Gestion de stock sous contrainte de quantité minimale de commande multi-références

Gaetan Deletoille

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Rouen Normandie

Gestion de stock sous contrainte de quantité minimale de commande multi-références

Présentée et soutenue par
GAETAN DELETOILLE

Thèse soutenue le 23/06/2022 devant le jury composé de		
M. JEAN-PHILIPPE GAYON	PROFESSEUR DES UNIVERSITES, UNIVERSITE CLERMONT AUVERGNE CLERMONT AU	Rapporteur du jury
MME SAFIA KEDAD-SIDHOUM	PROFESSEUR DES UNIVERSITES, CNAM (Paris)	Rapporteur du jury
M. MAXIME BERAR	MAITRE DE CONFERENCES, Université de Rouen Normandie	Membre du jury
M. ARNAUD KNIPPEL	MAITRE DE CONFERENCES, INSA DE ROUEN NORMANDIE	Membre du jury
M. VINCENT T'KINDT	PROFESSEUR DES UNIVERSITES, UNIVERSITE DE TOURS	Membre du jury
M. SÉBASTIEN ADAM	PROFESSEUR DES UNIVERSITES, Université de Rouen Normandie	Directeur de thèse

Thèse dirigée par SÉBASTIEN ADAM (Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes),

UNIVERSITÉ DE ROUEN NORMANDIE

MIIS

litis

Contents

Introduction

Context

In our globalized world, supply chains are at the core of the world's trade and economy. The ability to deliver and store supply efficiently is critical to many business's profitability [66]. Supply chain management can become so complex and intricate that companies may not have the knowledge or resources to properly optimize their processes by themselves. These businesses can delegate their inventory decisions to supply chain experts such as Lokad. Lokad initiated and funded this Phd, with the support of the ANRT through a CIFRE contract. Lokad is a company that provides inventory management as a service. Its clients send their relevant data (history of sales, stock on hand and on order, purchase and sell costs, estimated holding costs, etc), which Lokad uses to compute the best possible inventory control decision suggestions. Lokad deals with a large variety of businesses, each with their own supply chain challenges and constraints.

Inventory control is defined as the process of ensuring that the right amount of supply is available within a business. As such, the computation of inventory control decisions can be divided in two steps. The first step is to forecast the future need for supplies, that we call the demand. The second step is to determine the optimal inventory management decisions depending on this forecast, to ensure that the supply will be available where and when it will be needed. One of the specificity of Lokad's service is that its forecasting and optimization tools embrace the stochasticity of the demand. Sales in retail are most often the outcome of a multitude of unknowable causes, and are therefore impossible to forecast with certainty. One can account for this uncertainty by considering probabilistic forecasts, which associate a probability to every possible scenario. There exists many demand models and forecasting techniques [5] (some of which are implemented at Lokad), but the goal of this PhD is not to study or improve these techniques.

This thesis focuses on the second step, the optimization of the inventory control policies, assuming that the forecasts are considered as inputs of these policies.

Stochastic optimization

Forecast accuracy is crucial to ensure the good performance of these policies. It is however also important to note that using a probabilistic model for the demand leads to better (yet more complex) inventory control policies [5]. Indeed, if one trusts a naïve deterministic forecast, one only needs to maintain in stock the exact amount of supply that is predicted. Forecasting errors generated by the uncertainty of the demand can then prove very costly, since a greater demand than expected leads to a stock-out. It is therefore necessary to have a certain amount of safety stock available to avoid such scenarios. On the other hand, safety comes at a cost: unsold inventory leads to unnecessary holding costs. The right balance can be achieved by using probabilistic forecasts instead of deterministic forecasts. Businesses should estimate their objective function, that takes into account the probability of the different scenarios, as well as their associated costs: the costs and rewards of purchasing, selling, and holding supply, but also less tangible costs such as customer satisfaction. The optimization process then aims at maximizing the expectation of the objective function, over all the possible outcomes of the stochastic demand. The decisions are therefore

optimized considering all the possible futures, and not only the average scenario. The right balance between safety stocks (with the associated holding costs) and the probability of stock-out (associated with shortage costs) emerges naturally from the maximization of a well designed objective function [5].

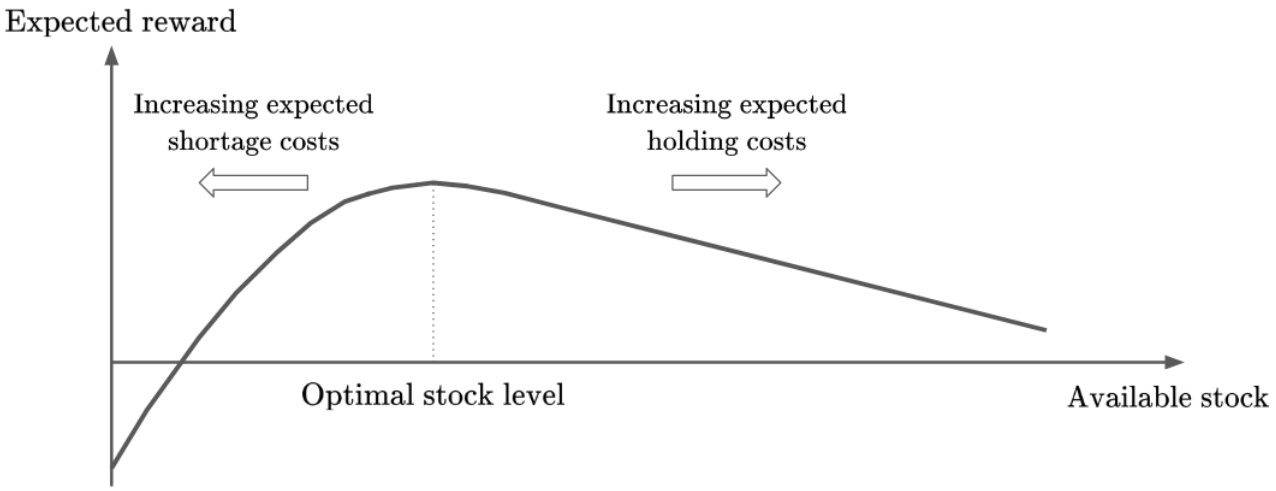


Figure 1: Optimal balance between shortage costs and holding costs

The figure 1 illustrates how an objective function can help determine the ideal safety stock. It displays an example of what an objective function can look like in inventory control problems. The objective function is the expected reward, and it is drawn depending on the available stock at a given time, before the stochastic demand starts consuming the supply. The objective function at first increases with the available stock: when the stock level is close to zero, the retailer can expect to sell all its supply. As the stock level increases, the probability of holding unsold inventory increases, and the probability of shortage decreases. The expected reward reaches a maximum, which defines the ideal safety stock. Beyond the optimal stock level, the expected holding costs keep increasing, and the expected reward decreases.

The Minimum Order Quantity (MOQ) constraint

Manufacturers and suppliers can often leverage economies of scale to reduce their manufacturing and transportation costs, since it dilutes the impact of fixed costs. Many actors in supply chains choose to shift the burden of these fixed costs to their purchasing entities, by introducing either fixed set-up (or delivery) fees [5, 107], bulk prizes [73], or mandatory minimum order size [31, 73].

Lokad's clients frequently face in their supply chains this latter constraint, also called the Minimum Order Quantity (MOQ): they must place replenishment orders that reach the minimum requirement imposed by their supplier. A real world example is analyzed in Musalem and Dekker [73]. Different items often share manufacturing or delivery constraints that lead suppliers to impose multi-item minimum order quantity constraints [114]. It means that the constraint is set on the total of the order. These constraints can be set on various economic or physical quantities related to the items such as the total value, volume, or even weight of the order. These type of multi-item constraints are harder to deal with, since they require coordinated replenishment to reduce costs [5].

We define the Minimum Order Quantity (MOQ) problem as the inventory control problem that the purchasing entity (the retailer) faces in this situation. The retailer holds an inventory of several items. As in any inventory control systems, at every time step an external stochastic demand consumes supply held in stock, generating a reward. At the beginning of every time step, the

retailer can replenish its inventory by placing orders to a single supplier. This supplier imposes the minimum order quantity constraint, that is shared across all items. The retailer needs to find the right balance of inventory stock levels, and time its replenishment orders adequately to maximize its expected reward. The MOQ problem is categorized as problem of decision making under uncertainty. It is a multi period problem, since the retailer can potentially place new orders at every time steps. The retailer can adapt its decisions depending on the outcome of the stochastic demand: this ability to adapt is crucial to maximize the reward, it but also makes the problem very complex. Indeed, it prevents the retailer from planning in advance all its future decisions.

The single item version of the MOQ problem is illustrated with figure 2, that displays the evolution of the inventory of an item. The retailer has the possibility to place replenishment orders periodically, and these replenishment order size must be greater or equal to Q. The present time is the timestep t: the uncertainty of the future demand means that the stock level evolution is also uncertain. The retailer has access to a probabilistic forecast of the demand: it is visually represented by ranges of decreasingly probable outcomes.

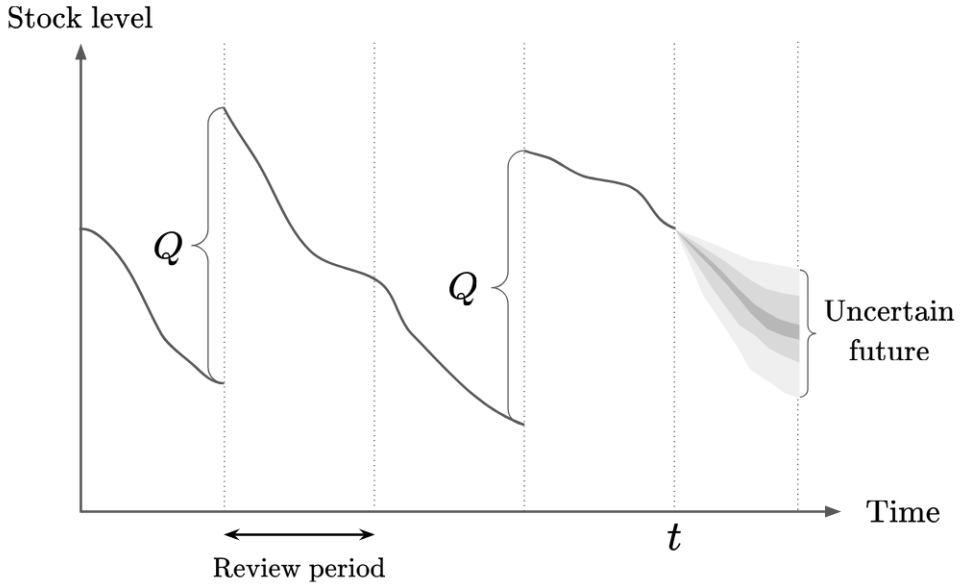


Figure 2: Evolution of a single item MOQ system subjected to stochastic demand

A two item version of the MOQ problem is illustrated with figure 3. The supplier imposes that the sum of the item 1 and item 2 ordered quantities are at least equal to Q. The figure displays an example of stock evolution for both items, and for the aggregate stock. The multi-item MOQ problem is significantly more complex than the single item MOQ problem: not only does the retailer need to determine when an order should be placed, but he also needs to decide what items should be prioritized. Having a large aggregate stock level is not sufficient to ensure that all the items are sufficiently stocked.

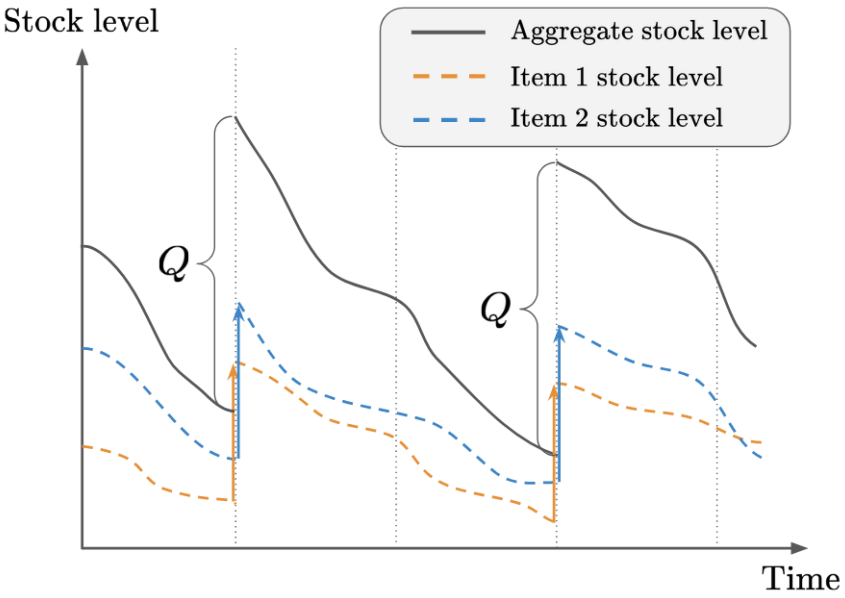


Figure 3: Evolution of a two-items MOQ system

While the existence of the MOQ problem was acknowledged in the inventory control literature, a surprisingly low number of publications have been attempting to provide practical solutions [61, 73, 112, 115], especially on the multi-item instance of the problem [76, 114]. To the best of our knowledge, there does not exist a satisfying solution to the multi-item MOQ problem subjected to non stationary demand. Lokad's clients frequently face this type of constraints: this Phd was therefore funded by Lokad to further understand this problem and find practical solutions for its clients. The title of this thesis is:

The multi-item Minimum Order Quantity inventory control problem

Contributions

The first contribution of this thesis is a mathematical and experimental analysis of the MOQ problem. We numerically estimate the scope of applicability of the backward recursion algorithm, a classical dynamic programming algorithm. We conclude that this algorithm can provide the optimal solution on some small scale instances of the multi-item version of the problem. We utilize this result to analyze the characteristics of the optimal policy on several toy examples. We notably study the optimal order size, the impact of a policy over time, and the dependency between the items. We finally consider the simplified single period (but multi-item) version of the problem: we develop an efficient solution based on the idea of atomizing the decisions into single unit increments of order, and compute their associated contribution to the reward. This analysis lays the foundations upon which the next two contributions are built.

The second and main contribution of this thesis is the w-policy, a heuristic policy that approximately solves the multi-item multi-period MOQ problem. The originality of the w-policy is that these computations are performed in a tractable way, leveraging several key assumptions that are based on the aforementioned analysis of the MOQ problem. We do not detail in this introduction the technical formulations of these assumptions, but the idea behind them is that the MOQ constraint can be seen as a constraint that forces the items to be reordered less frequently and simultaneously. While this induces a dependency between the items that complicates the problem, this dependency can be simplified if future reorder timings are known. By making a reasonable assumption on these future reorder timings, it is possible to compute very efficiently the best order allotment (the ordered quantities of each item) if the retailer was to place an order. Once the

allotment of the potential order is known, it becomes simpler to determine whether the expected reward of placing this order exceeds the expected reward of delaying it. This two step decision process is the w-policy. Our contribution includes a detailed explanation of the w-policy and the mathematical justifications upon which it is based. We also provide a study of its performance and its scope of applicability both on small scale toy problems and on two large scale real data sets (obtained from a Lokad's client). The w-policy exhibits near optimal performance on most MOQ problem instances, while being able to scale to instances with thousands of items, which was (to the best of our knowledge) never achieved before. The computation of a decision by the w-policy on a dataset dealing with a MOQ constraint over 11 607 items took a minute on average. The w-policy is however built upon several simplifying assumptions, which in theory reduce its scope of applicability. Our contribution includes an analysis of these limitations.

The third contribution of this thesis is our exploratory work on the applicability of reinforcement learning approaches to the MOQ problem, notably to overcome the aforementioned limitations of the w-policy. Reinforcement learning is a subset of machine learning, that attempts to learn how to behave in an environment to maximize a reward, utilizing experience gathered while exploring the environment. Reinforcement learning is a good candidate to solve the MOQ problem: under the hypothesis that the probabilistic forecasts are accurate, one can run as many simulations of the system as necessary by randomly generating the demand at each period (following the probability distributions of the forecasts). It is therefore possible for a learning agent to interact with the simulated system as many times as necessary for it to converge toward a satisfying solution. Firstly we provide an analysis of the feasability and limitations of state of the art reinforcement learning architectures to solve the MOQ problem. We then suggest a new method to overcome these limitations. This new method mixes reinforcement learning ideas and the w-policy to leverage the best of both approaches. We call it the hybrid policy. We provide guidelines to implement it in practice, and to make it scale to very large datasets. We compare its performance on the same toy problems and data sets as the w-policy, and demonstrate its ability to scale on very large instances of the problem (up to ten thousand items). We however conclude that while it is capable of dealing with the MOQ problem under a wider variety of settings than the w-policy, the hybrid policy remains hindered by the instability associated to deep reinforcement algorithm, and by the impossibility to fully explain and justify the chosen decision.

The final achievement of this Phd is that the w-policy was successfully implemented and integrated into Lokad's software, and it is solving MOQ problems for clients on a day to day basis. The safer approach of the w-policy was preferred to the hybrid policy by Lokad, since using a not-so-stable black-box component to suggest critical inventory control decisions proved too much of a risk to be used. The w-policy performance and scope of applicability proved largely sufficient for the clients needs. This Phd therefore delivered significant advances both to the academic inventory control literature and to Lokad's practical solutions.

Plan

Chapter I

This thesis is separated in four chapters. In chapter I, we provide an extensive overview of the inventory control literature. The inventory control literature is massive and diverse: there exists no methodological approach yet that has proven to be superior to tackle all the possible inventory control problems. Instead, a few 'off the shelf' methods have been developed, each having their own domain of applicability and limitations. It is also frequent to face problems that no existing approach can directly solve. This forces practitioners to develop and utilize specialized heuristics for their problem. These heuristics are often variations of the aforementioned 'off the shelf' methods, or belong to well known types of approaches. In chapter I, we first present a few important

inventory control concepts and definitions. We then detail the few main 'off the shelf' approaches, including classical reorder point methods, linear programming, stochastic dynamic programming, and reinforcement learning. Since the MOQ problem can be defined as stochastic, multi-stage, non stationary, multi-item problem, we analyze why these off-the-shelf solutions fail to solve the multi-item MOQ problem. Specialized heuristics therefore had to be developed: we then look into the rather scarce literature dedicated specifically to the MOQ problem. We detail the existing methods, and exhibit their shortcomings in terms of scope of applicability. Notably, the multi-item instances of the problem prove to be particularly difficult: no solution appears to be able to deal with the non stationary, multi-item instance of the MOQ problem.

Chapter II

Chapter II presents the first contribution of this thesis, a mathematical and experimental analysis of the MOQ problem. In section 1, we describe the model of the MOQ problem that we attempt to solve in this manuscript. We detail the equations that describe the evolution of the system, the equations for the costs and rewards, and the formulation of the objective function. We analyze the state and the action space (the set of possible situations and possible orders) and conclude that their size is exponential with the number of items affected by the MOQ. We also provide a notations summary. In section 2, we study the applicability of the backward recursion (a standard state of the art stochastic dynamic programming technique) to the MOQ problem. The backward recursion being very sensitive to the size of the state and action space, we conclude that this method of optimal control is only applicable to some small scale instances of the problem. While it is not sufficient for practical applications, having access to the optimal policy is very valuable to better understand the MOQ problem and evaluate the performance of sub-optimal policies. In section 3, we analyze how the MOQ impacts the overall optimal order size. We conclude that unless the demand is deterministic or large enough compared to Q (the minimum order quantity), assuming that the optimal order size is equal to Q is a reasonable approximation. We illustrate this conclusion with a few small scale experiments, conducted utilizing the backward recursion. In section 4, we investigate the impact a decision has on the system over time. We identify that the impact of a sub optimal decision can be mitigated over time by subsequent orders, which means that the time window between the reception of an order and the reception of the subsequent order is particularly important. We name this time window the 'inaction window'. We experimentally determine some characteristics of this inaction window. In section 5, we illustrate with concrete examples (always using the backward recursion) how items subjected to the same MOQ constraint impact each other. Notably, we plot the contribution of some items replenishment order to the expected reward depending on the other items state and replenishment orders. Finally, the section 6 presents an efficient solution to the simpler single period version of the MOQ problem. This solution is based on the idea of atomizing the decisions into single unit increments of order, and compute their associated contribution to the reward. The best possible order is then determined directly by ranking the increment of orders contributions to the reward. This approach of quantifying the impact on the reward of order increments is re-utilized in the w-policy.

Chapter III

Chapter III presents the w-policy, a heuristic designed to approximately solve the multi-period multi-item MOQ problem. From sections 1 to 5, we detail the ideas and assumptions behind the w-policy, and how to compute it in practice. An overview was already provided in the 'Contributions' section of this introduction. In section 6, we suggest a simple heuristic to estimate the value of w, which is crucial to the w-policy. Section 7 details a numerical trick to accelerate the computation of the w-policy in practice. Section 8 presents the results of the w-policy on small scale toy experiments.

The advantage of dealing with small scale experiments is that the optimal policy can be computed on such instances. A wide variety of settings is studied to exhibit the performance and limitations of the w-policy compared to the other state of the art methods, and to the optimal policy. The impact of all the simplifying assumptions is illustrated with numerical experiments. In section 9, we apply the w-policy to two large scale datasets. The first dataset considers a retailer dealing with 134 items, subjected to a stationary demand. In these settings, the (S, T)-policy [114] was the only state of the art method that was still computable, and able to compete with the w-policy. We then apply the w-policy to a significantly more complex dataset, that deals with 11 607 items with seasonal demand forecasts. The w-policy is to our knowledge the only solution capable of scaling to such instances of the MOQ problem.

Chapter IV

Chapter IV presents our exploratory work on the feasibility of reinforcement learning approaches for the MOQ problem. In section 1 we review the reinforcement learning literature and discuss how state of the art methods can be applied to the MOQ problem. We conclude that the size and the characteristics of the state and action spaces prevent the usage of most standard approaches. The only architectures capable of dealing with this problem in theory prove very difficult to implement in practice, due to the instability of the training phase. We identified the reasons behind these instabilities, which naturally led us to develop a new method capable of overcoming these shortcomings. In section 2, we propose a new algorithm (that we call the hybrid policy), that combines the very efficient allotment algorithm of the w-policy and a neural network that is trained to evaluate whether the order should be delayed or not. This hybrid architecture is more stable, but requires the computation of the allotment algorithm every time one generates a new training example. In section 3, we indicate how to optimize the training example generation, so that the neural network has enough data to learn. In section 4, we discuss and specify the configuration of the neural network utilized in the hybrid policy. It includes the choices of layer types, activation functions, depth and width of the network, etc. Finally in section 5 we present the results obtained on several experiments. We first consider small scale MOQ problems. We show that while the hybrid policy is capable of reaching near optimal performance in the right settings, its training remains too unstable to be utilized with confidence in a production environment. It demonstrates its ability to overcome some limitations of the w-policy, thank to the fact that it is based on a different set of assumptions. We also were able to scale the hybrid policy to the large instances of the MOQ problem, and applied it to the two aforementioned datasets (134 and 11 607 items). In these practical settings, the performance of the hybrid policy were competitive with the w-policy but did not perform better.

Conclusion

We summarize these results in the conclusion of this thesis. We recapitulate the advantages and limitations of the two methods we developed. We also provide a summary of the contributions that we believe to be the more useful in practice for supply chain practitioners. We will finally discuss how this thesis opens new exciting perspectives for future work related to this subject.

Chapter I

Inventory control state of the art

Introduction

The goal of this thesis is to propose solutions to solve the inventory control problem called the MOQ problem. The inventory control literature is massive and diverse: there exists no methodological approach yet that has proven to be superior to tackle all the possible inventory control problems. Instead, a few 'off the shelf' methods have been developed, each having their own domain of applicability and limitations. It is also frequent to face problems that no existing approach can directly solve. This forces practitioners to develop and utilize specialized heuristics for their problem. These heuristics are often variations of the aforementioned 'off the shelf' methods, or belong to well known types of approaches. A review is provided in [55].

We first present in section I.1 important inventory control concepts and definitions. We then detail the few main possible approaches to inventory control in sections I.2 to I.6. Since the MOQ problem can be defined as a stochastic, multi-stage, non stationary, multi-item problem, we analyze the applicability of these methods to problems with such characteristics. We also provide concrete examples of application of these methods to simple inventory control problems, in order to illustrate both their respective advantages and limitations.

We recapitulate the different scope of applicability of these methods in section I.7, and conclude that 'off the shelf' approaches are currently unable to directly solve the MOQ problem. This forced operational researchers to develop specialized policies and heuristics. We look into this -rather limited- dedicated literature in section I.8. While optimal or near optimal policies can be obtained on the single item version of the MOQ problem, our analysis of the literature concludes that there exists no satisfying solution to the non stationary multi-item MOQ problem yet.

I.1 Inventory control concepts and definitions

Supply chain practitioners face a very large variety of inventory control problems in the real world. In order to reflect this reality, many different models have been developed. It is crucial to understand all the underlying assumptions behind a model choice, since the end goal is always the concrete performance of the policies in the real world. The closer the problem definition is to an accurate reflection of the reality, the more confidence one can have on the corresponding policies. We present in this section a few important concepts that can differentiate supply chain models.

I.1.1 Single or multi-echelon

The main characteristic of an inventory control problem is whether it considers a single or multiple locations for its inventory flow. If the flow of goods only transfers through one location, the system

is single echelon. If inbound flows between locations is not negligible, the problem is multi-echelon. Single echelon problems are the more frequent, as actors often only deal with one level of the supply chain. Even for businesses dealing with end to end supply chains, it is possible to perform a local optimization at every echelon of the supply chain independently instead of optimizing the chain as a whole. On the other hand, multi-echelon supply chain optimizations centralize and leverage the data from all levels, so they are theoretically capable of further improvements compared to a local optimization approach [96]. They are for example capable of dealing with what is called the bullwhip effect. The bullwhip effect is a phenomenon that occurs in supply chains when small variations of demand at the lower supply chain levels generate much larger variations at higher levels. This phenomenon was first mentioned by Forrester [32], and formalized and popularized by Lee [67].

I.1.2 Continuous or periodic review

An inventory can be reviewed continuously or periodically. If the review is continuous, a new replenishment order can be placed anytime. Implementing a continuous review in practice requires to re-evaluate whether actions should be taken either at every instant, or every time a unit in stock is consumed by the demand. Re-evaluations of the situation always have associated costs and time constraints in practice, even if they can appear minimal. A continuous review model is therefore relevant to represent systems with a sparse enough demand, so that these costs remain negligible [5].

Business instead frequently apply a periodic review approach. New replenishment orders can only be placed at specific timings, at a fixed frequency. This ensures that the costs of the inventory control policy remain bounded. Periodic review systems are therefore more adapted to high demand systems. It can also be more practical for planning and organization, as the business processes can be scheduled accordingly. The duration of the period is adapted to the nature of the business.

The total temporal horizon considered also varies from one model to another. While infinite duration scenarios are not realistic, this assumption is often made as it removes the need to define an arbitrary finite temporal horizon.

I.1.3 Lead time

The lead time is the duration between the placement and the reception of a replenishment order. Non negligible lead times can significantly increase both the complexity of inventory control problems and the performance of a supply chain [25].

In the real world, lead times are often stochastic, since delays in the deliveries or manufacturing processes are common [46]. Business should therefore assess their confidence in their supplier's ability to deliver replenishment orders reliably when considering their inventory control model. Forecasting the leadtimes in addition to the demand better represents the reality of supply chains. However, while including stochastic lead times in a model may yield more accurate results, the overhead in complexity is not always worth it and it is common practice to assume deterministic lead times [5].

I.1.4 Holding costs

The holding costs are a critical component of the cost function of an inventory control model. They are defined as the direct and indirect costs associated to holding an inventory. They include the costs of managing warehouses for storage, but also the costs of deprecation of the stored goods, and the opportunity cost of spending a budget that could be used for potentially more profitable investments.

The most common assumption is that the holding costs are linear with the number of units held in stock. For a periodic review inventory system, this means that the holding costs over one review period can be expressed as :

$$\text{holding costs} = h\, s \tag{I.1}$$

s is the stock on hand remaining at the end of the period, and h is a constant called the holding cost parameter. The holding cost parameter is frequently expressed as a fraction of product value over a given period of time [6]. This holding parameter should be set depending on the items potential deprecation, the opportunity costs, and the direct storage costs. Detailed methods to derive this holding cost parameters are presented in [6], and applied to ten companies.

I.1.5 Shortage costs and lost sales

Shortage costs

One of the primary objective of inventory control is to reduce shortages (also called stock outs) to reasonable levels. Shortages can have a very different financial consequences depending on the type of business and product. The costs associated to a shortage are rarely limited to a missed opportunity cost. The impact shortages have on customers is hard to evaluate, but may lead to significant further indirect costs for businesses. The behavior of customers facing out of stocks in retail has been studied in [39, 103]. Their behavior is complex: some customers would simply substitute the item they were looking for for another similar one, some would delay their purchase, others would go to a different store or not buy the item at all. Another case study [75] found that for a manufacturer, the shortage cost are linked to the items gross profit. The shortage costs are therefore highly dependent on the businesses.

Lost sales and back orders models

In practice inventory control models usually model shortages in two primary ways. The lost sales hypothesis considers that when a shortage occurs, the demand is lost and cannot be recovered. In this case, the shortage represents a missed opportunity of a positive reward. In that sense, it can be considered as a loss, equal to the opposite of the reward that would have been generated if the inventory was sufficiently stocked. An additional stock-out penalty can be considered, linear to the number of lost sales, that models the additional costs linked to the impact of the stock out on the customer satisfaction. As an example, for a periodic review system, one may model the shortage costs for a period as:

$$\text{shortage cost} = m\,[d - s]^+ \tag{I.2}$$

m is the per unit shortage cost, that includes both the missed opportunity of reward and the additional stock-out penalty. s is the stock level at the start of the period, and d is the demand during the period.

The second possible hypothesis is that sales are not lost when a shortage occurs, but are instead backlogged. A back order penalty is applied, proportional to the number of units backlogged. When a replenishment occurs, the first units are assigned to cover the backlogged demand. This hypothesis can be well suited for industrial environments, but rarely so for retail businesses [13]. The difference between the two models is illustrated in figure I.1.

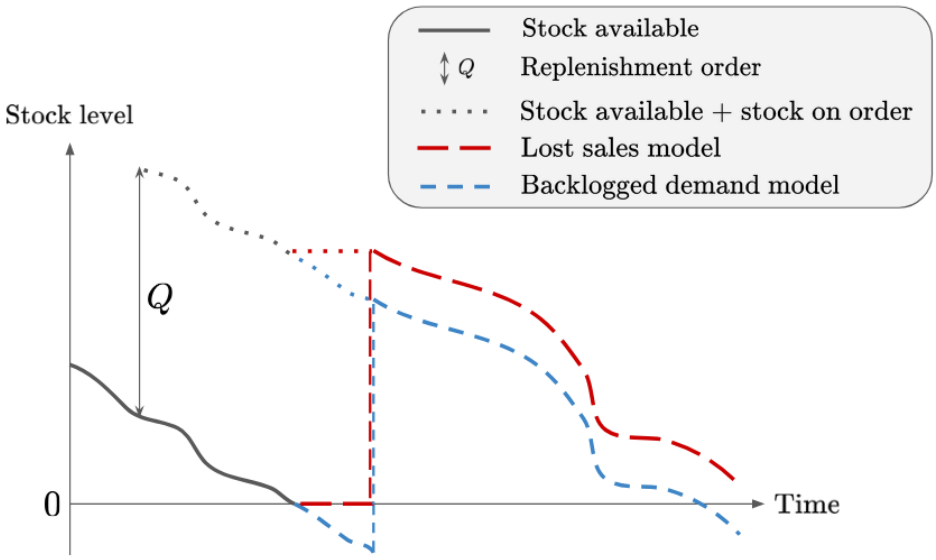


Figure I.1: Lost sales model and backlogged demand model

It is to be noted that systems with backorders are easier to analyze than systems with lost sales. Indeed, the fact that all the demand is eventually met with available stock or stock on hand means that the stock can conceptually go into the negative values. Therefore the stock available at any given time is not impacted by past shortages. On the contrary, under the lost sales hypothesis shortages have an impact on the end stock level. Figure I.2 provides a visualization of this fact. We consider two scenarios: in the first scenario, the replenishment order is delivered on time, and no shortage occurs. In the second scenario, the replenishment order is delayed, causing a shortage. If the demand was backlogged the available stock after reception of the order would be the same both scenarios. On the contrary under the lost sales hypothesis, the available stocks are not the same between the two scenarios after the order is received. Therefore the lost sales hypothesis makes the system more complex to analyze, but is necessary to adequately represent the reality of many real world supply chains [13].

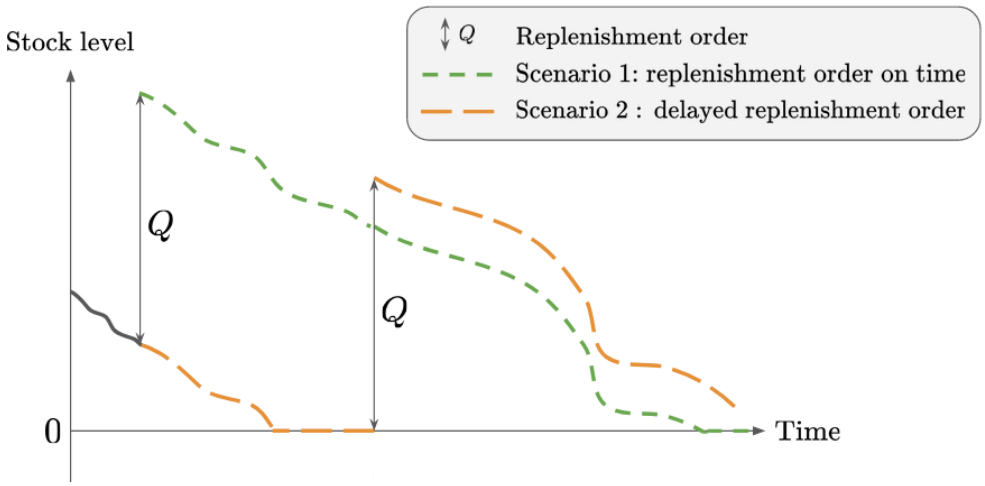


Figure I.2: Lost sales model

Conclusion

This section introduced a few important inventory control problems concepts. There exists no methodological approach yet that has proven to be superior to tackle all the possible inventory control problems. Instead, a few main approaches have been developed, each having their own domain of applicability and limitations. They are presented in the following sections.

I.2 Classical inventory control approaches

We first consider the popular EOQ and reorder point methods, which can be categorized as 'classical' inventory control approaches.

I.2.1 The EOQ formula

The classical Economic Order Quantity (EOQ) formula is one of the first and most well known result in inventory control. It was derived in 1913 by Harris [43], and studied in-depth in [107]. The goal of this formula is to find the order size Q to place when the stock reaches zero. It should find the optimal trade off between order set up costs and holding costs. It relies on several assumptions:

- The model is a continuous review system.
- The demand D is stationary, continuous and deterministic. In this model it is the annual demand, but it can be defined over any temporal horizon.
- The setup cost is fixed and noted K.
- The annual holding cost parameter is known and noted h.
- The lead time is equal to zero (instantaneous delivery).
- There is no shortages.

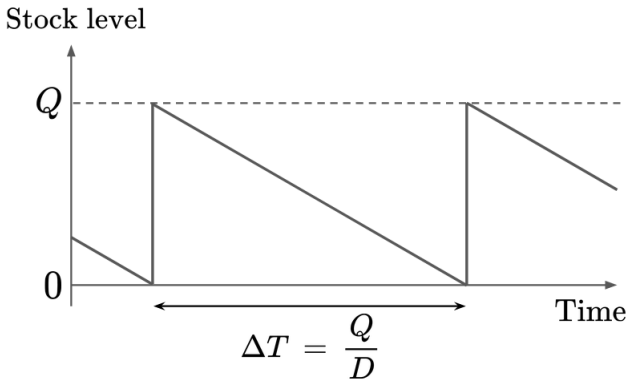


Figure I.3: Economic Order Quantity model

Order should be placed when the stock is about to be depleted. It is possible since the demand is known and deterministic and the lead time is equal to zero. The evolution of the stock according to these hypotheses is represented in figure I.3. The duration between replenishment orders is noted ΔT. The average quantity in stock is equal to $Q/2$, and one has to place on average D/Q orders in a year. The average annual total cost C depending on Q can be written:

$$C(Q) = \frac{hQ}{2} + \frac{KD}{Q}$$

We find the minimum of the function by deriving the expression and setting $C'(Q) = 0$.

$$0 = \frac{h}{2} - \frac{KD}{Q^2}$$

The optimal order size Q^* that minimizes the total long term costs is therefore equal to the classical EOQ formula:

$$Q^* = \sqrt{\frac{2DK}{h}}$$

Despite its very limiting hypotheses the EOQ formula is still widely used as a convenient rule of thumb in many businesses, or as a complement to more complex solutions [4].

I.2.2 Reorder points methods

Historically, simple heuristics based on a few well chosen parameters have obtained great successes to solve single item inventory control problems. The most common type of policies for single item problems are reorder points methods, such as the (r, Q) [34] and the (s, S) [87] policies. The (r, Q) policy states that if at review time, the inventory level is equal or lower to r, a new replenishment order of size Q is placed. The (s, S) policy states that if at review time, the inventory level is equal or lower to s, a new replenishment order is placed, bringing the inventory level to the order-up-to-level S. An example of the two policies is presented in figure I.4, with negligible lead times, with the parameters s and r being equal, and the parameter S being equal to $r + Q$.

The intuition behind both policies is the same: the stochasticity of the demand combined with delivery leadtimes incite supply chain practitioners to reorder before the inventory level reaches zero. The values r and s are sometimes called the safety stocks. These parametric policies can be used as efficient heuristics on complex problems even when they are not optimal.

The two policies are identical in continuous review systems, as the inventory level never reaches values smaller than r and s. One can therefore write $r = s$ and $Q = S - s$ and the two policies are strictly identical. For periodic review, the two policies are different, as shown by figure I.4.

These parametric policies are very popular, because they are very intuitive and even optimal in many instances of single-item inventory control problems. The conditions and hypotheses required for (s, S) and (r, Q) policies to be optimal have been extensively studied [51, 59, 87, 92, 101]. These proofs rely on the hypothesis that the stochastic processes of the system are stationary. This hypothesis implies that if the system ends up in the same state (in terms of inventory) at two different times, then the situations can be considered as strictly identical and the optimal action is the same. Moreover, under the stationarity assumption of the stochastic processes, it is possible to leverage fixed points methods. If there is a finite number of possible states for the system (a finite combination of reasonable stock on hand and on order), one can for example write :

$$X = T(X, \pi) \tag{I.3}$$

X is the vector of each state long-term probability (how frequently each state would be encountered on average if the system was to run indefinitely). π is the policy function, that outputs the vector of actions associated to every possible state. T is the transition function, that outputs the updated state probability vector, given the input state probability vector and the policy. According to equation I.3, X is therefore a fixed point, and the long term average reward per time step associated to the policy π can be computed over a single 'average' time-step, which starts in the

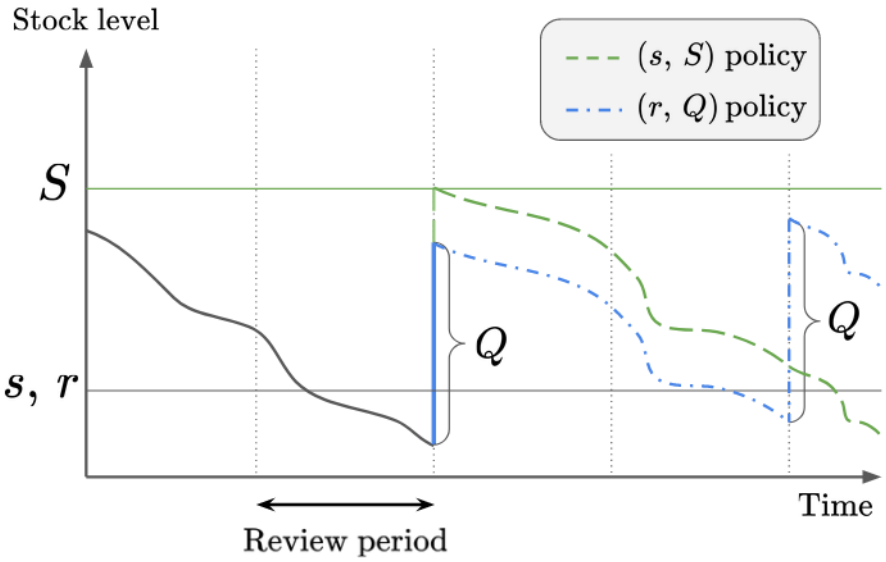


Figure I.4: Periodic review period, (s,S) and (r,Q) policies comparison

probabilistic state X. In these conditions, the (r, Q) policy has an interesting property: the long term inventory position probability distribution is uniform across $[r + 1, r + Q]$ [5]. This property can be useful to characterize and derive the optimal (r, Q) parameters.

While it is valuable to know the optimality conditions of (s, S) and (r, Q) policies, in practice supply chain practitioners are more interested in the computation of the parameters values. Many methods and heuristics have been developed to quickly compute the optimal values for (r, Q) and (s, S). They depend on the characteristics of the system. Notably in [113], the authors utilize the properties of the (s, S) policy and quick estimates of lower and upper bounds to develop an algorithm that has become a reference for both continuous and periodic review system. For the (r, Q) policies in continuous review systems, a similar idea was presented in [30].

As the assumption of stationary demand makes (s, S) policies practical, it is possible to utilize them as part of larger optimization problems. For supply chain design problems or joint location-inventory problems (choosing the optimal locations for production facilities or warehouses), it is common to assume that the demand is stationary, so that replenishment policies are simple (s, S) policies [3, 24, 110]. It is then possible to compute the associated long term expected shortage and holding costs per time steps, and to include these costs in a mixed integer linear [3] or non linear [24, 110] program formulation of the problem.

Example of a reorder point method applied to an inventory control problem

In order to illustrate the efficiency of reorder points methods, we provide a concrete applicative example. We consider an inventory control problem with a single item, and a weekly periodic review over an infinite horizon. We assume the demand d_t to be stochastic and stationary. For its probability distribution, we choose to represent the demand with a binomial distribution of parameters n, p at every time step, since it is one of the simplest demand model one can use. We note its mass probability function $\phi(d)$. For simplicity, we assume the lead time to be negligible and equal to zero. This assumption is generally not necessary to apply reorder point methods. We define x_t as the pre-order stock level at time t and a_t as the order. We consider a backlogged demand model, since optimality proofs of reorder point methods often rely on this hypothesis.

$$x_{t+1} = x_t + a_t - d_t$$

The costs (noted C_t) generated at any period t are composed of ordering costs, shortage costs, and holding costs. The shortage cost is equal to the number of missed sales multiplied by a parameter m. The holding cost is equal to the number of units still in stock after the demand process multiplied by the parameter h. The ordering cost is a function (noted $c(a)$) of the order size, and is composed of a fixed ordering cost K and a per unit ordering cost c.

$$c(a) = \begin{cases} 0 & \text{if } a = 0 \\ K + ca & \text{if } a > 0 \end{cases}$$

The expression of the cost C_t associated to time-step t is therefore:

$$C_t = c(a_t) - m\left[x_t + a_t - d_t\right]^- + h\left[x_t + a_t - d_t\right]^+$$

With $[x]^-$ and $[x]^+$ being $\min(0, x)$ and $\max(0, x)$ respectively. The expected C_t value knowing x_t and a_t is equal to:

$$\mathbb{E}[C_t | x_t = x, a_t = a] = c(a) + \sum_{d=0}^{x+a} (x + a - d)\, h\, \phi(d) + \sum_{d=x+a}^{\infty} (d - x - a)\, m\, \phi(d)$$

In order to demonstrate that the (s, S) policy is optimal on the infinite time steps instance of this problem, we first need to consider the same problem for a finite number of time steps. We consider that the first time step occurs at $t = 1$ and define t_{end} as the last time step. We define the value functions $v_t(x)$ as the expected value of the accumulated reward during the program between t and t_{end} (included) if the reorders are done optimally. The choice of maximizing a reward or minimizing a cost is a convention: we can define the reward as the opposite of the cost. One can then utilize the Bellman principle of optimality to describe these value functions recursively:

$$v_t(x) = \max_{a \geq 0} \left(-\mathbb{E}[C | x_t = x, a_t = a] + \sum_{d=0}^{\infty} v_{t+1}(x + a - d)\phi(d) \right)$$

For this model of ordering, holding and shortage costs, one can demonstrate recursively the K-concavity of the $v_t(x)$ value functions [87]. A function f is K-concave if:

$$K + f(x) - f(a + x) + a\frac{f(x) - f(x - b)}{b} > 0$$

This K-concavity of the value functions ensures that the optimal policy at time t is a (s, S) policy. This result is proven by contradiction in [87]. The optimality of the (s, S)-policy is extended to the infinite horizon version of the problem in [51].

The next step is to compute the best (s, S) values. We note $C_\infty(s, S)$ the long term average costs (over a single time step) associated to applying the (s, S) policy. $C_\infty(s, S)$ can be seen as $\lim_{t\to\infty} C_t$, and the goal is to minimize it.

The minimum long term pre-order stock level (noted x_{min}) is equal to $s + 1 - n$, since $s + 1$ is the minimum observable post-order stock and n is the maximum observable demand (the forecast is a binomial distribution of parameter n, p). We note $\pi_{s,S}(x)$ the order selected by the (s, S)-policy when x_t is equal to x.

We introduce $X = \{p_i\}_{i \in [x_{min}..S]}$, the vector containing the long term average probabilities of each possible pre-order stock levels (also called steady state probability vector), with $p_i = \lim_{t\to\infty} P(x_t = i)$. The maximum long term pre-order stock level is equal to S, since once the stock level gets below S, it necessarily remains below S as long as the (s, S) policy is applied. The X vector size is therefore equal to $S - s + n$. We note Π the transition matrix associated to the (s, S)-policy. Its terms $\pi_{j,i}$ correspond to the probability to end up with the post order stock level j from the pre-order stock level i. Its terms are equal to:

$$\Pi_{j,i} = \begin{cases} 1 & \forall i \in [x_{min}..s], j = S \\ 0 & \forall i \in [x_{min}..s], j \neq S \\ 1 & \forall i \in [s+1..S], j = i \\ 0 & \forall i \in [s+1..S], j \neq i \end{cases}$$

We introduce T, the transition matrix associated to the stochastic demand d.

$$T_{k,j} = P(d = j - k), \forall j \in [s+1..S], k \in [x_{min}..S]$$

The global state transition matrix P is equal to $P = T\Pi$, and one can write :

$$\begin{cases} X = PX \\ \sum_{i=x_{min}}^{S} p_i = 1 \end{cases}$$

This linear system can be solved to compute the steady state probability vector X associated with the selected (s, S) parameters. The corresponding long term average cost is equal to the expected reward associated to each possible state, multiplied by the long term probability of this state:

$$C_\infty(s, S) = \sum_{i=x_{min}}^{S} p_i \mathbb{E}[m\,[i + \pi_{s,S}(i) - d]^- + h[i + \pi_{s,S}(i) - d]^+]$$

One can then explore the (s, S) parameter space to determine the parameters (s^*, S^*) that minimize the long term average cost $C_\infty(s, S)$.

Conclusion

This example illustrated how reorder points methods can solve single item stochastic inventory control problems. The first limitation of these methods is that it is very often necessary to assume that the demand is stationary to both prove the optimality of such policies and to compute the best parameter values. Such an assumption is very limiting in real world supply chains, where demand forecasts often take into account seasonality. The second limitation is that these approaches can not deal with constraints shared between multiple items, which are common practice in supply chains. Despite these limitations, reorder point methods are widely used in the industry due to their simplicity and intuitive formulation. In order to overcome these limitations, it is very natural to consider classical operational research approaches, such as linear programming, integer programming, and non linear programming.

Problem characteristics	Deterministic	2-stage stochastic	n-stage stochastic	Stationary demand	Variable demand	Single item	Multi-item
Reorder points applicability	✓	✓	✓	✓	✗	✓	✗

Table I.1: Scope of applicability of reorder points methods

I.3 Standard operational research approaches

The preferred approach to solve many optimization problems is to frame them as linear programs, integer programming problems, or mixed integer programming problems. There exists many software tools dedicated to solve such programs. In particular, linear programming solvers are usually very efficient. For inventory control problems, the main challenge when using such approaches

is to simplify the problem enough to re-formulate it as a linear or non linear program, without significantly degrading the model of the real supply chain.

I.3.1 Concepts and definitions

In this subsection, we focus on deterministic systems, as they are often treated differently than stochastic systems. Without uncertainty, all the decisions can be computed at the beginning of the sequence, since no additional information will be available at later stages. Deterministic systems are therefore easier to deal with. Notably one can optimize the decisions directly instead of developing policies that consider every possible state of the system.

As standard state of the art optimization methods, Linear Programming (LP), Mixed-Integer Linear Programming (MILP) and Nonlinear Programming (NLP) are useful methods to solve inventory control problems. Linear programming is defined as an optimization method to maximize a linear objective function subjected to linear equality and inequality constraints. The canonical form of a linear program is:

$$\begin{cases} \text{Find a vector } x \\ \text{that maximizes } c^\intercal x \\ \text{subject to } Ax \leq b \\ \text{and } x \geq 0 \end{cases}$$

The components of x are the variables to optimize. A is a matrix given by the equations of the system. Similarly, b and c are given vectors expressing the objective function and constraints of the system. $c^\intercal x$ is called the objective function. The linear constraints ($Ax \leq b$ and $x \geq 0$) define a convex feasible region (a convex polytope) for x. Standard linear programming algorithms are usually either variants of the simplex algorithm or interior point methods. The simplex algorithm is based on the exploration of the edges of the convex polytope, while interior point methods explore solutions within the interior of the polytope [52].

For practical inventory control applications, the variables are often integers, as continuous variables can not represent the actual granularity of most transactions. In a linear program, if at least one of the variable is required to be an integer, it becomes an mixed-integer linear program. While linear programs can be solved efficiently with the previously mentioned methods, mixed-integer linear programs are often NP-hard [88]. It is therefore common for inventory control problems formulations to prefer continuous stocks and orders, since the computational gain given by the continuous approximation of discrete variables may outweigh the induced accuracy loss.

If one of the constraint or the objective function is not linear, the problem is a non-linear program. The algorithm choice for non linear programs depend on the structure of the problem. If both the objective function and the constraint sets are convex, convex optimization (usually polynomial-time) algorithms can be applied [18]. For non convex problems, branch and bound techniques are popular [22], and rely on dividing the program into sub-problems and lower bound approximations.

One can note that in such programs, the constraints and the objective function are not stochastic: they are mostly suited for deterministic inventory control problems. In practice, the hypothesis of a deterministic system is most common for inventory control problems that deal with production processes. In [41] a general framework for deterministic models of production processes relying on linear programming is presented. A concrete example of a real world refinery scheduling model is studied in [56] as a mixed integer non linear program. The authors re-formulate the problem into a large -but simpler to solve- mixed integer linear program. On such complicated problems, especially when dealing with non-linear problems, it is not always possible to ensure that the obtained solution is optimal.

Example of linear programming applied to a deterministic inventory control problem

In order to illustrate how linear programming can be applied to inventory control, we provide a concrete applicative example. We consider an inventory control problem with a single item and a weekly periodic review over a finite horizon of t_{end} weeks. The demand is deterministic but variable over time (if the demand was stochastic, this linear programming approach would not be feasible). We define d_t as the demand over the t_{end} weeks, with $t \in [0..t_{end}-1]$. For all $t \in [0..t_{end}-1]$, the stock level is noted x_t, and the replenishment order is noted a_t. We choose to allow x_t and a_t to be real values and not integers, since linear programs are significantly easier to solve than integer programs. The cost of replenishing the inventory is equal to $c_t\, a_t$, c_t being the unit replenishment cost. It varies over time and its evolution is known in advance. At the beginning of the episode, the retailer has a stock on hand x_0. The lead time is negligible and considered equal to zero (this assumption is not necessary to express the system as a linear program, but it simplifies the notations in this example). If the demand exceeds the available stock, the demand is backlogged. This last assumption is important, since lost sales model are non linear and would require the use of less efficient non-linear programming techniques. For all $t \in [0..t_{end}-2]$, one can write:

$$x_{t+1} = x_t + a_t - d_t \tag{I.4}$$

We extend the definition of x_t to $t = t_{end}$, with $x_{t_{end}}$ defined as: $x_{t_{end}} = x_{t_{end}-1} + a_{t_{end}-1} - d_{t_{end}-1}$. By immediate recursion, one can express x_t depending on the initial stock x_0 and the demand observed between $t' = 0$ and $t' = t-1$, for all t $[0..t_{end}]$:

$$x_t = x_0 + \sum_{t'=0}^{t-1} a_{t'} - d_{t'}$$

A shortage cost penalty m is generated for every backlogged unit of demand. If the available stock exceeds the demand, any unsold unit generates a holding cost h. The difficulty of this problem lies in the fact that the replenishment unit cost c_t varies over time: the retailer can anticipate to purchase inventory at lower price, even if it may induce supplementary holding costs. The equation of the loss of the system is:

$$L = \sum_{t=0}^{T-1} -m\,[x_{t+1}]^- + h\,[x_{t+1}]^+ + c_t\, a_t$$

The cost function is not completely linear due to the $[x_t]^-$ and $[x_t]^+$ terms. One can introduce the intermediate variables x_t^+ and x_t^- to alleviate this problem, and set the constraint: $x_t = x_t^+ - x_t^-$ and $x_t^+, x_t^- \geq 0$. With this definition, x_t^+ and x_t^- could in theory both be simultaneously non equal to zero, which would mean that the system is simultaneously overstocked and out of stock. In practice however the best solutions minimize the holding and shortage costs, and at least one of the two values is equal to zero. The equations can therefore be re-formulated as the following linear program:

$$\begin{cases} \text{Select } a_t, x_t^+, x_t^- \\ \text{that minimize: } L = \sum_{t=0}^{t_{end}-1} m\, x_{t+1}^- + h\, x_{t+1}^+ + c_t\, a_t \\ \text{subject to: } \forall t \in [0..t_{end}], x_t = x_0 + \sum_{t'=0}^{t-1} a_{t'} - d_{t'}, x_t = x_t^+ + x_t^- \\ \text{and } a_t, x_t^+, -x_t^- \geq 0 \end{cases}$$

This linear program can be solved using any off the shelf linear programming solver, and provides the best possible sequence of actions a_t.

I.3.2 Two stage stochastic programming

The hypothesis of a deterministic inventory control system makes possible to use linear and non linear programming optimization methods, since the sequence of optimal actions can be computed from the start. However most real world inventory control problems still remain stochastic, which prevents businesses from planning ahead all their decisions: stochastic outcomes mean that the ability to adapt is crucial.

Some approaches willingly ignore the stochasticity of the demand in order to be able to build approximate solutions using linear programming techniques (for example in [76]). It is however quite obvious that accounting for the demand stochasticity can significantly improve performance compared to naïve deterministic approaches (an example can be found in the last experiment in [21]).

If the number of stochastic time steps in an inventory system is limited to one, or if it can be approximated as such, the inventory control problem can be defined as a two-stage stochastic program. If the number of scenarios is finite (and a probability can be associated to each scenario), a stochastic linear program can be seen as a collection of non stochastic linear programs, one for each possible scenario. The stochastic linear program can be rewritten as a large deterministic equivalent linear program [14], that minimizes the sum of the non stochastic linear programs objective functions weighted by the respective probability of the scenario. By definition, it is equivalent to minimizing the expected value of the objective function. Such approaches are commonly applied to two-stage inventory control problems, see [23, 27].

The limitation of this approach is that it is not applicable to general multi-step inventory control problems: the number of scenarios and possible decision variables combinations increase exponentially with the number of time-steps. For example, the inventory control problem discussed in section I.2.2 can not be tackled with this type of approach, as the temporal horizon is infinite.

Conclusion

Classical operational research methods are therefore suited to solve many supply chain problems. Their advantage compared to reorder points methods is that they can deal with constraint shared between multiple items and with variable demand. When it comes to stochastic inventory control however, linear programming techniques can only provide solutions to two stage stochastic programs. Unfortunately, most inventory control problems are formulated as iterated games, where adaptability to the outcome of the stochastic processes is crucial. In order to deal with these problems, one therefore needs to consider the dedicated field of dynamic programming.

Problem characteristics	Deterministic	2-stage stochastic	n-stage stochastic	Stationary demand	Variable demand	Single item	Multi-item
Linear Programming	✓	✓	✗	✓	✓	✓	✓

Table I.2: Scope of applicability of reorder points methods

I.4 Dynamic programming

Dynamic programming is a mathematical optimization method introduced by Bellman in the 1950s [8]. It relies on the idea that some large problems can be broken down into simpler sub-problems, that can be solved recursively. Dynamic programming is well suited for inventory control problems, since the actor faces a slight variation of the same problem at every time step.

I.4.1 Deterministic dynamic programming

In this subsection, we focus on deterministic problems. In this situation if the system has a finite horizon, all the decisions over the entire horizon can be decided from the start. We introduce a few notations: at each time step t, the process is in some state x belonging to the state space X_t, and a decision maker may choose any action a in the action space A_t. The system then moves to the next state $x' \in X_{t+1}$ depending on x and a. During this state transition, the system generates the corresponding reward $R_t(x, a, x')$.

If one considers a problem with a finite number of time steps, one can define the gain G_t as the sum of reward from a time step t to the end of the horizon t_{end}. It depends on the state of the system and the future actions (that we note $\pi = \{a_{t'}\}_{t' \in [t..t_{end}]}$).

$$G_t(x, \pi) = \sum_{t'=t}^{t_{end}} R_{t'}$$

The goal is to find the optimal set of actions π that maximizes G_0. For periodic review system with a finite horizon, the dynamic programming approach leverages the sequence of value functions V_t. In a deterministic system, a value function $V_t(x)$ is defined as maximum gain G_t that can be generated from a state x. The value functions are described recursively by the Bellman equation:

$$V_t(x) = \max_{a \in A_t}(V_{t+1}(x') + R_t(x, a, x'))$$

With x' being the state reached when the action a is selected while in state x. This equation states that the value function associated to the state x can be computed from V_{t+1}. One first needs to compute, for every possible action, the associated immediate reward $R_t(x, a, x')$ and the long term reward $V_{t+1}(x')$. Then, one can simply select the action that maximizes the sum of the two components. This equation is schematized in figure I.5.

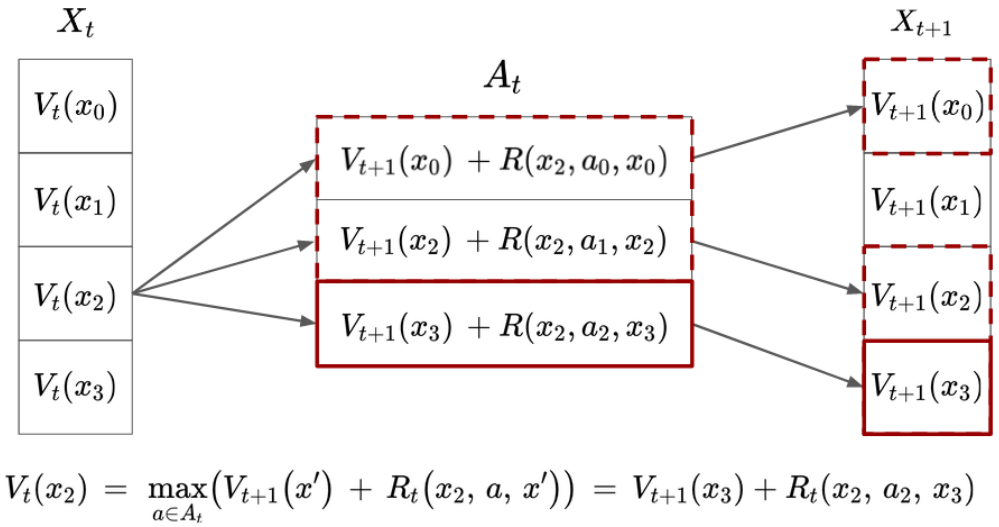


Figure I.5: The Bellman equation in a deterministic system

A classical dynamic programming algorithm is the backward recursion. If one can define the reward for each possible final stage x, one can build the equivalent $V_{t_{end}+1}(x)$. Since one can compute $V_t(x)$ for all x in X_t from V_{t+1}, one can recursively compute all the value functions up until V_0, and store the optimal action a associated to each state x at every time step. This process is called

the backward recursion. The Wagner Whitin algorithm [102] is an example of utilization of the dynamic programming approach on the classical dynamic lot size problem.

The main limitation of the backward recursion algorithm is that it requires to both compute and store at least one value per action, and one value per state, and to do so for every time step as illustrated in figure I.5. Its complexity C is equal to:

$$C \sim Card(X)\, Card(A)\, t_{end}$$

With $Card(X)$ being the size of the state space, $Card(A)$ being the size of the action space. The curse of dimensionality of the state and action spaces makes this algorithm impractical for many inventory control problems. As an example, if we consider an inventory control problem with a hundred items, subjected to a shared constraint that makes items evolution depend from each other. With ten possible actions for each item at every time-step, the size of the action space is equal to 10^{100}, and the backward recursion algorithm is clearly intractable. This limitation is shared by all the classical dynamic programming algorithms [11, 78].

For continuous review systems, one can not define a sequence of value functions V_t. Instead, the dynamic programming principle can be combined with control theory to provide optimal control laws [9]. Similarly to the periodic review case, this approach is only applicable to rather simple instances of inventory control problems.

Example of dynamic programming applied to a deterministic inventory control problem

We provide an example of application of the backward recursion algorithm. We apply it to the same inventory control problem as in section I.3.1. As a reminder, the problem has a weekly periodic review over a finite horizon of T weeks, with deterministic demand, no lost sales, and variable ordering costs over time. Since value function based methods usually utilize the convention of maximizing the reward rather than minimizing the loss, we slightly change the equations of the system to adopt this convention:

$$\begin{aligned} x_{t+1} &= x_t + a_t - d_t \\ r_t &= \sum_{t=0}^{T-1} m[x_{t+1}]^- - h[x_{t+1}]^+ - c_t a_t \\ a_t &\geq 0 \end{aligned}$$

The backward recursion algorithm can be applied only to finite state and action spaces. In this modelization of the problem, the stocks and orders are continuous variables, and the state and action spaces are not finite. One therefore has to proceed to an approximation, and divide the state and action space into discrete increments of constant size. Moreover, the state and action spaces need to be bounded. We therefore define x_{max}, the maximum possible post order reception stock level. The complexity of the backward recursion algorithm is very sensitive to the size of the state and action spaces. The lead time is assumed to be negligible. While this assumption was not necessary to apply reorder points methods, it is critical when framing the problem as a dynamic program, since the size of the state space $Card(X)$ in this problem is equal to:

$$Card(X) = Card(A)^l Card(S)$$

With $Card(A)$ being the size of the action space, $Card(S)$ being the number of possible stock levels, and l being the leadtime (expressed in number of time steps). The maximum state/action values and the increment size also impact the cardinality of the state and action spaces. The discretization of the state and action space is therefore critical: Larger increment size will decrease

the computational cost, but may lead to less accurate policies. In this example, the retailer can compare the policies performance for decreasing increment sizes, and stop whenever the performance are not significantly improved by decreasing the increment size.

Since the total demand until the end of the episode is bounded and equal to $\sum_{t'=t}^{t_{end}-1} d_{t'}$, one can set $x_{max} = \sum_{t'=t}^{t_{end}-1} d_{t'}$ and $a_{max} = x_{max} - x_t$. In practice, depending on the d_t and c_t values, one may choose smaller values of x_{max} to reduce the algorithm complexity.

In this problem, the remaining stock at the end of the episode is considered lost, but does not generate additional costs. Therefore one can write $\forall x \in X_{t_{end}}, V_{t_{end}}(x) = 0$. The equation of the reward for this problem is:

$$R_t(x, a, x') = m[x']^- - h[x']^+ - c_t a = m[x + a - d_t]^- - h[x + a - d_t]^+ - c_t a$$

We introduce the notation $\vec{V_t}$ for the vector associated to the value function, with one value per possible state. The vector size is equal to $x_{max} + 1$. Formally, one can write $\vec{V_t} = \{V_t(x_i)\}_{x_i \in [0..x_{max}]}$. Using the backward recursion, one can write that $\forall x_i \in [0..x_{max}]$:

$$V_t(x_i) = \max_{a \in [0, a_{max}]} (V_{t+1}(x_i + a - d_t) + m[x_i + a - d_t]^- - h[x_i + a - d_t]^+ - c_t a)$$

$\vec{V_t}$ can therefore be computed from d_t and $\vec{V_{t+1}}$. One can recursively reach $\vec{V_0}$. For every time step t, one can store in a table the argument a that provides the maximum for every x_i. This table defines the optimal policy $\pi^*(x)$ for every time step, for every possible state. One then inputs the initial state of the system x_0 to the optimal policy to determine the best order to place initially, determine the new stock level x_1, and run forward recursively to determine the best sequence of orders.

It is to be noted that the solution presented in section I.3.1 is theoretically superior since it provides an optimal solution and does not require to discretize of the state and action spaces.

I.4.2 Stochastic dynamic programming

In real world supply chains the demand is more frequently considered stochastic than deterministic. Stochastic inventory control problems can be framed as stochastic dynamic programs. Stochastic dynamic programming is a set of techniques to solve problems of decision making under uncertainty. The goal is to provide a policy that acts optimally, considering that the system is subjected to stochastic processes. A review of dynamic programming techniques for supply chains is presented in [86]

For periodic review inventory control systems, the markov decision process (MDP) formalization fits very well. In a MDP, a decision maker must decide at every time step the action a it should apply to the system, depending on its current state x. Once the action is applied, depending on the stochasticity of the system, the initial state x and the action a, a new state x' has a given probability of being reached. This transition between states generates a reward $R_t(x, a, x')$.

The goal is to determine a policy π, a function that specifies the action $a = \pi(x)$ that the decision maker should choose when in any state $x \in X$. This policy is built to maximize a function that represents the cumulative reward received in the long run. If the number of time steps is finite, we define this function as the gain G_t, the sum of reward from a time step t to the end of the horizon t_{end}.

$$G_t = \sum_{t'=t}^{t'=t_{end}} R_{t'}$$

If one considers a infinite number of time steps, one needs to introduce an exponential discount rate γ to the reward, so that the sum converges toward a finite value.

$$G_t = \sum_{t' \geq t} \gamma^{t'-t} R_{t'}$$

The value functions are useful concepts to deal with stochastic dynamic programs. The state-value function $v_t^\pi(x)$ is equal to the expected gain, if the one chooses to follow the policy π, when starting in state x.

$$v_t^\pi(x) = \mathbb{E}\left[G_t | x_t = x, a_{t' \geq t} = \pi(x_{t'})\right]$$

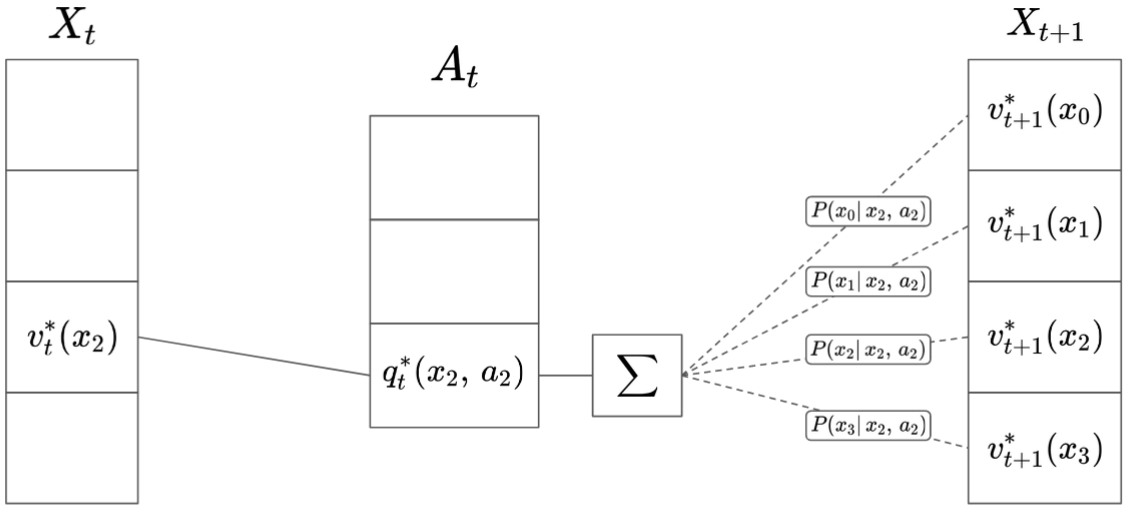
Similarly, the action value function $q^\pi(x, a)$ is defined as the expected gain starting in state x if action a is selected instead of $\pi(x)$ at t, and that subsequent actions are selected following the policy π.

$$q_t^\pi(x, a) = \mathbb{E}\left[G_t | x_t = x, a_t = a, a_{t' > t} = \pi(x_{t'})\right]$$

The expected reward for being in a particular state x and following the optimal policy π^* is described recursively by the following Bellman optimality equations:

$$q_t^*(x, a) = \sum_{x' \in X_{t+1}} \left(P(x_{t+1} = x' | x_t = x, a_t = a)(\gamma v_{t+1}^*(x') + R_t(x, a, x')\right) \tag{I.5}$$

$$v_t^*(x) = \max_{a \in A_t}(q_t^*(x, a)) \tag{I.6}$$



$$q_t^*(x_2, a_2) = \sum_{x' \in X_{t+1}} P\left(x_{t+1} = x' \,|\, x_t = x_2, a = a_2\right)\left(R\left(x_2, a_2, x'\right) + v_{t+1}^*\left(x'\right)\right)$$

Figure I.6: Bellman optimality equation in a stochastic system (part 1)

The equations I.5 and I.6 are schematized in figures I.6 and I.7, on a very simple MDP with four states and three actions, with $\gamma = 1$. The γ parameter is set to values smaller than 1 when dealing with infinite horizon, so that the value functions remain bounded. Using these equations and similarly to section I.4.1, the backward recursion algorithm can be applied [10] to determine the optimal policy π^*. The curse of dimensionality however still remains a critical limitation of the backward recursion algorithm, and can not be realistically applied on inventory control problems with large state or action spaces [11, 78].

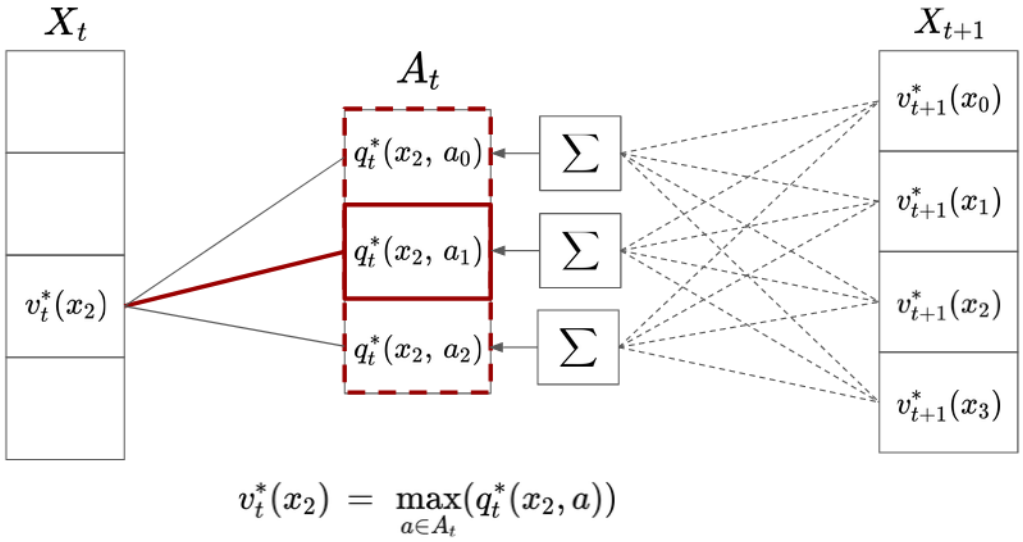


Figure I.7: Bellman optimality equation in a stochastic system (part 2)

Example of stochastic dynamic programming applied to a stochastic inventory control problem

The goal of this section is to provide an example of how the stochastic dynamic programming approach and the backward recursion algorithm can be applied to stochastic inventory control problems with relatively small state and action spaces. We consider the example already utilized in section I.2.2. As a reminder, it is an inventory control problem with a single item, and a weekly periodic review over an infinite horizon. The demand is stochastic, stationary, and to follow a binomial distribution of parameters n, p. Its probability mass function is noted $\phi(d)$. Sales are backlogged if no stock is available to cover the demand. The lead time is negligible, the orders are delivered instantaneously. Since value function based methods usually utilize the convention of maximizing the reward rather than minimizing the loss, we slightly change the equations of the system to adopt this convention:

$$
\begin{aligned}
&x_{t+1} = x_t + a_t - d_t \\
&r_t = -c(a_t) + m[x_t + a_t - d_t]^- - h[x_t + a_t - d_t]^+ \\
&c(a) = \begin{cases} 0 & \text{if } a = 0 \\ K + ca & \text{if } a > 0 \end{cases}
\end{aligned}
$$

Firstly, the backward recursion algorithm requires a finite number of time steps (noted t_{end}), since the computation relies on the ability to estimate $v_{t_{end}}$. We therefore approximate the infinite horizon by a large enough finite horizon, and assume $v_{t_{end}}(x) = 0$. The policy $\pi^*(x)$ should be approximated by the policy $\pi_1(x)$ suggested at the first time step by the backward recursion algorithm. Due to the stochasticity of the demand, the inaccuracies induced (at $t = 1$) by the approximation that $v_{t_{end}}(x) = 0$ decreases the greater t_{end} is. Therefore, one should increase t_{end} gradually until the policy $\pi_1(x)$ suggested by the backward recursion does not change.

Secondly, the backward recursion algorithm can only be applied to finite state and action spaces. Since the stock and the demand are already discrete variables, one does not need to discretize them. One however needs to bound the maximum stock and orders that should be realistically considered. We define x_{min} as the minimum observable stock level, and we set it to $-n$ since the demand is bounded by n by definition of the binomial distribution at every time step. We define x_{max} and a_{max}, the maximum realistically reachable stock levels and orders. In this case, the maximum possible

observable demand over the entire episode is equal to $n\,t_{end}$. We can therefore set $x_{max} = n\,t_{end}$ and $a_{max} = x_{max} - x_t$. This bound still remains very conservative and one may in practice consider smaller values of x_{max} to reduce the backward recursion complexity. For this inventory control problem, the equation I.5 becomes :

$$q_t^*(x,a) = \sum_{x=x_{min}}^{x_{max}} \phi(x' - x - a)\left(\gamma v_{t+1}^*(x') + m[x']^- - h[x']^+ - c(a)\right) \tag{I.7}$$

One can then apply the backward recursion algorithm, utilizing equations I.7 and I.6 to compute recursively the $q_t^*(x,a)$, $v_t^*(x,a)$ and $\pi_t^*(x)$ functions for all $t \in [1..t_{end}]$. One can then use $\pi_1^*(x)$ as an approximation for the actual best policy $\pi^*(x)$ in the infinite time steps scenario.

Conclusion

This example illustrated how powerful stochastic dynamic programming can prove to be on small scale inventory control problems. Any type of constraint can be dealt with, even constraints shared between multiple items: these constraints are included in the reward function or simply limit the action space that has to be explored. Any number of time steps can be considered: while the complexity of the forward recursion is exponential with the number of time steps, the backward recursion algorithm complexity is only linear with it. As stated previously, the main limitation of the backward or forward recursion algorithms is that their complexity is very sensitive to the state space and action space sizes. Unfortunately, the curse of dimensionality [11, 78] very often applies to the state and action spaces of non-trivial inventory control problems. Therefore, stochastic programming methods are in practice only applicable to simple instances of single item and multi-item inventory control problems. The fields of approximate dynamic programming and reinforcement learning have emerged as a response to this limitation of classical dynamic programming.

Problem characteristics	Deterministic	2-stage stochastic	n-stage stochastic	Stationary demand	Variable demand	Single item	Multi-item
Dynamic Programming	✓	✓	✓	✓	✓	(✓)	(✓)

Table I.3: Scope of applicability of dynamic programming

I.5 Approximate dynamic programming and reinforcement learning

For large scale multi-period multi-item inventory control problems with non stationary demands, the classical stochastic dynamic techniques (see section I.4) are not able to provide tractable solutions. Approximate dynamic programming is an umbrella term for algorithmic strategies that overcome this curse by applying well chosen approximations to the Bellman equation.

I.5.1 Value function approximations

The core idea of approximate dynamic programming is to approximate the state or action value functions. The model can then be fitted through analytical analysis or, most frequently, statistical approximations. In this latter case, the approximate dynamic programming field overlaps with the reinforcement learning field. Many reinforcement learning algorithms indeed rely on approximate value functions [95]. Value function approximations methods are also called Q-learning techniques.

On the contrary to the classical dynamic programming approach, Q-learning techniques typically apply forward propagation logic [78] instead of backward propagation. Simulations progress forward in time, from an initial state, and following a policy. The outcome of these simulations can be used to update the approximation of the value function. The choice of this policy is crucial, as the explored states and actions depend largely on it. It is subjected to the well documented exploitation-exploitation trade-off [95].

The simplest approximation for value functions is to use a look-up table, which means that for every explored state or state-action pair, a value is assigned. Every time a scenario goes through this state or action state pair, the value in the table is updated using a well chosen rule [95]. This approach removes the need to evaluate the value of all possible state and state-action pair (compared to a full backward recursion algorithm), but one still needs an approximate value for every state he might visit or every state-action pair he might choose. Depending on the problem, these numbers can be too high to be realistically stored into a look-up table. This simple approach is therefore in practice limited to small scale discrete state and action spaces. It has found practical applications for some inventory control problems, as demonstrated in [21, 35, 58].

For large scale problems, the preferred approach is to use parametric approximations of the value functions. The choice of the parametric approximation should be made depending on the characteristics of the problem. In [97], the authors apply several possible approximations, including linear and piecewise-linear value functions approximations, to a resource allocation problem. In [79], the authors describe a more general approach for resource allocation problem, with linear models of the value function using basis functions. They discuss the choices of the basis functions and methods to estimate parameters values.

I.5.2 Deep Q-networks (DQN)

Artificial neural networks are a very popular way of approximating value functions, notably using deep Q-networks [72]. Their advantage is that they are theoretically capable of approximating any function according to the universal approximation theorem [50]. While a poor choice of basis functions may hold back the ability to effectively approximate value functions (even with perfectly tuned parameters), deep neural networks have the theoretical ability to fit any function. For deep Q-networks, the input of the neural network is the state of the system, and the output layer has a single neuron for each valid action. The output value of the neuron associated to the action a, is the estimated action value $\bar{q}(s, a)$.

The neural network is initialized randomly, and has to be trained to minimize the difference between $q(x, a)$ and $\bar{q}(s, a)$. One does not have access to the $q(x, a)$ directly to regress and train the weights of the neural network. Deep Q-learning algorithms leverage the idea of temporal difference learning : $q(x, a)$ is replaced by its noisy estimate $R(x, a, x') - \gamma \max_{a' \in A} \bar{q}(x', a')$, with x' and a' being the state and action at time $t + 1$. In terms of loss, it can be written as:

$$\mathcal{L} = \|\bar{q}(x, a) - R(x, a, x') - \gamma \max_{a'} \bar{q}(x', a')\|$$

A concrete example of DQN applied to inventory control of perishable goods is presented in [104]. The applicability of deep reinforcement learning to inventory control has been discussed in [17]. The authors point out a major limitation when it comes to large scale inventory control problem : the size of the action space. Deep reinforcement learning elegantly solves the curse of dimensionality for the state space, as demonstrated by the ability of alpha Go, a DQN assisted tree search, to achieve super-human performance when playing the game of Go [90]. With DQN approaches, the network requires an output layer of a size equal to the number of legal actions. This is a major limitation for problems with continuous action spaces or very with large discrete action spaces. We reuse the example of an inventory control problem with a hundred items, subjected to a shared constraint

that makes items evolution depend from each other. With ten possible actions for each item at every time-step, the size of the action space is equal to 10^{100}: an output layer of 10^{100} neurons can not reasonably be considered.

I.5.3 Direct policy search

A possible approach to overcome the curse of dimensionality on the action space is to directly approximate the policy π^* with a deep neural network. For continuous action spaces, the network simply outputs the continuous values corresponding to the selected actions. It is not so simple in the case of discrete action spaces: it is not possible to directly output for discrete actions from a neural network, since neural networks are continuous (and differentiable) functions. For this reason, at least one output neuron has to be assigned for each possible action, may it be to output the associated estimated action value (for Deep Q networks) or the probability of selecting each action (for stochastic policies, that can be trained through gradient estimates provided by algorithms such as the REINFORCE method [106]). Similarly to deep Q-networks, policy based approaches on discrete action spaces are therefore not applicable to problems with very large action spaces, which is the case for many multi-item inventory control problems.

On continuous action spaces, this limitation is not a problem: the output of the neural network is directly the selected action. Such policies are trained through gradient ascent, to increase the expected reward. The problem is that the analytic expression of the gradient is generally not available. If the system's equations are differentiable, one can use the deterministic policy gradient (DPG) [89] to compute an estimate of the gradient. If the system is unknown or not differentiable, one can utilize the deep deterministic policy gradient algorithm (DDPG) [68], with an actor-critic architecture. The critic network is trained to approximate the action value function, and to provide a differentiable 'model' of the environment, from which the gradient that increases the expected approximate value function can be computed.

These policy-based algorithm on continuous action spaces have several limitations. Firstly, the action space of the problem needs to be continuous, or be approximated as continuous for such algorithms to apply. Secondly, the search for improved policies being gradient based, it is very possible that the trained policy ends up stuck in a local minimum in the space of possible policies. The probability of such issues occurring depends on the structure of the problem. For example, gradient based policy searches are unlikely to converge properly when applied to the MOQ problem (an explanation is provided in chapter IV). The third common issue with actor critic architectures is that state of the art algorithms are still very unstable [54], and require significant parameter tuning to provide satisfactory solutions. We believe that these reasons explain the lack of successful applications of continuous policy based reinforcement learning techniques (to the best of our knowledge) in the inventory control literature yet. Discrete action spaces and value based methods appear to be preferred [17, 35, 36, 58, 104].

Conclusion

Reinforcement learning is in theory a very powerful framework. It is capable of dealing with multi stage, stochastic, non stationary inventory control problems. It is also not constrained by the state space curse of dimensionality, that prevented stochastic dynamic programming from being applicable to many inventory control problems. On the other hand the size of the action space still remains an issue, since the most successful reinforcement learning approaches require the last layer of the neural network to have a size equal to the number of possible actions. While this is not an issue for single item problems, it can prove difficult to apply directly the state of the art reinforcement learning algorithms to multi-item inventory control problems. It is notably the case for the MOQ problem: Chapter IV will be dedicated to the use of reinforcement learning to solve

the MOQ problem. The field of deep reinforcement learning is still relatively recent and evolves rapidly. It is possible that new powerful algorithms emerge in a near future that would solve the aforementioned limitations.

Problem characteristics	Deterministic	2-stage stochastic	n-stage stochastic	Stationary demand	Variable demand	Single item	Multi-item
Reinforcement learning	✓	✓	✓	✓	✓	✓	(✓)

Table I.4: Scope of applicability of reinforcement learning

I.6 Other approaches

We briefly present a few other types of approach that mainly rely on the exploitation of the specific characteristics of the inventory control problem being faced. In that sense, they should not be considered as 'off the shelf' solutions, since they require significant work to be adapted to every problem.

I.6.1 Control theory approaches

Control theory deals with dynamic systems. Its goal is to provide algorithms that can control these systems to reliably bring them to a desired state. The diagrammatic style known as 'block diagram' is associated with control theory. It schematizes the different components of the system and their associated mathematical models. These block diagrams are based on the differential equations that rule the interactions between the inputs and outputs of the different blocks.

Control theory historically relied on the analysis of the systems in the frequency domain. The signals entering the system are converted from functions of time to functions of the frequency, using Fourier, Laplace, or Z transforms. In the frequency domain the differential equations can be translated into algebraic equations, and the frequency response of the system can therefore be more easily analyzed [32]. Such approaches are however limited to linear systems.

Control theory has been applied to supply chain systems as early as 1952, to control production rates [91]. It is however with the Forrester's Industrial dynamics methodology [32] that control theory became a common approach to tackle inventory control problems.

Control theory is especially valuable for multi-echelon systems, where the stability of the supply chain response can become an issue. The bullwhip effect is a phenomenon that occurs in supply chains when small variations of demand at the lower supply chain levels generate much larger variations at higher levels [32, 67].

Control theory applications to inventory and production control were presented in [98]. The desired state is a well chosen target stock, usually a multiple of the average expected demand. The control blocks are simple parametrized replenishment policies. The stability and the performance of the supply chain regarding its ability to reliably reach these target stocks can be evaluated with frequency control tests. The policy parameters are tuned in order to avoid potential instabilities in the form of the bullwhip effect. Similarly, a control theory based approach to study of the stability of different ordering policies was presented in [77].

To conclude, control theory techniques can be valuable for multi echelon supply chain problems, as their main selling point is how easy it is to perform stability analysis of the systems through the frequency domain. They are however limited to single item problems, and still require supply chain practitioners to set parametric replenishment policies.

I.6.2 Simulation based approaches

Simulation-based optimization is an optimization approach that relies on the analysis of the inputs influence on the performance, using simulated scenarios of a mathematical model of the system. The goal is to find the best possible inputs to the system. For inventory control problems, these inputs are generally the parameters of the replenishment policy. For stochastic systems, it is often too costly or practically impossible to evaluate analytically the expected performance of a parametrized policy. It can be more practical to run simulations using the considered inputs. If enough scenarios are considered, the law of large numbers ensures that the resulting average performance can be trusted to a certain degree to approach the actual expected performance value. The number of runs used for the estimation of the performance of a specific parametrized policy directly correlates to both the accuracy of the estimation and the computional cost. The question of the appropriate trade-off should be considered carefully [7].

Unless the set of possible parameters is bounded and small enough to be entirely explored, one must decide on a strategy to choose what parameter values should be fed to the simulations. The different possible selection strategies are called meta-heuristics. Many meta-heuristics have been developed [12, 15]. Simulated annealing [63], tabu search [38], iterated local search [70], particle swarm optimization [16] and genetic algorithms [71] are notable examples.

Simulation based methods have been applied to many inventory control and supply chain optimization. The optimization of a clinical trial supply chain system was conducted using a simulation assisted direct search of optimal safety stock levels, assisted by mixed integer linear programming [19]. Genetic algorithms have been applied with success to network design associated with replenishment policies optimization [26]. A simulation based approach, using a hybrid meta heuristic algorithm was applied to determine the best order-up to levels in a supply chain dealing with highly perishable products [28].

The convergence times of these methods are impacted by the size and the complexity of the search space (which is the parameters space for parametrized policies) [7]. While deep reinforcement learning relies on policies designed with very large number of parameters (the connections weights) and a great expressiveness power, simulation-based optimizations are best utilized when the policies are well crafted with limited numbers of meaningful parameters.

Therefore, despite their aforementioned successes on some supply chain problems, simulation based methods have been held back by the need to craft good parametrized policies in advance. The ability of supply chain practitioners to do so is highly dependent on the type of inventory control problem they may face.

I.7 'Off the shelf' solutions scope of applicability

We divide inventory control problems into two types : problems that require dedicated specific solutions, and problems simple enough that 'off the shelf' solutions can be directly applied. We consider that reorder points methods, linear programming techniques, (stochastic) dynamic programming techniques, and simple reinforcement learning architectures are the four main 'off the shelf' types of solutions for inventory control problems. Non-linear programming, control theory and simulation based approaches all generally require the user to craft specialized heuristics or to dedicate significant amount of work to adapt these methods to the problem.

In table I.5, we recapitulate the scope of applicability of the four main 'off the shelf' methods when it comes to inventory control problems. We restrict the table to single echelon inventory control problems, since in our experience, multi-echelon problems systematically require dedicated development. Classical reorder points methods are efficient for single item stationary stochastic problems, but are inapplicable when the demand is variable or if shared constraints are introduced.

If the problem can be formulated as a linear program, standard linear-programming techniques apply. It is to be noted however that multi-period stochastic systems are in general impossible to frame as linear programs. Standard dynamic programming and its extension to stochastic systems (stochastic dynamic programming) are relevant for inventory control problems, but suffer from the curse of dimensionality on their state and action spaces. For this reason, their applicability depends on the specific structure of the problem. Reinforcement learning techniques, which can be seen as approximate dynamic programming techniques, alleviate some of these limitations, notably on the size of the state space. Multi-item constraints can still be problematic even for reinforcement learning methods, especially when the constraints apply to the action space. State of the art reinforcement learning methods indeed do not scale well on large discrete action spaces.

Methods applicability	Deterministic	2-stage stochastic	n-stage stochastic	Stationary demand	Variable demand	Single item	Multi-item
Reorder points	✓	✓	✓	✓	✗	✓	✗
Linear Programming	✓	✓	✗	✓	✓	✓	✓
Dynamic Programming	✓	✓	✓	✓	✓	(✓)	(✓)
Reinforcement learning	✓	✓	✓	✓	✓	✓	(✓)

Table I.5: Methods scope of applicability on single echelon inventory control problems

In the real world, supply chain practitioners constantly face inventory control problems for which none of these approaches is directly applicable. Typically, one can deduce from table I.5 that multi-period stochastic problems, with constraints shared between several items are especially difficult to solve. The multi-item MOQ problem is an example of such a problem. It is therefore necessary to craft dedicated heuristics and methods to leverage the specific characteristics of the MOQ problem structure.

I.8 The MOQ Problem

I.8.1 Problem definition

Manufacturers and suppliers can often leverage economies of scale to reduce their manufacturing and transportation costs. The mandatory minimum order size is a simple and popular method used by suppliers and manufacturers to avoid unprofitable transactions, as it dilutes the impact of fixed costs they may face. A real world example of such a case is analyzed in Musalem and Dekker [73]. Minimum order constraints shift the burden to the purchasing entity. Different items also often share manufacturing or delivery constraints that lead suppliers to impose multi-item minimum order quantity constraints [114]. These constraints can be set on various economic or physical quantities related to the items such as the total value, volume, or even weight of the order.

In this context, the purchasing entity (the retailer) faces a specific inventory control problem that we call the multi-item Minimum Order Quantity (MOQ) problem. It holds an inventory of one or several items. As in any inventory control systems, an external stochastic demand consumes units held in stock, generating a reward. The retailer needs to find the right balance of inventory stock levels: unsold units generate inventory holding costs, but in case of stock-outs, potential reward is lost. The retailer can replenish its inventory by placing orders to a single source. This

source imposes the minimum order quantity constraint, that is shared across all items. The MOQ constraint can be formally defined as:

$$\sum_{i=1}^{I} a^i \geq Q \text{ or } a^i = 0, \forall i \in [1..I].$$

With Q being the minimum order quantity, I being the number of items, and a^i being the number of ordered units of item i.

The constraint is formulated on the number of units. In practice if the actual constraint is expressed in value or another shared property of the items instead of the number of units, one can easily normalize the considered incremental quantities to contribute equally to the MOQ constraint. We take the example of a retailer who manages an inventory composed of two items. The first item purchase price is 1\$, the second item purchase price is 2\$. The supplier imposes a minimum order value of 10 000\$. The retailer may consider increments of orders of two units for item 1, which brings the value of an increment to 2\$, while considering increments of single units for item 2. The order increments of item 1 and 2 are equally contributing to the minimum order value constraint, which can be seen as a minimum order quantity on these order increments.

We choose not to detail further a specific formulation of the MOQ problem at this point, as the many possible choices of models and notations vary depending on the preferences of the authors. We will present our preferred notations and model (that we based on Lokad's expertise of real world supply chains) in chapter II.

I.8.2 Single item

The single item instance of the MOQ problem is simpler than the multi-item problem, but is still far from trivial. In this section, we get rid of the i indices as only one item is considered. We note x the stock level before the reorder, a the order, and $\hat{x}$ the stock level after receiving the order, but before observing the demand. The MOQ constraint simplifies to $a \geq Q$ or $a = 0$, with a being the order. To simplify notations, the lead time is assumed to be negligible, and one can therefore write $\hat{x} = x + a$.

Continuous stock and order

In [112] Zhao and Katehakis studied the periodic review single item MOQ problem. They introduced the concept of M-increasing (and decreasing) functions, which allowed them to partially characterize the optimal policies. $\phi(x)$ is an M-decreasing function if $\phi(x) \geq \phi(x + M'), \forall x, \forall M' < M$.

We note $V_t(x)$ the value function associated to x and the optimal policy at time t. Zhao and Katehakis demonstrated that $V_t(x)$ is a M-decreasing function, with $M = Q$. To understand this result, we define $\hat{V}_t(\hat{x})$ the value function associated to $\hat{x}$ and the optimal policy. By definition of the optimal policy and the value function:

$$V_t(x) = \max_{a \in 0 \cup [Q, \infty[} \hat{V}_t(x + a)$$

Figure I.8 illustrates how this equation implies that $V_t(x)$ is M-decreasing with $M = Q$. We consider in this example an arbitrary post-order value function $\hat{V}_t(\hat{x})$, represented in light grey, defined on the continuous stock values. The curve of the post-order value function $\hat{V}_t(x + Q)$ is the curve of $\hat{V}_t(\hat{x})$ shifted by Q to the left (being at stock level x and ordering Q units is equivalent to being at stock level $x + Q$ and ordering 0 units). The curves of corresponding to $\hat{V}_t(x + a)$, with $a \in]Q, \infty[$ would be the same curves shifted even further to the left. For clarity, we only displayed the curve $\hat{V}_t(x+Q)$ (darker grey). By definition, the pre-order value function $V_t(x)$ is the maximum

of all the $\hat{V}_t(x+a)$ functions, with $a \in 0 \cup [Q, \infty[$. Intuitively, when in state x, the optimal order is the one that brings us to the best reachable state $\hat{x}$. The curve $V_t(x)$ is represented with the colored dashed line: the differently colored segments indicate whether the optimal action when in state $x_t = x$ is to order Q, more than Q, or nothing. From this figure one may get the intuition that $V_t(x)$ is indeed M-decreasing, with $M = Q$.

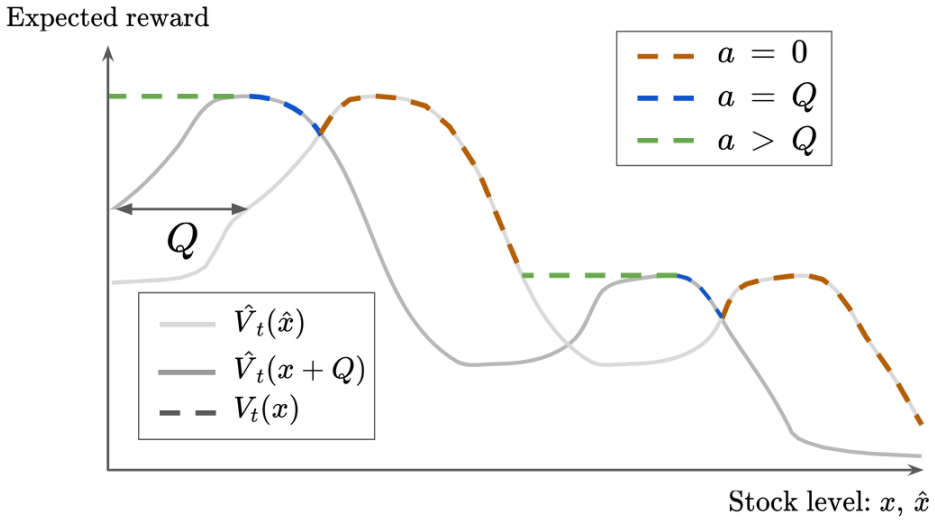


Figure I.8: Q-decreasing pre-order value function $V_t(x)$

Zhao and Katehakis demonstrated several results on the characteristics of the optimal policy. Importantly, they showed that the optimal policy is not necessarily monotone in terms of order quantity. This means that even if $x' > x$, the optimal orders a' may be smaller than a, (with a' being the optimal order when in state x', and a being the optimal order when in state x). The exact optimal policy is therefore still an open question in the continuous case, and the authors suggest the usage of approximate heuristics.

In [83], Robb and Silver suggest such a heuristic based on the idea of safety stocks and reorder point. They define two parameters k and κ, and their policy states that :

$$a = \begin{cases} \mu + k\,\sigma - x & \text{if } x \leq \mu + k\,\sigma - Q, \\ Q & \text{if } \mu + k\,\sigma - Q < x \leq \mu + \kappa\,\sigma, \\ 0 & \text{if } x > \mu + \kappa\,\sigma, \end{cases}$$

With μ being the expected demand over the review period and lead time period, and σ being the standard deviation of the demand forecast over the same period. The authors suggest several heuristics to estimate the parameters k and κ. They study the relative performance of these heuristics through an extensive simulation experiment.

The solution to the continuous single item MOQ problem is still unresolved, despite the existence of accurate heuristics. In practice however, it is rarely necessary to consider continuous values, as discrete state and action spaces are most frequently used in real supply chains.

Discrete stock and order

The MOQ problem can be framed as a stochastic dynamic problem. As such, the optimal control policy can be determined on the single item instance of the MOQ problem under a few assumptions, using the backward recursion algorithm [10] presented in section I.4.2. Firstly, the number of possible stock level and order sizes needs to be bounded and finite. The stock level and orders can therefore be discretized. Secondly, the temporal horizon needs to be finite, as the backward

recursion requires a 'starting point', which is the last period of the temporal horizon. The complexity of the algorithm is proportional to the number of time steps, the number of possible orders, and the number of potential different stock levels. This method provides the optimal policy, but is rather advanced for non specialists, and simpler heuristics have been developed on the single item MOQ problem.

In [115] Zhou, Zhao and Katehakis presented a heuristic called the (s,t) policy, applicable when the demand is stationary. The stock and orders are discretized. The policy is not optimal. The (s,t)-policy, as its name indicates, is based on the two parameters s and t. The intuition of the policy is similar to the idea of the classical reorder point method (s,S). The policy states that the order size a is equal to:

$$a = \begin{cases} s+Q-x & \text{if } x \leq s, \\ Q & \text{if } s < x \leq t, \\ 0 & \text{if } x > t. \end{cases}$$

With x being the pre-order stock level. While not optimal, the results of the (s,t) policy are in practice close to optimality, as demonstrated in the numerical experiments in [115]. The behavior of the (s,t) policy is illustrated in figure I.9.

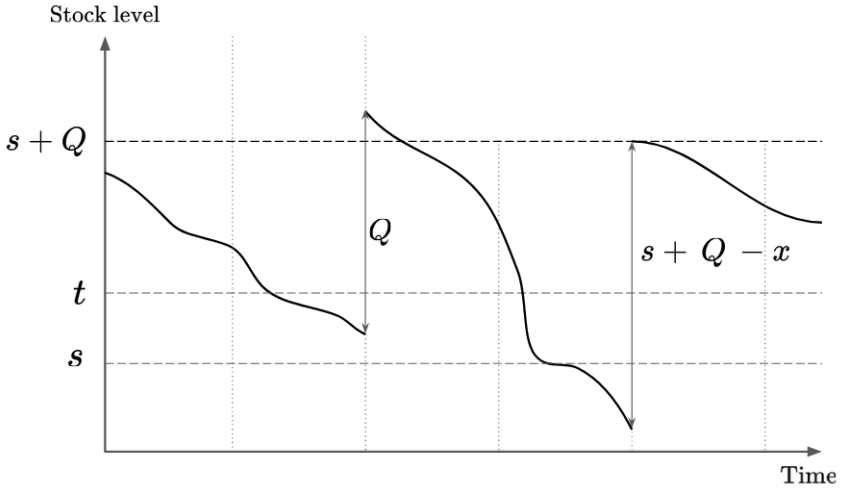


Figure I.9: (s,t) policy

The optimal s and t parameters can be determined thanks to the stationarity hypothesis of the stationary demand. We define $\hat{x} = x + a$. One can note that if the (s,t)-policy is applied, then $\hat{x}$ necessarily belongs to $]t, t+Q]$. One can define the q_i as the long run fraction of time such that $\hat{x} = t+i$. It is the long term probability of ending up with a post order equal to $t+i$. $\bar{q}$ is the steady state probability vector, and is defined as $\bar{q} = \{q_1, q_2, ..., q_Q\}$. One can also define the transition matrix of the system $P_{i,j}$, with each term $p_{i,j}$ of the matrix being the probability of ending up with a post order stock level equal to $t+j$, if the previous post order stock level is $t+i$. By definition of $\bar{q}$, one can write:

$$\begin{aligned} \bar{q} &= \bar{q} P_{i,j}, \\ \sum_{i=1}^{Q} q_i &= 1 \end{aligned} \tag{I.8}$$

$\bar{q}$ can therefore be computed by solving this linear system (equations I.8). The long term average cost of the policy (for a given s and t) can be determined from $\bar{q}$. One then needs to explore the

parameter space of s and t to determine the optimal parameter values. The authors suggest an algorithm to determine the optimal s and t values. Its complexity is proportional to $O(Q^4)$, and utilizes the characteristics (notably the convexity) of the parameter search space.

A simpler single parameter reorder-up to policies has been suggested in [61]. It has a theoretically worse performance than the (s,t) policy, but its parameter computation is significantly faster. The authors advocate for the use of their policy in practice, named the (R,S,Q) policy, as they believe that the ease of implementation and simplicity of their heuristic out-weights the slight performance drop. The (R,S,Q) policy states that the order should be equal to:

$$a = \begin{cases} S - x & \text{if } x \leq S - Q, \\ Q & \text{if } S - Q < x < S \\ 0 & \text{if } x \geq S + Q. \end{cases}$$

The (R,S,Q) is equivalent to a (s,t) policy with t set to S, and s set to $S - Q_{min}$. The authors suggest two methods to estimate the parameter S. The first one computes the optimal S in a similar fashion to the (s,t) policy. The second one is a rather sophisticated rule of thumb, which can be computed in an instant. The performance of the (R,S,Q) policy are compared to the (s,t) policy on simulated benchmarks, on which they appear to be almost identical.

I.8.3 Multi-item stationary demand

The complexity of the problem increases when several items are subjected to the same MOQ, or minimum order value (MOV). In general, constraints that impact several items simultaneously drastically increase the complexity of inventory control models, and the literature rarely provides solutions that scale up to more than a hundred items [60, 76, 81, 82].

In the case of the MOQ problem, if the demand is assumed to be stationary, Zhou [114] developed a lower bound approximation of the expected reward, as well as a heuristic based on it. The heuristic is called the (S,T) policy and is quite similar in its intent to the single-item (s,t) policy [115] (see section I.8.2). The main difference is that the method computes the actions based on the aggregate stocks and demands. We note the aggregate stock $X = \sum_{i \in I} x^i$, with x^i being the stock levels of the items $i \in I$. The (S,T) policy suggest to place an order of total size A:

$$A = \begin{cases} S + Q - X & \text{if } X \leq S, \\ Q & \text{if } S < X \leq T, \\ 0 & \text{if } X > T. \end{cases}$$

The specific quantities to order for each item are determined in a second step, by solving an allocation model based on the minimization of the short-term expected costs. The fact that the short term expected costs is utilized instead of the overall expected costs makes the allocation potentially sub-optimal. The stationarity of the demand however significantly alleviates the effect of this approximation in practice. The parameters S and T are determined similarly to the s and t parameters of the (s,t) policy. The complexity of the policy is not explicitly given in the article, but we expect it to be proportional to $O(Q^4)$.

This method has several limitations. Firstly, it is limited to stationary demand models. Secondly, the policy only considers the aggregate stock level when deciding to place an order. It can lead to obviously wrong behaviors, notably when an item is extremely overstocked, as the aggregate stock level may exceed T due to the sole contribution of the overstocked item. More generally, this method is unable to leverage the difference between a well balanced inventory (where all items roughly have the same stock out probabilities) and an imbalanced one (where some items are overstocked and others are almost stocked-out). The policy therefore tends to delay reorders compared to the optimal

policy, as it is overly confident in the ability of the existing stock to cover the demand. In a real world supply chain, it is reasonable to expect more imbalances in the stocks than expected in the models, may it be due to human errors, forecasting inaccuracies, or other external factor.

I.8.4 Multi-item variable demand

If the action and state spaces are discrete and bounded, the backward recursion algorithm [10] is applicable to the multi-item variable demand instances of the MOQ problem. It computes the optimal policy, but its main limitation is its complexity, which is exponential with the number of item considered, and linear with the number of time steps. This approach is explored with more details in chapter II. In practice, it is possible to apply the backward recursions on the smallest instances of the MOQ problem (up to three items).

To the best of our knowledge, Park, Kim and Shin [76] have provided the only attempt to tackle the larger scale multi-item variable demand MOQ problem. They considered up to fifty items. The authors first provide a solution for the deterministic MOQ problem case: if the demand is not uncertain, the problem can be formulated as an integer program, that can be solved by a dedicated solver. However, if the demand is stochastic the problem cannot be formulated easily into a tractable integer program. This is why the authors suggest to apply the same integer programming solution to the expected average values of demand and stock variables, as if the stochastic demand was actually deterministic and equal to its average. As explained by the authors, this approximation leads to unacceptably low service levels. The system indeed converges as close as possible (given the constraints) to a state where it holds a stock exactly equal to the expected demand, falling out of stock in scenarios where the demand exceeds its expected average value. In order to mitigate this issue, the authors introduce heuristic constraints, the safety factors z (one parameter per item). The safety factors impose a safety stock that is linear with the standard deviation of the demand. This heuristic has a huge impact on the performance of the solution: the fine tuning of these z parameters is essential to provide a decent solution, while no set of parameters can actually ensure optimality. Using this policy, the system for example triggers an order if a single item is close to being stocked out, even if this item stock out stocks are insignificant compared to all the other items holding costs. No indication is provided either on the choice of these parameters values.

Therefore to the best of our knowledge there is currently no algorithm capable of robustly dealing with large scale variable demand multi-item MOQ problems.

Conclusion

In this chapter, we presented the main existing approaches to solve inventory control problems. We illustrated their usage on simple concrete examples. We also assessed their applicability to stochastic, non stationary, and multi-item problems, since our goal is to solve instances of the MOQ problem that have all these characteristics. We concluded that no 'off the shelf' method was directly applicable, which explains the existence of heuristics an methods dedicated to solve even simple instances of the MOQ problem. We finally reviewed these dedicated methods, and concluded that they were not sufficient to deal with the non stationary, multi-item instance of the MOQ problem. This conclusion justifies the need to analyze further the MOQ problem (chapter II), in order to develop new solving methods leveraging its specific characteristics (chapter III and IV).

Chapter II

Analysis of the MOQ problem

Introduction

Chapter II is a detailed analysis of the MOQ problem. This analysis lays the foundations upon which the heuristics presented in chapter III and IV are built. We first define the chosen models and notations used in the rest of this manuscript (section II.1). We then analyze the scope of applicability of the backward recursion algorithm which provides an optimal solution for small scale instances of the problem in section II.2. In sections II.3, II.4 and II.5, we utilize this optimal solution to analyze different aspects of the MOQ problem: the optimal order size, the short term and long term impacts of decisions, and the dependency between items. Finally in section II.6, we consider a simplified version of the MOQ problem, when the system has a single period. We provide an efficient and perfect resolution method for this specific instance. This method lies on several ideas that will be reused for the heuristic methods discussed in chapter III.

II.1 Chosen MOQ problem model and notations

In this section we describe the model of the MOQ problem we aimed to solve in this manuscript. We will also define the notations that will be used in the remainder of this thesis.

II.1.1 Evolution of the system

Firstly, we consider periodic reviews rather than continuous review for our system, as this assumption greatly simplifies actual policy computations, while being realistic (as discussed in chapter I, section I.1.2). The time is therefore divided into time steps of equal duration. These time steps can represent days, weeks, or even months depending on the situation. We choose to bound the number of time steps. We believe that it is unnecessary to consider an infinite number of time step to accurately describe real world problems: no supply chain is everlasting. Instead, we consider finite episodes, with time-steps being integers comprised between 0 and t_{end}. t_{end} is the final stage of the system. This choice of a finite episode duration has the advantage that the system will necessarily generate a finite total reward.

Dealing with continuous state and action spaces is significantly more complex than dealing with discrete state and action spaces, since the former have infinite cardinality even if they are bounded. Therefore, we choose to discretize the orders and stock levels. For the sake of simplicity, we discretize at the unit level, but note that one could discretize the values at a different scale depending on the order of magnitude of the actual values and the desired accuracy. In practice, it does not make sense to consider infinitely large stocks and orders if the economic parameters and the forecasts are defined in a realistic way. We therefore define a maximum realistic stock level s^i_{max}, and a maximum

realistic ordered quantity a^i_{max} for each item $i \in [1..I]$. These bounds should be set considering the forecasts of the demand. The retailer has an available stock s^i_t for each item $i \in [1..I]$ at the beginning of every time-step. We call these the pre-reception stock levels. By definition of s^i_{max}, $s^i_t \in [0..s^i_{max}]$. The action space A of the retailer is the space of possible orders a_t that the retailer can place to a unique supplier to replenish the inventory. The order is delivered l time-steps later, with $l \in \mathbb{N}$ being the duration from the placement of an order to its delivery, also called the lead time. In real supply chains, lead times are often stochastic (as discussed in chapter I section I.1.3). In the vast majority of Lokad's use cases however, taking into account this stochasticity is either unnecessary or overly complicated (as forecasting lead times uncertainty is rather complex). We therefore assume the lead time to be known, deterministic and constant.

The order is composed of an ordered amount of every item $a^i_t \in [0..a^i_{max}]$. The stock on order o^i_t is the ordered quantity of item i that is scheduled to arrive at the beginning of time step t (the quantity ordered by a_{t-l} if $t \geq l$). This quantity will be available to satisfy the demand observed during the time step t.

Both o^i_t and s^i_t are available to be consumed by the stochastic demand observed during the time step t. The forecast of the demand over the time step t for every item is a discrete probability distribution f^i_t. The forecasts are assumed to be perfectly accurate and therefore the demand value $d^i_t \in \mathbb{N}$ is a random variable, which probability distribution is exactly f^i_t. If the stock after the reception of the order (the post-reception stock) is not sufficient to satisfy the demand, the potential sales are considered lost, as there is no back order mechanism. This 'lost-sales' assumption is common in the inventory control literature as discussed in chapter I section I.1.5. We choose the lost sales assumption over the backlogged demand assumption since this model is the one preferred by the vast majority of Lokad's clients. It is moreover an assumption that leads to more complex systems and solutions, and policies that provide good results under this more constraining assumption are likely to be easily adaptable to the backlogged demand case. If the lead time l is greater than zero, the transition equations of the system are:

$$\begin{aligned} s^i_{t+1} &= [s^i_t + o^i_t - d^i_t]^+ \\ o^i_{t+l} &= a^i_t \end{aligned} \tag{II.1}$$

We introduce the intermediate variable $\hat{s}^i_t$ as the stock level after the reception of the order, but before any demand is observed, that we call the post-reception stock level:

$$\hat{s}^i_t = s^i_t + o^i_t$$

Figures II.1 and II.2 illustrate these equations.

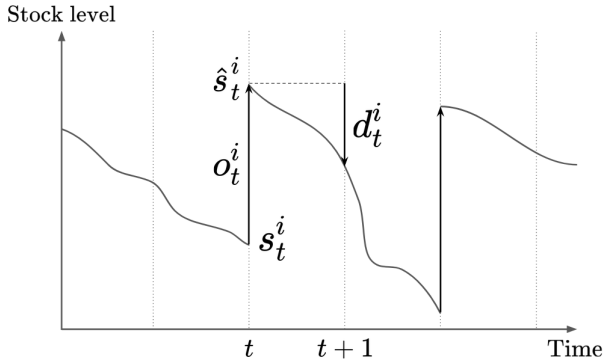


Figure II.1: Stock consumption

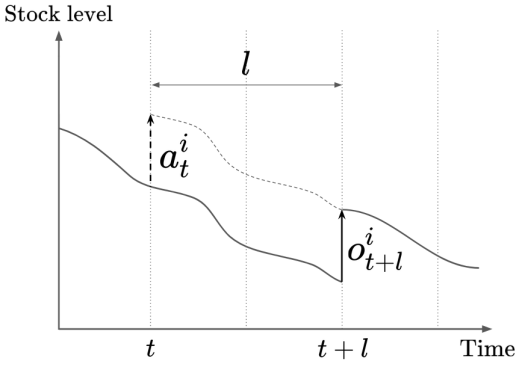


Figure II.2: Order and stock replenishment

II.1.2 State and action space

If there was no MOQ constraint, the size of the action space would be equal to $\prod_{i=1}^{I}(a_{max}^i + 1)$, as each item order quantity could be chosen from $[0..a_{max}^i]$. The action space however is constrained by the MOQ. The retailer must order at least Q units, or order nothing at all ($a_\emptyset$).

$$\sum_{i=1}^{I} a_t^i \geq Q \text{ or } a_t^i = 0, \forall i \in I$$

This constraint can be seen as defining a simplex of dimension I within the action space where orders are not possible, except for $a = a_\emptyset$. For a single item, this simplex is the segment from 1 to $Q - 1$ included. In the two items case, it is the area of the isosceles right triangle, with legs sizes equal to $Q - 1$. An example of the action space in this case is schematized in figure II.3.

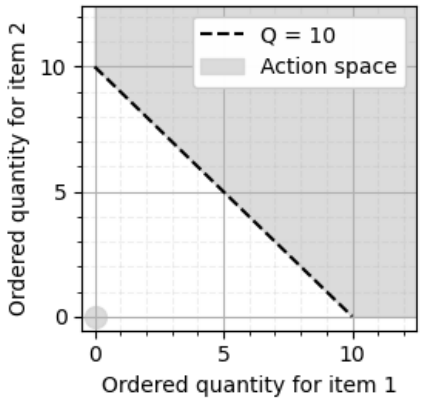


Figure II.3: Action space in a two items scenario

The volume of the simplex of dimension I generated by the 'impossible' orders and the order $a_\emptyset$) is equal to $\frac{(Q-1)^i}{I!}$. Therefore the cardinality of the action space is equal to:

$$Card(A) = 1 - \frac{(Q-1)^i}{I!} + \prod_{i=1}^{I}(a_{max}^i + 1)$$

The state of the system is defined by both the stock levels and the stock on order. We note x_t the state of the system at time t. It can be written as:

$$x_t = \left\{(s_t^i, \{o_{t'}^i\}_{\forall t' \in [t..t+l-1]})\right\}_{\forall i \in [1..I]} \tag{II.2}$$

The stock on order is generated by orders selected from the action space, that have not yet been delivered. The size of the 'stock on order space' is therefore equal to the number of potential orders that have not been delivered (this number is equal to the lead time l) multiplied by the size of the action space. The size of the stock level space is simply equal to $\prod_{i=1}^{I}(s^i_{max}+1)$, as each stock from 0 to s^i_{max} could be observed for every item. The cardinality of the state space X is therefore equal to:

$$Card(X) = Card(A)^l \prod_{i=1}^{I}(s^i_{max}+1)$$

As an example, we consider a MOQ problem with a hundred items, with $s^i_{max} = a^i_{max} = 100$ for all $i \in [1..I]$, a leadtime of three time-steps $l = 3$, and $Q = 1000$. In this example, $Card(A) \approx 10^{200}$, and $Card(X) \approx 10^{800}$.

II.1.3 Costs and rewards

The reward $R^i_t(x^i_t, a^i_t, x^i_{t+1})$ generated during a transition from state x^i_t to x^i_{t+1} for a single item i is equal to the number of units sold multiplied by its margin m^i, minus the number of units remaining in stock multiplied by its holding cost h^i for every item ($m^i > 0$ and $h^i > 0$).

$$R^i_t(x^i_t, a^i_t, x^i_{t+1}) = m^i \min(d^i_t, \hat{s}^i_t) - h^i[\hat{s}^i_t - d^i_t]^+ \tag{II.3}$$

This equation is illustrated with the figures II.4 and II.5, which describe what happens when the demand is smaller or greater than the post-reception stock level respectively. The choice of modelling the holding cost by the remaining stock multiplied by a holding cost parameter h^i (for every item) is classic in supply chain as discussed in chapter I section I.1.4. This holding costs model has therefore been the preferred choice at Lokad for years. The reward generated by a sale includes both the margin associated to this sale and the avoided supplementary shortage costs (see chapter I section I.1.5). This reward is defined as the number of sales multiplied by the parameter m^i for every item.

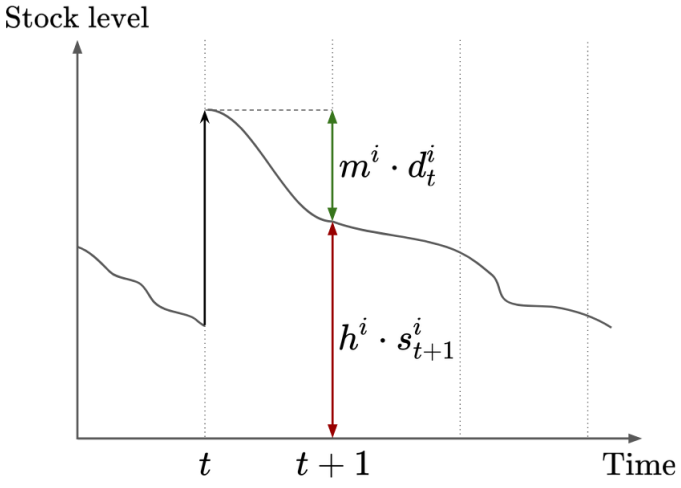


Figure II.4: Reward when $d^i_t \leq \hat{s}^i_t$

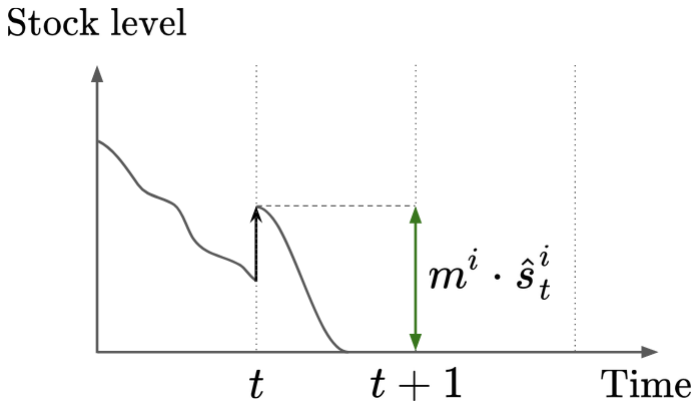


Figure II.5: Reward when $d_t^i > \hat{s}_t^i$

II.1.4 Objective function

The retailer's goal is to find a policy that maximizes the expected gain $\mathbb{E}[G_t]$:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[G_t | a_{t' \geq t} = \pi(x_{t'})] \tag{II.4}$$

$$G_t = \sum_{t'=t}^{t_{end}} \sum_{i=1}^{I} R_t^i(x_t^i, a_t^i, x_{t+1}^i) \tag{II.5}$$

For a given t, d_t^i and $\hat{s}_t^i$ are independent random variables. Since we consider a finite number of time steps, we do not need to introduce an artificial exponential discount rate γ. The discount rate is often used to avoid convergence problems when the horizon is infinite, at the cost of over-valuing short term rewards. We therefore choose not to introduce a discount factor, in order to weight the rewards equally across time. Opportunity costs can be included in the holding cost parameter h.

For clarity, we capitalize the random variables $s_t^i, \hat{s}_t^i, a_t^i, o_t^i, d_t^i$ into $S_t^i, \hat{S}_t^i, A_t^i, O_t^i, D_t^i$ at some points in the manuscript to explicit the fact they are uncertain from the actor point of view, as opposed to a specific outcome of the stochastic system evolution. For example, at time $t \in [0..t_{end}]$, the retailer has observed the pre-reception stock $s_{t'}^i, t' \in [0..t]$, but the future pre-reception stock $S_{t'}^i, t' \in [t..t_{end}]$ are not observed yet and their future actual value is uncertain.

Since $P(\hat{S}_t^i \geq k) = \sum_{\hat{S}_t^i=k}^{s_{max}^i} P(\hat{S}_t^i = k)$, $P(D_t^i \geq k) = \sum_{D_t^i=k}^{\infty} P(D_t^i = k)$, and using the definition of the expected value of a random variable, the expected gain can be re-formulated using sum indexes manipulations as the following expression:

$$\mathbb{E}[G_t] = \sum_{t'=t}^{t_{end}} \sum_{i=1}^{I} \sum_{k=1}^{s_{max}^i} P(\hat{S}_{t'}^i \geq k) \left(P(D_{t'}^i \geq k) m^i - P(D_{t'}^i < k) h^i \right) \tag{II.6}$$

The equation II.6 reflects the idea that any potential stock level has a given probability of being reached or exceeded by the post-reception stock level, and if so, the last unit associated to this potential stock level may be sold or not with a given probability at the end of the time step, generating the corresponding reward. $P(D_{t'}^i \geq k)$ and $P(D_{t'}^i < k)$ are known since we have access to the demand forecasts $f_{t'}^i$.

Equation II.6 demonstration

We first consider the equation II.3, that expresses the reward. We divide the contribution as increments of contributions for each unit sold or still in stock.

$$R_t^i(x_t^i, a_t^i, x_{t+1}^i) = m^i \min(d_t^i, \hat{s}_t^i) - h^i[\hat{s}_t^i - d_t^i]^+ = \sum_{k=1}^{\min(d_t^i, \hat{s}_t^i)} m^i - \sum_{k=d_t^i+1}^{\hat{s}_t^i} h^i$$

If the order policy respects causality (actions probability distributions are independent from the outcome of future stochastic processes), D_t^i and $\hat{S}_t^i$ are independent random variables. Therefore :

$$\mathbb{E}[R_t^i(x_t^i, a_t^i, x_{t+1}^i)] = \sum_{d=0}^{\infty} \sum_{\hat{s}=0}^{s_{max}^i} P(D_t^i = d) P(\hat{S}_t^i = \hat{s}) \left(\sum_{k=1}^{\min(d,\hat{s})} m^i - \sum_{k=d+1}^{\hat{s}} h^i \right)$$

We extend to sums from 0 to s_{max}^i, using the indicator function $\mathbb{1}_{k \leq y}$, that is equal to 1 if $k \leq y$ and to 0 otherwise.

$$= \sum_{d=0}^{\infty} \sum_{\hat{s}=0}^{s_{max}^i} P(D_t^i = d) P(\hat{S}_t^i = \hat{s}) \left(\sum_{k=1}^{s_{max}^i} \mathbb{1}_{k \leq d}\, \mathbb{1}_{k \leq \hat{s}}\, m^i - \sum_{k=1}^{s_{max}^i} \mathbb{1}_{k > d}\, \mathbb{1}_{k \leq \hat{s}}\, h^i \right)$$

We then proceed to reduce the sums over d and $\hat{s}$, by moving the sum over k to be the 'outer' summation.

$$= \sum_{k=1}^{s_{max}^i} \left(\sum_{d=0}^{\infty} P(D_t^i = d) \mathbb{1}_{k \leq d} \sum_{\hat{s}=0}^{s_{max}^i} P(\hat{S}_t^i = \hat{s}) \mathbb{1}_{k \leq \hat{s}}\, m^i - \sum_{d=0}^{\infty} P(D_t^i = d) \mathbb{1}_{k > d} \sum_{\hat{s}=0}^{s_{max}^i} P(\hat{S}_t^i = \hat{s}) \mathbb{1}_{k \leq \hat{s}}\, h^i \right)$$

$$= \sum_{k=1}^{s_{max}^i} \left(\sum_{d=k}^{\infty} P(D_t^i = d)\, m^i \sum_{\hat{s}=k}^{s_{max}^i} P(\hat{S}_t^i = \hat{s}) - \sum_{d=0}^{k-1} P(D_t^i = d)\, h^i \sum_{\hat{s}=k}^{s_{max}^i} P(\hat{S}_t^i = \hat{s}) \right)$$

We then use the fact that $\sum_{\hat{s}=k}^{s_{max}^i} P(\hat{S}_t^i = \hat{s}) = P(\hat{S}_t^i \geq k)$.

$$= \sum_{k=1}^{s_{max}^i} \left(P(\hat{S}_t^i \geq k)\, m^i \sum_{d=k}^{\infty} P(D_t^i = d) - P(\hat{S}_t^i \geq k)\, h^i \sum_{d=0}^{k-1} P(D_t^i = d) \right)$$

Similarly, we use the fact that $\sum_{d=k}^{\infty} P(D_t^i = d) = P(D_t^i \geq k)$ and $\sum_{d=0}^{k-1} P(D_t^i = d) = P(D_t^i < k)$.

$$= \sum_{k=1}^{s_{max}^i} P(\hat{S}_t^i \geq k) \left(P(D_t^i \geq k)\, m^i - P(D_t^i < k)\, h^i \right)$$

By combining this last equation with equation II.3, we demonstrate that:

$$\mathbb{E}[G_t] = \sum_{t'=t}^{t_{end}} \sum_{i=1}^{I} \sum_{j=1}^{s_{max}^i} P(\hat{S}_t^i \geq j) \left(P(D_t^i \geq j)\, m^i + P(D_t^i < j)\, h^i \right)$$

II.1.5 Conclusion and notations summary

In this section we detailed the equations of the MOQ problem, and we reformulated the objective function as a a triple sum of contributions, over the time steps, over the items, and over the potential units in stock. We will utilize this formulation of the objective function in the rest of the chapter to analyze the MOQ problem. We summarize the notations introduced in this section.

The system constants:

- I is the number of different items.
- t_{end} is the last period of the episode, $t_{end} + 1$ is the number of time steps.
- Q is the minimum order size.
- l is the lead time, the duration between the placement and the reception of orders.
- m^i is the margin parameter, the reward generated by the sale of one unit of item $i \in [1..I]$ (it can be seen as an avoided shortage cost).
- h^i is the holding cost parameter, the reward lost per unit of item $i \in [1..I]$ held in stock at the end of a period.
- s^i_{max} is the maximum realistically considered stock for item i.
- a^i_{max} is the maximum realistically considered order quantity for item i.

The system variables:

- s^i_t is the pre-reception stock level of item i at time t.
- $\hat{s}^i_t$ is the post-reception stock level of item i at time t.
- a^i_t is the quantity ordered at time t of item i (received at $t + l$).
- o^i_t is the quantity received at time t of item i (ordered at $t - l$).
- d^i_t is the demand at time t of item i.
- x_t is the state of the system at time t.

II.2 Optimal solution on small scale examples

The backward recursion algorithm described in chapter 1, section I.4 is applicable to this instance of the MOQ problem, since the action and state spaces are discrete. Having access to the optimal policy is very valuable to better understand the MOQ problem and evaluate the different policies. The limitation of the backward recursion is its complexity C, which is exponential with the number of considered item, and linear with the number of time steps:

$$C \sim Card(X)\, Card(A)\, (t_{end} - t)$$

In practice, it is only possible to apply the backward recursions on the smallest instances of the MOQ problem. Figure II.6 plots the CPU time of the computation for increasing numbers of items. The maximum considered stocks s^i_{max} were set to 20, 50, and 100. The lead time l was set to zero to reduce the state space size. The experiments were conducted on a Intel Core i7-7700HQ @ 2.80GHz CPU.

As expected, the computation duration increases with s^i_{max} and the number of items. In practice, problems dealing with large quantities can be discretized into batches: instead of considering stock and demand increments of single units, one can consider increments that represent a batch of units. s^i_{max} can therefore represent maximum number of batches. Accuracy decreases when s^i_{max} decreases and the batch sizes have to be increased. Figure II.6 shows that unless the stock level are discretized into only twenty increments, it is expensive in practice to compute the solution for instances larger than two items. In the rest of this chapter we therefore limit our experiments to one or two-items instances of the problem.

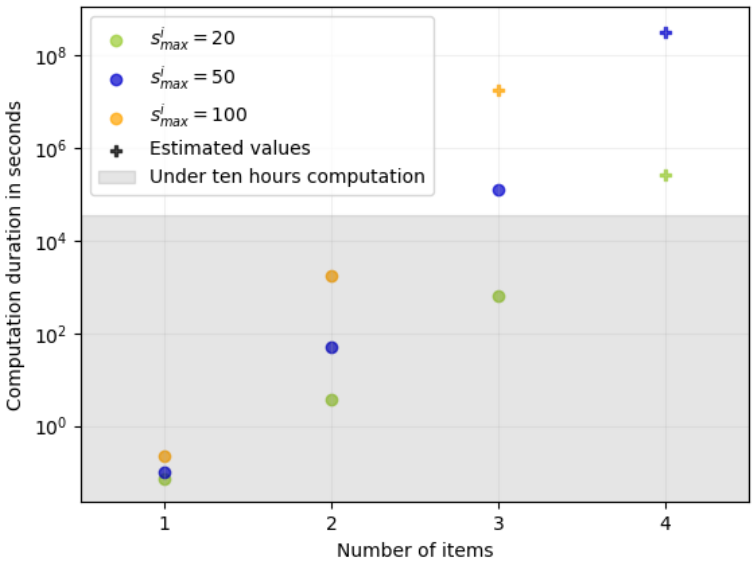


Figure II.6: Backward recursion algorithm computation times

The backward recursion provides the optimal policy by explicitly computing the state and action value functions associated to every possible states and actions. These functions were already introduced in section I.4.2, that introduced stochastic dynamic programming. They are useful to estimate the economic cost of taking sub-optimal action, under the assumption that all the subsequent actions will be optimal. The function $A_t(x, a)$ that represents this cost is the difference between the action value and the value functions $q^{\pi*}_t(x, a) - v^{\pi*}_t(x)$, and is generally referred to as the advantage function (with π^* being the optimal policy). The advantage function can be seen as the expected additional loss induced by taking an action. The advantage function value of the optimal order is therefore equal to zero, and it is the maximum of the advantage function. $A_t(x, a) \leq 0$ (for a being a legal action).

$$A_t(x, a) = q^{\pi*}_t(x, a) - v^{\pi*}_t(x) = \mathbb{E}[G_t | a_t = a, a_{t'>t} = \pi^*(x_{t'})] - \mathbb{E}[G_t | a_{t'\geq t} = \pi^*(x_{t'})]$$

Figure II.7 plots an example of an advantage function at $t = 0$, for a single item instance of the MOQ problem. The demand is stationary and its distribution is a Poisson distribution, of parameter $\lambda = 5$. The economic parameters are set to $m = 1, h = 0.1$. The total number of time-step of the episode is 50. The leadtime l and the initial stock are set to zero. In this situation, the optimal quantity to order would be 7 units, but it would not satisfy the MOQ. The optimal action is therefore to order 10 units.

The figure II.8 displays another example, this time with two identical items, with similar settings than for figure II.7, as $h^1 = h^2 = 0.1, m^1 = m^2 = 1, Q = 10, l = 0$. The demand is a Poisson distribution for both items, with parameters $\lambda^1 = \lambda^2 = 2.5$. Both items have empty stocks initially.

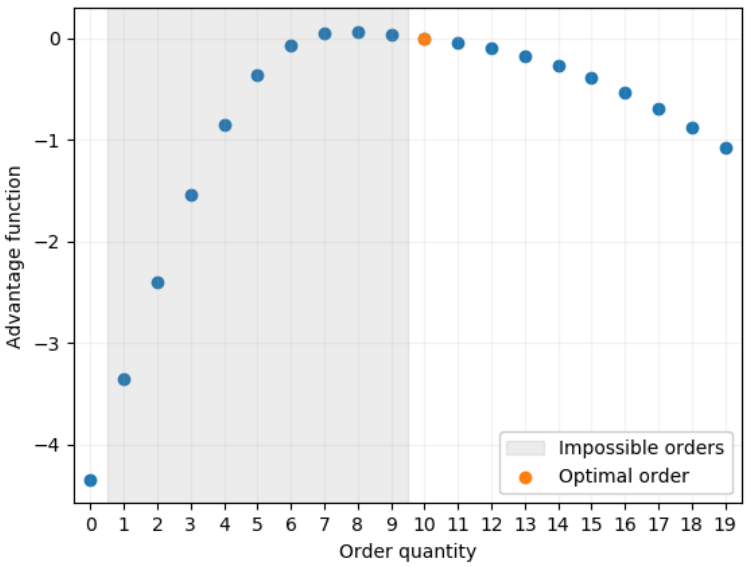


Figure II.7: Advantage function for a Poisson demand distribution, $\lambda = 5$, $Q = 10$

The advantage function is symmetrical along the $y = x$ axis since the two items are strictly identical. The optimal action in this case is to order 5 units of each item.

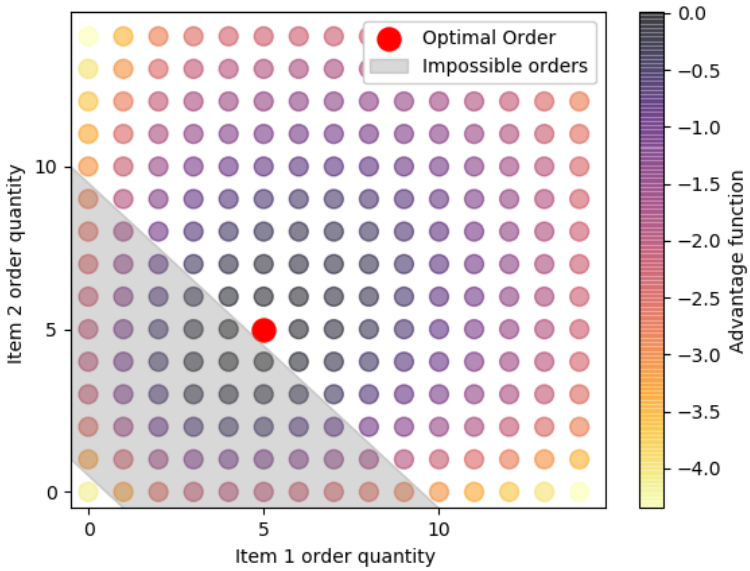


Figure II.8: Heat map of the advantage function for a two items scenario, $Q = 10$

II.3 Analysis of the optimal order size

In this section we analyze the size of the optimal orders. Specifically, we will study the conditions under which the optimal total order size equals Q. If the conditions are met so that the order size is always equal to zero or Q, it simplifies the problem as the optimal order is easier to determine (as the space of eligible orders is restricted).

Intuitively, one could expect the optimal order size to always be equal to Q or zero, so that the expected holding costs are reduced: if one orders more than Q, the duration between orders will probably increase, and the average stock levels are going to be higher. There are however two situations where the optimal order size exceeds Q. Firstly, if the demand is greater than the MOQ and the stock on hand, ordering only Q generates high risk of stock out right at the end of the first period after the order has been received. In this situation, the maximum of the action value function is greater than Q and the MOQ is not actually restrictive.

We illustrated this situation with a simple experiment. We considered a single item, with a stationary demand probability distribution equal to a Poisson distribution of parameter λ at every time step. We set $l = 0, m = 1, h = 0.1$. The initial stock is equal to zero. We considered two scenarios : we set two different values for λ, 11 and 3. We set $Q = 10$. The figure II.9 displays the advantage function, assuming that the future decisions will be taken optimally. As expected, the maximum of the advantage function is greater than Q in scenario 1, and the optimal action is to order 15 units. In scenario 2, the optimal action is to order 10 units, as the MOQ constraint prevents us from ordering only 6 units.

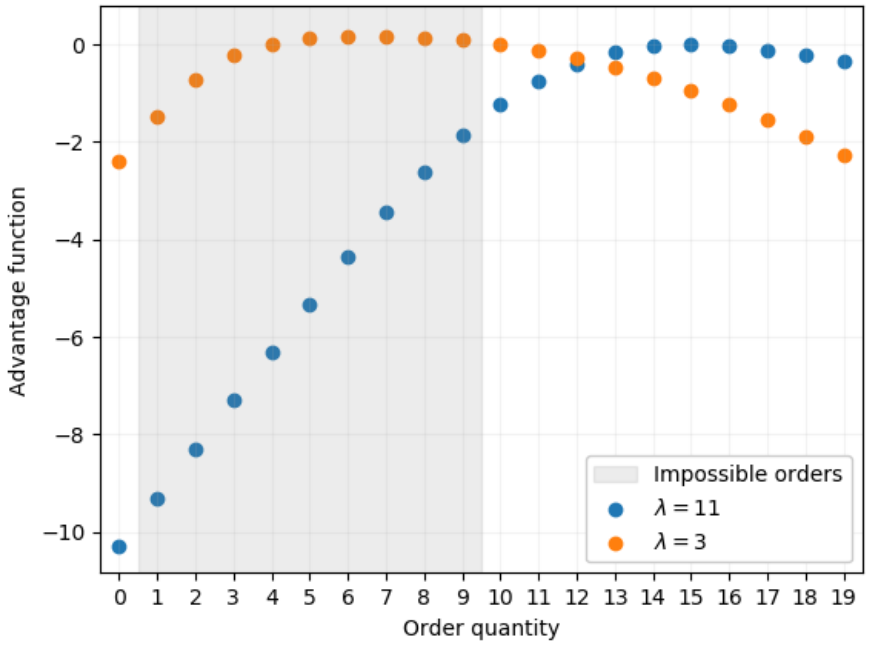


Figure II.9: Advantage function for λ equal to 11 and 3, $Q = 10$

We then looked into the optimal order size for various values of λ, with the same settings. The results are displayed in figure II.10. We can see that there is a cutoff point (equal to 7 in this case) above which it is worth ordering more than Q. Therefore, one can conclude that if the demand per time step is large enough compared to Q, the optimal order size may exceed Q.

The second situation is if the demand is deterministic or almost certain. If the demand is deterministic, the expected reward equation II.6 becomes:

$$\mathbb{E}[G_t] = \mathbb{E}\left[\sum_{t'=t}^{t_{end}} \sum_{i=1}^{I} m^i \min(d^i_{t'}, \hat{s}^i_{t'}) - h^i[\hat{s}^i_{t'} - d^i_{t'}]^+\right] \tag{II.7}$$

The equations II.1 (detailed at the beginning of this chapter) explicit how the orders $o^i_{t'}$ impact the stocks $\hat{s}^i_{t'}$. If the demand is deterministic, it is reasonable to expect the retailer to be able to

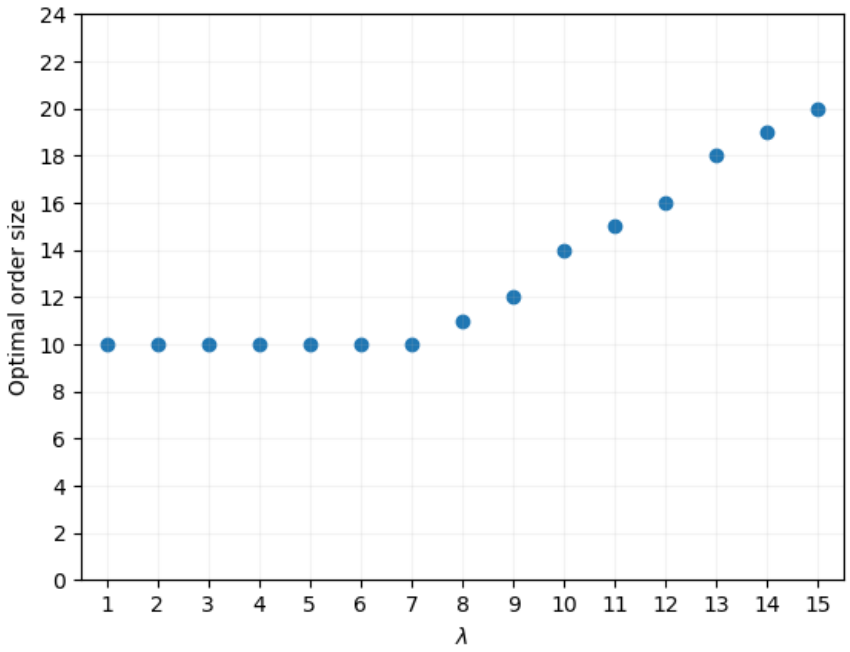


Figure II.10: Optimal order size depending on λ, $Q = 10$

choose orders $o^i_{t'}$ that completely avoid stock-outs (under the assumption that stock outs are significantly more costly than holding costs). In this situation, by immediate recursion from equations II.1, one can express the stock level as :

$$s^i_{t'} = s^i_t + \sum_{t''=t}^{t'-1} (o^i_{t''} - d^i_{t''}) \tag{II.8}$$

Under this assumption, minimizing $\mathbb{E}[G_t]$ is equivalent to minimizing the average $s^i_{t'}$ according to equation II.7. As a side note, in many scenarios minimizing $\mathbb{E}[G_t]$ can be done by setting the stock levels $s^i_{t'}$ to exactly zero as frequently as possible. One can set the stock level $s^i_{t'}$ to zero by placing the following order $o^i_{t'-1}$ (according to equation II.8):

$$o^i_{t'-1} = d_{t'-1} - s^i_t + \sum_{t''=t}^{t'-2} (d^i_{t''} - o^i_{t''}) \tag{II.9}$$

The idea of equation II.9 is that the retailer can safely order just enough to end with zero stock right before receiving the next order, without risking stock outs. Since the demand is deterministic, the retailer can take all the reorder decisions in advance, and the timing of the next reorder is known in advance with certainty.

To illustrate this, we again considered a single item problem, with $Q = 8, m = 1, h = 0.1, l = 0$, an empty initial stock, and a stationary deterministic demand equal to 5 per time step. In figure II.11, we plotted the advantage function (the expected difference of reward to the optimal for every order). One can see the local minimums in 5, 10 and 15, that correspond to the observed demand after 1, 2, and 3 time steps respectively : If the retailer orders 8 or 9 units, the next order should be placed one step later (at $t + 1$) to avoid stock outs. If he orders between 10 and 14 units, the next order should ideally be placed at $t + 2$, if he orders between 15 and 19 units the next order should ideally be placed at $t + 3$. For this reason orders of size 10 and 15 correspond to local maximums

in the expected reward, as they minimize the number of remaining units at the end of the last time step before reordering. The optimal order subjected to the MOQ is 10 in this situation.

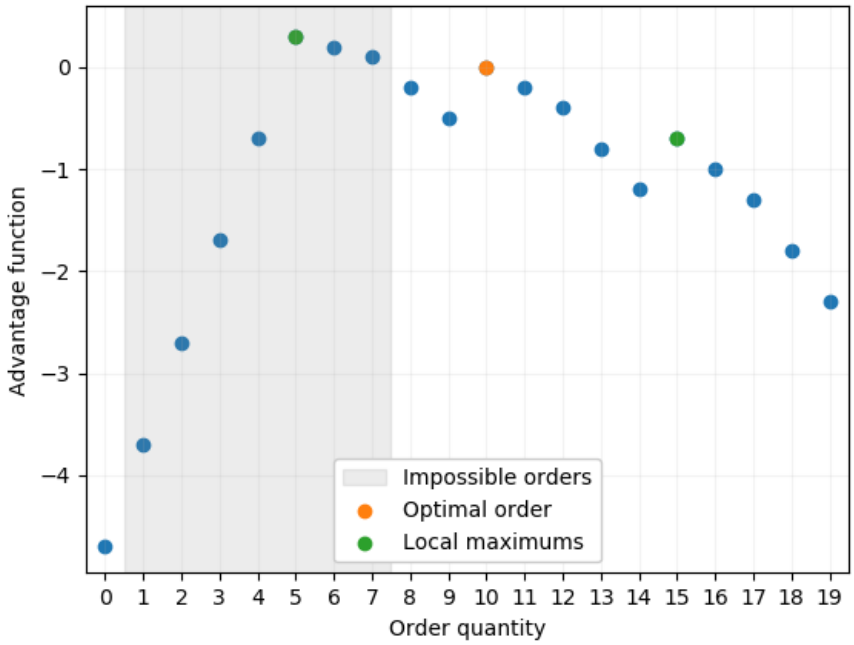


Figure II.11: Advantage function, deterministic demand equal to 5

While this type of logic applies to a deterministic demand, if the demand is actually stochastic these local maximums in the action value function will probably get smoothened, as risks of stock-outs will prevent the actor from staying at low stock levels. In the equation II.6, the term $P(D_t^i \geq k)\, m^i$ incites the retailer to set the post-reception stock levels $\hat{s}_t^i$ to greater values than $\mathbb{E}[d_t^i]$ if $P(D_t^i \geq k)$ is not negligible for small k values.

One can vizualize how the stochasticity smooths the action value function in figure II.12. In the same settings as the previous experiments, the demand is now considered stochastic and equal to a binomial distribution, with a mean equal to 5. The value functions are drawn for two values of standard deviation ($\sigma = 0$ is the deterministic case). In these settings, even with small standard deviation values, the optimal order size is equal to Q.

The exact theoretical formulation of the conditions under which the optimal order size is equal to Q are however unclear, and arguably very complex. Even for the simpler single item instance of the MOQ problem, the question of the optimal order size is unresolved [112,115]. For practical purposes, we believe that unless the demand is deterministic or large enough compared to Q, assuming that the optimal order size is equal to Q is a reasonable approximation. This conclusion helps reducing the complexity of practical problem resolution, as these conditions are usually satisfied on real life instances of the MOQ problem.

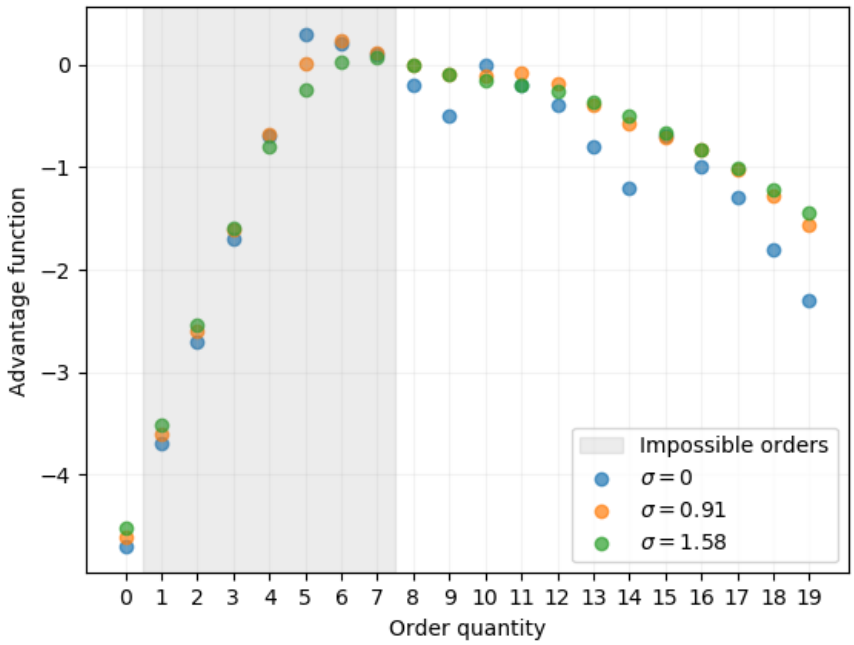


Figure II.12: Advantage function, stochastic demand

II.4 Impact of a decision over time

In this section, we study the impact of sub-optimal orders on the expected reward. It is interesting to understand how the difference of expected reward (compared to taking the optimal order) accumulates over time, and at which point in time the decision taken at time t stops affecting the system.

Formally, the advantage function can be re-formulated as a sum of expected difference of reward over time :

$$A_t(x,a) = \sum_{t'=t}^{t_{end}} \mathbb{E}[R_{t'}|x_t = x, a_t = a, a_{t''>t} = \pi^*(x_{t''})] - \mathbb{E}[R_{t'}|x_t = x, a_{t''\geq t} = \pi^*(x_{t''})]$$

We set up a simple experiment, with two items. Their margin and holding cost parameters are identical ($m^1 = m^2 = 1, h^1 = h^2 = 0.1$), as well as their forecast, which is stationary and equal to Poisson distribution with parameters $\lambda^1 = \lambda^2 = 2.5$. We set $Q = 14$. The initial stocks are set to zero.

We then compare two scenarios: in the first scenario, we place the optimal order $a^1 = 7, a^2 = 7$, and keep applying the optimal policy subsequently. In the second scenario, we place the sub-optimal order $a^1 = 10, a^2 = 4$, and still apply the optimal policy subsequently. It is a way to isolate the impact of a single sub-optimal decision. The figure II.13 represents the expected difference of reward at every time step between the two scenarios. This figure was generated by running both scenarios ten thousand times, and by averaging the reward at each time step over all the runs. One can see that the impact of the sub-optimal decision decreases over time. The difference of reward is mostly generated before time step 3. In scenario 2, the sub-optimal order induces both higher holding costs for item 1, and additional risk of stock out for item 2. It makes the retailer reorder sooner than it would have with scenario 1, or to undergo bigger stock out risks. On average, this induced an important loss of overall expected reward during the first few time steps.

However, as time advances and stock decrease, the actor eventually decides to place a new order.

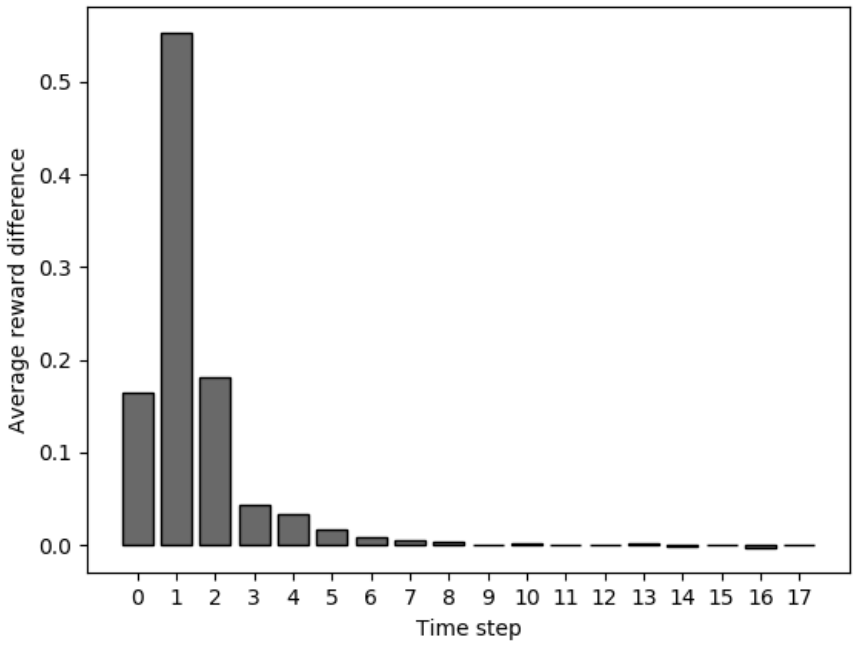


Figure II.13: Expected difference of reward between an optimal and a sub-optimal scenario.

This order is able to 'correct' the imbalance generated by the sub-optimal order. Intuitively, if we order too many units of an item (item 1 in this example), the subsequent orders will tend to order less of this item to compensate. On the contrary, if an item is stocked-out or close to be (item 2 in this example), it will be prioritized in the subsequent orders. The difference in expected reward therefore drastically decreases once the system has the opportunity to re-balance the inventory by placing an order.

We define the inaction window as the duration between the current order and the next non-zero order the retailer will place. The inaction window depends on the states of every item, on the order placed at time t, on the policy applied at future time steps, and on the outcome of the stochastic demand process. When applying the optimal policy, an order is placed if (and only if) $q(x, a_+)$ is greater than $q(x, a_\emptyset)$, with a_+ being defined as the best non empty order, and A^+ being the corresponding subset action space (such as $A = A^+ \cup a_\emptyset$):

$$a_+ = \arg\max_{a \in A^+} q(x, a)$$

In this case, the stock levels decrease over time, until $q(x, a_+) > q(x, a_\emptyset)$. If the observed demand is significantly greater than the expected demand, the cut-off point will occur sooner, the new order will be placed sooner, and the inaction window will be shorter. On the other hand, if the observed demand is lower than expected, one might not need to reorder for longer than expected, and the inaction window will be longer.

The figure II.14 displays the inaction window probability distributions for scenario 1 and 2. As expected, when the order is not optimal, the system tends to reorder sooner. One can note how the inaction window probability mass function is spread on several possible values because of the stochasticity of the demand. When placing the order at time 0, the retailer does not know for sure if he will reorder at timestep 1 or at time step 4.

In this section we introduced the concept of inaction window, defined as the duration between the current order and the next order. The inaction window is a helpful concept to intuitively

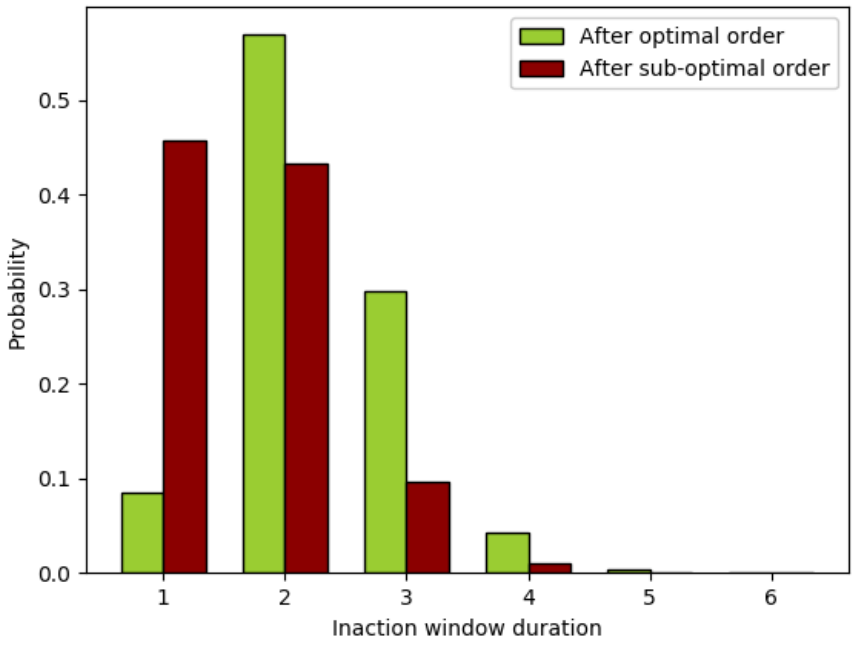


Figure II.14: Inaction windows duration probability distributions

understand how an action impacts the system over time, and how subsequent orders can mitigate the long term effects of a sub optimal decision.

II.5 Vizualizing the interdependence between items

The MOQ restricts the possibility to reorder every item independently whenever needed. Instead, orders must be placed simultaneously for all items to be able to reach the MOQ. The dependency between the different items in the MOQ problem is at the root of a large part of the complexity of the problem. Since items can not be considered independently, the size of the state and action spaces are subjected to the curse of dimensionality. This section's goal is to provide some illustrations of this dependency, in order to help the reader better understand the MOQ problem challenges.

Formally, the advantage function can be re-formulated as a sum of expected differences of the items contribution to the reward :

$$A_t(x,a) = \sum_{i=1}^{I} \sum_{t'=t}^{t_{end}} \mathbb{E}[R^i_{t'}|x_t = x, a_t = a, a_{t''>t} = \pi^*(x_{t''})] - \mathbb{E}[R^i_{t'}|x_t = x, a_{t''\geq t} = \pi^*(x_{t''})]$$

We define the contribution of item i to the advantage function C^i_t as:

$$C^i_t(x,a) = \sum_{t'=t}^{t_{end}} \mathbb{E}[R^i_{t'}|x_t = x, a_t = a, a_{t''>t} = \pi^*(x_{t''})] - \mathbb{E}[R^i_{t'}|x_t = x, a_{t''\geq t} = \pi^*(x_{t''})]$$

This way, one can write

$$A_t(x,a) = \sum_{i=1}^{I} C^i_t(x,a)$$

.

We illustrate the items dependency with a simple experiment. We consider two items, with the following economic parameters $h^1 = h^2 = 0.1, m^1 = m^2 = 1$. We set $Q = 10, l = 0$. The demand is a Poisson distribution for both items, with parameters $\lambda^1 = \lambda^2 = 2.5$. Both items have empty stocks initially. The two components C^1 and C^2 can be computed via backward recursion, similarly to the computation of the 'aggregated' value functions. The figure II.16 presents the two expected items contributions C^1 and C^2 to the advantage function for every possible action, while figure II.15 displays the advantage function (the sum of the two contributions). One can verify that an item contribution to the expected reward is impacted by the other item order: if it was not the case, all the lines on the left figure and the columns on the right figure would be identical.

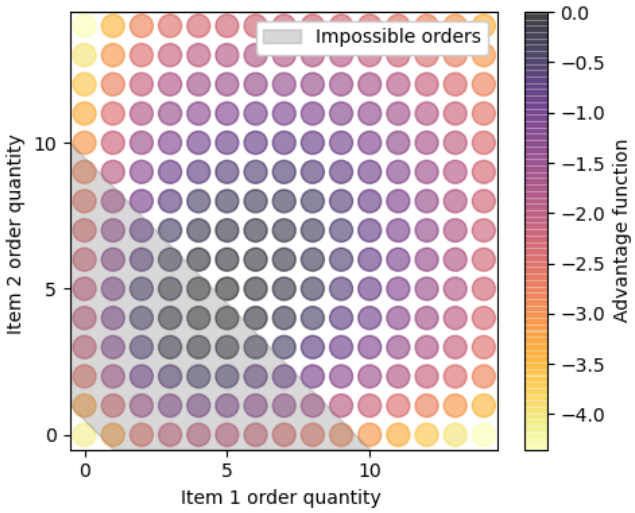


Figure II.15: Heat map of the advantage function for a two items scenario, $Q = 10$

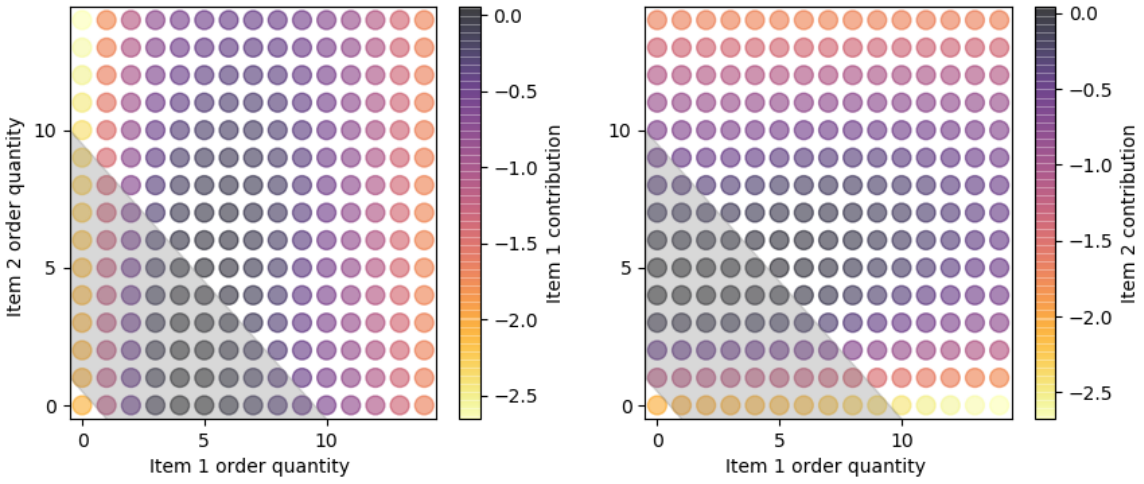


Figure II.16: Heat map of the per-item contribution to the advantage function

We then focus on the item 1 contribution, and how this contribution is affected by the ordered quantity of item 2. The figure II.17 displays the items 1 contribution advantage function depending on a^1, for several different fixed a^2 values (one value per color blue/orange/green/red). In practice, only the values $a^1 \geq Q - a^2$ can be ordered.

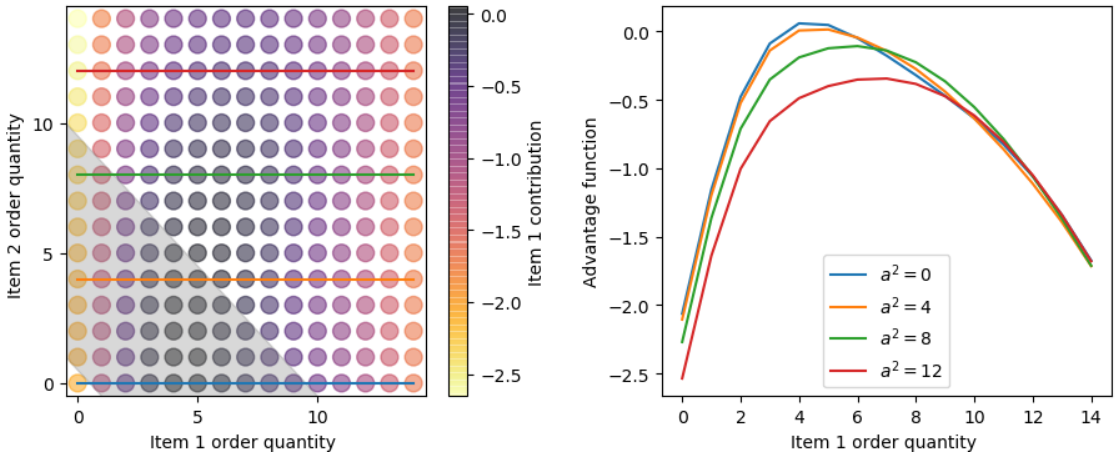


Figure II.17: Item 2 impact on item 1 contribution to the advantage function

As expected, this plot suggests that the item 1 ordered quantity has a much larger impact on items 1 contribution to the expected reward than item 2 ordered quantity. This is due to to the fact that the formula for C_t^1 explicitly depends on a_t^1 according to equation II.3, while a_t^2 only impacts C_t^1 via the constraint that $a_{t'>t}^1 \geq Q - a_{t'>t}^2$.

In this example, the assumptions for the non zero optimal order to be equal to Q are satisfied. Therefore, we know that the non zero optimal order is on the line defined by $a^2 = Q - a^1$ (black line on figure II.18). The figure II.18 displays the separate contributions of the two items for every possible order combination (given $a^1 + a^2 = Q$), as well as the sum of the contributions. The optimal order is in this case equal to $a^1 = a^2 = 5$.

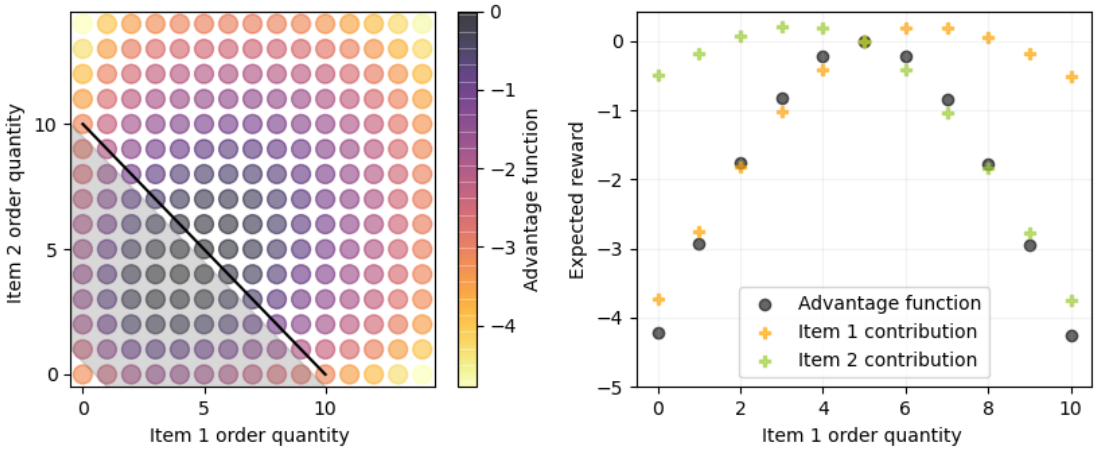


Figure II.18: Advantage function if the order size equals Q

The conclusion of this section is that the advantage function can be expressed as the sum of the items contributions $C^i(x, a)$, and that these contributions C^i depend on the other items stock levels (and stock on order) due to the MOQ constraint.

II.6 Single period problem analysis

In this section we study the single period multi item version of the MOQ problem. Since it is much simpler, this version can be solved optimally, even for a great number of items. This study highlights some key ideas that helped us to build heuristic policies for the more complex multi-period version of the problem.

In the single period version of the MOQ problem, a single time step is considered, with for each item an initial pre-reception stock level s_0^i, a final stock level s_1^i, a demand forecast f^i, and an order a^i which arrives instantaneously ($l = 0$). The state initial state x_0 is only composed of the pre-reception stock variables s_0^i. The reward is generated in a single step. The goal is to find the action a_* that maximizes the reward:

$$a_* = \arg\max_{a' \in A} \mathbb{E}[G_0 | a = a'] \tag{II.10}$$

With $A = \{a_\emptyset\} \cup \{a = \{a_i\}_{i \in [1..I]}, \sum_{i=1}^{I} a^i \geq Q\}$, and with the gain G_0 equal to:

$$G_0 = \sum_{i=1}^{I} m^i \min(D^i, s_0^i + a^i) - h^i s_1^i = \sum_{i=1}^{I} m^i \min(D^i, s_0^i + a^i) - h^i [s_0^i + a^i - D^i]^+ \tag{II.11}$$

The single period problem can be re-formulated as a two-stage stochastic integer linear program, and as such there exists a deterministic equivalent integer linear program [57]. While any linear programming solver provides a satisfactory solution, we present in this section one resolution method with a specific approach, that will introduce the concepts behind the w-policy presented in chapter 3, which aims to solve the multi-period problem.

The optimal policy is to select a^* so that $a^* = \arg\max_{a \in A} q(x_0, a)$. One could compute $q(x_0, a)$ for every possible $a \in A$. However as the number of items increases, the size of A increases exponentially, and so does the computation cost. One can derive from equation II.11 that in the single period case (on the contrary to the multi-period problem), the reward R_0^i generated by an item i is dependent only on its own pre-reception stock level s_0^i and stock on order a_0^i. The action value function can therefore be decomposed into I independent terms:

$$q(x_0, a) = \sum_{i=1}^{I} q^i(s_0^i, a^i)$$

$$q^i(s_0^i, a^i) = P(D^i > s_0^i + a^i)(s_0^i + a^i)m^i + \sum_{d=0}^{s_0^i + a^i} P(D^i = d)(m^i\, d - h^i(s_0^i + a^i - d))$$

The probabilities associated to the demands $P(d^i = d)$ are known since we have access to the demand forecasts f_0^i.

The figure II.19 plots an example of these separate components of the action value function, in a two items case. In this example, we set $m^1 = 20, m^2 = 10, h^1 = 2, h^2 = 1, l = 0, Q = 14$. The demands are assumed to be stationary and to follow Poisson distributions of parameters $\lambda^1 = 4$ and $\lambda^2 = 1$, and the initial stock levels are set to zero.

If we abstractly represent our stock as a First-In First-Out (FIFO) system, we can consider and label individual units incrementally, as if they were consumed and added to the stock one by one. The overall reward can then be considered as the sum of the rewards generated by the individual units, using sum indexes manipulations similarly to equation II.6. The reward $\delta q^i(k)$ associated to a unit k is the probability of selling at least k-units (of item i) multiplied by the reward generated

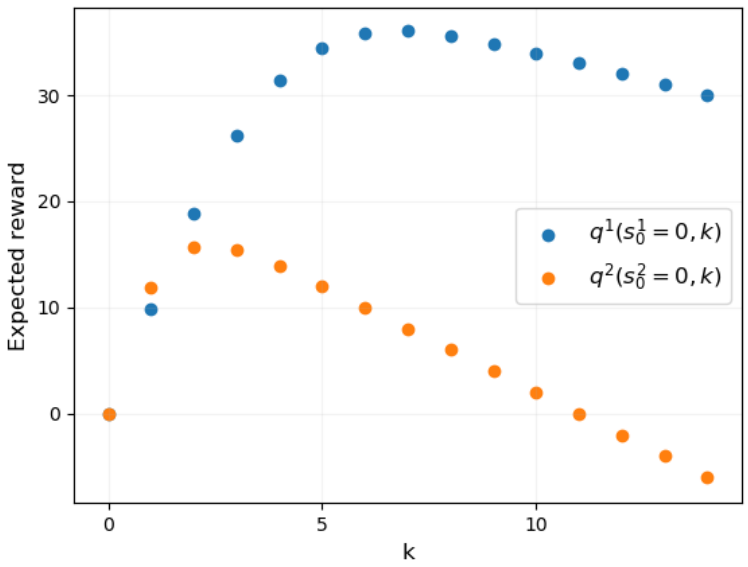


Figure II.19: Example of the components of the action value function in a two items scenario

by the sale of one unit, minus the holding cost of one unit multiplied by the probability of selling less than k units.

$$q^i(s_0^i, a^i) = \sum_{k=1}^{s_0^i + a^i} \delta q^i(k)$$

$$\delta q^i(k) = P(D^i \geq k)\, m^i - P(D^i < k)\, h^i \tag{II.12}$$

All units labeled with $k \in [1..s_0^i]$ represent units already in stock before the order. If $k \in [s_0^i + 1..s_0^i + a_{max}^i]$, the associated unit is a potential additionally ordered unit. If $\delta q^i(k) < 0$ and $k > s_0^i$, it means that ordering this additional unit will decrease the expected reward, and it is not worth adding it to the stock. Let $a_{opt}^i \in [1..a_{max}^i]$ be the greatest a^i so that $\delta q^i(s_0^i + a^i) > 0$. If it does not exist, we set $a_{opt}^i = 0$. Since $\delta q^i(k)$ is a decreasing function in k, by definition a_{opt}^i is the order of item i that maximizes $q^i(s_0^i, a^i)$ if there is no MOQ constraint involved.

If $\sum_{i=1}^{I} a_{opt}^i \geq Q$, then the optimal action is $a_* = \prod_{i=1}^{I} \{a_{opt}^i\}$. If $\sum_{i=1}^{I} a_{opt}^i < Q$, the best action a_Q of size exactly Q can be built using the $\delta q^i(k)$ values and the algorithm 1.

Algorithm 1: Potential order building

Data: $\{\delta q^i(k)\}_{i \in [1..I], k \in [1..s_{max}^i]}$
Result: Best action a_Q of size Q

forall $i \in [1..I]$ **do**
  $a^i = 0$;
end
while $\sum_{i=1}^{I} a^i < Q$ **do**
  $i = \arg\max_{j \in [1..I]} \delta q^j(s_0^j + a^j + 1)$;
  $a^i = a^i + 1$;
end
Return $a_Q = \{a^i\}_{i \in [1..I]}$;

Since $\forall k \in [s_0^i + 1..s_0^i + a_{max}^i - 1], P(d^i \geq k+1) \leq P(d^i \geq k)$, then $\forall k \in [1..s_0^i + a_{max}^i - 1], \delta q^i(k+1) < \delta q^i(k)$. Therefore the order produced by algorithm 1 maximizes the expected reward. We can compute the complexity c of the algorithm 1: The first selection at line 5 requires a full sort, but if the result of the sorting is kept in a sorted list, one must only remove the greatest element $\delta q^i(s_0^i + a^i + 1)$, while adding the new element $\delta q^i(s_0^i + a^i + 2)$ to the list.

$$c = O(I \log(I) + Q\, I) \tag{II.13}$$

The figures II.20 and II.21 illustrate this algorithm applied to the same example as figure II.19. The $\delta q^i(k)$ curves in figure II.20 are the increments of the $q^i(k)$ curves displayed in Figure II.19. The arrows show in which order the increments are selected, and one can see that they are chosen by decreasing δq values. This figure also illustrates why no other order selection of size Q would generate more reward, as the Q selected order increments are the ones associated to the Q greatest reward increment. The way this order building procedure translates in the action space is schematized in figure II.21.

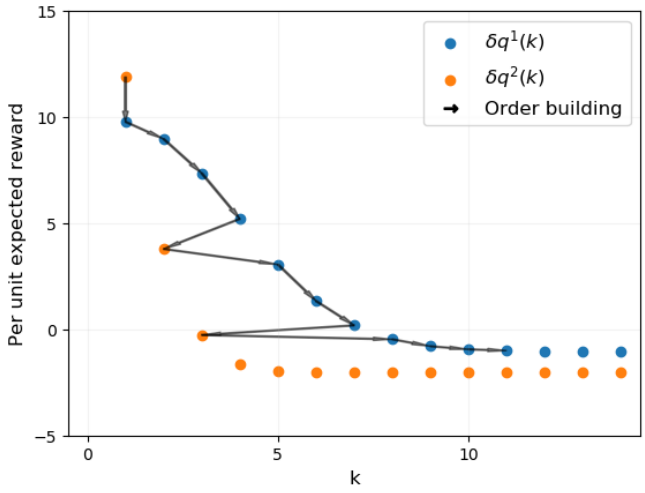


Figure II.20: Order building procedure in a two items scenario

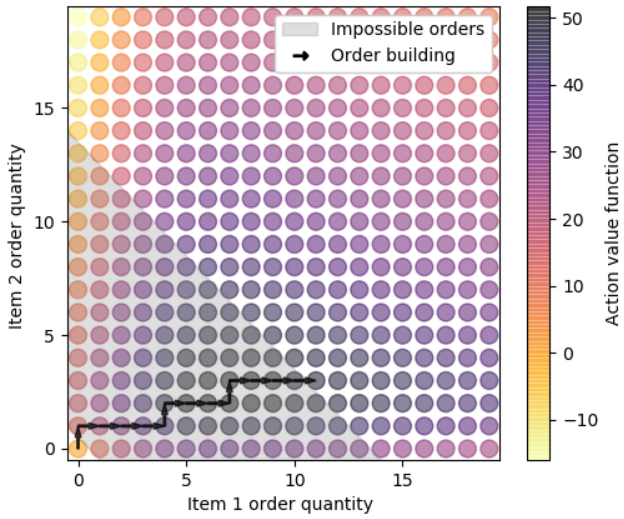


Figure II.21: Order building in the 2D action space in a two items scenario

Once the best order a_Q of size Q is built, one can compare its expected reward $q(x_0, a_Q)$ to the empty order $a_\emptyset$ expected reward $q(x_0, a_\emptyset)$, in order to know whether it is better to not order at all. Since the only difference in reward is generated by the additional ordered units, we have the equation:

$$q(x_0, a_Q) - q(x_0, a_\emptyset) = \sum_{i=1}^{I} \sum_{a'=1}^{a_Q^i} \delta q^i(s_0^i + a') \tag{II.14}$$

Therefore we can determine the solution a_*.

If $\sum_{i=1}^{I} a_{opt}^i \geq Q$, then $a_* = \prod_{i=1}^{I} \{a_{opt}^i\}$. Otherwise, $a_* = \underset{a \in \{a_Q, a_\emptyset\}}{\arg\max}(q(x_0, a))$

In this section, we therefore described a full resolution of the single period problem based on the idea of atomizing the action value function into a sum of single unit contributions to the expected reward. This idea of atomizing the action value function is core to the heuristic developed in chapter 3 to address the multi-period problem.

Conclusion

In this chapter, we first presented in section II.1 a possible formalization of the MOQ problem. We then demonstrated in section II.2 that is was possible to apply the backward recursions to compute the optimal policy and advantage functions, but only on the smallest instances of the MOQ problem. In section II.3, II.4 and II.5, we utilized small scale examples to analyze the MOQ problem. The conclusion of II.3 is that unless the demand is deterministic or large enough compared to Q, assuming that the optimal order size is equal to Q is a reasonable approximation. In section II.4 we introduced the concept of inaction window, which is a helpful concept to intuitively understand how an action impacts the system over time, and how subsequent orders can mitigate the long term effects of a sub optimal decision. In section II.5, we illustrated the dependency between the items, and how the MOQ impacts their respective contributions to the reward. We finally presented in section II.6 an efficient solution for a simplified version of the MOQ problem (the single period case). this solution is based on idea of atomizing the action value function into a sum of single unit contributions to the expected reward. In the next chapters we leverage the results and intuitions presented this chapter to provide heuristic solutions to the multi-period MOQ problem.

Chapter III

The w-policy

III.1 Introduction and methodology overview

This chapter presents the w-policy, a heuristic designed to approximately solve the MOQ problem. The w-policy leverages the problem analysis of chapter II to make suitable assumptions that drastically reduce the complexity of the problem. The MOQ problem can be decomposed into two sub-problems: firstly one must determine whether an order should be placed, and secondly, if there is an order, one must determine its composition. With the w-policy, we proceed the other way around: we first assume that we should place an order, and we compute the best one (section III.3). Once this potential action is known, we can compare its expected reward to the expected reward of delaying the order (section III.4).

The originality of the w-policy is that these computations are performed in a tractable way, leveraging several key assumptions. The technical formulations of these assumptions will be detailed in the rest of the chapter, but the idea behind them is that the MOQ constraint can be seen as a constraint that forces the items to be reordered less frequently and simultaneously. While this induces a dependency between the items contributions to the action value function, this dependency can be simplified if future reorder timings are known. We first make an assumption on these timings (section III.2). We then use this assumption to simplify the expression of the action value function. In section III.3, we demonstrate that with further approximations, the action value function can be re-formulated as a sum of order increment contributions, similarly to the single period MOQ problem detailed in section II.6. This drastically reduces the complexity of selecting the best potential order, since the order building procedure detailed in algorithm 1 can be used. This best potential order (that we note a_Q) is used in section III.4 to estimate the difference in expected reward between ordering immediately and delaying the order. A few more simplifying assumptions are made to make the computation of the sign of this difference tractable. The entire w-policy and the associated algorithm are summed up in section III.5. A method to estimate of the value of w, which is at the core of the idea of the w-policy, is presented in section III.6. We then introduce in section III.7 a practical advice on the shape of the forecasts to reduce the complexity of the w-policy on the large instances of the MOQ problem. In section III.8, an extensive analysis of the w-policy is conducted on small scale toy problems. This analysis exhibits the near optimal performance of the w-policy within its scope of applicability (where the impact of the simplifying assumptions is negligible). It also illustrates the limitations of the w-policy outside of this scope of applicability. In section III.9, the performance of the w-policy are studied on large scale real data sets. The w-policy demonstrates both its efficiency and its ability to scale up to very large instances of the MOQ problem (up to ten thousand items). We introduce many notations in this chapter: to help the reader, we provide a glossary of these new notations in appendix at the end of the chapter.

III.2 The inaction window approximation

If there was no MOQ constraint, the optimal policy would be to order small quantities at every time-step to remain at an ideal safety stock. The MOQ restricts the possibility to reorder every item independently whenever needed. Instead, orders must be placed simultaneously for all items to be able to reach the MOQ. The duration between the current order and the next one (the inaction window) is therefore critical to estimate how much demand the retailer will face for every item before it has the opportunity to replenish its stock again. This is particularly true when the demand is not stationary, as the items seasonality pattern may greatly influence the items relative importance.

This problem is illustrated in figure III.1. We consider a scenario with a system with two items subjected to an MOQ. The two items have seasonal demand patterns. We assume the lead time to be equal to 15 weeks. When placing an order at time step 0, the retailer must carefully consider the duration of the inaction window. Indeed, if he knows that the next order will be placed 15 weeks later, he should prioritize item 1 over item 2, since the item 1 expected demand D^1 between the reception of the current order and the reception of the next order is larger than item 2 expected demand D^2. On the other hand, if the inaction window is equal to 30 weeks, the two items should be prioritized rather equally, since the expected demands are similar.

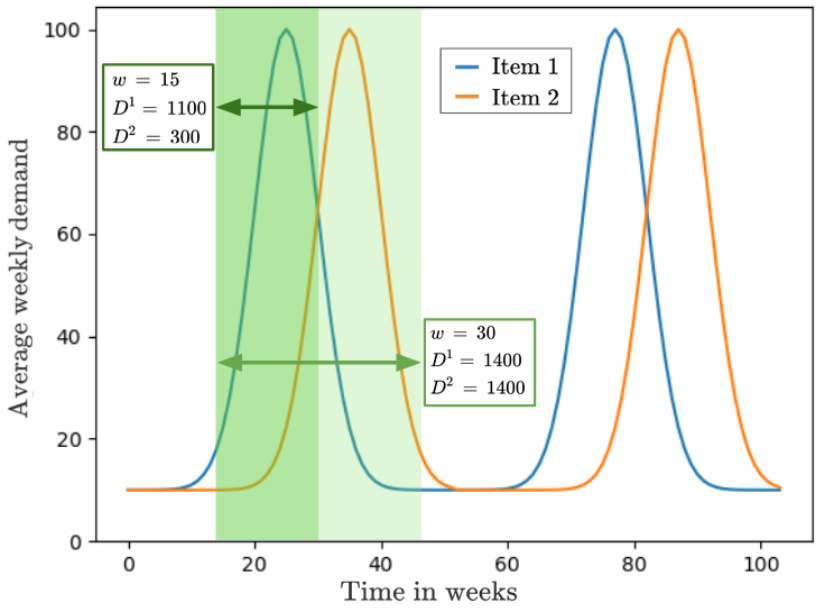


Figure III.1: Impact of the inaction window on items prioritization

The only impact an item has on the other items expected rewards is by having an influence on the future reorder timings. The inaction window can be seen as a higher-level variable that encapsulates the information about the global state of the system.

We define the w-policy as a heuristic policy that makes the assumption that if the retailer places an order of size Q at time t, the inaction window is known with certainty. This means that we assume that the next order will always be placed at time $t + w$, with w being an integer. This does not mean that the actor will necessarily wait exactly w time steps before placing the next order: it simply means that, when computing the order at time t, the actor assumes that the inaction window will be equal to w. The w-policy is then re-applied at every time step, and is therefore capable of adapting to the observed demand to reorder sooner or later than initially assumed.

The value for w should be chosen considering Q and the demand probabilities, as the demand between $t + l$ and $t + l + w$ should ideally be covered by the pre-reception stock at time $t + l$ plus

the order placed at time t, and any demand occurring after $t + l + w$ should ideally be covered by subsequent orders, not the order placed at time t. Figure III.2 provides a visualisation of the different relevant time windows, with $I_1 = [t..t + l[$, $I_2 = [t + l..t + l + w[$, and $I_3 = [t + l + w..t_{end}]$. One can note that the lead-time l may be smaller or greater than the inaction window w, it does not change the definition of these time windows. The practical way to compute w is discussed in section III.6. w may not be a constant value (especially if the demand is non stationary), and can be re-computed before applying the w-policy at every time step. If necessary, one may therefore use the notation w_t to name the inaction window estimate that starts at time t, instead of the simpler but potentially confusing w notation.

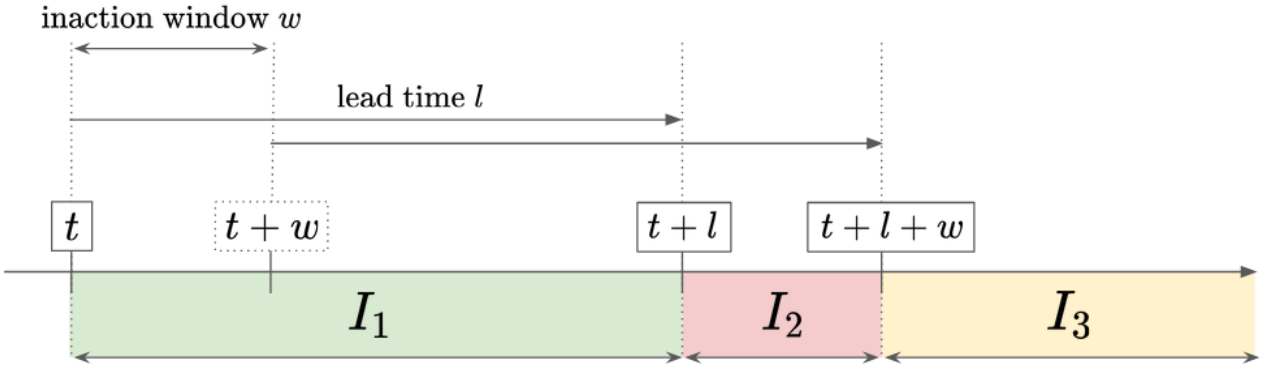


Figure III.2: MOQ problem relevant time windows

If $w = 1$, it means that the average demand per time step exceeds the MOQ. It is a trivial instance of the MOQ problem, as the MOQ is not a limiting factor when ordering. The w-policy is therefore built to handle the cases where $w > 1$. In these cases, we conjecture that the best total order size does rarely exceeds Q if the demand is not deterministic. The scope of applicability of this assumption was already discussed in section II.3. The w-policy is therefore built to always suggest orders of size exactly Q.

We now consider the impact the inaction window assumption has on the equation II.6 (presented in chapter II). The action value can be re-written:

$$q_t^\pi(x, a, w) = \sum_{t' \in I_1 \cup I_2 \cup I_3}^{t_{end}} \sum_{i=1}^{I} \sum_{k=1}^{s^i_{\max}} \left(P(D^i_{t'} \geq k) m^i - P(D^i_{t'} < k) h^i \right) P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi(X_{t''}), x_t = x)) \quad \text{(III.1)}$$

The inaction window assumption lets us consider differently the stock level probabilities $\hat{S}^i_{t'}$ during I_1, I_2 and I_3.

For all t' in I_1, the stock level $\hat{S}^i_{t'}$ only depends on the stock available and stock on order at t, and the demand of item i between t and t': the order placed at time t has no impact on the stock levels before it is received (at time $t + l$). We can therefore write that:

$$\forall t' \in I_1, P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi(X_{t''}), x_t = x) = P(\hat{S}^i_{t'} \geq k | x^i_t = x^i)$$

It is to be noted that if $t' \in I_1$, $P(\hat{S}^i_{t'} \geq k)$ does not depend on the state of the items $i' \neq i$ or on the policy π.

Once the order is received, the time window I_2 starts. By definition of w (since there is no new order placed between t and $t + w$), we have:

$$\forall t' \in I_2, \hat{S}^i_{t'} = \left[S^i_{t+l} + a^i_t - \sum_{t''=t+l}^{t'-1} D^i_{t''} \right]^+$$

Therefore, if $t' \in I_2$, $\hat{S}^i_{t'}$ only depends on S^i_{t+l}, a^i_t, and $D^i_{t''}$ ($t'' \in [t+l..t'[$). S^i_{t+l} can be determined given that x^i_t and the forecasts $f^i_{t'' \in [t..t+l[}$ are known. Therefore, if x^i_t, a^i_t and the forecasts $f^i_{t'' \in [t..t']}$ are known, the $\hat{S}^i_{t'}$ probability distribution can be explicitly computed for every $i \in [1..I]$ and every $t' \in I_2$. We can therefore write:

$$\begin{aligned} \forall t' \in I_2, P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi(X_{t''}), x_t = x)) \\ = P(\hat{S}^i_{t'} \geq k | a^i_t = a^i, a^i_{t'' \in [t+1..t']} = 0, x^i_t = x^i)) \quad \text{(III.2)} \end{aligned}$$

Once the order placed at time $t+w$ is received, the time window I_3 starts. During I_3, the stock level $\hat{S}^i_{t'}$ depends again on the stock available and stock on order at t, the demand between t and t', and the quantity ordered at time t of item i. It additionally depends on the policy π, and on the other items states at time t, since the quantity ordered of item i at time $t+w$ depends on π, and on the other items stock levels.

Dependency of $P(\hat{S}^i_{t'} \geq k)$ to	$\forall t' \in I_1$	$\forall t' \in I_2$	$\forall t' \in I_3$
x^i	✓	✓	✓
$\{D^i_{t''}\}_{t'' \in [t..t']}$	✓	✓	✓
a^i	✗	✓	✓
$\{x^j\}_{j \neq i}$	✗	✗	✓
$\{D^j_{t''}\}_{t'' \in [t..t'], j \neq i}$	✗	✗	✓
$\{a^j\}_{j \neq i}$	✗	✗	✓
π	✗	✗	✓

Table III.1: $P(\hat{S}^i_{t'} \geq k)$ dependencies

The dependencies of $P(\hat{S}^i_{t'} \geq k)$ for each interval I_1, I_2 and I_3 is recapitulated in table III.1. The table indicates in grey when the computation of $P(\hat{S}^i_{t'} \geq k)$ does not depend on the considered variable. When a cell is green, it indicates that $P(\hat{S}^i_{t'} \geq k)$ depends on the considered variable, and that this variable only relates to the item i. The column for I_1 and I_2 are only green and gray, which means that $P(\hat{S}^i_{t'} \geq k)$ can be explicitly computed, knowing only the state at t, the forecasts between t and t' and the item i ordered quantity at t (a^i).

When a cell is orange, it indicates that $P(\hat{S}^i_{t'} \geq k)$ depends on this variable, and that this variable relates to other items $j \neq i$. It means that the knowledge of the other items state and order is necessary to compute $P(\hat{S}^i_{t'} \geq k)$ explicitly. It is only the case when t' is in I_3. The red cell indicates that $P(\hat{S}^i_{t'} \geq k)$ depends on the policy applied after $t+w$, which means that its theoretical computation would require to compute the optimal policy for every possible scenario at every time step until t_{end}. It is again only the case when t' is in I_3.

We can rewrite equation III.1 to simplify the dependencies of the probabilities $P(\hat{S}^i_{t'} \geq k)$, and illustrate the aforementioned dependencies by coloring the equation's terms. The terms that can be explicitly computed without knowing π and $x^j, j \neq i$ are colored in green, while the terms that can not are colored in red.

$$
\begin{aligned}
q_t^{\pi}(x,a,w) = & \sum_{t' \in I_1} \sum_{i=1}^{I} \sum_{k=1}^{s^i_{\max}} \left(P(D^i_{t'} \geq k)\, m^i - P(D^i_{t'} < k)\, h^i \right) P(\hat{S}^i_{t'} \geq k | x^i_t = x^i)) \\
& + \sum_{t' \in I_2} \sum_{i=1}^{I} \sum_{k=1}^{s^i_{\max}} \left(P(D^i_{t'} \geq k)\, m^i - P(D^i_{t'} < k)\, h^i \right) P(\hat{S}^i_{t'} \geq k | a^i_t = a^i, a^i_{t'' \in [t+1..t']} = 0, x^i_t = x^i)) \\
& + \sum_{t' \in I_3} \sum_{i=1}^{I} \sum_{k=1}^{s^i_{\max}} \left(P(D^i_{t'} \geq k)\, m^i - P(D^i_{t'} < k)\, h^i \right) P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_{\emptyset}, a_{t'' \geq t+w} = \pi(X_{t''}), x_t = x))
\end{aligned}
\tag{III.3}
$$

The inaction window therefore allows to divide the future into three separate time windows. Thanks to this approximation, the reward expression on the time windows I_1 and I_2 can be simplified to an expression that can be computed without prior knowledge of the policy. However, the reward expression on the time windows I_3 still remains intractable.

III.3 Building the best potential order

The goal of this section is to compute the best potential order a_Q of size exactly Q in a tractable way. The intuition behind the w-policy is that every ordered unit will generate a contribution to the global reward. In practice, each unit will either be sold or generate a holding cost at the end of each period. By abstractly representing the inventory as a FIFO system, each unit in an order can be associated with an expected reward δq^i (similarly to section II.6).

The practical computation of the expected reward associated to each order increment still requires to apply several simplifying assumptions. Firstly, the w-policy makes the inaction window assumption described in the previous section. It is however not sufficient: equation III.3 illustrates that while the expected reward on I_1 and I_2 can be easily computed without considering the policy π, the computation of the expected reward on I_3 will require further approximations. Building the best potential order a_Q can therefore be divided into the following steps:

1. Find a good approximation of $q_t^{\pi}(x,a,w) - q_t^{\pi}(x,a_{\emptyset},w)$ that can be divided into independent order increment contributions δq^i.
2. Determine the expression of the order increments contributions δq^i.
3. Apply the optimal allotment algorithm (algorithm 1 in section II.6) to build a_Q using the δq^i values.

III.3.1 $\Delta q^*(x,a,w)$, $\Delta \tilde{q}(x,a,w)$ and 'oracle' assumption

For a given time t, we define $q^*(x,a,w)$ as the action-value of taking action a in state x, if the next order is necessarily happening w iterations later, and if all decisions taken from $t+w$ to t_{end} are optimal. We dropped the t indexes on $q^*(x,a,w)$ for readability reasons.

We define $\Delta q^*(x,a,w) = q^*(x,a,w) - q^*(x,a_{\emptyset},w)$. In this subsection, we focus on the expression of $\Delta q^*(x,a,w)$ rather than the expression of $q^*(x,a,w)$. Indeed, there are two advantages to dealing with a difference of action value function rather than the action value directly. The first one is that some components are identical in the two scenarios and therefore simplify themselves. The second one is that the difference of expected reward tends to decrease over time (under the assumption that the retailer acts optimally for $t' > t$): future action will be able to compensate for a sub-optimal decision taken at time t. This effect was studied in detail in section II.4.

It is therefore advantageous to consider the difference of all the $q^*(x,a,w)$ to the same reference point. It would be natural to consider $q^*(x,a,w) - q^*(x,a^*,w)$ with a^* being the optimal order, since it is the definition of the advantage function. We however do not know a^*, since the goal of the w-policy is to determine it. We therefore instead consider $q^*(x,a_\emptyset,w)$ as the reference point. It may appear unnatural, since we are comparing the action values for actions of size exactly Q (the assumption that the inaction window is equal to w is based on the assumption that the action taken has a size equal to Q), but the size of the order of the reference point does not actually matter. It is simply a convenient convention. $q^*(x,a_\emptyset,w)$ is the expected reward if the retailer was forced to wait until $t+w$ before placing an order. Using equation III.1, one can rewrite $\Delta q^*(x,a,w)$ as:

$$\begin{aligned}\Delta q^*(x,a,w) = \sum_{t' \in I_1 \cup I_2 \cup I_3} \sum_{i=1}^{I} \sum_{k=1}^{s^i_{max}} \big(P(D^i_{t'} \geq k)\, m^i - P(D^i_{t'} < k)\, h^i\big) \\ (P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t'' \in [t..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x)) \quad \text{(III.4)}\end{aligned}$$

$P(D^i_{t'} \geq k)$ and $P(D^i_{t'} < k)$ are known since we have access to the demand forecasts f^i_t. We also know that $\forall t' \in I_1, \forall k \in \mathbb{N}, P(\hat{S}^i_{t'} \geq k | x_t = x, a_t = a) = P(\hat{S}^i_{t'} \geq k | x_t = x, a_t = a_\emptyset)$. Indeed, the impact of the order a is delayed by l time steps, so the post-reception stock levels $\hat{S}^i_{t'}$ are identical until $t+l$. Therefore the sum on I_1 in the $\Delta q^*(x,a,w)$ formula (equation III.4) is equal to zero.

$$\begin{aligned}\forall t' \in I_1, P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t'' \in [t..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x)) \\ = P(\hat{S}^i_{t'} \geq k | x^i_t = x^i) - P(\hat{S}^i_{t'} \geq k | x^i_t = x^i) = 0\end{aligned}$$

$\forall t' \in I_2$, we previously determined that the $\hat{S}^i_{t'}$ probability distribution can be explicitly computed if x^i_t, a^i_t and the forecasts $f^i_{t'' \in [t..t']}$ are known (equation III.2). The difference between the probabilities on I_2 can be rewritten:

$$\begin{aligned}\forall t' \in I_2, P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t'' \in [t..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x)) \\ = P(\hat{S}^i_{t'} \geq k | a^i_t = a^i, a^i_{t'' \in [t+1..t+w[} = 0, x^i_t = x^i) - P(\hat{S}^i_{t'} \geq k | a^i_{t'' \in [t..t+w[} = 0, x^i_t = x^i)\end{aligned}$$

Therefore, the sum on I_2 in the $\Delta q^*(x,a,w)$ formula (equation III.4) can also be computed explicitly.

$\forall t' \in I_3$, as shown in table III.1, the post-reception stock levels $\hat{S}^i_{t'}$ depend on the future decisions $a_{t''}$, $t'' \in [t+w..t'-l]$, that themselves depend on the policy, and on the other items stock levels and demands. In order to make the computation of $\Delta q^*(x,a,w)$ on I_3 tractable, we are forced to introduce a new simplifying assumption, that we call the 'oracle' assumption. The 'oracle' assumption states that every action after time $t+w$ will be perfect in the sense that the retailer will be able to order the exact amount necessary to satisfy the demand, regardless of MOQ constraints and the demand stochasticity. It could only be true if one was able to predict the outcome of the stochastic process. It is a stronger assumption than the optimality of future decisions, as optimal decisions only maximize the expectation of the reward if we can not anticipate the outcome of the demand stochastic process. Expressed in terms of stock level, this 'oracle' assumption is equivalent to write that $\forall t' \in I_3$:

$$\hat{S}^i_{t'} = \max(D^i_{t'}, S^i_{t+l} + a^i_t - \sum_{t''=t+l}^{t'-1} D^i_{t''})$$

The post-reception stock decreases at every time step until it is not sufficient to satisfy the observed demand, at which point the 'oracle' actions set the post-reception stock levels to the exact amount necessary to satisfy the demand. It is therefore possible to compute the $\hat{S}^i_{t'}$ probability values in I_3 (and therefore the sum on I_3 in equation III.4) without the knowledge of the policy π and the other items states and demand forecasts. Therefore one can write that $\forall t' \in I_3$, $P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi(X_{t''}), x_t = x))$ is a function of only k, x^i, the action a^i and the demand forecast of item i between t and t'. We note this function $\tilde{\phi}^i_{t'}$:

$$\text{If } \hat{S}^i_{t'} = \max(D^i_{t'}, S^i_{t+l} + a^i_t - \sum_{t''=t+l}^{t'-1} D^i_{t''}), \text{ then } \forall t' \in I_3,$$
$$P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi(X_{t''}), x_t = x)) = \tilde{\phi}^i_{t'}(k, x^i, \{f^i_{t''}\}_{t'' \in [t..t']}, a^i)$$

$\tilde{\phi}^i_{t'}$ is a biased approximation of $P(\hat{S}^i_{t'} \geq k)$ because of the oracle assumption. The expected reward is over-estimated from the moment it is applied, since we assume that we can take perfect decisions. However, we use the same 'oracle' assumption for the two compared scenarios, which alleviates the issue since the bias will mostly cancel out when we consider the difference. The actual approximation we make is therefore that :

$$\begin{aligned}\forall t' \in I_3, P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t'' \in [t..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ \approx \tilde{\phi}^i_{t'}(k, x^i, \{f^i_{t''}\}_{t'' \in [t..t']}, a^i) - \tilde{\phi}^i_{t'}(k, x^i, \{f^i_{t''}\}_{t'' \in [t..t']}, 0)\end{aligned}$$

The advantage of making this approximation is that the $\tilde{\phi}^i_{t'}$ function does not depend on the policy π and on the other items $j \neq i$ state and orders. The computation of $\tilde{\phi}^i_{t'}$ is therefore tractable, and comparable in terms of complexity to the computation of $P(\hat{S}^i_{t'} \geq k)$ for $t' \in I_1 \cup I_2$. Table III.2 illustrates this idea by comparing the variables required to compute $\tilde{\phi}^i_{t'}$ and $P(\hat{S}^i_{t'} \geq k)$ for $t' \in I_3$.

Dependency of ... to ...	$\tilde{\phi}^i_{t'}(k, x^i, \{f^i_{t''}\}_{t'' \in [t..t']}, a^i)$	$P(\hat{S}^i_{t'} \geq k), \forall t' \in I_3$
x^i	✓	✓
$\{f^i_{t''}\}_{t'' \in [t..t']}$	✓	✓
a^i	✓	✓
$\{x^j\}_{j \neq i}$	✗	✓
$\{f^j_{t''}\}_{t'' \in [t..t'], j \neq i}$	✗	✓
$\{a^j\}_{j \neq i}$	✗	✓
π	✗	✓

Table III.2: Impact of the oracle assumption on the dependencies

We define $\Delta\tilde{q}(x, a, w)$ as the approximation of $\Delta q^*(x, a, w)$ that relies on this 'oracle' assumption. The combination of the inaction window approximation and of the 'oracle' assumption removes the dependency of the sum over I_3 in $\Delta\tilde{q}(x, a, w)$ to the policy π. The computation of $\Delta\tilde{q}(x, a, w)$ is therefore tractable, and can be written as:

$$\Delta\tilde{q}(x,a,w) = \sum_{t'\in I_2}\sum_{i=1}^{I}\sum_{k=1}^{s^i_{\max}} \left(P(D^i_{t'} \geq k)m^i - P(D^i_{t'} < k)h^i\right)$$
$$\left(P(\hat{S}^i_{t'} \geq k | a^i_t = a^i, a^i_{t''\in[t+1..t+w[} = 0, x^i_t = x^i) - P(\hat{S}^i_{t'} \geq k | a^i_{t''\in[t..t+w[} = 0, x^i_t = x^i)\right)$$
$$+ \sum_{t'\in I_3}\sum_{i=1}^{I}\sum_{k=1}^{s^i_{\max}} \left(P(D^i_{t'} \geq k)m^i - P(D^i_{t'} < k)h^i\right)\left(\tilde{\phi}^i_{t'}(k, x^i, \{f^i_{t''}\}_{t''\in[t..t']}, a^i) - \tilde{\phi}^i_{t'}(k, x^i, \{f^i_{t''}\}_{t''\in[t..t']}, 0)\right)$$

III.3.2 Dividing $\Delta\tilde{q}(x,a,w)$ into order increment contributions $\delta\tilde{q}^i(x^i,a^i,w)$

While we could theoretically compute $\Delta\tilde{q}(x,a,w)$ for every a of size exactly Q and select the a^* that maximizes $\Delta\tilde{q}(x,a,w)$, it would be intractable to do so when a large number of items is considered. Instead, we re-formulate $\Delta\tilde{q}(x,a,w)$ as a sum of order increments contributions:

$$\Delta\tilde{q}(x,a,w) = \sum_{i=1}^{I} \Delta\tilde{q}^i(x^i,a^i,w) \tag{III.5}$$

With $\Delta\tilde{q}^i(x^i,a^i,w)$ being the difference in reward generated by an item i. The contribution of an item i can itself be formulated as the sum of incremental order rewards $\delta\tilde{q}^i(x^i,k,w)$:

$$\Delta\tilde{q}^i(x^i,a^i,w) = \sum_{k=1}^{a^i} \delta\tilde{q}^i(x^i,k,w) \tag{III.6}$$

With $\delta\tilde{q}^i(x^i,k,w)$ being the contribution of the incremental order to the expected reward. Its formula is (a formal demonstration is provided in appendix of this chapter):

$$\delta\tilde{q}^i(x^i,k,w) = P\left(\sum_{t'\in I_2} D^i_{t'} \geq S^i_{t+l} + k \,\middle|\, x^i_t = x^i\right)m^i - \sum_{t'\in I_2\cup I_3} P\left(\sum_{t''=t+l}^{t'} D^i_{t''} < S^i_{t+l} + k \,\middle|\, x^i_t = x^i\right)h^i \tag{III.7}$$

$\delta\tilde{q}^i(x^i,k,w)$ is conceptually very close to $\delta q^i(k)$ presented in section II.6, as the expected reward generated by specific units of the order. $\delta\tilde{q}^i(x^i,k,w)$ can be seen as the additional reward generated by a unit k if we select it in the order.

$P\left(\sum_{t'\in I_2} D^i_{t'} \geq S^i_{t+l} + k \middle| x^i_t = x^i\right)$ (in the first part of the formula) is the probability that the k-th unit of the order is sold during I_2 (if the system follows a FIFO logic). If it is sold during I_2, it will generate a reward that would not have been generated if no order was placed. If the unit is sold after the reception of the next order, its order could have been delayed and the margin would have been generated anyway. We therefore generate an additional m^i only if the unit is sold during I_2.

$P\left(\sum_{t''=t+l}^{t'} D^i_{t''} < S^i_{t+l} + k \middle| x^i_t = x^i\right)$ (in the second half of the formula) represents the probability that the k-th unit of the order generates the holding cost h^i at the end of every time step. In I_2, the additional unit may generate an additional holding cost compared to the case where the unit was not ordered. In I_3, the 'oracle' assumption tells us that if we did not order the unit at time t, we would have ordered it so that it would be received right before being sold, generating no holding cost. However, if we order it now, we may generate holding costs during I_3 as even an 'oracle' order can not be negative and compensate for overstocks. These probability values can be deduced from the forecasts and the state of the system at time t.

III.3.3 Order building procedure

By considering these $\delta\tilde{q}^i(x^i, a^i, w)$ similarly to the $\delta q^i(k)$ of section II.6, one can build the order a_Q of size exactly Q that maximizes the approximate expected reward $\Delta\tilde{q}(x, a, w)$ (by maximizing each $\Delta\tilde{q}^i(x^i, a^i, w)$ independently). We recapitulate the assumptions we made to be able to compute this potential order.

- The best non-zero order size is exactly Q.
- The inaction window if we place an order of size Q is known and equal to w.
- All orders will be perfect from $t + w$ to t_{end} ('oracle' assumption).

As a reminder, the inaction window assumption does not mean that the actor will necessarily wait exactly w time steps between each order: it simply means that, when computing the order at time t, the actor assumes that the inaction window will be equal to w. The w-policy is then re-applied at every time step.

III.4 Deciding to place or to delay the order

The next problem is to determine whether or not it is better to order the previously computed a_Q, or to delay the order. The answer should be given by comparing the action value functions $q_t^\pi(x, a_Q)$ and $q_t^\pi(x, a_\emptyset)$: if $q_t^\pi(x, a_Q) > q_t^\pi(x, a_\emptyset)$, ordering is worth it. Otherwise, it is more profitable to wait. Similarly to section III.3, we must make a few simplifying assumptions in order to build an easily computable approximation of $q_t^\pi(x, a_Q) - q_t^\pi(x, a_\emptyset)$.

Firstly we assume that the inaction window if we order a_Q is known and equal to w. Secondly we assume that the inaction window if we do not order (that we call the delay, and that we note w') is smaller than w and is an integer ($w' \in [1..w]$). Similarly to I_2 and I_3, we define $I'_2 = [t+l..t+l+w'[$ and $I'_3 = [t + l + w'..t_{end}]$. The time division is schematized in figure III.3.

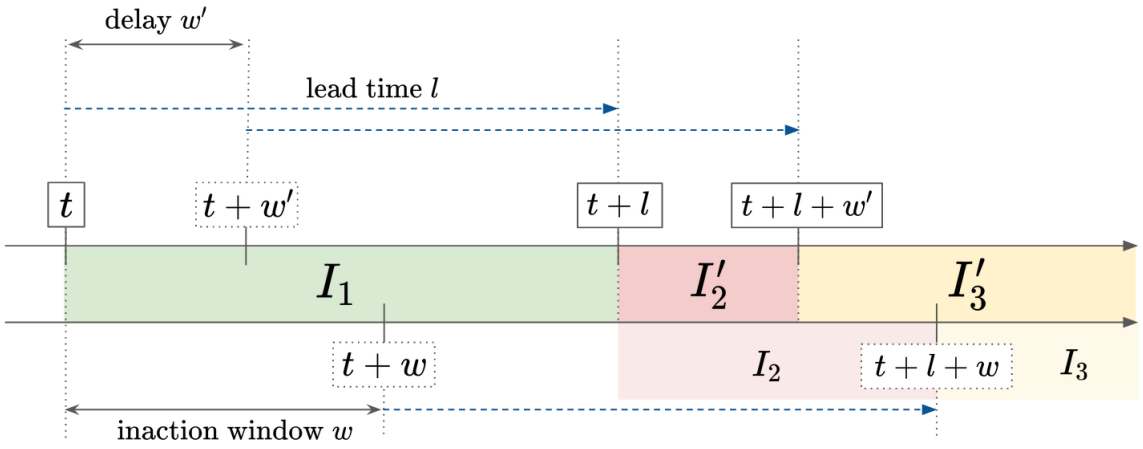


Figure III.3: Timeline, delay and inaction window

Thirdly we assume that the sign of $q_t^\pi(x, a_Q) - q_t^\pi(x, a_\emptyset)$ can be approximated by the sign of $\min_{w' \in [1..w]} (q^*(x, a_Q, w) - q^*(x, a_\emptyset, w'))$ (we again drop the t indexes for readability). Intuitively, we consider delaying the order by any realistic w', and compare the expected return of delaying and the expected return of ordering a_Q. The assumption behind this approximation is that the delay w' should be fixed at time t, when in reality the retailer can adapt the delay depending on the demand

observed after t. In practice, the sign of $\min_{w' \in [1..w]} (q^*(x, a_Q, w) - q^*(x, a_\emptyset, w'))$ is in most situations equal to the sign of $q^*(x, a_Q, w) - q^*(x, a_\emptyset, 1)$. Indeed, if it is optimal to delay the order by several timesteps, it is very likely that it is also better to delay the order by one timestep rather than not delaying the order. The only exception occurs for very specific situations with seasonal patterns: we take the example of a single item, at the end of a season of high demand, and with a stock nearly depleted. The best choice might be to delay the order up until the start of the next high demand season (to avoid paying holding costs during the low demand season, even if it means missing a few sales). But between ordering and delaying the order of only one timestep, it might be better to order immediately: if an order has to be placed, one might as well avoid being stocked out. Approximating the sign of $q_t^\pi(x, a_Q) - q_t^\pi(x, a_\emptyset)$ by the sign of $\min_{w' \in [1..w]} (q^*(x, a_Q, w) - q^*(x, a_\emptyset, w'))$ instead of the sign of $q^*(x, a_Q, w) - q^*(x, a_\emptyset, 1)$ lets the w-policy be more robust in these specific scenarios.

We need to compute the sign of $q^*(x, a_Q, w) - q^*(x, a_\emptyset, w')$, which can be re-formulated as:

$$\begin{aligned} q^*(x, a_Q, w) - q^*(x, a_\emptyset, w') = \sum_{t' \in I_1 \cup I_2' \cup I_3'} \sum_{i=1}^{I} \sum_{k=1}^{s^i_{max}} \big(P(D^i_{t'} \geq k)\, m^i - P(D^i_{t'} < k)\, h^i\big) \\ (P(\hat{S}^i_{t'} \geq k | a_t = a_Q, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t'' \in [t..t+w'[} = a_\emptyset, a_{t'' \geq t+w'} = \pi^*(X_{t''}), x_t = x)) \quad \text{(III.8)} \end{aligned}$$

Again, the difference of expected reward generated during I_1 is equal to zero given that the impact of the order is delayed by the leadtime.

$$\begin{aligned} \forall t' \in I_1, P(\hat{S}^i_{t'} \geq k | a_t = a_Q, a_{t'' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t'' \in [t..t+w'[} = a_\emptyset, a_{t'' \geq t+w'} = \pi^*(X_{t''}), x_t = x) = 0 \end{aligned}$$

$\forall t' \in I_2'$, by definition of w (since there is no new order placed between t and $t+w$), we have:

$$\hat{S}^i_{t'} = \left[S^i_{t+l} + a^i_t - \sum_{t''=t+l}^{t'-1} D^i_{t''} \right]^+$$

Therefore $\forall t' \in I_2'$, $\hat{S}^i_{t'}$ only depends on S^i_{t+l}, a^i_t, and $\{D^i_{t''}\}_{t'' \in [t+l..t'[}$. Since a_t is known, the $\hat{S}^i_{t'}$ probability distributions can be explicitly computed from the demand forecasts, and so can the sum on I_2' in the $\Delta q^*(x, a, w)$ formula (equation III.8).

$$\begin{aligned} \forall t' \in I_2', (P(\hat{S}^i_{t'} \geq k | a_t = a_Q, a_{t' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w'} = \pi^*(X_{t''}), x_t = x) \\ - P(\hat{S}^i_{t'} \geq k | a_{t' \in [t..t+w'[} = a_\emptyset, a_{t'' \geq t+w'} = \pi^*(X_{t''}), x_t = x)) \\ = (P(\hat{S}^i_{t'} \geq k | a^i_t = a^i_Q, a^i_{t' \in [t+1..t+w[} = 0, x^i_t = x^i) - P(\hat{S}^i_{t'} \geq k | a_{t' \in [t..t+w'[} = a_\emptyset, x_t = x)) \end{aligned}$$

$\forall t' \in I_3'$, the post-reception stock levels $\hat{S}^i_{t'}$ depend on the future decisions $a_{t''}$, $t'' \in [t+w..t'-l]$, that themselves depend on the policy, and on the other items stock levels and demands. In this section, we call scenario 1 the scenario where the order a is placed at t ($a_t = a, a_{t' \in [t+1..t+w[} = a_\emptyset, a_{t' \geq t+w} = \pi(X_{t'})$) and scenario 2 the scenario where the order is delayed to $t+w'$ ($a_{t' \in [t..t+w'[} = a_\emptyset, a_{t' \geq t+w'} = \pi(X_{t'})$). In order to compute the difference in the expected reward between the two scenarios, one would need to consider every possible outcome of the stochastic demand and

the associated optimal orders, and do so at each time step after $t + w'$. In order to make the computation on I'_3 tractable, we are forced to introduce another simplifying assumption.

On the contrary to the computation of $\Delta\tilde{q}$ in section III.3, it is not recommended to use the oracle assumption in this case: the oracle assumption is biased and overestimates the expected reward from the moment it is applied. In the scenario 1, the perfect decisions would in practice be applied after $t + l + w$, while in scenario 2 it would be applied sooner, at $t + l + w'$. The scenario 2 would therefore have an unfair advantage over scenario 1 in terms of expected reward. For this reason, the 'oracle' assumption can not be used.

Instead, one can assume that the difference in the reward generated during I'_3 is negligible compared to the difference in reward generated during I'_2. This approximation can be seen as a myopic approach of the problem: we only consider the impact of the order up to a certain point in time $t + l + w'$. In mathematical terms, it means that we assume that:

$$\begin{aligned}&\forall t' \in I_3,\\ &\left(P(D^i_{t'} \geq k)\, m^i - P(D^i_{t'} < k)\, h^i\right) \Big(P(\hat{S}^i_{t'} \geq k | a_t = a, a_{t' \in [t+1..t+w[} = a_\emptyset, a_{t'' \geq t+w'} = \pi^*(X_{t''}), x_t = x)\\ &\qquad - P(\hat{S}^i_{t'} \geq k | a_{t' \in [t..t+w[} = a_\emptyset, a_{t'' \geq t+w'} = \pi^*(X_{t''}), x_t = x)\Big) \approx 0\end{aligned}$$

The intuition behind this myopic assumption is that once the delayed order is received, the state of the system is likely to be very similar for the two scenarios. Indeed, if there is no out of stock during I'_2 and that the exact same order is placed at time t in scenario 1 and at $t + w'$ in scenario 2, the stock levels at $t + l + w'$ will be identical in the two scenarios. A difference can occur between the scenarios stocks past $t+l+w'$ if there are stock outs during I'_2, since we chose a lost sales model. In general, out of stocks are much costlier than holding costs: if significant stock outs occur during I'_2, the difference of reward generated during I'_2 will be massive. Comparatively, the difference in holding costs generated during I'_3 will be negligible. The only exception is when the holding cost parameter h^i are of the same order of magnitude than the parameter m^i. In this cases, the myopic assumption is very inaccurate. This limitation was however not encountered on the real datasets we studied. One way to alleviate this issue if necessary would be to build a more accurate estimate of $q^\pi_t(x, a_Q) - q^\pi_t(x, a_\emptyset)$.

One can wonder why the myopic assumption was not applied for the computation of $\Delta\tilde{q}(x, a, w)$. The reason is that the computation of $\Delta\tilde{q}(x, a, w)$ is utilized to determine the best allotment of the order, and therefore requires more accuracy: between two slightly different orders, the difference in expected reward often does not come from the short term stock-out probabilities (which are likely to be small), but rather from the long term holding costs differences. The 'oracle' assumption is better suited to leverage these potential long term holding cost differences.

The myopic approximation allows us to simplify equation III.8, and to define $\Delta\dot{q}(x, a_Q, w')$ as the estimate of $q^*(x, a_Q, w) - q^*(x, a_\emptyset, w')$ under this myopic assumption, and that is equal to:

$$\Delta\dot{q}(x, a_Q, w') = \sum_{i=1}^{I} \Delta\dot{q}^i(x^i, a^i_Q, w') \tag{III.9}$$

With $\Delta\dot{q}^i(x^i, a^i, w')$ being the difference in reward generated by an item i, defined as the sum of incremental order rewards $\delta\tilde{q}^i(x^i, k, w)$:

$$\Delta\dot{q}^i(x^i, a^i_Q, w') = \sum_{k=1}^{a^i_Q} \delta\dot{q}^i(x^i, k, w') \tag{III.10}$$

$$\delta\dot{q}^i(x^i,k,w') = P\left(\sum_{t'\in I'_2} D^i_{t'} \geq S^i_{t+l}+k \middle| x^i_t = x^i\right) m^i - \sum_{t'\in I'_2} P\left(\sum_{t''=t+l}^{t'} D^i_{t''} < S^i_{t+l}+k \middle| x^i_t = x^i\right) h^i \tag{III.11}$$

The equation III.11 can be demonstrated following the same steps as the demonstration of the $\delta\tilde{q}^i(x^i,k,w)$ formula (equation III.7, demonstration in appendix of this chapter). $\Delta\dot{q}(x,a_Q,w')$ can be computed for every $w' \in [1..w]$. If $\forall w' \in [1..w], \Delta\dot{q}(x,a_Q,w') \geq 0$ the w-policy recommends to order now, since no delay was found to be providing better reward. Otherwise, if $\exists w' \in [1..w], \Delta\dot{q}(x,a_Q,w') < 0$ the w-policy recommends to delay the order. It does not mean that the actor will delay the order by exactly w', it just means that it will not order at t, and re-apply the w-policy at $t+1$ to determine whether an order should be placed. We recapitulate the assumptions we made to be able to estimate whether it was worth delaying the order.

- If we order, we order a_Q computed section III.3.
- The inaction window if we place the order a_Q at t is known and equal to w.
- The sign of $q^*(x,a_Q) - q^*(x,a_\emptyset)$ is equal to the sign of $\min_{w'\in[1..w]}(q^*(x,a_Q,w) - q^*(x,a_\emptyset,w'))$.
- Between ordering or delaying the order, the difference of expected reward generated during I'_3 is negligible compared to the difference in expected reward generated during I'_2.

III.5 w-policy algorithm

The w-policy is the heuristic policy that relies on the previously detailed assumptions. These assumptions let us compute the $\delta\tilde{q}^i(x^i,a^i,w)$ values. From these values we build the hypothetical order $a_Q \in A$ using the algorithm 1. We then determine whether it is better to place this order a_Q or to delay it by computing the sign of $\Delta\dot{q}(x,a_Q,w')$ for all $w' \leq w$. This process is recapitulated in algorithm 2, and schematized in figure III.4. A few notations are introduced in figure III.4. We note δ_a the boolean variable that correspond to the statement 'we place an order'. The bloc B simply selects a_Q or $a_\emptyset$ depending on δ_a. We note π^+_w the function that builds the best potential order from the state of the system and w (lines 1 to 11 in algorithm 2).

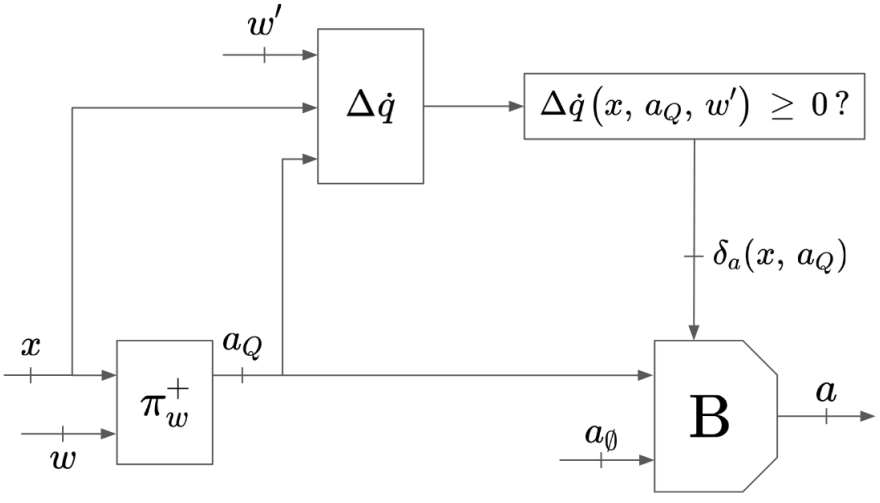


Figure III.4: w-policy

Algorithm 2: w-policy

Data: $\{m^i\}\{h^i\}_{i\in[1..I]}, \{f^i_{t'}\}_{i\in[1..I],t'\in[t..t_{end}]}, \{x^i_t\}_{i\in[1..I]}, w$
Result: Action a_t suggested by the w-policy

forall $i \in [1..I]$ **do**
  forall $k \in [1..a^i_{max}]$ **do**
    Compute $\delta\tilde{q}^i(x^i_t, k, w)$ using equation III.7;
  end
end
$a_Q = a_\emptyset$;
while $\sum_{i=1}^{I} a^i < Q$ **do**
  $i = \arg\max_{j\in[1..I]} \delta\tilde{q}^j(s^j_t + a^j_Q + 1)$;
  $a^i_Q = a^i_Q + 1$;
end
$a_Q = \{a^i_Q\}_{i\in[1..I]}$;
forall $w' \in [1..w]$ **do**
  forall $i \in [1..I]$ **do**
    forall $k \in [1..a^i_Q]$ **do**
      Compute $\delta\dot{q}^i(x^i, k, w')$ using equation III.11;
    end
    Compute $\Delta\dot{q}^i(x^i, a^i_Q, w')$ using equation III.10;
  end
  Compute $\Delta\dot{q}(x, a_Q, w')$ using equation III.9;
  if $\Delta\dot{q}(x, a_Q, w') < 0$ **then**
    Return $a_t = a_\emptyset$;
  end
end
Return $a_t = a_Q$;

If all the $\delta\tilde{q}$ and $\delta\dot{q}$ values are pre-computed, the complexity of the algorithm 2 is similar to the complexity $c = O(I\log(I) + Q\,I)$ of algorithm 1. To pre-compute the δq values, one must evaluate the probability $P(\sum_{t''=t+l}^{t'} d^i_{t''} \geq s^i_{t+l}+k)$ for every $k^i \in [1..a^i_{max}]$ and for every $t' \in [t+l..t_{end}]$ exactly once. By using memoization on these probability values, the complexity c' of pre-computing the δq values is therefore linear with the remaining time steps, the sum of the a^i_{max}, and the average complexity K of evaluating a probability $P(\sum_{t''=t+l}^{t'} D^i_{t''} \geq S^i_{t+l} + k)$ from the forecasts f^i_t:

$$c' = O\left(K(t_{end} - t)\sum_{i=1}^{I} a^i_{max}\right) \tag{III.12}$$

In practice, c' is several order of magnitude larger than c in all the scenarios we faced. Our method is therefore linear with the number of items and as such, it is applicable to large scale versions of the MOQ problem.

III.6 w value estimation

We know that the actual inaction window is a probability distribution (see section II.4). The inaction window approximation assumes that the inaction window is a dirac distribution in w. The choice of the w is critical for the w-policy success, since it impacts the allotment of the order: the potential order a_Q is built to 'aim' for an optimal safety stock at the end of the time window I_2.

With a^* being the optimal order $\pi^*(x)$, we can write that $q^*_t(x, a^*) \geq q^*_t(x, a_Q)$. Therefore if $q^*_t(x, a^*) - q^*_t(x, a_\emptyset)$ is negative, so will be $q^*_t(x, a_Q) - q^*_t(x, a_\emptyset)$. The w-policy is built around the assumption that the sign of $\min_{w'\in[1..w]} \Delta\dot{q}(x, a_Q, w')$ is a good approximation of the sign of $q^*_t(x, a_Q) - q^*_t(x, a_\emptyset)$, and it is therefore not a stronger assumption to state that if $q^*_t(x, a^*) - q^*_t(x, a_\emptyset)$ is negative, so will be $\min_{w'\in[1..w]} \Delta\dot{q}(x, a_Q, w')$ for any a_Q.

This means that the potential order a_Q allotment does not matter if the retailer is sufficiently stocked and does not need to place an order. The estimation of w therefore only matters in practice if an order is actually placed. As a consequence, we can build our w estimate under the assumption that the system is in a state x where it is optimal to place a new order.

We suggest to use a simple rule of thumb for the evaluation of w based on this assumption: w is the smallest $w' \in [1..t_{end} - t]$ that satisfies

$$\sum_{i=1}^{I}\sum_{t'=t}^{t+w'} \mathbb{E}[d^i_t] \geq Q \tag{III.13}$$

The intuition behind this rule of thumb is that if we only place orders of size Q the average duration between two orders should be the average time it takes to consume Q units.

The w value can be updated at every time-step, as this rule of thumb may give different results for different time-steps t. This rule of thumb was the one applied in all the experiments of sections III.8 and III.9. While it is possible to consider more sophisticated w estimates, we did not manage to craft methods to estimate w that were significantly superior in practice. This simple rule of thumb proved sufficient in all our experiments.

We conjecture that the reason for this is that the impact of the inaction window approximation on the overall efficiency of the policy is negligible as long as the average actual inaction window is not too far from w. This conjecture is supported by the results presented in section III.8, which indicate that the w-policy performance using the presented rule of thumb is almost optimal even when the average error on the inaction window estimate is significant but unbiased.

III.7 Forecast sampling approximation

The computational cost K of retrieving the probability values $P\left(\sum_{t'\in I_2} D^i_{t'} \geq S^i_{t+l} + k \middle| x^i_t = x^i\right)$ from the demand forecasts has a linear impact on the computational cost of the overall method (equation III.12). Reducing K is therefore critical to let the w-policy scale up to very large instances of the MOQ problem. The explicit computation of the probability distributions associated to $\sum_{t'\in I_2} D^i_{t'}$ and $S^i_{t+l} + k$ can prove expensive, especially if the forecasts are complex.

We therefore suggest to provide the forecasts f^i_t under the shape of randomly drawn samples instead of parametric distributions. If $k_{samples}$ are drawn from f^i_t, for each $t' \in [t..t_{end}]$, one can compute $k_{samples}$ different trajectories for the stock evolution $S_{t'}$.

To compute $P\left(\sum_{t'\in I_2} D^i_{t'} \geq S^i_{t+l} + k \middle| x^i_t = x^i\right)$, one only has needs count the number of trajectories that ended up with $\sum_{t'\in I_2} D^i_{t'} \geq S^i_{t+l} + k$, and divide it by $k_{samples}$. The complexity of drawing random samples from a parametric distribution is typically orders of magnitude faster than explicitly computing the probability mass function of the distribution. The acceleration provided by this approximation can be up to multiple orders of magnitude depending on the complexity of the initial parametric forecasts and the number of samples drawn. We analyze the impact of approximating parametric forecast with randomly drawn samples of the demand in section III.9, and show that the impact on the reward is negligible in practice, while providing a major acceleration of the computation.

III.8 Small scale toy experiments

The subject of this thesis is the optimization of the decisions under uncertainty, the goal is not to study the accuracy of forecasting methods. For this reason, we assumed our probabilistic forecasts to be perfectly accurate in the following sections. We first consider toy problems with only two items and no lead time. At this scale the exact solution is computationally feasible with the backward recursion algorithm, which allows us to estimate under what circumstances the w-policy provides satisfactory solutions.

III.8.1 Comparison between $\Delta q^*(x, a, w)$ and $\Delta\tilde{q}(x, a, w)$

The goal of our first experiment is to assess the quality of $\Delta\tilde{q}(x, a, w)$ as an approximation of $\Delta q^*(x, a, w)$. We set up a simple system with two items, with a stationary weekly demands, with probability distributions following Poisson laws of parameter λ^i. We first consider that the two items are identical with $m^1 = m^2 = 1$, $h^1 = h^2 = 0.1$, $\lambda^1 = \lambda^2 = 2.5$. We set $Q = 10$ and we set the initial stock levels to zero ($s^1 = s^2 = 0$). The exact state and action value functions are explicitly computed with the backward recursion algorithm. We can therefore compute the exact values of $\Delta q^*(x, a, w)$, and the expected contribution of each item $\Delta q^1_*(x, a, w)$ and $\Delta q^2_*(x, a, w)$. One can define $\delta q^1(x, \{k, a^2\}, w)$, as the difference between $\Delta q^1_*(x, \{a^1 = k, a^2\}, w)$ and $\Delta q^1_*(x, \{a^1 = k-1, a^2\}, w)$. We then compare $\delta q^1(x, \{k, a^2\}, w)$ to its approximation $\delta\tilde{q}^1(x^1, k, w)$. In figure III.5, we plotted $\delta\tilde{q}^1(x^1, k, w)$ for all the possible values of k, and compared it to the $\delta q^1_*(x, \{k, a^2\}, w)$ values for three fixed values of a^2, and for $x = \{s^1 = 0, s^2 = 0\}, w = 2$.

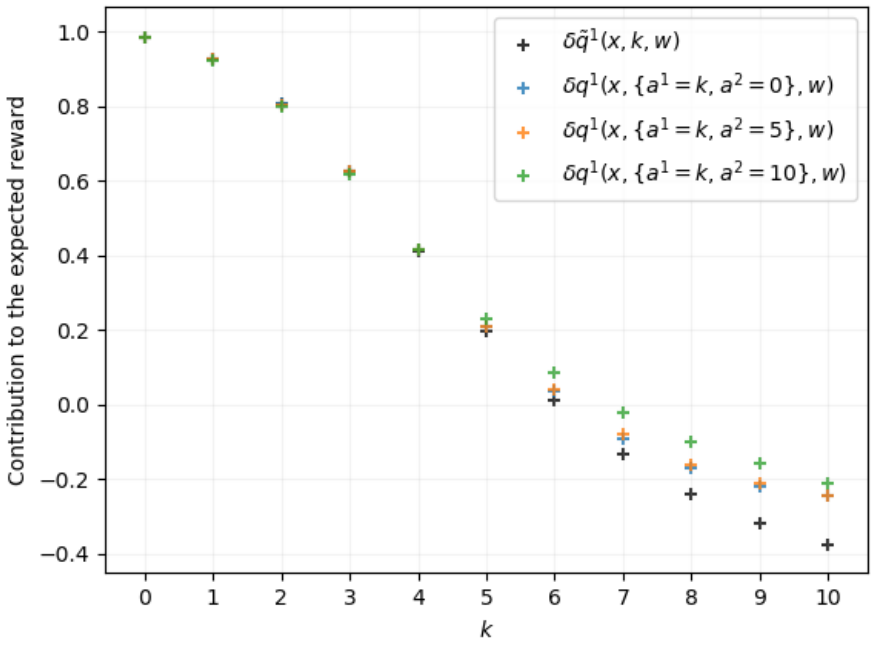


Figure III.5: δq values and approximation

The first idea that figure III.5 illustrates is that the true δq^1 values are not very dependent on a^2. This tends to confirm that the inter-item dependency is mostly included in the next reorder timing only, and validates our intuition that the inaction window is key to building a good potential order. The optimal order in this situation is $a^1 = a^2 = 5$. Interestingly, the approximation $\delta\tilde{q}$ tends deviate from the true δq values when $k > 5$. This is due to the oracle assumption: up until 5, units have a significant chance of being sold during I_2. Beyond 5, the units are more and more likely to be sold during I_3, and therefore rely more heavily on the oracle assumption for the computation of their associated $\delta\tilde{q}$ values. These inaccuracies tend to not have a significant impact on a_Q, since the units selected for the order a_Q are naturally the ones that are more likely to be sold during I_2.

We next consider a slightly different toy example. The settings are identical, except that we set $Q = 10$, and that the items forecast are not identical: $\lambda^1 = 7.5, \lambda^2 = 2.5$. Figure III.6 displays the $\delta\tilde{q}$ values, the corresponding true δq values, and it also displays the order increments that are selected to build a_Q ($a^1 = 15$ and $a^2 = 5$). The order building selection was illustrated similarly on figure II.20 in chapter II on the single period problem. One can notice that, again, the $\delta\tilde{q}$ approximation is less accurate for order increments that end up not being selected. We purposely selected an example where slightly less accurate estimate of the δq could lead to a different potential order with $a^1 = 16$ and $a^2 = 4$. Even if it was the case, the induced loss of expected reward would be very limited. In that sense, the order building procedure is rather robust, since the more probable it is that a sub-optimal decision is selected, the lower the impact of this sub-optimal decision on the expected reward.

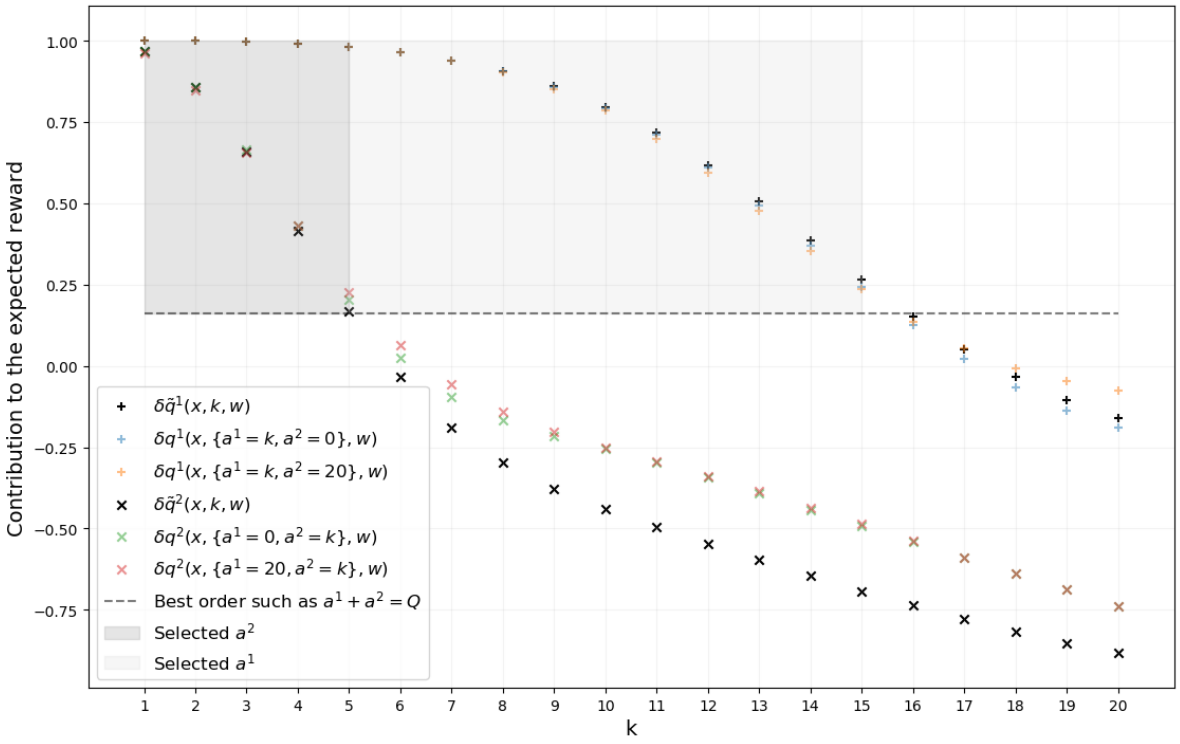


Figure III.6: Potential order a_Q and δq values

III.8.2 Impact of the inaction window assumption

We then explore whether the assumption that the inaction window is a fixed known value is true, and how it impacts the w-policy performance. In our first experiment, we assume the demand to be stationary (as it is a necessary condition for the (S,T) policy). We set Q to 25. The initial pre-reception stocks have been set to zero for the two items. We set the economic parameters to $m^1 = m^2 = 1$, $h^1 = h^2 = 0.1$. The demand probability distribution (and the corresponding perfectly accurate forecast f_t^i probability distribution) over a week for each item is a negative binomial distribution with a mean μ equal to 5. We used in this experiment (and in all the experiments of this section) our simple rule of thumb (equation III.13) to determine the w value. With $Q = 25$, we have $w = 3$.

Figure III.7 displays, for increasingly large values of standard deviation of our forecast, the average efficiency of the w-policy and the (S,T)-policy compared to the optimal policy over a thousand episodes of 52 weeks. We also estimated the average absolute error between the estimated w, equal to 3, and the actual inaction window. As expected, the error on the inaction window estimation increases when the demand uncertainty increases : the duration between orders has increasingly more chances of being shorter or longer than 3 time-steps. The performance of the w-policy are almost optimal in these settings, and even outperform the (S,T)-policy. This experiment illustrates that as long as w is set to a value that is close to the average actual inaction window, the assumption that the inaction window is fixed and equal to w has a very limited impact on the performance.

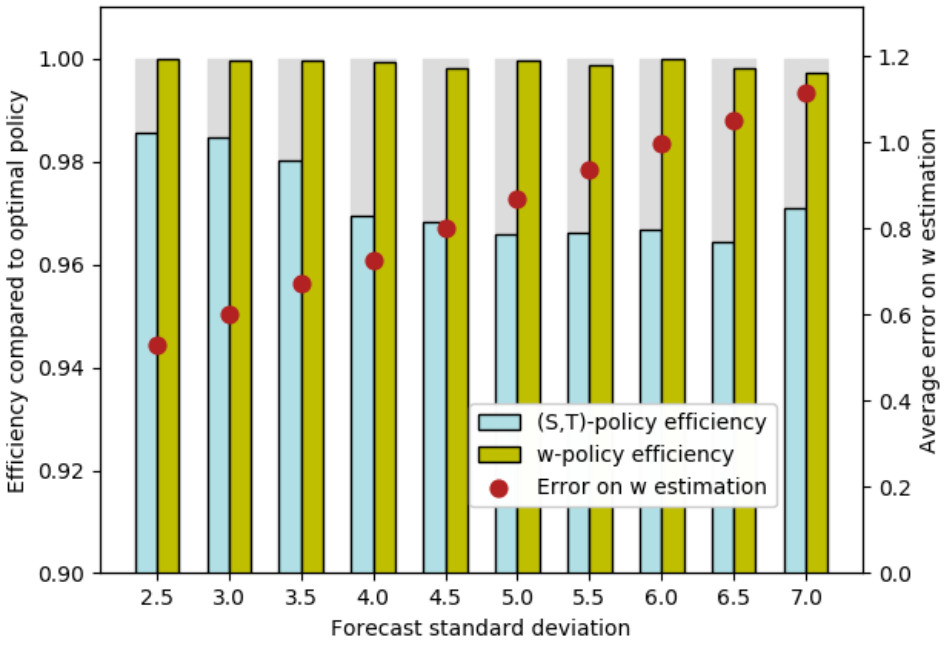


Figure III.7: Impact of the forecast uncertainty.

III.8.3 Reorder zone and policy robustness

In order to explain the superior performance of the w-policy compared to the (S,T)-policy, we run an experiment similar to the previous one. The settings are identical, but the demand forecast is set to a Poisson distribution with a parameter $\lambda = 5$. The figure III.8 displays at which pre-reception stock levels an order is triggered, according to the three considered policies. In this situation, the best T value according to the (S,T) policy is equal to 8.

The greyed area includes all the states (the combination of stock level for the items) where the optimal policy would recommend to place a new order. We can see that whenever one of the item is close to stock-out, it is justified to place an order even if the other item is already properly stocked. The w-policy reorder frontier follows the same pattern as the optimal reorder frontier, only being slightly more conservative when both items have non-negligible chances of stock out.

The bottom left corner of the figure includes all the states in which the sum of the stock levels is lower or equal to 8. This zone is the reorder zone of the (S,T) policy. This illustrates one of the limitation of the (S,T) policy as, if one item has a pre-reception stock superior to T and the other one is stocked-out, the (S,T) policy suggests to place no order. This can lead to unnecessary long stock outs, as a retailer following the (S,T) policy would wait for the second item stock to decrease below 8 before placing a new order.

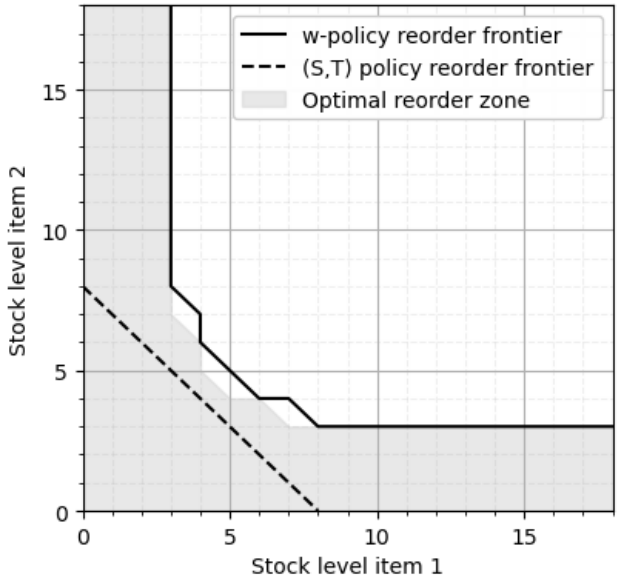


Figure III.8: Reorder frontiers in a two items scenario.

III.8.4 Impact of the myopic approximation

Apart from the assumption on the inaction window, the main assumption of the w-policy lies in the estimation of the sign of $q^*(x, a_Q) - q^*(x, a_\emptyset)$. This assumption remains largely true if the holding costs remain small compared to the margin, meaning that it is rarely worth risking significant stock-outs in order to reduce holding costs. The goal of our third experiment to exhibit this limitation. Similarly to experiment 1, we set Q to 25 and the margins to $m^1 = m^2 = 1$. The weekly demand probability distributions are Poisson distributions with parameters $\lambda^i = 5$.

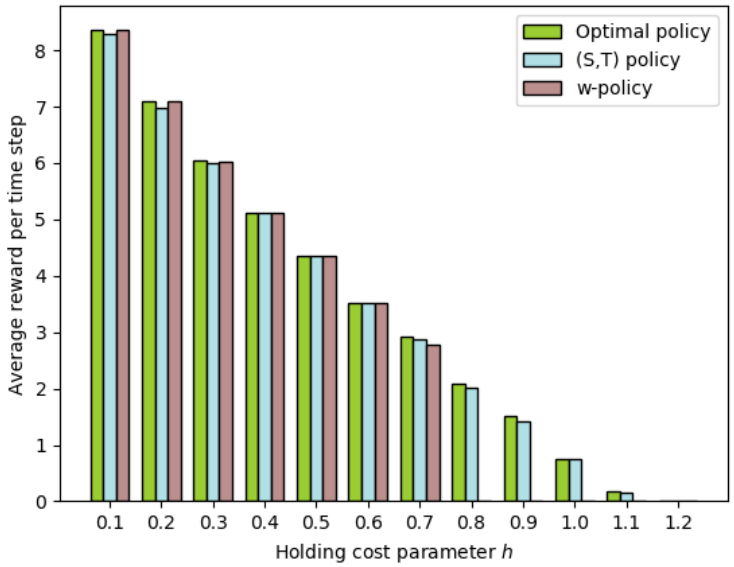


Figure III.9: Impact of the holding cost parameter h.

The figure III.9 displays the average reward per time step for the optimal and the w-policy, over a thousand episodes of 52 weeks. The w-policy exhibits near-optimal performance when the

holding costs parameters h^i remain small compared to the margins m^i. When the h^i exceed 0.7, the w-policy almost never suggests to reorder, while the (S, T) and optimal policy still manage to generate significant profits up until the h^i exceed 1.1. When using the w-policy, one must be aware of this limitation.

III.8.5 Seasonal demand

We finally set up a last toy experiment to assess how the w-policy performs in a variable demand environment. The monthly demand probability distributions used for the simulation are Poisson distributions with parameters λ_t^i. We simulate a seasonal pattern for the two items: item 1 is mostly sold in summer ($\lambda_t^1 = 7.5$ during summer months, $\lambda_t^1 = 2.5$ during winter, and $\lambda_t^1 = 5$ in spring and fall), while item 2 is mostly sold in winter ($\lambda_t^2 = 2.5$ during summer months, $\lambda_t^2 = 7.5$ during winter, and $\lambda_t^2 = 5$ in spring and fall). We set $m^1 = m^2 = 1$, $h^1 = h^2 = 0.1$, $l = 0$.

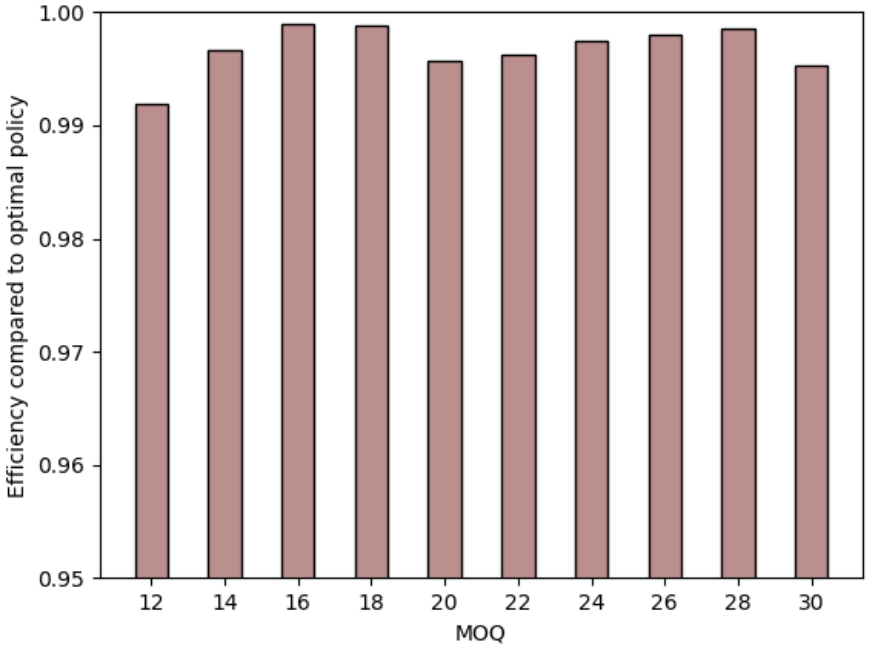


Figure III.10: w-policy efficiency with seasonal demand.

The figure III.10 displays the efficiency of the w-policy compared to the optimal policy, for increasing large values of Q, over a thousand episodes of three years. The efficiency remains above 99% in these settings. This example illustrates the capability of the w-policy to provide good results on the non-stationary demand instance of the MOQ problem.

III.9 Results on large scale datasets

The reason for the development of the w-policy was that, to the best of our knowledge, there currently exists no satisfying solution to the multi-item MOQ problem with variable demand. The primary advantages of the w-policy is therefore its ability to both deal with variable demand and scale to large instances of the MOQ problem. To demonstrate its ability to do so, we tested the w-policy on real large scale datasets, obtained from Lokad's clients. These datasets are not public. The demand forecasts for these datasets were provided by a different group of experts working at Lokad.

III.9.1 Dataset 1

The first goal of the this experiment is to quantify how much performance is affected by this sampling approximation of the forecasts. The second goal is to validate that the w-policy remained competitive with the (S, T) policy on real data, and on a larger scale (where the optimal policy computation is intractable). We used a data set provided by La Redoute, a French multi channel retailer operating in 26 countries and having more than 10 million active customers. From this data set we extracted the margin and the estimated holding costs of 134 items. The (S, T) policy being applicable only to stationary demand models, a stationary weekly demand forecast was generated from a three-years history of recorded sales. The weekly demand probability distribution for each item is a Poisson distribution with a parameter λ^i. The initial pre-reception stocks have been set to zero for every item. The sampling based forecasts are approximated using a hundred samples per item.

The figure III.11 displays the average reward per time step for an episode of 52 weeks, repeated a hundred times with different seeds. The error bar displayed is the variance over the hundred trials. The figure shows that the w-policy proves to be performing as well as the state of the art (S, T) policy on the stationary demand case. The limitations of the (S, T) policy identified with figure III.8 barely have an effect in this scenario, since the demand forecasts are perfectly accurate and no human error interferes with the policy, for the sake of the experiment. It is however crucial to note that in a real world supply chain, it is completely reasonable to expect accidental extreme overstocks on certain references, may it be due to human error or forecasting inaccuracies.

The sampling based approximation of the demand distributions does not appear to significantly impact the performance. This conclusion makes the w-policy easier to scale to larger problems, as demonstrated in the fifth experiment. The figure III.12 displays the average reward, holding costs, and opportunity cost of missed sales generated over the hundred trials when the w-policy was followed. These values are normalized by the maximum reward that would have been generated if the post-reception stock levels were always exactly equal to the observed demands (all the demand is covered and there is no stock remaining at the end of the time step). As expected, the figure shows that when the MOQ increases, the reward decreases due to the increase of holding costs.

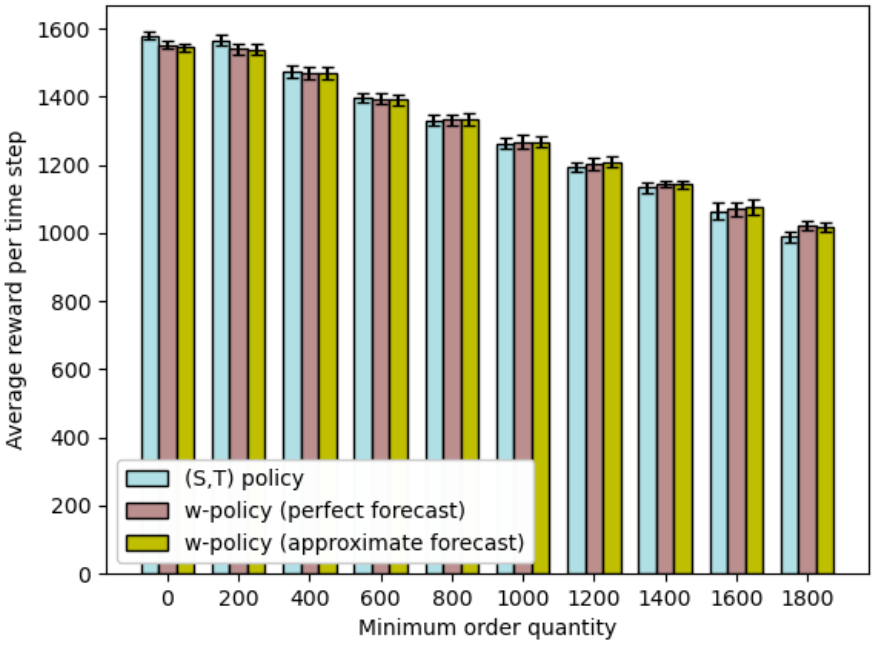


Figure III.11: Policies performance in a 134 items scenario.

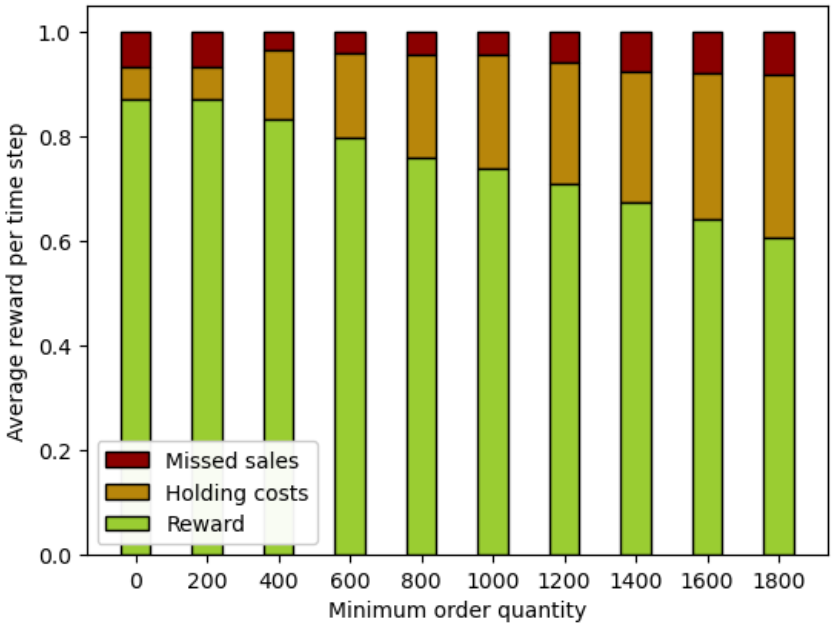


Figure III.12: Ratios of reward, holding costs, and missed sales.

III.9.2 Dataset 2

In most real world supply chains, the demand is actually seasonal and variable over time. The goal of this experiment is to validate the ability of the w-policy to scale up to very large scale real-world MOQ problems, while being able to leverage non-stationary demand forecasts.

For this experiment, we used another data set provided by La Redoute, which includes the sales history, sell prices, and estimated holding costs of 11 607 items. These items are clustered into 17 categories, each category being associated to a seasonality profile. The weekly demand probability distributions used for the simulation are Poisson distributions with parameters λ_t^i. We considered $\lambda_t^i = \lambda^i \, seasonality^i(t)$, with λ^i being the average weekly demand over the last three years, and $seasonality^i$ being the normalized seasonality profile over a year associated to the cluster of item i. The forecasts f_t^i are provided to the w-policy under the shape of a hundred randomly drawn samples per item (the samples are drawn from the Poisson distributions). The total number of units sold on average over a year is approximately equal to 730 000. Therefore around 14 000 units are sold on an average week.

Similarly to figure III.12, the figure III.13 displays the ratios of reward, holding costs, and missed sales for different Q values. These values were obtained after a single episode of 52 weeks, so it is more interesting to consider the trends than the exact values. The plot starts at 0.7 for clarity reasons. As expected, when the MOQ forces higher post-reception stock levels, the reward the holding costs increase. In this scenarios, even for large Q values, the losses induced by both the stochasticity of the system and the imperfection of the w-policy do not exceed 10% of the theoretical maximum attainable reward. The computation of one instance of the w-policy, in these settings, took on average a minute to complete on a Intel Core i7-7700HQ @ 2.80GHz CPU. The fact that it is possible to compute the w-policy on a desktop computer in such a short amount of time demonstrates its ability to scale to the largest instances of MOQ problems encountered in real supply.

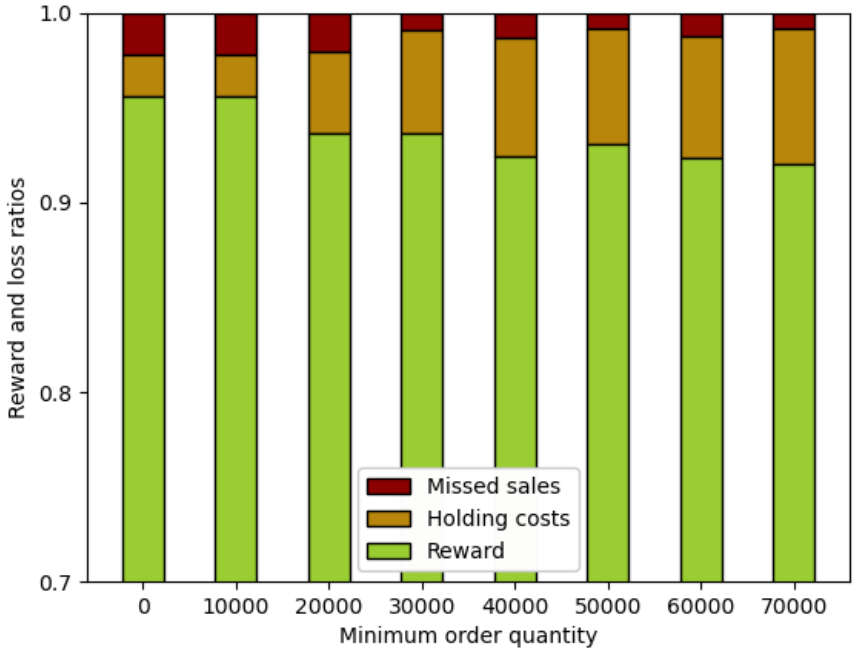


Figure III.13: Ratios of reward, holding costs, and missed sales.

Conclusion

In this chapter, we introduced the w-policy, a heuristic tailored to tackle the multi-item variable demand MOQ problem. The w-policy leverages the problem analysis of chapter II to make suitable assumptions that drastically reduce the complexity of the problem. The core idea of the w-policy is to make simplifying assumptions on the future reorders (the inaction window assumption and the 'oracle assumption'), so that any order increment contribution to the reward can be quantitatively compared to the other items order increment contributions. This makes the computation of the potential best order tractable. The expected reward when applying this best potential order is then compared to the expected reward of delaying the order. Again, simplifying assumptions need to be applied to make the computation tractable, notably a myopic assumption that ignores the long term impact on the expected reward.

Using both toy experiments and real large scale data sets, we demonstrated the near optimal performance of the policy in a wide variety of settings, and demonstrated its ability to solve large scale instances of this problem like no other existing method. Still, in order to make its computational cost reasonable, the aforementioned simplifying assumptions had to be introduced. We exhibited the associated limitations of the w-policy, and showed that the myopic assumption was the most impactful and could lead to unsatisfying performance under some specific settings. In the next chapter, we attempt to overcome this limitation by utilizing state of the art reinforcement learning techniques. The reason for using reinforcement learning algorithms is that they do not typically require simplifying assumptions like the w-policy does, and are in consequence not subjected to the limitations in terms of scope of applicability.

Appendix 1: $\delta\tilde{q}(x^i, a^i, w)$ formula demonstration

We introduce the notation $\mathcal{D}_{t_0}^{t_1}$ as the sum of the observed demand of item i between t_0 and t_1, if the item i considered is obvious. We also define $\mathcal{D}_{I_2}$ as the total demand of item i on I_2:

$$\mathcal{D}_{t_0}^{t_1} = \sum_{t=t_0}^{t_1} D_t^i,\ \mathcal{D}_{I_2} = \sum_{t \in I_2} D_t^i$$

The goal of this appendix is to demonstrate that :

Under the oracle assumption, i.e. if $\forall t' \in I_3$:

$$\hat{S}_{t'}^i = \max(D_{t'}^i, S_{t+l}^i + a_t^i - \mathcal{D}_{t+l}^{t'-1})$$

Then :

$$\Delta\tilde{q}(x, a, w) = \sum_{i=1}^{I} \Delta\tilde{q}^i(x^i, a^i, w)$$

With :

$$\Delta\tilde{q}^i(x^i, a^i, w) = \sum_{k=1}^{a^i} \delta\tilde{q}^i(x^i, k, w)$$

And with:

$$\delta\tilde{q}^i(x^i, k, w) = P(\mathcal{D}_{I_2} \geq S_{t+l}^i + k | x_t^i = x^i)\, m^i + \sum_{t' \in I_2 \cup I_3} P(\mathcal{D}_{t+l}^{t'} < S_{t+l}^i + k | x_t^i = x^i)\, h^i \qquad \text{(III.14)}$$

We split the equation III.14 into three components:

$$\delta\tilde{q}^i(x^i, k, w) = P(\mathcal{D}_{I_2} \geq S_{t+l}^i + k | x_t^i = x^i)\, m^i + \sum_{t' \in I_2} P(\mathcal{D}_{t+l}^{t'} < S_{t+l}^i + k | x_t^i = x^i)\, h^i + \sum_{t' \in I_3} P(\mathcal{D}_{t+l}^{t'} < S_{t+l}^i + k | x_t^i = x^i)\, h^i \qquad \text{(III.15)}$$

The green term is the contribution of the k-th unit of the order to the expected reward generated by additional potential sales during I_2. The blue and red terms are the contribution of the k-th unit of the order to the expected additional holding costs generated during I_2 and I_3, respectively.

To demonstrate this result, we go back to the definition of the q-values as expected reward. $\Delta\tilde{q}(x, a, w)$ can be expressed as a sum over each item, and each time period in $I_2 \cup I_3$ of the difference of expected reward between the two scenarios.

$$\Delta\tilde{q}(x, a, w) = \sum_{i=1}^{I} \sum_{t' \in I_2 \cup I_3} \mathbb{E}[R_{t'}^i(X_{t'}^i, A_{t'}^i, X_{t'+1}^i) | a_t = a, w_t = w] - \mathbb{E}[R_{t'}^i(X_{t'}^i, A_{t'}^i, X_{t'+1}^i) | a_t = a_0, w_t = w] \qquad \text{(III.16)}$$

Step 1: We first consider the difference of expected reward generated during I_2. The goal is to demonstrate the formula of the green and blue terms in equation III.15. As a reminder, the expression of the reward per item per time step is equal to:

$$R_{t'}^i(X_{t'}^i, A_{t'}^i, X_{t'+1}^i) = m^i \min(D_{t'}^i, S_{t'}^i) + h^i S_{t'+1}^i$$

The S^i_{t+l} variables are identical in the two scenarios since the impact of an order is delayed by l time steps. On the other hand, the $\hat{S}^i_{t'}$ variables for $t' \in I_2$ are not identical in the two scenarios. The $\hat{S}^i_{t'}$ variables can be expressed as a function of the stocks S^i_{t+l} and the actions a^i_t, since no order is placed between $t+1$ and $t+w$. Therefore, for both scenarios, $\forall t' \in I_2$:

$$\hat{S}^i_{t'} = \max(0, S^i_{t+l} + a^i_t - \mathcal{D}^{t'-1}_{t+l})$$
$$\hat{S}^i_{t'+1} = \max(0, S^i_{t+l} + a^i_t - \mathcal{D}^{t'}_{t+l})$$

The difference of expected reward per item per time step (in I_2) can therefore be expressed as:

$$\begin{aligned}
&\mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a, w_t = w] - R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a_0, w_t = w)] \\
&\quad = \mathbb{E}[m^i \min(D^i_{t'}, \max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'-1}_{t+l})) + h^i \max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'}_{t+l}) \\
&\qquad\qquad - m^i \min(D^i_{t'}, \max(0, S^i_{t+l} - \mathcal{D}^{t'-1}_{t+l})) - h^i \max(0, S^i_{t+l} - \mathcal{D}^{t'}_{t+l})]
\end{aligned} \tag{III.17}$$

The green terms correspond to the difference in reward generated by sales (if we do not order, we may sell less units than if we do) during I_2. The blue terms correspond to the difference of holding costs generated during I_2 (if we do not order, we may pay less holding costs than if we do).

We first focus on the reward generated by sales (green term of the equation). The number of sales during I_2 is equal either $\hat{S}^i_{t+l}$ if the item i is out of stock at the end of I_2, or equal to the total demand over I_2, noted $\mathcal{D}_{I_2}$ if item i is not stocked out. This result is easily demonstrated recursively. The formulation of the number of sales over I_2 as a sum of sales over every timesteps can therefore be simplified:

$$\sum_{t' \in I_2} \min(D^i_{t'}, \hat{S}^i_{t'}) = \min(\mathcal{D}_{I_2}, \hat{S}^i_{t+l})$$

The difference of reward generated by sales during I_2 (corresponding to the sum over I_2 of the green terms in equation III.17) is therefore equal to :

$$\begin{aligned}
&\mathbb{E}\left[m^i \sum_{t' \in I_2} \min(D^i_{t'}, \max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'-1}_{t+l})) - \min(D^i_{t'}, \max(0, S^i_{t+l} - \mathcal{D}^{t'-1}_{t+l}))\right] \\
&\qquad = m^i \mathbb{E}\left[(\min(\mathcal{D}^{t+l+w-1}_{t+l}, S^i_{t+l} + a^i) - \min(\mathcal{D}^{t+l+w-1}_{t+l}, S^i_{t+l})\right]
\end{aligned}$$

We then adopt the point of view of individual sale increments:

$$= m^i \mathbb{E}\left[\sum_{k=1}^{\min(\mathcal{D}^{t+l+w-1}_{t+l}, S^i_{t+l}+a^i)} 1 - \sum_{k=1}^{\min(\mathcal{D}^{t+l+w-1}_{t+l}, S^i_{t+l})} 1\right]$$

$$= m^i \sum_{D=0}^{\infty} P(\mathcal{D}^{t+l+w-1}_{t+l} = D) \sum_{k=1}^{s^i_{t+l}+a^i} \mathbb{1}_{k \leq \mathcal{D}^{t+l+w-1}_{t+l}} - \sum_{k=1}^{s^i_{t+l}} \mathbb{1}_{k \leq \mathcal{D}^{t+l+w-1}_{t+l}}$$

Sales that would occur in both scenarios are simplified:

$$= m^i \sum_{k=s^i_{t+l}+1}^{s^i_{t+l}+a^i} \sum_{D=0}^{\infty} P(\mathcal{D}^{t+l+w-1}_{t+l} = D) \mathbb{1}_{k \leq \mathcal{D}^{t+l+w-1}_{t+l}}$$

We finally switch from the point of view of individual sale increments to the point of view of order increments, and how likely it is that additional ordered units are sold:

$$\mathbb{E}\left[m^i \sum_{t' \in I_2} \min(D^i_{t'}, \max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'-1}_{t+l})) - \min(D^i_{t'}, \max(0, S^i_{t+l} - \mathcal{D}^{t'-1}_{t+l}))\right] = m^i \sum_{k=1}^{a^i} P(\mathcal{D}^{t+l+w-1}_{t+l} \geq S^i_{t+l} + k) \tag{III.18}$$

Equation III.18 therefore explicits the computation of the sum over I_2 of the green terms in equation III.17, the expected difference of reward generated by additional sales during I_2. We then focus on the holding costs generated at the end of $t' \in I_2$ (corresponding to the blue terms in equation III.17).

$$\mathbb{E}[h^i(\max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'}_{t+l}) - \max(0, S^i_{t+l} - \mathcal{D}^{t'}_{t+l}))] = \mathbb{E}[h^i(\sum_{k=1}^{S^i_{t+l}+a^i-\mathcal{D}^{t'}_{t+l}} 1) - \sum_{k=1}^{S^i_{t+l}-\mathcal{D}^{t'}_{t+l}} 1))]$$

$$= \mathbb{E}[h^i \sum_{k=S^i_{t+l}-\mathcal{D}^{t'}_{t+l}+1}^{S^i_{t+l}+a^i-\mathcal{D}^{t'}_{t+l}} \mathbb{1}_{k>0})]$$

Again we switch to the point of view of order increments, and how likely it is that additional ordered units generate an additional holding cost h^i at the end of period t':

$$= \mathbb{E}[h^i \sum_{k=1}^{a^i} \mathbb{1}_{S^i_{t+l}-\mathcal{D}^{t'}_{t+l}+k>0}]$$

We deduce from this expression the following equation III.19, that explicits the computation of the expected difference of reward generated by additional holding costs at the end of $t' \in I_2$ (blue term in equation III.17):

$$\mathbb{E}[h^i \max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'}_{t+l}) - h^i \max(0, S^i_{t+l} - \mathcal{D}^{t'}_{t+l})] = \sum_{k=1}^{a^i} P(S^i_{t+l} + k > \mathcal{D}^{t'}_{t+l} | x^i_t = x^i, w_t = w)) h^i \tag{III.19}$$

By combining equations III.17, III.18 and III.19, we have the following expression for the difference of expected reward generated during I_2:

$$\sum_{i=1}^{I} \sum_{t' \in I_2} \mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1}) | a_t = a, w_t = w] - \mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1}) | a_t = a_0, w_t = w]$$
$$= \sum_{i=1}^{I} \sum_{k=1}^{s^i_{\max}} \left(P(\mathcal{D}_{I_2} \geq S^i_{t+l} + k | x^i_t = x^i, w_t = w)\, m^i + \sum_{t' \in I_2} P(\mathcal{D}^{t'}_{t+l} < S^i_{t+l} + k | x^i_t = x^i, w_t = w)\, h^i \right) \tag{III.20}$$

Step 2: We consider the difference of expected reward generated during I_3. The goal is to demonstrate the formula of the red terms in equation III.15.
According to the 'oracle' assumption, we have $\forall t' \in I_3$:

$$\hat{S}^i_{t'} = \max(D^i_{t'}, S^i_{t+l} + a^i_t - \mathcal{D}^{t'-1}_{t+l})$$

Therefore $\hat{S}^i_{t'} \geq D^i_{t'}$ and :

$$R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1}) = m^i D^i_{t'} - h^i S^i_{t'+1} = m^i D^i_{t'} - h^i \max(0, S^i_{t+l} + a^i_t - \mathcal{D}^{t'}_{t+l})$$

Therefore $\forall t' \in I_3$:

$$\begin{aligned}\mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a, w_t = w] - R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a_0, w_t = w)] \\ = \mathbb{E}[m^i D^i_{t'} - h^i \max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'}_{t+l}) - m^i D^i_{t'} + h^i \max(0, S^i_{t+l} - \mathcal{D}^{t'}_{t+l})]\end{aligned}$$

The same number of units is sold at every time step in I_3 whether we place an order or not at time t, since according to our assumption the system is never stocked out during I_3. The greyed terms therefore simplify.

$$= \mathbb{E}[h^i(\max(0, S^i_{t+l} + a^i - \mathcal{D}^{t'}_{t+l}) - \max(0, S^i_{t+l} - \mathcal{D}^{t'}_{t+l}))]$$

We apply the same computation than for the difference of expected holding costs generated during a timestep in I_2 (that leads to equation III.18). The difference of expected reward generated during $t' \in I_3$ is therefore equal to:

$$\begin{aligned}\mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a, w_t = w] - R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a_0, w_t = w)] \\ = \sum_{k=1}^{a^i} P(\mathcal{D}^{t'}_{t+l} < S^i_{t+l} + k | x^i_t = x^i, w_t = w)) h^i\end{aligned}$$

If we sum this formula for every item and every time step in I_3, we get:

$$\begin{aligned}\sum_{i=1}^{I} \sum_{t' \in I_3} \mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a, w_t = w] - \mathbb{E}[R^i_{t'}(X^i_{t'}, A^i_{t'}, X^i_{t'+1})|a_t = a_0, w_t = w] \\ = \sum_{i=1}^{I} \sum_{k=1}^{a^i} \sum_{t' \in I_3} P(\mathcal{D}^{t'}_{t+l} < S^i_{t+l} + k | x^i_t = x^i, w_t = w))\, h^i \quad \text{(III.21)}\end{aligned}$$

Therefore, by combining equations III.16, III.20 and III.21, we can conclude that :

If $\forall t' \in I_3$:

$$\hat{S}^i_{t'} = \max(D^i_{t'}, S^i_{t+l} + a^i_t - \mathcal{D}^{t'-1}_{t+l})$$

Then :

$$\Delta\tilde{q}(x, a, w) = \sum_{i=1}^{I} \sum_{k=1}^{a^i} \delta\tilde{q}^i(x^i, k, w)$$

$$\delta\tilde{q}^i(x^i, k, w) = P(\mathcal{D}^{t+l+w-1}_{t+l} \geq S^i_{t+l} + k | x^i_t = x^i)\, m^i + \sum_{t' \in I_2 \cup I_3} P(\mathcal{D}^{t'}_{t+l} < S^i_{t+l} + k | x^i_t = x^i)\, h^i$$

Appendix 2: w-policy notations and definitions glossary

- Inaction window : duration between an order and the next reorder.
- w is the number of time steps that is assumed to be equal to the inaction window when applying the w-policy.
- w_t is w at time t.
- $q_t^{\pi}(x, a, w)$ is the action value, under the assumption that $x_t = x, a_t = a, a_{t' \in [t+1..t+w-1]} = a_{\emptyset}$ and $a_{t' \geq t+w} = \pi(X_{t'})$.
- I_1, I_2, I_3 are the time periods $I_1 = [t..t + l[$, $I_2 = [t + l..t + l + w[$, and $I_3 = [t + l + w..t_{end}]$.
- $q^*(x, a, w)$ is the action value at time t (the index t is dropped to avoid overloading indexes), under the assumption that $x_t = x, a_t = a, a_{t' \in [t+1..t+w-1]} = a_{\emptyset}$ and $a_{t' \geq t+w} = \pi^*(X_{t'})$.
- $\Delta q^*(x, a, w)$ is the difference between $q_t^*(x, a, w)$ and $q_t^*(x, a_{\emptyset}, w)$.
- $\tilde{\phi}_{t'}^i(k, x^i, \{f_{t''}^i\}_{t'' \in [t..t']}, a^i)$ is the approximation of $P(\hat{S}_{t'}^i \geq k)$ for $t' \in I_3$, under the 'oracle' assumption.
- $\Delta\tilde{q}(x, a, w)$ is the approximation of $\Delta q^*(x, a, w)$ under the 'oracle' assumption.
- $\delta\tilde{q}^i(x^i, k, w)$ is the per item per order increment contribution to $\Delta\tilde{q}(x, a, w)$.
- w' is the number of time steps that is assumed to be equal to the inaction window if no order is placed at t.
- I_2', I_3' are the time periods $I_2' = [t + l..t + l + w'[$ and $I_3' = [t + l + w'..t_{end}]$.
- $\Delta\dot{q}(x, a_Q, w')$ is the myopic approximation of $q^*(x, a_Q, w) - q^*(x, a_{\emptyset}, w')$.
- $\delta\dot{q}^i(x^i, k, w')$ is the per item per order increment contribution to $\Delta\dot{q}(x, a_Q, w)$.
- δ_a is the boolean variable that correspond to the statement 'an order is placed'.
- $\pi_w^+(x)$ is the function that builds the best potential order of size Q according to the w-policy given the state of the system $x_t = x$ and w.

Chapter IV

Reinforcement learning approaches

Introduction

Reinforcement learning can be seen as an extension of stochastic dynamic programming when conventional techniques do not scale: estimates of the value functions or of the optimal policy are incrementally improved by interacting with the environment. Thus, they are learnt from experience. Most recent successes rely on the use of deep neural network as value or policy function approximators. A prime example of the power of reinforcement learning combined with deep neural network was provided in [90], where it was demonstrated that beyond-human abilities were achievable on the very challenging game of go. In light of this success, one could expect the MOQ problem to be rather easily solved by state of the art reinforcement learning methods: the complexity of the MOQ problem may seem unimpressive compared to the game of Go. As detailed in section IV.1, there are however several limitations that prevent state of the art techniques from being applied in practice. As a consequence, the objective of this chapter is not to solve the MOQ problem using only reinforcement learning, and we had to settle for a more modest goal. In the previous chapter, we showed that the w-policy is capable of providing the near optimal allotment of the order very robustly. Its main weakness lies in its inability to correctly assess (in some specific settings) whether an order should be placed or not. Our goal is therefore to fix this limitation using reinforcement learning.

We introduce the core reinforcement learning and deep reinforcement learning concepts in section IV.1, and detail the limitations of reinforcement learning approaches when applied to the MOQ problem. In section IV.2, we introduce an hybrid policy, that mixes reinforcement learning and the w-policy described in chapter III. The idea is to keep using the efficient allotment algorithm (the w^+ bloc in figure III.4) of the w-policy, but to replace the less robust binary decision process δ_a (do we delay the order or not ?) by a state of the art reinforcement learning algorithm. In section IV.3 and IV.4, we provide guidelines to implement this hybrid policy in practice, and to make it scale to very large datasets. In section IV.5, we demonstrate the ability of the hybrid policy to scale on very large instances of the problem (up to ten thousand items). We also present its performance compared to the w-policy: on toy problems, we show that the hybrid policy has a larger scope of applicability than the w-policy. On real datasets however, the hybrid policy did not appear to be able to outperform the w-policy.

IV.1 Reinforcement learning literature and MOQ problem

In this section, we first provide a quick overview of important reinforcement learning and deep reinforcement learning concepts. We then focus on the reinforcement learning literature that could potentially be relevant to solve the MOQ problem. A specificity of the MOQ problem compared

to other reinforcement learning environments is that its state space is likely to change frequently. Indeed, it is frequent in real supply chains to add or remove items from the inventory. Every time such a change occurs, the entire model needs to be trained again. This means that weeks long training times are not practical in real supply chains, and training a new effective model should therefore not exceed a few hours. Similarly, training stability is also a problem if training times are a limiting factor, since a worse stability needs to be compensated for with more training trials.

In chapter II and III, we considered episodes with finite number of time-steps. The w-policy utilizes the forecasts $f^i_{t'\geq t}$ over the rest of the episode to compute its solution. The different state and action value functions were defined relatively to t. For the same state x, defined by the stocks and stocks on order, $v_t(x)$ would not necessarily be equal to $v_{t'}(x)$, if $t' \neq t$. This choice of model is a problem when using reinforcement learning techniques, and specifically deep reinforcement learning. Indeed, the input size of neural networks can not be variable. If one wants to utilize the same neural networks to approximate the policy or value functions at different time steps, these functions need to have fixed size inputs. One could use a different network for every time step, but this would drastically increase the complexity of the architecture and its training.

Instead, we change our model so that it fits the framework of Markov decision processes, with an infinite number of time steps. Policies and value functions in this framework take as inputs the variables that are susceptible to change over time. For the MOQ problem, it includes the stock levels s^i_t, the stock on order $o^i_{t'\in[t,t+l]}$, but also the forecasts for the rest of the episode $f^i_{t'\geq t}$.

It is impractical to consider an infinite number of forecasts $f^i_{t'\geq t}$ as input of our policy and value functions. We define a forecasting horizon h_f. Only the forecasts between t and $t+h_f$ will be provided as inputs. The forecasting horizon obviously needs to be chosen depending on the forecasts and Q. It should be long enough, so that the forecasts beyond $t+h_f$ have a negligible impact on the decision taken at time t. Thus in this chapter the forecasts are considered part of the state variable x, in the state space X. The state space is therefore the same at every time step, but the forecasts can be variable over time. The minimum order quantity Q and the economic parameters m^i and h^i are assumed constant over time: these values are not inputs of the policy, and do not need to be included in the state space. For the remainder of this chapter, we define the input state x as the cartesian product of the stocks, the stocks on order, and the forecasts over h_f for every item. x_t can therefore be written:

$$x_t = \left\{(s^i_t, \{o^i_{t'}\}_{\forall t'\in[t..t+l-1]}\}, \{f^i_{t'}\}_{\forall t'\in[t..t+h_f]})\right\}_{\forall i\in[1..I]} \tag{IV.1}$$

With this model and under the assumption that h_f is large enough, the value function becomes independent from time, and can be written as:

$$v^\pi(x) = \mathbb{E}\left[G_t | x_t = x, a_{t'\geq t} = \pi(x_{t'})\right], G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} R_{t'}$$

With $\mathbb{E}\left[G_t | x = x, a_{t'\geq t} = \pi(x_{t'})\right]$ being the expectancy over the possible outcomes of the stochastic process, but also over every time step t.

One can note the necessity of the γ parameter ($\gamma \in]0,1[$), which ensures that $v^\pi(x)$ remains theoretically bounded (as a sum of exponentially decreasing terms). The side effect is that long term reward are under-valued compared to the short term reward. The reward generated at time $t+\Delta T$ is therefore $\gamma^{\Delta T}$ times less considered that the reward generated at time t.

IV.1.1 Reinforcement learning concepts

In this subsection, we provide a quick overview of important reinforcement learning concepts. The goal is to determine the best actions to perform in an environment, in order to maximize an objective

function. The main idea of reinforcement learning is to repeatedly interact with the environment, to learn the best possible policy from these experiences. Notably, determining the value function associated to a policy π is key to improving it, and many reinforcement learning algorithms rely on value function approximations.

Monte-Carlo approximation

The simplest method to approximate an expected value is to resort to a sampling approximation, and use a Monte-Carlo method [42]. One can apply the policy π, from state x at time t, and collect the observed $R_{t' \in [t, t_{end}]}$ values until the end of the episode. The time t at which x is reached has no impact on the expected reward as we defined it. We note k_{ep} the number of episodes generated, and we define $\bar{v}^\pi(x)$ as an approximation of $v^\pi(x)$:

$$\bar{v}^\pi(x) = \frac{1}{k_{ep}} \sum_{k=1}^{k_{ep}} G_t^k = \frac{1}{k_{ep}} \sum_{k=1}^{k_{ep}} \sum_{t'=t}^{t_{end}} \gamma^{t'-t} R_{t'}^k$$

We note $R_{t'}^k$ and G_t^k the reward and gain obtained at episode k. The law of large numbers guarantees that $\bar{v}^\pi(x)$ converges toward $v^\pi(x)$ when k_{ep} tends to infinity. One can also dynamically update the value function approximation using for example a constant step size parameter $\alpha \in [0, 1]$:

$$\bar{v}^\pi(x) \leftarrow \bar{v}^\pi(x) + \alpha(G_t^k - \bar{v}^\pi(x)) \tag{IV.2}$$

With $\alpha \in [0, 1]$. α can be seen as a exponential smoothing of the weight of the observations, that prioritizes more recent observations. The advantage of such approximations is that one does not need to wait until all the episodes are finished before having access to an updated value for $\bar{v}^\pi(x)$.

Monte-Carlo methods must still wait until the end of an episode before updating the $\bar{v}^\pi(x)$ value. One central idea of reinforcement learning is to get rid of this constraint by utilizing temporal difference (TD) learning [94].

Temporal Difference (TD) learning

The idea of TD-learning is that one does not need to compute the entire episode to update $\bar{v}^\pi(x_t)$ if one has access to an approximation of $v^\pi(x_{t'})$, with $t' > t$ and $x_{t'}$ being the state observed at t'. For simplicity sake, we consider $t' = t + 1$, which is called TD(0). G_t^k is approximated by:

$$G_t^k \approx R_t^k + \gamma \bar{v}^\pi(x_{t+1})$$

When replacing G_t^k in equation IV.2, it becomes:

$$\bar{v}^\pi(x_t) \leftarrow \bar{v}^\pi(x_t) + \alpha(R_t^k + \gamma \bar{v}^\pi(x_{t+1}) - \bar{v}^\pi(x_t))$$

In this case, α can be seen as learning rate. This update rule combines the sampling approximation of the Monte-Carlo method with the recursive evaluation of dynamic programming methods (such as the backward recursion). The main advantage is that this rule does not require to compute the entire episode to update $\bar{v}^\pi(x_t)$.

It may appear counter intuitive to use $\bar{v}(x_{t+1})$ to improve the estimate of $\bar{v}(x_t)$ since $\bar{v}(x_{t+1})$ may not be a good estimate itself. Such algorithms therefore rely on the hypothesis that $\bar{v}(x_{t+1})$ will itself be improved during training.

TD-learning can be applied to compute an approximate action value function similarly to the state value function, for example using the SARSA algorithm [84]. The SARSA algorithm assumes that the action at $t + 1$ will be selected according to the π policy:

$$G_t^k \approx R_t^k + \gamma \bar{q}^\pi(x_{t+1}, \pi(x_{t+1}))$$

It translates into the following update rule for the action value approximation $\bar{q}^\pi(x_t, a_t)$:

$$\bar{q}^\pi(x_t, a_t) \leftarrow \bar{q}^\pi(x_t, a_t) + \alpha(R_t^k + \gamma \bar{q}^\pi(x_{t+1}, \pi(x_{t+1})) - \bar{q}^\pi(x_t, a_t))$$

ε-greedy policy

We presented the Monte carlo approximation and TD-learning, which are two techniques that help evaluate the performance of a policy. The goal of reinforcement learning is however to improve the policy, not to simply evaluate it.

A possible approach to apply TD-learning to improve a policy π is to consider the action value function $q^\pi(x, a)$ rather than a state value function. The natural policy π one could derive from the action value function is the greedy policy, that selects the action that maximizes $\bar{q}(x, a)$: $\pi(x) = \arg\max_a(\bar{q}(x, a))$. The problem is that if the training samples are generated following the greedy policy, many state and actions may never be explored (depending on the initialization of the q-values). The greedy policy never tries actions that are deemed sub-optimal by its imperfect $\bar{q}(x, a)$ evaluation. Such actions are therefore never selected, and their associated $\bar{q}(x, a)$ values are never updated.

One solution to this exploration problem is to introduce randomness in the policy. The simplest example is the ε-greedy policy, which selects the greedy action with a probability $1 - \varepsilon$, but can also selects (with a probability ε) a completely random action. ε may change during learning, as the policy first needs to explore a lot, and then needs to focus more and more on exploiting the more promising actions (ε decreases towards 0). The value of ε represents the trade-off between exploration and exploitation [95].

Q-learning

A important idea in reinforcement learning is that it is not necessary for the training samples to be generated by the policy that is being trained. One can define the target policy, which purpose is to maximize the expected gain, and the behavior policy, which purpose is to generate the training examples. Algorithms that utilize examples generated by a behavior policy to train the target policy are called 'off-policy'. Q-learning [105] is a very important example of such an algorithm, that also uses TD learning. The idea of Q-learning is to approximate directly $q^*(x_t, a_t)$. It does so by assuming that the action at $t + 1$ will be selected according to the greedy policy when applying TD-learning:

$$G_t^k \approx R_t^k + \gamma \max_{a_{t+1}} \bar{q}^*(x_{t+1}, a_{t+1})$$

Which translates into the following update equation:

$$\bar{q}^*(x_t, a_t) \leftarrow \bar{q}^*(x_t, a_t) + \alpha(R_t^k + \gamma \max_{a_{t+1}} \bar{q}^*(x_{t+1}, a_{t+1}) - \bar{q}^*(x_t, a_t))$$

One can note that the policy that was used to generate the a_t does not appear in the formula. The way the training example tuple $(x_t, a_t, R_t^k, x_{t+1})$ was generated does not matter: one can improve the action value estimate $q^*(x_t, a_t)$ regardless.

Tabular approaches and parametric models

Up until this point, we focused on different methods and algorithms that help improve value function approximations from experience. We now consider how the $\bar{v}$ and $\bar{q}$ functions map the state x or the state-action pair (x, a) to an expected reward in practice.

The simplest solution is to keep in memory a value $\bar{v}$ (or $\bar{q}$) for every possible state (or state-action pair). This implies that the state space is discrete, and rather limited in size. Such approach suffer from the curse of dimensionality, similarly to classical stochastic dynamic programming.

The other possibility is to use parametric models of the value functions. The goal therefore becomes to update the parameters θ of the parametric model to fit. The aforementioned concepts (Monte Carlo, TD-learning, ε-greedy policy and Q-learning) remain valid: they provide the gradient necessary to update the θ parameters.

There exist many different possible parametric models, including linear, piecewise-linear, or more complex linear models [95]. The most popular way of approximating value functions has lately be to use artificial neural networks [72]. Their advantage is that they are theoretically capable of approximating any function according to the universal approximation theorem [50], while being differentiable and easy to train using the backpropagation algorithm.

IV.1.2 Discrete action spaces and Deep Q-networks

State of the art reinforcement learning methods for small discrete action space rely on deep Q-learning [72,90]. The idea is to approximate the action value function $q(x, a)$ with a neural network. The input of the neural network is the state of the system x, and the output layer has a single neuron for each valid action.

There is therefore a finite number of possible actions a. We note this number N, with $N = Card(A)$. One can enumerate and associate a number k to each possible action, thus one can write $A = \{a^k\}_{k\in[1..N]}$. In the context of this section, a^k ($k \in [1..N]$) therefore represents a potential action, and should not be confused with the per-item orders a^i ($i \in [1..I]$) that is specific to the MOQ problem. When applied to the MOQ problem, an action a^k is one of the possible combination of all the a^i: $a^k = \{a^i\}_{i\in[1..I]}$.

The output value of the neuron associated to the action a^k is the estimated action value $\bar{q}(x, a^k)$. The policy associated to the deep Q-network is then simply to select the action that maximizes $\bar{q}(x, a)$ (see figure IV.1).

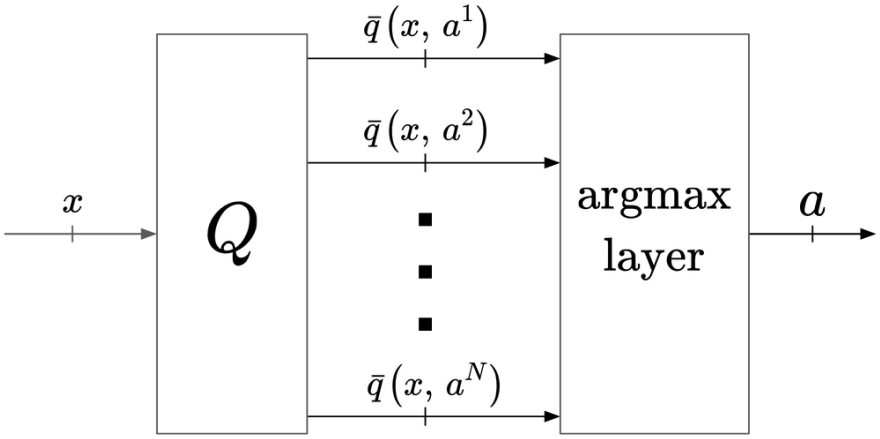


Figure IV.1: Deep Q-network

The neural network is initialized randomly, and has to be trained to minimize the difference between $q(x, a^k)$ and $\bar{q}(x, a^k)$, whenever a^k is selected. The $\bar{q}(x, a^k)$ are trained by temporal difference learning. In terms of loss, it can be written as:

$$\mathcal{L} = \|\bar{q}(x_t, a^k) - R(x_t, a^k, x_{t+1}) - \gamma \max_{a_{t+1} \in A} \bar{q}(x_{t+1}, a_{t+1})\|$$

With a^k being the action selected at time t. The main limitation of this method when applied to the multi-item MOQ problem is that the action space size is exponential with the number of items. We take the example of a simple MOQ problem with I items, with replenishment orders taking discrete values from 0 to 99. The discrete action space size is equal to 100^I, as it is the number of possible combination of orders. It is unrealistic to build and train neural networks with one output neuron per possible action unless only a couple of items are considered.

IV.1.3 Deterministic policy gradient (DPG)

When dealing with large scale discrete action spaces, the curse of dimensionality can sometimes be alleviated by the use of a well chosen continuous approximation of its action space [29]. Our focus up to this point were methods relying on approximating the value functions, in order to deduce the best action to take in any given state from these functions. Policy based approximations instead try to directly approximate the policy. The state of the system is provided as an input, and the output is the recommended action. In the case of continuous actions, the policy neural network (the actor) directly outputs a vector of continuous values. These values are the actions selected for a given input state. The policy is deterministic. The policy is generally trained through gradient ascent, to increase the expected reward. The problem is that the analytic expression of the gradient is generally not available.

If the system is differentiable, one can use the deterministic policy gradient (DPG) [89] to compute an estimate of the gradient. Indeed, the gradient that would increase the reward on the considered training example (a single outcome of the stochastic process) can be computed. Each example suggests a different direction of policy improvement. The idea of the DPG is that the average gradient of the training examples provides a noisy estimate of the 'true' gradient.

We can approximate the MOQ problem to make use of these type of approach, which can alleviate the curse of dimensionality of some discrete action spaces [29]. If there exist a natural way to map a continuous action space of dimension D into the discrete action space of cardinality $C >> D$, one can use a policy that outputs a continuous action that is then mapped to the corresponding discrete action.

Applicability to the MOQ problem

We can apply this idea to MOQ problem. We re-consider the example of a simple MOQ problem with I items, with replenishment orders taking discrete values from 0 to 99. The discrete action space size is equal to 100^I, but if we approximate the policy with a continuous policy, the action space is simply the combination of the I continuous ordered quantities (one for each item). When the order is actually placed, the continuous values are transformed into the closest discrete actions. While we theoretically extended the size of the action space, in practice we only need I output neurons to generate a policy. This is a massive improvement compared to the 100^I neurons necessary to build a policy from the deep Q-network.

The problem however is that the neural network can output an action such as $0 < \sum_{i=0}^{I} a^i < Q$, which violates the MOQ constraint. The action space has a discontinuity between not ordering and ordering at least Q units. This discontinuity is illustrated by figure IV.2, for an example with only two items and $Q = 10$.

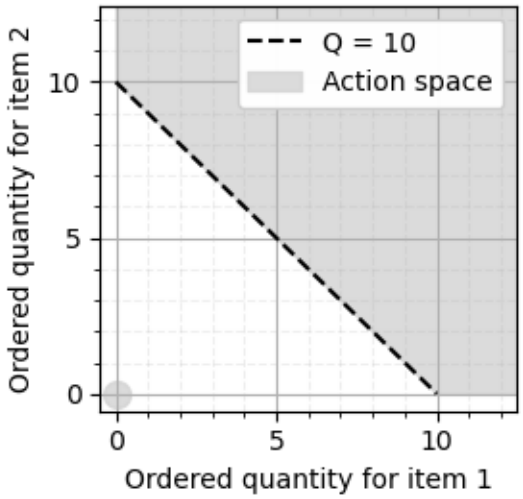


Figure IV.2: Example of continuous action space in a two items scenario

In order to alleviate this issue, the network could be trained to avoid illegal moves by penalizing them. One could for example add a term to the loss that penalizes selecting an action linearly with its distance to the closest legal move. The problem with such approaches is that legal actions are separated by an illegal zone in the action space. A policy trained with gradient ascent is unable to 'learn' to jump over such a zone, since gradient ascent theoretically converges toward the closest local optimum. The inability to use the gradient ascent to train policies to solve the MOQ problem is illustrated with an example in figure IV.3. If there was no MOQ, the optimal order size would be smaller than Q. Given the presence of the MOQ, the optimal action is to order nothing. If the policy was initialized to order at least Q units, the gradient will push the policy away from ordering less than Q units: the policy will never converge towards ordering zero unit, even if is more profitable.

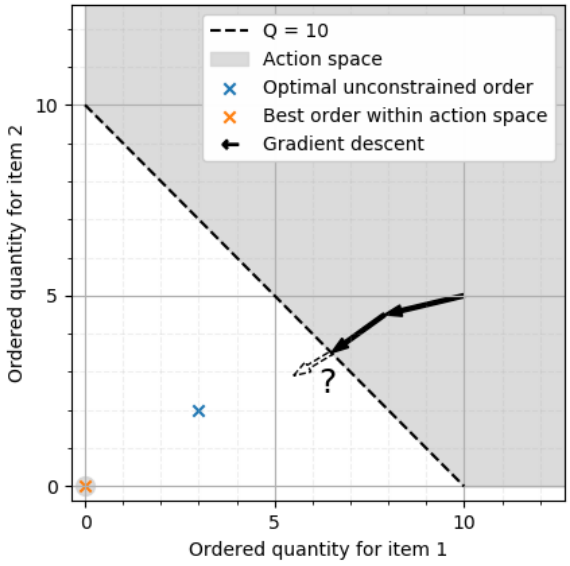


Figure IV.3: Illustration of the convergence difficulty

IV.1.4 Actor-critic architectures for continuous actions

The deep deterministic policy gradient algorithm (DDPG) [68] extends the scope of applicability of the DPG to systems that are unknown or not differentiable. The gradient associated to a training example is in these cases not directly available. Instead, two networks are used in an actor critic architecture: the actor outputs the action given the state, and the critic outputs the estimated action value. The critic network network has both the state of the system and the output of the actor network as inputs. It will output the associated action-value estimate and be trained by temporal difference learning. Its purpose is to provide a differentiable estimate of the value function (and therefore an estimate of the 'true' gradient): the actor can then be trained by gradient ascent to maximize the output of the critic network. When it comes to the MOQ problem, the problem of the discontinuity of the action space illustrated with figure IV.3 is not solved by this method. We will however see in section IV.1.5 that the DDPG was the inspiration for mixed discrete-continuous action space methods.

A different approach was suggested in [40], with the introduction of the normalized advantage function (NAF). Instead of trying to approximate the policy gradient, the authors use an estimate of the best action to train the policy network, similarly to supervised learning (where each training input is associated to a training output). The first idea is to separate the action value function into the state value function (that depends only on the state) and the advantage function (that represents the expected reward lost from electing an action). The second idea is to approximate the advantage function as a quadratic function of the state and selected action. The action that would maximize the estimated action value function can be easily computed (thanks to the quadratic form). The neural network of the actor can then be trained to minimize the distance between its selected actions and the actions that maximize the estimate of the action-value function. The drawback of this method is that it relies on the approximation that the advantage function can be approximated by a quadratic function. In the case of the MOQ problem, this approximation does not hold, notably because of the discontinuity in the action space.

IV.1.5 Reinforcement learning and discrete-continuous action spaces

A discrete-continuous action space consists of a set of 'high level' discrete actions, each of which has its own subspace of continuous action. As a simple example, one can consider a robot, that can either rotate or move forward. When moving forward, the robot can adjust its speed in a continuous range. Similarly, it can adjust its angular velocity in a continuous range when rotating. In this example, the action space is both discrete and continuous.

Formally, we define a discrete continuous action space $\mathcal{A}$ as a set of N different discrete actions, each of which has an associated continuous action space $\mathcal{A}^k$:

$$\mathcal{A} = \bigcup_{k \in [1..N]} \mathcal{A}^k$$

Most reinforcement learning architectures consider either discrete or continuous action space solely. However, some algorithms have been developed for discrete-continuous hybrid action space [108] [44] [74] [33]. One of the promising solution to handle these cases is the P-DQN networks, which is based on the same logic than the DDPG. Figure IV.4 schematizes the architecture of P-DQN networks.

The idea of P-DQN is to use an actor-critic architecture, with an actor network A that outputs one vector of continuous values for each possible discrete action. The critic C is then given as an input all these vectors, as well as the state representation x, and outputs as many Q-values as there are discrete actions. The discrete action k, associated to the continuous vector a^k, that gives the highest estimated Q-value is then selected. To sum up : the actor computes the best continuous

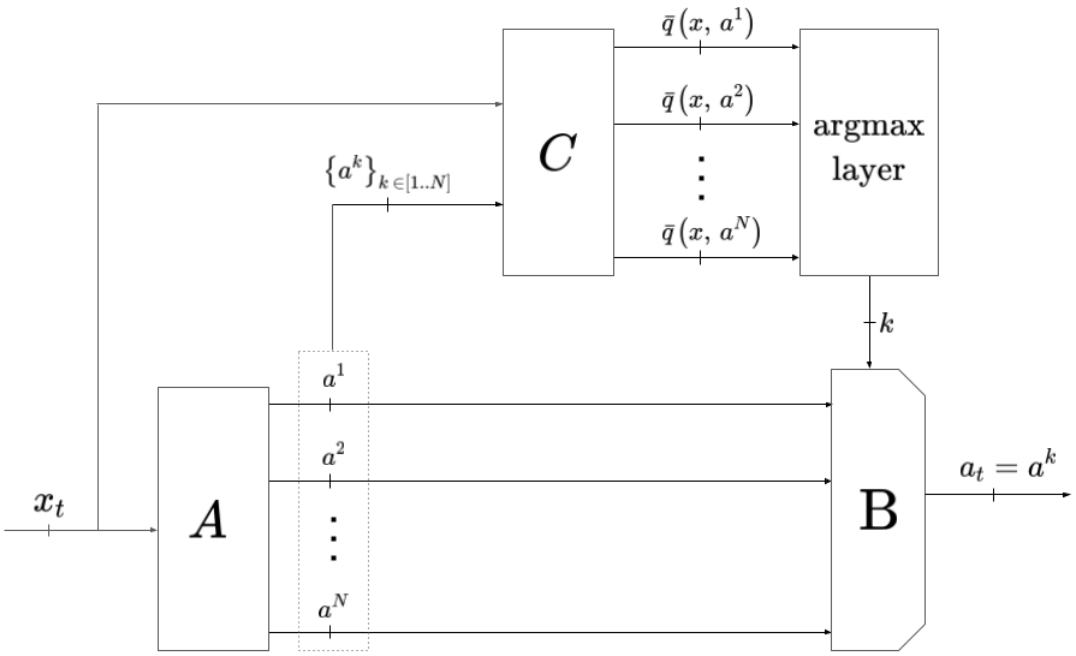


Figure IV.4: P-DQN architecture

action for every separate discrete case, and the critic then estimates which discrete-continuous action is the most likely to give the more reward.

The critic is trained by temporal difference learning. The actor is trained using the deep deterministic policy gradient algorithm [68]: the critic network being differentiable, it is possible to compute the gradient on a^k that would lead to a local increase of $\bar{q}(x, a^k)$. The gradient on the actor parameters is deduced by backward propagation. One can see the actor training when the discrete action k is selected as a minimization of the loss function $\mathcal{L}^A$ equal to the opposite of the action value function:

$$\mathcal{L}^A = -\bar{q}(x, a^k)$$

Applicability to the MOQ problem

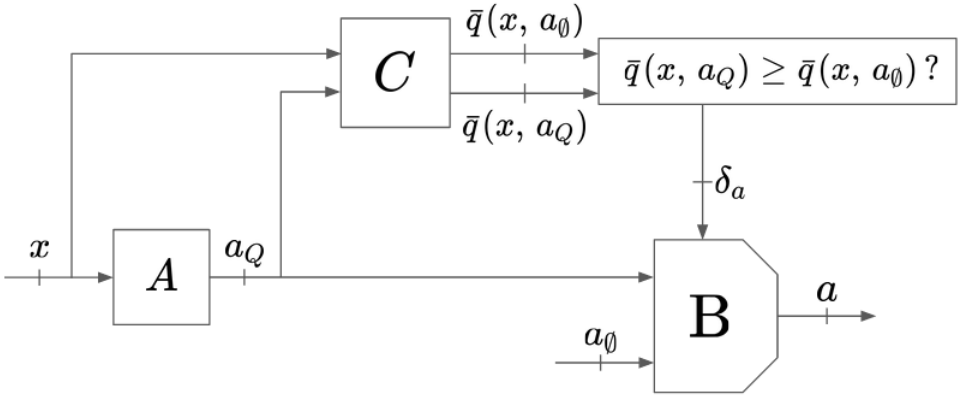


Figure IV.5: P-DQN architecture applied to the MOQ problem

For the MOQ problem, this would translate into the architecture displayed in figure IV.5. The discrete action space is divided into only two possibilities, whether an order is placed or not. When

no order is placed, there is no need to compute any quantity a vector. Thus the actor network A only computes the action that will be applied if the critic network C determines that the expected reward of ordering is larger than the expected reward of delaying the order.

As mentioned in section IV.1.3, the actor A outputs a continuous action that can violate the MOQ constraint. We must train the network to ensure that the action satisfies the MOQ constraint. On the contrary to section IV.1.3 however, the action $a_\emptyset$ does not need to be considered as a legal move: this action is considered separately, by the critic. Therefore one can simply penalize actions with order size smaller than Q, by adding a linear term to the loss. The loss for the gradient descent on the actor parameters can be re-written as :

$$\mathcal{L}^A = -\bar{q}(x, a_Q) + \kappa[Q - \sum_{i=1}^{I} a^i]^+ \tag{IV.3}$$

The parameter κ should be selected large enough so that an action whose size is smaller than Q is never the minimum of the loss function.

This type of architecture is notoriously hard to train, since the actor is trained to increase $\bar{q}(x, a_Q)$, which is not the true $q(x, a_Q)$ but an approximation, that is trained in parallel. In practice the inter-dependent training of two deep neural networks is a very unstable process [54].

We attempted to implement this architecture on simple instances of the MOQ problem. We were unsuccessful, mostly due to the extreme instability of the solution. As discussed in chapter II, in most MOQ problems the optimal order size is almost always equal to Q. The optimal order is therefore almost always located near the edges of the simplex of dimension I generated by the 'impossible' orders. The loss equation IV.3 leads to abrupt gradient changes around the frontier of this simplex. We conjecture that this leads to large gradient oscillations in and out of the 'impossible orders' simplex, that dilute the subtle relevant gradient changes along the simplex edges. We believe that this problem combined with the well known stability issues of actor critic architectures prevent such solutions from converging. We moreover believe that the computation of the best a_Q requires rather deep actor neural networks, which does not help with the stability issues.

For these reasons, we were unable to achieve satisfying results on the MOQ problem with P-DQN networks. The shortcomings of this method however led us to investigate more promising approaches: the major problem of this architecture appears to be the computation of a_Q, which is fortunately the w-policy strong suit.

IV.2 Hybrid policy

We developed an hybrid method in order to leverage the quick and robust order building mechanism of the w-policy, and to combine it with the powerful and robust deep Q-learning algorithms. We call this policy the hybrid w-policy.

IV.2.1 Core idea

A main limitation of the w-policy is its dependency to several key assumptions. The scope of applicability of these assumptions is detailed in section III.8. While it was shown that the π_w^+ allotment policy robustly provided optimal or near optimal order allotment in a wide variety of settings, the approximation $\Delta\dot{q}(x, a_Q)$ of the sign of $q_t^\pi(x, a_Q) - q_t^\pi(x, a_\emptyset)$ proved to be problematic under certain circumstances. This approximation is particularly critical, since it determines $\delta_a(x, a_Q)$ (whether the order is delayed or not). We exhibited that when the holding cost parameter h is of the same order of magnitude as m, this approximation would lead the w-policy to indefinitely delay the orders. The idea of the hybrid-policy is to improve the approximation of the sign of $q_t^\pi(x, a_Q) - q_t^\pi(x, a_\emptyset)$

by using a deep Q-network, while still leveraging the good performance of the π_w^+ allotment policy to avoid the limitations of actor critic architectures.

We divide the policy into two separate logic blocks. The first block, called π_w^+, computes the best non zero order from the state of the system and under the assumption that the inaction window is known and equal to w. It is the same as the π_w^+ block of the w-policy (see figure III.4). Its exact computation is detailed in section III.3. The input of this block is the state x of the system.

The second block is a function that determines whether the order should be placed or not. It does so by computing an estimate of the expected reward if we order ($\bar{q}_Q(x)$) and if we do not order ($\bar{q}_\emptyset(x)$). These two estimates are provided by a deep Q network that we call Q. Its input is the state of the system x. One can note that it does not have the order a_Q as an input, on the contrary to the DQN architecture. We define $\delta_a(x)$ as the boolean variable that corresponds to the statement 'we place an order'. Formally:

$$\delta_a(x) = \begin{cases} 0 & \text{if } \bar{q}_\emptyset(x) > \bar{q}_Q(x) \\ 1 & \text{if } \bar{q}_Q(x) \geq \bar{q}_\emptyset(x) \end{cases}$$

To simplify the notations, we define $\bar{q}(x, a)$ as:

$$\bar{q}(x, a) = \begin{cases} \bar{q}_\emptyset(x) & \text{if } a = a_\emptyset \\ \bar{q}_Q(x) & \text{if } a = \pi^+(x) \end{cases}$$

This hybrid architecture is schematized in figure IV.6. Such an approach is superior to the classical deep Q-network approach for large scale multi-item MOQ problems, since it is not necessary to output one $\bar{q}_a(x)$ value for every possible a (which is a problem when the action space is very large): the output layer size of the Q-network is equal to two. The advantage of using the hybrid policy compared to a classic actor-critic architecture is that the actor policy π_w^+ does not need to be trained, which greatly simplifies the training of the Q-network: the hybrid policy does not suffer from the instabilities associated with training two networks in parallel, which typically hinders the use of actor-critic architectures in practice. The bloc B then simply selects a_Q if $\delta_a = 1$, and $a_\emptyset$ otherwise.

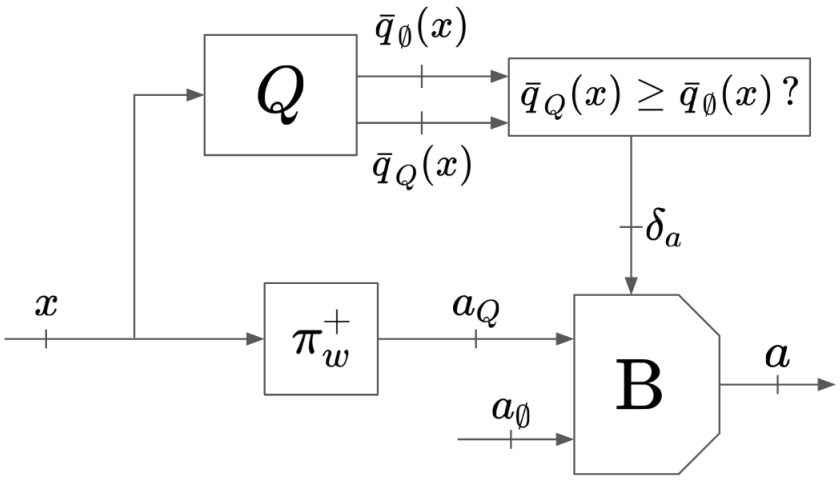


Figure IV.6: Hybrid policy architecture

The algorithm 3 recapitulates the steps of the computation the hybrid policy. It does not include the training algorithm of the Q network.

Algorithm 3: Hybrid policy

Data: x_t
Result: Order a_t

$a_Q = \pi_w^+(x_t)$;
$(\bar{q}_\emptyset, \bar{q}_Q) = Q(x_t)$;
if $\bar{q}_Q > \bar{q}_\emptyset$ **then**
  Return a_Q ;
else
  Return $a_\emptyset$;
end

IV.2.2 Loss function

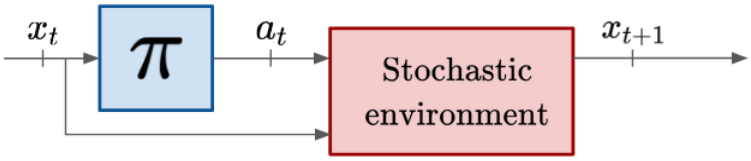


Figure IV.7: Simple reinforcement learning process

In classic reinforcement learning architectures, a policy π selects an action a from an input state x. The action is applied to the environment, which evolves in a stochastic manner to new state x', generating a reward $R(x, a, x')$. This process is shematized in figure IV.7. With our hybrid architecture, the computation of the selected action a_Q can be seen as an external process which is part of the environment, from the point of view of the policy that determines the binary decision δ_a. Figure IV.8 shows how the binary decision process of our hybrid policy can be isolated from the 'extended' stochastic environment.

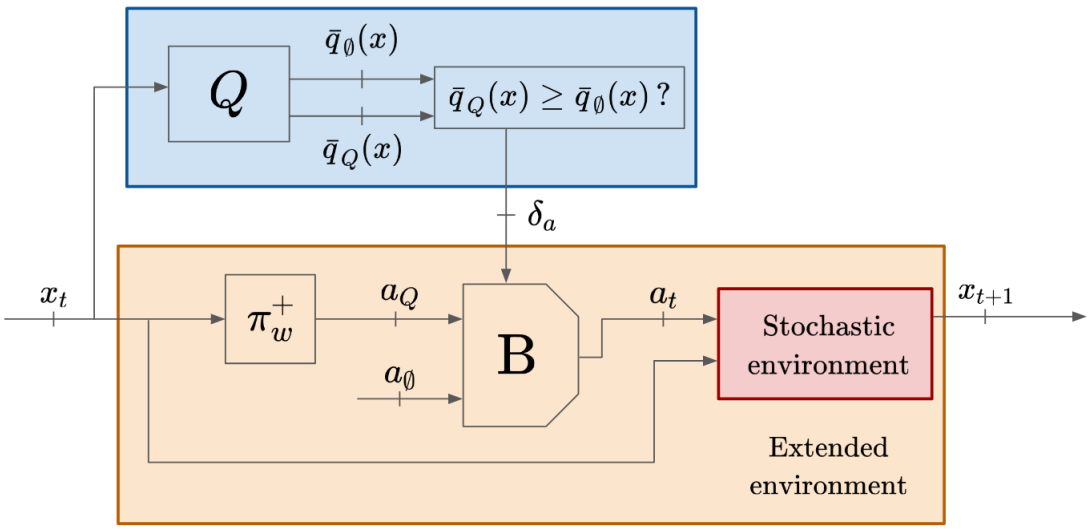


Figure IV.8: The hybrid policy as a reinforcement learning algorithm

The neural network is trained just like any deep Q-Network: we utilize the noisy estimate of the gradient provided by temporal difference algorithms. In this case, we utilize the off-policy Q-learning algorithm [95], which is a classic method when the DQN only estimates the action value function.

The state value function $v(x')$ is approximated by the maximum action value $\max_{a \in A_t} \bar{q}(x', a)$. The loss for the off policy Q learning (when applying a policy π) is written:

$$\mathbb{L} = \|\bar{q}(x, a) - R(x, a, x') - \gamma \bar{q}^{\pi}(x', \pi(x'))\|$$

In our case, the policy is the arg max of the action value estimate. The loss for the learning process is therefore written:

$$\mathbb{L} = \|\bar{q}(x, a) - R(x, a, x') - \gamma \max(\bar{q}(x', a_{\emptyset}), \bar{q}_Q(x', \pi^+(x'))\| \tag{IV.4}$$

This loss expression gives us the freedom to learn the network 'off-policy'. 'Off-policy' means that the network can be trained using examples generated following a policy different from π. This is due to the fact that our network approximates the action-value function for every possible action: we can still improve the estimate associated to an action that our policy would not have selected. This freedom is a significant advantage, since one should not use a deterministic policy to generate the training examples. A deterministic policy is unable to explore its environment, and may never explore the most valuable actions simply because they were under-valued at initialization. If we had to learn on-policy, we would have to artificially introduce randomness in the policy $\delta_a(x)$. Instead, we simply utilize a different (stochastic) policy, that will be detailed in section IV.3.

One issue with this loss formulation (equation IV.4) is that it leads to an inherently unstable learning behavior. If one uses a single network to compute both $\bar{q}(x, a_t)$ and $\bar{q}(x', a_{t+1})$, learning is unstable and tends to diverge due to the max term, which amplifies over-estimations of the action values. This effect is well studied and documented. To circumvent this issue, state of the art methods utilize a simple trick called Double Q-learning [99].

The idea is to define two networks that are initialized and trained differently, but aim to approximate the same action values. When one (that we call network A) is utilized to estimate the action values for the gradient estimate at time t, the other one (network B) is utilized to determine the action values at time $t + 1$. The output of network A also determines the action selected at $t + 1$. In practice, it changes the loss expression:

$$\mathbb{L} = \begin{cases} \|\bar{q}^A(x, a) - (R(x, a, x') + \gamma \bar{q}^B(x', a_{\emptyset})\| & \text{if } \bar{q}^A(x', a_{\emptyset}) > \bar{q}^A(x', \pi(x')) \\ \|\bar{q}^A(x, a) - (R(x, a, x') + \gamma \bar{q}^B(x', \pi^+(x'))\| & \text{if } \bar{q}^A(x', \pi^+(x')) \geq \bar{q}^A(x', a_{\emptyset}) \end{cases} \tag{IV.5}$$

The loss expression to train network B is the same as equation IV.5 with A and B swapped. This technique decorrelates the selection of the action a_{t+1} and its corresponding action value estimate. For example, if network A overestimates the action value of one action, the corresponding action is not necessarily selected by network B. If network B overestimates the action value of one action, the corresponding action is selected, but the action value estimated by the network A is not necessarily overestimated. It therefore avoids an over estimation of the state value associated to x'.

A simpler way to implement an idea similar to double-Q learning is to use a target network [72]. A target network is a copy of the 'main' network, that is updated differently. It is not updated through gradient descent. Instead, it is set to the main network every k_{target} training steps. The target network is then utilized for the q^B values in equation IV.5. Target networks are more widely used in practice than double Q learning, and this is the solution that we chose for our architecture.

IV.3 Training examples generation

Neural networks trained with reinforcement learning require large training sets. The minimum required dataset size to train the policy obviously depends on the characteristics of every specific problems. The MOQ problem is rather simple given that we reduced the complexity of the decision

to the minimum with our hybrid architecture that delegates the order allotment decision to a deterministic algorithm. We detail the number of examples we used for each experiment in section IV.4.5.

IV.3.1 Complexity of example generation

To train the neural network, one needs to provide training examples. Given the formula of the loss (equation IV.5), one training example should include the state x, the action a selected while in state x, the new state x', and the associated reward $R(x, a, x')$.

Since the a_Q is considered to be part of the stochastic environment from the binary actor point of view, we need to compute the potential action $\pi^+(x)$ unless the action a is equal to $a_\emptyset$. One can note that since we are using the off-policy Q-learning algorithm, we do not need to also evaluate $\pi^+(x')$. The computation complexity of $\pi^+(x)$ is detailed in section III.5. The computation time is not an issue in practice when it is computed only once (as it is the case for the w-policy): for the second La Redoute dataset, it took one minute on average. However one would need to iterate this computation at least thousands of times to generate an entire training set. We therefore need to find a way to accelerate the computation.

The costliest operation when computing $\pi^+(x)$ is the $\delta\tilde{q}^i(x^i, k, w)$ computation (for every $k \in [1..a^i_{max}]$). Instead of re-computing the $\delta\tilde{q}^i(x, k, w)$ for every computation of the allotment policy $\pi^+(x)$, one can pre-compute once the $\tilde{q}^i$ values for every timestep, for every possible state, and keep it in memory. This computation is done only once, and one can then utilize these stored values to compute quickly $\pi^+(x)$ for every state x that will be encountered in the training set generation. Only the partial sort of the Q best incremental orders remains to be done for computing $\pi^+(x)$. One does not need to consider several w values $\delta\tilde{q}^i(x^i, k, w)$ for a given t, since the w value depends only on the time t, and does not depend on the state of the system if one uses the equation III.13 (presented in chapter III) to evaluate it. Pre-computing of all the necessary $\delta\tilde{q}^i(x^i, k, w)$ values still remains very expensive in the general case. However, if $l = 0$, one can make use of an interesting property of the $\delta\tilde{q}^i$:

$$\delta\tilde{q}^i(0, k_1 + k_2, w) = \delta\tilde{q}^i(k_1, k_2, w)$$

This result can be deduced from equation III.11 (in chapter III), since $s^i_{t+l} = s^i_t = x^i_t$. Thanks to this property one only needs to pre-compute $\sum_{i=1}^{I}(s^i_{max} + 1)$ different $\delta\tilde{q}$ values (one for each $k \in [0..s^i_{max}]$, for each item) as if the stock available s^i_t was always equal to zero. For all the other scenarios, we have $\delta\tilde{q}^i(s^i_t, k, w) = \delta\tilde{q}^i(0, s^i_t + k, w)$, and there is therefore no need to store these values. As a concrete example, for the second La Redoute dataset (with a temporal horizon of 52 steps), the size of the memory required to store the pre-computed values in double-precision floating-point format is approximately 0.5 Go.

IV.3.2 Behavior policy

As said before in section IV.1.1, the loss evaluation presented in section IV.2.2 can be done off-policy. It means that an example generated following a different policy (also called behavior policy) can be utilized to evaluate the loss and train the Q-network. This Q-network is then utilized to determine the best greedy policy $\delta_a(x)$ (also called target policy).

The behavior policy is critical in the success of the training of the target policy. One can not generate an infinite amount of training examples due to the example generation cost. Having at least one example for every possible input state is also unreasonable due to the size of the state space. Firstly, the behavior policy should explore the possible states and actions, so that the target policy has been trained on diverse enough examples and does not miss potentially very profitable sequence

of actions. On the other hand, the behavior policy should also generate enough examples that utilize the knowledge acquired from the exploration, so that the full potential of promising sequence of actions can be fully exploited. Finding the right balance between these two contradicting objectives is known as the exploration vs exploitation trade off. The necessity for exploitation prevents us from generating the entire training set offline in advance, following a completely random policy, since we need to exploit the improvements of the partially trained target policy.

A simple yet widely used method to build a stochastic behavior policy from a deterministic target policy is to use an ε-greedy exploration method. The idea is that at every time step, the target policy is applied with a probability $1 - \varepsilon$. Alternatively, with a probability ε, an action is selected at random.

In the case of the hybrid policy, the action is either to order the quantity Q or to wait. When Q is larger than the average demand over two periods, the optimal policy on average recommends more frequently to not place the order rather than placing it. When an action is selected at random, the probability to order is equal to the probability of not ordering.

If ε is too large, the frequency of randomly placed orders may exceed the optimal reorder frequency: the stock levels never have the time to decrease to reasonable levels before a new order is randomly placed, even if the target policy always recommends to delay new orders. This leads to increasingly larger stocks, that eventually diverge toward infinity. We illustrate this problem with an example: we set Q to be approximately equal to the average demand over ten periods. When following the optimal policy the retailer would place an order every ten periods on average. If $\varepsilon = 0.5$, the behavior policy has a 25% chance of suggesting to place an order if the target policy recommends not to place an order. Therefore, even if the target policy systematically suggests to delay new orders, the behavior policy will on average place an order every four periods, and the demand will not be able to consume the stock as fast as it arrives.

Thus the choice of ε should depend on the average expected demand and the order size Q. We estimate an upper bound of ε for the MOQ problem. If we assume that the target policy never chooses to place an order, with a behavior policy being the ε-greedy version of the target policy, the probability to place an order is equal to $\frac{\varepsilon}{2}$. On average, $\frac{\varepsilon\, t_{end}}{2}$ orders of size Q are placed during an episode. We note D_{total} the sum of the average forecast over the entire episode and every items, $D_{total} = \sum_{i=1}^{I} \sum_{t=1}^{t_{end}} \mathbb{E}[f_t^i]$. We do not want the average total ordered quantity (equal to $\frac{\varepsilon\, t_{end}\, Q}{2}$) to exceed D_{total} if the target policy is to never order. Therefore, one can write the following bound for ε:

$$\varepsilon < \frac{2\, D_{total}}{t_{end}\, Q}$$

For our architecture, we set the behavior policy to be the epsilon greedy variation of the target policy. The selected ε values for each experiment is detailed in section IV.4.5.

IV.3.3 Example partial pre-computation

Despite the pre-computation of the $\delta\tilde{q}$ values presented in the section IV.3.1, the example generation remains costly compared to the learning itself. In order to further reduce the example generation cost, we suggest to use a second trick. From a state and an action, one can randomly draw several outcomes of the stochastic process instead of single one from the forecasts. We note k_S the number of generated outcomes. A single π^+ computation (which is the costly step of an example generation) is therefore sufficient to generate k_S examples.

The example generation process is schematized in figure IV.9, for $k_S = 3$. The w_t and $\delta\tilde{q}_t^i(0, k, w_t)$ are first pre-computed $\forall i \in [1..I], \forall t \in [0..t_{end}], \forall k \in [0..x_{max}^i]$. These values are saved and stored. Then during training, whenever more training examples are required, an episode is played following π^ϵ, that utilizes the pre-computed $\delta\tilde{q}_t^i$ values to compute a_t. Then we draw k_S samples of the

outcome of the stochastic process and compute the associated x_{t+1}^k. Combining x_t, a_t and the k_S x_{t+1} values, we have k_S new training examples. Finally we select one of the x_{t+1}^k to become the initial state of the next example generation process. Algorithm 4 is an example of such a process.

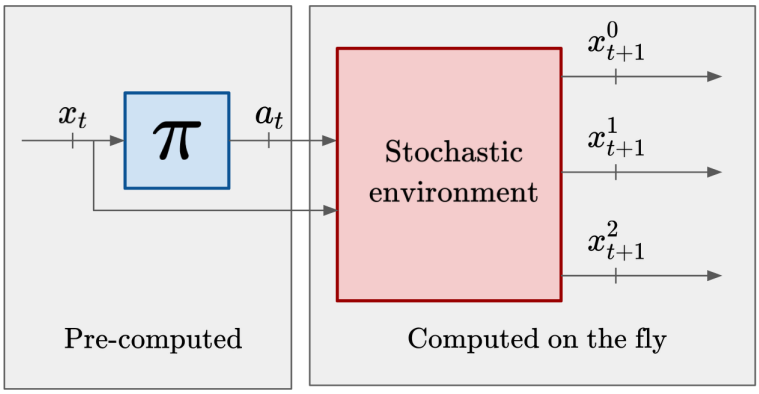


Figure IV.9: Training examples generation process of the hybrid policy

Algorithm 4: Training example generation

Data: $\left\{(s_0^i, \{o_{t'}^i\}_{\forall t' \in [0..l-1]}, \{f_{t'}^i\}_{\forall t' \in [0..t_{end}]})\right\}_{\forall i \in [1..I]}$

Result: $\left\{\{(x_t, a_t, R_t^k, x_{t+1}^k)\}_{\forall k \in [0..k_S-1]}\right\}_{\forall t \in [0..t_{end}-1]}$

forall $t \in [0..t_{end} - 1]$ **do**
  $a_t = \pi(x_t)$;
  forall $k \in [0..k_S - 1]$ **do**
    forall $i \in [1..I]$ **do**
      d_t^i = Randomly drawn sample from f_t^i;
      $o_{t+l}^i = a_t^i$;
      $s_{t+1}^i = [s_t^i + o_t^i - d_t^i]^+$;
      $R_t^i = m^i \min(d_t^i, s_t^i + o_t^i) - h^i[s_{t+1}^i - d_t^i]^+$;
    end
    $R_t^k = \sum_{i=1}^I R_t^i$;
    $x_{t+1}^k = \{(s_{t+1}^i, \{o_{t'}^i\}_{\forall t' \in [t+1..t+l]}\}, \{f_{t'}^i\}_{\forall t' \in [t+1..t+h_f+1]})\}_{\forall i \in [1..I]}$;
  end
  $x_{t+1} = x_{t+1}^0$;
end
Return $\left\{\{(x_t, a_t, R_t^k, x_{t+1}^k)\}_{\forall k \in [0..k_S-1]}\right\}_{\forall t \in [0..t_{end}-1]}$;

The number of parameters in dense layers is extremely large, which means that neural networks are prone to over-fitting. A statistical model is over-fitting when it provides good predictions on its training set, but fails to extend such performance to additional data. It generally occurs in machine learning when the same training data are re-utilized too many times. Generating k_S examples from the same initial situation does not generate exactly the same training examples, but these examples are still more similar than completely randomly generated examples. The user must therefore be aware that increasing this parameter too much may lead to over-fitting.

IV.3.4 Experience replay

The usage of replay buffer is common to improve training stability of reinforcement learning models [69]. The idea of 'experience replay' is to store the training examples into a fixed sized replay

buffer. The training batches are ramdomly selected from this replay buffer. The replay buffer is constantly updated with new examples, and more ancient examples are removed with more or less randomness depending on the preferred algorithm. There are several advantages to such a buffer. Firstly, examples are utilized multiples times, which reduces the need to generate more experiences. Similarly to the k_S idea, the drawback is the possibility of over-fitting. Secondly, experience replay tends to stabilize learning. Indeed, new examples are likely to be highly correlated to each other: one of the requirement for the stochastic gradient descent optimization to work is that training data should be independent. The replay buffer alleviates this issue by providing a mix of new and old experiences as training examples.

IV.4 The hybdrid policy Q-network configurations

In this section, we detail and justify the implementation choices of the Q-networks we ended up using. We do not argue that these choices are optimal, but that they proved good enough for our use cases, and the experiments presented in section IV.5.

IV.4.1 Layer types

Deep-Q networks in the literature have varied architecture and technical implementation choices depending on their application. The simplest layers are the fully connected layers, also called dense layers. A dense layer is formally defined as:

$$Y = \phi(AX + b)$$

Y is the output vector, X is the input vector, A is the kernel, the matrix of the weights connecting each input and output neurons, b is the bias vector, and ϕ is the activation function. One of the main scalability issue of fully connected layers is the number of parameters required when the number of neurons of the input and the output of the layer is large. The number of weights of a dense layer is roughly equal to the size of its inputs multiplied by the size of its outputs. As an example, for the second La Redoute dataset, there are 11607 items. If one utilizes the stock level and the average forecast over the next ten weeks for each items as inputs, it amounts to 127677 neurons for the input of the first layer. If we set the first layer output size to be the same as the input size, we still get a total of 16 billion parameters. If these parameters are 32 bits floats, the storage necessary for the parameters of one layer is around 64 Go of memory.

The image processing community faced this problem, with input images being very heavy. The solution to reduce the number of parameters in these cases is to utilize convolutional neural networks [65]. Convolutional neural networks are built around convolutional layers. The idea of convolutional layers is to apply the same small scale dense layer to many limited divisions of the inputs, in order to exploit the recurring patterns of the data along certain input data dimensions. Convolutional neural networks are for example very effective when applied to pattern detection in 2D-images. In this case, one can define a small scale dense layer with an input size equal to k^2, which correspond to a $k \times k$ patch in the image. This layer is then applied to every $k \times k$ patch in the image, and can be trained to detect patterns that fit in this patch (sometimes called receptive field). One of the main advantage of convolutional layer is that they have far fewer parameters than dense layers. The number of parameters is equal to the size of the receptive field. The limited number of parameters allows convolutional neural network to go both deep (number of layers) and wide (size of the hidden layers). The dimensionality of the data is then reduced by using pooling layers. The idea of these layers is to combine the output of several neurons into a single output neuron. Max-pooling and average-pooling are popular pooling methods: the output neuron is simply the maximum or the average of the several input neurons. Once the dimensionality of the data is reduced enough, one

can apply dense layers without having nearly as much parameters as one would have if a dense layer was applied directly on the inputs of the network.

The idea of convolutional layers is however not helpful to reduce the number of parameters in the case of the MOQ problem. Indeed the input neurons for the MOQ problem represent data from independent items. On the contrary to images for example, where the input pixels have a natural spatial correlations with their neighbours, there is no natural structure between independent items. One could therefore only apply a convolutional layer that would have a receptive field corresponding to a single item related inputs (stock available, stock on order, forecast). It would however make no sense to connect certain items with a selection of items but not the others. In that sense, our input data does not have a natural hierarchy like an image can have (with larger and larger neighborhood), and we have no other choice than to use large dense layers.

Another popular type of architecture are recurrent neural networks (RNN). RNN are a class of artificial neural networks that are built to be applied at every step of a sequence, while maintaining an internal state (memory). The main types of RNN are Long Short Term Memory (LSTM) networks [49], Gated Recurent Units (GRU) [20], and transformers [100]. These architectures are valuable when data can be lost when moving forward in the sequence. For example, RNN have proven very useful for text analysis and natural language processing [100]. When analyzing a sentence, the network is applied sequentially to every word. The input of the RNN is a specific word of the sentence. In order to analyze it in the context of the previously observed words, the RNN leverages its internal memory, where the relevant information about the beginning of the sentence is encoded. This type of approach is not relevant for the MOQ problem. Indeed, there is no need for an embedded memory for the network since the state already encapsulates all the information necessary to take a decision. No relevant information is lost when moving forward in time: the actor only needs the data of the current state to compute the optimal action. The past evolution of the system has no impact on the decision that should be taken at a time t, only the current state matters. The Q-network of our hybrid policy is therefore built as a basic fully connected feed forward neural network.

IV.4.2 Activation function

The activation function purpose is to introduce a non-linearity, so that the network can approximate non linear functions. The performance of the different possible activation functions have been extensively studied in the machine learning literature [80]. The ReLu activation function [37] is generally considered the simplest and generally best performing activation function for deep neural networks, since they suffer less from the vanishing gradient problem.

$$\text{ReLu}(x) = \max(0, x)$$

The vanishing gradient problem still arises with Relu activation functions in the form of 'dying' neurons [2]. Since the output for negative values is always equal to zero, the slope is also equal to zero on negative values, and no gradient is back-propagated. This is not an issue as long as the output value is not equal to zero for every training examples. If it is the case, the weights can never be updated to relevant values, and neuron will remain 'dead'. A neuron may end up in this state if at some point the gradient 'pushes' the weights too far to contribute negatively. It commonly occurs during the initial phases of training, when the loss and associated gradients are large. There are several ways to mitigate this issue. One can detect the neurons that never activate over a large number of training steps, and reinitialize their weights during the training process. It is also common to utilize a modified version of the ReLu activation function, called the leaky Relu [109]:

$$\phi(x) = \max(\varepsilon\, x, x), \varepsilon \in]0, 1[$$

IV.4.3 Deep learning tricks

The depth of the network is its number of layers. The width of a layer is its number of neurons. In theory according to the universal approximation theorem, a neural network with a single large enough hidden layer is sufficient to approximate any function. In practice however, the layer may be infeasibly large and impossible to train correctly. It is more practical to increase the depth of the network to approximate complex behavior than it is to infinitely increase the width a very limited number of layers. A network that uses multiple hidden layers is considered deep. A few issues arise from the depth increase of a network. The first one is the vanishing gradient problem [48]. This problem arises from the equations utilized to propagate the gradient backward. Deriving the equations with the chain rule, one can deduce that on average the gradient absolute values decrease the further the layer is from the output layer. The usage of ReLu activation functions and batch normalization [53] are known to mitigate this issue. Batch normalization is a technique to set the means and variances of the inputs of layers to desired normalized values. The normalization is done over the batch size. The theoretical reasons behind the batch normalization benefits are still unclear, even if these benefits are largely recognized [85]. A side effect of batch normalization is that it tends to amplify the exploding gradient problem, especially during the first training iterations. The exploding gradient occurs when the average gradients are multiplied layers after layers by values greater than one, resulting in exponentially larger weight updates. Solutions to the exploding gradient include gradient clipping [111] and weight regularization [64]. The idea of gradient clipping is to directly cap the gradient values to avoid large weights variations. Weight regularization applies a loss penalty the larger the weight values are, and therefore indirectly reduces large weights variations. For very deep networks however, these techniques are not sufficient. Very deep architecture have been obtained in practice by leveraging residual neural networks (ResNet) [45]. The idea behind Resnets is to introduce skip connections between the layers, that can be seen as shortcuts towards the output of the network. The relative 'distance' of any layer to the output layer is therefore drastically reduced.

Given the limited depth of the networks we used for the hybrid policy, the usage of gradient clipping, weight regularization and batch normalization proved sufficient in practice. The training of a neural network can be impacted by the weight initialization choice [93]. For our networks, we chose the state of the art glorot uniform initialization method. It is to be noted that switching from a glorot uniform to a truncated normal initialization did not significantly impact the performance of our networks in practice.

IV.4.4 Learning rate

The learning rate computation is critical for a successful training. We utilized the state of the art ADAM update rule to adapt the learning rate on the fly [62]. ADAM is a standard update rule for deep learning training. The hyper-parameter of ADAM that proved critical for our network training on very large scale instances of the problem was the initial learning rate l_{init}. Indeed, when the initial learning rate l_{init} was set to standard values such as 0.01, the loss increased to very large values after a single learning step. We identified that the issue was coming from the large discrepancy between the dense layers input and out sizes. The state vector size for the second dataset is equal to 127677 when considering a forecasting horizon of ten weeks, and the output size is equal to just two. When a gradient, even very small, is applied to every parameter of the network to slightly increase or decrease one of the output value, the cumulative effect can be massive and throw the output values to massive values. This makes the initial phase of the learning very unstable, until the ADAM updates decreases the learning rate enough that this phenomenon stops. When it does, the weights and the bias values may have already been set to large values, and many neurons may be 'dead' (see section IV.4.2), which makes training harder. This issue disappears if

the initial learning rate is set to low enough values. The values chosen for the different experiments are detailed in section IV.4.5.

IV.4.5 Configurations for practical experiments

In this subsection, we detail the configuration and hyper-parameters values of the hybrid policy used in practice in the experiments detailed in section IV.5. The experiments are divided into three groups, across three different scales: the two-items version of the problem, the first La Redoute dataset (134 items), and the second La Redoute dataset (11 607 items). The table IV.1 shows the configuration that we ended up using for the three different scales. These hyper parameters were selected by trial and error: determining good hyper-parameters is especially time-consuming on the large scale instances of the problem.

We used a fixed size for all the hidden layers of the deep Q-networks, and we called this size the 'width' of the network. The depth is the number of dense layers. One can see that in these instances, the depth required to achieve the near optimal performance is rather small. Increasing the depth did not improve performance, and usually simply made training more unstable.

Dataset	2-items	Dataset 1	Dataset 2
Depth	3	3	3
Width	128	256	3e4
ε	0.15	0.1	0.1
γ	0.95	0.95	0.95
k_S	1	8	8
k_{target}	128	128	128
l_{init}	1e-3	1e-3	1e-5
Batch size	32	32	32
Number of examples	500 000	200 000	100 000
Replay buffer size	2048	2048	2048

Table IV.1: Hybrid policy configuration for the section IV.5 experiments

IV.5 Results

In this section, we present several experiments to assess the scalability and the performance of the hybrid policy on the variable demand multi-item MOQ problem. We first consider small scale toy experiments, before applying the hybrid policy to two large scale real world datasets.

IV.5.1 Small scale experiments

Similarly to section III.8, we consider two items instances of the MOQ problem, so that the optimal policy can be computed and act as a baseline. The goal is to validate the ability of the hybrid policy to solve the MOQ problem, while also exhibiting its limitations.

Training stability

For this experiment, our goal is to assess the stability of the training process. We use the same settings as the experiment detailed in section section III.8.3. We assume the demand to be stationary. We set Q to 25. The initial pre-reception stocks have been set to zero for the two items. We set the economic parameters to $m^1 = m^2 = 1$, $h^1 = h^2 = 0.1$. The demand probability distribution at

every time-step for each item is a Poisson distribution with a parameter $\lambda = 5$. The leadtime l is assumed negligible.

We consider ten instances of the training process. At each instance, the neural network weights are initialized with a different seed, and training examples are also generated with a different seed. At regular intervals during training, the performance of the hybrid policy are evaluated on the same testing examples. This testing set is built with 10 episodes of 52 timesteps.

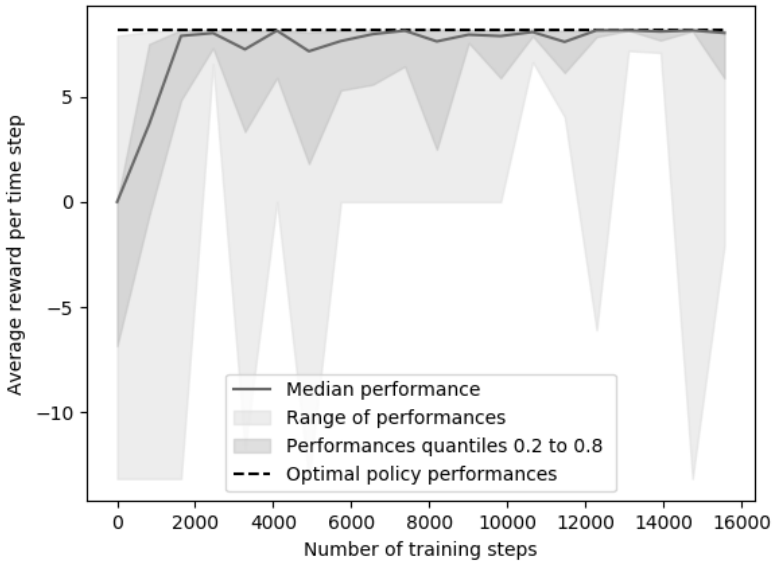


Figure IV.10: Hybrid policy performance across learning phase

Figure IV.10 displays the evolution of the best and worse performing training instances, as well as the second best and worst, and the median performance. The first conclusion one can draw is that this architecture is capable of reaching near-optimal performance in these settings. The optimal policy was obtained using the backward recursion algorithm. The majority of the trained policies converge towards satisfactory behavior. However even with the same parameters (see table IV.1), training instances can end up having widely different performance. Instabilities are still present even after 16 000 training steps, while near-optimal performance is achieved after only 3 000 training steps for at least half the training instances. In an industrial environment, this instability should be accounted for and safeguards should be put in place. One could for example launch several training instances and only keep the best performing ones.

Reorder zone

We then look into the obtained reorder policy in the same settings as in the previous sub-section. We selected the best performing instance of the previous experiment, and drew its reorder zone in figure IV.11. One can compare this figure to figure III.8 in chapter III, which displayed the reorder zone of the w-policy. The reorder zone of the hybrid policy does not perfectly match the optimal policy reorder zone. The hybrid policy reorder zone is not even symmetrical with regard to the items, while this instance of the problem is symmetrical. The performance of the hybrid policy are nearly optimal (see figure IV.10), yet its shortcomings are obvious when looking at this figure. This apparent paradox can be explained by looking at the probability of occurring of every state.

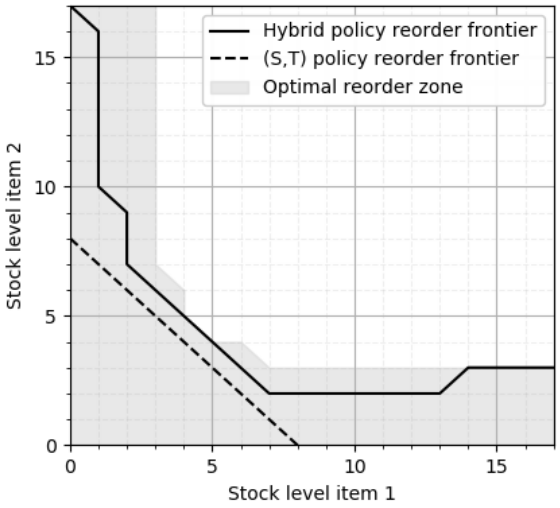


Figure IV.11: Reorder zones in a two items scenario

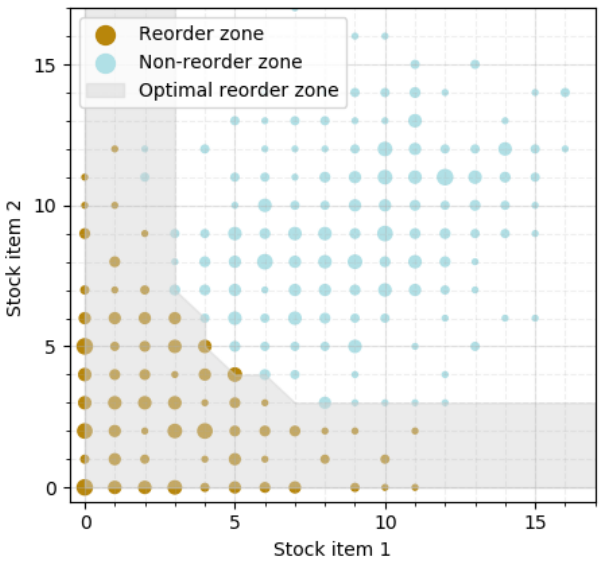


Figure IV.12: Hit map and reorder zone of the hybrid policy on the testing set

The figure IV.12 displays the number of times each state is visited when applying the hybrid policy to the testing set (the same testing set as the previous section). The area of each point is proportional to the number of times each state is visited. The initial states (equal to $s^1 = s^2 = 0$) are excluded to avoid having an artificially much larger circle in $(0, 0)$. One can see from this figure that the hybrid policy almost always takes the same binary decision δ_a as the optimal policy. The states where the behavior are different are rarely visited in practice. We conjecture that the hybrid policy behaves sub-optimally in these low probability states because it did not face enough examples of similar situations during training. However the difference in expected reward between delaying the order or not is not significant enough to matter, since these states are rarely encountered. This explains the near-optimal performance of the hybrid policy in terms of reward, even if the policy behavior appears obviously sub-optimal when drawing the reorder zone.

Varying h

One of the objective of the hybrid architecture was to alleviate the problem of the binary decision δ_a of the w-policy being conservative when facing potentially large holding costs. We therefore reproduced the experiment presented in section III.8.4: the settings of the experiment are the same as the previous one, but with increasingly large values of h^1 and h^2 (we set $h^1 = h^2 = h$). The performance of the optimal policy, the w-policy, and the hybrid policy are displayed in figure IV.13. The hybrid policy manages to alleviate the problem associated with the w-policy: when the holding cost parameters values exceed 0.8, the w-policy stops placing new orders entirely. On the contrary, the hybrid policy is able to recognize that short term losses induced by large holding costs can be compensated by later sales. Its performance remain competitive with the optimal policy. This experiment demonstrates the ability of the hybrid policy to cover the main limitation of the w-policy.

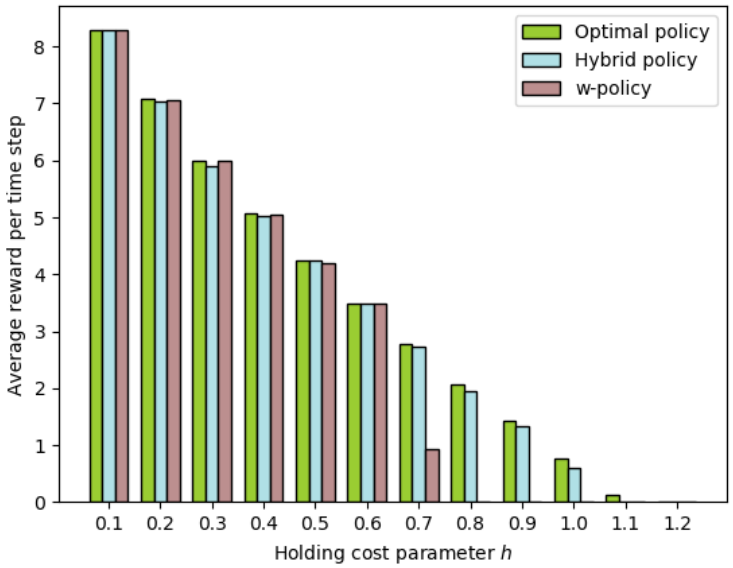


Figure IV.13: Performance of the different policies for increasing h values

IV.5.2 Large scale experiments

Scaling to larger MOQ problem instances is much more challenging for the hybrid policy than it is for the w-policy. One must find the right architecture and hyper parameters of the network, and generate training data efficiently enough so that training times do not get too long. As a reminder, we consider that the order of magnitude of the training time should not exceed a couple of hours for practical usage in real supply chains.

The goal of this subsection is therefore to assess the ability of the hybrid policy to scale to the very large instances of the MOQ problem, and to compare its performance to the w-policy. The hybrid policy relies on different assumptions than the w-policy. Comparing their performance is therefore both important to determine which solution is better suited for practical usage in the industry, and to assess the impact on the performance of these different assumptions.

Dataset 1

We first consider the results obtained on the first La Redoute dataset detailed in section III.9.1. As a reminder, there are 134 items in this dataset, and the demand of each item is assumed stationary

and with a Poisson probability distribution. The performance of the hybrid policy is presented in figure IV.14. We consider increasing values of Q. For this experiment, we ran a hundred episodes of 52 weeks per data point, with different random seeds. One can see that the hybrid policy is competitive with the w-policy. This result shows that the assumptions made to approximate the sign of $q_t^\pi(x, a_Q) - q_t^\pi(x, a_\emptyset)$ by $\Delta\dot{q}$ (in the w-policy) are valid when the holding cost parameters are low enough. Indeed, even using a Q-network to compute δ_a without being restricted by the same assumptions, the hybrid policy is not able to perform better than the w-policy. The average training time on this problem was around 40 minutes. The decision time once the model is trained is negligible.

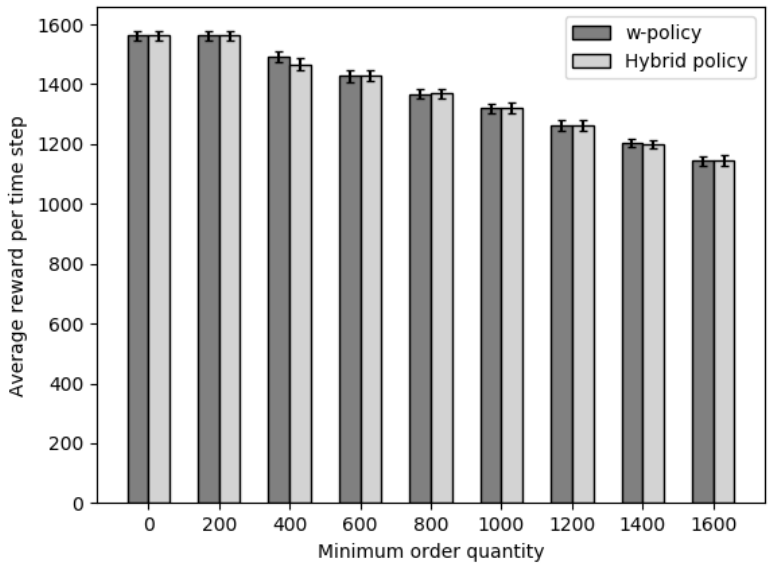


Figure IV.14: w-policy and hybrid policy performance for increasingly large Q on dataset 1

Dataset 2

We finally consider the second (and larger) La Redoute dataset presented in section III.9.2. The pre-computation of the delta values took on average one hour. The training on a specific instance of the MOQ problem took on average three hours, for a total of four hours, which is at the limit of what is practical for actual supply chains. It is to be noted however that finding the right parameters for the network and training settings was in practice no small task. It was very time consuming. The presented solution could therefore maybe be improved by fine tuning all the hyper parameters. The performance of the hybrid and the w-policy are presented in figure IV.15, for several values of Q. Again, the two policies perform similarly, up until MOQ values above 60 000, where the w-policy outperforms the hybrid policy. This experiment demonstrates the ability of the hybrid policy to scale even to very large instances of the MOQ problem, but also showcases one of its limitation: while it should in theory be able to perform at least as well as the w-policy, we were unable to find a parametrization that performs well across the whole range of MOQ values. Despite relying on further simplifying assumptions, the w-policy demonstrates again its good performance in practice.

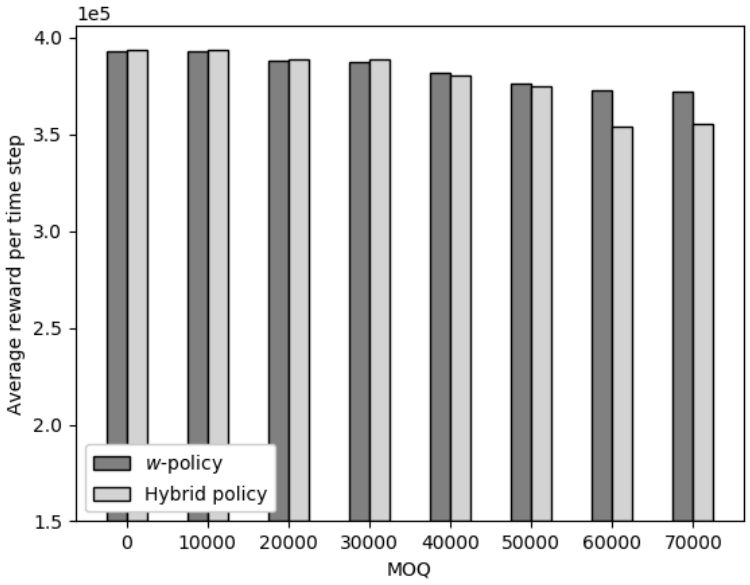


Figure IV.15: w-policy and hybrid policy performance for increasingly large Q on dataset 2

Conclusion

Our analysis of the current reinforcement learning literature leads us to conclude that many state of the art reinforcement learning architectures are inapplicable to the MOQ problem. Only the architectures capable of dealing with discrete-continuous action spaces can be considered (see section IV.1.5). We attempted to implement these solutions without success: these solutions appear to be too complex and unstable to be realistically implemented in an actual supply chain. We analyze the reasons behind these architectures instabilities, which led us to develop an hybrid policy that combines the allotment policy of the w-policy and a deep Q-network to determine whether an order should be placed or not. The usage of the w-policy for the allotment decision significantly improves the stability of the architecture. We detailed how to implement this hybrid policy in practice, and provided guidelines to help scale up to very large instances of the MOQ problem.

The hybrid policy has proven capable of avoiding certain limitations of the w-policy, notably when the holding costs are large compared to the margins. The hybrid policy was however unable to outperform the w-policy on practical use cases: this result tends to indicate that the w-policy is performing sufficiently well for practical usage. In such situations, the w-policy also appears to be significantly superior in terms of explainability, robustness, adaptability, computation time, and ease of implementation. Therefore, unless the system is outside the scope of applicability of the w-policy, we recommend to use the w-policy to solve large scale multi-item MOQ problems.

Reinforcement learning approaches still remain very promising for complex inventory control problems: while the w-policy appears superior to solve practical MOQ problems, it is was crafted specifically for it. The adaptability of reinforcement learning solutions to newly introduced constraints or variables is a big advantage over hand-crafted policies. For example, one could consider a more complex version of the problem, where the economic parameters m varies over time due to items being on sale. Suppliers sometimes also have set-up costs additionally to MOQ constraints, or have uncertain delivery times. Hopefully, the work presented in this chapter can be a step toward the development of solutions capable of adapting to such a variety of complex inventory control problems.

Conclusion

Summary

In the introduction, we presented the context of this PhD: this PhD was funded and initiated by the company Lokad, that provides supply chain optimization as a service. Many of its clients face what we called the multi-item Minimum Order Quantity inventory control problem. We detailed why this problem is both crucial and difficult to solve: the first reason is that it is a problem of repeated decision making under uncertainty. It would be sub-optimal for the retailer to plan in advance all its future decisions, since he is able to adapt his decisions depending on the outcome of the stochastic demand. The second reason for its complexity is that the minimum order quantity can be shared between multiple items, which forces retailers to coordinate the replenishment orders of these items. In chapter I, we provided a review of the inventory control literature. We concluded that the 'off the shelf' methods utilized for many inventory control problems were not adequate to solve the MOQ problem. Classical approaches relying on reorder point methods are unable to deal with such multi-item problems. Linear programming or stochastic linear programming are unable to deal with problems of decision making under uncertainty with multiple periods. Dynamic programming methods such as the backward recursion can deal with small scale multi-item MOQ problems. Their complexity is however tied to the size of the state and action spaces. Yet, these spaces grow exponentially with the number of items considered. Such methods are therefore inapplicable in practice. Reinforcement learning algorithms can be a valid alternative to dynamic programming approaches when the latter fail to scale on more complex problems. We discuss the usage of these algorithm in the inventory control literature. We show that while these approaches are promising, their utilization remains quite rare on complex inventory control problems. We conjecture that this might be due to the difficulty of utilizing the right architecture and implementing it in practice. We then studied the literature specifically dedicated to the MOQ problem. Several methods have been developed on simplified version of the multi-item MOQ problem. When only a single item is considered, reorder points methods have been developed [115]. An approximate solution, inspired from the single item version of the problem, was also designed to solve the multi-item version of the problem, but requires the demand to be stationary [114]. This simplifying assumption is inapplicable for most Lokad's clients. To the best of our knowledge, there were therefore no method capable of dealing with the multi-item MOQ problem with variable demand before our work.

In chapter II, we presented an analysis of different aspects of the MOQ problem. One can compute the optimal solution to the MOQ problem on its small scale instances using the backward recursion algorithm. We therefore were able to study the properties of the optimal policy. We looked into the shape of the state and action value functions associated to the optimal policy. We then assessed that the conditions under which the optimal order size was equal to the minimum order quantity Q. We studied the impact of decisions over time. We notably exhibited that a well designed policy could partly correct the negative impact of a sub-optimal decision with subsequent orders. We therefore introduced the concept of inaction window, which is the duration between an order and the subsequent one. This concept is key in the w-policy, presented in chapter III. Finally, we presented an efficient solution to solve the single period version of the MOQ problem. This solution

is based on the idea of atomizing the decisions into single unit increments of order, and compute their associated contribution to the reward. The best possible order is then determined directly by ranking the increment of orders contributions to the reward. This approach of quantifying the impact on the reward of order increments was re-utilized in the w-policy.

In chapter III, we presented one of our main contribution, the w-policy. The w-policy computation is divided in two steps. First, the retailer assumes that a new order should be placed, and computes the best possible one (the best allotment between the items). It then becomes easier to determine whether ordering is worth it or if it would be more profitable to delay the order, because one knows what would be the order if we were to place it. These two steps are made possible by computing approximations of the action value functions, that take advantage of some well chosen 'nearly true' assumptions. The analysis presented in chapter II helped justify these approximations. The key idea of the w-policy is that the dependency between the items can be almost completely described by the timing of the next replenishment order. The multi-period can therefore be simplified, and becomes very similar to the single period version of the problem. An expected reward can be associated to each order increment, and it becomes possible to prioritize adequately between the items when building the best potential order.

We validated the performance of the w-policy with extensive numerical experiments. We first considered small scale toy instances of the MOQ problem, where the optimal policy could be computed. On such instances, the w-policy demonstrated near optimal performance, and even proved more robust than the (S, T) policy (when a comparison was possible, since the (S, T) policy is only applicable when the demand is stationary). The only exception occurred when the settings of the experiment were chosen to be outside the scope of applicability of the assumptions upon which the w-policy is built. Notably, when the holding costs per period parameters h^i are large (when they reach the same order of magnitude as the m^i parameter), the w-policy may recommend to indefinitely delay new replenishment orders. We then applied the w-policy to large scale real datasets (obtained from a Lokad's client). It demonstrated the ability of the policy to scale up to 10 000 items (with a decision taking on average a minute to be computed), which was unprecedented.

In chapter IV we investigated whether reinforcement learning provide solutions to the multi-item MOQ problem. We showed that the most popular architectures in the field were not directly applicable to it due to the characteristics of its state and action space. The only applicable methods were the ones capable of dealing with discrete-continuous action spaces, such as P-DQN networks [108]. Such methods are based on actor critic architectures, which are notoriously difficult to train. We detailed our attempts at applying them to the MOQ problem without success, due to extreme instabilities in the training phase. We then presented an alternative solution to solve these instabilities, the hybrid policy. This policy utilizes the allotment mechanism of the w-policy to determine the best potential order, and a deep Q-network to determine whether this potential order should be placed or delayed. This solution removed the need for an actor-critic architecture, and proved to be much easier to train in practice. We detailed the choices we made in terms of network parametrization and architecture. The training example generation appeared to be a limiting factor for large scale instances of the MOQ problem, and we therefore suggested a way to generate them more efficiently.

We then tested this hybrid solution on small scale MOQ problem, similarly to the experiments in chapter III. The hybrid policy performance were on par with the w-policy, but its training occasionally failed to demonstrate a robustness satisfying enough for real world utilization. It still proved capable of dealing with the instances of the MOQ problem the w-policy struggled with (when the holding costs parameters h^i were increased). We also managed to make the hybrid solution scale up to the very large scale real datasets. We concluded that reinforcement learning has a promising potential to solve complex large scale inventory control problems under uncertainty, but remains difficult to implement in practice.

The w-policy was therefore preferred to be implemented and integrated into Lokad's solution.

It is utilized on a day to day basis to drive client's inventory control decisions. Its performance and scope of applicability proved largely sufficient in practice, and safeguards were implemented to detect the unlikely situations that lead to the w-policy behaving poorly.

Conclusion for supply chain practitioners

One of the goal of this PhD was to produce algorithms and solutions that could prove valuable in practice and be integrated in Lokad's software. Such solutions can unsurprisingly be difficult to publish given the constraints of academic research publication processes. Publishing in serious journals and conferences understandably requires well defined problems and rigorous benchmarks. On the other hand, the best solutions in practice are not necessarily the ones that can be proven theoretically superior under specific settings. Instead, simple, scalable and adaptable solutions will often be preferred by supply chain practitioners. This discrepancy between actual supply chain practices and academic research on inventory control is well documented [1, 47].

I believe that the conclusion supply chain practitioners should draw from this thesis is not only that if they happen to face a multi-item MOQ problem, they can implement and use the w-policy. Instead, they should consider how and why the w-policy was built this way: by atomizing order into independent order increment, and by assigning each of them an expected reward, the optimization process was greatly simplified. This simple idea can be utilized to prioritize purchase decisions for many inventory control problems subjected to constraints that are shared by multiple items. A generalized version of this action value function approximation via atomization was implemented in Envision (Lokad's programmation language for supply chain optimization) under the name of 'action reward'. The idea of the inaction window is translated into the simpler idea of 'responsibility window': when ordering, one should estimate the time window over which the current decision is responsible in case of stock out. The order increment associated to an ordered unit is likely to have a positive impact if this unit has a high probability of being sold during this responsibility window. If it is sold after the responsibility window, a subsequent order could have covered the demand, and it was not necessary to order it immediately. The holding costs associated to an order increment can however extend far beyond the responsibility window, up until the associated unit is finally sold. More information on Envision and the 'action reward' function can be found on Lokad's website and YouTube channel.

We consider a simple application example of the 'action reward', with a retailer subjected to a budget constraint when reordering. Being able to evaluate an approximate expected reward for every micro-decision is very helpful in this situation. All the micro-decisions can be ranked by decreasing return on investment (per increment reward divided by per increment ordering cost). This creates a prioritized purchase list, that indicates the retailer how to allocate its budget. This prioritized purchase list has the advantage of being completely explainable: if the suggested decisions appear sub-optimal, it is very easy to investigate how the prioritized purchase list was built, and correct potential anomalies in the input data. If the retailer also simultaneously faces additional constraints, such as minimal service levels or fillrate, or even order set-up costs, these constraints can easily be taken into account in the purchase list prioritization criterion, or in the order selection process from the prioritized purchase list. The inaccuracies generated by the assumptions necessary to build and use the 'action reward' in combination with a prioritized purchase list in these problems are likely to be negligible compared to the many sources of inaccuracies of a real supply chain (holding costs models, forecasting methods, improper stock tracking, etc). The main advantage of such an approach is that it provides an adaptable, scalable, and explainable framework for inventory control.

Perspectives

I believe that the work presented in this thesis naturally leads towards many promising questions and potential solutions to complex problems. Firstly, the approximations utilized in the w-policy could be seen as not very sophisticated. This includes the assumption that the inaction window is equal to an integer, the oracle assumption, or the myopic assumption for example. A natural way of continuing this work could be to refine these approximations to further improve the performance and scope of applicability of the w-policy. One could for example consider probabilistic estimates of the inaction window, determined through simulated experiments. The 'oracle' assumption could also be replaced by a the assumption that the retailer will be able to reach a given safety stock instead of satisfying exactly the demand. These possibilities were considered before being discarded, not only to avoid developing nearly incomprehensible methods, but also because we simply ran out of time.

Secondly, as mentioned in my conclusion for supply chain practitioners, I am convinced that the idea of atomizing the contributions to the action value function is simple yet powerful. I believe that it could be applied to similar inventory control problems, whose complexity arise from the fact that different items share a constraint. Under the right circumstances, the dependency between the items may be greatly simplified by making approximations similar to the inaction window or the 'oracle' assumptions.

Finally, I believe that our work on a reinforcement learning solution is an example of the challenges met by the inventory control community when trying to apply these exciting new techniques to their most complex problems. While the w-policy was preferred in the end for practical usage, it lacks the expressiveness of reinforcement learning algorithms: one could imagine more complex versions of the MOQ problems, where the w-policy would be inapplicable. For example, one could imagine that the economic parameters m^i and h^i could vary over time, or that the lead times could be stochastic instead of deterministic. In these cases, reinforcement learning architectures have the valuable capability to adapt.

Bibliography

[1] Russell L Ackoff. The future of operational research is past. *Journal of the operational research society*, 30(2):93–104, 1979.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[3] Alper Atamtürk, Gemma Berenguer, and Zuo-Jun Shen. A conic integer programming approach to stochastic joint location-inventory problems. *Operations Research*, 60(2):366–381, 2012.

[4] Sven Axsäter. Using the deterministic eoq formula in stochastic inventory control. *Management Science*, 42(6):830–834, 1996.

[5] Sven Axsäter. *Inventory control*, volume 225. Springer, 2015.

[6] Anna Azzi, Daria Battini, Maurizio Faccio, Alessandro Persona, and Fabio Sgarbossa. Inventory holding costs measurement: a multi-case study. *The International Journal of Logistics Management*, 2014.

[7] Jerry Banks. *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons, 1998.

[8] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[9] Daniel P Berovic and Richard B Vinter. The application of dynamic programming to optimal inventory control. *IEEE Transactions on automatic control*, 49(5):676–685, 2004.

[10] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

[11] Dimitri P Bertsekas. Approximate dynamic programming. 2008.

[12] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[13] Marco Bijvank and Iris FA Vis. Lost-sales inventory theory: A review. *European Journal of Operational Research*, 215(1):1–13, 2011.

[14] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[15] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

[16] Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: a review. *Evolutionary computation*, 25(1):1–54, 2017.

[17] Robert N Boute, Joren Gijsbrechts, Willem van Jaarsveld, and Nathalie Vanvuchelen. Deep reinforcement learning for inventory control: a roadmap. *European Journal of Operational Research*, 2021.

[18] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[19] Ye Chen, Joseph F Pekny, and Gintaras V Reklaitis. Integrated planning and optimization of clinical trial supply chain system with risk pooling. *Industrial & engineering chemistry research*, 52(1):152–165, 2013.

[20] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[21] Mustafa Çimen and Chris Kirkbride. Approximate dynamic programming algorithms for multidimensional flexible production-inventory problems. *International Journal of Production Research*, 55(7):2034–2050, 2017.

[22] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[23] Paulo SA Cunha, Fernanda MP Raupp, and Fabricio Oliveira. A two-stage stochastic programming model for periodic replenishment control system under demand uncertainty. *Computers & Industrial Engineering*, 107:313–326, 2017.

[24] Mark S Daskin, Collette R Coullard, and Zuo-Jun Max Shen. An inventory-location model: Formulation, solution algorithm and computational results. *Annals of operations research*, 110(1):83–106, 2002.

[25] Suzanne De Treville, Roy D Shapiro, and Ari-Pekka Hameri. From supply chain to demand chain: the role of lead time reduction in improving demand chain performance. *Journal of operations management*, 21(6):613–627, 2004.

[26] Hongwei Ding, Lyes Benyoucef, and Xiaolan Xie. Stochastic multi-objective production-distribution network design using simulation-based optimization. *International Journal of Production Research*, 47(2):479–505, 2009.

[27] Felipe Silva Placido dos Santos and Fabricio Oliveira. An enhanced l-shaped method for optimizing periodic-review inventory control problems modeled via two-stage stochastic programming. *European Journal of Operational Research*, 275(2):677–693, 2019.

[28] Qinglin Duan and T Warren Liao. A new age-based replenishment policy for supply chain inventory optimization of highly perishable products. *International journal of production economics*, 145(2):658–671, 2013.

[29] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

[30] Awi Federgruen and Yu-Sheng Zheng. An efficient algorithm for computing an optimal (r, q) policy in continuous review stochastic inventory systems. *Operations research*, 40(4):808–813, 1992.

[31] Marshall Fisher and Ananth Raman. Reducing the cost of demand uncertainty through accurate response to early sales. *Operations research*, 44(1):87–99, 1996.

[32] Jay Wright Forrester. Industrial dynamics. *Journal of the Operational Research Society*, 48(10):1037–1041, 1997.

[33] Haotian Fu, Hongyao Tang, Jianye Hao, Zihan Lei, Yingfeng Chen, and Changjie Fan. Deep multi-agent reinforcement learning with discrete-continuous hybrid action spaces. *arXiv preprint arXiv:1903.04959*, 2019.

[34] HP Galliher, Philip M Morse, and M Simond. Dynamics of two classes of continuous-review inventory systems. *Operations Research*, 7(3):362–384, 1959.

[35] Ilaria Giannoccaro and Pierpaolo Pontrandolfo. Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153–161, 2002.

[36] Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis Zhang. Can deep reinforcement learning improve inventory management? performance on dual sourcing, lost sales and multi-echelon problems. *Performance on Dual Sourcing, Lost Sales and Multi-Echelon Problems (October 6, 2020)*, 2020.

[37] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[38] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[39] Thomas W Gruen, Daniel S Corsten, and Sundar Bharadwaj. *Retail out-of-stocks: A worldwide examination of extent, causes and consumer responses*. Grocery Manufacturers of America Washington, DC, 2002.

[40] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR, 2016.

[41] Steven T Hackman and Robert C Leachman. A general framework for modeling production. *Management Science*, 35(4):478–495, 1989.

[42] John Hammersley. *Monte carlo methods*. Springer Science & Business Media, 2013.

[43] Ford W Harris. How many parts to make at once. 1913.

[44] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[46] Jafar Heydari, Reza Baradaran Kazemzadeh, and S Kamal Chaharsooghi. A study of lead time variation impact on supply chain performance. *The International Journal of Advanced Manufacturing Technology*, 40(11-12):1206–1215, 2009.

[47] Danny CK Ho, KF Au, and Edward Newton. Empirical research on supply chain management: a critical review and recommendations. *International journal of production research*, 40(17):4415–4430, 2002.

[48] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[50] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[51] Donald L Iglehart. Optimality of (s, s) policies in the infinite horizon dynamic inventory problem. *Management science*, 9(2):259–267, 1963.

[52] Tibor Illés and Tamas Terlaky. Pivot versus interior point methods: Pros and cons. *European Journal of Operational Research*, 140(2):170–190, 2002.

[53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[54] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[55] Ilya Jackson, Jurijs Tolujevs, and Zhandos Kegenbekov. Review of inventory control models: A classification based on methods of obtaining optimal control parameters. *Transport and Telecommunication*, 21(3):191–202, 2020.

[56] M Joly and JM Pinto. Mixed-integer programming techniques for the scheduling of fuel oil and asphalt production. *Chemical Engineering Research and Design*, 81(4):427–447, 2003.

[57] Peter Kall, Stein W Wallace, and Peter Kall. *Stochastic programming*. Springer, 1994.

[58] Ahmet Kara and Ibrahim Dogan. Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications*, 91:150–158, 2018.

[59] Samuel Karlin. Dynamic inventory policy with varying stochastic demands. *Management Science*, 6(3):231–258, 1960.

[60] Soheyl Khalilpourazari and Seyed Hamid Reza Pasandideh. Multi-objective optimization of multi-item eoq model with partial backordering and defective batches and stochastic constraints using mowca and mogwo. *Operational Research*, pages 1–33, 2018.

[61] Gudrun P Kiesmüller, AG De Kok, and S Dabia. Single item inventory control under periodic review and a minimum order quantity. *International Journal of Production Economics*, 133(1):280–285, 2011.

[62] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[63] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[64] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[65] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[66] Hau L Lee and Corey Billington. Managing supply chain inventory: pitfalls and opportunities. *Sloan management review*, 33(3):65–73, 1992.

[67] Hau L Lee, Venkata Padmanabhan, and Seungjin Whang. The bullwhip effect in supply chains. *Sloan management review*, 38:93–102, 1997.

[68] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[69] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. In *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 478–485. IEEE, 2018.

[70] Helena Ramalhinho Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer, 2019.

[71] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[72] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[73] Eric Porras Musalem and Rommert Dekker. Controlling inventories in a supply chain: A case study. *International Journal of Production Economics*, 93:179–188, 2005.

[74] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Conference on Robot Learning*, pages 735–751. PMLR, 2020.

[75] Muhittin Oral, Michael S Salvador, Arnold Reisman, and Burton V Dean. On the evaluation of shortage costs for inventory control of finished goods. *Management Science*, 18(6):B–344, 1972.

[76] Jun Hyeong Park, Jong Soo Kim, and Ki Young Shin. Inventory control model for a supply chain system with multiple types of items and minimum order size requirements. *International Transactions in Operational Research*, 25(6):1927–1946, 2018.

[77] Edgar Perea, Ignacio Grossmann, Erik Ydstie, and Turaj Tahmassebi. Dynamic modeling and classical control theory for supply chain management. *Computers & Chemical Engineering*, 24(2-7):1143–1149, 2000.

[78] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[79] Warren B Powell, Abraham George, Belgacem Bouzaiene-Ayari, and Hugo P Simao. Approximate dynamic programming for high dimensional resource allocation problems. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 5, pages 2989–2994. IEEE, 2005.

[80] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[81] Jafar Rezaei and Mansoor Davoodi. A deterministic, multi-item inventory model with supplier selection and imperfect quality. *Applied Mathematical Modelling*, 32(10):2106–2116, 2008.

[82] Jafar Rezaei and Mansoor Davoodi. Multi-objective models for lot-sizing with supplier selection. *International Journal of Production Economics*, 130(1):77–86, 2011.

[83] David John Robb and Edward Allen Silver. Inventory management with periodic ordering and minimum order quantities. *Journal of the Operational Research Society*, 49(10):1085–1094, 1998.

[84] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

[85] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems*, pages 2488–2498, 2018.

[86] Haralambos Sarimveis, Panagiotis Patrinos, Chris D Tarantilis, and Chris T Kiranoudis. Dynamic modeling and control of supply chain systems: A review. *Computers & operations research*, 35(11):3530–3561, 2008.

[87] Herbert Scarf. The optimality of (5, 5) policies in the dynamic inventory problem. *Optimal pricing, inflation, and the cost of price adjustment*, 1959.

[88] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[89] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[90] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[91] Herbert A Simon. On the application of servomechanism theory in the study of production control. *Econometrica: Journal of the Econometric Society*, pages 247–268, 1952.

[92] Jing-Sheng Song and Paul Zipkin. Inventory control in a fluctuating demand environment. *Operations Research*, 41(2):351–370, 1993.

[93] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[94] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[95] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[96] Douglas J Thomas and Paul M Griffin. Coordinated supply chain management. *European journal of operational research*, 94(1):1–15, 1996.

[97] Huseyin Topaloglu and Warren B Powell. Dynamic-programming approximations for stochastic time-staged integer multicommodity-flow problems. *INFORMS Journal on Computing*, 18(1):31–42, 2006.

[98] Denis R Towill. Dynamic analysis of an inventory and order based production control system. *The international journal of production research*, 20(6):671–687, 1982.

[99] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[100] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[101] Arthur F Veinott, Jr. On the opimality of (s,s) inventory policies: New conditions and a new proof. *SIAM Journal on Applied Mathematics*, 14(5):1067–1083, 1966.

[102] Harvey M Wagner and Thomson M Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1):89–96, 1958.

[103] Clyde K Walter and John R Grabner. Stockout cost models: Empirical tests in a retail situation. *The Journal of Marketing*, pages 56–60, 1975.

[104] Rui Wang, Xianghua Gan, Qing Li, and Xiao Yan. Solving a joint pricing and inventory control problem for perishables via deep reinforcement learning. *Complexity*, 2021, 2021.

[105] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[106] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[107] RH Wilson. *A scientific routine for stock control*. Harvard Univ., 1934.

[108] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

[109] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[110] Fengqi You and Ignacio E Grossmann. Mixed-integer nonlinear programming models and algorithms for large-scale supply chain design with stochastic inventory management. *Industrial & Engineering Chemistry Research*, 47(20):7802–7817, 2008.

[111] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019.

[112] Yao Zhao and Michael N Katehakis. On the structure of optimal ordering policies for stochastic inventory systems with minimum order quantity. *Probability in the Engineering and Informational Sciences*, 20(2):257, 2006.

[113] Yu-Sheng Zheng and Awi Federgruen. Finding optimal (s, s) policies is about as simple as evaluating a single policy. *Operations research*, 39(4):654–665, 1991.

[114] Bin Zhou. Inventory management of multi-item systems with order size constraint. *International journal of systems science*, 41(10):1209–1219, 2010.

[115] Bin Zhou, Yao Zhao, and Michael N Katehakis. Effective control policies for stochastic inventory systems with a minimum order quantity and linear costs. *International Journal of Production Economics*, 106(2):523–531, 2007.

Mots clés : Gestion de stock, opttimisation de la chaine logistique, programmation dynamique stochastique, apprentissage par renforcement.
Keywords : Inventory Control, Supply Chain Optimization, Stochastic Dynamic Programming, Reinforcement Learning.

Résumé en Français

Dans de nombreuses chaînes logistiques, il est courant que les fournisseurs imposent une quantité minimale d'achat à leurs acheteurs (par exemple sur la valeur totale de la commande), dans le but de diluer l'impact de leurs coûts fixes. Dans le cas où cet acheteur est lui même un revendeur, ce dernier est alors soumis à un problème d'optimisation de stock difficile : il doit d'un côté maintenir un stock suffisant pour satisfaire la demande (instable et non certaine) de ses clients, et de l'autre réduire ses coûts de stock, tout en respectant la contrainte de son fournisseur. C'est ce problème de prise de décision sous incertitude que nous avons cherché à résoudre au cours de cette thèse. Nous l'appelons *problème de quantité minimale de commande multi-références*.

Il existe plusieurs méthodes dans la littérature pour résoudre des versions simplifiées de ce problème, notamment lorsqu'une seule référence est concernée par la contrainte, ou lorsque la demande est supposée stationnaire. Cependant, aucune solution de l'état de l'art n'apportait jusqu'alors de réponse satisfaisante à la version multi-référence de ce problème, avec un inventaire soumis à une demande stochastique et variable dans le temps.

Les principales contributions de cette thèse sont deux méthodes de résolution approximative de ce problème. La première est une heuristique que nous appelons la *w-policy*. Cette heuristique repose sur plusieurs approximations du comportement du système. Ces approximations, basées sur une analyse poussée du problème, permettent de réduire drastiquement la complexité du calcul des fonctions de valeur et d'en déduire une solution approximative. Cependant, en raison de ces hypothèses, la *w-policy* ne peut pas être appliquée dans certains cas particuliers. Dans le but de remédier à cette limite, nous avons développé une deuxième méthode que nous appelons la *hybrid policy*. Cette méthode combine des techniques issues de l'apprentissage par renforcement (notamment le deep Q-learning) avec certaines idées issues de la *w-policy*. Nous montrons la capacité de ces deux méthodes à résoudre efficacement le problème posé, en l'appliquant à des jeux de données réelles et simulées. Sur les versions de grande taille du problème (jusqu'à dix mille références), elles sont les seules aujourd'hui à apporter une solution calculable en un temps raisonnable.

Résumé en Anglais

Manufacturers and suppliers can often leverage economies of scale to reduce their manufacturing and transportation costs, since it dilutes the impact of fixed costs. Many actors in supply chains choose to shift the burden of these fixed costs to their purchasing entities, by introducing mandatory minimum order size (for example on the total monetary value of the order). We define the Minimum Order Quantity (MOQ) problem as the inventory control problem that purchasing entities which are themselves retailers face in this situation. A retailer in this situation should aim to replenish its inventory stock levels to a balanced state, to avoid stock-outs and overstocks, while satisfying its supplier constraint.

There are several methods in the literature that deal with simplified versions of this problem, notably for the single item or the stationary demand versions of the problem. However, no state of the art solution was able to provide a realistic solution to the multi-item, variable demand version of the problem.

The main contributions of this thesis are two methods that compute approximate solutions to this problem. The first one is the w-policy, a heuristic based on several assumptions of the system. These assumptions were justified by a extensive analysis of the MOQ problem. They drastically reduce the complexity of the computation of the value functions, which leads to an efficient computation of an approximate solution. The scope of applicability of the w-policy is however bounded, and this policy is inapplicable in some specific settings. In order to overcome this limitation, we developed a second method that we called the 'hybdrid' policy. This method combines reinforcement learning techniques (notably deep Q-learning) with some ideas from the w-policy. We demonstrate the ability of these two methods to solve the MOQ problem efficiently on simulated and real datasets, on scales that were unprecedented (up to ten thousand items).