making problem under uncertainty, Deep Reinforcement Learning (DRL) is proposed as a solution method. The objective of this thesis is to assess the suitability and performance of DRL to solve this problem. A Markov Decision Process is formulated where the multi-dimensional action space is effectively handled by decomposing the decisions into sub-decisions. Furthermore, the model is extended to allow for non-stationary demand. To tackle the problem, the Deep Controlled Learning (DCL) algorithm is applied. A numerical case study is conducted on a variety of supply chain settings to better assess the suitability. Experimental results demonstrate that DRL is a promising general-purpose solution technique to solve the addressed problem. The DCL algorithm obtains (near-)optimal policies in all supply chain settings in which the optimal solution can be computed. Moreover, the DCL algorithm significantly outperforms base-stock policies, specifically designed for this problem, in all considered supply chains with non-stationary demand. Analyzing the neural network policy obtained by the DCL algorithm revealed that this policy yields lower costs and higher service levels, by holding more safety stock compared to the base-stock policies. This study provides evidence that neural networks can effectively be used to represent good policies in inventory problems.

Keywords: Tactical production-inventory planning; Multi-echelon supply chain; Deep reinforcement learning; Non-stationary demand; Markov decision processes

Contents

List of Figures

List of Tables

1 Introduction

A new crisis has emerged in recent years, the global chip shortage. Demand for chips has skyrocketed and the semiconductor industry is not close to keeping up with the supply. With the development of technologies, chips are now found in more products than ever, from cars and computers to refrigerators and smart light bulbs. Moreover, the number of chips in a single product has also increased rapidly: a typical car, for example, has around $50 - 150$ chips, while modern cars have up to 3000 chips. During the pandemic, the demand for chips continued to grow faster than ever, with one of the main reasons being that people who worked from home all needed technology such as a computer and a screen. A comprehensive elaboration of the underlying factors for the semiconductor shortage is given by Voas et al. (2021). Chip shortages are a crisis that has become truly problematic in many industries. CNBC reported in the summer of 2021 that "Chip shortage expected to cost the auto industry $210 billion in revenue in 2021" and Sky News reported recently that "PS5 shortages set to continue due to global chip shortage, warns Sony". The shortage has even become a priority for policymakers worldwide (Voas et al., 2021). In the summer of 2022, president Biden announced the 50 billion fund to boost domestic semiconductor research and development while at the same time countering China (Swanson, 2022). Moreover, the global chip shortage is not expected to be over yet, as CNBC reported in April 2022 "Intel CEO now expects chip shortage to last into 2024". Clearly, it is ever more important for the semiconductor industry and the suppliers of the semiconductor industry to have optimal production planning.

Production planning is one of the most important aspects affecting the performance of a supply chain. Mainly due to demand uncertainty, companies do not know how much inventory to stock to meet customer demand. To hedge against these uncertainties, firms can build up inventory, also referred to as safety stock. One can imagine that a large safety stock leads to unnecessarily high holding costs. On the other hand, a relatively small amount of build-up inventory may result in dissatisfied customers finding the product out-of-stock. Finding inventory management that satisfies a significantly large part of the customers while minimizing the holding costs is a problem that arises in almost all companies and industries. In the high-tech manufacturing industry, production planning is very challenging but even more critical. This industry is characterized by high procurement costs, very long lead times, short product life cycles, and significant demand and supply uncertainty. Because of the long lead times, decisions must be made well in advance while demand and supply are still highly uncertain.

This thesis sets out to optimize tactical production-inventory planning at ASML, the world's leading supplier of lithography systems for the semiconductor industry. The lithography industry is one of the most outstanding examples of the high-tech manufacturing industry. ASML operates in a high-tech low-volume industry with extremely expensive products having very long lead times. Its latest systems cost up to 250 million dollars with supply chain-wide lead times ranging from

a few months to more than a year. ASML faces demand that is low in volume and extremely volatile with a total of 345 machines sold in 2022 (ASML, 2022a). The combination of very high procurement costs, volatile low-volume demand, and long lead times make production planning a highly complex task, yet all the more critical. Imagine the impact of selling one additional system, with the same capacities, as a result of different production planning. On the other hand, imagine the costs associated with unused overcapacity due to non-optimal production planning. In addition to the effect on revenue, production planning becomes even more important when considering the impact of customer service performance on market share potential.

Traditionally, optimization problems in inventory management are solved with exact methods and heuristics. Exact methods provide global optimal inventory policies after evaluating all possible options. Considering all possible options is very time-consuming and computationally inefficient in many real-world settings where there are simply too many possibilities to evaluate them all. Heuristics provide solutions in an acceptable time, but these solutions may not have any theoretical guarantee. To arrive at a (near-)optimal solution, these methods often make high-level assumptions oversimplifying the problem. Much emphasis in the literature has been put on deriving optimization strategies with provable optimality, often at the expense of simplifying reality (Kosasih & Brintrup, 2021). However, companies are typically more interested in models that reflect reality. A 2011 report from Chief Supply Chain Officer revealed that many companies still tend to use rather simplistic models for inventory management (CSCO, 2011). This indicates a great need for models that not only find optimal policies but also reflect reality.

Thanks to the development in technologies, computer equipment, and data science techniques, reinforcement learning (RL) methods are finding ever more success in deriving optimal solutions to complex real-world problems. RL is designed to solve sequential decision-making problems under uncertainty, that is, to obtain optimal behavior in a situation that maximizes reward. The learner is not told what to do but learns which actions yield the most reward by trying them out. This has the advantage that no problem-specific rules need to be designed. While training the agent might be computationally expensive and time-consuming, the agent is then able to make repeated decisions within seconds. Motivated by this success of RL, the method has rapidly gained interest over the last couple of years and has gradually become one of the most active research areas in machine learning, artificial intelligence, and neural network research (Sutton & Barto, 2018). The inclusion of neural networks to approximate functions within RL, known as deep reinforcement learning (DRL), has led to significant performance improvements. DRL has been successfully applied in many different fields, such as gaming (Mnih et al., 2015), robotics (Kober et al., 2013), chemistry (Zhou et al., 2017), and inventory control (Boute et al., 2021). One of the breakthroughs of DRL is AlphaGo, developed by Silver et al. (2016). AlphaGo beat the human Go champion by five games to zero in the game of Go, which is considered the most challenging of the classic

games for artificial intelligence because of the absurdly large number of possible sequences of moves.

An excellent example of sequential decision-making under uncertainty is production planning. At each time step, e.g., day, week, or month, the same decision under supply and demand uncertainty must be made on how much to produce at each production step. DRL methods can be applied to this type of problem, as they might help develop near-optimal policies when other methods fail because DRL is not guided in the solution search. Despite decades of extensive research, the optimal policy for many inventory problems remains unknown. No optimal inventory policies have been found in the literature for multi-echelon inventory problems with finite capacities across the entire supply chain. Moreover, non-stationary demand distributions complicate the problem and the optimal policy also remains unknown for most problems under non-stationary uncertain demand. Therefore, this thesis applies a DRL technique to optimize the tactical production-inventory planning in multi-echelon supply chain settings focused on the situation of ASML. ASML is the centralized decision maker that minimizes the total supply chain-wide costs. Most supply chain networks can be modeled as multi-echelon including ASML's network. It is clear that ASML's supply chain has capacities and so a finite capacity is introduced at each node for the production quantity. Furthermore, lead times are assumed for production at each node. Both stationary and non-stationary uncertain demands are considered with all excess demands backlogged. The main objective of this thesis is to assess the suitability of DRL in these problems and compare the performance against other optimization techniques.

1.1 Introduction to ASML

ASML is the world's leading supplier of lithography systems for the semiconductor industry. It develops and manufactures these extremely complex and advanced systems. Lithography systems can be distinguished into extreme ultraviolet (EUV) and deep ultraviolet (DUV). ASML is the only company in the world that manufactures EUV systems for chip production which is operating at the cutting edge of technology. The company leads global innovation within the chip industry by continuously improving accuracy and speed, where speed can be thought of as wafers processed per hour. Moore's law states that the number of transistors on a chip, and thus the performance of a chip, doubles approximately every 18 to 24 months. To keep up with this law, new techniques and products are constantly being developed. The perfect example of this is the EUV technology in which more than six billion euros in R&D is invested over 17 years (ASML, 2022b).

ASML was founded in 1984 as a joint venture between Philips and Advanced Semiconductor Materials International (ASMI) in a leaky shed next to a Philips office in Eindhoven, the Netherlands. Over the past decades, ASML has grown tremendously with today more than 39.000 employees of 143 different nationalities in more than 60 locations worldwide. Last year, the company reported net sales of 21.2 billion euros and sold 345 systems (ASML, 2022a). Furthermore, R&D costs of

3.25 billion euros in 2022 show the importance of innovation. ASML expects to continue growing and believes it has a chance to reach annual net sales between approximately 30 billion and 40 billion euros by 2025 based on various market scenarios (ASML, 2022a).

The industry in which ASML operates can be characterized as high-tech low-volume. The company works with the newest and most advanced technologies to create the latest generation of lithography systems. Characteristics of the high-tech industry such as high procurement costs, long lead times, complex supply chains, and short product life cycles are all familiar to ASML. The demand ASML faces can be regarded as very low, considering that a total 345 systems are sold in 2022 with sales of only a few systems per quarter for some products. The low demand is highly volatile, mainly because of the direct dependence on investments from chip manufacturers in combination with the well-known bullwhip effect. Semiconductor firms are located upstream in the supply chain of electronic goods. ASML is located even further upstream in this supply chain, making the bullwhip effect even stronger and thus demand extremely volatile.

1.2 Production Planning

Production planning is the planning of the availability of resources and materials needed to transform materials into finished products. These planning decisions include both order size and timing. Depending on the planning level, different decisions, objectives, and uncertainties are considered and thus different methods and models are applied. Production planning can be distinguished into three different levels: strategic, tactical, and operational. At the strategic level, long-term plannings are made in which decisions mainly involve capacity decisions. In strategic-level planning, time is aggregated into quartiles or months, and products and resources are aggregated into groups. Strategic production plans typically cover a horizon of several years. Mid-term plannings are made at the tactical level, where tactical decisions involve the size and timing of material flow through the supply chain. The horizon of tactical production planning is usually several months. At the operational level, short-term plans are created that form a detailed daily schedule. This plan includes detailed planning for every production step with available information on both supply and demand. This thesis focuses on tactical production planning also referred to as tactical production-inventory planning.

Most of the literature on production planning studies high-volume industries, while the low-volume industries are not adequately represented (see e.g. (Stadtler, 2005)). One of the core differences in production planning between high-volume and low-volume industries is the need for discrete integer production quantities. In high-volume industries, continuous production quantities can be rounded without many issues. In contrast, rounding in low-volume industries can cause serious problems. Specifically, rounding up is likely to lead to capacity issues while rounding down is likely to result in significant supply shortages and thus unsatisfied customers. Therefore, models in low-volume

industries must output integer production quantities. This difference between high-volume and low-volume is significant because integer variables make solving mathematical models much more difficult.

The two main mathematical models currently used at ASML to support tactical production-inventory planning are RampFlex and CFO. RampFlex is primarily used to evaluate and compare the outcomes of a proposed buffer policy under multiple demand scenarios. It incorporates capacity constraints but is slow in computation. The general multi-stage stochastic programming model that forms the basis of RampFlex together with the rolling horizon framework are introduced in Fleuren et al. (2022). On the other hand, the CFO algorithm is used to rapidly achieve production planning but it does not efficiently handles the capacity restrictions. This illustrates the need for a decision support tool for tactical production-inventory planning that incorporates the capacities and is not very time-consuming.

1.3 Research Objective

This thesis applies a DRL algorithm to optimize the tactical production-inventory planning in serial multi-echelon supply chain networks to assess the suitability of the algorithm. The decisions of the entire supply chain are based on the customer demand to minimize the total cost of the supply chain, and thus, there is a central decision-maker which is ASML. The demand that ASML faces is low in volume which requires integer production quantities that complicate the problem. The demand is first modeled as stationary demand which is later extended to non-stationary demand. As ASML cannot keep up with the demand, capacities are introduced on the number of products that can be processed at each node in one time step. Moreover, deterministic lead times are assumed for production at each node. All excess demands are backlogged which comes with a costly backlog penalty. Due to the extremely volatile demand, the DRL algorithm applied is the Deep Controlled Learning (DCL) algorithm proposed by van Jaarsveld (2021).

For applying DRL, it is crucial to formulate the complex sequential decision-making problem as a Markov decision process. This includes determining the state vector, action space, reward structure, and transition dynamics. The agent observes the environment, takes an action, receives a random reward, and then the environment randomly transitions to a new state. The state vector represents the environment that includes everything outside the decision-maker. Thus, all the information the agent would need to know to make a decision must be included in the state vector. For the transition dynamics, we are mainly interested in the demand uncertainty. First, stationary uncertain demand is implemented. Later, this will be extended to non-stationary demand, which is more similar to the real world but for which limited research has been done. Multi-echelon inventory problems generally have a large state space. Furthermore, ASML takes multiple actions simultaneously resulting in a multi-dimensional action space. This is a concern since DRL algorithms become less

suitable for inventory problems with a large action space (Boute et al., 2021). Therefore, efficient modeling is required. Multiple supply chains are considered to demonstrate the applicability of the algorithm. For evaluating the performance of the DCL algorithm, several base-stock policies are specifically designed for this problem. Furthermore, under certain conditions, the optimal solution can be computed and is used as a benchmark policy. The policies are mainly evaluated in terms of costs, but also in terms of other key performance indicators. Lastly, the actions prescribed by the algorithm are looked into to try and give an intuition behind them. Namely, neural network policies are extremely difficult to interpret which complicates adopting these in practice.

This thesis sets out to answer the following research question by considering the thereafter formulated research sub-questions:

Research Question

To what extent is deep reinforcement learning suitable to solve the tactical production-inventory problem in serial multi-echelon supply chains with finite production capcity, lead times, and uncertain low-volume demand?

Research Sub-Questions

1. *How should the Markov decision process be defined? That is, defining the state and action space, the environment's dynamics, and the reward structure. In particular, how can demand uncertainty be modeled?*
2. *How can the Markov decision process be efficiently modeled?*
3. *How can the Markov decision process be extended to model the non-stationary demand uncertainty?*
4. *How does the solution of the DRL algorithm compare to other well-known heuristics and, if possible, the optimal solution in terms of expected costs and other key performance indicators?*
5. *Does the DRL algorithm prescribe other actions than the heuristics and if so, in which situations?*

1.4 Outline

In addition to this introductory section, this thesis contains seven other sections. The next section contains a review of the existing literature on inventory optimization in multi-echelon supply chain systems, followed by works that have applied reinforcement learning methods to inventory optimization problems. Section 3 introduces the preliminaries for a detailed description of the essential

concepts. Section 4 elaborates on the problem setting, including the supply chain under consideration and the formulation of the Markov decision process. The general deep reinforcement learning algorithm is presented in Section 5. Section 6 introduces additional methodology for solving the tactical production-inventory planning problem using the DRL algorithm. An extensive numerical study is provided in Section 7 to evaluate the performance of the DRL algorithm in several supply chains against other optimization techniques. Lastly, conclusions and recommendations for future research directions, along with a discussion of the limitations of this study are provided in Section 8.

2 Literature Review

This section provides an overview of the relevant literature. This study aims to solve the production planning in a multi-echelon supply chain and therefore, Section 2.1 discusses multi-echelon inventory optimization. Section 2.2 zooms in on inventory management in multi-echelon supply chain settings that assume finite capacity across the supply chain. Then the literature on inventory problems under non-stationary demand uncertainty is considered in Section 2.3. Next, we discuss works that have applied reinforcement learning in inventory optimization. Lastly, we formulate our intended contribution by reviewing the relevant literature.

2.1 Multi-Echelon Inventory Optimization

Inventory management is concerned with determining the amount of additional inventory that must be held to meet target service levels so that a large part of the customer demand can be met while minimizing the costs. The safety stock is used to hedge against uncertainties. Inventory management is critical to the performance of a supply chain network and is therefore studied extensively. Many real-world supply chain networks can be modeled as multi-echelon systems consisting of multiple stages. Therefore, much research has been devoted to multi-echelon inventory models. Multi-echelon inventory management involves determining how safety stocks can be appropriately allocated across all echelons to achieve target service levels at the lowest inventory holding costs. In reality, there are many uncertainties with customer demand usually the main uncertainty. Stochastic models with uncertain parameters are considered complex models to solve. A comprehensive literature review of multi-echelon inventory optimization under demand uncertainty, classified by model assumptions, research goals, and applied methodologies is provided by (de Kok et al., 2018).

Two main approaches to inventory optimization problems for the stochastic multi-echelon system exist in the literature: the stochastic-service model (SSM) and the guaranteed service model (GSM), which were introduced by Clark and Scarf (1960) and Simpson Jr (1958), respectively. In SSM, each stage in the supply chain determines a base-stock inventory level and meets the demand from downstream stages whenever possible using this base-stock level. Any excess unsatisfied demands are backordered. The literature on SSM is extensive. We refer the reader to Simchi-Levi and Zhao (2012) for a literature review. The SSM approach is generally seen as cumbersome to implement. This is mainly because the actual lead times seen by downstream stages are stochastic due to possible stock-outs in the upstream stages that must be backordered (Graves & Willems, 2003). Stochastic lead times lead to stochastic replenishment time for the stage, which is challenging to characterize and implement. In the GSM approach, each stage sets a committed service time (CST) within which it guarantees to meet each demand. In this method, demand is assumed to be bounded from above. These guaranteed service times ensure that the replenishment time for downstream stages is deterministic. Many models have been proposed in the literature that extends

the GSM to include more realistic characteristics such as non-stationary demand, stochastic lead times, capacity constraints, and continuous review. We refer the reader to Eruguz et al. (2016) for a comprehensive survey of the GSM. The underlying assumptions of these models are not suitable for solving the capacitated complex problems considered in this thesis and therefore, we do not discuss these models further.

For the classical multi-stage serial supply chain systems with linear costs, deterministic lead times, no capacities, and stationary random demand where unsatisfied demands are backlogged, Clark and Scarf (1960) show that an echelon base-stock policy is the optimal policy. The optimal base stock levels can be computed by recursively minimizing N nested convex functions, where N equals the number of nodes. Since the implementation and computation of this can still be difficult, heuristics are proposed in the literature to approximate the base-stock levels. A well-known heuristic to solve the inventory problem in such settings is the Shang and Song heuristic introduced by Shang and Song (2003). The heuristic provides bounds on the optimal base-stock levels and approximates the optimal base-stock level by a simple average of the two bounds. This approximation is proven to perform quite well with a maximum error of less than 1.5% on their test instances compared to the exact solution.

2.2 Capacitated Multi-Echelon Inventory Problem

As discussed in the introduction, the semiconductor industry and the suppliers of the semiconductor industry do not keep up with the demand, which invalidates the assumption of infinite capacity in the above-reviewed models. In our setting, the supply chain is a multi-echelon inventory problem where finite capacity is considered across the entire supply chain. This significantly increases the complexity of the system (see, e.g., Simchi-Levi and Zhao, 2012). de Kok et al. (2018) state: "To date, there are no analytical results that enable calculation of (close-to-)optimal policies under finite capacity across a multi-echelon system". When capacities are introduced on the amount that can be processed in a single period, the result of Clark and Scarf (1960) no longer holds. Consequently, the optimal policy is unknown in these systems. Except for single-stage problems where a base-stock policy remains optimal, which is shown by Federgruen and Zipkin (1986). When assuming a base-stock policy, Glasserman and Tayur (1995) developed a simulation-based optimization algorithm for estimating the gradients of the cost function with respect to policy parameters in a capacitated multi-echelon inventory system. They use the Infinitesimal Perturbation Analysis (IPA) method to estimate these gradients. Since this method is still computationally intensive when a large number of scenarios are considered, Glasserman and Tayur (1996) present a simple approximation for identifying near-optimal base-stock levels. They have simplicity explicitly as one of the objectives of the approximation. The average relative error in 72 test instances, where each instance is a 5-echelon serial system, was 1.9% for their proposed approximation base-stock levels. Janakiraman and Muckstadt (2009) provide the state-of-the-art regarding serial multi-echelon in-

ventory models with finite capacity. They identify the high-dimensional structure of the optimal policy. For practical purposes, this policy is intractable because of its high-dimensional structure.

2.3 Non-Stationary Demand

Most inventory management problems occur in situations where demand is non-stationary and uncertain. Neale and Willems (2009) argue that non-stationary demand is the rule rather than the exception in most industries today. Non-stationary demand means that the demand probability distribution changes over time. This might arise due to many different reasons, e.g., technological innovation, different phases in the product life-cycle, or economic fluctuations. It is inevitable that future demand distribution will come from a distribution that differs from what governs historical demand (Li et al., 2011). As practical demand patterns are typically non-stationary one would expect that a lot of research has been devoted to inventory management in non-stationary demand situations. However, relative to the literature pertaining to stationary demand inventory management, the literature on non-stationary demand is limited. This is mainly due to the lack of theoretical structure or the high computational complexity of non-stationary models. Kurawarwala and Matsuo (1996) state that the unique characteristics of non-stationary demand preclude the use of traditional forecasting methods which are not designed for these environments.

Under non-stationary demand, the result of Clark and Scarf (1960), that is, a fixed echelon base stock level is optimal in all periods, does no longer hold as the demand distribution might change over time. Therefore, many studies propose dynamic base stock levels under non-stationary demand. Dynamics base stock levels are proposed to solve the single-stage probabilistic lot-sizing problem (see, e.g., Bookbinder and Tan (1988), Tarim and Kingsman (2004), and Tarim and Kingsman (2006)). Based on the GSM framework, a periodic review base-stock replenishment policy is proposed under the CST policy where for each period a dynamic base stock level is calculated such that the 100% of the demand can be satisfied within the CST (see, e.g., Graves and Willems (2008), Neale and Willems (2009)).

2.4 Reinforcement Leaning for Inventory Optimization

Reinforcement learning has been successfully applied to solve various problems in inventory management. Many of the works in the literature use tabular-based RL techniques. However, these algorithms are only suitable for limited state spaces and therefore, cannot deal with problems like that addressed in this work. Recently, RL has been combined with deep neural networks to approximate the value and policy functions, resulting in deep reinforcement learning (DRL). Motivated by the great success of DRL, several DRL-based methods are proposed to solve inventory, ordering, and production problems. Yan et al. (2022) provide a review of the development and applications of RL techniques in supply chain management. For a clear overview of RL in inventory management,

we first present works that deal with tabular RL techniques and then works that use DRL.

Giannoccaro and Pontrandolfo (2002) and Pontrandolfo et al. (2002) are among the first to apply RL for inventory optimization. They proposed a semi-Markov average reward technique (SMART). In a three-tier linear supply chain with only one agent per echelon minimizing the costs of the entire supply chain, the integrated inventory policy proposed by the SMART algorithm leads to better results than the periodic up-to-order-level policy (Giannoccaro & Pontrandolfo, 2002). Pontrandolfo et al. (2002) show that the SMART algorithm outperforms a local heuristic and a balanced heuristic in a distributed multi-country production system in the context of global supply chain management. Later, Chaharsooghi et al. (2008) used a tabular Q-learning algorithm to handle an ordering management problem in a four-tier supply chain. They tested their algorithm on the well-known beer game which can be modeled as a serial supply chain where decentralized agents cooperatively attempt to minimize the total supply chain costs. They demonstrated the effectiveness of the Q-learning algorithm in complex scenarios where analytical solutions are not available. In a similar setting, Mortazavi et al. (2015) proposed a simulated-based optimization framework using Q-learning.

More recent studies use DRL to solve similar problems in larger complex settings where tabular methods are computationally inefficient. For a review of DRL for inventory control, we refer the reader to Boute et al. (2021).

Peng et al. (2019) use two Vanilla Policy Gradient methods in a capacitated supply chain setting with one plant warehouse and three retailers where customer demand is stochastic and seasonal. Capacity involves both production and storage for the warehouse and the retailers. The two methods differ in the use of the neural network output. One method directly uses the output as policy and the other uses a new activation function to enforce actions in the feasible region. The two DRL-based methods are shown to achieve a better policy in terms of total rewards incurred than the reorder-point/order-quantity policy, (r, Q)-policy, in all experimented settings.

Oroojlooyjadid et al. (2022) propose a transfer-learning Shaped Reward Deep Q-Network (SRDQN) algorithm to play the well-known beer game. The SRDQN algorithm is based on the classical deep Q-network algorithm proposed by Mnih et al. (2015). One agent is the learning agent and follows the SRDQN algorithm, while the other agents follow a base-stock policy or a more realistic model of ordering behavior and act irrationally (Sterman policy, Sterman (1989)). When the SRDQN agent plays alongside teammates who follow a base-stock policy, the algorithm obtains near-optimal solutions. Moreover, it performs significantly better than a base-stock policy when other agents act more realistically and thus irrationally. In addition to the cost comparison, they also evaluated the inventory levels, order quantities, and order up to levels over time of the three agents.

Gijsbrechts et al. (2021) study the performance of the Asynchronous Advantage Actor Critic (A3C) DRL algorithm on lost sales, dual-sourcing, and multi-echelon inventory models. In the multi-echelon setup, they apply the A3C algorithm on two different scenarios with one warehouse and ten retailers where all locations have limited capacity on both the inventory position and the production rate. They consider stochastic demands and deterministic lead times. The states are the inventory positions and the outstanding orders for the warehouse and the retailers. The action space is two-dimensional, consisting of a state-dependent base-stock level for the warehouse and one for the retailers. They showed that the proposed A3C algorithm outperforms a base-stock policy with constant base-stock levels.

Vanvuchelen et al. (2020) solve the joint replenishment problem of two shippers using the proximal policy optimization (PPO, Schulman et al. (2017)) in sizeable problems where dynamic-programming algorithms suffer. The PPO outperforms the periodic minimum order quantity and the periodic review dynamic order-up-to heuristics. By looking at the inventory levels of the different shippers, they conclude that the PPO algorithm resembles the optimal policy.

Alves and Mateus (2020) use the PPO algorithm in a centralized 4-echelon supply chain with two nodes per echelon and uncertain stochastic demands, deterministic lead times, capacities, and lost sales. Capacities are regarding the local stock for each node, production for the suppliers, and processing capacities for the factories. A state and an action are respectively a 27-dimensional and a 14-dimensional continuous vector whose values are the quantity to be produced at each supplier and the number of materials to be sent by each node to the successor nodes. In minimizing the total cost of the entire supply chain within a planning horizon of 360 time steps, the PPO agent showed great performance and achieved slightly better results than an LP agent. Furthermore, they demonstrate that the PPO agent showed better performance in terms of unmet demands. Later, Alves et al. (2021) applied and compared five state-of-the-art policy gradient algorithms in the same supply chain settings as Alves and Mateus (2020). They conclude that PPO has the best performance of the five and may be a good choice to use in practice for the problem addressed. Recently, Alves and Mateus (2022) extend their previous work of 2020 by taking into account uncertain seasonal demands, stochastic lead times, and processing capacities. The PPO agent performed well in all scenarios with better or roughly the same performance as an LP agent. The greater the uncertainty in customer demand and lead times, the better the performance of the PPO agent compared to the LP agent. Moreover, the number of unmet demands is lower for the PPO agent in both the constant and stochastic lead times settings.

Perez et al. (2021) propose and compare a deterministic linear programming model, a multi-stage stochastic program, and a reinforcement learning model for solving an inventory management prob-

lem in a four-echelon supply chain with $2-3-2-1$ nodes per echelon. Furthermore, they consider stochastic demands, deterministic lead times, and production capacity, but no capacitated stocks. In a planning horizon of 30 periods, the reinforcement learning model shows promise in using RL for supply chain applications. While their work is interesting in the way they compare the three different models, their experimental methodology lacks hyperparameter tuning and is therefore less suitable from an RL perspective.

These papers show the great success of DRL in multiple MDPs. However, in these benchmarks, the impact of any randomness is usually limited. Real-world supply chain and logistics problems are highly stochastic with the result that algorithms like A3C and PPO may not be the best choice (van Jaarsveld, 2021). To overcome these limitations of the algorithms that are commonly used in the literature to learn neural network policies, van Jaarsveld (2021) proposes a deep controlled learning (DCL) algorithm that is based on roll-outs and incorporates model-based variance reduction techniques. This algorithm is shown to be the first generic machine learning algorithm to obtain near-optimal results in terms of costs for the lost-sales problem, being superior to the model-free algorithm A3C proposed by Gijsbrechts et al. (2021) and the best heuristic benchmarks. The ideas of this algorithm are generic, making DCL a suitable candidate to apply to other stochastic dynamic problems in supply chain and logistics.

2.5 Contribution of the Current Study

For our research, we mainly build on the work of (van Jaarsveld, 2021) and apply his algorithm to a serial multi-echelon supply chain with unsatisfied demands backlogged, deterministic production lead times, and capacities at all nodes of the supply chain. This study focuses on the situation at ASML with low-volume but all the more stochastic demand making this algorithm the obvious choice for solving the tactical production-inventory planning problem at ASML.

Multi-echelon inventory optimization problems have been studied extensively for more than 60 years. Over the years, researchers have proposed a wide range of models, broadening the field of applications, allowing for many different assumptions, and expanding the scope. Despite decades of research, the optimal inventory policy remains unknown to many inventory problems with more realistic supply chain characteristics such as finite capacities and non-stationary demands. Therefore, this thesis applies a newly emerging method from the field of Artificial Intelligence (AI), DRL, to solve the tactical production-inventory planning problem in serial multi-echelon supply chain networks that include these characteristics. This model can support decision-making and help develop near-optimal policies.

The application of DRL methods for inventory optimization has received increasing research interest in recent years. However, this remains a relatively unexplored direction within both DRL and

inventory management. Results are promising, as several studies have shown great performance in finding near-optimal inventory policies using DRL methods. Studies that assume coordinated decisions optimizing the entire supply chain network typically investigate relatively small networks. To our best knowledge, applying DRL techniques for coordinated decision optimization in multi-echelon supply chain networks with more than two echelons has only been studied by Alves and Mateus (2020), Alves and Mateus (2022), and Perez et al. (2021). Therefore, our work is among the four studies in this area. Moreover, we extend the model to incorporate non-stationary uncertain demand. This is a limited research area of inventory management, and to our best knowledge, our work is the first to apply a DRL method for an inventory problem under non-stationary demand.

In summary, our contribution is the following:

1. Effectively applying the DCL algorithm to optimize the tactical production-inventory planning problem in serial multi-echelon supply chain networks with finite production capacity, lead time, and uncertain demand.
2. Extending this to solve the tactical production-inventory planning problem under non-stationary uncertain demand.

3 Preliminaries

This section introduces the essential concepts so that the reader can fully understand this thesis. First, the concepts of RL are explained in Section 3.1. To solve the problem using RL techniques, the sequential decision-making problem is modeled as a Markov Decision Process. Markov Decision Processes are introduced in Section 3.2. The DCL algorithm builds on the well-known method of policy improvement. Policies and policy improvement are discussed in Sections 3.3 and 3.4, respectively. Neural networks are used to represent policies and therefore, neural networks are explained in Section 3.5, and their use to represent policies is discussed in Section 3.6. All introductions can be safely skipped by readers familiar with the topic.

3.1 Reinforcement Learning; the core-idea

RL is a sub-area of machine learning, in which a decision maker, also referred to as an agent, interacts with its environment to learn which actions yield the highest cumulative reward within the environment. The environment comprises everything outside the agent. At each discrete time step, the agent observes the current representation of the environment, chooses an action, receives a random reward, and then the environment randomly transitions to a new state. This framework is mathematically known as a Markov Decision Process which is illustrated in Figure 3.1. With the obtained results from trying different actions in states, the agent learns which action yields the highest reward in the states. When repeating this process many times, the agent has learned which actions to take in which states. This is the learned policy that prescribes in each state an action. The goal of RL can be formulated as maximizing the expected discounted sum of rewards over an infinite horizon, that is, finding a policy that maximizes the expected discounted sum of rewards. To solve the large complex problems considered in this thesis, policies can not be represented by tables because there are too many options. Therefore, policies are represented as neural networks. A common method to find the optimal policy is policy improvement and iteration.

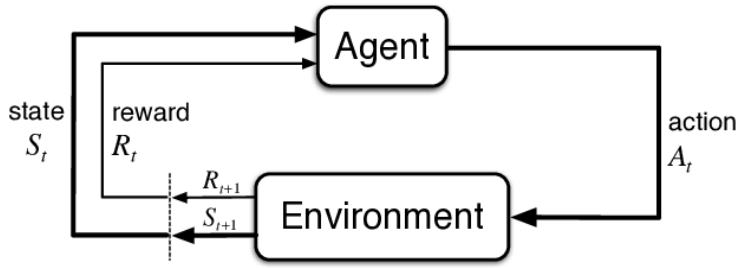


Figure 3.1: The agent–environment interaction in a Markov decision process (Sutton & Barto, 2018)

To find actions that maximize the expected discounted sum of rewards, the trade-off between exploration and exploitation is utterly important. Agents must exploit actions that they have already

experienced to get a reliable estimate of the expected reward, but must also explore new actions that may yield better rewards. Too much focus on exploitation may cause agents to get stuck in a local optimum where good but not optimal actions are chosen. On the other hand, too much focus on exploration leads to both "good" and "bad" actions being continuously explored, resulting in a much longer time to converge to optimality. The ϵ-Greedy method is a common method for balancing exploration and exploitation. In this method, the agent takes a random action with probability ϵ (exploration) and takes an action based on previous choices with probability $1 - \epsilon$ (exploitation). The choice of ϵ is one of the parameters of the algorithm.

RL differs from *supervised learning* which learns from a training data set that specifies the correct action in a given situation. Furthermore, RL differs from *unsupervised learning*, this type of learning attempts to find a structure hidden in unlabeled data.

3.2 Markov Decision Process

The algorithm proposed in this thesis makes extensive use of a Markov Decision Process (MDP) (cf. Puterman, 2014) with a finite action space $\mathcal{A} = \{1, \ldots, m\}$ and a discrete state space $\mathcal{S}$, which can be finite or infinite. While observing the systems state $\mathbf{s} \in \mathcal{S}$, the agent chooses an action a from the set of allowable actions in state $\mathbf{s}$, $\mathcal{A}_\mathbf{s}$. As a result of this action, the agent receives a random reward $R(\mathbf{s}, a)$, where $r(\mathbf{s}, a) = \mathbb{E}[R(\mathbf{s}, a)]$ is the corresponding expected reward of taking action a in state $\mathbf{s}$. The agent's goal is to maximize the expected cumulative discounted reward over an infinite horizon. After taking action a in state $\mathbf{s}$ the agent finds itself in a new state $\mathbf{s'}$. The transition to state $\mathbf{s'}$ from state $\mathbf{s}$ upon taking action a occurs with a random transition probability, denoted by $\mathbf{P}(\mathbf{s'}|\mathbf{s}, a)$. This random transition is influenced by some random exogenous information that arrives after taking an action. Typically, in inventory management, the reward is defined as costs and the exogenous information is the demand.

3.3 Policy

A policy is a solution to the Markov Decision Process. It is a mapping from states to actions, which can be probabilistic. The goal of a Markov Decision Process is to find the policy that maximizes the expected reward. The probability of selecting a possible action $a \in \mathcal{A}_\mathbf{s}$ when being in state $\mathbf{s} \in \mathcal{S}$ following policy π is equal to $\pi(a|\mathbf{s})$. Starting in state $\mathbf{s}$ following policy π results in a random trajectory which is a sequence of states and actions visited over time. The discounted sum of rewards ($G_\pi(\mathbf{s})$) obtained for such a trajectory is

$$G_\pi(\mathbf{s}) = \sum_{t=0}^{\infty} \gamma^t r_t \text{ with } \mathbf{s}_0 = \mathbf{s}, a_0 = \pi(\cdot|\mathbf{s}_0), r_t \sim R(\mathbf{s}_t, a_t), \mathbf{s}_{t+1} \sim \mathbf{P}(\cdot|\mathbf{s}_t, a_t),\ a_{t+1} \sim \pi(\cdot|\mathbf{s}_{t+1})\ \forall t \tag{1}$$

with $0 < \gamma < 1$ as discount factor. The value function of a state $\mathbf{s}$ for policy π is the expected sum of discounted rewards incurred over an infinite time horizon when starting in $\mathbf{s}$ and following π thereafter. This is denoted by $v_\pi(\mathbf{s})$. These *state-value functions* satisfy recursive relationships with the value functions of the possible successor states $\mathbf{s}'$:

$$v_\pi(\mathbf{s}) = \mathbb{E}[G_\pi(\mathbf{s})|\mathbf{s}] = \sum_a \pi(a|\mathbf{s}) \left(r(\mathbf{s}, a) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}'|\mathbf{s}, a) v_\pi(\mathbf{s}') \right), \ \forall \mathbf{s} \in \mathcal{S} \tag{2}$$

Similarly, the value of taking action a in state $\mathbf{s}$ under policy π, denoted by $q_\pi(\mathbf{s}, a)$, is the expected cumulative discounted reward when starting in $\mathbf{s}$, taking action a, and following π thereafter. These functions are defined as *action-value functions*:

$$q_\pi(\mathbf{s}, a) = \mathbb{E}[G_\pi|\mathbf{s}, a] = r(\mathbf{s}, a) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}'|\mathbf{s}, a) v_\pi(\mathbf{s}'), \ \forall \mathbf{s} \in \mathcal{S}, \ \forall a \in \mathcal{A} \tag{3}$$

The state-value and action-value functions only differ in the first action. The optimal reward-maximizing state-value functions $v^*(\mathbf{s})$ and action-value functions $q^*(\mathbf{s}, a)$ can be obtained by solving the well-known *Bellman equations* (Bellman, 1954):

$$v^*(\mathbf{s}) = \max_{a \in \mathcal{A}_s} \left\{ r(\mathbf{s}, a) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}'|\mathbf{s}, a) v^*(\mathbf{s}') \right\} \ \forall \mathbf{s} \in \mathcal{S} \tag{4}$$

$$q^*(\mathbf{s}, a) = \max_{a'} \left\{ r(\mathbf{s}, a) + \gamma \sum_{s'} \mathbf{P}(\mathbf{s}'|\mathbf{s}, a) q^*(\mathbf{s}', a') \right\} \ \forall \mathbf{s} \in \mathcal{S}, \forall a \in \mathcal{A} \tag{5}$$

The optimal policy follows directly from $v^*(\mathbf{s})$ and $q^*(\mathbf{s}, a)$. Explicitly solving the Bellman equations provides one route to finding the optimal policy. However, in practice one is generally not able to implement this solution exactly mainly because the environment is rarely exactly known or it is computationally intractable due to large state, action, or outcome spaces. Hence, the proposed algorithm approximately solves the Bellman equations by policy improvement and iteration.

3.4 Policy Improvement and Iteration

To improve any (initial) policy, the algorithm used in this thesis makes use of policy improvement steps. By iterating the policy improvement algorithm, an optimal policy is found under certain assumptions (Puterman, 2014). To improve a policy π that prescribes action a in state $\mathbf{s}$, one would want to know whether the policy needs to be changed to take a different action $a' \neq \pi(\mathbf{s})$ in state $\mathbf{s}$ and thereafter, follow π. Therefore, the rewards received over a trajectory starting in state $\mathbf{s}$ taking action $a' \neq \pi(\mathbf{s})$ and following π after this initial action is considered. The expected reward of this trajectory equals $q_\pi(\mathbf{s}, a')$. The new policy π', which takes action a' and then follows π, is as good as, or better than π when $q_\pi(\mathbf{s}, a') \geq v_\pi(\mathbf{s})$. Extend this to all states and select for each

state the action that appears best according to state-action functions. Define the improved policy $\pi'(\cdot)$ by $\pi'(\mathbf{s}) = \arg\max_{a \in \mathcal{A}_s} q_\pi(\mathbf{s}, a)$. The algorithm proposed in this paper enables approximate policy improvement without solving (2) (van Jaarsveld, 2021). These approximations rely on neural networks, which are discussed next.

3.5 Neural Networks

In DRL, neural networks are used to represent policies for Markov decision processes. Neural networks can be seen as parameterized functions that map an input vector $\mathbf{x} \in \mathbb{R}^u$ to an output vector $\mathbf{y} \in \mathbb{R}^v$, for some $u, v \in \mathbb{N}$. Denote the parameterized function by $N_\theta(\cdot)$ with θ as neural network parameters. The elements of the input vector are also referred to as features. For each $\mathbf{x} \in \mathbb{R}^u$ and set of parameters θ, $\mathbf{y} = N_\theta(\mathbf{x}) \in \mathbf{R}^v$. The algorithms in this paper utilize the multi-layer perceptron (MLP) (Feed-Forward Neural Networks, Kroese et al., 2019). The MLP consists of multiple (hidden) layers including an input and output layer where each layer consists of a certain number of nodes. Each layer combines an affine transformation of the variables in the preceding layer with a non-linear activation function.

A neural network with $L + 1$ layers has the input layer ($l = 0$), the output layer ($l = L$), and the hidden layers ($l = \{1, \ldots, L-1\}$ where each layer l has p_l number of nodes and parameters given by some weight matrix W_l and a bias vector $\mathbf{b}_l$. Thus, the parameters of the entire neural network are $\theta = \{W_1, \mathbf{b}_1, \ldots, W_L, \mathbf{b}_L\}$. Let $\mathbf{x}_l \in \mathbb{R}^{p_l}$ be the variables of layer l. Furthermore, let $f_l(\cdot) : \mathbb{R}^{p_l} \to \mathbb{R}^{p_l}$ be the activation function which is $f_l(\cdot) = max(\mathbf{x}_l, 0)$ for $l < L$ and $f_L(\mathbf{x}) = \mathbf{x}$. Here, the maximum is taken element-wise. Layer operations in the neural network relate these variables by the following formula:

$$\mathbf{x}_l = f_l(W_l \mathbf{x}_{l-1} + \mathbf{b}_l) \qquad \forall l \in \{1, \ldots, L\} \tag{6}$$

The number of columns in W_l equals the number of rows in W_{l-1} and the dimension of $\mathbf{b}_l$ equals p_l for all $l \in \{2, \ldots, L\}$ to ensure (6). The MLP sets $\mathbf{y} = N_\theta(\mathbf{x}) := \mathbf{x}_L$, where $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{x}_l$ follows (6) for $l \in \{1, \ldots, L\}$. The number of hidden layers and the number of nodes per layer are hyperparameters of the algorithm.

3.6 Neural Network Policies

A state $\mathbf{s}$ can be represented by a vector in $\mathbb{R}^N$ for some $N \in \mathbb{N}$ and the actions space is $\mathcal{A} = \{1, \ldots, M\}$ which can be seen as a vector in $\mathbb{R}^M$. Hence, any neural network $N_\theta(\cdot) : \mathbb{R}^N \to \mathbb{R}^M$ with as input the state vector and as output the action-value functions can be used as a policy. This is depicted in Figure 3.2 which also clearly visualizes the use of the hidden layer and the neural network parameters.

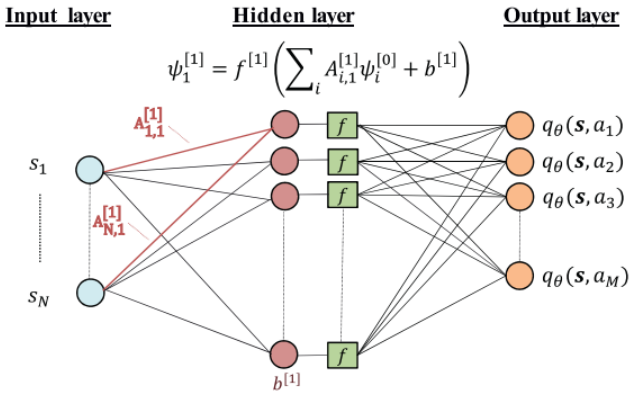


Figure 3.2: DRL makes use of neural networks to represent the action-value functions (Boute et al., 2021)

A policy based on the neural network prescribes in state $\mathbf{s}$ the allowable action $a \in \mathcal{A}_s$ which corresponds to the highest neural network output $N_\theta(\cdot)_a$. $\mathbf{x}_a$ denotes the ath element of the vector $\mathbf{x}$. The policy is based on the neural network parameters θ and is therefore denoted as $\pi_\theta(\cdot)$:

$$\pi_\theta(\mathbf{s}) = \arg\max_{a \in \mathcal{A}_s}[(N_\theta(\mathbf{s}))_a] \tag{7}$$

This policy is a deterministic policy since it prescribes a single action. This differs from a stochastic policy which prescribes for each state a probability distribution over all actions. In stochastic policies, the action with the highest neural network output receives the highest probability of being chosen while the action with the lowest neural network output gets the lowest probability. Model-free algorithms like A3C (e.g. Gijsbrechts et al. (2021)) or PPO (e.g. Vanvuchelen et al. (2020)) optimize stochastic policies. The existence of optimal deterministic policies for MDPs (see e.g. Puterman (2014)) has been the motivation for using deterministic instead of stochastic policies. The DRL algorithm aims to find parameters θ for a neural network with a given structure such that the neural network policy is a well-performing policy.

4 The Problem

This section presents the problem. Section 4.1 presents the supply chain characteristics. To make a production planning for the supply chain addressed using DRL, the problem is formulated as a Markov Decision Process in Section 4.2.

4.1 The Supply Chain

This section formalizes the supply chain networks under consideration. This thesis optimizes tactical production-inventory planning problems in multi-echelon serial supply chain settings, focusing on the situation at ASML. ASML is the central decision-maker for the entire supply chain and makes decisions based on customer demand. The goal is to minimize supply chain-wide costs where a unit cost penalty is incurred for each unsatisfied demand. At each time step, decisions are made on how much to produce in each node. The transport of materials from a node to its successor is assumed to occur with a lead time of zero when the agent decides to produce in the successor node. Furthermore, local inventory levels follow directly from the flow of materials in the supply chain.

Several multi-echelon serial supply chain networks are considered in this thesis. Figure 4.1 shows a 4-echelon supply chain for visualization. Suppliers supply components that are processed into other components by the nodes that eventually become finished products in the most downstream node of the supply chain network. Here, infinite supply availability is assumed and the most downstream node (node 1) faces stochastic customer demand. This demand is an integer in value due to the low-volume industry. Each node can hold local stock to hedge against uncertain future customer demand. This is also referred to as on-hand inventory. ASML has a production capacity at each node in the supply chain network that limits the number of components that can start to be produced at the node in one time period. Furthermore, each node has a deterministic lead time for the production of the component. The objective is to operate the entire supply chain in such a way that customer demand is met at minimum total costs. These costs are related to holding and backlog costs. Specifically, holding costs arise when the component is processed by a node but must wait for production at the successor's node or when finished end-products are held in stock to meet future demand. Holding costs are equal to the sum of on-hand inventory multiplied by the unit holding cost in that node. In addition, backlog costs arise when demands can not be met immediately from the finished products in stock, assuming complete backorder. There is a unit penalty cost associated with each period during which demand is not met.

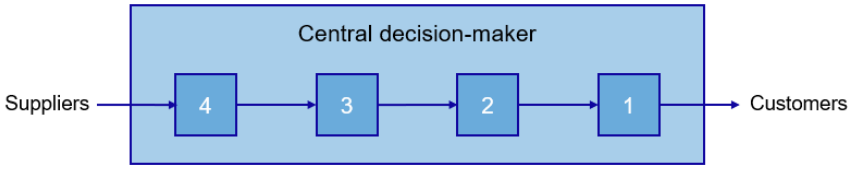


Figure 4.1: A 4-echelon serial supply chain with central decision-maker

Next, the dynamics of the supply chain are stated. At the beginning of the planning horizon ($t = 0$), there is an initial amount of inventory in stock and production for each node. Steps 1 through 4 all happen at the start of the time period. The following steps are repeated at each time step until the end of the planning horizon:

1. Material inflow takes place. That is, products whose production lead time has expired become available and are stored as inventory in stock. All other inventories in production have one less period to become available as stock.

2. Material outflow takes place. Uncertain customer demands are realized and met from the finished end-products in stock, i.e., the inventory in stock at the most downstream node. All excess demands are backordered.

3. The agent decides the production quantity at each node. This quantity is produced and removed from the inventory in stock of the predecessor's node.

4. Costs are incurred. The costs are the holding costs and any backlog costs.

5. A new time step is considered, $t = t + 1$.

4.2 MDP Formulation

One of the main tasks to solve the problem using RL techniques is to model the complex sequential decision-making problem as a Markov decision process. An MDP is formulated by defining the states, actions, rewards, and transition dynamics which are all presented in this section.

Production planning is a sequential decision-making problem because at each time step a decision must be made on the production quantity at each node. The planning follows a policy that prescribes production quantities at each production step while observing the current environment. To optimize production planning, one looks at the planning that minimizes the total expected discounted costs over the entire planning horizon, which might be infinite, while meeting customer demands:

$$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t C(\mathbf{s}_t, \mathbf{X}_t^{\pi}) | \mathbf{s}_0\Big] \tag{8}$$

Before defining the MDP, the sets, parameters, and variables needed to model the MDP are introduced.

Sets

$\mathcal{T}$: time slots, with index t ($\mathcal{T} = \{0, 1, \ldots, T\}$)
$\mathcal{N}$: nodes, with index n
$\mathcal{L}_n$: time slots for inventory in production at node n, with index i ($\mathcal{L}_n = \{1, \ldots, l_n - 1\}$)

Parameters

$l_n \in \mathbb{N}_0$: production lead time at node n
$p \in \mathbb{R}_+$: unit backlog penalty
$h_n \in \mathbb{R}_+$: unit inventory holding cost at node n
$c \in \mathbb{N}_0$: production capacity
$d_t \in \mathbb{N}_0$: demand for final product in time t
$T \in \mathbb{N}_0$: length of the planning horizon, which can be infinite

Variables

$X_{t,n} \in \mathbb{N}_0$: production quantity at node n in time t
$IL_{t,n} \in \mathbb{N}_0$: inventory in stock at node $n \in \mathcal{N} \setminus \{1\}$ in time t
$IL_{t,1} \in \mathbb{Z}$: inventory in stock at node 1 in time t, where negative inventory is backlog
$P_{t,n,i} \in \mathbb{N}_0$: inventory in production at node n in time t that becomes available as stock in i time periods

4.2.1 State Space

The state includes everything from the environment. In our model, this is a multi-dimensional vector consisting of the inventory in stock and production of each node. The agent is a central decision maker who observes the entire supply chain and therefore, observes this inventory vector for each node. Due to the production lead times, products that are decided to be produced in this time step are not available immediately, but only after the lead time amount of time. The agent knows exactly when the products become available as stock since the production lead times are deterministic.

$$\mathbf{s}_t = [IL_{t,n}, P_{t,n,1}, \ldots, P_{t,n,l_n-1}]_{\forall n \in \mathcal{N}} \tag{9}$$

Demand is not included in the state vector in models where the customer demand is stationary and independent and identically distributed. Here, the current observed demand does not provide information about future demand. Therefore, the current demand does not determine the state of the process. On the other hand, when the current demand provides information about future

demands, it must be included in the state vector.

4.2.2 Action Space

The agent is a centralized decision-maker, deciding how much to produce at each node while observing the state vector. In a supply chain with N-echelons, the action space is an N-dimensional vector in which each element refers to a production quantity at a node in that period. Production quantities must be integers because of the low-volume industry. Moreover, they are constrained by the production capacity.

For production at a node, the agent can only decide to produce if the materials are available. The available materials are the inventory in stock at the predecessor's node. As mentioned earlier, an infinite availability of supply is assumed for the most upstream node. After observing the current state, a set of all possible production quantities is defined. This set is the set of allowable actions $\mathcal{A}_{\mathbf{s}}$ described in the MDP explanation in Section 3.2.

$$
\begin{aligned}
& [\mathbf{X}_t] = [X_{t,n}]_{\forall n \in \mathcal{N}} \\
\text{s.t.} \quad & X_{t,n} \leq c && \forall n \in \mathcal{N} && \text{(10a)} \\
& X_{t,n} \leq IL_{t,n+1,0} && \forall n \in \mathcal{N} \setminus \{N\} && \text{(10b)} \\
& X_{t,n} \in \mathbb{N}_0 && \forall n \in \mathcal{N} && \text{(10c)}
\end{aligned}
$$

Constraints (10a) ensure that the production capacity is not exceeded, whereas constraints (10b) ensure that the materials are available. Constraints (10c) pose integrality requirements on the production quantities.

4.2.3 Reward Function

Taking an action in a state produces a random reward through which the agent learns the best action. Thus, a crucial part of solving the problem using RL methods is the design of the reward function. Since the supply chain problem is a cost minimization problem, it seems logical to set the reward equal to the negative value of the total cost. Thus, the reward is the negative value of the sum of the holding and backlog costs at that time step. As mentioned earlier, backlog cost is demand backlogged multiplied by the unit backlog penalty. Holding costs are the sum of the holding costs per node, that is, the inventory in stock in that node multiplied by the unit holding costs for that node. The costs are incurred after ASML decides on the production quantities for this period. Thus, no holding costs are incurred during this period for products that are put into production. Furthermore, the demand is already subtracted from the inventory in stock at the most

downstream node before the cost is computed. This results in the following cost function:

$$C(\mathbf{s}_t, \mathbf{X}_t) = \sum_{n \in \mathcal{N} \setminus \{1\}} h_n(IL_{t,n} - X_{t,n-1}) + h_1(IL_{t,1})^+ - p(IL_{t,1})^- \tag{11}$$

4.2.4 Transition Dynamics

Taking an action while being in a state with some realized demand results in a transition to a new state. Therefore, the random transition to a new state depends on the current state, the actions taken, and the random demand realized. The new state is considered to be the state after the material outflow step in the dynamics of the supply chain. Denote the function of this random transition $\mathbf{s}_t$ by $f(\mathbf{s}_{t-1}, \mathbf{X}_{t-1}, d_t)$, $f : \mathcal{S} \times \mathcal{A} \times D \to \mathcal{S}$. The transition dynamics are as follows:

$$\begin{aligned}
& [\mathbf{s}_t] = f(\mathbf{s}_{t-1}, \mathbf{X}_{t-1}, d_t) \\
\text{s.t.} \quad & P_{t,n,i} = P_{t-1,n,i+1} && \forall i \in \mathcal{L}_n \setminus \{l_n - 1\}, \forall n \in \mathcal{N} && \text{(12a)} \\
& P_{t,n,l_n-1} = X_{t-1,n} && \forall n \in \mathcal{N} && \text{(12b)} \\
& IL_{t,n} = IL_{t-1,n} + P_{t-1,n,1} - X_{t-1,n-1} && \forall n \in \mathcal{N} \setminus \{1\} && \text{(12c)} \\
& IL_{t,1} = IL_{t-1,1} + P_{t-1,1,1} - d_t && && \text{(12d)}
\end{aligned}$$

Constraints (12a) ensure that all inventories in production require one period less to become available as stock. Constraints (12b) put the decided production quantities at each node in the previous time step in production for the upcoming production lead time minus one periods. Constraints (12c) are the recursive equations for the inventory in stock levels in each node that does not meet customer demand, while constraint (12d) is the recursive equation for the inventory in stock level in the most downstream node. Negative inventory in stock at the most downstream node is the backlog in this period.

5 The Deep Controlled Learning Algorithm

This section presents the DRL algorithm used to solve the tactical production-inventory planning problem for the considered supply chain networks and MDP formulations. The DRL algorithm used is the Deep Controlled Learning (DCL) algorithm proposed by van Jaarsveld (2021). Section 5.1 gives an overview of the algorithm and the subsequent sections detail the steps of the algorithm. Section 5.2 describes the subset of states selection. Section 5.3 explains the approach for obtaining the action that yields the lowest simulation-based costs while being a state and following the policy. Lastly, Section 5.4 discusses the training of the neural network. For a more detailed explanation of the algorithm, we refer the reader to van Jaarsveld (2021).

5.1 Overview of Approximate Policy Improvement Steps

The DCL algorithm makes *approximate* policy improvement steps, each of which involves both Monte Carlo simulation and the training of a neural network (van Jaarsveld, 2021). In most multi-echelon supply chains, the MDP introduced in the previous section cannot be solved numerically due to the large state or action space. Therefore, exact policy improvement is intractable and DRL methods are used to make approximate policy improvement steps where neural networks represent the policy. The DCL algorithm obtains policy π_{i+1} from policy π_i using the following approximate policy improvement steps proposed by van Jaarsveld (2021):

1. Select a subset of states $\{\mathbf{s}_k | k \in \{1, \ldots, K\}\}$ (Section 5.2).
2. Use the simulation-based policy $\hat{\pi}_i^+(\cdot)$ to obtain $\mathcal{K}_i = \{(\mathbf{s}_k, \hat{\pi}_i^+(\mathbf{s}_k)) | k \in \{1, \ldots, K\}\}$ (Section 5.3).
3. Obtain the neural network parameters $\theta(\mathcal{K}_i)$ by training the neural network on the data $\mathcal{K}_i$ through supervised learning (Section 5.4).
4. Set $\pi_{i+1} = \pi_{\theta(\mathcal{K}_i)}$

The DCL algorithm obtains $\pi_1, \pi_2, \pi_3, \ldots$ by repeatedly doing the abovementioned steps starting from an initial policy π_0. It is well known that the exact policy iteration typically converges to near-optimal solutions in a few iterations. Therefore, the policy obtained after a few approximate policy improvements can already yield near-optimal performance. As a neural network is used to represent the policy, the objective of the algorithm is to find the parameters $\theta(\mathcal{K}_i)$ such that the neural network policy is a well-performing policy. A neural network is trained and a neural network policy is obtained in each iteration of the policy improvement step. The following sections elaborate on the steps of the approximate policy improvement algorithm.

5.2 Subset of States Selection

To obtain the policy π_{i+1} from policy π_i, a subset of states are selected for which by simulations the best actions is determined. Then, this is added to a dataset containing the states with the corresponding simulations-based prescribed actions. This happens in Step 2 of the algorithm and is discussed in Section 5.3.

For the selection of states, it is assumed that some initial state $\mathbf{s} \in \mathcal{S}$, which might be random, is available. The states selected and added to the dataset are the states that the agent encounters when taking the prescribed simulation-based action with probability $1 - \beta$, and a random action with probability β in each time step. β a is parameter of the algorithm, where $\beta \in [0, 1]$. This method is the same as the ϵ-Greedy method explained in Section 3.1 with β replaced by ϵ.

5.3 Simulation-Based Policy

This section presents the method for estimating the action that yields the lowest expected discounted costs for each state. This involves assessing whether action a or a' is preferred, that is, whether $q_\pi(\mathbf{s}, a) - q_\pi(\mathbf{s}, a')$ is greater than or less than zero. The state-action values for all allowed actions are estimated by simulating many trajectories where only the first action is changed and the policy is followed thereafter. The simulation-based prescribed action is the allowed action that yields the lowest simulation-based discounted costs in that state. This gives the simulation-based policy for that state $\hat{\pi}^+(\mathbf{s}) = \arg\min_a q_\pi(\mathbf{s}, a)$.

The remainder of this section first discusses the method to determine the simulation-based costs, followed by the approach to find the simulation-based policy. In this section, the state $\mathbf{s}$ and the policy π are fixed.

5.3.1 Simulation-Based Costs

The state-action value of taking action a in state $\mathbf{s}$ is estimated by simulating the cost accumulated over an infinite horizon, when starting in state $\mathbf{s}$, taking action a, and following π thereafter. Recall that this is $Q_\pi(\mathbf{s}, a)$ with $q_\pi(\mathbf{s}, a)$ as the corresponding expectation, $\mathbb{E}[Q_\pi(\mathbf{s}, a)]$. This can be calculated using the following equation:

$$\mathbb{E}\sum_{t=0}^{\infty} \gamma^t c_t = \mathbb{E}\sum_{t=0}^{T} c_t \tag{13}$$

where T is geometrically distributed with the discount parameter γ ($T \sim Geo(\gamma)$). This implies that the expected sum of discounted costs over an infinite time horizon is equal to the expected sum of costs obtained in a trajectory of length T where $T \sim Geo(\gamma)$. Thus, unbiased estimators of

$q_\pi(\mathbf{s}, a)$ are obtained by summing the total costs over a trajectory of T periods, starting from state $\mathbf{s}$, taking action a, and then following π.

Deep *controlled* learning departs from the assumption that the randomness in transitions and costs are caused by uncertain factors or events that influence the trajectory, and that exist independently of the trajectory (van Jaarsveld, 2021). These uncertain factors are contained in a composite random variable ξ. In the model addressed in this thesis, these factors are the per-period demand D and the trajectory length T. Hence, $\xi = (T(\xi), \forall t \in \{1, \ldots, T(\xi)\} : D_t(\xi))$, where, for every period, $D_t(\xi)$ is an independent replication of D. Taking action a in state $\mathbf{s}$ with some realized demand $d \sim D$ yields random costs and results in a transition to a new random state **s'**. There exist functions for the random costs and the random transition to a new state. In the MDP formulation of the problem addressed, these are the reward function (11) and the transition dynamics (12), respectively. In general, denote the random costs function $g(\mathbf{s}, a, d)$, $g : \mathcal{S} \times \mathcal{A} \times D \to \mathbb{R}$ and the random transition to a new state function $f(\mathbf{s}, a, d)$, $f : \mathcal{S} \times \mathcal{A} \times D \to \mathcal{S}$. The accumulated discounted costs over an infinite horizon, when starting in state $\mathbf{s} = \mathbf{s}_1$, taking action a_1, and following π thereafter, can be formulated as follows:

$$
\begin{aligned}
&Q_\pi(\mathbf{s}_1, a_1|\xi) := \sum_{t=1}^{T(\xi)} c_t \\
&\forall t \in \{1, \ldots, T(\xi)\} : c_t = g(\mathbf{s}_t, a_t, D_t(\xi)), \mathbf{s}_{t+1} = f(\mathbf{s}_t, a_t, D_t(\xi)), a_{t+1} = \pi(\mathbf{s}_{t+1}). \qquad (14)
\end{aligned}
$$

By (13) - (14), $\mathbb{E}[Q_\pi(\mathbf{s}, a|\xi)] = q_\pi(\mathbf{s}, a)$ which is in line with the formulas of the MDP.

5.3.2 Simulation-Based Policy Approach

The algorithm determines the action that yields the lowest cost in state $\mathbf{s}$ using estimates of the costs based on replications of (14). Many replications are required to determine with reasonable accuracy whether action a or a' is preferred in state $\mathbf{s}$. Therefore, the algorithm uses the *common random number* approach, that is, the uncertainty is fixed while comparing the costs of different actions.

The algorithm draws random independent samples ξ_i and, from each sample, estimates of $q_\pi(\mathbf{s}, a)$ and $q_\pi(\mathbf{s}, a)$–$q_\pi(\mathbf{s}, a')$ are obtained using (14). The latter estimate uses variance control and thus, a covariance term appears in the variance of this estimate. This covariance term is typically positive in inventory problems. Namely, a sequence with consecutively high demands will lead to stockouts with high backlog costs, regardless of the initial action being a or a'. This positive covariance term may significantly reduce the number of replications r required to have an acceptable variance of the estimator of $q_\pi(\mathbf{s}, a)$–$q_\pi(\mathbf{s}, a')$.

To further reduce the number of replications, computational resources are allocated so that minimum resources are wasted for actions for which it is already clear that they do not yield the lowest costs. After estimating $Q_\pi(\mathbf{s}, a|\xi_i)$ for all allowed actions $a \in \mathcal{A}_\mathbf{s}$ for several samples ξ_i, it may already be clear that some actions will not yield the lowest costs, even if it is still unclear which action does. Then, for these actions, the algorithm does not compute $Q_\pi(\mathbf{s}, a|\xi_i)$ for new samples of ξ_i. Each sample is referred to as a roll-out and this procedure is governed by the number of initial roll-outs $\underline{n}$ and the maximum number of roll-outs $\bar{n}$ with $1 < \underline{n} \leq \bar{n}$. For the initial roll-outs, $Q_\pi(\mathbf{s}, a|\xi_i)$ is calculated for each allowed action. Thereafter, $Q_\pi(\mathbf{s}, a|\xi_i)$ is computed only for actions that are not dominated by other actions using a simple criterion with $1 - \epsilon$ confidence. This approach terminates when only one action remains that dominates all other actions, or when the maximum number of roll-outs is reached. This action is the prescribed simulation-based action in state $\mathbf{s}$ which together are included in the dataset $\mathcal{K}_i$.

5.4 Training the Neural Network

A neural network with a given structure is trained in each approximate policy improvement step. The structure of the neural network consists of the number of hidden layers and the number of nodes per layer, which are hyperparameters of the algorithm. The algorithm obtains the neural network parameters $\theta(\mathcal{K}_i)$ via supervised learning on the data set $\mathcal{K}_i$ obtained in Step 2 of the algorithm. This data set is first randomly split into a training set and test set with 95% of the samples assigned to the training set. For each sample, the loss is defined as the cross-entropy loss between 1) the softmax/softargmax of the output of the neural network for state $\mathbf{s}_k$, where the actions that are not applicable for the state are masked out and 2) the prescribed simulation-based action a (van Jaarsveld, 2021). This loss decreases when the neural network better prescribes the best action in each state, that is when the output to the best action increases or when the output of the other actions decreases. The parameters of the neural network are updated to minimize the average loss over the training set using stochastic gradient descent.

Each epoch, the training set is divided into mini-batches and while iterating over the mini-batches, the neural network parameters are updated in the opposite direction of the gradient of the average loss with respect to the neural network parameters. The size of the mini-batches is a hyperparameter. After every five epochs, the loss is computed on the test set. When the best-obtained test loss does not decrease for 20 consecutive epochs, the algorithm terminates and the best-performing neural network at that time is the neural network policy of this generation.

6 Methodology

This section provides the complementary methodology to the generic problems description and algorithm. Section 6.1 discusses how the large action space is effectively handled by decomposing the decision into sub-decisions. Section 6.2 presents the model with stationary uncertain demand. In Section 6.3, this problem is extended to model the non-stationary uncertain demand. In addition, both sections introduce the benchmarks to evaluate the performance of the DRL algorithm for their corresponding demand uncertainty. Section 6.4 discusses the key performance indicators used to evaluate the performance of the neural network policies. Lastly, an additional agent is proposed in Section 6.5 that has different features of the neural network.

6.1 Decompose into Sub-Decisions

This section explains how the large action space is effectively handled by decomposing the decision into sub-decisions. Production planning can be seen as a sequential decision-making problem that is modeled as a Markov Decision Process in Section 4.2. Each period, the same decision is made about the production quantity at each node. The decisions at all nodes are made simultaneously. ASML simultaneously decides how many products to produce at each node in that period. The resulting action space is N-dimensional, see (10). Consider a supply chain with four nodes and a production capacity of four. Here, the total number of possible actions is $5^4 = 625$ since a production capacity of four results in five possible actions where not producing is also a possibility. This number increases heavily with expansions in the number of nodes or the production capacity, which can become problematic if DRL is applied to supply chain problems of realistic size. As Vanvuchelen and Boute (2022) state, "Although DRL algorithms are able to handle problems with large state spaces well, current applications of DRL in- and outside inventory control remain limited to rather small and stylized problems due to scalability issues in the action space". Therefore, the decision in a period is cut up as N sub-decisions, one for each node. First, the generic idea and reduction of the action space are discussed. Then, the extended MDP is presented and the advantages for the algorithm are explained.

6.1.1 Action Space Reduction

The decision on the production quantities at each node is solved by breaking it down into sub-decisions, namely the production quantity decision at a single node. This results in sequential decision-making within one time period. First, the production quantity at the most upstream node is determined, then the algorithm iterates to the successor nodes until the most downstream node is reached. This greatly reduces the action space since each sub-decision is one-dimensional. The action space of the supply chain with four nodes and a production capacity of four is reduced to $5 \times 4 = 20$ since there are five possible actions times the four nodes. The idea of splitting the decision into sub-decisions comes from the principle of dynamic programming. Specifically, dynamic

programming solves complex MDPs by breaking them into smaller subproblems, where the optimal policy for the MDP provides the optimal solution to all subproblems of the MDP (Bellman, 1966). This is true when computing the exact solution is possible. In general, however, the exact solution can not be computed due to the curse of dimensionality. This may be encountered when the state, action, or outcome space is too large. For these systems, there is no theoretical guarantee that this relationship holds since the optimal solution is simply not available. On the other hand, the reasoning remains that if the algorithm finds the optimal solution to all subproblems, this should lead to good or near-optimal policies or even the optimal policy. Moreover, decomposing the decision into sub-decisions has a great advantage for obtaining the simulation-based policy and training the neural network which will be discussed below. First, the resulting extensions to the MDP formulation are presented.

6.1.2 Markov Decision Process for the Sub-Decision

The model explained in this section is an extension to the MDP as explained in Section 4.2. First, the additional variables are introduced and then the extended MDP is presented in which only the reward function remains the same. Appendix A.1 presents the entire MDP formulation.

Variables

$P_{t,n,l_n} \in \mathbb{N}_0$: inventory in production at node n in time t that becomes available as stock in l_n time units

$K \in \mathcal{N}$: the current sub-decision

When observing the environment the agent must know the current sub-decision, that is, the current node in which the action is being taken. Therefore, a number corresponding to the current node is included in the state vector. Furthermore, all previously made sub-decisions in this period must be included in the state vector. For instance, when the agent decides the production quantity at the most downstream node, the agent must observe the prescribed production quantities at all other nodes. The inventory vectors of all upstream nodes of the current node contain the prescribed production quantity in this period. The state vector is the feature of the neural network that must always have the same dimension. Therefore, only adding an extra variable to nodes in which a sub-decision is already made is not possible. Accordingly, an extra variable is added to the inventory vector of each node. This variable is equal to the prescribed production quantity of nodes upstream of the current node, and zero otherwise. All prescribed production quantities in this period will become available as stock after production lead time periods and hence, the number is P_{t,n,l_n}. The new state vector is as follows:

$$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \dots, P_{t,n,l_n-1}, P_{t,n,l_n}]_{\forall n \in \mathcal{N}}, K] \tag{15a}$$

Next to the action space being one-dimensional, the sub-decisions update the set of allowable

actions. The set of allowable actions (10a) - (10c) is replaced by (16a) - (16c), where the constraints apply to the corresponding node instead of to every node.

$$
\begin{aligned}
& X_{t,n} && && \\
\text{s.t.}\ & X_{t,n} \leq c && && \text{(16a)} \\
& X_{t,n} \leq IL_{t,n+1,0} && \text{if } n \neq N && \text{(16b)} \\
& X_{t,n} \in \mathbb{N}_0 && && \text{(16c)}
\end{aligned}
$$

Taking a sub-decision while being in a state results in a transition to a new state, which can be distinguished into a random and deterministic transition. The transition to states in between sub-decisions is the deterministic transition since no random exogenous information, the demand, becomes available in between sub-decisions. The demand is realized after taking all sub-decisions and, therefore, the transition to a new state after all sub-decisions are taken is random. It is assumed that after the final sub-decision is taken, first the deterministic occurs. Subsequently, in the next time step after the material outflow takes place, the random transition occurs. The deterministic transition dynamics are as follows:

$$
\begin{aligned}
& [\mathbf{s}_t] = f(\mathbf{s}_t, X_{t,K}) && && \\
\text{s.t.}\ & P_{t,K,l_K} = X_{t,K} && && \text{(17a)} \\
& IL_{t,K+1} = IL_{t,K+1} - X_{t,K} && \text{if } K \neq N && \text{(17b)} \\
& K = K - 1 && && \text{(17c)}
\end{aligned}
$$

The taken sub-decision is in production at the current node for the next production lead time periods by constraint (17a). Constraint (17b) subtracts the production quantity from the inventory in stock at the predecessor's node. The next sub-decision is for the successor's node which is ensured by constraint (17c). Here, the current time period remains the same. Next, the updated random transition dynamics are presented.

$$
\begin{aligned}
& [\mathbf{s}_t] = f(\mathbf{s}_{t-1}, \mathbf{X}_{t-1,0}, d_t) && && \\
\text{s.t.}\ & P_{t,n,i} = P_{t-1,n,i+1} && \forall i \in \mathcal{L}_n,\ \forall n \in \mathcal{N} && \text{(18a)} \\
& P_{t,n,l_n} = 0 && \forall n \in \mathcal{N} && \text{(18b)} \\
& IL_{t,n} = IL_{t-1,n} + P_{t-1,n,1} && \forall n \in \mathcal{N} \setminus \{1\} && \text{(18c)} \\
& IL_{t,1} = IL_{t-1,1} + P_{t-1,1,1} - d_t && && \text{(18d)} \\
& K = N && && \text{(18e)}
\end{aligned}
$$

Constraints (18a) together with constraint (17a) replace constraints (12a) and (12b) of the transition dynamics in Section 4.2.4. These constraints ensure that the decided production quantities are in

production and that all products that were in production need one time period less before becoming available as stock. Constraints (18b) set all sub-decisions in this period equal to zero periods since no decisions are taken yet. Constraints (12c) defining the inventory in stock levels at every node that does not satisfy customer demands are substituted by constraints (18c) and (17b). Constraint (18d) remains the same. Lastly, constraint (18e) forces that the current sub-decision is for the most upstream node.

6.1.3 Advantages for the DRL Algorithm

Section 5 introduces the DCL algorithm to obtain the neural network policy. The algorithm obtains this policy by approximate policy improvement steps, where a neural network is trained on a data set of states with the corresponding simulation-based prescribed action. Simulations are used to determine the action that yields the lowest costs in a state. It takes many simulations to determine with reasonable accuracy which action is preferable among all possible actions. Cutting up the decision into sub-decisions greatly reduces the number of possible actions in a state. This results in a large reduction in the number of simulations required. Considering the example with four nodes and a production capacity of four, cutting up the decision into sub-decisions results in finding the preferred action out of five actions in each state, whereas previously the best action had to be found out of 625. The algorithm obtains the simulation-based policy for a great number of states, thus cutting up the decision extremely decreases the number of simulations needed. This results in a reduction of computational time. On the other hand, cutting up the decision increases the number of states (see (15)). This results in an increase in the number of states to be selected in Step 1 of the approximate policy improvement steps. However, the overall computational time decreases since the reduction in computational time for finding the simulation-based prescribed action for each state is greater than the increase due to the larger number of states selected.

Besides reducing the number of simulations needed to find the simulation-based policy, cutting up the decision into sub-decisions has a major advantage in training the neural network. As mentioned earlier, in each state $\mathbf{s}$, the neural network policy prescribes the action corresponding to the highest neural network output of all actions in the allowable action set $a \in \mathcal{A}_{\mathbf{s}}$. By training the neural network, the neural network parameters are found such that the output for the action that yields the lowest costs is the highest for each input state $\mathbf{s}$.

Decomposing the decision into sub-decisions results in a significantly lower dimension of the output layer. Moreover, the neural network was required to train all its parameters for each state. Here, the action was a multi-dimensional vector consisting of the production quantities at each node, and one action is prescribed for each state from a large number of possible actions. Accordingly, the output of the neural network for the simulation-based prescribed action must be the highest out of all possible output nodes. In the example of the supply chain setting described in this section, the

neural network is trained to give the highest output to the corresponding action of 625 for each state.

The number of possible actions in each state is extremely reduced when cutting up the decision. Furthermore, only part of the output layer is relevant, since a state corresponds to an action for a specific node. As a result of including a number for the current node, the neural network understands the current node and therefore also recognizes the current part of the output layer corresponding to this state. This enables specific training of these parameters. In other words, the neural network can specifically train the part of the neural network relevant to the input state. Consider the same supply chain settings, but with the decisions cut up into sub-decisions, the neural network is trained to give the corresponding best action out of five actions for each state. One can imagine the training advantage when a neural network must give the highest output to one of five actions compared to one of 625 possible actions, for each state. To illustrate this, Figure 6.1 shows that only part of the output layer is relevant in a supply chain with four nodes and a production capacity of two. When the input state corresponds to the most upstream node, only the top three output elements are relevant.

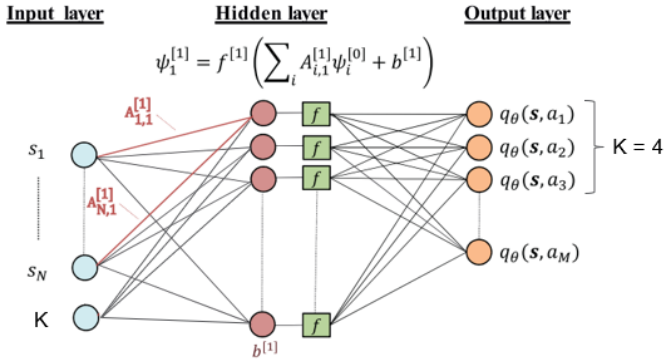


Figure 6.1: The neural network with node number included as feature

6.2 Stationary Demand

Two different scenarios of demand uncertainty are considered: stationary and non-stationary demand. In addition, the considered supply chain settings are distinguished into "exact" and "large" cases which respectively refer to problems for which the optimal solution can be computed and for which it is computationally infeasible. This section discusses the stationary demand uncertainty in more detail. First, Section 6.2.1 presents the stationary demand distribution. Then, Section 6.2.2 introduces additional constraints and the method to obtain the optimal solution for the exact cases. Lastly, Section 6.2.3 proposes a benchmark to evaluate the performance of the DCL algorithm in large cases.

6.2.1 Stationary Demand Distribution

Stationary demand uncertainty means that the demand has a constant distribution over time. That is, the demand is sampled from the same demand distribution in all time steps. Most of the literature on inventory management considers stationary demand uncertainty mainly because of the theoretical structure. Accordingly, benchmarks from the literature are considered in cases with stationary demand.

ASML faces low-volume demand which requires the demand to be an integer in value. Hence, the demand is sampled from a discrete distribution with values being non-negative integers. Furthermore, the number of possible demand realizations must both be finite and not extremely large to be able to compute the optimal solution. This enables iterating over all possible demand realizations. Section 6.2.2 will detail obtaining the optimal solution. In summary, the demand distribution is required to have a finite number of non-negative integer values.

In the stationary demand case, the objective function is to minimize the total supply chain-wide expected discounted costs over an infinite time horizon. The objective function directly follows from equation (8) with T infinite and equation (13). Here, T is geometrically distributed with discount parameter γ.

$$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} C(\mathbf{s}_t, \mathbf{X}_t^{\pi}) | \mathbf{s}_0\Big] \tag{19}$$

where the costs function is equal to equation 11.

6.2.2 Exact Cases - Stationary Demand

This section considers problems for which the exact optimal solution can be computed. However, this can only be found in settings where the number of possible state-action pairs is finite yet not extremely large. This requires incorporating additional assumptions in the model, which are discussed below. Subsequently, a method for finding the optimal solution is presented.

Additional Constraints

The state vector explained in Section 6.1 has an infinite number of possibilities for two reasons. First, the inventory in stock for the most downstream node can take any integer value. Second, the inventory in stock for the other nodes can take any non-negative integer value. The other variables in the state vector already have a finite number of possibilities. Furthermore, the action space is also finite. To obtain a finite number of state possibilities and thus state-action pairs, additional constraints are introduced into the model. An inventory capacity is introduced to constrain the inventory at each node. That is, the sum of inventory in stock and production can not exceed the inventory capacity. Consequently, the inventory in stock for each node is bounded from above by

the inventory capacity. A second constraint is a maximum backlog. Each unit of unmet demand that exceeds the maximum backlog is considered a lost sale, which is associated with a very costly lost sales penalty. This bounds the inventory in stock for the most downstream node from below.

These additional constraints not only ensure that the number of state-action possibilities is finite but also extend the MDP explained in Section 6.1. First, an additional variable and some additional parameters are introduced. Here, the extensions of the MDP are discussed whereas Appendix A.2 presents the entire MDP formulation.

Variables
$L_t \in \mathbb{N}_0$: Units of lost sales in time t
Parameters
$b \in \mathbb{Z}_-$: Maximum units of backlog demand
$v \in \mathbb{R}_+$: unit lost sales penalty
$i_n \in \mathbb{R}_+$: inventory capacity at node n

The lost sales variable must be observed by the agent when deciding on the production quantities and thus, it is included in the state vector. This results in the following state vector.

$$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \dots, P_{t,n,l_n-1}, P_{t,n,l_n}]_{\forall n \in \mathcal{N}}, K, L_t] \tag{20a}$$

The inventory capacity modifies the set of allowable actions because the agent can decide to produce a quantity only as long as the total inventory for that node, including the decided produced quantity, is less than or equal to the inventory capacity. In addition to the inventory capacity, the production quantity is still constrained by the production capacity and material availability and must be an integer. Therefore, the set of allowable actions while being in state $\mathbf{s}$ is the same as (16) with the following constraint included to ensure that the inventory capacity is not exceeded.

$$IL_{t,n} + \sum_{i \in \mathcal{L}_n} P_{t,n,i} + X_{t,n} \leq i_n \tag{21}$$

The reward function is updated to incorporate the lost sales penalty, if necessary.

$$C(\mathbf{s}_t, \mathbf{X}_t) = \sum_{n \in \mathcal{N} \setminus \{1\}} [h_n \cdot (IL_{t,n} - X_{t,n-1})] + z_t \tag{22}$$

$$\text{s.t.} \quad z_t = \begin{cases} h_{1t,1} & \text{if demand satisfied} \\ p \cdot -IL_{t,1} & \text{if backlog} \\ v \cdot L_t + p \cdot b & \text{if backlog + lost sales} \end{cases}$$

The random transition dynamics (18) also need to be updated. Constraint (23a) is included and

constraint (18d) is replaced by (23b).

$$L_t = \begin{cases} -IL_{t-1,1} - P_{t-1,1,1} + d_t - b & \text{if lost sales } (IL_{t-1,1} + P_{t-1,1,1} - d_t \leq b) \\ 0 & \text{otherwise} \end{cases} \tag{23a}$$

$$IL_{t,1} = \begin{cases} b & \text{if lost sales } (IL_{t-1,1} + P_{t-1,1,1} - d_t \leq b) \\ IL_{t-1,1} + P_{t-1,1,1} - d_t & \text{otherwise} \end{cases} \tag{23b}$$

Constraint (23a) ensures that the lost sales variable is set to the number of lost sales demand if there are lost sales otherwise this variable is set to zero. Constraint (23b) defines the recursive equation for the inventory level at the most downstream node if there are no lost sales and otherwise sets the inventory level to the maximum units of backlog.

Optimal Policy

In these settings with the additional assumptions, the number of possible state-action pairs is finite and with certain supply chain settings small enough to perform a tabular method. The exact optimal solution is found by explicitly solving the Bellman equations (4). This requires iterating over all possible state-action pairs with the probabilities to transition to new states. These transition probabilities are completely known due to the demand distribution that is known. Solving the Bellman equations is done with a hybrid between policy and value iteration where in the latter the actions are kept fixed. The optimal policy is found if, after one policy iteration step, the state-value functions have converged. That is, after one iteration the maximum change in state-value function of every state is smaller than a threshold.

Each value iteration updates for every state the expected cost when starting in state $\mathbf{s}$ and following policy π thereafter. In other words, updates the value function of a state $\mathbf{s}$ for a fixed policy π. Each updated state-value function is obtained by using the Bellman equation (4) as an update rule where the actions are determined by the policy which is kept fixed. The notation is related to the standard notation of Section 3.3.

$$v_{k+1}(\mathbf{s}) = r(\mathbf{s}, a_\pi) + \gamma \sum_{\mathbf{s}'} \mathbf{P}(\mathbf{s}'|\mathbf{s}, a_\pi) v_k(\mathbf{s}') \tag{24}$$

After 100 value iterations, the policy is updated to prescribe in each state the action corresponding to the lowest costs. Then, the algorithm performs one additional value iteration where the actions are prescribed by the new policy. This process is repeated until the state-value functions, after the one value iteration when the policy is updated, have converged. The optimal cost is the state-value function of the initial state of the MDP.

6.2.3 Large Case - Stationary Demand

In supply chains with more nodes and longer production lead times, the number of possible state vectors is too large to numerically solve the MDP and find the optimal policy. Therefore, a benchmark policy is introduced to evaluate the performance of the algorithm. For uncapacitated multi-echelon serial supply chain systems with linear costs, stationary demands, deterministic lead times between nodes, and unsatisfied demands backlogged, it is known that the optimal policy is a base-stock policy. The optimal base stock levels can be computed by recursively minimizing N nested convex functions, where N equals the number of nodes. Since the implementation and computation of this can still be difficult, heuristics are proposed in the literature to compute the base-stock levels. A well-known heuristic that is proven to have near-optimal performance when applied to solve inventory management in these supply chain settings is the Shang and Song heuristic introduced by Shang and Song (2003). A brief explanation of the heuristic is given below, followed by two modifications to make this heuristic applicable to the problem considered in this thesis. These large cases do not have the additional constraints explained in the previous section.

The Capacitated Shang and Song Heuristic

The Shang and Song heuristic determines echelon base stock levels for the base-stock policy. This policy places an order to raise the echelon inventory position to the echelon base stock level when it falls below the base stock level. Denote S_j as the echelon base stock level for echelon $j = 1, \ldots, N$ where N is the most upstream node. For node j, truncate the supply chain at node j, that is, remove all upstream nodes. Replace the truncated system with a Newsvendor system with as demand the demand over lead-time $L_j = \sum_{i=1}^{j} L_i$, as stockout penalty $p' = p + \sum_{i=j+1}^{N} h_i$, and with two different values for the holding cost. The values are $h'_u = h_j$ and $h'_l = \sum_{i=1}^{j} h_i$. Here, h_i is the echelon holding costs, that is, the additional holding cost in node i due to the value added. The echelon base-stock level is a simple average of the two optimal Newsvendor solutions where F^{-1} is the inverse of the demand over lead time function:

$$S_j = \frac{1}{2}\Big[F^{-1}\Big(\frac{p'}{p' + h'_u}\Big) + F^{-1}\Big(\frac{p'}{p' + h'_l}\Big)\Big], \forall j = \{1, \ldots, N\} \tag{25}$$

The actions when following the base-stock policy with the base stock levels calculated by (25) is the production quantity that brings the echelon inventory position up to S_j. However, these actions are not always possible due to capacity constraints or unavailable materials. Therefore, all actions prescribed by the base stock policy are checked to satisfy the set of allowable actions (16), with constraint (21) included in small cases. If the materials are not available or the capacity constraints are exceeded, the action prescribed by the base-stock policy is not allowed. In other words, this prescribed production quantity is too high. Therefore, this value is reduced by one until it is allowed, where not producing is always a feasible action.

The other modification is the order in which the algorithm considers the echelons. The Shang and Song heuristic considers the most downstream echelon ($j = 1$) first and then iterates to the most upstream echelon ($j = N$). However, our algorithm first considers the most upstream echelon and then moves to the most downstream echelon. The heuristic is adapted accordingly to match our algorithm and considers the most upstream echelon first. The production quantities are not affected by this since considering the most upstream echelon first prescribes the same production quantities as considering the most downstream node first. This benchmark is referred to as BS-SS.

The problem addressed is a periodic-review system with independent and identically distributed demands. Applying the Shang and Song heuristic in this system requires careful thinking about the lead time matching the dynamics of the supply chain. In our case, the costs are assessed at the beginning of the period during which the material inflows, outflows, and decisions also occur. Consequently, the lead time per node is equal to the number of production periods before the product is available as stock in that node. On the other hand, an additional period must be added to the lead time when the demand and decisions are realized at the beginning of the period, while costs are incurred at the end of the period. Moreover, the simple average of the two optimal Newsvendor solutions may be a decimal number that contradicts the assumption of low-volume demand. In our setting, the demand can only take discrete integer values, and therefore, the average of two integer values is an integer or has a decimal number of 0.5. Shang and Song (2003) argue that when p is small, say smaller than 39, truncation provides a slightly better approximation, for large p round-up. The backlog penalty is small for all cases considered, so the average base-stock level is rounded down to the nearest integer.

6.3 Non-Stationary Demand

This section considers problems where the demand distribution may change over time. This demand is called non-stationary uncertain demand. This type of demand uncertainty is typically observed in inventory problems since most practical demand patterns change over time. The demand in this section is sampled from a demand tree which is discussed in Section 6.3.1. The optimal policy can be computed when the demand is sampled in this way which is discussed in Section 6.3.2. In large cases with this non-stationary demand, no policies are known to be optimal or near-optimal. Therefore, three dynamic base-stock policies are introduced as benchmarks in Section 6.3.3.

6.3.1 Demand Tree

In the non-stationary setting, demand is sampled from a demand tree and a random noise distribution. Each period, the agent is at a position in the demand tree that corresponds to a certain average demand for that period. The realized demand is equal to the average demand of the position in the demand tree plus the realized random noise. The mean demand changes over time due to the demand tree and thus this covers the non-stationarity. The considered demand tree only

has integer mean demand values due to the low-volume industry. For the next period, the agent may be in the same horizontal position corresponding to keeping the same mean demand or moving upwards increasing the mean demand. This increase should be the same throughout the demand tree. Moreover, the probability of an upwards movement is the same throughout the entire time horizon and in every position. Figure 6.2 shows an example of a demand tree.

Given a forecast for the next period, there is always some uncertainty, which is captured by random noise. The random noise is sampled from a discrete distribution with a finite number of integer values. The noise can be negative when the lower bound of the noise distribution is set to a negative number, as long as the minimum of the noise plus the mean demand is greater than or equal to zero. Furthermore, the noise is identically distributed for the entire time horizon and is independent of the position in the tree.

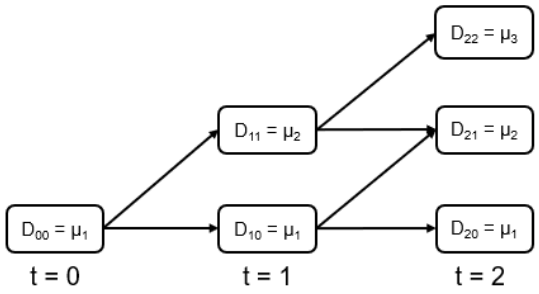


Figure 6.2: Example of the demand tree

The exogenous information is not the realized demand but consists of the realized random noise and the move in the demand tree for that period. The realized customer demand follows directly from the current state and the exogenous information. In other words, it is state dependent. The demand is equal to the mean demand corresponding to the current position in the tree after the move plus the realized noise. On the other hand, the exogenous information is identically distributed and independent of the state since both the probability of an upwards move and the noise is independent and identically distributed in the entire demand tree.

In non-stationary demand settings, the length of the planning horizon is finite and an input parameter. Subsequently, the demand tree is constructed for T periods. The objective of the algorithm is to minimize the total supply chain-wide expected cumulative discounted costs over the entire planning horizon.

$$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t C(\mathbf{s}_t, \mathbf{X}_t^{\pi}) | \mathbf{s}_0\Big] \tag{26}$$

The MDP in supply chain settings where demand is non-stationary is an extension of the MDP given in Section 6.1. This section presents the resulting changes in the MDP, while Appendix A.3 presents the full MDP formulation. First, an additional set and some additional parameters are introduced.

Sets

$\mathcal{Y}$: vertical positions in the tree, with index y

Parameters

$a_{t,y}$: mean demand of demand tree in position t, y

id_t : current vertical position in the tree in time t

$t \in \mathcal{T}$: current time period

$k_t \in \mathbb{Z}$: noise in time t

$m_t \in \{0, 1\}$: move upwards in tree in time t

$$m_t = \begin{cases} 1 & \text{if moved upwards in the demand tree in time } t \\ 0 & \text{keeping the same horizontal position in the demand tree in time } t \end{cases}$$

The state vector is updated to include the current vertical position in the demand tree and the current time period. Both variables determine the current position in the tree, which provides information about possible future positions and thus, future demand. Thus, the agent must observe them when making a decision. This results in the following state vector at time t.

$$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \dots, P_{t,n,l_n-1}]_{\forall n \in \mathcal{N}}, K, t, id_t] \tag{27}$$

The exogenous information of the MDP is updated and consists of the move in the demand tree and realized noise and is not the demand. Therefore, the MDP is updated accordingly. The demand in constraint (18d) of the transition dynamics is replaced by (28). Furthermore, constraints (29) and (30) are included in the random transition dynamics to update the vertical position in the tree and time period, respectively.

$$d_t = a_{t,id_t} + k_t \tag{28}$$

$$id_t = id_{t-1} + m_t \tag{29}$$

$$t = t + 1 \tag{30}$$

6.3.2 Exact Cases - Non-Stationary Demand

The optimal solution can be obtained when the demand is sampled from a demand tree plus noise distribution. First, the extended MDP in small cases is presented. Then, it is explained why the optimal solution can be obtained when the demand is sampled in this way.

The MDP of the exact case with stationary demand must be updated to incorporate the demand tree and noise. This MDP is an extension of the MDP explained in Section 6.2.2 with the same extensions as discussed in the previous section. Therefore, the current vertical position in the demand tree id_t and the current time period t are included in the state vector (20), and the transition dynamics are updated accordingly. That is, the demand equals (28) and the constraints (29) - (30) are included. The full MDP formulation can be found in Appendix A.4.

The main reason for modeling the non-stationary demand with a demand tree plus random noise is that the optimal solution can still be computed. Moreover, this nicely models the changing average demand over time. The optimal solution can be computed because the number of demand realizations is finite and not extremely large. Here, it is assumed that the realized demand does not deviate much from the forecast by the demand tree. Therefore, the noise distribution should only have a low number of values. The same method as described in Section 6.2.2 can be applied to find the optimal solution when there are finite and not extremely large number of possible state and demand realizations. However, the optimal solution can only be computed in supply chain settings with a smaller number of nodes or shorter production lead time than in stationary cases. This is because of the two extra variables that are included in the state vector while the limit on the total number of possible state-action pairs for which the optimal solution is available is the same. Furthermore, this implies that the length of the planning horizon can not be too large.

6.3.3 Large Case - Non-Stationary Demand

In settings where the optimal solution is computationally intractable, other benchmarks are introduced to evaluate the performance of the DRL algorithm. In non-stationary settings, there is no benchmark in the literature that is known to perform well. Therefore, three dynamic echelon base-stock policies are introduced that are specifically designed for this problem. Base-stock policies are considered since in the literature these policies are shown to be optimal in the uncapacitated setting with stationary demand. Furthermore, many studies propose dynamic base-stock policies for inventory problems with non-stationary demand. Dynamic base-stock policies are policies where the base stock levels can change over time. This seems reasonable in the non-stationary demand settings since the demand distribution can also change over time. The proposed base-stock policies all use the Shang and Song heuristic to determine the base stock levels. Therefore, the base stock levels are calculated using (25). The differences between the three benchmarks are the assumptions on the demand over lead time.

Non-Stationary Benchmark Policies

The first benchmark policy is an echelon base-stock policy which assumes that the demand per period is always distributed as in the root node. That is, the noise distribution plus the mean demand corresponding to the root node of the demand tree. This per period demand distribution is used

to calculate the demand over lead time. The first benchmark assumes that demand distribution does not change over time and therefore, the base stock levels are the same throughout the time horizon. This benchmark is referred to as *Base Stock - Root Node* (BS-RN) policy.

In non-stationary settings, the assumption that the demand distribution does not change over time is not valid. Therefore, the second benchmark policy assumes that the demand per period is distributed as the current position in the demand tree. That is, the noise distribution plus the mean demand corresponding to the current position in the tree. This results in a dynamic base-stock policy where the echelon base stock levels can change when the vertical position in the tree changes. The demand over lead time distribution is calculated for lead time periods with per period demand distribution being the demand of the current node plus the random noise distribution. Here it is assumed that the demand distribution does not change within the lead time. This benchmark is referred to as *Base Stock - Current Node* (BS-CN) policy.

The third benchmark policy relaxes the assumption that within lead time periods, the demand distribution does not change. In the next period, the system can move vertically in the tree and then the assumption that the demand distribution does not change within lead time is invalid. The third benchmark policy assumes that the mean of the demand distribution within lead time follows the demand tree from the current position in the tree and lead time periods ahead. This results in a probability function of the cumulative mean demand over lead time. In the end, the noise distribution is added to each cumulative mean demand value. From the resulting distribution, the echelon base stock levels are determined. These base stock levels can change when the system moves upwards. This benchmark is referred to as *Base Stock - Tree* (BS-T) policy.

Due to the increase in mean demand being equal for each move upwards and the equal probability of moving upwards in the demand tree, the dynamic base stock levels in the BS-CN and BS-T policies do not need to be recalculated after each move upwards. They follow directly from the echelon base stock levels calculated for the root node and the current position. At the start of the trajectory when the position is the root node, the echelon base stock levels are calculated for all three dynamic base-stock policy benchmarks. When the system moves upwards, the base stock levels can change in the BS-CN and BS-T policies. Instead of recalculating the echelon base stock levels, the echelon base stock levels will be equal to the root node echelon base stock levels plus echelon production lead time multiplied by the difference in the mean demand of the current position and the mean demand in the root node. This is because the demand over lead time distribution in the root node is the same as in the current position but then all values are increased by the echelon production lead time multiplied by the difference in the mean demand of the current position and the mean demand in the root node.

6.4 Key Performance Indicators

The DCL algorithm obtains the neural network parameters for the neural network policy by performing multiple approximate policy improvement steps, each following the procedure explained in Section 5. In Step 2 of the approximate policy improvement steps, a simulation-based policy is obtained for each state in the selected subset of states from Step 1. Here, (14) is used to determine the simulation-based prescribed action. For each approximate policy improvement step, the algorithm obtains a neural network policy. To determine the best-performing neural network policy, the policies are evaluated by their policy costs. These neural networks are also analyzed in terms of three other KPIs to gain additional insights into the performance. It should be noted that these KPIs are not included in the objective function. First, the method for obtaining the policy costs is discussed and then, the other three KPIs are presented.

Policy Costs

The cost of the neural network policy is computed for each neural network after all generations of the neural network have been trained. Clearly, the best-performing neural network is the one with the lowest cost. The costs are computed using different methods for exact and large instances with additional differences in the computation for the large instances with stationary and non-stationary demand. The latter difference is caused by the infinite time horizon in the stationary case, while the non-stationary case has a finite time horizon of length T. The costs in exact instances are computed as expected cumulative discounted costs when starting in the initial state $\mathbf{s}$ and following the policy thereafter. This is equal to the state-value function of state $\mathbf{s}$ to relate this to the MDP formulation in Section 3.2. In large instances with stationary demand, the cost of the policy is computed as the average cost per time unit, which seems most logical with an infinite time horizon. For large instances with non-stationary demand, the cost of the policy is computed as the average cumulative cost over the entire trajectory. First, the method for obtaining the expected policy cost in exact instances is discussed, and then the costs in large instances.

The expected cumulative policy cost in exact instances is the converged state-value function of the initial state. This is computed by performing value iterations with the actions determined by the policy until the state-values functions have converged. That is, iterate over (24) until the maximum change in the state-value function of each state is smaller than a small threshold. The expected cumulative policy cost of starting in the initial state $\mathbf{s}$ is the converged state-value function of this state. The only difference between this method and that for obtaining the optimal policy is that here all actions are fixed by the policy, whereas in the latter, policy iterations are performed to optimize the policy.

The average cost per time unit in large instances with stationary demand is calculated by simulating many trajectories. First, the average cost per time unit per trajectory is computed as the cumula-

tive cost over the trajectory divided by the number of periods in that trajectory. Then, the average cost per time unit of the policy is the average of all average cost per time unit per trajectory. The infinite horizon in instances with stationary demand allows obtaining long trajectories with many actions taken and demands realized. To eliminate the bias of starting in the same initial state, each trajectory has a different starting state. These initial states must be reasonable states that have a positive probability of occurring in practice. Therefore, a reasonable deterministic initial state is available, after which a warm-up period is simulated in which actions are determined by the policy. In this warm-up period, a trajectory is simulated for a warm-up number of periods starting from the initial state. The first trajectory from which the average cost per time unit is calculated begins in the final state of the warm-up period. Subsequent trajectories start in the final state of the previous trajectory. This eliminates the bias of starting in the same initial states and ensures that the states are realistic as they are visited by the agent. This method is governed by the following parameters: the warm-up period (W), the length of the trajectory for obtaining the average costs (LC), and the number of trajectories (NC_s). Here, a large number of trajectories with very long time horizons are simulated.

The average cumulative policy cost of large instances with non-stationary demand is calculated as the average of the costs obtained when simulating many trajectories of T periods. Each trajectory must start from a state with the root node in the demand tree as the current position. In addition, all initial inventories in stock and production must be values that reflect practice. Therefore, each trajectory starts from the same initial state, and due to the different demand realizations, each trajectory will result in a different sequence of states and actions. The parameter for this procedure is the number of trajectories (NC_{ns}). Here, a very large number of trajectories of length T is simulated. Policy costs in large instances have a standard deviation since the costs are computed as the average over many numbers. On the other hand, the policy costs in the exact cases are the converged expected cumulative policy costs and thus, have no standard deviation.

KPIs of Best-Performing Network

The best-performing neural network policy is further analyzed in terms of the following three KPIs:

- *Requested Line Item Performance (RLIP)*: the percentage of requested demand that is immediately satisfied
- *Average Lateness (AL)*: the average number of periods that the demand is delivered too late, that is, the periods between delivery and requested demand.
- *Average Inventory Value (AIV)*: the average value of on-hand inventory in all nodes of the supply chain over the full horizon

These three KPIs are commonly used within ASML where the combination of RLIP and AL can offer good insight into the service level of the prescribed production planning and the AIV provides

insight into the cost components. Typically, a high RLIP indicates a high service level because much demand is met immediately, whereas a low RLIP percentage indicates the opposite. However, the RLIP can not provide any insight into demand delay. In the RLIP, a demand met one period late has the same value as a demand not met for many periods. Therefore, the RLIP is usually considered together with the AL, with the AL providing additional insight into the delay of the demand.

Calculating these KPIs requires the simulation of a large number of trajectories and tracking the states, actions, and costs for each period in each trajectory. From an initial state, the steps of the dynamics of the supply chain, as explained in Section 4.1, are repeated until the end of the planning horizon. The initial state is the same for each trajectory of a given test instance. In the stationary case, the length of the planning horizon is equal to some input parameter (NH_s). From all visited states in the trajectory, the RLIP, AL, and AIV are calculated for this trajectory. For the AIV, also the unit holding costs per node must be known. This process is repeated for NK number of trajectories.

The KPI values of the best-performing neural network policy are compared with the values of the benchmark policies. The same method described above is used to compute the KPI values of the benchmark policies. For reliable comparison, the realized exogenous variable is kept the same for each policy That is, the realized exogenous variable is the same for the best-performing neural network and all benchmark policies in each period of each trajectory. Furthermore, the trajectories of the benchmark policies start in the same initial state as the best-performing neural network policy. As each policy may take different actions in a state, the trajectory of the policies and thus, the KPI values may be different for the policies.

6.5 Explorative Additional Neural Network Agent

The policy obtained by the DCL algorithm is represented by a neural network where the state vector is given as an input and the elements of the output vector are the actions. The prescribed action is the action corresponding to the highest neural network output of all allowed actions. The policy is contained in the neural network parameters. The goal of the DCL algorithm is to obtain the neural network parameters such that the neural network policy is a well-performing policy. This section presents an additional agent that also obtains a neural network policy by following the steps of the DCL algorithm described in Section 5, but with different features of the neural network.

The additional agent makes decisions based on the echelon inventory positions, that is, the sum of all inventories in stock and in production of all nodes between, and including, this node and the most downstream node. These inventory positions are the features of the neural network. The other agent makes decisions based on all information available and has the entire state vector as the features of the neural network. This additional agent follows a dynamic base-stock policy where

the base-stock levels are determined by the neural network parameters. The state vector remains the same but this agent only makes decisions based on the echelon inventory positions. This results in the same procedure for training this agent, see Section 5, where only the states in the data set are converted to the echelon inventory positions in Step 3. This agent can investigate whether prescribing the same actions for the same echelon inventory levels changes the costs of the policy, whereas the other agent can differentiate in prescribed actions with the same echelon inventory position. Furthermore, the number of features of the neural network is reduced in this agent, which may affect the training of the neural network.

7 Computational Experiments

This section presents a numerical case study to assess the suitability of the DCL algorithm to solve the tactical production-inventory problem. First, Section 7.1 describes the different instances in which the performance of the DCL algorithm is analyzed. The choice of hyperparameters is discussed in Section 7.2. The main results in terms of the policy costs are reported in Section 7.3. Insight into the production planning prescribed by the policies is provided in Section 7.4 by looking at the KPIs. Section 7.5 provide further insight into the prescribed actions in instances with two nodes. Lastly, the results obtained by the additional neural network agent are reported in Section 7.6.

7.1 Experimental Scenarios

This section presents the considered instances to assess the performance and suitability of the DCL algorithm to solve the tactical production-inventory planning problem. The algorithm is tested on a total of 28 different instances where each instance consists of a supply chain setting scenario and a demand scenario. The instances are designed such that there is a variety in terms of demand uncertainty type (stationary or non-stationary), demand scenarios, number of nodes, production lead time, holding costs, and backlog penalty. Production capacity is four in all considered scenarios. Therefore, the tightness of the capacity depends on the demand scenario. First, the demand scenarios are presented, and then the characteristics of the supply chain scenarios. These two scenarios are combined to create the test instances.

Demand Scenarios

The demand can be divided into stationary and non-stationary uncertain demand with each type of uncertainty having different demand scenarios. Stationary demand scenarios are presented first, followed by non-stationary demand scenarios.

Four different stationary demand scenarios are considered, each corresponding to a different demand distribution. Table 7.1 reports the proposed stationary demand scenarios. Columns two through eight show the distribution function per scenario, e.g., scenario S1 has 10% probability on realized demand of zero per period, 10% on one, 15% on two, 30% on three, and 35% on four. The last column is the average demand of the distribution in columns two through eight.

Table 7.1: Stationary Demand Probabilities per Scenario

	Demand per period							
Scenario	0	1	2	3	4	5	6	μ
S1	0.10	0.10	0.15	0.30	0.35	0.00	0.00	2.7
S2	0.10	0.15	0.15	0.20	0.20	0.15	0.05	2.9
S3	0.10	0.10	0.10	0.25	0.20	0.15	0.10	3.2
S4	0.05	0.10	0.10	0.20	0.25	0.20	0.10	3.5

The demand distribution in scenario S1 has the maximum per period demand equal to the production capacity. As mentioned earlier, ASML is not able to keep up with the demand and hence, this demand distribution seems to not represent the current situation. Nevertheless, this scenario is considered because it is known that the BS-SS should achieve near-optimal results in this setting. In demand scenario S2, the demand per period has 20% probability of exceeding the production capacity. In this scenario, the production capacity is not very tight because the average demand is significantly lower than the production capacity. Scenario S3 has a tighter capacity which can be seen by the increase in mean demand. Scenario S4 has the tightest capacity with a mean demand of 3.5 and a production capacity of 4. Moreover, the per-period demand has a 30% probability to exceed the production capacity.

In the non-stationary demand scenarios, demand is equal to the mean demand corresponding to the current position in the demand tree plus the realized random noise. The considered demand tree is a tree that starts with mean demand equal to two and with each upward movement the mean demand is increased by one. The probability of moving upwards is equal to 5% in every position of the tree. The tree is constructed for T periods with T as the input variable. Figure 7.1 shows the considered demand tree where the number above the arcs indicates the moving probability. The noise distribution is the discrete uniform distribution from -2 to 2 with only integer values. Table 7.2 presents the different non-stationary demand scenarios where only the length of the time horizon differs. The longer the time horizon, the tighter the capacity, since there is a higher probability of upward moves.

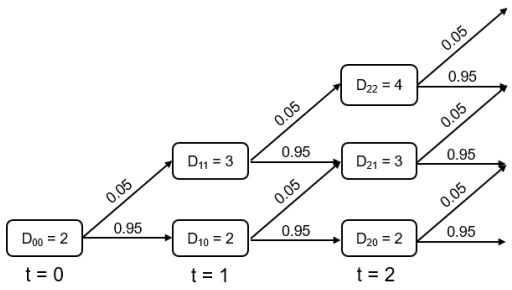


Figure 7.1: The demand tree under consideration

Scenario	T	Noise	$P[m_t = 1]$
NS10	10	$U[-2, 2]$	0.05
NS15	15	$U[-2, 2]$	0.05
NS20	20	$U[-2, 2]$	0.05
NS25	25	$U[-2, 2]$	0.05

Table 7.2: Non-Stationary Demand Scenarios

Supply Chain Setting Scenarios

Table 7.3 and Table 7.4 present all parameter values of the considered supply chain setting scenarios where the exact optimal solution is computationally feasible and infeasible, respectively. Each supply chain is a serial multi-echelon supply chain with a variety of the following parameter values: the number of nodes (N), the unit holding costs per node (h_n), the backlog penalty (p), and the production lead time per node (l_n). Furthermore, the exact cases also have additional parameter values for the maximum units of backlog (b), the lost sales penalty (v), and the inventory capacity per node (i_n). All parameter values for each scenario are shown in the tables. The production capacity is four in all scenarios and added to the table for a clear overview of all parameter values. The parameters that can have different values per node are represented with vectors, whose dimension is equal to the number of nodes. The first element of the vector corresponds to the most upstream node and the following elements for the corresponding successor nodes, e.g., the unit holding costs in scenario E23 is equal to one and four for the upstream and downstream node, respectively. The L223 supply chain is visualized in Figure 7.2, where the production lead times are given by the numbers on the arcs. Furthermore, the numbers on the nodes and customers are the unit holding cost and backlog penalty, respectively. The last column of the table lists all considered demand scenarios in combination with that supply chain setting scenario. Each scenario is named, with E and L corresponding to scenarios for which the optimal solution can be computed and those for which this is computationally infeasible, respectively.

The tables show that exact instances with stationary demand have two nodes with a maximum cumulative production lead time of five periods. On the other hand, the exact instances with non-stationary demand have two nodes but with a maximum cumulative production lead time of three periods and a horizon length of 10. These instances have shorter production lead times because of the two additional variables included in the state vector representing the current position in the tree (see (27)). In particular, this can be seen in instances E23 and NE23, which have the same production lead times, but for E23 with stationary demand the optimal solution can be computed

and for L23 with non-stationary demand this is infeasible.

Table 7.3: Supply Chain Setting Scenarios - Small cases

Scenario	N	h	p	l	c	b	v	i	Demand Scenarios
E23	2	$[1, 4]$	16	$[2, 3]$	4	12	1000	$[15, 18]$	S1 - S4
E11	2	$[1, 4]$	16	$[1, 1]$	4	12	1000	$[15, 18]$	NS10
E12	2	$[1, 4]$	16	$[1, 2]$	4	12	1000	$[15, 18]$	NS10
E21	2	$[1, 4]$	16	$[2, 1]$	4	12	1000	$[15, 18]$	NS10

Table 7.4: Supply Chain Setting Scenarios - Large cases

Scenario	N	h	p	l	c	Demand Scenarios
L23	2	$[2, 3]$	8	$[2, 3]$	4	NS15 - NS25
L223	3	$[1, 2, 3]$	8	$[2, 2, 3]$	4	S1 - S4 & NS15 - NS25
L1122	4	$[1, 2, 3, 4]$	8	$[1, 1, 2, 2]$	4	S1 - S4
L2223	4	$[1, 2, 3, 4]$	8	$[2, 2, 2, 3]$	4	S1 - S4 & NS15 - NS25

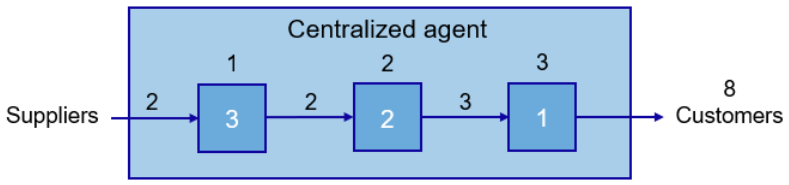


Figure 7.2: The L223 supply chain

Test Instances

Each instance consists of a supply chain setting scenario and a demand scenario. Thus, the instances have a variety in terms of demand scenarios per demand uncertainty type, number of nodes, production lead time, holding costs, and backlog penalty. Each instance is referred to these scenarios by the following instance name: supply chain setting scenario - demand scenario, for example, the instance with supply chain settings equal to scenario L23 and demand scenario NS15 is instance L23-NS15. A total of 28 test instances are created. Appendix B lists all instances and their parameter values.

7.2 Hyperparameters Settings

This section discusses the choice of hyperparameters for our experiments with the DCL algorithm. For each test instance, three generations of neural networks are trained following the steps in Section 5, requiring one approximate policy improvement step for each generation. All neural networks have four hidden layers with dimensions 256, 128, 128, and 64. The neural network is trained on a

data set with $K = 50,000$ and $25,000$ samples in the instances with stationary demand and non-stationary demand, respectively. These samples are obtained with $\underline{n} = 50$, $\bar{n} = 500$, and $\beta = 0.05$. Furthermore, the minibatch size is equal to 64 and $\epsilon = 0.02$, which is the same as proposed in van Jaarsveld (2021) where details about the latter hyperparameter can also be found. The initial policy for the first iteration of the approximate policy improvement steps is set to be the BS-SS and BS-T policy in instances with stationary and non-stationary demand, respectively. Each initial state has all inventories in stock and production equal to three and four, respectively.

The discount factor γ is set to one such that there is no discounting. This means that the length of the time horizon for instances with stationary demand must be changed since the geometric distribution will yield an infinite horizon length that results in never stopping trajectories. Therefore, the trajectories are truncated with a length of 40 periods. This can still be seen as minimizing the total expected costs over an infinite time horizon since the states do not contain any information about the trajectory length. The length is set to 40 periods because this is long enough relative to the accumulated production lead time but not too long to avoid unnecessarily long computation times. Namely, longer trajectories lead to longer computation times. Moreover, the roll-outs are also truncated with a length of 40 periods.

The choice of hyperparameters was mainly based on earlier experiments with the DCL algorithm and related MDP models. In this study, there was almost no tuning involved, except for K, $\underline{n}$, and $\bar{n}$. The objective of tuning the parameters was to find good performance in all cases where there was a trade-off between sampling and training time, and performance. Initially, the number of samples was set a lot lower ($K = 12,000$) with $\underline{n}$ and $\bar{n}$ higher ($\underline{n} = 500$ and $\bar{n} = 4,000$). Here, the neural network was trained using $12,000$ samples with, for each sample, a large probability that the simulation-based prescribed action was the action that yields the lowest costs. Namely, the higher $\underline{n}$ and $\bar{n}$ the higher the probability that, for each sample, the simulation-based prescribed action is the correct action that yields the lowest costs. However, this comes at a cost of longer sampling times. When tuning the parameters, it was found that the performance increased when the number of samples increased. Furthermore, decreasing $\underline{n}$ and $\bar{n}$ yielded approximately the same performance while reducing the computation time. First, K was set to $25,000$, $\underline{n} = 50$, and $\bar{n} = 500$, which resulted in better performance at about the same computation time compared to the initial settings. Further increasing the number of samples increased the performance but at the expense of longer computation times in instances with stationary demand. On the other hand, in instances with non-stationary demand, it led to similar performance but also to longer computation times. Therefore, each generation of the neural network is trained using $50,000$ and $25,000$ samples in the instances with stationary and non-stationary demand, respectively. In addition, the initial policy for the first iteration of the approximate policy improvement steps was initially set to a random policy. This policy takes a random action from the set of allowable actions where each allowed

action has an equal probability of being prescribed. In the first iteration, this policy determines the actions taken in the roll-outs after the first action. Furthermore, this policy determines the subset of states selection when an infinite time horizon is considered and thus, in our case, in the instances with stationary demand. The random policy was a very poor starting point, requiring much sampling and training to obtain good results. Therefore, changing the starting policy to these base-stock policies resulted in both better results and low computation times. Moreover, tuning showed that the results are rather insensitive to the precise choice of the hyperparameters, confirming other experiments with the DCL algorithm.

7.3 Main Results

This section presents the results in terms of the policy costs of the DCL algorithm compared to the benchmarks in all experimental scenarios. The experimentation was conducted using the Dutch national e-infrastructure with the support of the SURF Cooperative using grant no. EINF-5192. The DCL algorithm and the supply chain simulations were implemented in c++17 with PyTorch for the neural network training and inference (van Jaarsveld, 2021). Computations were performed on one thin node in Snellius which is the Dutch National supercomputer, operated by SURF. Each thin node has 2 AMD Rome CPUs running at 2.6 GHz and has a total of 128 CPU cores and 256 GiB of RAM (SURF, 2021).

The remainder of this section is organized as follows. Section 7.3.1 summarizes the results for the exact instances, followed by the performance in the larger cases in Section 7.3.2. All reported results are of the best-performing neural network in that instance.

7.3.1 Results in Exact Cases

Table 7.5 and Table 7.6 report the optimality gap for respectively the exact instances with stationary and non-stationary demand. The optimality gap is computed as $(C(\pi) - C(\pi^*))/(C(\pi^*)) \times 100\%$, with $C(\pi)$ and $C(\pi^*)$ denoting the expected cumulative cost of the policy and optimal policy, respectively. The expected cumulative policy costs are obtained using the procedure of Section 6.4 with a threshold of 1×10^{-6}. The optimality gaps are reported with rounding to two decimal places.

Both tables show the great performance of the DCL algorithm in instances that can be solved optimally. The optimality gaps for the DCL algorithm are very close to zero in all test instances. Moreover, the DCL algorithm reports similar costs as the optimal agent in two instances. The results show that the costs obtained by the neural network policy are lower in 6 of 7 test instances and equal in the other instance E23-S1, where the costs obtained by the neural network policy and the BS-SS policy are equal to the optimal cost. This illustrates the ability of the DCL algorithm to learn how to reduce supply chain-wide costs and obtain (near-)optimal policy costs. Moreover,

Table 7.5: Optimality Gap for Stationary Exact Instances

	Instance			
Policy	E23-S1	E23-S2	E23-S3	E23-S4
BS-SS	0.00%	0.41%	6.34%	23.81%
DCL	0.00%	0.31%	0.00%	0.45% [1]

Table 7.6: Optimality Gap for Non-Stationary Exact Instances

	Instance		
Policy	E11-NS10	E12-NS10	E21-NS10
BS-RN	23.86%	16.52%	21.19%
BS-CN	7.52%	2.43%	5.39%
BS-T	7.52%	8.32%	16.59%
DCL	0.20%	0.03%	0.39%

this shows that the neural networks can capture the structure of the optimal policy. The optimality gaps of the DCL algorithm do not appear to follow a pattern.

The results reported in Table 7.5 show that the base-stock policy is near-optimal in instances with the lowest production capacity relative to the demand distribution (instances E23-S1 and E23-S2). Here, the demand distribution is set in a way that the production capacity is much higher than the mean demand. Furthermore, the results show a decrease in the performance of the base-stock policy when the production capacity becomes tighter relative to the demand distribution. This confirms our expectations, as Shang and Song heuristic is proven to be (near-)optimal in uncapacitated multi-echelon supply chains with these characteristics, while the heuristic does not take capacity into account. Table 7.6 reveals that the benchmark policies are not close to the optimum in instances with non-stationary demand. It also follows from this table that the results of the BS-CN policy are the best of the three benchmarks.

In conclusion, the results presented in Table 7.5 and Table 7.6 demonstrate that the neural network policies obtained from the DCL algorithm have near-optimal performance for all tested instances for which the optimal solution can be computed.

7.3.2 Results in Large Cases

Table 7.7 and Table 7.8 present the results of the DCL algorithm in large instances. The results are reported as the difference between the cost of the neural network policy and the benchmark policy as a percentage of the benchmark policy costs. Thus, this is computed as $(C(\pi^{DCL}) -$

[1] This instance required four generations of training

$C(\pi))/(C(\pi)) \times 100\%$ with the neural network policy and benchmark policy denoted as π^{DCL} and π, respectively. The policy cost for instances with stationary demand is the average cost per time unit, while the policy cost for instances with non-stationary demand is the average accumulated cost of the trajectory with length T. These costs are computed following the method explained in Section 6.4 with $W = 100$, $LC = 10,000$, $NC_s = 100$, and $NC_{ns} = 10,000$. Furthermore, the tables report the standard deviation of the neural network policy cost in the percentage of the policy cost which is computed as $\sigma_{\pi^{DCL}}/C(\pi^{DCL}) \times 100\%$.

The tables show promising results for the DCL algorithm in large instances, where the DCL algorithm significantly outperforms all benchmark policies in 17 of 21 test instances. In particular, the neural network costs are significantly lower compared to the benchmark policies in 12 of 16 instances with stationary demand and in all instances with non-stationary demand. The costs obtained by the neural network in the four other instances with stationary demand do not significantly differ from the BS-SS policy since both costs are within the standard deviation of the neural network policy costs. Furthermore, when looking at the results reported in both tables, the performance of the DCL algorithm relative to the benchmark policies typically increases when the production capacity becomes tighter. Table 7.7 and Table 7.8 report the highest relative cost decrease in demand scenario S4 and NS25, respectively. This observation makes sense because the base-stock levels in the benchmark policies do not take into account the capacity. On the contrary, the results in the stationary cases demonstrate that the performance of the DCL algorithm relative to the BS-SS policy decreases when the number of nodes and the production lead time increase. Here, in the largest instances (L2223) the neural network policy obtains insignificant costs difference in two of the four instances. On the other hand, this relationship is not so clearly observed in instances with non-stationary demand. Moreover, Table 7.7 shows that the neural network policy can not significantly improve upon the costs obtained by the BS-SS policy in the three test instances with demand scenario S2. This is not in line with the expectations as it was expected that the performance of the DCL algorithm relative to the BS-SS policy would be similar in demand scenarios S1 and S2. This may be because of one set of hyperparameters that are used for all instances with stationary demand but future research should be conducted to evidence this. In summary, the neural network policies show good performance in the large cases where the policy learned to decrease the supply chain-wide costs in 17 of 21 test instances but we also found that the performance DCL algorithm in instances with stationary demand relatively decreases when the network size increases.

Table 7.7: Gap to benchmark policy in large instances with stationary demand

	Demand Scenario			
Supply chain settings	S1	S2	S3	S4
L223	−0.53% (±0.15)	−0.13% (±0.18)	−2.54% (±0.23)	−7.03% (±0.34)
L1122	−1.63% (±0.15)	+0.08% (±0.16)	−2.36% (±0.22)	−4.10% (±0.35)
L2223	−0.59% (±0.17)	+0.13% (±0.19)	+0.05% (±0.22)	−3.47% (±0.31)

Table 7.8: Gap to benchmark policies in large instances with non-stationary demand

		Demand Scenario		
Supply chain settings	Policy	NS15	NS20	NS25
L23	BS-RN	−22.13% (±0.50)	−30.38% (±0.62)	−35.39% (±0.79)
	BS-CN	−5.95% (±0.50)	−8.51% (±0.62)	−10.31% (±0.79)
	BS-T	−9.50% (±0.50)	−11.26% (±0.62)	−12.49% (±0.79)
L223	BS-RN	−21.97% (±0.46)	−31.05% (±0.55)	−36.87% (±0.72)
	BS-CN	−7.73% (±0.46)	−11.27% (±0.55)	−13.66% (±0.72)
	BS-T	−11.03% (±0.46)	−13.83% (±0.55)	−15.69% (±0.72)
L2223	BS-RN	−17.23% (±0.41)	−25.77% (±0.47)	−32.41% (±0.59)
	BS-CN	−6.13% (±0.41)	−9.08% (±0.47)	−12.07% (±0.59)
	BS-T	−10.04% (±0.41)	−9.93% (±0.47)	−10.14% (±0.59)

Learning the neural networks took a bit under 2 hours and 45 minutes for all 28 instances of Table 7.5, Table 7.6, Table 7.7, and Table 7.8 together. This includes the time for obtaining samples on which the neural network is trained through supervised learning. Of the total learning time, just over two hours was required for the instances with stationary demand, of which approximately 65% was for obtaining the samples. Individual training times ranged from six to ten minutes with increasing training times for instances with a larger number of nodes and longer production lead times. On the other hand, individual training times for instances with non-stationary demand ranged from only two to five minutes. Here, also the training times increased when the number of nodes increases and production lead times become longer, but also when the time horizon becomes longer. The larger training time in instances with stationary demand compared to non-stationary demand can be explained by both the higher number of samples and the longer planning horizon.

7.4 Results Key Performance Indicators

This section evaluates the performance of the neural network policy in terms of the following three KPIs: the *Requested Line Item Performance*, *Average Lateness*, and *Average Inventory Value*. All KPI values are computed for the best-performing neural network policy and the benchmark policies for each instance following the procedure explained in Section 6.4 with $NK = 1,500$ and $NH_s = 40$.

That is, the KPIs are computed for $1,500$ trajectories consisting of 40 and T periods in instances with stationary and non-stationary demand, respectively. The variable T depends on the supply chain setting scenario of the instance. The KPIs are analyzed based on the boxplots given in Figure 7.3 and Figure 7.4. Each figure consists of 12 boxplots where each column presents the results of a different KPI and each row corresponds to a different supply chain setting scenario. Each boxplot shows the KPI values obtained for all tested policies for the different considered demand scenarios. These values strongly depend on the choice of holding costs and backlog penalty, whose values are presented in Section 7.1. A clear comparison can be made between the demand scenarios since these parameter values are the same in each row of boxplots due to similar supply chain settings. The observed KPI values are compared against each other but not analyzed by the exact values since the choice of parameter values does not depend on research into ASML's supply chain.

Requested Line Item Performance (RLIP)

The RLIP is given in the first column of boxplots in Figure 7.3 and Figure 7.4 for instances with stationary and non-stationary demand, respectively. A high RLIP value implies that a large part of the demand is met immediately, indicating a high service level and thus, better performance. The results in Figure 7.3 show that in the test instances with stationary demand, the neural network policy achieves better performance in terms of RLIP compared to the BS-SS policy in 10 out of 16 instances, similar performance in 4 instances, and worse performance in 2 instances. Similar performance in terms of RLIP is considered when the minimum, maximum, median, and quartiles of both policies are close to each other and the averages are within 1% of each other. The latter two instances with worse RLIP performance for the neural network policy than the BS-SS policy are L1122-S1 and L1122-S2. The results in Figure 7.4 reveal that better performance is achieved by the neural network policy compared to the base-stock policies in all instances with non-stationary demand. Here, the minimum, median, and quartiles are higher for the neural network policy compared to the base-stock policies with only the maximum value being similar and equal to one in all test instances and policies. Moreover, both figures display similar RLIP values for the neural network policy and the optimal policy in all exact instances, except in instance E23-S4 where the RLIP is on average lower than the optimal policy. When the results presented in both figures and in the tables of the previous section are combined, it can be seen that a larger part of the demand is met immediately by the neural network policies compared to the base-stock benchmark policies in all instances where the neural network policy also reports lower costs.

Both figures demonstrate declining RLIP values in all base-stock benchmark policies when the capacity relative to the demand distribution becomes tighter. This confirms our expectations since the base-stock levels calculated by the Shang and Song bounds do not take into account the capacity. This relationship also seems to hold for the neural network policy in instances with non-stationary demand, but this decrease is smaller than that of the base-stock policies. Here, it should be noted

that the capacity is the same for all three instances in the top row of Figure 7.4. On the contrary, this relationship does not seem to hold for the neural network policies in instances with stationary demand. Although the minimum value decreases when the capacity becomes more tight in scenarios L223 and L1122 with stationary demand, the median, maximum, and quartiles do not decline. In instances E23 and L2223 with stationary demand, the boxplots reveal on average declining RLIP values when the capacity becomes tighter, but again this decrease is less compared to the base-stock policies. Hence, the performance in terms of RLIP of the neural network policy relative to the base-stock policies typically increases when the capacity gets tighter.

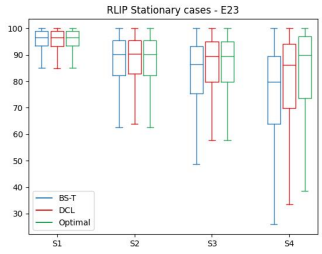
Average Lateness (AL)

The second column of boxplots in Figure 7.3 and Figure 7.4 reports the results obtained in terms of AL. This KPI shows similar patterns compared to the RLIP but here a low AL value indicates a high service level and better performance. The boxplots show that the neural network policies typically achieve lower AL values compared to base-stock policies in all instances where also higher RLIP values are reported. In these instances, the neural network policy cost is also lower than the cost incurred when following the base-stock policies. Only instance L2223-S1 stands out, as in this instance the AL performance achieved by the neural network is worse than that of the BS-SS policy while the RLIP performance is better. Furthermore, in instances where the neural network policy and the BS-SS policy report similar RLIP performance, the AL values are also approximately the same. In the two instances where a smaller part of the demand is immediately met when following the neural network policy compared to the BS-SS policy, the average delay is longer. The AL values reported in instances with non-stationary demand summarize that here the neural network policy outperforms the base-stock policies in terms of AL. Moreover, the same performance conclusion about the comparison between the neural network policy and the optimal policy can be made for the AL as the RLIP.

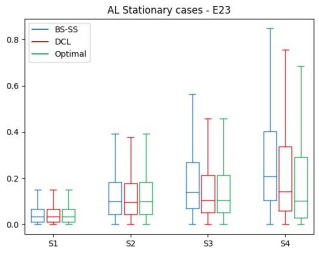
These figures also reveal that the AL values increase for all policies in all supply chain setting scenarios when the capacity becomes tighter relative to the demand, except for the neural network policy from instance L1122-S2 to L1122-S3. This relationship is most clear for the base-stock policy but this relationship also holds, but with a relatively smaller increase, for the neural network policy. Accordingly, this implies that the performance in terms of AL achieved by the neural network policy typically increases relative to the base-stock policies when the capacity gets tighter.

Average Inventory Value (AIV)

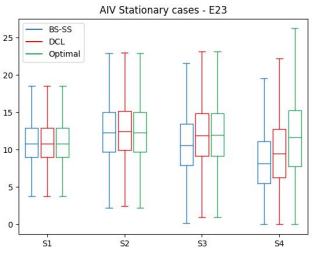
The last column of Figure 7.3 and Figure 7.4 presents the AIV results obtained for different test instances. The results demonstrate that the neural network policy reports on average higher AIV values compared to the base-stock policies in all test instances where the neural network policy achieves better performance in terms of RLIP and AL. Hence, this holds for 10 out of 16 instances



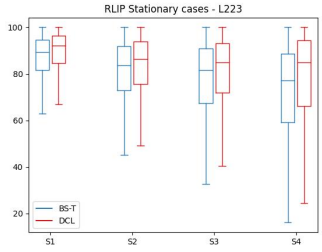
(a) RLIP - E23



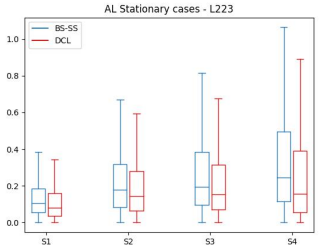
(b) AL - E23



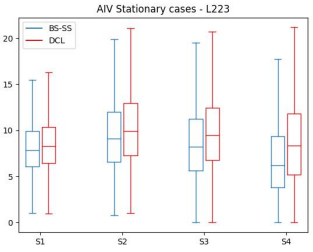
(c) AIV - E23



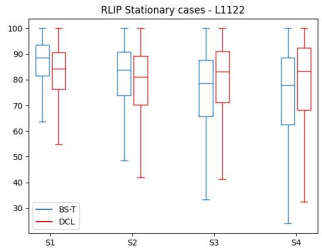
(d) RLIP - L223



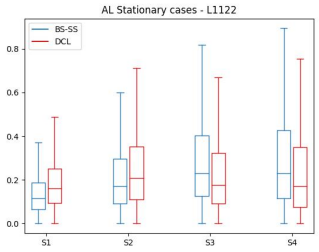
(e) AL - L223



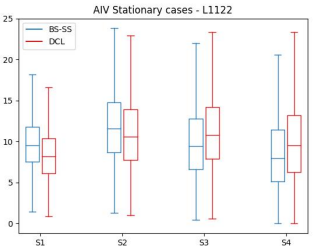
(f) AIV - L223



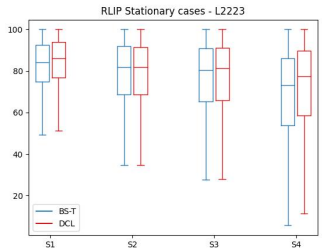
(g) RLIP - L1122



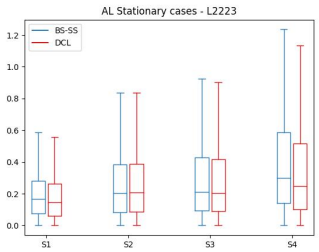
(h) AL - L1122



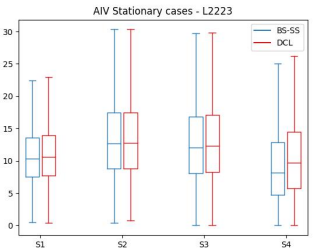
(i) AIV - L1122



(j) RLIP - L2223

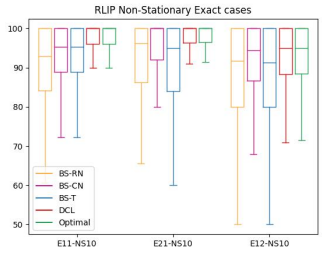


(k) AL - L2223

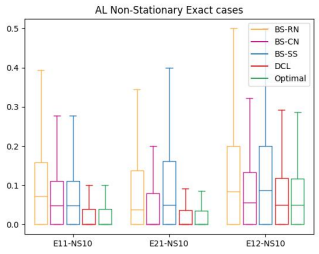


(l) AIV - L2223

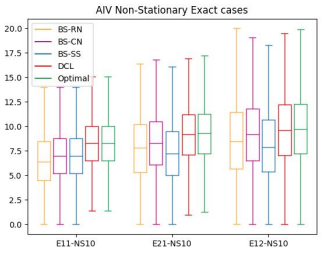
Figure 7.3: Boxplots of KPIs in instances with stationary demand



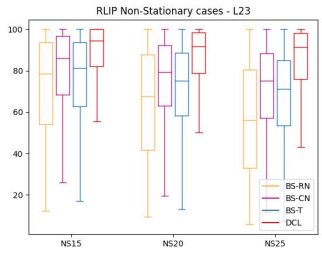
(a) RLIP - E-NS10



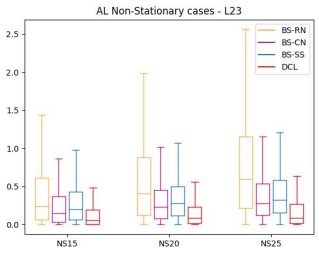
(b) AL - E-NS10



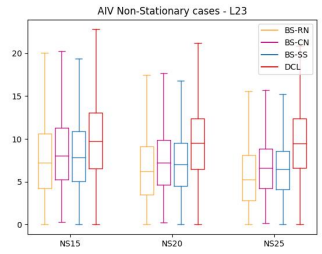
(c) AIV - E-NS10



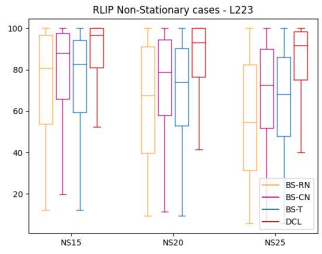
(d) RLIP - L23



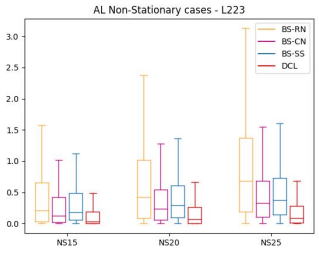
(e) AL - L23



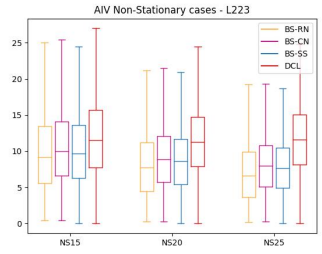
(f) AIV - L23



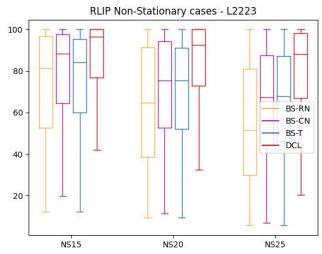
(g) RLIP - L223



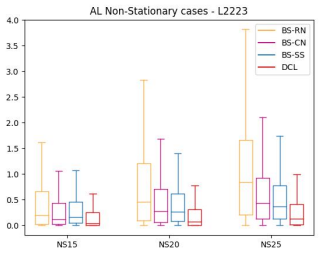
(h) AL - L223



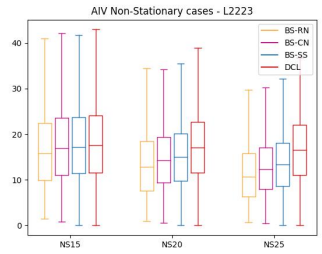
(i) AIV - L223



(j) RLIP - L2223



(k) AL - L2223



(l) AIV - L2223

Figure 7.4: Boxplots of KPIs in instances with non-stationary demand

with stationary demand and for every instance with non-stationary demand. The four instances with stationary demand that yield similar RLIP and AL values for the neural network policy and the BS-SS policy also have similar AIV values. The neural network policy has a lower average inventory value in the two instances where this policy has worse RLIP and AL performance compared to the BS-SS policy. The boxplots in Figure 7.3 display that the AIV values are on average lower in demand scenario S1 compared to S2. This seems logical since in S2 the mean demand is higher and the capacity is tighter relative to the demand distribution. Thus, more buffers may need to be placed to fulfill the uncertain demand. On the other hand, these values are lowered from S3 to S4 even though the mean demand increased. This implies that less safety stock is held in the latter demand scenario. For the BS-SS policy, the AIV values typically decrease from scenario S2 to S4. Moreover, the results display that in most cases the average inventory value held by the neural network policy relative to the BS-SS policy increases when the capacity becomes tighter. Figure 7.4 reports that the neural network policy obtains on average higher AIV values than all base-stock policies in the instances with non-stationary demand. Furthermore, the average AIV value decreases from demand scenario NS15 to NS25.

Conclusion on KPI Results

The neural network policy typically has a higher safety stock and a higher service level than the base-stock policies in instances where the neural network policy costs are lower than the base-stock policy costs. This is evidenced by the higher AIV and RLIP values and the lower AL values for these instances.

7.5 Actions taken by the Policy

This section analyses the differences in the actions taken by the policies. These results reported in the previous sections showed great performance of the neural network policies, but it remains unclear which actions lead to this increase in performance. The neural network policy must prescribe different actions in some states to obtain different (better) performance. This section set out to explain the neural network policies in instances with two nodes. Neural network policies are extremely difficult to interpret, but interpreting these might help with the adoption of neural network policies in practice. Here, the base-stock policies sharply contrast the neural network policies since they have an intuitive character and are thus easy to interpret.

A policy prescribes in each state an action where a state is a multi-dimensional vector. Accordingly, each element of the vector may affect the prescribed action. Neural network policies prescribe the allowed action that has the highest output after the neural network maps the state vector to the output action vector. The neural networks have multiple hidden layers consisting of many nodes where each layer performs a layer operation. This complicates explaining and interpreting a neural network policy. Moreover, the total number of possible states is extremely high making the policy

not suitable to explain using a table. In particular, the smallest stationary instances (E23) already have over one million possible states. To visualize the policy, the effects of changing two state variables are discussed by showing the actions prescribed by the policy for different values of these variables. The effects of only two variables are taken into account for visualization purposes since each variable adds a dimension to the graphs. The policies are visualized for instances with two nodes, where the effect of the on-hand inventory positions on the prescribed actions is investigated. First, the combinations of on-hand inventories common in practice are determined. Then, the actions prescribed for these inventory positions are visualized.

Actions in instance E23-S3

For the instance with stationary demand and two nodes, the actions are visualized with demand scenario S3. Here, the neural network policy outperformed the BS-SS policy and learned to obtain optimal costs. First, the on-hand inventory positions that are common in practice are analyzed by keeping track of the on-hand inventory values of both nodes over 1500 trajectories of 40 periods. These inventory values are the inventory values right before an action must be taken. This is after Step 2 of the dynamics of the supply chain discussed in Section 4.1. Figure 7.5 displays the probability of the combination of on-hand inventories right before taking an action. Here, nodes 2 and 1 are the upstream and downstream nodes, respectively. The figure demonstrates that the optimal policy and the neural network policy have approximately the same on-hand inventory position probabilities while the BS-SS policy has a higher probability of having lower inventory both in node 2 and node 1. This is in line with the results in terms of the KPIs for this instance where the neural network policy and optimal policy have equivalent values for the RLIP, AL, and AIV while the BS-SS policy has a lower AIV and RLIP value and a higher AL value. As the prescribed actions should only be visualized for combinations of inventory positions that have a positive probability to occur, the figure is discussed in more detail. Node 2 has the highest probability of an on-hand inventory position of four with positions ranging from 3 to 8. On the other hand, the inventory position of node 1 has a range from -10 to 10. Furthermore, it is observed that a combination of inventory in node 2 higher than five and inventory in node 1 lower than zero never occurs.

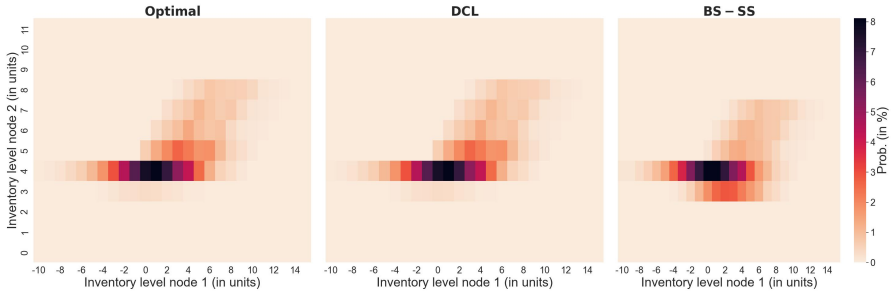


Figure 7.5: On-hand inventory positions probabilities in instance E23-S3

The prescribed actions in instance E23-S3 for different combinations of on-hand inventories are presented in Figure 7.6 for the three policies. The top and bottom rows of the figure show the prescribed actions for the upstream node and downstream node, respectively. In each figure, a part is masked out that corresponds to a combination of on-hand inventories with zero probability of occurring, see Figure 7.5. All inventories in production ($P_{t,n,i}$) at each node are set to three and kept constant. Here, three seems the most logical choice since the mean demand is equal to 3.2. Thus, all variables in the state vector are kept constant except the on-hand inventory positions and sub-decision variables. The latter variable is two in the top row and one in the bottom row corresponding to the node where the action is taken. The figure demonstrates that the neural network policy learns and is able to capture the structure of the optimal policy. All three policies have a diagonal where the actions change from four to zero at the upstream node. For the optimal policy and the neural network policy, this diagonal is similar. On the other hand, this diagonal is moved one inventory level at node 1 to the left for the BS-SS policy. This results in the BS-SS policy producing one unit less in the upstream node in these combinations. Consider the inventory position $(4, 4)$, $(4, 5)$, $(4, 6)$ where the first element is the inventory at node 1, the actions prescribed by the optimal policy, the neural network policy, and the BS-SS policy are $4, 3, 2$, $4, 3, 2$, and $3, 2, 1$, respectively. Here, it is clear that the BS-SS policy has the diagonal moved one inventory position of node 1 to the left. All three considered combinations of inventory positions have a positive and relatively high probability of occurring. In the actions prescribed for the downstream node, the neural network policy also captures the structure of the optimal policy and thus prescribes similar actions in almost all combinations of inventory positions.

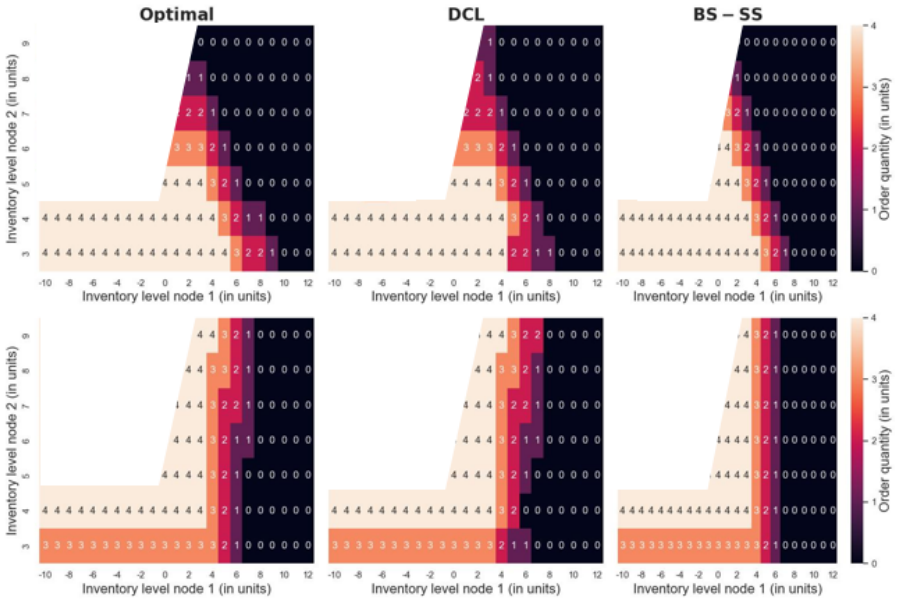
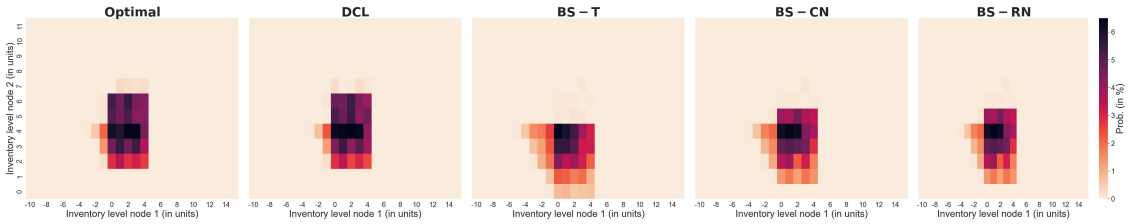


Figure 7.6: Prescribed actions in instance E23-S3 with $P_t, n, i = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n$. The top row shows the actions for node 2 and bottom row the actions for node 1

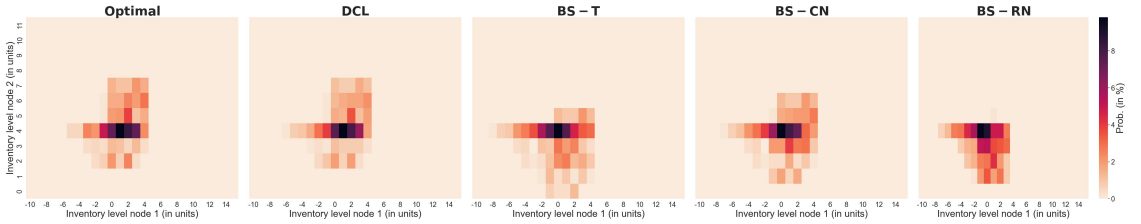
The actions are also visualized against other inventories in production. These figures are presented in Appendix C.1 where also the Figure 7.5 is stated for a clear overview. Figure C.2 and Figure C.3 show the prescribed actions for different on-and inventory levels when the inventory in production that become available as stock in one period for each node is set equal to two and four, respectively. All other inventories in production are kept at three. These figures demonstrate that the neural network is also able to capture the structure of the optimal policy in all three tested inventories in production vectors. Moreover, these figures illustrate that the policy learns to prescribe higher production quantities when the inventories in production are relatively low (2), and the opposite hold when the inventories in production are relatively high (4). In summary, the neural network policies learn to capture the structure of the optimal policy in all test settings for instance E23-S3.

Actions in instance E21-NS10

The same procedure is applied to visualize the actions of the policies in instance E21-NS10. However, each trajectory of the instances with non-stationary demand has the same initial state and each state has variables included corresponding to the current time period and current vertical position in the tree. Therefore, the current time period and vertical position must be kept fixed when visualizing the effects of the on-hand inventory positions on the prescribed actions. The prescribed actions are visualized for the fifth period. This is the middle of the time horizon where there is enough variability between trajectories due to the first five periods and the end-of-horizon



(a) Zero vertical movements



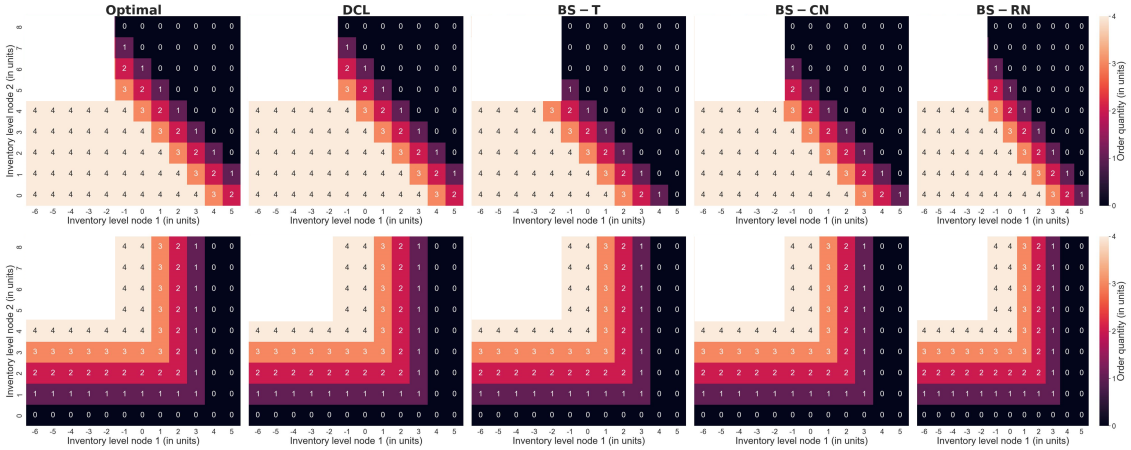
(b) One vertical movement

Figure 7.7: On-hand inventory positions probabilities in instance E21-NS10

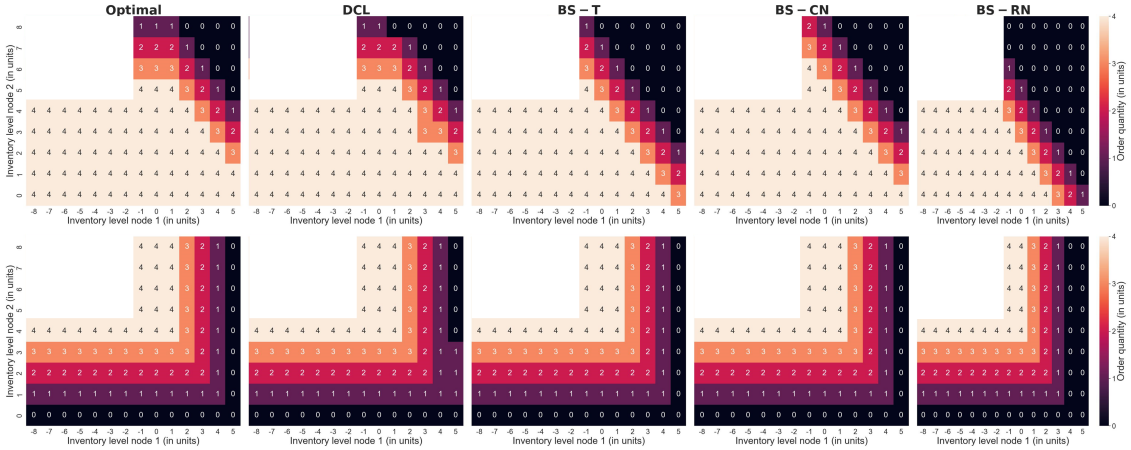
effect is not encountered. Furthermore, the vertical position is both set to zero and one to visualize that the neural network learns to take different actions when an upward movement has been made. The dynamic base-stock policies BS-CN and BS-T should also prescribe different actions when a move upwards has been made while the BS-RN policy should prescribe the same actions. First, the on-hand inventories are determined that are common in practice. This is done by simulating 2500 trajectories where the positions right before taking an action in the fifth period are stored with the right corresponding vertical position. In total 1911 trajectories had a vertical position of zero in the fifth period and 528 moved upwards once. The 61 other instances moved upwards at least twice. The probabilities of inventory positions in the fifth period before taking an action are presented in Figure 7.7 where the results for a vertical position of zero and one are reported in Figure 7.7a and Figure 7.7b, respectively. These figures show similar on-hand inventory positions when following the optimal policy or the neural network policy. These positions are higher than the positions of all base-stock policies. This is also in line with the results obtained in the previous section.

Figure 7.8 visualize the actions prescribed by each policy for different on-hand inventory positions. Figure 7.8a shows the actions when the current vertical position is equal to zero where Figure 7.8b report the actions when one vertical movement has taken place in or before the fifth period. The top row in both figures shows the prescribed actions for the upstream node and the bottom row for the downstream node which is similar to Figure 7.6. All inventories in production are also set to three at each node. Clearly, the neural network policies are able to learn the structure of the optimal policy. For the upstream node, these figures show that the neural network policy produces equal or more units compared to the base-stock policies. The diagonal is again moved to the

left for the base-stock policies compared to the neural network policy. For the downstream node, the base-stock policy is an optimal policy which is also prescribed by the neural network policy. Moreover, comparing Figure 7.8a with Figure 7.8b shows that the neural network learns to produce more when the system moved upwards once. The dynamic base-stock policies, BS-T and BS-CN, also learn to produce more while the prescribed production quantities are unchanged in the BS-RN policy. The same observation when comparing the actions for the neural network policy and the base-stock policies can be made that the neural network policies prescribe equal or more production quantities. The optimal production quantities for the downstream node when the vertical position is one is also a base-stock policy but with higher base-stock levels which is also captured by the neural network. These figures report the same patterns as in instance E23-S3 and are thus, in line with the boxplots of the previous section. Furthermore, the same visualizations are made for the actions with other inventories in production for both zero and one vertical movements which are given in Appendix C.2. These figures show a similar appearance where the neural network captures the structure of the optimal policy. Moreover, the neural network policy learns to prescribe higher production quantities when the inventory in production that becomes available in one period is equal to two compared to when these are three. On the other hand, lower production quantities are prescribed when these values are set to four. It can also be observed for all figures that the neural network policy prescribes equal or higher production quantities compared to all base-stock policies in the tested settings. Concluding, the neural network policies can capture the structure of the optimal policy which produces an equal or higher number of products compared to the base-stock policies.



(a) Zero vertical movements



(b) One vertical movement

Figure 7.8: Prescribed actions in instance E21-NS10 with $P_t, n, i = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n$. The top row shows the actions for node 2 and bottom row the actions for node 1

7.6 Results of the Additional Neural Network Agent

This section presents the results of the additional agent that has the echelon inventory positions as features of the neural network. This agent is referred to as the second neural network agent. In this section, the agent that has the entire state vector as features of the neural network is denoted as the first neural network agent. Table 7.9 and Table 7.10 report the differences in policy cost between the second neural network agent and the first neural network agent as a percentage of the first neural network agent. The reported gap is computed as $(C(\pi^2) - C(\pi^1))/(C(\pi^1)) \times 100\%$ with π^1 and π^2 the first and second neural network policy, respectively. The costs are obtained following the methods described in Section 6.4 with the same parameter values described in Section 7.3.2. That is, the costs in exact instances are the exact converged policy cost from starting in the initial state, the costs in large instances with stationary demand are computed as average cost per time unit, and the costs in large instances with non-stationary costs are the average accumulated costs over a trajectory of length T. The latter two costs are computed with $W = 100$, $LC = 10,000$, $NC_s = 100$, and $Nc_{ns} = 10,000$. The same initial state per test instance is used in the first and second neural network policies.

Table 7.9: Gap to Neural Network Agent in instances with stationary demand

Instance	Cost difference
E23-S1	+0.07%
E23-S2	0.00%
E23-S3	+0.89%
E23-S4	+0.42% [2]
L223-S1	0.00% (±0.15)
L223-S2	0.00% (±0.19)
L223-S3	−0.05% (±0.24)
L223-S4	−0.13% (±0.34)
L1122-S1	0.00% (±0.15)
L1122-S2	+0.07% (±0.17)
L1122-S3	−0.03% (±0.22)
L1122-S4	0.00% (±0.35)
L2223-S1	0.00% (±0.17)
L2223-S2	−0.13% (±0.19)
L2223-S3	−0.07% (±0.23)
L2223-S4	−0.36% (±0.31)

Table 7.10: Gap to Neural Network Agent in instances with non-stationary demand

Instance	Percentage cost difference
E12-NS10	+0.10%
E21-NS10	−0.19%
E11-NS10	+0.07%
L23-NS15	+0.02% (±0.51)
L23-NS20	−0.14% (±0.61)
L23-NS25	−0.03% (±0.81)
L223-NS15	−0.08% (±0.48)
L223-NS20	−0.17% (±0.54)
L223-NS25	−0.25% (±0.73)
L2223-NS15	−0.18% (±0.42)
L2223-NS20	−0.36% (±0.46)
L2223-NS25	−0.26% (±0.58)

Both tables show that the first neural network policy obtains lower costs in five of seven exact instances, the same cost in one instance, and in one instance the second neural network policy

[2] This instance required four generations of training

obtained lower costs, but it has to be noted that all cost differences are within 1%. In large instances, the tables summarize that there is no significant difference between the incurred costs when following one of the two policies. The cost difference is not greater than two standard deviations in all large test instances. The training and sampling time of the second agent was on average 7% higher which is solely on the training of the neural network since the sampling remains the same. The training is only needed once and thus, this increase does not seem to be a problem. As there is no significant cost difference, no further research has been devoted to the second neural network agent.

8 Conclusions and Future Research

This section contains the conclusions resulting from the problem definition, MDP formulations, methodology, and experimental results presented in the previous sections. The section closes with limitations and suggestions for future research.

8.1 Conclusion

The production-inventory problem is a sequential decision-making problem under uncertainty that arises in inventory management and is critical to the performance of a supply chain network. Despite extensive research, the optimal policy remains unknown for the inventory problems studied in this thesis. Therefore, this thesis applies the Deep Controlled Learning (DCL) algorithm to solve the tactical production-inventory planning problem in serial multi-echelon supply chains with finite production capacity, production lead times, and uncertain demand.

A central decision-maker chooses production quantities to meet customer demand while minimizing the total cost of the entire supply chain. This problem is formulated as a Markov Decision Process and is solved by applying the DCL algorithm. This algorithm obtains policies by iterating the procedure where a dataset consisting of a subset of states and their corresponding simulation-based best actions is sampled and a neural network is trained on this dataset which is used to represent policies. As multiple decisions must be made simultaneously, the resulting action space in the MDP formulation is multi-dimensional. However, large action spaces remain difficult to handle for DRL algorithms due to scalability issues in the action space. Therefore, the decisions are decomposed the decisions into sub-decisions, where each sub-decision decides on the production quantity for a single node. This effectively handles the multi-dimensional action space and has a great advantage in both training the neural network and finding the simulation-based prescribed action while providing good results.

Furthermore, the model is extended to allow for non-stationary demand. Non-stationary demand is sampled from a demand tree and a random noise distribution, with the demand tree modeling the changing average demand over time. On the contrary, stationary demand is sampled from the same distribution throughout the time horizon. The other key difference between these models is that the time horizon is infinite when stationary demand is considered whereas the time horizon is finite in models with non-stationary demand. Moreover, the models are further divided into two cases. One for which the optimal solution can be computed and the other for which this is computationally infeasible. For computing the optimal solution, two additional constraints are introduced into the model to ensure that the total number of possible state-action pairs is finite. Thus, a total of four different MDPs are formulated, and for each case, multiple different supply chains are considered to better assess the suitability and performance of the DCL algorithm.

The algorithm has been tested on 28 supply chain networks, each with variations in demand uncertainty, number of nodes, production lead times, and cost parameters. The experiments show highly promising results for the DCL algorithm, which demonstrates (near-)optimal behavior in all instances for which the optimal solution can be found. Furthermore, the neural network policies obtained from the DCL algorithm significantly outperform the base-stock policies in most test instances. The results illustrate that the costs incurred by the DCL algorithm relative to the base-stock policies are significantly lower in all instances with non-stationary demand and in instances with tight capacity relative to the demand distribution. However, the DCL algorithm does not vastly outperform the base-stock policy in all test instances with stationary demand, especially when either the capacity is relatively low compared to the demand distribution, or when the network has a relatively large number of nodes and long production lead times. Nevertheless, the DCL algorithm seems a promising general-purpose solution technique to solve the tactical production-inventory problem in serial multi-echelon supply chains. Moreover, the tuning of the hyperparameters appears to be a relatively simple task and the results are rather insensitive to the precise choice of these parameters.

To provide insight into the neural network policy, the policy is analyzed through an extensive study in terms of three key performance indicators: Requested Line Item Performance (RLIP), Average Lateness (AL), and Average Inventory Value (AIV). This study revealed that the neural network policy generally achieves lower costs by increasing the safety stocks resulting in higher service levels which is evidenced by the higher RLIP and AIV, and lower AL values. Additionally, to investigate which different actions result in better performance, the actions prescribed by the policies were analyzed for different in supply chain networks with two nodes. This showed that the neural network policies typically prescribe an equal or higher production quantity compared to the base-stock policies. This is consistent with the results in terms of the KPIs. Moreover, the neural network policy in these instances is shown to capture the structure of the optimal policy. It should be acknowledged that neural network policies remain extremely hard to interpret.

In conclusion, the results demonstrate the ability of the DCL algorithm to not only learn how to decrease the cost of the entire supply chain but also to obtain policies that can match and, in most test experiments, significantly outperform the base-stock policies designed specifically for this problem. On the other hand, in supply chains with stationary demand, the performance of the DCL algorithm in supply chains relative to the base-stock policy decreased with longer production lead times and more nodes. This study further provides evidence that neural networks can effectively be used to represent policies in inventory problems. Yet, neural network policies remain a black box. Thus, DRL seems a promising general-purpose solution technique to solve tactical production-inventory problems in serial multi-echelon supply chains.

8.2 Limitations and Future Research

This thesis has its limitations, mainly because of simplifications that had to be made to model the complex supply chain of ASML. Therefore, more research can be conducted in this very promising area. This section divides the limitations and suggestions for future research into two main parts. The first part is concerning the modeling and the other concerning the interpretation of the results and the possibilities of the neural network policies.

The Model

In modeling the supply chain network and formulating the MDP, several simplifications have been made that need to be mitigated in the future. The first simplification concerns the demand uncertainty which is modeled as stationary demand uncertainty or with a demand tree and random noise. This choice was made because the optimal solution can be computed when sampling the demand in this way. However, no research was performed on historical demand data for ASML or demand forecasts in the coming years. Analysis of this could thus provide additional insight to better represent the demand that ASML faces. There are most likely other models that better capture the demand uncertainty but it should be noted that the exact benchmark with most other demand models will not be available. Another possibility is to analyze the demand for better parameter values of the demand distribution while keeping the same demand model.

Furthermore, ASML faces significant supply uncertainty which is not included in our model. Here, the assumption is made that infinite supply is available with no uncertainties. For a good representation of the supply uncertainty, this should first be analyzed. On the other hand, adding some initial supply uncertainty is a relatively easy task where, for example, one could introduce probabilities for each item ordered regarding product delivery or delay.

All supply chain parameters given in Table 7.3 and Table 7.4 are not based on an analysis of ASML's supply chain. Therefore, updating these values to more realistic values will result in better representations of the supply chain of ASML. In doing so, a decision must be made about the length of the time periods. ASML plans on a weekly and monthly basis, where this decision has a major impact on the production lead time and capacity. Setting the decision periods to weeks results in a large number of periods as production lead time while the production capacity is relatively low. On the other hand, months mean that the lead times are not too large but the production capacity must be increased. Large production lead times increase the dimension of the state vector, while large production capacities increase the number of possible actions. In both scenarios, more research must be conducted on the performance of the DCL algorithm in these settings.

Another simplification is that this thesis considers only one product which is manufactured using one component at each node. However, in practice, each system has a bill of materials consisting of

multiple components. Thus, each node has multiple components that can be stored and produced. This results in decisions that must be made on how much of each component to produce at each node at each period. This increases the total number of decisions that must be made in one period. Furthermore, this requires consideration of the production capacity, as the capacity can be aggregated production capacity over all components it produces or for subsets of components. A natural further extension is to consider multiple end-products. Different end-products typically have similar components at upstream nodes which introduces commonality. As a result, the decision as to for which end product this component will be used can be made later in the supply chain. This turns the supply chain into a divergent system.

Moreover, a characteristic of the supply chain network of ASML is sparse parts demand, that is, demand at the middle nodes. Therefore, a natural extension to this paper is to incorporate this into our model. The exogenous information should then include the spare parts demand, and a decision should be made to meet the spare parts demand or to further produce the parts for finished end systems. This will also turn the network into a divergent system.

All of these extensions may change the state vector, action space, and transition dynamics of the MDP, but they all appear suitable for inclusion in the model. Further research is needed to get insight into the effects on performance.

Results Interpretation and Future Possibilities

In addition to the simplifications to the supply chain network, this thesis has its limitations in interpreting the results and the neural network policies. Therefore, future research may help to adopt DRL and neural network policies in practice.

There are limitations in interpreting the results, especially in non-stationary cases. Here, three base-stock benchmark policies are proposed which are shown to be quite far from optimal. Thus, evaluating the performance of the DCL algorithm against benchmarks that report relatively high costs may give a wrong impression. Therefore, more research should be done on other benchmark policies that report costs closer to optimality in exact instances with non-stationary demand. One base-stock policy that may show good performance is an extension of the BS-T benchmark policy that includes the noise distribution in each period of demand distribution over lead time instead of only adding the noise in the last period. This still ensures that the demand distribution within lead time follows the tree. The cumulative lead time demand distribution will then follow the tree with the noise distribution included in every position. This will result in higher base-stock levels, which may bring the costs obtained by this base-stock policy closer to the DCL and optimal policy.

Another limitation of the model with non-stationary demand is that the current period is included in the state vector. This, together with the vertical position, determines the position in the demand tree. However, this results in an end-of-horizon effect as the agent knows the current period and the total length of the horizon, the latter one being known by simulations. This end-of-horizon effect may result in an inventory vector of the neural network policy at the end of the time horizon that is not preferred. Namely, the algorithm learns that these quantities will not be used to meet customer demand as the cumulative production lead time is higher than the remaining periods in the trajectory. Future research should investigate ways to eliminate this. One possibility could be to only include the vertical position in the state vector since only this corresponds to the mean demand in the demand tree and the agent would then not know when it is approaching the end of the time horizon.

Moreover, instances with non-stationary demand all start in the same initial state. This is necessary for the root node of the tree, but the initial inventories on-hand and in production could slightly differ from one trajectory to another. Starting from the same initial state results in similar states that are sampled in the approximate policy iterations steps. This introduces a bias of starting in the same initial state when computing the policy costs and other KPI values. Thus, it is preferable to eliminate this. It should be noted that for reasonable initial inventory values, the randomness is limited but some randomness could be added.

A limitation of neural network policy, in general, is that it is extremely difficult to interpret and, therefore, remains a black box. This limitation is also mentioned in the literature (see, e.g., (Boute et al., 2021), (Gijsbrechts et al., 2021), and (Yan et al., 2022)). More research is required to explain the neural network policies and to generate intuitive policies or structural insights from the neural networks that can support decision-making. This study addresses this only slightly by visualizing the KPIs and analyzing the effect of the on-hand inventories on the policy in a supply chain network with two nodes. However, much more research can be done here, which will also help in adopting DRL and neural network policies in practice. If our black-box model prescribes production quantities, which cannot be explained, decision-makers are not likely to follow the prescribed production quantities. On the other hand, when the suggestions have an intuition behind them, the probability of adopting these actions increases. This may come at the expense of performance since interpretable policies might not be as flexible as neural networks.

Lastly, more research could be performed in fine-tuning the algorithm and the choice of hyperparameters. A sensitivity analysis on the latter could help find the best hyperparameters for the considered problems. Although the algorithm is quite robust to the precise choice of hyperparameters, further fine-tuning could improve the performance of the algorithm.

References

Alves, J. C., & Mateus, G. R. (2022). Multi-echelon supply chains with uncertain seasonal demands and lead times using deep reinforcement learning. *arXiv preprint arXiv:2201.04651*.

Alves, J. C., Silva, D. M. d., & Mateus, G. R. (2021). Applying and comparing policy gradient methods to multi-echelon supply chains with uncertain demands and lead times. *International Conference on Artificial Intelligence and Soft Computing*, 229–239.

Alves, J. C., & Mateus, G. R. (2020). Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands. *International Conference on Computational Logistics*, 584–599.

ASML. (2022a). *Asml annual report 2022*. Annual Report 2022.

ASML. (2022b). *Euv lithography systems*. `https://www.asml.com/en/products/euv-lithography-systems`

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, *60*(6), 503–515.

Bellman, R. (1966). Dynamic programming. *Science*, *153*(3731), 34–37.

Bookbinder, J. H., & Tan, J.-Y. (1988). Strategies for the probabilistic lot-sizing problem with service-level constraints. *Management Science*, *34*(9), 1096–1108.

Boute, R. N., Gijsbrechts, J., van Jaarsveld, W., & Vanvuchelen, N. (2021). Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*.

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.

Clark, A. J., & Scarf, H. (1960). Optimal policies for a multi-echelon inventory problem. *Management science*, *6*(4), 475–490.

CSCO. (2011). Five strategies for improving inventory management across complex supply chain networks.

de Kok, T., Grob, C., Laumanns, M., Minner, S., Rambau, J., & Schade, K. (2018). A typology and literature review on stochastic multi-echelon inventory models. *European Journal of Operational Research*, *269*(3), 955–983.

Eruguz, A. S., Sahin, E., Jemai, Z., & Dallery, Y. (2016). A comprehensive survey of guaranteed-service models for multi-echelon inventory optimization. *International Journal of Production Economics*, *172*, 110–125.

Federgruen, A., & Zipkin, P. (1986). An inventory model with limited production capacity and uncertain demands i. the average-cost criterion. *Mathematics of Operations Research*, *11*(2), 193–207.

Fleuren, T., Yasemin, M., Hendriks, M., & Renata, S. (2022). Tactical production planning and strategic buffer placement under demand and supply uncertainty in the high-tech manufacturing industry. *working paper*.

Giannoccaro, I., & Pontrandolfo, P. (2002). Inventory management in supply chains: A reinforcement learning approach. *International Journal of Production Economics*, *78*(2), 153–161.

Gijsbrechts, J., Boute, R. N., Van Mieghem, J. A., & Zhang, D. (2021). Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. *Manufacturing & Service Operations Management*.

Glasserman, P., & Tayur, S. (1995). Sensitivity analysis for base-stock levels in multiechelon production-inventory systems. *Management Science*, *41*(2), 263–281.

Glasserman, P., & Tayur, S. (1996). A simple approximation for a multistage capacitated production-inventory system. *Naval Research Logistics (NRL)*, *43*(1), 41–58.

Graves, S. C., & Willems, S. P. (2003). Supply chain design: Safety stock placement and supply chain configuration. *Handbooks in operations research and management science*, *11*, 95–132.

Graves, S. C., & Willems, S. P. (2008). Strategic inventory placement in supply chains: Nonstationary demand. *Manufacturing & service operations management*, *10*(2), 278–287.

Janakiraman, G., & Muckstadt, J. A. (2009). A decomposition approach for a class of capacitated serial systems. *Operations Research*, *57*(6), 1384–1393.

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274.

Kosasih, E. E., & Brintrup, A. (2021). Reinforcement learning provides a flexible approach for realistic supply chain safety stock optimisation. *arXiv preprint arXiv:2107.00913*.

Kroese, D. P., Botev, Z. I., Taimre, T., & Vaisman, R. (2019). *Data science and machine learning: Mathematical and statistical methods*. Chapman; Hall/CRC. `https://people.smp.uq.edu.au/DirkKroese/DSML/`

Kurawarwala, A. A., & Matsuo, H. (1996). Forecasting and inventory management of short life-cycle products. *Operations Research*, *44*(1), 131–150.

Li, B., Wang, H.-W., Yang, J.-B., Guo, M., & Qi, C. (2011). A belief-rule-based inventory control method under nonstationary and uncertain demand. *Expert Systems with Applications*, *38*(12), 14997–15008.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Mortazavi, A., Khamseh, A. A., & Azimi, P. (2015). Designing of an intelligent self-adaptive model for supply chain ordering management system. *Engineering Applications of Artificial Intelligence*, *37*, 207–220.

Neale, J. J., & Willems, S. P. (2009). Managing inventory in supply chains with nonstationary demand. *Interfaces*, *39*(5), 388–399.

Oroojlooyjadid, A., Nazari, M., Snyder, L. V., & Takáč, M. (2022). A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, *24*(1), 285–304.

Peng, Z., Zhang, Y., Feng, Y., Zhang, T., Wu, Z., & Su, H. (2019). Deep reinforcement learning approach for capacitated supply chain optimization under demand uncertainty. *2019 Chinese Automation Congress (CAC)*, 3512–3517.

Perez, H. D., Hubbs, C. D., Li, C., & Grossmann, I. E. (2021). Algorithmic approaches to inventory management optimization. *Processes*, *9*(1), 102.

Pontrandolfo, P., Gosavi, A., Okogbaa, O. G., & Das, T. K. (2002). Global supply chain management: A reinforcement learning approach. *International Journal of Production Research*, *40*(6), 1299–1317.

Puterman, M. L. (2014). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shang, K. H., & Song, J.-S. (2003). Newsvendor bounds and heuristic for optimal policies in serial supply chains. *Management Science*, *49*(5), 618–638.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484–489.

Simchi-Levi, D., & Zhao, Y. (2012). Performance evaluation of stochastic multi-echelon inventory systems: A survey. *Advances in Operations Research*, *2012*.

Simpson Jr, K. F. (1958). In-process inventories. *Operations Research*, *6*(6), 863–873.

Stadtler, H. (2005). Supply chain management and advanced planning—-basics, overview and challenges. *European journal of operational research*, *163*(3), 575–588.

Sterman, J. D. (1989). Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment. *Management science*, *35*(3), 321–339.

SURF. (2021). *Meet Snellius*. `https://visualization.surf.nl/snellius-virtual-tour/#/`

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Swanson, A. (2022). Biden administration releases plan for $50 billion investment in chips. *The New York Times*. `https://www.nytimes.com/2022/09/06/business/economy/biden-tech-chips.html`

Tarim, S. A., & Kingsman, B. G. (2004). The stochastic dynamic production/inventory lot-sizing problem with service-level constraints. *International Journal of Production Economics*, *88*(1), 105–119.

Tarim, S. A., & Kingsman, B. G. (2006). Modelling and computing (rn, sn) policies for inventory systems with non-stationary stochastic demand. *European Journal of Operational Research*, *174*(1), 581–599.

van Jaarsveld, W. (2021). Deep controlled learning of dynamic policies with an application to lost-sales inventory control. *arXiv preprint arXiv:2011.15122*.

Vanvuchelen, N., & Boute, R. N. (2022). The use of continuous action representations to scale deep reinforcement learning: An application to inventory control. *Available at SSRN 4253600*.

Vanvuchelen, N., Gijsbrechts, J., & Boute, R. (2020). Use of proximal policy optimization for the joint replenishment problem. *Computers in Industry*, *119*, 103239.

Voas, J., Kshetri, N., & DeFranco, J. F. (2021). Scarcity and global insecurity: The semiconductor shortage. *IT Professional*, *23*(5), 78–82.

Yan, Y., Chow, A. H., Ho, C. P., Kuo, Y.-H., Wu, Q., & Ying, C. (2022). Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, *162*, 102712.

Zhou, Z., Li, X., & Zare, R. N. (2017). Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, *3*(12), 1337–1344.

A Markov Decision Process Formulations

A.1 MDP for Sub-Decisions

Objective	$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t C(\mathbf{s}_t, \mathbf{X}_t^{\pi}) \vert \mathbf{s}_0\Big]$
State vector	$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \ldots, P_{t,n,l_n-1}, P_{t,n,l_n}]_{\forall n \in \mathcal{N}}, K]$
Allowed action set	$X_{t,n}$ s.t. $X_{t,n} \leq c$ $X_{t,n} \leq IL_{t,n+1,0}$ if $n \neq N$ $X_{t,n} \in \mathbb{N}_0$
Reward function	$C(\mathbf{s}_t, \mathbf{X}_t) = \sum_{n \in \mathcal{N} \setminus \{1\}} h_n(IL_{t,n} - X_{t,n-1}) + h_1(IL_{t,1})^+ - p(IL_{t,1})^-$
Random transition dynamics	$[\mathbf{s}_t] = f(\mathbf{s}_{t-1}, X_{t-1,0})$ s.t. $P_{t,n,i} = P_{t-1,n,i+1}$ $\quad \forall i \in \mathcal{L}_n, \forall n \in \mathcal{N}$ $P_{t,n,l_n} = 0$ $\quad \forall n \in \mathcal{N}$ $IL_{t,n} = IL_{t-1,n} + P_{t-1,n,1}$ $\quad \forall n \in \mathcal{N} \setminus \{1\}$ $IL_{t,1} = IL_{t-1,1} + P_{t-1,1,1} - d_t$ $K = N$
Deterministic transition dynamics	$[\mathbf{s}_t] = f(\mathbf{s}_t, X_{t,K})$ s.t. $P_{t,K,l_K} = X_{t,K}$ $IL_{t,K+1} = IL_{t,K+1} - X_{t,K}$ if $K \neq N$ $K = K - 1$

A.2 MDP for Exact Case with Stationary Demand

Objective

$$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t C(\mathbf{s}_t, \mathbf{X}_t^{\pi}) | \mathbf{s}_0\Big]$$

State vector

$$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \dots, P_{t,n,l_n-1}, P_{t,n,l_n}]_{\forall n \in \mathcal{N}}, K, L_t]$$

Allowed action set

$$\begin{aligned} & X_{t,n} \\ \text{s.t. } & X_{t,n} \leq c \\ & X_{t,n} \leq IL_{t,n+1,0} && \text{if } n \neq N \\ & IL_{t,n} + \sum_{i \in \mathcal{L}_n} P_{t,n,i} + X_{t,n} \leq i_n \\ & X_{t,n} \in \mathbb{N}_0 \end{aligned}$$

Reward function

$$\begin{aligned} & C(\mathbf{s}_t, \mathbf{X}_t) = \sum_{n \in \mathcal{N} \setminus \{1\}} [h_n \cdot (IL_{t,n} - X_{t,n-1})] + z_t \\ \text{s.t. } & z_t = \begin{cases} h_{1t,1} & \text{if demand satisfied} \\ p \cdot -IL_{t,1} & \text{if backlog} \\ v \cdot L_t + p \cdot b & \text{if backlog + lost sales} \end{cases} \end{aligned}$$

Random transition dynamics

$$\begin{aligned} & [\mathbf{s}_t] = f(\mathbf{s}_{t-1}, X_{t-1,0}) \\ \text{s.t. } & P_{t,n,i} = P_{t-1,n,i+1} && \forall i \in \mathcal{L}_n, \forall n \in \mathcal{N} \\ & P_{t,n,l_n} = 0 && \forall n \in \mathcal{N} \\ & IL_{t,n} = IL_{t-1,n} + P_{t-1,n,1} && \forall n \in \mathcal{N} \setminus \{1\} \\ & L_t = \begin{cases} -IL_{t-1,1} - P_{t-1,1,1} + d_t - b & \text{if lost sales } (IL_{t-1,1} + P_{t-1,1,1} - d_t \leq b) \\ 0 & \text{otherwise} \end{cases} \\ & IL_{t,1} = \begin{cases} b & \text{if lost sales } (IL_{t-1,1} + P_{t-1,1,1} - d_t \leq b) \\ IL_{t-1,1} + P_{t-1,1,1} - d_t & \text{otherwise} \end{cases} \\ & K = N \end{aligned}$$

Deterministic transition dynamics

$$\begin{aligned} & [\mathbf{s}_t] = f(\mathbf{s}_t, X_{t,K}) \\ \text{s.t. } & P_{t,K,l_K} = X_{t,K} \\ & IL_{t,K+1} = IL_{t,K+1} - X_{t,K} && \text{if } K \neq N \\ & K = K - 1 \end{aligned}$$

A.3 MDP with Demand Tree

Objective

$$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} C(\mathbf{s}_t, \mathbf{X}_t^{\pi})|\mathbf{s}_0\Big]$$

State vector

$$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \ldots, P_{t,n,l_n-1}]_{\forall n \in \mathcal{N}}, K, t, id_t]$$

Allowed action set

$$\begin{aligned} & X_{t,n} \\ \text{s.t. } & X_{t,n} \leq c \\ & X_{t,n} \leq IL_{t,n+1,0} && \text{if } n \neq N \\ & IL_{t,n} + \sum_{i \in \mathcal{L}_n} P_{t,n,i} + X_{t,n} \leq i_n \\ & X_{t,n} \in \mathbb{N}_0 \end{aligned}$$

Reward function

$$\begin{aligned} & C(\mathbf{s}_t, \mathbf{X}_t) = \sum_{n \in \mathcal{N} \setminus \{1\}} [h_n \cdot (IL_{t,n} - X_{t,n-1})] + z_t \\ \text{s.t. } & z_t = \begin{cases} h_{1t,1} & \text{if demand satisfied} \\ p \cdot -IL_{t,1} & \text{if backlog} \\ v \cdot L_t + p \cdot b & \text{if backlog + lost sales} \end{cases} \end{aligned}$$

Random transition dynamics

$$\begin{aligned} & [\mathbf{s}_t] = f(\mathbf{s}_{t-1}, X_{t-1,0}) \\ \text{s.t. } & P_{t,n,i} = P_{t-1,n,i+1} && \forall i \in \mathcal{L}_n, \forall n \in \mathcal{N} \\ & P_{t,n,l_n} = 0 && \forall n \in \mathcal{N} \\ & IL_{t,n} = IL_{t-1,n} + P_{t-1,n,1} && \forall n \in \mathcal{N} \setminus \{1\} \\ & IL_{t,1} = IL_{t-1,1} + P_{t-1,1,1} - a_{t,id_t} - k_t \\ & id_t = id_{t-1} + m_t \\ & t = t + 1 \\ & K = N \end{aligned}$$

Deterministic transition dynamics

$$\begin{aligned} & [\mathbf{s}_t] = f(\mathbf{s}_t, X_{t,K}) \\ \text{s.t. } & P_{t,K,l_K} = X_{t,K} \\ & IL_{t,K+1} = IL_{t,K+1} - X_{t,K} \quad \text{if } K \neq N \\ & K = K - 1 \end{aligned}$$

A.4 MDP for Exact Case with Demand Tree

Objective

$$\inf_{\pi} \mathbb{E}\Big[\sum_{t=0}^{T} C(\mathbf{s}_t, \mathbf{X}_t^{\pi})|\mathbf{s}_0\Big]$$

State vector

$$\mathbf{s}_t = [[IL_{t,n}, P_{t,n,1}, \ldots, P_{t,n,l_n-1}, P_{t,n,l_n}]_{\forall n \in \mathcal{N}}, K, L_t, t, id_t]$$

Allowed action set

$$\begin{aligned} & X_{t,n} \\ \text{s.t. } & X_{t,n} \leq c \\ & X_{t,n} \leq IL_{t,n+1,0} \qquad \text{if } n \neq N \\ & IL_{t,n} + \sum_{i \in \mathcal{L}_n} P_{t,n,i} + X_{t,n} \leq i_n \\ & X_{t,n} \in \mathbb{N}_0 \end{aligned}$$

Reward function

$$C(\mathbf{s}_t, \mathbf{X}_t) = \sum_{n \in \mathcal{N} \setminus \{1\}} [h_n \cdot (IL_{t,n} - X_{t,n-1})] + z_t$$

$$\text{s.t. } z_t = \begin{cases} h_{1t,1} & \text{if demand satisfied} \\ p \cdot -IL_{t,1} & \text{if backlog} \\ v \cdot L_t + p \cdot b & \text{if backlog + lost sales} \end{cases}$$

Random transition dynamics

$$\begin{aligned} & [\mathbf{s}_t] = f(\mathbf{s}_{t-1}, X_{t-1,0}) \\ \text{s.t. } & P_{t,n,i} = P_{t-1,n,i+1} && \forall i \in \mathcal{L}_n, \forall n \in \mathcal{N} \\ & P_{t,n,l_n} = 0 && \forall n \in \mathcal{N} \\ & IL_{t,n} = IL_{t-1,n} + P_{t-1,n,1} && \forall n \in \mathcal{N} \setminus \{1\} \\ & L_t = \begin{cases} -IL_{t-1,1} - P_{t-1,1,1} + a_{t,id_t} + k_t - b & \text{if lost sales } (IL_{t-1,1} + P_{t-1,1,1} - a_{t,id_t} - k_t \leq b) \\ 0 & \text{otherwise} \end{cases} \\ & IL_{t,1} = \begin{cases} b & \text{if lost sales } (IL_{t-1,1} + P_{t-1,1,1} - a_{t,id_t} - k_t \leq b) \\ IL_{t-1,1} + P_{t-1,1,1} - d_t & \text{otherwise} \end{cases} \\ & K = N \\ & id_t = id_{t-1} + m_t \\ & t = t + 1 \end{aligned}$$

Deterministic transition dynamics

$$\begin{aligned} & [\mathbf{s}_t] = f(\mathbf{s}_t, X_{t,K}) \\ \text{s.t. } & P_{t,K,l_K} = X_{t,K} \\ & IL_{t,K+1} = IL_{t,K+1} - X_{t,K} \quad \text{if } K \neq N \\ & K = K - 1 \end{aligned}$$

B Test Instances

B.1 Test Instances with Stationary Demand

Table B.1: Exact Instances with Stationary Demand

									Demand per period								
Scenario	N	h	p	l	c	b	v	i	0	1	2	3	4	5	6	μ	σ
E23-S1	2	$[1,4]$	16	$[2,3]$	4	12	1000	$[15,18]$	0.10	0.10	0.15	0.30	0.35	0.00	0.00	2.7	..
E23-S2	2	$[1,4]$	16	$[2,3]$	4	12	1000	$[15,18]$	0.10	0.15	0.15	0.20	0.20	0.15	0.05	2.9	..
E23-S3	2	$[1,4]$	16	$[2,3]$	4	12	1000	$[15,18]$	0.10	0.10	0.10	0.25	0.20	0.15	0.10	3.2	..
E23-S4	2	$[1,4]$	16	$[2,3]$	4	12	1000	$[15,18]$	0.05	0.10	0.10	0.20	0.25	0.20	0.10	3.5	..

Table B.2: Large Instances with Stationary Demand

						Demand per period								
Scenario	N	h	p	l	c	0	1	2	3	4	5	6	μ	σ
L223-S1	3	$[1,2,3]$	8	$[2,2,3]$	4	0.10	0.10	0.15	0.30	0.35	0.00	0.00	2.7	..
L223-S2	3	$[1,2,3]$	8	$[2,2,3]$	4	0.10	0.15	0.15	0.20	0.20	0.15	0.05	2.9	..
L223-S3	3	$[1,2,3]$	8	$[2,2,3]$	4	0.10	0.10	0.10	0.25	0.20	0.15	0.10	3.2	..
L223-S4	3	$[1,2,3]$	8	$[2,2,3]$	4	0.05	0.10	0.10	0.20	0.25	0.20	0.10	3.5	..
L1122-S1	4	$[1,2,3,4]$	8	$[1,1,2,2]$	4	0.10	0.10	0.15	0.30	0.35	0.00	0.00	2.7	..
L1122-S2	4	$[1,2,3,4]$	8	$[1,1,2,2]$	4	0.10	0.15	0.15	0.20	0.20	0.15	0.05	2.9	..
L1122-S3	4	$[1,2,3,4]$	8	$[1,1,2,2]$	4	0.10	0.10	0.10	0.25	0.20	0.15	0.10	3.2	..
L1122-S4	4	$[1,2,3,4]$	8	$[1,1,2,2]$	4	0.05	0.10	0.10	0.20	0.25	0.20	0.10	3.5	..
L2223-S1	4	$[1,2,3,4]$	8	$[2,2,2,3]$	4	0.10	0.10	0.15	0.30	0.35	0.00	0.00	2.7	..
L2223-S2	4	$[1,2,3,4]$	8	$[2,2,2,3]$	4	0.10	0.15	0.15	0.20	0.20	0.15	0.05	2.9	..
L2223-S3	4	$[1,2,3,4]$	8	$[2,2,2,3]$	4	0.10	0.10	0.10	0.25	0.20	0.15	0.10	3.2	..
L2223-S4	4	$[1,2,3,4]$	8	$[2,2,2,3]$	4	0.05	0.10	0.10	0.20	0.25	0.20	0.10	3.5	..

B.2 Test Instances with Non-Stationary Demand

Table B.3: Exact Instances with Non-Stationary Demand

Scenario	N	h	p	l	c	b	v	i	T	Noise	$P[m_t = 1]$
E11-NS10	2	$[1,4]$	16	$[1,1]$	4	12	1000	$[15,18]$	10	$U[-2,2]$	0.05
E12-NS10	2	$[1,4]$	16	$[1,2]$	4	12	1000	$[15,18]$	10	$U[-2,2]$	0.05
E21-NS10	2	$[1,4]$	16	$[2,1]$	4	12	1000	$[15,18]$	10	$U[-2,2]$	0.05

Table B.4: Large Instances with Non-Stationary demand

Scenario	N	h	p	l	c	T	Noise	$P[m_t = 1]$
L23-NS15	2	$[2, 3]$	8	$[2, 3]$	4	15	$U[-2, 2]$	0.05
L23-NS20	2	$[2, 3]$	8	$[2, 3]$	4	20	$U[-2, 2]$	0.05
L23-NS25	2	$[2, 3]$	8	$[2, 3]$	4	25	$U[-2, 2]$	0.05
L223-NS15	3	$[1, 2, 3]$	8	$[2, 2, 3]$	4	15	$U[-2, 2]$	0.05
L223-NS20	3	$[1, 2, 3]$	8	$[2, 2, 3]$	4	20	$U[-2, 2]$	0.05
L223-NS25	3	$[1, 2, 3]$	8	$[2, 2, 3]$	4	25	$U[-2, 2]$	0.05
L2223-NS15	4	$[1, 2, 3, 4]$	8	$[2, 2, 2, 3]$	4	15	$U[-2, 2]$	0.05
L2223-NS20	4	$[1, 2, 3, 4]$	8	$[2, 2, 2, 3]$	4	20	$U[-2, 2]$	0.05
L2223-NS25	4	$[1, 2, 3, 4]$	8	$[2, 2, 2, 3]$	4	25	$U[-2, 2]$	0.05

C Prescribed Actions

C.1 Prescribed Actions in Instance E23-S3

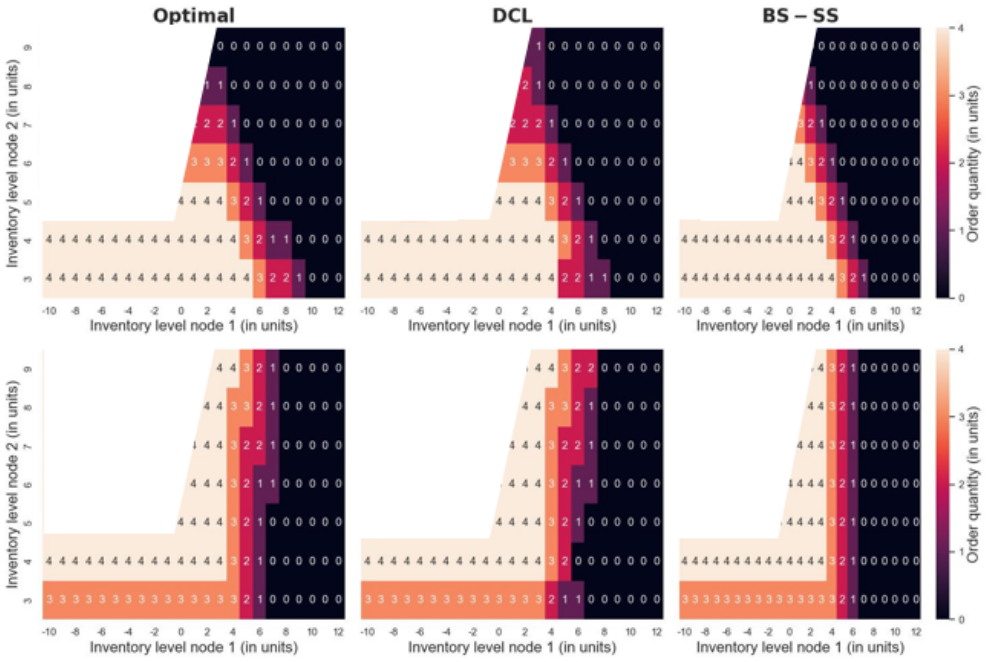


Figure C.1: Prescribed actions in instance E23-S3 with $P_t, n, i = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n$. The top row shows the actions for node 2 and bottom row the actions for node 1

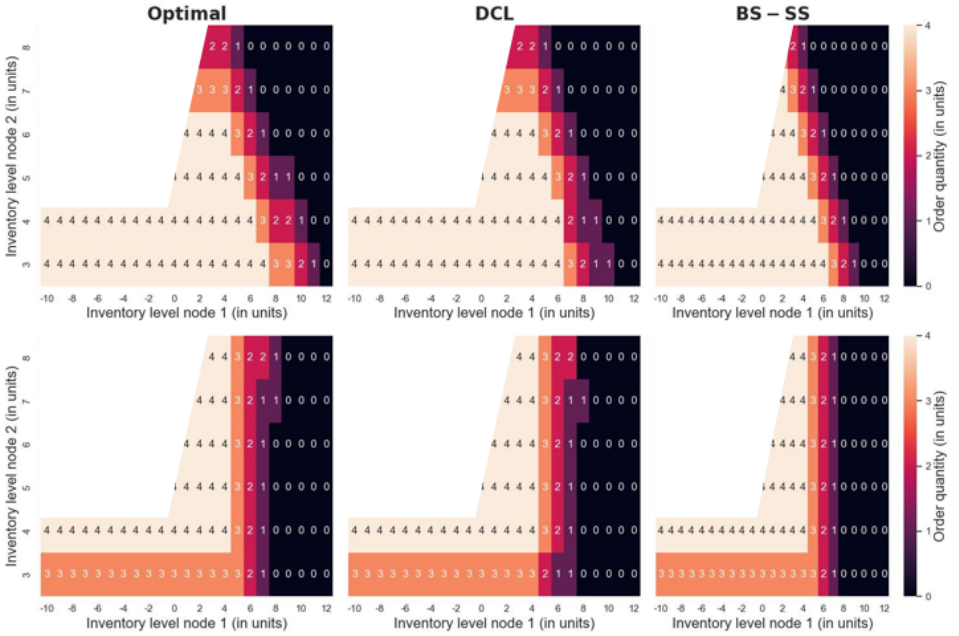


Figure C.2: Prescribed actions in instance E23-S3 with $P_t, n, 1 = 2$ and $P_{tni} = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n \setminus 1$

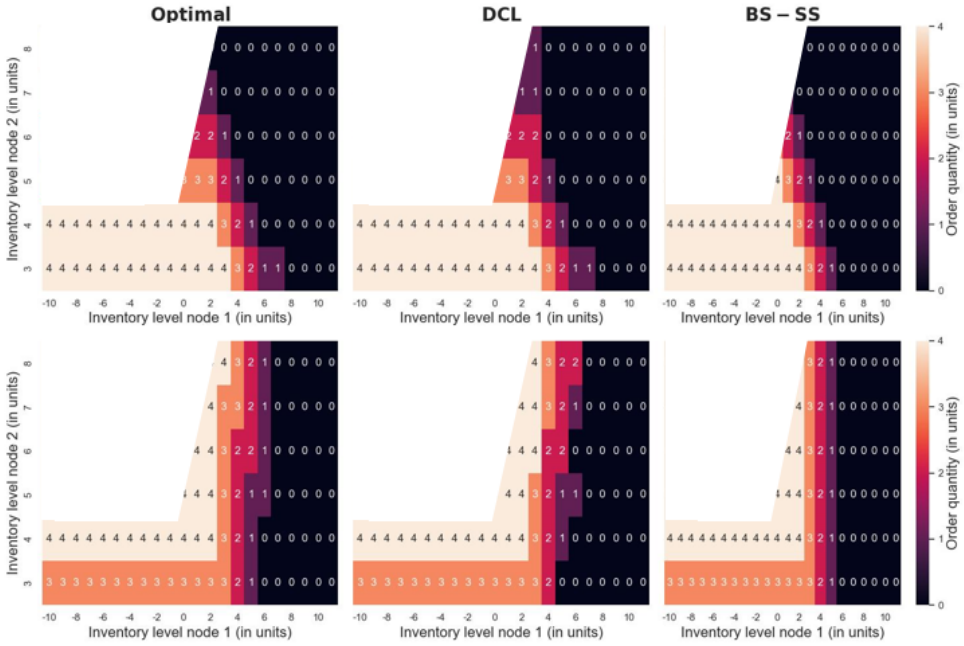


Figure C.3: Prescribed actions in instance E23-S3 with $P_t, n, 1 = 4$ and $P_{tni} = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n \setminus 1$

C.2 Prescribed Actions in Instance E21-NS10

Zero Vertical Movements

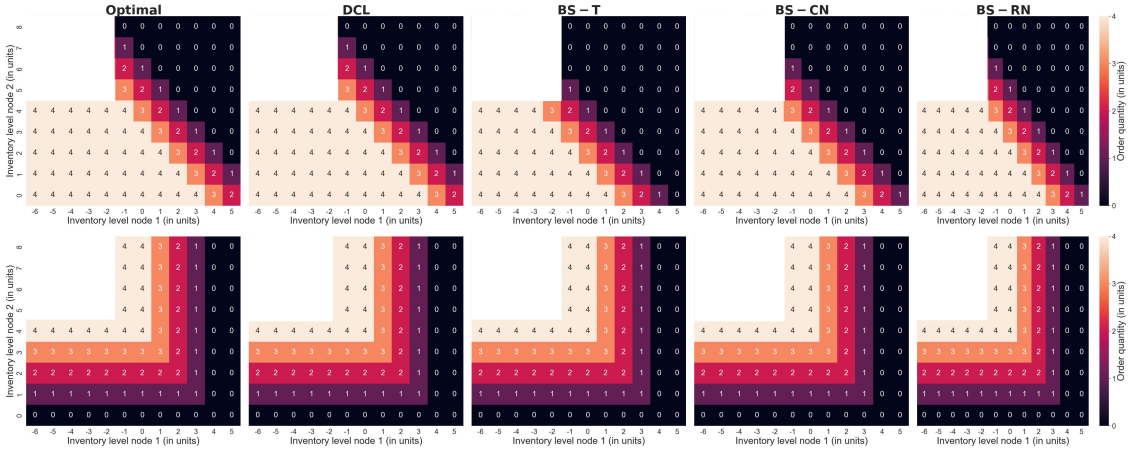


Figure C.4: Prescribed actions in instance E21-NS10 with $P_t, n, i = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n$ and $id_5 = 0$

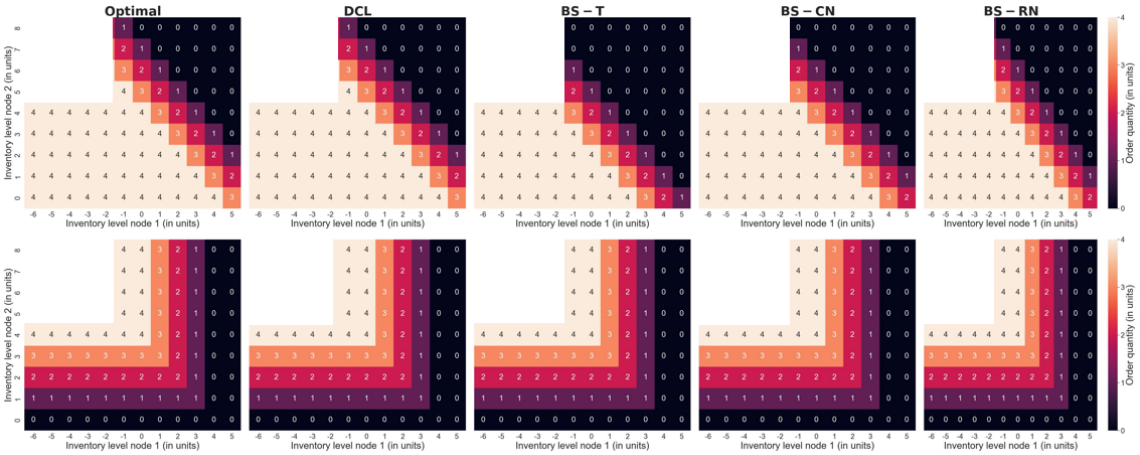


Figure C.5: Prescribed actions in instance E21-NS10 with $P_t, n, 1 = 2$, $P_{tni} = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n \setminus 1$, and $id_5 = 0$

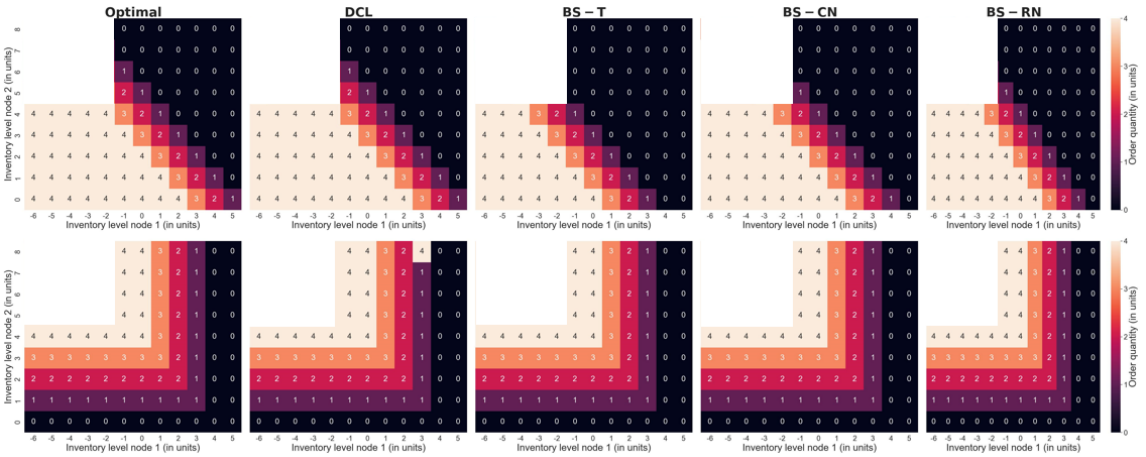


Figure C.6: Prescribed actions in instance E21-NS10 with $P_t, n, 1 = 4$, $P_{tni} = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n \setminus 1$, and $id_5 = 0$

One Vertical Movement

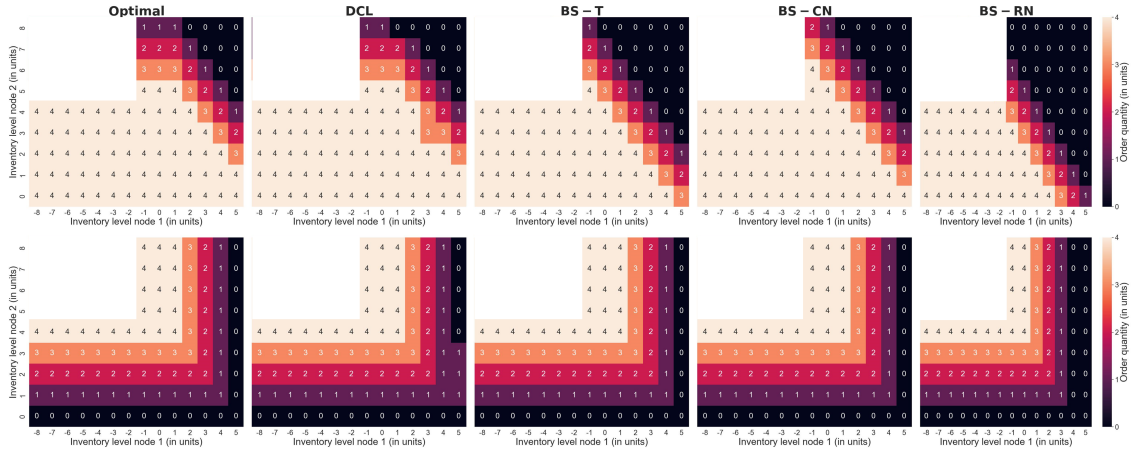


Figure C.7: Prescribed actions in instance E21-NS10 with $P_t, n, i = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n$ and $id_5 = 1$

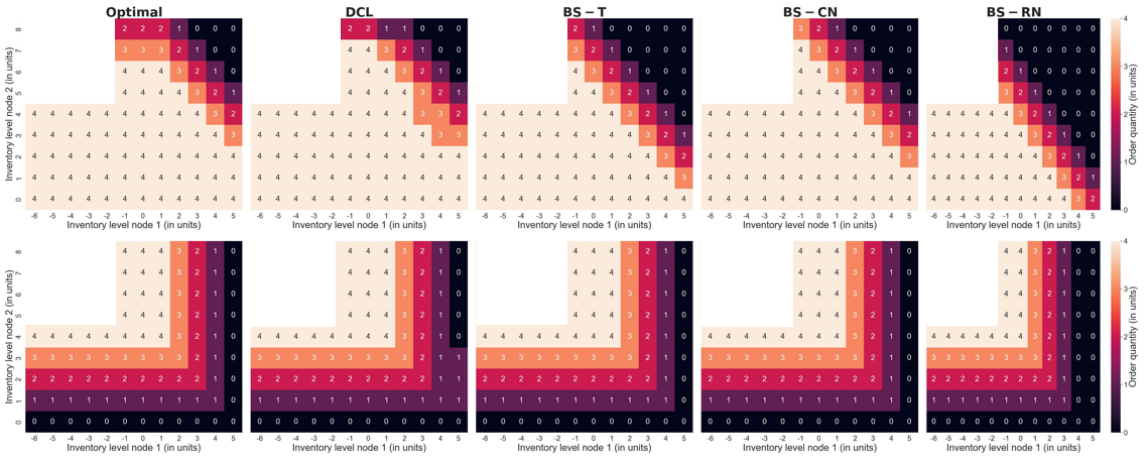


Figure C.8: Prescribed actions in instance E21-NS10 with $P_t, n, 1 = 2$, $P_{tni} = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n \setminus 1$, and $id_5 = 1$

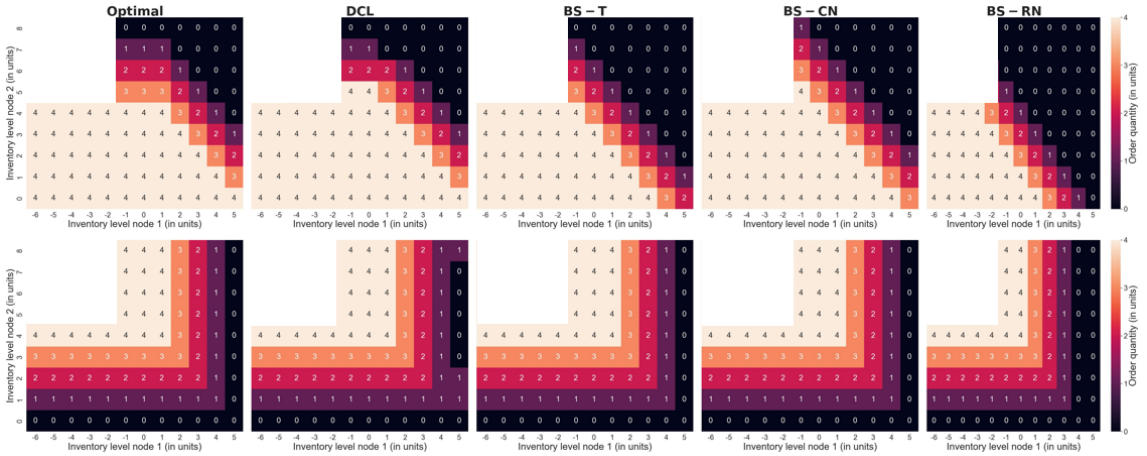


Figure C.9: Prescribed actions in instance E21-NS10 with $P_t, n, 1 = 4$, $P_{tni} = 3\ \forall n \in \mathcal{N}, \forall i \in \mathcal{L}_n \setminus 1$, and $id_5 = 1$