Computing Policy Parameters for Stochastic Inventory Control Using Stochastic Dynamic Programming Approaches

Andrea Visentin
MSC
114104989

Thesis submitted for the degree of Doctor of Philosophy

NATIONAL UNIVERSITY OF IRELAND, CORK

COLLEGE OF SCIENCE, ENGINEERING AND FOOD SCIENCE

SCHOOL OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY
INSIGHT CENTRE FOR DATA ANALYTICS

August 2020

Head of Department: Professor Cormac J. Sreenan

Supervisors: Dr Steven Prestwich
Professor Ken Brown

Contents

List of Figures

I, Andrea Visentin, certify that this thesis is my own work and has not been submitted for another degree at University College Cork or elsewhere.

Andrea Visentin

Abstract

The objective of this work is to introduce techniques for the computation of optimal and near-optimal inventory control policy parameters for the stochastic inventory control problem under Scarf's setting. A common aspect of the solutions presented herein is the usage of stochastic dynamic programming approaches, a mathematical programming technique introduced by Bellman. Stochastic dynamic programming is hybridised with branch-and-bound, binary search, constraint programming and other computational techniques to develop innovative and competitive solutions.

In this work, the classic single-item, single location-inventory control with penalty cost under the independent stochastic demand is extended to model a fixed review cost. This cost is charged when the inventory level is assessed at the beginning of a period. This operation is costly in practice and including it can lead to significant savings. This makes it possible to model an order cancellation penalty charge.

The first contribution hereby presented is the first stochastic dynamic programming that captures Bookbinder and Tan's static-dynamic uncertainty control policy with penalty cost. Numerous techniques are available in the literature to compute such parameters; however, they all make assumptions on the demand probability distribution. This technique has many similarities to Scarf's stochastic dynamic programming formulation, and it does not require any external solver to be deployed. Memoisation and binary search techniques are deployed to improve computational performances. Extensive computational studies show that this new model has a tighter optimality gap compared to the state of the art.

The second contribution is to introduce the first procedure to compute cost-optimal parameters for the well-known (R, s, S) policy. Practitioners widely use such a policy; however, the determination of its parameters is considered computationally prohibitive. A technique that hybridises stochastic dynamic programming and branch-and-bound is presented, alongside with computational enhancements. Computing the optimal policy allows the determination of optimality gaps for future heuristics. This approach can solve instances of considerable size, making it usable by practitioners. The computational study shows the reduction of the cost that such a system can provide.

Thirdly, this work presents the first heuristics for determining the near-optimal parameters for the (R, s, S) policy. The first is an algorithm that formally models the (R, s, S) policy computation in the form of a functional equation. The second is a heuristic formed by a hybridisation of (R, S) and (s, S) policy parameters solvers. These heuristics can compute near-optimal parameters in a fraction of time compared to the exact methods. They can be used to speed up the optimal branch-and-bound technique.

The last contribution is the introduction of a technique to encode dynamic programming in constraint programming. Constraint programming provides the user with an expressive modelling language and delegates the search for the solution to a specific solver. The possibility to seamlessly encode dynamic programming provides new modelling options, e.g. the computation of optimal (R, s, S) policy parameters. The performances in this specific application are not competitive with the other techniques proposed herein; however, this encoding opens up new connections between constraint programming and dynamic programming. The encoding allows deploying DP based constraints in modelling languages such as MiniZinc. The computational study shows how this technique can outperform a similar encoding for mixed-integer programming.

Acknowledgements

Firstly, I would like to express my deepest gratitude to my supervisor Dr Steven Prestwich for the incredible opportunity he offered me and for the continuous support of my PhD study and related research. Undertaking this PhD has been a truly life-changing experience, and it would not be possible without his guidance. I sincerely thank him for being so supportive and encouraging with the different projects we tackled together. I learned so much from him; I hope and believe that this work is the beginning of a longer collaboration.

Besides my supervisor, I would like to thank Prof. S Armagan Tarim and Prof. Roberto Rossi. Their deep and honest passion for Operations Research deeply inspired me. Working with them taught me a lot, I want to thank them for their insightful feedback and for their hard questions which pushed me to improve my work.

I would like to thank the examiners, Prof Gregory Provan and Prof. Ian Miguel for the interesting discussion and the useful feedback. This work improved thanks to them.

I wish to express my sincere gratitude to all the Insight Centre For Data Analytics. To Prof. Barry O'Sullivan for widening my research perspective and for allowing me to give a small contribution to his work for the European Commission, an institution that I strongly admire. My thanks go to all the staff of Insight for their professionalism and kindness.

I want to thank all the Computer Science Department. In particular, Prof. Cormac J. Sreenan for allowing me to teach my first module. And Dr Sabin Tabirca for his passion for developing new talents and for choosing me as team leader for the Irish representative at the International Olympiad in Informatics. I want to thank the IOI athletes; especially Andrew and Kieran, for the interesting discussions that inspired a part of this work.

Some of the brightest people I ever met passed through our lab in these years. We shared exciting discussions, challenges, anxiety, scopette, lunches, boardgames, travels and much more. I am proud to call many of you not only colleagues but friends. A special mention goes to (almost Dr) Diego Carraro and (almost Dr) Federico Toffano, we met at the beginning of our academic journey more than ten years ago, and we conquered many challenges together. My PhD would not have been the same without two lifelong friends alongside me.

My personal acknowledgements must start with my family. You do not choose your family, but I could not have chosen better. I am incredibly fortunate to have such wonderful parents and a lovely sister. I want to thank all the Visentin and Michielin. They have been a source of inspiration, laughs, love and kindness. My whole family is the dearest thing to me, from the one that already let ones to the one that will join us this year.

A special mention goes to my girlfriend, Ana. For her support during these two years. And of course, for putting up with me during the quarantine. I am fortunate to have her by my side.

I want to thank: Lara, for being there for me and for giving me the honour of being her bridesman, I wish her and Matteo the best for their future. Simone, because you can be best friends even thousands of kilometres away, and congratulations to him and Alice for the little Leo. All the TFS, Marta, the Caonada team and the Carri.

There are so many people that I should thank for making my Irish experience unforgettable. The International Students Society, the Handball and Futsal Club, the Club Exec, the IOI team for all the friends I met during those experiences and for all the things that I learned. The Ciaccarelli because they can make you feel at home in Italy even on a rainy day. To Nico and all the amazing people I met during my Erasmus.

I cannot thank all the people that have been important to me in these years, but I will try to express my gratitude to them as much as I can.

Chapter 1

Introduction

In this chapter, we briefly describe the goal and scope of this thesis. Section 1.1 presents the motivations behind this work. We briefly introduce inventory control and we describe its importance for practitioners. We present the contributions of this work in Section 1.2. Section 1.3 lists the publications. Finally, the thesis structure is presented in Section 1.4.

1.1 Motivation

An inventory control system is used to control the number of products in the inventory to satisfy the demand adequately. These systems are used to decide the timing and size of the replenishments. They can be in an autonomous way or as a support decision tool. Optimising these decisions can lead to important savings, and their impact is increasing nowadays thanks to advances in information, communication and transport technologies. More data are available to process, and their accuracy increased as well. These innovations allow companies to cope with an evolving situation, characterised by shortened product life cycles, globalised supply chain, and eCommerce market [EL13].

Recent statistics confirm the growing interest in inventory management. The number of warehouses in the U.S. is continuously increasing. More than 3000 new warehouses opened in the country in the last seven years. The annual increase is rising since 2009, regardless of the economic crisis (U.S. Bureau of Labor Statistics, 2019).

These data are consistent with the Motorola Warehouse Vision Survey, where

more than one-quarter of the surveyed professionals claimed that their distribution centres are viewed as an asset that can drive the growth for the business, and 35% of them plan to increase the number of warehouses they operate. The companies are increasing their investments in inventory operations technologies. In particular, approximately two-thirds of the retailers plan to increasingly automate the process by deploying new technology solutions. A 100% increase of the digitisation of cycle counting and validation will provide more reliable information for the techniques proposed herein. These warehouses reduce shipping costs and cut down potential border tariffs, a critical discussion topic in modern politics. Moreover, having them closer to the demand allows for dealing with demand changes faster.

Inventory control is one of the most successful branches of Operational Research. The first work in this area is considered to be [Har13]. Harris introduces the well known *Economic Order Quantity (EOQ)* model. This model answers the problem of determining when an order should be placed and its size. While this pioneering work is a fundamental part of the inventory control field, it is limited by its strong assumptions. Harris assumptions are:

- The demand is constant and deterministic.
- The order quantity can be non-integer.
- There are no restrictions on the order size (min/max).
- The purchasing unit cost is not affected by the order size; no quantity discounts are allowed.
- The cost factors are constant over the planning horizon.
- Items are treated independently, no joint review/shipping and no substitute items.
- There is no lead time, i.e. orders are delivered instantaneously when placed, all the items are delivered at the same time.
- Shortages are not allowed.
- The planning horizon is very long, and the conditions stay the same for all of it.

A wide part of inventory control literature provides techniques to answer the two research questions when these assumptions are relaxed or changed.

In this work, we focus on methods that relax the first assumption. In real-world inventory systems, the demand is hardly known in advance, especially in business-to-consumer applications. This is due to the inherent qualities of the business and its customer base. Moreover, demand can change over time. Many products are affected by seasonality, by a shorter life-cycle or by customer preference shift. Modelling this uncertainty can lead to more robust and better-performing inventory planning. For example, a drop in the demand can cause a high amount of leftovers that consume physical space and block capital that could be invested otherwise. The leftovers can expire, become outdated, get damaged or stolen. On the other hand, an unexpected spike in demand can lead to a shortage, and this can cause lost sales and lower customer service. [TKTE11] conducts a comparison study on the cost of adopting stationary policies in a non-stationary demand situation.

A well-developed branch of inventory control focuses on techniques tackling these particular issues: the non-stationary stochastic lot sizing. In this setting, the demand is a set of stochastic variables of which we know the probability distributions. These variables are generally considered independent and can be different in each period. Modelling these possibilities allows to better cope with the uncertainty of the demand and the seasonality or life cycle of some products. [AHM51] is the first known work in stochastic inventory models.

An important class of these problems is the single-item single-location non-stationary stochastic lot sizing under linear holding costs, penalty costs and both linear and fixed ordering costs. Different policies can be used to determine the size and timing of the orders on such setting [Sil81].

In his seminal work [Sca59], Scarf characterises the structure of the optimal policy for such problem. The framework proposed by Bookbinder and Tan [BT88] divides the policies into three classes: static uncertainty, dynamic uncertainty and static-dynamic uncertainty. These classes differ on the moment in which the decisions are taken. A policy is static uncertainty if the fixing of the ordering moments and sizes happens at the beginning of the time horizon. If these decisions are taken after observing the demand for previous periods, the strategy is dynamic uncertainty. Otherwise, if the two parameters are fixed at different times, the policy is static-dynamic. This research motivated several following works, including this thesis. Different comparison studies have been conducted recently to benchmark different aspects of these policies: Kilic et al. [KT11] extends a measure of planning instability for the non-

stationary stochastic lot sizing; Dural et al. [DSRKT19] compares different policies performances in the receding horizon. A broad picture of the state-of-the-art in lot sizing can be found in a recent review of papers reviews [BGJK15].

The approaches available in the stochastic lot sizing literature present several drawbacks. To the best of our knowledge, no work in the literature considers an inventory review cost factor. This cost is charged when the inventory level is assessed at the beginning of a period. Inventory review (also known as stock-taking) is costly in practice, and its modelisation can lead to significantly savings. No inventory strategy has been proved cost-optimal for this extension. This cost can be used to model order cancellations as well. If the demand observed is less than expected, the management can cancel a previously scheduled order. Order cancellation is usually associated with a penalty cost. To the best of our knowledge, stochastic lot sizing literature does not include works including the possibility of cancelling an order.

The (R, s, S) is a type of inventory control policy that we describe in Chapter 3. A stronger interest towards it is a direct consequence of the introduction of the review cost factor. This inventory control policy is known since the "golden age" of inventory research of the 1950s, and it has a strong value for practitioners. The inventory is reviewed at review intervals R. If the inventory level is lower than an order level s, an order is placed to raise it to an order-up-to-level S. In the absence of a review cost, it is equivalent to the cost-optimal (s, S) policy with a periodic planning horizon. Including this cost factor, the policy outperforms the (s, S) one. Computing optimal, or near-optimal, (R, s, S) parameters has been considered too computationally expensive. It is one of the most general and frequently used inventory policies; however as pointed out by [Sil81], "*the determination of the exact best values of the three parameters is extremely difficult*". To the best of our knowledge, no heuristic or optimal approach to computing them has been presented in the literature surveyed in Section 3.4.5.

Another problem that has been widely tackled in the literature is the computation of parameters for the (R, S) policy, also known as the static-dynamic strategy. This policy can outperform the (s, S) one in the presence of a review cost. However, no pure stochastic dynamic programming formulation is available for the computation of such parameters under the assumption of a linear penalty cost in case of a stockout. Furthermore, as Section 3.4.4 shows, most of the techniques available make strong assumptions on the demand type.

This thesis aims to cover these unsolved research shortfalls. To do so, we model the review cost and adapt existing methods to it. We adapt existing techniques to compute optimal (R, s, S) policy parameters. We introduce new optimal techniques and heuristics for the same problem, all based on stochastic dynamic programming (SDP).

Bellman [Bel66] has originally introduced SDP. It is a technique used to model and solve problems of decision making involving uncertainty. Some of the parameters of the problem are random variables of which the probability distribution is known. SDP involves aspects of stochastic programming and dynamic programming (DP). Scarf's SDP [Sca59] is the most famous stochastic lot sizing algorithm, and it is widely used to compute the expected value of the cost-optimal policy, e.g. [RTHP11, DSRKT19].

In this work, we combine SDP with different techniques to achieve competitive algorithms. The most important are: branch-and-bound (BnB), DP and constraint programming (CP). BnB is an algorithm design paradigm based on a rooted search tree. The algorithm explores the branches of the tree, computing partial solutions. This partial assignment of the decision variables is compared with bounds to check if it can produce a better solution than the best so far. If a partial solution is proved to be non-optimal, a pruning of the branch occurs. The bounds are computed using a DP solution. The DP approach builds an optimal solution by breaking the problem down into sub-problems and solving each to optimality in a recursive manner, achieving great efficiency by solving each sub-problem only once.

CP is one of the most active fields in artificial intelligence. Designed to solve optimisation and decision problems, it provides expressive modelling languages, development tools and global constraints. CP has already been used successfully in lot sizing, e.g. [RTHP08, RTHP12]. An overview of the current status of CP and its challenges can be found in [Fre18].

Thesis

The review cost is an important cost factor of an inventory control system. Its introduction makes the (R, s, S) and the (R, S) policies more cost-efficient compared to the (s, S) one. SDP based techniques can be used to compute (near-)optimal (R, s, S) policy parameters in single-item, single-location, non-stationary stochastic lot sizing with review cost. The resulting solutions

outperform cost-wise the state-of-the-art approaches.

The work herein fully support this thesis; the next chapters present algorithms that are:

- *Innovative in the problem configuration*. The stochastic lot sizing problem with non-stationary demand and review cost has not been tackled in the literature. The introduction of the review cost allows deploying more accurate models of real-world applications.
- *Innovative in terms of the policy used*. The (R, s, S) policy has many advantages in terms of flexibility and robustness to unexpected demand. For these reasons, it is widely used by practitioners. No algorithm to compute a (near-)optimal set of its parameters is available in the literature. The reason is the high computational effort required to compute them.
- *Computable in reasonable time*. The techniques we propose herein outperform a naive solution by orders of magnitude, making them usable by practitioners.
- *Novel in terms of hybridisation of techniques*. A hybridisation of SDP and BnB is a novel approach in lot sizing. To speed it up, we introduce novel DP bounds. We present the first modelling approach to encode DP and SDP into a CP model, this encoding open new possible integrations between these two fields.
- *Optimal or near-optimal*. We present the first cost-optimal solution for the problem of computing (R, s, S) policy parameters. We describe novel heuristics that exhibit a close optimality gap and that are competitive in terms of computational performances.

The complexity of supply chains has increased significantly over the past years. This is due to an increase in the number of items to be managed and a globalised production. In this environment, optimised management of the stock provides important saves for a business. The closer the mathematical model of the real problem is, the higher are the potential savings. The work presented in this thesis aims to extend the current modelling of stochastic non-stationary lot sizing and present a (near-)optimal algorithms to compute its parameters.

1.2 Contributions

The contributions of this thesis are:

- The modelling of a new cost factor in the lot sizing formulation.
- An SDP compact formulation for the (R, S) policy under penalty cost.
- An extension of the well known SDP solution for the (s, S) policy [Sca59] that allows it to compute optimal parameters for the (R, s, S) policy.
- The first algorithm to compute the optimal parameters of an (R, s, S) policy under the stochastic non-stationary assumption. This solution is based on SDP and BnB.
- A set of techniques to strongly improve the performances of the previous solution, including a state space reduction and bounds computed with dynamic programming.
- An SDP designed to compute the parameters for the (R, s, S) policy. This formulation is a combination of the (s, S) and the (R, S) SDP formulations.
- The introduction of a heuristic for the (R, s, S) policy that is based on the combination of (R, S) and (s, S) algorithms. This heuristic can be used to improve the performances of the BnB solution.
- A new technique that allows to model dynamic programming formulations in constraint programming. This modelling technique can be used to solve to optimality the problem of computing (R, s, S) policy parameters. The solution based on this formulation is not competitive compared to the others proposed herein. However, the modelling technique has an intrinsic value and many possible applications, e.g. in network interdiction problems [IW02].

1.3 Publications

We have published the work described in this thesis in international journals, conferences, their workshops and poster sessions. These publications are listed as follows:

- Andrea Visentin and Steven Prestwich and Roberto Rossi and S Armagan Tarim: *Computing Optimal (R, s, S) Policy Parameters by a Hybrid of Branch-and-Bound and Stochastic Dynamic Programming*, in European Journal of Operational Research, Under review.
- Andrea Visentin and Steven Prestwich and Roberto Rossi and S Armagan Tarim : *Modelling Dynamic Programming-Based Global Constraints in Constraint Programming*, in proceedings of the 6th World Congress on Global Optimization, Optimisation of Complex Systems: Theory, Models, Algorithms and Applications.
- Steven Prestwich and Roberto Rossi and S Armagan Tarim and Andrea Visentin: *Towards a Closer Integration of Dynamic Programming and Constraint Programming*, in proceedings of the 4th Global Conference on Artificial Intelligence 55, 202.
- Andrea Visentin: *Optimal (R, s, S) policy for single-item inventory lot sizing problem with stochastic non-stationary demand*: in 10th International Workshop on Lot Sizing.

Unrelated to our work on inventory control, we also published the following:

- Andrea Visentin and Alessia Nardotto and Barry O'Sullivan: *Predicting Judicial Decisions: A Statistically Rigorous Approach and a New Ensemble Classifier*, in Proceedings of 31st International Conference on Tools with Artificial Intelligence.
- Andrea Visentin and Steven Prestwich and S Armagan Tarim: *Robust principal component analysis by reverse iterative linear programming*, in Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases

1.4 Thesis Structure

- In Chapter 1, we present the motivation behind our work and its goals. Finally, we summarise the contributions and the thesis' structure.
- In Chapter 2, we describe the lot sizing problem and the settings used in this work. Then, we extend the existing mathematical model to incorporate the review cost.

- In Chapter 3, we introduce the key techniques used in this work, and we describe the inventory control policies used. Then, we position our problem settings in the lot sizing literature, and we survey the existing techniques to compute the policy parameters for the non-stationary stochastic lot sizing.
- In Chapter 4, we describe Scarf's SDP and the K-convexity property. Then, we present the first computable formulation of the (R, S) policy computation into an SDP functional equation. This formulation shares the same settings of Scarf's one, with no assumption on the demand type. We then present a series of computational enhancements that strongly reduce the computational time without compromising the optimality of the technique. Extensive computational experiments prove the quality of the technique that holds a better optimality gap compared to the state-of-the-art approach.
- In Chapter 5, firstly, we present a naive approach based on Scarf's work that we use as a comparison. Then, we present the first algorithm to compute cost-optimal parameters for the (R, s, S) policy. This technique is a hybridisation of SDP and BnB, improved by tight bounds computed with DP. It outperforms the baseline by several orders of magnitude, making it usable by practitioners. An extensive computational study proves the quality of the solution.
- In Chapter 6, we present a pure SDP formulation for the computation of the (R, s, S) policy parameters. This formulation inherits aspects from Scarf's SDP, like the K-convexity property, and some from the (R, S) SDP model of Chapter 4. We present a new heuristic based on a two-step approach: firstly the replenishment cycles length is determined using the (R, S) policy, then the order levels and the order-up-to-levels are computed using the resulting (s, S) policy. This approach offers a quick heuristic to compute cost-efficient parameters faster than the other solutions. The near-optimal solution can be used to speed up the technique proposed in 6.
- In Chapter 7, we present the dynamic programming encoding, a novel approach to model DP and SDP in constraint programming. This can be used to model the computation of policy parameters as a monolithic CP model. This technique offers higher flexibility in terms of additional constraints. We then show the potential of this application in the

development of DP based constraints in CP.

- In Chapter 8, we draw the conclusions of this thesis, and we analyse possible future works.

Chapter 2

Inventory Control Concepts

This chapter aims to describe all the aspects of the problem tackled in this work. Inventory control problems can arise in a wide variety of real-world situations, each with its peculiarities. These aspects of the practical problem can be modelled in different ways; a model closer to reality can lead to significant savings.

Section 2.1 gives a description of an inventory control system and define the problem type tackled herein. We describe the different modelling alternatives, focusing on the one used. The mathematical formulation is presented in Section 2.2.

2.1 Problem setting

In this section, we aim to give an extensive problem definition. We describe the aspects of an inventory control system and the different settings that can be used, this allows us to position our work in the lot sizing literature and to explain the connections with the real-world problem. This Section is mainly based on [SPT16].

We classify the problem type according to the framework proposed in [SPT16]. They propose six functional decision categories for controlling inventories: cycle stock, congestion stock, safety stock, anticipation inventories, pipeline inventories, and decoupling stock. Our problem is classified as cycle stock. Cycle inventories order or produce in batches instead of one unit at a time. The amount of inventory on-hand, at any point, that results from these

batches is called cycle stock, we will refer at it merely as stock from now on. The reasons for batch replenishments include economies of scale (because of substantial setup/ordering costs), quantity discounts in the purchase price, and technological restrictions such as the fixed size of a processing tank in a chemical process. We are considering the case in which there is a considerable setup/ordering cost. The frequency of the orders affects the amount of cycle stock on-hand at any time.

The literature refers to this problem configuration as Scarf's setting [Sca59]. We decided to tackle it because it is a widely studied configuration that can be easily extended to accommodate different practical problem. Moreover, the literature offers numerous algorithms that we can use as comparison.

In the next sections, we describe the configuration of the problem. Section2.1.1 gives a brief introduction to the goals of an inventory control system. In Section 2.1.2, we describe the stochastic part of the problem and its connection with real-world applications. Section 2.1.4 contains the different type of costs that an inventory control system can face. Section 2.1.3 classifies how our problem is positioned in a supply chain plan. Section 2.1.5 presents the lead time, the time that occurs between the decision of placing the order and having the items ready to satisfy the demand. Section 2.1.6 introduces the way we deal with stockouts. In Section 2.1.7, we give the inventories definitions we use herein. The ways to make a model avoid stockouts are presented in Section 2.1.9. Section 2.1.10 explains which decisions an inventory control system has to take. Finally, Section 2.1.11 describes what an inventory control policy is.

2.1.1 Measures of Effectiveness

Analysts in the inventory area have tended to concentrate on a single measure of effectiveness, the expected cost of a policy. Other objectives are usually modelled through constraints, such as limited space, desired customer service, the nervousness of the inventory. The system nervousness is a complication of unforeseen demand that often occurs in supply chain networks, where the decision maker continually change the size and timing of scheduled replacements. The lot-sizing literature defines two type of nervousness: the setup-oriented involves a cancellation of reschedule of a planned order, the quantity oriented refers to revision in replenishment quantities.

In a practical situation, the impact of a policy is not restricted to a single

measure of effectiveness. [SPT16] lists other objectives that are difficult to quantify:

1. Minimising the political conflicts within the organisation.
2. Maintaining a high level of flexibility to cope with an uncertain future.
3. Maximising the chance of survival of the firm or the individual manager's position within the organisation.
4. Keeping at an acceptable level the amount of human effort expended in the planning and operation of a decision system.

While the expected cost is the most used measure in the literature, alternatives objective have been considered as well. For example, Azaron et al. [ABTM08] consider the minimisation of the sum of current investment costs, the minimisation of the variance of the total cost and the minimisation of the financial risk or the probability of not meeting a specific budget. As in most of the literature, we focus on the objective that is easier measurable, the cost. The optimal inventory control system is the one that provides the minimum cost.

2.1.2 Non-stationary stochastic demand

The demand to satisfy can change over time. The time-varying demand situations allow modelling a broader range of practical situations, including:

- Life cycle products. Where the demand rapidly increases in the early phases, stabilising during the central part and slowly reduces with obsolescence.
- Production to contract, where the contract requires that certain quantities have to be delivered to the customer on specified dates.
- Items having a seasonal demand pattern.
- Multiechelon assembly operations.
- Replacement part for an item out of production.

This type of demand makes the usage of the same order quantity generally not economically competitive. We consider the demand information over a finite period. This period is known as the planning horizon, and its length can have a substantial effect on the total relevant costs of the selected strategy. Usually, it

is preferred to have the planning horizon as short as possible; the farther into the future we look for demand information, the less accurate it is likely to be.

Stochastic demand allows us to better cope with the uncertainties of practical applications. In the case of stochastic demand, the variation is represented by different stochastic variables that represent the demand of each period. We assume that we have perfect knowledge of the probability distribution of these variables. This is a more general case of the deterministic one, all the stochastic solution work also for the deterministic problem. In the case of non-deterministic demand, the complexity of computing the optimal policy strongly increases. Stockouts can happen if unexpectedly high demand arrives, or high level of unsold inventory in case of low demand. Maintaining a minimum customer service level is particularly challenging. This problem is known as non-stationary stochastic or time-varying probabilistic demand. In this situation, the policy's cost is represented by the expected cost of the policy over the time horizon.

A wider variety of problem parameters can involve stochasticity. However, the uncertainty of the demand is the most frequent in a real-world problem, and its modelling leads to more significant saves. Since we present algorithms for policies previously unused in a stochastic lot sizing setting, we prefer to use the most common problem configuration. We consider treating other parameters as stochastic as an extension of this work; we present some of them in Section 8.1. The non-stationarity can involve other parameters as well. All the algorithms that this thesis introduces can deal with non-stationary cost parameters.

2.1.3 Single-echelon, single-item, single-location

The configuration analysed in this work is the single-echelon, single-item and single-location one. In single-echelon problems, we need to deal with an individual part of the supply chain. The inventory control system manages the transportation of the items only from the supplier, the customer (or entity that provides the demand) is directly collecting the item or managing the shipping part. For example, in a retail environment, this can be used to model a shop.

In the multi-echelon problems, the inventory control system determines the correct levels of inventory across different nodes of the supply chain, considering their interaction. Following the previous example, the problem of managing

the inventory level of both a distribution centre and the shops supplied by it is a two-echelon problem, see [SR91, SRK95].

In inventory control, items are generally divided in a minimum of three different classes (generally called A, B and C). These classes divide the items based on how much they individually contribute to the total annual monetary usage. Depending on the type of product, the individual procurement and sale of the products can be managed individually or in batches. We consider single item inventory policies.

Even when dealing with a single-item single-echelon problem, there is the possibility of dealing with multiple warehouses. Herein we analyse the single-location non-capacitated setting, a single storage location for the items with virtually infinite capacity.

2.1.4 Cost factors

In this section, we briefly describe the cost factors modelled in our problem. These are not all the factors that affect the total cost of an inventory system. However, they are generally considered the most relevant in real-world applications. In this work, we do not analyse techniques to estimate these cost factors. We consider their values as parameters provided by the management.

These costs are:

- *The unit variable cost*: this cost ideally should measure the actual amount of money that has been spent on the item to make it available for usage. It is commonly called "book value". For a merchant, it is the price paid for the item to the supplier, plus any cost incurred to handle it and making it ready to sell. In the case of production management, it is generally the cost occurred to manufacture the single item; this takes into account the costs of labour, rent, instruments, R&D, etc. For a producer, this value is usually more difficult to determine.
- *The cost of carrying items in inventory*: this cost represents the cost of the money invested, the expenses incurred in running a warehouse, the costs of special storage requirements, handling and counting costs, deterioration of stock, damage, theft, obsolescence, insurance, and taxes. The cost is usually defined for the single item and for a specific amount

of time; for example, the cost of keeping one item in the inventory for a month.

- *The ordering or setup cost:* it models the fixed cost associated with a replenishment. In the case of a buyer, it is called an ordering cost. This includes the cost of placing the order, shipping cost, managing the reception of the order, inspection of the items, follow up on unexpected situations and handling of vendor invoices. For a production system, it is called setup cost. It represents the cost of starting the production of a particular item. It can include the wages of a technician who has to reconfigure the machine, the cost to gather the material for the production and the cost of a period of time during which the facility is producing at a slower speed while the equipment is fine-tuned and the operator adjusts to the new item ("learning effect"). Notice that during the setup and learning period, opportunity costs are, in effect, incurred because production time on the equipment is being lost during which some other item could be manufactured. The ordering cost includes all these components because they are the result of a decision to place an order.
- *The cost of insufficient capacity in the short run*: this represents the costs relative to a stockout, a stockout is when the inventory on-hand can not satisfy the demand. They can also consider the costs of avoiding a stockout. For a merchant, they include emergency shipments substitution of a less-profitable item, lost of sales, cost of the reduction the customer service and loss of reputation. Some of these costs can be estimated reasonably well; others are more nebulous. In a production situation, they include the expenses that result from changing over equipment to run emergency orders and the attendant costs of expediting, rescheduling, split lots.
- *System control cost*: this represents the cost of the operation of the particular decision system selected. These include the costs of data acquisition, data storage and maintenance, and computation. In many real-world applications, there is no real-time tracking of the inventory level. An inventory review (also known as stock-taking) has to be carried out to assess the inventory level. The cancellation costs are included in this category. For a merchant, these are the cost of cancelling a contract or a scheduled order. For a producer, it is the cost to cancel a scheduled

production. Besides, there are less-tangible costs of human interpretation of results, training, alienation of employees, and so on.

The model used in this work takes into account a cost parameter for each of the five cost factors presented in this Section. An order's cost comprises a fixed ordering cost, that is charged regardless of the order size, and a linear ordering cost charged for each item in the order. A linear holding cost is charged for every item in the inventory at the end of each period. The cost of insufficient capacity is represented by a penalty cost of having a stockout; this cost is linearly dependent on the size of the stockout. Finally, a fixed cost for assessing the inventory level is charged. The reduction of the expected cost is the goal of all the techniques presented in this thesis.

2.1.5 Lead time

From [Sil81]:

> We define the *replenishment lead time*, as the time that elapses from the moment at which it is decided to place an order until it is physically on the shelf ready to satisfy customers' demands.

From a merchant perspective, the lead time is composed of five different components: administrative time at the stocking point, transit time to the supplier, time at the supplier, transit time back to the stocking point and the time that elapses from the receipt until the items are available on the shelf. Herein, without loss of generality, we assume a null lead time. So that the inventory level updates instantly after the order is placed. This is a common assumption in the lot sizing literature [Sca59, BM99, TK04].

2.1.6 Demand backlogging vs lost sales

Define what happens when the inventory level is lower than the demand is crucial in an inventory control system. There are two main ways to model this situation:

- *Backordering*: the part of the demand that cannot be satisfied is carried on to the next period; this process continues until an adequate order arrives. The backlogging of the demand is generally associated with a penalty cost or with a service level to satisfy; we introduce these concepts in the

next section. Complete backordering is a common assumption in classic inventory control [SPT16, Zip00, Sca59] and in the lot sizing literature, e.g. [RTHP08, Tar96, DSKTR16]. We can find this situation in a quasi-monopoly market; for example, government organisation and exclusive dealerships.

- *Lost sales*: all the demand that can not be satisfied with the inventory on-hand is lost. The customer satisfies his/her needs from another supplier or buys a substitute product. This model is most common in the retail sector. Lost sales models have been studied in the literature with combinations of different settings, e.g. [Zip08, BV12b, HJMR09]. For a complete literature survey on such systems, we refer to [BV12a].

Most of the inventory systems model only one of these two extremes. However, in practical situations, there is a combination of these two approaches. Gruen et al. [GCB02] show that, in case of a stock out, 24% of the customers delay the purchase, 15% of them wait for the item to be in the shelf again and 9% do not buy the product at all. The remaining 76% of them either buy a different product (45%) or visit a different store (31%). These values are strongly dependent by the type of product, but they give an idea of the variability in the demand behaviour. Nevertheless, most of the models are a reasonable approximation, and they do not vary much using the two systems. This is due to the high customer service level that is generally considered. Most of the models try to make the stockout events as infrequent as reasonably possible.

In this thesis, we consider the complete backordering of the exceeding demand. This assumption is more frequently used in the literature.

2.1.7 Stock levels

According to [SPT16], we define the *on-hand stock* (or on-hand inventory) as the stock that is physically on the shelf. This value can never be negative. This inventory is used to satisfy the demand when the demand is higher than the on-hand inventory a stockout occurs.

The *net stock* (or net inventory) is the on-hand stock minus the backorders. It can become negative in the case of stockouts. Finally, the *inventory position* (or inventory level) is the net stock plus the quantity ordered and minus the committed demand. In this thesis, due to the null lead time and the absence

of delayed demand, the *net stock* is equivalent to the *inventory position* and we use them interchangeably.

2.1.8 Reviews

The inventory control systems that we analyse herein operates with a finite time horizon. The horizon is divided into periods; all the decisions have to be taken at the beginning of each period. This is a normal assumption in many practical applications. For example, in a retail shop, the inventory assessment or the decision of placing an order can be made every evening after the closing. Alternatively, even if the order can be placed at any point of the day, they will be processed by the supplier together the next morning. Most of the literature in time-varying stochastic demand follow the periodic finite time horizon model.

The review interval is the period of time that occurs between two inventory assessments (or review moments). The literature presents two alternatives :

- *Continuous review*: the status of the inventory is always known. The system updates the inventory level immediately at every transaction (supplier delivery, receipt, use, demand, etc.). It is possible to place an order at every moment. Most of the time, it is not possible to know the exact level of the inventory. Even with the new technologies (POS data collection systems, RFID), many unpredictable factors affect the inventory level in an unpredictable way; for example, faulty or expired products, shoplifting, products damaged during the stockage. The main disadvantage of a continuous review system is that is generally more expensive in terms of reviewing costs. However, since the uncertainty on the inventory level is lower, it generally leads to the same level of customer service with less safety stock. In case of a discrete periodic time horizon, we consider as continuous review policies the ones that assess the inventory level at the beginning (or end) of each period. Section 3.4.3 presents a policy with continuous review in a periodic time horizon.
- *Periodic review*: the inventory on-hand is determined only at fixed periods. This can be due to different reasons; for example, the possibility of reviewing the inventory only at fixed time slots (for a soda vending machine is when the operator visit it) or due to the high review costs. An early schedule of the replenishments moments offers an advantage since the transportation mode can be planned more efficiently and better

prices can be dealt with the suppliers. A rhythmic, rather than a random, pattern is usually preferred. However, the flexibility of the review interval is important in the management of items with time-varying demand. A clear example is seasonal items, in periods in which the demand of an item is low or almost deterministic, the interval between reviews can be longer and lead to savings; e.g. it might be non-convenient to check the inventory of ski boots weekly during the summer. When the demand has a higher variability, it is important to assess it more often since the inventory level change faster. In Sections 3.4.4 and 3.4.5 we will show two polices with periodic review.

2.1.9 Penalty cost vs Service level

An inventory management system has to avoid stockouts. As seen in Section 2.1.6, they lead to lost sales, reputation damage and penalties for not fulfilling supply contracts. While modelling a real-world problem, there are two main ways to avoid stockouts:

- *Service level*: the probability of satisfying the demand with the on-hand inventory has to be higher than the service level in every period. This can be seen as the probability of not having a stockout; it is a conventional service level measure used in practice. This limitation is used particularly in situations in which the stockouts are costly regardless of their magnitude or duration; for example, in production systems that face costly stoppages due to stockouts. The traditional way to model it is to add a constraint to satisfy. The service level generally assumes a high value, $> 95\%$. This has been widely studied in the literature [TK04, TDÖR11, RTHP11] and is popular in practice [PSS10, TBS10]. A recent comparison of this type of inventory control policy is available in [Bij14].
- *Penalty cost*: not satisfying the demand of a customer is costly for a company. In this case, a penalty cost occurs with every stockout. The cost is generally dependent on the size and length of a stockout. This model is preferred in retail applications. The models using the penalty costs are also known as full-cost models since the model's objective includes the penalty cost. The main drawback of this is that these shortage costs are usually difficult to asses. In particular, the cost of losing customer goodwill

that will lead to future lost sales. However, this is a common assumption in most of the classic inventory control literature [Sca59, ZF91, BM99] and it is widely used [ÖDT12, XRMBT18]. A comparison of some techniques and their behaviour with receding horizon can be found in [DSRKT19].

There is a strong relationship between the two of them. In the newsvendor problem, a single period stochastic inventory control problem, the penalty cost divided by the penalty plus holding cost is equal to the service level. As in Scarf's setting, we use a per unit penalty cost.

2.1.10 Decisions

Decisions in an organisation are a hierarchy. Starting from a long-range strategic planning down to short-range operational control decisions. In inventory management the hierarchy can be conceptualized as:

- At the highest level, one chooses a particular type of control system settings. For example, the definition of the stock level, continuous or period review, backorders or lost sales.
- At the middle level, one selects the values of the specific parameters; e.g. fixing the value of the service level.
- Finally, one operationalises the system, including data collection, calculations, reporting of results, and so on [Bro82].

The solutions presented in this work are designed to be implemented in scheduling software. These decision helping tools are considered a type of physical aid for the management. The computer can handle vast amount of data faster than the manager could ever dream of doing manually. There are three types of strategies that involve modellings. They differ on the different level of abstraction and mathematical formulation

- Detailed modelling and analytic selection of the values of a limited number of decision variables.
- Broader-scope modelling, with less attempt at optimisation.
- Minimisation of inventories with very little in the way of associated mathematical models.

We focus on the first one. The strategy here is to develop a mathematical

model that permits the selection of the values of a limited set of variables so that some reasonable measure of effectiveness can be optimised. In general, a mathematical model may permit a deductive solution, an iterative solution, or a solution by some form of trial-and-error procedure. In this work, we propose different algorithms of the last two categories; one of them is a hybrid between these two. The goal stochastic inventory control system is to address the following three points:

- When the inventory should be reviewed to assess the net stock.
- When an order should be placed.
- The size of each order.

These issues are particularly relevant under the assumption of stochastic demand. The first two are trivial in conditions of deterministic demand, the inventory level is always known, and the order can be placed when the inventory level goes under a safety stock level. While the third can be addressed with well-known techniques presented in [SPT16]. In the case of probabilistic demand, the first issue is crucial because assessing the inventory level is costly. On the other hand, the longer is the period between reviews, the higher the uncertainty of the inventory level is. The system has to protect itself from the consequences of unforeseen demand to provide the desired customer service. The second problem involves trade-offs between multiple factors. Frequently orders are more resilient to demand uncertainty, but costly due to fixed ordering charges. Moreover, less frequent orders implicate bigger orders and a higher average net stock, which increases the cost of carrying extra stock. The orders' sizes strongly depend on their frequency and on the type on service level or penalty cost that occurs in the case of a stockout.

2.1.11 Inventory Control Policy

An inventory control policy aims to determine replenishment quantities and their timing to minimise the total inventory cost. We define as review moment, or review period, a period in which a stock-taking occurs and the order can be placed. Inventory items can be organised in three classes according to an inventory categorization called ABC analysis. This technique is widely used out materials management. Each type of element in an inventory is classified as A, B or C depending on their contribution to the total sales and the sale

frequency. The policies presented in this work are generally used for the so-called B items, which comprise the majority of items in a typical inventory situation. The use of a reasonably sophisticated control system usually leads to relevant savings. On the other hand, these savings are not as high as those for A items, items with the highest amount of sales. This allows the deployment of a management-by-exception control system; a tool reasonably sophisticated that requires human intervention on a relatively infrequent basis. Alternatively, the system can simply provide decision support to the management. The algorithms proposed in this work are the basis of such software and they can be modified to represent the real problem with higher fidelity. In the previous sections, we presented how the different modelling techniques can be combine to represent a wide variety of problems encountered by practitioners.

Example of real-world systems that can be modelled with the setting similar to the one presented in this chapter are: the management of liquefied natural gas [ABKV20], inventory management in a hospital for medical equipment [BV12a] and for blood supply [GC15] or dealing with demand seasonality in a retail environment [EHVW14].

2.2 Mathematical model

In this section, we introduce the notation, and we provide a mathematical description of the problem. Bookbinder and Tan [BT88] introduce the original stochastic programming formulation for the non-stationary stochastic lot-sizing problem. We consider the inventory control problem over a T-period planning horizon. Demands d_t in each period t are independent random variables with known probability distributions $g_t(\cdot)$ and cumulative distribution function $G_t(\cdot)$. Backlogging of excess demand is assumed, so if the demand in a period exceeds on-hand inventory, the rest of the demand is carried to the next period; a linear penalty cost b is charged for each unit of back-ordered demand at the end of a period.

We define as Q_t the discrete quantity of the order placed in period t, negative orders (returns) are not allowed. Without loss of generality, we assume that orders are placed at the beginning of each period and that the lead time is zero. Binary variable δ_t takes value 1 if an order is placed on period t. These two sets represent the decision variables of the problem.

Ordering costs are represented by a fixed value K and a per unit cost c. At the end of each period, a linear holding cost h is charged for every unit carried from one period to the next. All the cost factors are represented by continuous parameters.

We denote the closing inventory level for each period by I_t, and the given initial inventory level by I_0. When a stockout occurs, all the demand is backlogged (negative inventory level) and satisfied as soon as an adequate supply arrives. The original model is not considering a penalty cost for unsatisfied demand. However, there is a service level constraint enforcing a non-stockout probability of at least α at the end of each period. We assume a high value of α (generally over 90%) in order to incorporate management's perception of the stockout costs and ignore possible penalty costs. The objective of the model is to minimise the expected total cost $E[TC]$.

The mathematical formulation of the problem proposed in [BT88]:

$$\min_{E[TC]} = \int_{d_1}\int_{d_2}\cdots\int_{d_T}\sum_{t=1}^{T} K\delta_t + hmax(I_t,0) + cQ_t \times g_1(d_1)g_2(d_2)\ldots g_T(d_T) \tag{2.1a}$$

s.t. for $t = 1, \ldots T$

$$I_t = I_0 + \sum_{i=1}^{t}(Q_i - d_i) \tag{2.1b}$$

$$\delta_t = \begin{cases} 1 & \text{if } Q_t > 0 \\ 0 & otherwise \end{cases} \tag{2.1c}$$

$$Pr\{I_t \geq 0\} \geq \alpha \tag{2.1d}$$

$$Q_t \geq 0, \delta_t \in \{0,1\} \tag{2.1e}$$

where Equation 2.1a models the total inventory cost for the planning horizon, Equation 2.1b fixes the closing inventory levels and Equation 2.1c guarantees that the order quantity is null if an order is not placed. Finally, Equation 2.1d is known in the literature as "α service level" constraint.

This model can be adapted to use the penalty cost scheme instead of the service level. If the demand in a period exceeds on-hand inventory, the rest of the demand is carried to the next period; a linear penalty cost b is charged for any back-ordered unit at the end of a period. As done in [TK06], we need to remove the service level constraint 2.1d and add the penalty cost to the

objective function. The resulting model is:

$$\min_{E[TC]} = \int_{d_1}\int_{d_2}\cdots\int_{d_T}\sum_{t=1}^{T} K\delta_t + hmax(I_t,0) + bmax(-I_t,0) + cQ_t\times \\ g_1(d_1)g_2(d_2)\dots g_T(d_T) \tag{2.2a}$$

s.t. for $t = 1,\dots T$

$$I_t = I_0 + \sum_{i=1}^{t}(Q_i - d_i) \tag{2.2b}$$

$$\delta_t = \begin{cases} 1 & \text{if } Q_t > 0 \\ 0 & otherwise \end{cases} \tag{2.2c}$$

$$Q_t \geq 0, \delta_t \in \{0,1\} \tag{2.2d}$$

a peculiarity of this work is the modelling of an inventory review cost. This can represent the physical cost of assessing the inventory level, or a penalty cost faced in case of an order cancellation. We consider an inventory review cost W. Binary variable γ_t takes value 1 if the inventory is reviewed at period t. Before an order is placed, an inventory review must occur. The resulting model is:

$$\min_{E[TC]} = \int_{d_1}\int_{d_2}\cdots\int_{d_T}\sum_{t=1}^{T} K\delta_t + W\gamma_t + hmax(I_t,0) + bmax(-I_t,0) + cQ_t\times \\ g_1(d_1)g_2(d_2)\dots g_T(d_T) \tag{2.3a}$$

s.t. for $t = 1,\dots T$

$$I_t = I_0 + \sum_{i=1}^{t}(Q_i - d_i) \tag{2.3b}$$

$$\delta_t = \begin{cases} 1 & \text{if } Q_t > 0 \\ 0 & otherwise \end{cases} \tag{2.3c}$$

$$\gamma_t \geq \delta_t \tag{2.3d}$$

$$Q_t \geq 0, \delta_t \in \{0,1\} \tag{2.3e}$$

Equation 2.3d ensures that an order is placed only after reviewing the inventory. The decision variables are, for every period t:

- γ_t: if the inventory has to be assessed.
- δ_t: if an order has to be placed.
- Q_t: the quantity of the order.

An inventory control policy aims to fix the values of these variables. This is in agreement with Section 2.1.10. Model 2.3 is the mathematical representation of the problem tackled in this thesis. We use this notation and symbols for the rest of this work; a full list of symbols is available in Appendix 8.1.

Chapter 3

Related Work

In this chapter, we define the basic concepts and policies used in this thesis, and we survey the relevant literature. The goal is to provide a common notation and a picture of the state-of-the-art in lot sizing with non-stationary stochastic demand.

In Section 3.1, we introduce dynamic programming; a successful algorithm design approach used in a lot of different fields. All the algorithms that we introduce in this work use stochastic dynamic programming algorithms; the branch of DP that deals with problems involving uncertainty. Section 3.2 presents the branch and bound approach. The solution presented in Chapter 6 uses a BnB technique to explore the search space. Constraint programming, a paradigm that offers higher expressivity compared to mixed-integer programming, is introduced in Section 3.3.

In Section 3.4, we define an inventory control policy, we present some possible applications, and we briefly survey reviews covering aspects of inventory control not tackled in this work. We illustrate a widely use inventory control strategy classification in Section 3.4.1, and we extend it to consider the review cost. In the rest of the chapter, the different policies are presented, and we survey the relevant literature. This survey focuses on techniques to solve the stochastic lot sizing problem presented in the previous chapter.

Finally, Section 3.5 summarises the chapter and highlights the shortcomings in the literature that we aim to cover in this thesis.

3.1 Dynamic Programming

In this section, we introduce dynamic programming (DP) and stochastic dynamic programming (SDP). We aim to define a structured way to describe a DP algorithm. This section is inspired by [Whi69, Pow07, Put14].

DP is a widely used optimisation procedure originated by the extensive work of Richard Bellman, e.g. [Bel66]. DP approaches solve problems by breaking them down into smaller sub-problems recursively. The optimal solution can be computed using the optimal solutions of the sub-problems in a recursive way; this is called optimal substructure. The sub-problems' solutions are stored to avoid their recomputation. DP has a great variety of possible implementations and applications. It is used in DNA sequence alignment, scheduling of maintenance, knapsack and packing problems and parsing of context-free languages. Inventory control frequently uses DP, the lot sizing problem is a standard example in DP books, and innovative DP techniques are part of the state-of-the-art algorithms, e.g. [Sca59, Igl61, Tar96, Xia19].

DP is generally presented by examples; lot sizing is one of the most used for this purpose. This problem is one of the numerous examples of a sequential decision-making model. Such a model is characterised by the following key elements:

- A set of decision stages.
- A set of system states.
- A set of available actions.
- A set of immediate cost dependent on a state and action.
- A set of transaction that connects a couple state and action to a different state.

In the case of a stochastic problem, the last element is replaced by a set of transaction probabilities. We assume that all these elements are known by the decision-maker. At each decision stage, the system state contains all the necessary information for selecting an action from the set of available actions. Each pair of state and action is associated with a cost and a probability of the system to be transferred to a different state. The new state has to be part of a future decision stage. The decision-maker accumulates a sequence of costs through time.

A policy determines which action has to choose at each future stage. Deploying a policy generates a set of costs, each associated with a decision. The sum of all of them is the total cost associated with the policy. A policy with the minimum cost (or maximum reward) is defined to be an optimal policy; the set of optimal policies might not be a singleton.

DP approaches can be deployed in a variety of ways. However, a common structure is present in most of the algorithms. In line with [XRMBT19], we describe the DP algorithms according to the following structure:

1. **Stage**. Decisions are taken in moments referred to as decision epochs. This thesis' algorithms deal with discrete-time problems; i.e. the time horizon is divided into stages or periods. The stage concept guarantees a hierarchy of the decisions. In this setting, decisions are taken in all the stages.

2. **States**. At each stage, the system is represented by a state. A state denotes a sub-problem of the overall problem. The states can be continuous or discrete; herein we deal with discrete states. The state contains all the information necessary to make future decisions. An optimal cost can be associated with a state, that is the minimum cost of the optimal policy for a system starting from that state. Each state is associated with a set of possible actions. The initial state is a particular state that represents the system at the beginning of the time horizon before any decision occurs.

3. **Action**. At a decision epoch, the decision-maker observes the state of the system and s/he chooses an action form a set of allowable actions. The set of allowable actions depends on the state of the system. Actions can be chosen deterministically or randomly, according to a probability distribution. The action can involve multiple decision variables at the same time. An immediate cost and a transition probability are associated to a pair action, state. A policy defines which action is selected at each stage. In inventory control is generally associated with the decision of placing an order.

4. **Immediate cost**. As a result of choosing an action in a state:

    - an immediate cost is charged to the decision-maker. In a maximisation problem, this amount is called reward or profit.

    - the system evolves into a different state. The state must be part of

a later stage of the problem. The transition can be deterministic or stochastic. In the first one, the decision-maker knows in which state the system will evolve after his/her decision. In the stochastic one, the future state is determined by a probability distribution.

The immediate cost is a real function of the state and the action. It represents the cost of making that decision; in a stochastic problem, it is the expected cost. This function estimates the costs that occur until the next decision epoch. In our mathematical formulation, the immediate cost comprises a set of practical cost factors.

5. **Objective function**. Also known as the functional equation, see [Bel54]. It is a recursive function that computes the expected cost of a specific state considering the cost of future stages. It incorporates the transition probability of moving to a new state given the current state and the possible action. This function evaluates the possible actions, and it selects the optimal one. For a minimisation problem the functional equation C for the state x_t on stage t can be modelled as:

$$C(x_t) = \min_{a_t}(f(x_t, a_t) + \alpha C(x_{t+1})) \tag{3.1}$$

where a_t is a possible action of state x_t, $f(x_t, a_t)$ is the immediate cost of choosing action a_t in x_t. We denote as T the transformation operator, $x_{t+1} = T(x_t, a_t)$ where x_{t+1} is the state in which the system evolves if the action a_t is chosen at state x_t. The fundamental characteristic of DP is the use of the functional equation to relate the expected cost of a stage to the cost of the successive ones. The value of the functional equation computed in the intial state represents the expected cost of the policy over the planning horizon.

A prerequisite for the application of DP is that the *Principle of Optimality* should hold. Bellman states his principle as follows:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

It perfectly describes the principle of solving a problem by breaking it into subproblems and solving them to optimality recursively. This does not mean that past events do not influence future decisions at any stage; past decisions

lead to a specific system represented by the state description.

DP's algorithms performances strongly depend on: the optimisation of the implementation, the programming language used, the data structure used for the memoisation. The implementation of the recursion plays an important role in the computing-efficiency, removing a single instruction on a block that is repeated a high amount of times can lead to measurable improvement. We developed all our solutions using the same programming language (Python), using the same data structures, and we optimised them to the best of our capacities. For this reason, we believe that the scalability experiments available in the next chapters measure the differences in the complexity of the algorithms and not differences in the implementation.

3.1.1 Examples

In this section, we present some of DP's possible applications. An exhaustive definition of DP that covers all its possible applications is complex to achieve. DP books describe the approach alongside with many examples for the sake of clarity, e.g. [Bel66, Whi69]. We aim to do the same in this section.

Fibonacci numbers

A clear example of DP memoisation is the computation of the Fibonacci numbers. Even if it is a straightforward example, it has been used many times.

To find the n^{th} Fibonacci number we use the following approach:

$$X_1 = 1 \tag{3.2}$$

$$x_2 = 1 \tag{3.3}$$

and the recursive equation:

$$x_i = x_{i-1} + x_{i-2} i = 3, \ldots, n \tag{3.4}$$

A simple recursive solution has an exponential complexity; the same computations are repeated multiple times. However, storing the values of x_i in an array and using the recursive equation as a functional equation avoids all the recomputations.

Change problem

A widely used and less obvious example for DP is the change problem. This problem's goal is to find the minimum number of coins adding up to a given total of Y. The set if available coin denominations is $c_1 \dots c_D$. The problem can be modelled in CP using the following mathematical formulation:

$$\min \sum_i x_i \tag{3.5}$$

$$\text{s.t. } \sum_i x_i c_i = Y \tag{3.6}$$

where the x_i are finite domain variables denoting the number of coins from denomination i. This formulation is solved through search on the x_i domains. However, the problem can be solved in pseudo-polynomial time by DP. We can describe the algorithm with the structure presented before:

1. **Stage**. A stage represent a given total i for which a set of coins that adds up to has to be computed. The total number of stages is $Y + 1$.
2. **States**. This problem has a single state for each stage, since i is the only information needed to define the system status.
3. **Action**. An action is represented by the using of a coin of denomination c_i when the change left to give is i.
4. **Immediate cost**. Each action has an immediate cost of 1.
5. **Objective function**. Let N_i be the minimum number of coins of which the sum is i. The functional equation can be defined as:

$$N_i = \min_{c_i}(N_{i-c_i} + 1) \tag{3.7}$$

where, the boundary condition is $N_0 = 0$. The recurrence relation refers to states $N_{-1} \dots n_{1-M}$. We need to add further base cases:

$$n_i = Y \qquad i = -1, \dots, 1 - M \tag{3.8}$$

where $M = \max_i(c_i)$. Y is used here as a large value that will never be chosen by the DP min function.

3.2 Branch and bound

Branch and bound (BnB) [Cla99] is a paradigm used to solve discrete and combinatorial problems. It is an implicit method that organises the state space as a rooted search tree. Each leaf of the tree represents a possible assignment of the problem variables. An internal node represents a partial solution of the problem; the sub-tree rooted in it contains all the solutions that have that partial assignment. The technique is based on the computation of upper and lower bounds for a partial solution. A common structure of a BnB algorithm comprises three steps:

- **Branching**. A variable is fixed to a value. The problem is divided into two subproblems. In the first, the variable is considered a fixed parameter with the value mentioned above. In the second, the value is removed from the possible assignments of the variable. The two subproblems are analysed sequentially.
- **Bounding**. A lower/upper bound of the objective function is computed for the subproblem.
- **Pruning**. If the bounds prove the non-optimality (or the unfeasibility) of the solutions containing that value assignement, the branch of the tree is pruned. The of the tree proceeds with a different subproblem.

We use BnB in Chapter 5 to explore different possible replenishment plans, this allows to avoid recomputations and to prune a consistent number of non-optimal plans without computing them.

3.3 Constraint programming

Constraint Programming (CP) is a paradigm for solving combinatorial search problems. We aim to give a brief introduction to this paradigm. This section is inspired by [RVBW06].

The basic idea of CP is that the user states the constraints of a problem and a general-purpose solver is used to solve it. According to Freuder [Fre97]:

> Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.

This approach has some similarities to integer linear programming. However, CP offers a higher expressivity and variety of constraints. CP relies on a wide range of techniques from artificial intelligence, computer science, databases and operations research. It is applied to many domains; e.g. shift and production scheduling, vehicle routing, facility location and inventory control.

A constraint satisfaction problem (CSP) consists of decision variables and constraints. A domain set is associated with every decision variable. The goal of a solver is to find an assignment of all the variables that satisfy all the constraints. A constraint solver searches this assignment in the solution space created by the domains of the variable. The search can be systematic, where backtracking and branch and bound solutions are used or rely on forms of local search. In the systematic search, the solver branches by assigning to a decision variable a value from its domain. The solver then uses constraint propagation; an inference technique that consists in propagating the information to neighbouring constraints to reduce the domain of the decision variables not yet fixed. Its goal is it to reduce the part of the search space that needs to be visited. If due to propagation, the domain of a non-fixed variable becomes empty, it means that the partial assignment is unfeasible and a backtrack occurs. If the problem involves the optimisation of an objective function, it is classified as a constraint optimisation problem.

To deal with uncertainty, Walsh [Wal02] introduces stochastic constraint programming, a combination of CP, stochastic integer programming and stochastic satisfiability. An analysis of CP's state of the art can be found in [Fre18]. For a more in-depth discussion on the topic, we refer the reader to [RVBW06].

In Chapter 7, we present a technique to encode DP into a CP model seamlessly. Using this encoding is possible to delegate the search of the optimal replenishment plan to the CP solver.

3.3.1 Constraint Programming and Dynamic Programming

In this thesis, we use a hybridisation of CP and DP. In the literature, there are several interesting connections between them. For example:

- DP is used to implement several global constraints within CP solvers. Trick [Tri03] uses DP to propagate a constraint. For example, DP can be found in grammar constraints [QW06, KS08] and knapsack constraints

[MSVH08]. Quimper [QW07] uses it to decompose a grammar constraint into AND/OR clauses and solve it with a SAT solver. The global constraint catalogue includes a specific tag for DP approaches [BCR12].

- Focacci and Milano [FM01] use CP to model a DP relaxed version of the TSP after a state space reduction.
- The DP feature of solving each subproblem once only is used in CP [CS09], in Constraint Logic Programming languages including Picat [ZKF15] and PRISM [ZKS10]. In logic programming [War92] uses it to remember the results of sub-tree searches using memoisation. These techniques can strongly improve search performances. Pruning on the search tree can be achieved by analysing the dominance between subproblems [CDLBS12].
- DP approaches are widely used in binary decision diagrams and multi-value decision diagrams. Bergman et al. [BCVHH16] use them to exploit recursive structure on relaxed binary decision diagrams. Other approaches use these techniques to develop CP constraints, with a focus on their propagation [HVHH10].

Many DP solutions can be modelled from the prototypical viewpoint of finding the shortest paths in a rooted direct acyclic graph. The vertices of the DAG represent the states of the search space. The edges represent the decisions, and their weights are the cost of taking the corresponding decisions. For example, this is the case of the knapsack problem [Mar90] or the deterministic lot sizing problem [EM87].

Martin [Mar87] introduces a variable redefinition technique that encodes these DP solutions in MIP. In this encoding, binary variable model the edges of the graph. The variables take value 1 if the corresponding decision is taken. Each node is represented by a constraint that balances its flow; the number of entering arcs with value 1 is equal to the number leaving arcs with the same value. The sum of the arcs leaving the source has to be one; the same is valid for the arcs entering in the sink. This encoding can be deployed in CP as well since CP provides a higher expressivity compared to MIP. Martin and Eppen [EM87] deploy the new technique to solve the multi-item capacitated lot-sizing problem.

This formulation is inadequate for more complex discrete optimisation problems, where a decision involves composing two or more partial solution ele-

ment into a single one. It is extended in [MRC90] to overcome that problem, allowing it to encode all DPs that can be modelled as directed acyclic hypergraphs.

In many problems, the resulting formulations are more compact and with better bounds than a model based on the problem description; e.g. [MM10, BSH18]. Raffensperger [Raf99] presents an interesting tutorial with practical applications. This technique is also known as the flow formulation of DP into MIP. A similar flow formulation is present in MiniZinc in [BBB+08] to solve the shortest path problem.

Quimper and Walsh [QW07] use a similar approach in their decomposition of the grammar constraint. They consider the CYK parser, a popular DP-based algorithm used for the parsing of context-free grammars. They remodel the problem as a rooted DAG and decompose it into clausal normal form. This work differs from the DPE because: they transform the path into SAT instead of CP; it is an implementation of a global constraint, not a general technique for DP in CP; their variables represent DAG edges, not DP states.

Another successful combination of DP and CP is in Binary Decision Diagrams [Ake78]. In this case, DP is used here to speed up the computations by reducing the state space and obtaining an approximate one. There are many examples of this being used to develop constraints and propagators [BCvH14] or as part of the solution [HOOS09].

3.4 Stochastic Lot Sizing Problem and Uncertainty Strategies

The newsvendor problem [Edg88] is the simplest case of single-period single-item stochastic lot sizing. The problem aims to find the quantity of an item to order to satisfy a single demand of which the probability distribution is known. Wagner and Whitin's work [WW58] is considered to be the first work in multiperiod stochastic lot sizing. They present a heuristic to solve the problem. Their paper has been included in the ten most influential articles in the first 50 years of *Management Science* [Hop04]. Scarf [Sca59] proves the optimality of the (s, S) policy under the assumption that ordering and holding costs are linear.

As described in the previous chapter, this work focuses on the non-stationary single-item single-location inventory control under stochastic demand. In this section, we survey the relevant literature with a focus on algorithms for the computation of replenishment policy parameters based on Bookbinder and Tan model [BT88]. A recent literature review on this problem is available in [MRA19]. Stochastic non-stationary demand is more common than its stationary deterministic version in real-world settings. This is due to seasonal patterns, life cycles and trend. Yet, stationary policies are still widely used in industrial applications. Tunc et al. [TKTE11] estimate the cost of applying a stationary policy when the demand is stochastic and varies over time. They compare the optimal (s, S) policy with the best possible stationary policy. The approximation is efficient only when the uncertainty of the demand and the fix ordering cost are high, and the penalty one is low.

Inventory control policy literature is wide; we restrict this survey to the stochastic inventory control problem. For a broader view of the inventory lot sizing problem, [Sil08] includes an overview of inventory management. Bijvank [BV11] surveys the literature that assumes that the excess of demand is lost in case of a stockout. Glock et al. [GGR14] provides a recent generic literature review on the field. A survey on the deterministic single-item lot sizing problem is available in [BDPNN06]. The same authors in [BADPN17] present an updated survey that involves some stochastic solutions. Finally, a review of review papers on the topic is available in [BGJK15].

These algorithms can be applied to a wide range of theoretical and real-world problems. The mathematical formulation of the lot sizing problem can be easily adapted to solve other practical logistic applications. Bruno et al. [BGP14] describe adaptation processes to model bus terminal schedule optimisation, cross-docking operations and check-in service as lot sizing problems.

3.4.1 Bookbinder and Tan classification

In the traditional lot sizing problem, the review cost is not included; the decisions involve only the order timing and size. For this configuration, Bookbinder and Tan [BT88] discuss three main control strategies that can be adopted:

- *Static uncertainty*: the decision-maker fixes the timing and size of each order at the beginning of the time horizon before any demand is observed.

In this policy, the values of δ_t and Q_t are fixed at the beginning of the time horizon. This approach is also called (R, Q) policy, see Section 3.4.2.

- *Dynamic uncertainty*: the decision-maker observes the current inventory position at the beginning of each time period. After the observation, s/he decides if to place an order and its size. The values of δ_t and Q_t are fixed at the beginning of period t. This approach is also called (s, S) policy, Section 3.4.3.
- *Static-Dynamic uncertainty*: the decision-maker fixes the complete replenishment schedule at the beginning of the time horizon; whereas order quantities are determined at the moment the order is placed. The values of δ_t are fixed before observing the demand. The order quantity Q_t is fixed at the beginning of period t. This approach is also called (R, S) policy, Section 3.4.4.

For the stochastic lot sizing model with no review cost, Scarf [Sca59] proves the optimality of the dynamic uncertainty strategy. Tunc et al. [TKTE11] provide a numerical analysis of the cost performances of these strategies. The authors show that the static-dynamic strategy is very competitive, while the static one performs poorly. The static strategies have the advantage of a fixed replenishment schedule, that is appealing from the management point of view. A recent study [DSRKT19] compares these strategies and their receding horizon versions [KH06].

All these works do not model the review cost. The introduction of the review costs makes the dynamic uncertainty strategy non-cost-optimal since it has to review the inventory at every period. For example, in a situation with a low holding cost, low demand and high review cost the (s, S) policy is hardly optimal.

We can extend the same classification framework to fit the review cost. The decision-maker has three sets of variables to fix: review moments, replenishments times and their size. The difference between dynamic and static is when the decision-making occurs; if a decision is taken before observing any demand is static, otherwise is dynamic. Extending the Bookbinder and Tan classification to the review planning:

- *Static uncertainty*: the decision-maker fixes all the decision variables at the beginning of the planning horizon before any demand is observed. Intuitively, this approach is equivalent to the (R, Q) policy. Since no de-

cision can be taken after observing the demand, and the order quantities are fixed the review of the inventory level is useless.

- *Dynamic uncertainty*: the decision-maker decides at the beginning of each period if reviewing the inventory or not. If a stock-taking occurs, the decision-maker can decide to place an order and its size. Decision variables γ_t are determined at the beginning of period t, the values of δ_t and Q_t are decided after the review. In classical inventory control, no policy models this strategy.
- *Static-Dynamic uncertainty*: the decision-maker fixes the complete review schedule at the beginning of the time horizon; whereas order quantities are determined at the moment the order is placed. The values of γ_t are fixed before the demand's observation. At the beginning of a review period t ($\gamma_t = 1$) δ_t and Q_t are set. This approach is known as (R, s, S) policy, Section 3.4.5.

No strategy is proved to be optimal for this cost configuration. The next sections describe these policies, along with a brief discussion on their advantages and disadvantages. We then survey the respective literature. These policies take their name from their parameters fixed at the beginning of the time horizon.

3.4.2 Periodic-Review, Order-Quantity (R, Q) System

A decision-maker adopting this policy takes all the decisions before observing the realisations of the demand. It is customary in the literature to denote as R the timing, where R is the number of periods between consecutive replenishments. While Q denotes the quantity of the order. For this reason, the policy is denoted as (R, Q) policy. Neither time nor quantity of the orders is decided in response to the observed demand. Figure 3.4.2 shows an example of inventory system managed with a (R, Q) policy.

Being unable to react to additional information obtained by the observation of the demand makes this policy less cost-efficient compared to the more dynamic ones [TKTE11]. However, this strategy has organisational advantages when considerable preparation time is needed to place an order. It provides a completely stable production environment, that is appealing in an industrial environment with a low degree of flexibility, and it generally allows to negotiate better prices with suppliers. When the demand is non-stationary, the values of

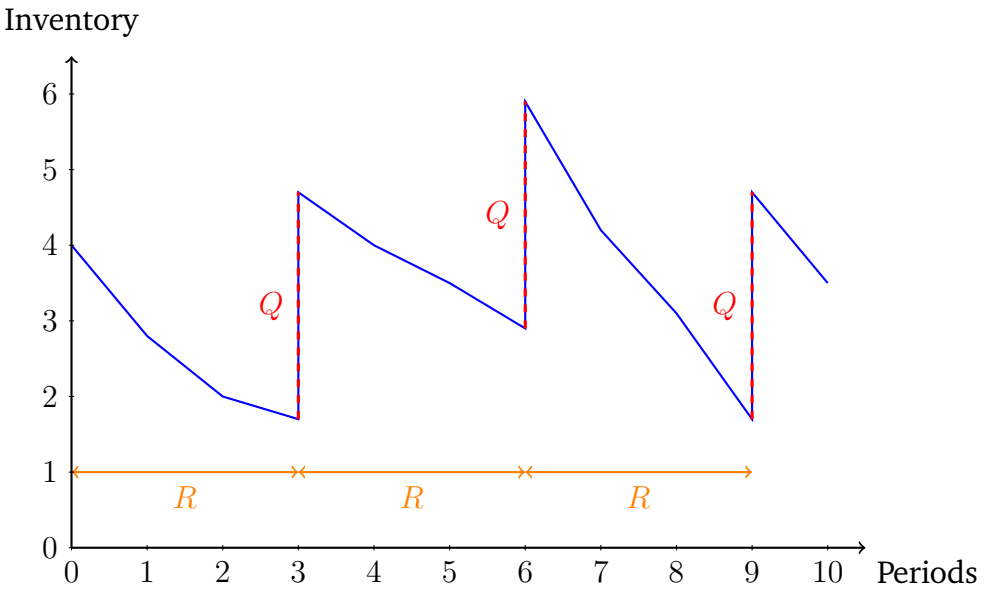

Figure 3.1: The expected inventory level under the (R, Q) policy

R and Q can change across the time horizon. In this case, the policy assumes the form (R_t, Q_t) where R_t is the length of the replenishment cycle starting in period t by placing an order of size Q_t.

Literature review

Sox [Sox97] introduces the first mixed-integer non-linear program formulation. His solution is based on [WW58] deterministic work with the addition of feasibility constraints. In their setting, the cost factors can change over time. The algorithm transforms the objective cost function into a series of multi-period newsvendor problems with various constraints by decomposition. A forward DP recursion is used to find the minimum cost setup sequence.

They derive two properties of the optimal (R, Q) policy. The first one is used to speed up the computations; the second one demonstrates that the lot sizes computed are an upper bound of the ones computed by the well-known SDP formulation. [Var09] shows that the same SDP is equivalent to the shortest path problem in a specific acyclic network. The algorithm proceeds in two stages: in the first one the replenishment quantities for any replenishment epoch are computed, in the second the optimal sequence of replenishment epochs is computed as the shortest path. The assumptions are the same as [WW58] with the addition of a backlogging penalty cost. In the same article, they present a solution to the normally distributed demand case. The idea of modelling

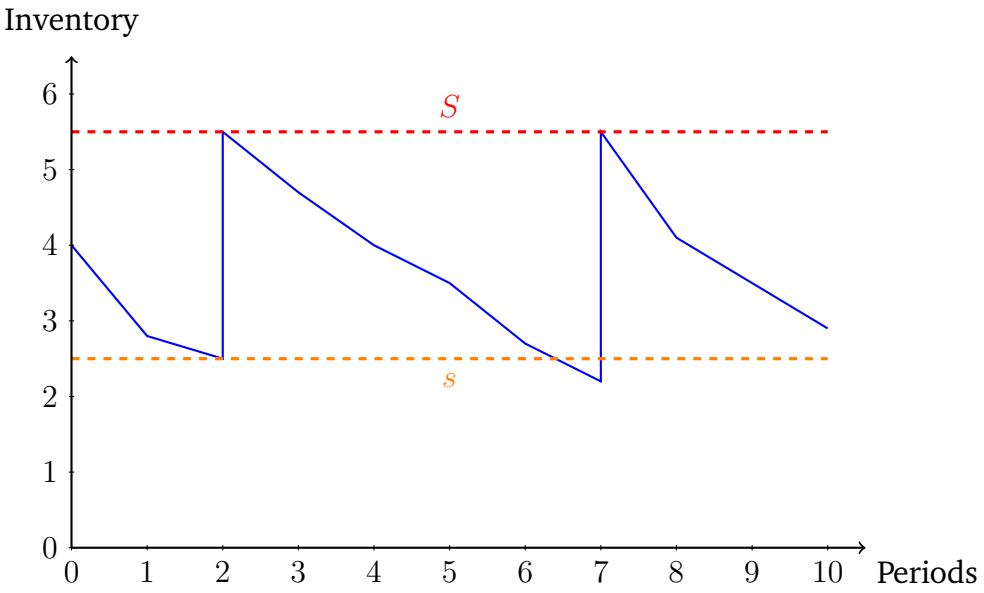

Figure 3.2: The expected inventory level under the (s, S) policy

the computation of the policy parameters as a shortest path problem has been widely used later on in the inventory control literature, e.g. [RTHP11]. In this case, the replenishment epochs are considered independently as edges of the graphs.

3.4.3 Order-Point, Order-Up-to-Level (s, S) System

This system assumes a continuous review. The inventory level is assessed at the beginning of each period. An order is placed whenever the inventory drops to (or under) the order-point (or reorder level) s. When an order is placed, the inventory level is raised to S, also known as the order-up-to-level. So, the quantity of the order is $Q = S - I$, where I is the current inventory level. The optimal policy determines the values s and S that minimise the total expected cost; for this reason, it is known in the literature as (s, S) policy. Figure 3.4.3 gives an example of this policy. This policy classifies as dynamic uncertainty strategy since orders' timing and size are decided at the beginning of each period.

When the demand is non-stationary, the parameters can change and the policy assumes the form (s_t, S_t).

As mentioned previously, [Sca59] proves that this strategy is the optimal policy. However, the computation of its parameters is a computationally intensive task.

The policy is often associated with a "just-in-time" production or a supplier with high stock levels able to dispatch orders with short notice. A disadvantage of this policy is the variable order quantity. Suppliers prefer the predictability of fixed order quantity, especially if the lot size is convenient for packing or shipping. This policy has been widely used in practice. It constitutes the heart of many material management modules in most of the Enterprise Resource Planning (ERP) packages. However, in most of the practical software, the levels s and S are set manually despite the considerable improvements with an optimal algorithmic determination [SK97].

Literature review

The literature regarding this policy is particularly broad. An SDP formulation for the problem is used as an example in Bellman's work [Bel66]. The seminal work of Scarf proves the optimality of the (s, S) policy for the finite horizon inventory system with convex holding and penalty cost, and fixed linear and ordering cost. To prove it, Scarf introduces the K-convexity property. This property considerably reduces the computational effort of the SDP, as Section 4.4.1 shows.

His work inspired a series of extensions. Iglehart [Igl63] provides bounds for the sequence of s_t and S_t parameters, and it proves the optimality of the (s, S) policy for the infinite horizon problem. Veinott [VJ66] proves the optimality under different assumptions: in Scarf's setting the one period expected costs are convex while in his work their negative must be unimodal, this is a weaker assumption. However, Veinott assumes that the absolute minima of the one period expected costs are rising over time. Finally, Aneja and Noori [AN87] extend the K-convexity property to the case in which a fixed penalty cost is charged together with the linear one. These early works focus on the optimality and the properties of the (s, S) policy; however, they do not provide any computational study.

The first heuristics for computing the (s, S) policy parameters are extensions of the Silver and Meal algorithm [Sil73] for the non-stationary deterministic problem. The algorithm in [Sil78] uses a greedy approach. The model uses a three-stage procedure to decide: when to order, how many periods the order should cover and order quantity. The safety stock for each cycle is determined considering two variants of the service level: a specified probability

of no stockout per replenishment cycle or a specified fraction of demand to be satisfied from stock.

Askin [Ask81] proposes a different heuristic, where the cost effect of probabilistic demand is taken into account when deciding the ordering times. The model uses a least period cost approach to determine the optimal order-up-to-level for the specific cycle length. Several modifications of the basic model to include a fixed shortage cost, simplify the computations or include an every-period-review approach. These two heuristics can be used to compute (R, S) parameters as well. His solution requires the convolution of the demand, so it requires higher computational effort for demand distributions different than the normal one.

A simulation-based procedure that tackles the (s, S) inventory control problem with service level constraint and random lead time is available in [BF98]. This particular setting is not widely tackled in the literature, and their approach achieves a 5% optimality gap in the vast majority of the cases.

A simple myopic heuristic that does not require the convolution is presented in [BM99]. This solution comprises two steps. The first one is based on the stationary approximation of an order cycle. Reorder level, order-up-to-level, length of the cycle and total expected demand are solved considering the cycles as stationary problems. The results are tabulated for all the possible mean demand of the cycles. The technique proposed in [ZF91] is used to populate the stationary table. The second step consists in reading the parameters from the table by averaging the non-stationary parameters between ordering periods. This approach has the advantage of needing only the probability distribution of a few periods into the future, while other approaches need the demands of all the planning horizon. However, this stationary approximation assumes that the mean is the only demand parameter that changes across different periods. If the number of parameters increases the dimension of the table increases exponentially with them, leading to higher computational complexity.

The most recent heuristic for this policy is [XRMBT18]. They propose a mixed-integer non-linear programming model that uses the piecewise linear approximation of the period cost function proposed in [RTPH14]. This creates models that can be solved by off-the-shelf solvers. They introduce a heuristic based on the previous MIP formulations that uses binary search to improve the computational performances. In the computational study, they compare the solutions optimality gap with the ones of [Ask81] and [BM99] computed

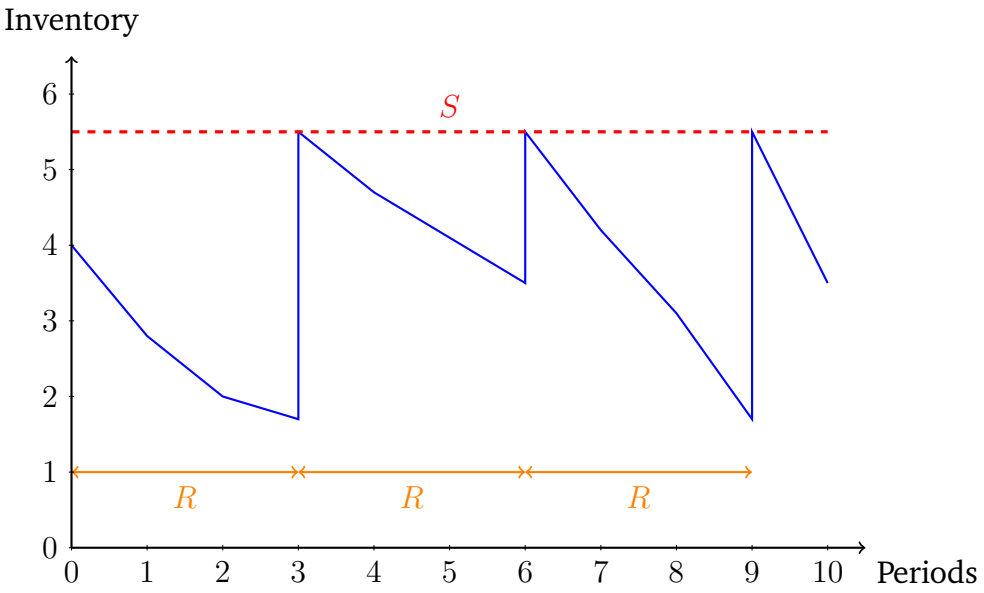

Figure 3.3: The expected inventory level under the (R, S) policy

in [DSRKT19] since the same dataset is used. The new heuristics outperform the existing ones.

3.4.4 Periodic-Review, Order-Up-to-Level (R, S) System

This policy is the one originally classified as static-dynamic in [BT88]. It provides a stable replenishment plan by determining the timing of future orders at the beginning of the planning horizon. The order quantities are decided after observing the actual inventory level, so after observing the demand for previous periods. This policy uses the order-up-to-level model. This means that at every order, the inventory is raised at level S; so, as in the previous policy, the order quantity is $Q = S - I$. For this reason, this system is known in the literature as (R, S) policy. See Figure 3.4.4 for an example of this policy.

This policy can handle uncertainty in a better way compared to a static policy. Even without considering the review cost is competitive compared to the dynamic strategy [DSRKT19]. Moreover, the (s, S) policy presents a high degree of nervousness. This is due to the variation of the original replenishments schedule, which frequently changes after the demand realisation. De Kok and Inderfurth [DKI97] reveal that in terms of nervousness, the dynamic strategy exhibits the worst performance among the policies considered. Heisig [Hei98] studies the planning stability of the policy and shows that the nervousness is affected by the demand uncertainty and the minimum lot size. In his later

work [Hei01], he founds similar results for the rolling horizon problem.

Because of the fixed order schedule, the (R, S) policy is much preferred in terms of coordinating the replenishment of related items and in material requirement planning [SPP+98]. For example, when ordering from overseas, it is often necessary to fill a container to reduce the shipping cost per item. However, it offers a good cost performance even in systems with stochastic demand since the order quantities are decided after realising the previous demands.

When the demand is non-stationary, the parameters can change and the policy assumes the form (R_t, S_t).

Literature review

In this section, we survey the literature regarding the problem of computing policy parameters for the (R, S) policy. With few exceptions, the policies analysed here use Scarf's setting with backlogging of the excessive demand and penalty costs or service level constraints.

Bookbinder and Tan [BT88] is considered the first algorithm to compute (R, S) parameters. They propose a heuristic structured in two-stages for the dynamic uncertainty strategy with α service level constraint. The first stage consists in fixing the replenishment cycles, by using [WW58] technique. The second part is a linear programming model that computes the optimal order-up-to-levels for the fixed replenishment plan.

Tarim and Kingsman [TK04] present a MIP formulation of the same problem. Their model computes replenishment plan and order-up-to-level jointly, taking into account also a linear holding cost previously neglected in [BT88].

The same authors [TK06] provide another MIP formulation that computes the parameters for the linear penalty cost setting. This version of the problem is significantly more complicated than the service level one due to the non-linearity of the cycle cost function. The cycle cost function computes the expected holding and penalty cost that occurs in the periods between two replenishments. They adopt a piecewise linear approximation of the cost function. A series of linear segments approximate the cost function, each additional segment adds a constraint to the model. The explicit formulation assumes that the demand is normally distributed; however the linearization parameters are the same for every normally distributed demand. To the best of

our knowledge, this is the first paper of the static-dynamic approach to present a computational study.

The early work of Rossi et al. [RTHP08] introduces the first optimal solution for the problem with normally distributed demand. They propose a stochastic constraint programming model for computing (R, S) policy parameters. This model is based on a novel concept, global chance constraint. The work of [CC59] inspires this constraint as a means of handling uncertainty. They specify a confidence level at which it is desired that the stochastic constraint holds. The model's scalability is limited since the number of binary variables increases polynomially with respect to the planning length.

Tarim et al. [TS08] present a different CP approach for the same problem. This formulation has the advantage of having fewer decision variables compared to [RTHP08]. Their work comprises a computational study in which the new approach shows to be more solvable compared to [TK04] MIP formulation. The paper also proposes two domain reduction techniques to improve the computational performance of the MIP and CP formulations.

If we relax the constraint that enforces the non-negativity of the orders, the replenishment cycles can be considered independently. This allows the restitution of unsold items to the supplier. Under this assumption, the problem can be model as a shortest path problem. However, for limited cases, the solution found is unfeasible for the service level settings. Rossi et al. [RTHP11] propose a DP approach for solving the relaxed problem. They introduce a filtering procedure to rule out sub-optimal replenishment cycles and a state augmentation technique to extend the relaxed graph to solve instances with negative orders.

In [TDÖR11] a MIP solution for the relaxed graph is presented. This solution is computationally efficient and exhaustive numerical experiments prove that it provides the optimal solution most of the time. When the solution is unfeasible, it provides a tight lower bound to the cost of the optimal policy. An unfeasible solution can be modified to obtain a solution, which yields an upper bound of the optimal cost. These approaches are not computationally affected by the magnitude of the demand. However, they use the convolution of the demand.

Özen et al. [ÖDT12] prove the optimality of the base stock policy (R, S) for the static uncertainty strategy. The proof is valid for both penalty cost and service level configuration. They propose an optimal formulation of the problem;

however, the complexity of this formulation is too high to make them usable on problems of reasonable size. They propose two heuristics to compute the policy parameters: a mathematical model and a DP based one. The algorithms and the optimality proof are then extended to three variants of the problem: the capacitated version where the inventory stock can not exceed the storage capacity, the minimum order quantity and purchase cost per unit.

In many real-life industrial settings, the lead time can not be considered null. Approximations that ignore this aspect can lead to a significant increase in the inventory cost. Some of the methods presented so far can be extended to model a deterministic lead time. Hua et al. [HYHX09] present a rolling horizon approach to solving the (R, S) policy computation problem with fixed lead time and linear penalty cost. In some situations, the uncertainty regards the lead time as well.

Rossi et al. [RTHP10] present the first approach in the literature that tackles the problem with non-stationary stochastic lead time. They use the same global chance constraint presented in [RTHP08]. The same authors, in [RTHP12], adapt this work to the more complex shortage penalty cost settings. The exact CP formulation provided uses a cost-based filtering technique to improve search performances.

A linear reformulation of the Tarim and Kingsman MIP model [TK04] is presented in [TKTE14]. Tunc et al. reformulate the previous work into a deterministic equivalent MIP model by means of alternative decision variables which provides a stronger linear relaxation. A stronger relaxation leads to a more effective pruning of the search tree, so better computational performances.

Rossi et al. [RKT15] generalise the MIP discussion regarding the stochastic dynamic strategy. They present a unified MIP formulation that can be used to model a variety of situations. The model can cover both backlogged and lost demand in case of a stockout; to avoid stockouts linear penalty cost or three different service levels can be considered. To linearize the cycle cost function, [RTPH14] is used. This approach provides a lower and upper bound of the expected cost. This work enables the modelling for several variants of the stochastic lot sizing with a fully linear formulation. When the demand is normally distributed, the model can be simplified and [RTPH14] provides the linearisation parameters up to 11 segments.

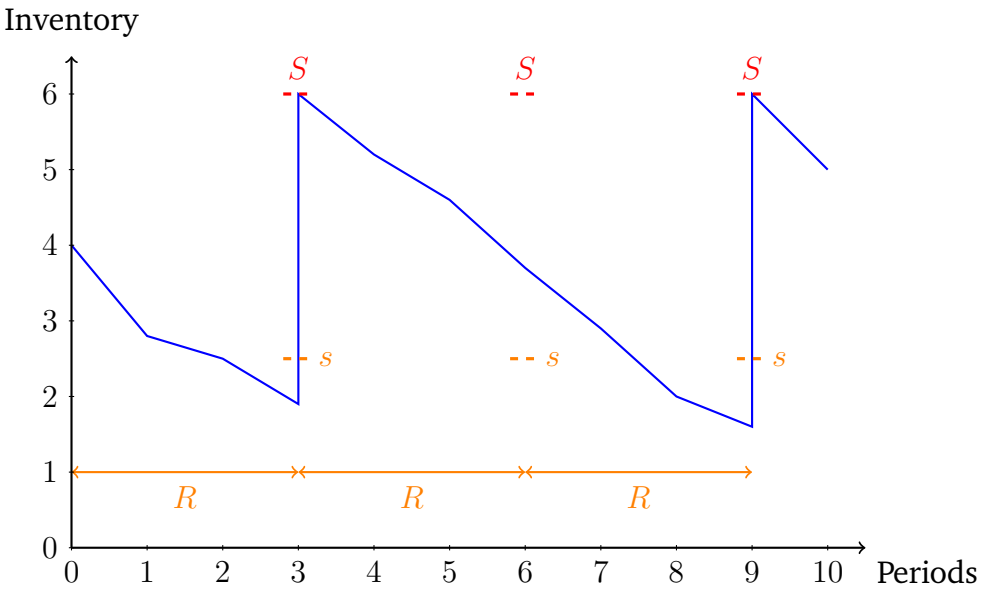

Figure 3.4: The expected inventory level under the (R, s, S) policy

Tunc et al. [TKTR18] extend the MIP formulations just presented. They blend the work of [TKTE14] and [RKT15] by replacing the linear approximation of the cycle cost with a dynamic cut generation approach, achieving the computational efficiency of the first and the modelling variety of the second.

3.4.5 (R, s, S) System

This policy is a combination of the previous two. In an (R, s, S) system, the inventory level is checked only at review moments fixed at the beginning of the planning horizon. If the inventory level is at or below the reorder point s, an order is placed to raise the inventory level to S. Figure 3.4.5 shows an example of an inventory system managed with a (R, s, S) policy.

(s, S) and (R, S) are two special cases of this policy. If we review the inventory in each period the (R, s, S) is reduced to an (s, S) policy. This is common when no costs are charged for reviewing the inventory. Part of the literature in lot sizing use it with a fixed $R = 1$, that makes the policy equivalent to a periodic version of the (s, S) policy widely studied. The (R, S) policy can be seen as a particular version of the (R, s, S) one in which $s = S - 1$ so that all the review period lead to an order; this is based on the assumption that the current inventory level exceed S only on very rare occasions that can be ignored. The computational effort to determine the best values of these parameters is much greater than the other policies. Therefore, in many practical applications,

simpler techniques are deployed even if less cost-efficient.

When the demand is non-stationary, the parameters can change and the policy assumes the form (R_t, s_t, S_t).

Literature review

In this section, we first survey the literature addressing the (R, s, S) policy with the review cycle length fixed across the time horizon. We then survey techniques developed to compute policy parameters for different problem configurations, e.g. two-echelon or multi-items. Finally, we survey real-world applications using the (R, s, S) policy. Determining the optimal value of the policy parameters is considered computationally prohibitive in the literature [BST10]. To the best of our knowledge, there are currently neither simple procedures nor algorithms to give the optimal values of s and S in any particular practical situations [BST10].

The work of this thesis differs from the ongoing research by considering (R_t, s_t, S_t) policy under non-stationary stochastic demand. Due to its complexity, the surveyed papers consider the value R to be constant across the time horizon. This reduces the policy into a periodic (s, S) policy, also referred to as (T, s, S) in the literature. Even with the constant review cycle length, most of the solutions rely on heuristics [SRK95, MY08] and simulation techniques [GBDG15, SM$^+$02]. To the best of our knowledge, [SK97, BST10] are the only two empirical comparative studies on the performances of periodic (s, S) heuristics. In [XRMBT18] we can find a mixed-integer non-linear programming formulation for determining near-optimal parameters and an updated literature review for this problem.

These policies have been studied for different problem configurations as well. Here we survey some of the related research. Two heuristics to compute periodic (s, S) policy parameters with periodic review in a two-echelon inventory system with one warehouse and multiple retailers have been introduced in [SR91, SRK95]. [SM$^+$02] proposes a technique to simulate an (R, s, S) inventory system where the parameters stay constant across the periods. It can be used to compute fill rates or to find parameters values to achieve a prescribed service level. Chen and Lin [CL09] adopt a hedge based (R, s, S) policy portfolio with constant parameters in the short term for a multi-product inventory control problem. [GBDG15] addresses an optimisation

via simulation technique to determine optimal (R, s, S) policy for distribution centres in a two-echelon inventory system with lost sales.

The (R, s, S) policy is widely used in the literature, usually not independently but as a component of complex supply chains. Here we analyse some recent models involving this policy. In [BV12a] is described as an inventory control system for point-of-use location. They compare the performance of (R, s, Q) policies (where the order quantities are fixed) against the (R, s, S) in the presence of stochastic stationary demand. Since they consider stationary demand, the parameters of the policy are constant through the time horizon. [AMP18] and [MY08] tackle a capacitated two-echelon inventory system with one warehouse and multiple retailers. They use a heuristic based on [SRK95], for the (R, s_n, S_n) policy. Cabrera et al. [CMC$^+$13] consider a similar two-level supply chain in which a single plant serves a set of warehouses, which in turn serve a set of end customers or retailers. They develop a heuristic to solve an inventory location model on this configuration. The warehouses model is based on (R, s, S). The same problem has been tackled by Araya-Sassi et al. [ASMPB18] using Lagrangian relaxation and the subgradient method. In [BV12b], we can find an analysis of lost-sales inventory control policies with a service level constraint. They define an optimal policy solved starting from the (s, S) SDP introduced by Scarf. They present a value-iteration algorithm to find the (R, s, S) parameters that minimise the inventory cost subjected to service constraints. They compute fixed parameters for the policy; this makes their solution non-suitable for a non-stationary problem.

3.4.6 Receding horizon deployment

The receding horizon [KH06] (or rolling horizon) is a way to deploy the policies that this chapter presents. Receding horizon control proceeds as follows:

1. The policy parameters are computed for the entire planning horizon,
2. Only the decisions relevant to the current period are implemented,
3. The policy parameters for the remaining of the time horizon are recomputed at the beginning of each period.

Receding horizon is widely used in the lot sizing literature, e.g. [KT11]. In a recent computational study [DSRKT19], Dural et al. show that when an inventory control policy is used in a receding horizon framework, the

cost performances improve considerably, and the differences between policies become insignificant. However, this approach increases the nervousness of the system since the timing of the orders can change multiple times.

3.5 Conclusions

This chapter gives a brief introduction to the main concepts used herein and provides a clear picture of the state-of-the-art regarding the problems tackled in this thesis. This chapter highlights the literature's shortfalls that we aim to fill. An approach common to the solutions presented is SDP; this shows the great variety and flexibility of this technique.

- No functional equation full formulation is available for the static-dynamic strategy under linear penalty cost. This formulation is harder to tackle compared to the service level one due to the non-linearity of the period cost function. A similar formulation is the most well known and used approach for the (s, S) policy. The majority of the algorithms available make assumptions on the type of random variable representing the demand. Many solutions rely on commercial MIP solvers, which licence for non-academic usage is expensive. This can limit the deployability of the solutions. Chapter 4 presents the first pure SDP formulation to compute the optimal parameters for the (R, S) policy. Extensive computational results compare it with the state-of-the-art, proving it to be efficient from the computational point of view and for the quality of the solution.
- The (R, s, S) policy has substantial practical importance. However, the computation of its parameters is considered too computationally expensive, especially in the non-stationary stochastic case. For this reason, it is generally approximated to a (s, S) policy with periodic reviews. In Chapter 5 we fill this gap by introducing the first algorithm for the computation of the optimal parameters; this solution is a hybridisation of SDP and BnB with ad-hoc bounds computed in a DP way. The optimal policy allows computing optimality gaps for the heuristics presented in the following chapter.
- A set of heuristics are presented in Chapter 6, these approaches allow to compute near-optimal parameters of large instances in a reasonable time.

- To model a DP solution in CP, the well-known flow formulation needs to be used. However, this variable redefinition technique performs poorly in CP solvers. In Chapter 7, we present the dynamic programming encoding (DPE) technique, a novel approach to model some DP and SDP solutions in CP. This allows formulating the problem as a CP model that does not require any tailor-made global constraint. While this approach is not computationally competitive with the algorithm presented in Chapter 5, the encoding can be used successfully in other situations. An example is the designing of non-solver specific DP based constraints.

‘

Chapter 4

Stochastic Dynamic Programming for the (R, S) policy

In this chapter, we present a novel approach to compute the static-dynamic policy parameters. The (R, S) policy has strong practical value, since the stable replenishment plan reduces the nervousness of the system; while the policy deals efficaciously with unexpected demand and has a good cost performance.

In the literature surveyed in Section 3.4.4, no method can find the optimal solution dealing with a generic demand; most of them work with normal or symmetric distributions. No formulation to compute the optimal parameters using SDP is available. Scarf's SDP solution for the (s, S) parameters computation is probably the most famous and used algorithm in stochastic lot sizing, and it is widely used to compute the optimality gap in many works, e.g. [XRMBT18, DSRKT19]. We present his algorithm and the K-convexity property in Section 4.1. We use this algorithm and property through the rest of this thesis, so it is essential to have a complete definition of this approach. Another important reason to describe in detail the algorithm is that not in all the literature, the approach is deployed using the K-convexity property. This property has a crucial impact on the algorithm's performances.

The algorithm presented in this chapter is the first formulation of the (R, S) policy with penalty cost as a functional equation. A series of optimisation techniques greatly reduce the computation effort required by this solution. No assumptions on the demand type are necessary. The policies computed have a lower cost, a more accurate expected cost and a lower optimality gap compared to the state-of-the-art.

In Section 4.2, we present the SDP formulation for the (R, S) problem and we provide the algorithm's pseudocode. Then, we introduce three different techniques to improve the computational performances of the SDP model: a memoisation technique that avoids recomputation (Section 4.2.4), a filtering technique to prune the state space without compromising the optimality (Section 4.2.5 and a binary search approach to speed up the search for the optimal order-up-to-level (Section4.2.6). In the following section, we present the MIP model of Rossi et al. [RKT15]. We use it as a competitor since it is the best algorithm for the static-dynamic policy analysed in the recent comparison of Dural et al. [DSRKT19]. Section 4.4 shows the results of an extensive experimental analysis of the approaches. Finally, Section 4.5 concludes the chapter.

4.1 Stochastic Dynamic Program for (s, S) policy

Scarf's seminal paper [Sca59] addressed the computation of the (s, S) policy parameters over a finite planning horizon. Its setting is characterised by discrete time periods, non-stationary stochastic demands, a fixed ordering cost, and linear holding and shortage costs. Scarf proves that the (s, S) policy (more precisely the (s_t, S_t) policy) is cost-optimal for this particular configuration. However, it ignores the operational cost generated by the stock-taking. Considering the review cost makes the (s, S) policy non-cost-optimal. In a situation where the review cost is high compared to the ordering and holding cost is not convenient to review the inventory at every period. The same is true in the case in which the demand is deterministic, that is a special case of the stochastic version.

4.1.1 Model

Without loss of generality, we consider the proportional ordering cost to be zero. The extension of our solution to the case of a non-zero unit production/purchasing is straightforward; as this cost can be reduced to a function of the expected closing inventory level at the very last period [TK04].

The problem can be formulated as SDP as:

1. **Stage.** A stage represents a time period $t = 1, \ldots, T$ for a T-period stochastic lot-sizing problem.

2. **State.** We define N_t as the state of the system at the beginning of period t before replenishment. State $N_t = I_{t-1}$ includes the opening inventory level of period t.

3. **Action.** An action is represented by the scheduling of an order with quantity Q_t at the beginning of period t.

4. **Immediate cost.** Let $f_t(I_{t-1}, Q_t)$ be the expected immediate cost compromising ordering, holding and penalty cost, given the state $N_t = I_{t-1}$ and action Q_t.

$$f_t(I_{t-1}, Q_t) = \quad K\mathbb{1}\{Q_t > 0\} + E[h\max(I_{t-1} + Q_t - d_t, 0) + b\max(d_t - I_{t-1} - Q_t, 0)] \tag{4.1}$$

   where E denotes the expected value with respect to the random variable d_t and $\mathbb{1}$ is the indicator function.

5. **Objective function.** Let $C_t(I_{t-1})$ denote the expected total cost of an optimal policy over periods $t, \ldots, T$ associated with state $N_t = I_{t-1}$. Then, $C_t(I_{t-1})$ can be written as:

$$C_t(I_{t-1}) = \min_{Q_t}(f_t(I_{t-1}, Q_t) + E[C_{t+1}(I_{t-1} + Q_t - d_t)]) \tag{4.2}$$

   The boundary condition is:

$$C_{T+1}(I_T) = 0 \tag{4.3}$$

$C_1(I_0)$, where I_0 is the initial inventory, contains the expected cost for the optimal (s,S) policy.

It is possible to extend this solution to consider the review cost W as well. The expected cost of the policy is:

$$C_1(I_0) + WT \tag{4.4}$$

4.1.2 Pseudocode

Algorithm 1 shows the procedure to compute the SDP backwards. Lines 1-2 contains the boundary condition. Line 4 search through all the possible starting inventory levels and line 6 through all the possible order quantities.

Algorithm 1 sS-SDP()

```
1: for i from min_inventory to max_inventory do
2:     C_{T+1}(i) = 0
3: for t from T down to 1 do
4:     for i from min_inventory to max_inventory do
5:         C_t(i) ← ∞
6:         for q from 0 to max_order do
7:             expected_cost ← f_t(i, q) + E[C_{t+1}(i + q − d_t)]
8:             if expected_cost < C_t(i) then
9:                 C_t(i) ← expected_cost
```

4.1.3 Time complexity

The complexity analysis is based on [CLRS09]. In most of the DP solutions, the time complexity is the number of states multiplied by the number of possible action multiplied by the complexity of evaluating the action. Let D be the maximum demand d_t with non-zero probability and T the length of the planning horizon. The maximum possible inventory level can be bounded to DT, the situation in which we place an order in the first period that guarantees to satisfy the maximum possible demand. Similarly, we can calculate the minimum possible inventory. Starting from the initial inventory, we realise the maximum demand in all the periods without placing any order; the minimum inventory is $-DT$. All the possible inventory levels evaluated in line 5 of Algorithm 1 are $2DT$. The total number of DP states is $2DT^2$.

We can consider the maximum order size to be DT, an order placed at the initial inventory level used to reach the maximum. So, the cycle in line 6 iterates DT times. The computational effort of each cycle depends on the immediate cost and the expected cost of future periods, both of them are expected values depending on the random discrete positive variable d_t, the computation of each requires $\mathcal{O}(D)$ operations. So, the complexity of the computation of each SDP state is

$$\mathcal{O}(D^2T) \tag{4.5}$$

So, the overall complexity is:

$$\mathcal{O}(D^3T^3) \tag{4.6}$$

4.1.4 K-convexity

We can exploit the property of K-convexity presented in [Sca59] in solving the dynamic program.

The property is defined as:
Definition 4.1.1. *Let $K \geq 0$, then function $f(x)$ is K-convex if:*

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

for all positive a, b and x.

Using this property, Scarf proves the optimality of the (s_t, S_t) policy under non-stationary stochastic demand. Moreover, he shows that considering s_t^* and S_t^* the optimal reorder level and order up-to level for period t:

$$C_t(I_{t-1}) = \begin{cases} f(I_{t-1}, 0) + E[C_{t+1}(I_{t-1} - d_t)] & s_t^* \leq I_{t-1} \leq S_t^* \\ f(I_{t-1}, 0) + E[C_{t+1}(S_t^* - d_t)] + K & 0 \leq I_{t-1} < s_t^* \end{cases} \tag{4.7}$$

This is done by computing the $C_t(y)$ for different values of y starting from a high value that is an upper bound of S_t. The value y is then decremented by a unit each time, and the lowest value of C_t is remembered. The search terminates when the cost is greater than the lowest value so far added to K, the fixed ordering cost. S_t is the inventory level in which the cost assumes the minimum value, s_t is the one in which we stop the search. This approach greatly speeds up the computation of the SDP.

The reason for the improvement is clear in Algorithm 2. There is no need to search for the best order quantity Q_t. Moreover, when the order level s_t is determined all the lower inventory levels assumes the same expected cost. In Section 4.4, we quantify the improvement provided by this property.

The worse case complexity is strongly reduced in this case. Each state requires a single computation of the immediate cost and the expected cost of future periods, so its worse case complexity become $\mathcal{O}(D)$. For all the inventory levels lower than the optimal s_t the complexity of computing them is constant; however, it strongly depends on the instance type. The overall complexity is

Algorithm 2 sS-SDP-KConvex()

for i from $min\_inventory$ to $max\_inventory$ **do**
    $C_{T+1}(i) = 0$
for t from T down to 1 **do**
    $best\_cost \leftarrow \infty$
    for i from $max\_inventory$ down to $min\_inventory$ **do**
        $C_t(i) \leftarrow f_t(i, 0) + E[C_{t+1}(I_{t-1} + Q_t - d_t)]$
        if $C_t(i) < best\_cost$ **then**
            $best\_cost \leftarrow C_t(i)$
            $S_t \leftarrow i$
        if $C_t(i) > best\_cost + K$ **then**
            $s_t \leftarrow i$
            break for
    for i from $min\_inventory$ to s_t **do**
        $C_t(i) \leftarrow C_t(s_t)$

reduced to:

$$\mathcal{O}(D^2T^2) \tag{4.8}$$

4.1.5 Demand over consecutive periods

To simplify the notation of the next models, we introduce a new random variable that represents the demands faced over consecutive periods. We define it as $d_{i,j}$, where i is the initial period and j the final one.

$$d_{i,j} = \sum_{k=i}^{j-1} d_k \tag{4.9}$$

by definition d_t and $d_{t,t+1}$ are equal.

4.2 Stochastic Dynamic Program for (R, S) policy

This section introduces an optimal SDP solution for the dynamic uncertainty strategy.

We consider the individual ordering cost, as mentioned before the extension to a non-zero ordering cost is straightforward. In [XRMBT19], we can find the first attempt to derive the (R, S) policy in the form of a functional equation.

However, their formulation differs to Scarf's only by the functional equation, while herein, we have a different state space and action. Moreover, their formulation requires to compute the whole state space for all the possible replenishment plans, a brute force evaluation. For this reason, they do not implement the technique in their experimental analysis.

4.2.1 Model

The formulation presented herein differs with Scarf's. Stages are equivalent; however, it uses a different state structure. In the (s, S) SDP, each state is associated with a period and an inventory level. Since we have a null per unit ordering cost, we can neglect the inventory level at the beginning of a period in which an order is placed and focus only on the order-up-to-level. Action introduces a new dimension. We are not only searching the best order-up-to-level S_t but also how many periods demand the order should cover R_t. The immediate cost regards all the periods covered by the order. The review W and ordering cost K are always charged together since there is no distinction between them in the dynamic uncertainty policy.

Let R_t be the length of the review period starting in period t, and S_t be the order-up-to-level in period t. If the inventory level I_{t-1} is higher than the order-up-to-level S_t in an order instant, an empty order is placed. The problem can be formulated as SDP as:

1. **Stage.** A stage represents a time period $t = 1, \ldots, T$ for a T-period stochastic lot-sizing problem.
2. **State.** We define N_t as the state of the system at the beginning of period t before replenishment.
3. **Action.** An action is represented by the pair (S_t, R_t). They represents the scheduling of an order that raise the inventory level to S_t at the beginning of period t that aims to cover the demand of the next R_t periods. So, the next review moment is in period $t + R_t$.
4. **Immediate cost.** Let $f_t(S_t, R_t)$ be the expected immediate cost compromising ordering, holding and penalty cost for periods $t, \ldots, t + R_t - 1$,

given the state $N_t = I_{t-1}$ and action S_t, R_t.

$$f_t(S_t, R_t) = \quad K + W + \sum_{i=1}^{R_t} E[h \max(\max(S_t, I_{t-1}) - d_{t,t+i}, 0) + b \max(-\max(S_t, I_{t-1}) + d_{t,t+i}, 0)] \tag{4.10}$$

where E denotes the expected value with respect to the random variable $d_{t,t+i}$.

5. **Objective function.** Let C_t denote the expected total cost of an optimal policy over periods $t, \ldots, T$ that places an order in period t and associated with state N_t. Then, C_t can be written as:

$$C_t = \min_{R_t}(\min_{S_t}(f_t(S_t, R_t) + C_{t+R_t})) \tag{4.11}$$

The boundary condition is:

$$C_{T+1} = 0 \tag{4.12}$$

C_1 contains the expected cost for the optimal parameters.

4.2.2 Pseudocode

Algorithm 3 shows the pseudocode of the SDP. At the end of $RS - SDP()$, the variables R_t and S_t will contain the optimal parameters for the (R, S) policy. Line 1 represent the boundary condition. Lines 4-6 represent the functional Equation 4.11.

Algorithm 3 RS-SDP()

```
1:  C_{T+1} = 0
2:  for t from T down to 1 do
3:      C_t ← ∞
4:      for r from 1 to T − t + 1 do
5:          for s from 0 to max_inventory do
6:              expected_cost ← f_t(s, r) + C_{t+1}
7:              if expected_cost < C_t then
8:                  C_t ← expected_cost
9:                  S_t ← s
10:                 R_t ← r
```

For clarity and for the sake of the future enhancements, we separate the computation of the immediate cost. Let $\zeta_{t,t+j}$ be a value of the random variable

$d_{t,t+j}$ and $P(\zeta_{t,t+j})$ be the probability of assuming that value. Algorithm 4 summarises the computation of Equation 4.10.

Algorithm 4 $f_t(s, r)$

```
1: cost ← W + K
2: for j from 1 to r do
3:     for each ζ_{t,t+j} value of d_{t,t+j} do
4:         close_inv ← s − ζ_{t,t+j}
5:         if close_inv ≥ 0 then
6:             cost ← cost + h close_inv P(ζ_{t,t+j})
7:         else
8:             cost ← cost − b close_inv P(ζ_{t,t+j})
   return cost
```

4.2.3 Time complexity

We use the same notation and reasoning presented in Section 4.1.3. The number of states is T. The iteration over all the possible replenishment cycles (line 4 of Algorithm 4) cycle on average $T/2$ times, so $\mathcal{O}(T)$. The range of possible values for S_t is from 0 to the maximum inventory level, consequently the cycle in line 5 iterates DT times. The computation's complexity of the immediate cost can be derived from Algorithm **??**. The for cycles in lines 2 and 3 iterate respectively a maximum of T and D times. The overall complexity of the algorithm becomes:

$$\mathcal{O}(D^2T^4) \tag{4.13}$$

4.2.4 Immediate Cost Memoisation

The calculation of the immediate cost is particularly time demanding. There is a summation of expected costs over multiple periods. However, it is possible to identify situations in which the same computations occur multiple times. Let $l_t(I_t, R_t)$ be the function that computes the holding and penalty expected cost of starting at the end of period t with closing inventory I_t and with the next review moment in R_t periods. This new function can be defined as:

$$l_t(I_t, R_t) = \sum_{i=1}^{R_t} E[h \max(I_t - d_{t+1,t+i}, 0) + b \max(-I_t + d_{t+1,t+i}, 0)] \tag{4.14}$$

considering $d_{i,j} = 0$ when $i = j$. Equation 4.10 can be rewritten as:

$$f_t(S_t, R_t) = K + W + E[l_t(S_t - d_t, R_t)] \tag{4.15}$$

The $l_t(I_t, R_t)$ function can be computed in a recursive way:

$$l_t(I_t, R_t) = h \max(I_t, 0) + b \max(-I_t, 0) + E[l_{t+1}(I_t - d_{t+1}, R_t - 1)] \tag{4.16}$$

this can be considered as the functional equation of an SDP, where the holding/penalty cost of period t is the immediate cost. There are two boundary conditions:

$$l_{T+1}(I_T + 1, R_t) = 0 \tag{4.17}$$

$$l_t(I_t, 0) = 0 \tag{4.18}$$

The states are represented by the tuple (t, I_t, R_t) and are computed in a forward manner. We store the computed tuples in a dictionary with constant access time to avoid recomputations.

Algorithm 5 $l_t(i, r)$

Data: $memo$ is the data structure that contains the solutions of the states. $memo(t, i, r)$ contains the solution of $l_t(i, r)$

if (t, i, r) in $memo$ **then return** $memo(t, i, r)$
$cost \leftarrow 0$
for each $\zeta_{t,t+j}$ value of $d_{t,t+j}$ **do**
    $close\_inv \leftarrow i - \zeta_{t,t+j}$
    if r not 0 **then**
        $cost \leftarrow cost + l_{t+1}(close\_inv, r - 1)\ P(\zeta_t)$
    if $close\_inv \geq 0$ **then**
        $cost \leftarrow cost + h\ close\_inv\ P(\zeta_t)$
    else
        $cost \leftarrow cost - b\ close\_inv\ P(\zeta_t)$
return $cost$

Complexity wise, it is hard to assess the performances of the memoisation since they are strongly case dependent. The experimental section analyses the empirical impact.

4.2.5 Replenishment Cycle Length Filtering

In the previous section, we presented a memoisation technique to avoid recomputations. We now aim to reduce *a priori* the number of calls to that function. We exploit a reduction procedure based on the structure of the function.

Property 4.2.1. *Let C_t^+ be an upper bound of C_t and R_t^a a fixed possible review cycle for period t. If:*

$$C_t^+ \leq \min_{S_t}(f_t(S_t, R_t^a)) \tag{4.19}$$

The optimal replenishment cycle for period t is smaller than R_t^a.

Property 4.2.1 is derived by:

Property 4.2.2.

$$\min_{S_t}(f_t(S_t, R)) \leq \min_{S_t}(f_t(S_t, R+1)) \tag{4.20}$$

the minimum holding/penalty cost increase if the cycle length increases since the demand is non-negative.

For a fixed R_t^a, the expected cost C_t^a is, according to the functional equation:

$$C_t^a = \min_{S_t}(f_t(S_t, R_t^a) + C_{t+R_t^a}) = C_{t+R_t^a} + \min_{S_t}(f_t(S_t, R_t^a)) \tag{4.21}$$

finally for Equations 4.19, 4.20 and 4.21:

$$C_t^+ \leq \min_{S_t}(f_t(S_t, R_t^a)) \leq C_{t+R_t^a} + \min_{S_t}(f_t(S_t, R_t^a)) = C_t^a \tag{4.22}$$

We can consider as cost upper bound C_t^+ the best C_t computed so far. This property allows to stop the search for the optimal replenishment cycle earlier and save computations. Line 4 of Algorithm 3 iterates over all the possible cycle length R_t, we can use Property 4.2.1 to break this cycle earlier.

4.2.6 Binary Search of the Order-Up-To-Level

The K-convexity property can not be exploited in the solving of this SDP model. However, we can adopt a similar approach with standard convexity and limit the search of the optimal S_t thanks to the convexity of $f_t(S_t, R_t)$.

Definition 4.2.3. *(Convex function). A function f is convex, if for every x, y and*

$0 \leq \delta \leq 1$ *the next inequality holds:*

$$f(\delta x + (1 - \delta)y) \leq \delta f(x) + (1 - \delta)f(y) \tag{4.23}$$

A convex function shows the following properties:

Property 4.2.4.
- *Every linear function is convex.*
- *The non-negative weighted sum of convex functions is convex. If $w_1, \ldots, w_n \geq 0$ and $f_1, \ldots, f_n$ are all convex, then $w_1 f_1 + \cdots + w_n f_n$ is convex. This property extends to infinite sums and integrals.*
- *The maximum of convex functions is convex. If $f_1, \ldots, f_n$ are all convex, then $max\{f_1 + \cdots + f_n\}$ is convex.*

Thanks to these properties, it is possible to prove by induction the convexity of $f_t(S_t, R_t)$ for a fixed R_t.

Considering R_t constant, the immediate cost is depends only on the S_t variable.
Theorem 4.2.5. *Function $f_t(S_t, R_t)$ is convex with respect to variable S_t.*

To prove it we need first to prove that l_t presented in Section 4.2.4 is convex for a fixed R_t. It can be proved by mathematical induction:

- **Base case**. $l_{T+1}(I_t, R_t)$ is a constant function, so it is convex.
- **Induction step**. Let l_{t+1} be convex, considering 4.16, $E[l_{t+1}(x, R_t - 1)]$ is convex for Property 4.2.4 since it is a weighted sum of convex functions. $h \max(I_t, 0)$ is convex as well since it is the maximum of two linear functions, same goes for $b \max(-I_t, 0)$. If $l_{T+1}(I_t, R_t)$ is convex, so is $l_t(I_t, R_t)$.

Finally, if $l_t(I_t, R_t)$ is convex also Equation 4.15 is convex since it is a weighted sum of convex functions.

Considering the functional Equation 4.11, for a fixed R_t the the right hand function is convex. We are looking for the global minimum of this function since the function is convex there are no local minimums.

To find the global minimum, we can use binary search. Line 5 of Algorithm 3 is replaced by a binary search in the same interval. The binary search reduces the complexity, since the block of code inside the cycle in line 5 is iterated $\log(DT)$

times instead of DT. The worse case complexity becomes:

$$\mathcal{O}(DT^3 \log(DT) \tag{4.24}$$

4.2.7 Extension to unit cost

The algorithm can be extended to model the per unit ordering cost. There are two options, reducing it to a function of the expected closing inventory, e.g. [TK04]; or including it in the immediate cost function. To do so, we need to modify the boundary condition for the computation of the $l_t(I_t, R_t)$ function.

Let v be the per unit ordering/production cost, Equation 4.18 is replaced by:

$$l_t(I_t, 0) = v \ \max(S_t - I_t, 0) \tag{4.25}$$

This adds the cost to increase the inventory level from I_t to S_t.

4.3 Rossi et al. MIP formulation

A unified MIP modelling approach to compute near-optimal parameters for the (R, S) policy is introduced in [RKT15]. This solution has the best optimality gap according to the recent computational study of [DSRKT19]. We use it as a comparison in the experimental section of this chapter. We describe only the modelisation of the problem with backorders and penalty cost since it is the setting tackled in this work.

Rossi et al. approach can model a variety of settings: backorder or lost sales for the unmet demand; backorder per unit penalty cost, non-stockout probability and fill rate constraint for assuring the quality of service. The model can work by approximating the first-order loss function with an upper or a lower bound. Considering a random variable ω and a scalar variable x, the first order loss function is defined as $\mathcal{L}(x, \omega) = E[max(\omega - x, 0)$, where E denotes the expected value. The complementary function is defined as $\hat{\mathcal{L}}(x, \omega) = E[max(x - \omega, 0)]$. These two functions are used to compute the expected cost of a replenishment cycle. Their model is similar to Tarim and Kingsman's [TK04]. However, their linearization approach is based on [RTPH14]. In this method, the support (Ω) of the random variable ω is divided in Y regions, $\Omega_1, \ldots, \Omega_Y$. For each

compact region two parameters are computed: p_i that is the probability of ω to get a value in Ω_i ($p_i = Pr\{\omega \in \Omega_i\}$) and the conditional expectation of ω in Ω_i ($E[\omega|\Omega_i]$). When the demand is normally distributed, the linearization parameters that minimise the error depend only on the expected value and the standard deviation. In their paper, Rossi et al. provide these parameters up to 11 segments.

The MIP model to compute the lower bound for the cost of optimal plan in case of a normal distributed demand is:

$$\min_{E[TC]} = -vI_0 + v\sum_{t=1}^{T}\tilde{d} + \sum_{t=1}^{T}(K\delta_t + h\tilde{I}_t^{lb} + b\tilde{I}_t^{lb}) + v\tilde{I}_T \tag{4.26a}$$

s.t. for $t = 1, \ldots, T$

$$\tilde{I}_t + \tilde{d}_t - \tilde{I}_{t-1} \geq 0 \tag{4.26b}$$

$$\tilde{I}_t + \tilde{d}_t - \tilde{I}_{t-1} \leq M\delta_t \tag{4.26c}$$

$$\sum_{j=1}^{t} P_{jt} = 1 \qquad j = 1, \ldots, t \tag{4.26d}$$

$$P_{jt} \geq \delta_j - \sum_{k=j+1}^{t} \delta_k \tag{4.26e}$$

$$\tilde{I}_t^{lb} \geq \tilde{I}_t \sum_{k=1}^{i} p_k - \sum_{j=1}^{t}(\sum_{k=1}^{i} p_k E[Z|\Omega_k])P_{jt}\sigma_{d_{jt}} \qquad i = 1, \ldots, Y \tag{4.26f}$$

$$\tilde{B}_t^{lb} \geq -\tilde{I}_t + \tilde{I}_t + \sum_{k=1}^{i} p_k - \sum_{j=1}^{t}(\sum_{k=1}^{i} p_k E[Z|\Omega_k])P_{jt}\sigma_{d_{jt}} \quad i = 1, \ldots, Y \tag{4.26g}$$

$$\delta_t \in \{0, 1\} \tag{4.26h}$$

$$P_{jt} \in \{0, 1\} \qquad j = 1, \ldots, t \tag{4.26i}$$

(4.26j)

the model uses the same notation used in Section 2.2, with the addition of $\tilde{I}_t$ is the expected closing inventory level in period t, P_{jt} is a binary variable which is set to one if and only if the most recent inventory review before period t was carried out in period j, Z is a standard normal variable, and σd_{jt} that is the standard deviation of the random variable d_{tj}. $\tilde{I}_t^{lb}$ and $\tilde{B}_t^{lb}$ are lower bounds respectively for the true value of $E[\max(I_t, 0)]$ and $E[-\min(I_t, 0)]$. Appendix 8.1 contains a list of all the symbols used.

This model computes the policy parameters and a lower bound of the expected cost. They provide a model to compute an upper bound of the policy expected

cost as well. Equations 4.26f has to be replaced with:

$$\tilde{I}_t^{ub} \geq \tilde{I}_t \sum_{k=1}^{i} p_k + \sum_{j=1}^{t}(e_Y - \sum_{k=1}^{i} p_k E[Z|\Omega_k])P_{jt}\sigma_{d_{jt}} \quad \begin{matrix} i = 1, \ldots, Y \\ t = 1, \ldots, T \end{matrix} \tag{4.27}$$

and Equation 4.26g with:

$$\tilde{B}_t^{ub} \geq \tilde{I}_t + \sum_{k=1}^{i} p_k + \sum_{j=1}^{t}(e_Y - \sum_{k=1}^{i} p_k E[Z|\Omega_k])P_{jt}\sigma_{d_{jt}} \quad \begin{matrix} i = 1, \ldots, Y \\ t = 1, \ldots, T \end{matrix} \tag{4.28}$$

where e_Y is the maximum approximation error associated with the piecewise linearization of the standard normal function in Y regions. These two equations are the upper bounds respectively for the first order loss function and for its complementary.

4.4 Experimental results

In this section, we evaluate empirically the algorithms presented in this chapter. This section focuses on:

- Understanding the impact of K-convexity on the computational performances of the (s, S) policy computation. This property is well-known since the 1960s; however, no computational study is present in the literature that assesses its computational contribution. In some recent works, the algorithm is deployed without using it, e.g. [XRMBT18]. The authors of that study decided not to use K-convexity because it makes the DP model not solvable by general purpose SDP libraries such as the Java Stochastic Dynamic Programming Library[1]. As the experiments here shows, the impact of this property is a game-changer for the computational effort required.
- Evaluating the scalability of the method proposed in Section 4.2 and the impact of the techniques introduced to speed up its performances. Being able to solve real-world instances in a reasonable time is a key factor for the applicability of a technique.
- Compare the quality of the policy computed by the new method with the one computed by the MIP formulation of Section 4.3. We assess the

[1] https://gwr3n.github.io/jsdp/

policies by comparing their optimality gap with respect to the cost-optimal policy.

- Compare the accuracy of the expected cost of the presented methods. The accuracy is evaluated by comparing the expected cost to the simulated one.

The algorithms compared herein are:

- **sS-SDP**, the SDP technique described in Section 4.1.
- **sS-SDP-Kconv**, the SDP technique deployed using the K-convexity property. We consider this algorithm to be the current state-of-the-art in computing optimal (s, S) policy parameters.
- **RS-SDP**, the SDP technique described in Section 4.2.
- **RS-SDP-Opt**, the previous technique enhanced with memoisation, filtering and binary search of the optimal order-up-to-level.
- **RS-MIP-LB6** and **RS-MIP-LB10**, the MIP model of Section 4.26 that computes a lower bound of the optimal cost. The linearization of the first order loss function is done respectively with 6 and 10 breakpoints that minimise the approximation error.
- **RS-MIP-UB6** and **RS-MIP-UB10**, the upper bound model with 6 and 10 breakpoints.

All the experiments are computed on Intel®Xeon®CPU E5620 @ 2.40GHz. The methods are coded in Python 3.6, and Gurobi Optimizer 9.0 is used as MIP solver.

4.4.1 SDP comparison analysis

In the experimental studies of this thesis, we use adaptations of the set of instances originally proposed by [Ber72]. These instances widely used in the literature, e.g. [RTHP08, DSKTR16, XRMBT18].

In this experiment, we assume a Poisson distributed demand. Poisson demand has been widely used in the literature to model stochastic demand rates, e.g. [Duc93]. We decide to use this instead of a normally distributed demand to avoid negative demand values and errors in the discretisation process.

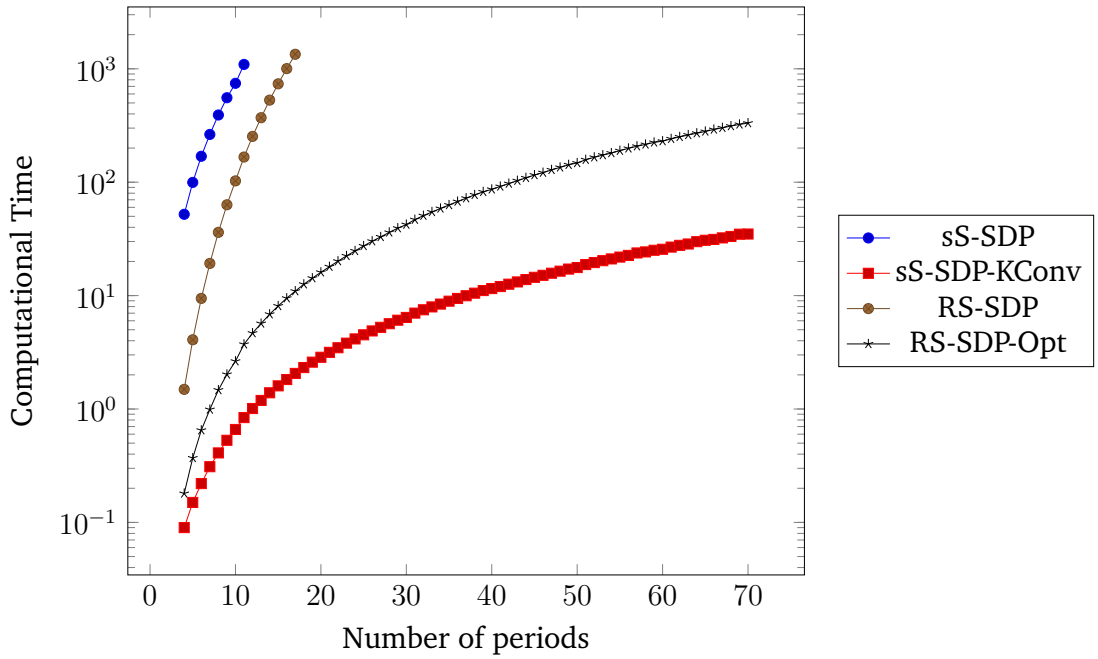

Figure 4.1: Computational time over the number of periods. Time limit 20 min.

In the first part, we use a set of parameters randomly generated, and we increase the number of periods progressively. We fix the holding cost per unit $h = 1$. The other cost parameters are drawn from uniform random variables; the ordering cost is in the range $K \in [80, 320]$ and penalty cost per unit $b \in [4, 16]$. The average demands per period are drawn by a uniform random variable with range 30 to 70. We generate 100 different instances. We replicate the experiments for periods in range 4 to 70. The values of the cost parameters are consistent with the relevant literature surveyed in the previous chapter and provide a challenging benchmark for the algorithms presented.

Figure 4.1 shows the logarithm of computational time of the four SDP algorithms. Between the two basic SDP the (R, S) one slightly outperform the (s, S). This is due to a smaller state space to explore; even if the computation of the immediate cost for the first model is more complex. We can see that the impact of the K-convexity property is crucial to compute (s, S) policy parameters in a reasonable time. The standard SDP grows exponentially, exceeding the time limit at instances with 12 periods, while the optimised one can solve instances up to 70 periods in less than a minute. The computational time seems to grow linearly with the instance size. A similar effect can be seen in the (R, S) SDP, where the basic version can solve only instances up to 17 periods. The impact of the optimisation techniques greatly affect the performances of the solution. For

the sake of brevity, we grouped all the optimisation techniques for the (R, S) SDP in a single solution. We tested the contribution of each of them individually and in pairs: the binary search has the strongest and less instance dependent impact, the memoisation follows with a considerable impact especially in larger instances, the impact of the cycle length filtering is neglectable for smaller instances, but it provides a good speedup for longer planning horizons. All the techniques (individually and in pairs) contribute to the improvement of the computational performances.

In the second part, we aim to evaluate the quality of the computed policy and check the accuracy of the expected cost. The metrics we consider in this section are two ratios: the optimality gap and the expected cost error. The optimality gap is the estimated extra-cost of using the computed policy instead of the cost-optimal one for a particular problem. It is computed as:

$$\text{Optimality gap} = \frac{\text{Policy cost} - \text{Optimal cost}}{\text{Optimal cost}} \tag{4.29}$$

A better policy exhibits a lower optimality gap. It can be used as an estimate of the inventory cost of deploying a non-optimal system. The optimal policy for the setting with zero review cost is the (s, S) as Scarf proves in [Sca59]. The policy cost is computed by simulating the inventory control 100 000 times and averaging the cost; all the policies are simulated on the same instances. We can consider the simulated cost as a close approximation to the real one. This is a normal practice in the literature(e.g. [DSRKT19]) also because some of the techniques do not provide an expected cost. An accurate expected cost can be used to compute the optimality gap as well.

The expected cost error measures the accuracy of the expected cost. It can be computed as:

$$e_{cost} = \frac{|\text{Expected cost} - \text{Policy cost}|}{\text{Policy cost}} \tag{4.30}$$

A low expected cost error means that the algorithm computes better estimations of the real cost of the policy.

The results of the comparisons are shown in Figure 4.2 and 4.3. The first plot shows the optimality gap of the (R, S) SDP compared to the optimal policy for both the expected and simulated cost. We can see that the extra inventory cost of deploying an (R, S) policy instead of an (s, S) one increases with the number of periods for instances up to 20 periods. It then becomes more or less stable

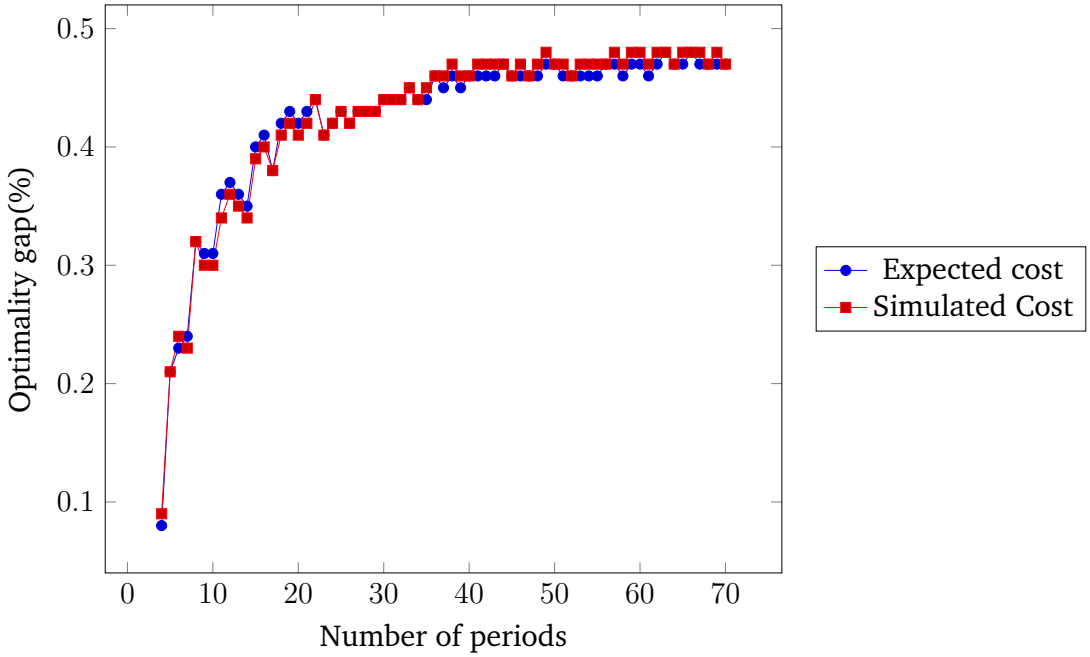

Figure 4.2: Optimality gap of the expected cost and simulated cost for the (R, S) SDP over the number of periods.

on values lower than a half per cent. This value is particularly low, especially considering the reduction of nervousness of the inventory provided by the usage of an (R, S) policy. In the next section, we compare it with the state-of-the-art in (R, S) policy parameters computation.

Figure 4.3 shows that the accuracy of the expected cost is similar for both approaches. The fluctuations are caused by a few simulated instances that exhibit extreme demands. This is why both solutions present similar gaps. When the number of periods increases, the effect of unexpected demand is mitigated, and the overall accuracy is stable between 0.017% and 0.03%. It is interesting to see how the estimate for the (R, S) policy is slightly more inaccurate than Scarf's SDP. The reason is that the (R, S) SDP considers possible negative orders in the estimation of the expected cost. While this is highly unlikely, it still has a small measurable impact on the expected cost estimation.

4.4.2 Comparison with Rossi et al. Model

In this experiment, we consider a normally distributed demand to use [RKT15] models with the linearization parameters available in [RTPH14]. We repeated

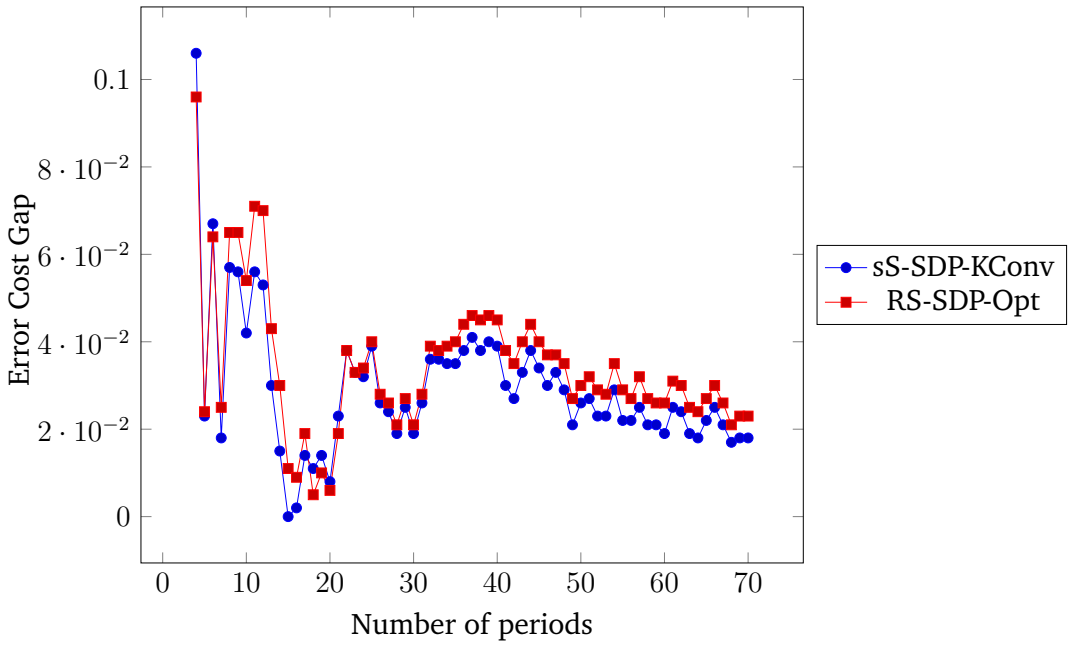

Figure 4.3: Expected cost error for the (s, S) SDP and the (R, S) SDP

the same experiments of the previous section for three standard deviations of the demand, $\sigma \in [0.1, 0.2, 0.3]$. These values are the most used in the literature when dealing with normally distributed demand, e.g. [TK04, TS08, DSRKT19]. Using normal demand introduces discretisation and bounding errors since we need a non-negative integer demand. Nevertheless, it allows analysing the performances with a higher level of uncertainty. For the sake of readability, we present only the most significant plots. All the remaining ones are available in Appendix A.

Figures 4.4, 4.5 and 4.6 show the optimality gap computed using the simulated cost for the three standard deviation. The first clear observation is that the MIP model with upper bounds computes worse policies compared to the lower bound model. For this reason, we exclude them from the next plots to focus on the best competitors. The SDP algorithm outperforms the MIP formulation in all the situations, and its optimality gap is more stable. For all the algorithms, the optimality gap increases when the uncertainty increases. This is an intuitive result, the (s, S) policy reviews the inventory in every period; it can react in a faster way to unexpectedly high demand, that is more likely when the standard deviation is higher. Another piece of information we can gain from these experiments is that the optimality gap of the SDP has smaller fluctuations compared to the other techniques. This can be seen in Figure 4.7, where we analyse the variance of the optimality gap for $\sigma = 0.3$. The variance of the

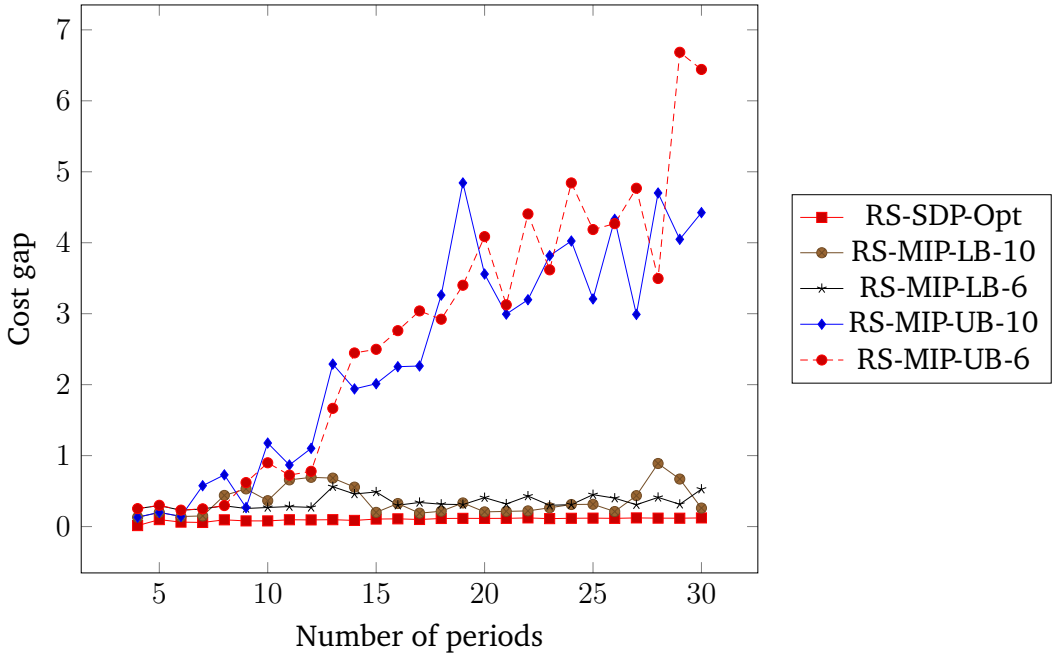

Figure 4.4: Optimality gap of the simulated cost over the number of periods for $\sigma = 0.1$

optimality gap of the SDP is considerably smaller compared to the MIP solution.

Not only does the SDP technique have a lower optimality gap, but also its expected cost is more accurate compared to the state-of-the-art. As we can see in Figure 4.8, the expected cost of the SDP is comparable with the Scarf's solution and clearly outperforms the competitors.

Finally, we compare the computational time. We restrict the solvers to use a single thread, so all the techniques can use the same hardware resources. Figure 4.9 and Figure 4.10 show the average computational time for instances with different coefficient of variation of the demand, respectively $\sigma = 0.1$ and $\sigma = 0.3$. The MIP based algorithms are faster for small-medium instances and with a smaller uncertainty of the demand; while the SDP outperforms them for longer time horizons. This is an indicative comparison since the SDP algorithm, and the solvers are developed in different languages. Moreover, MIP solvers are optimised to use multithreading, while in the SDP it has to be manually coded. The MIP based techniques are marginally affected by the increase of the average demand since they only consider the distribution parameters; while this has an impact on the size of the state space in SDP solutions.

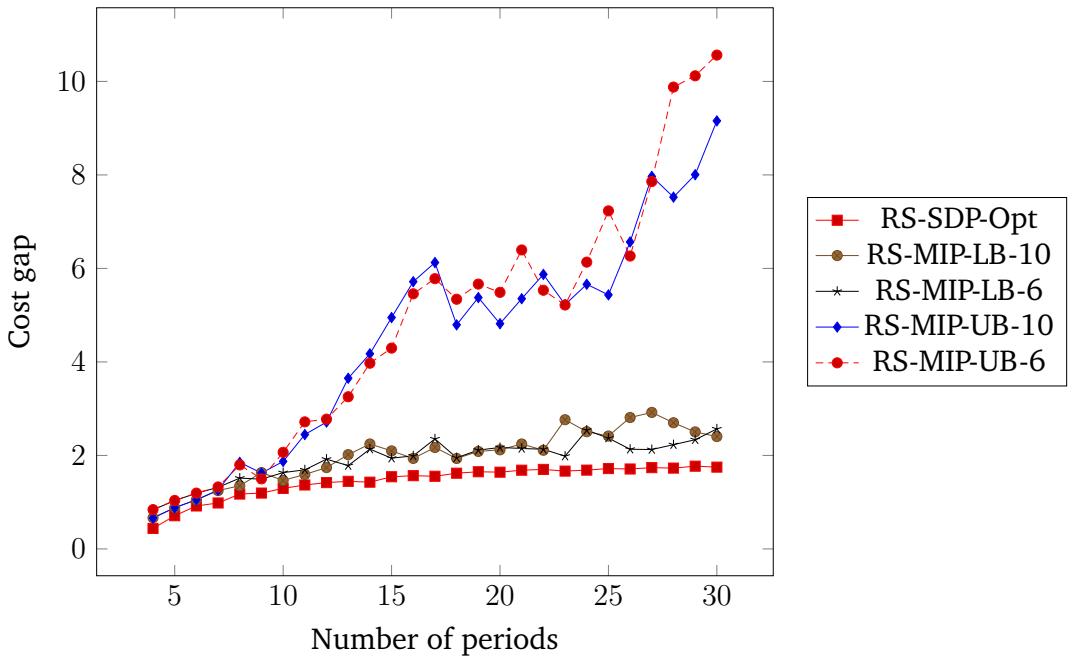

Figure 4.5: Optimality gap of the simulated cost over the number of periods for $\sigma = 0.2$

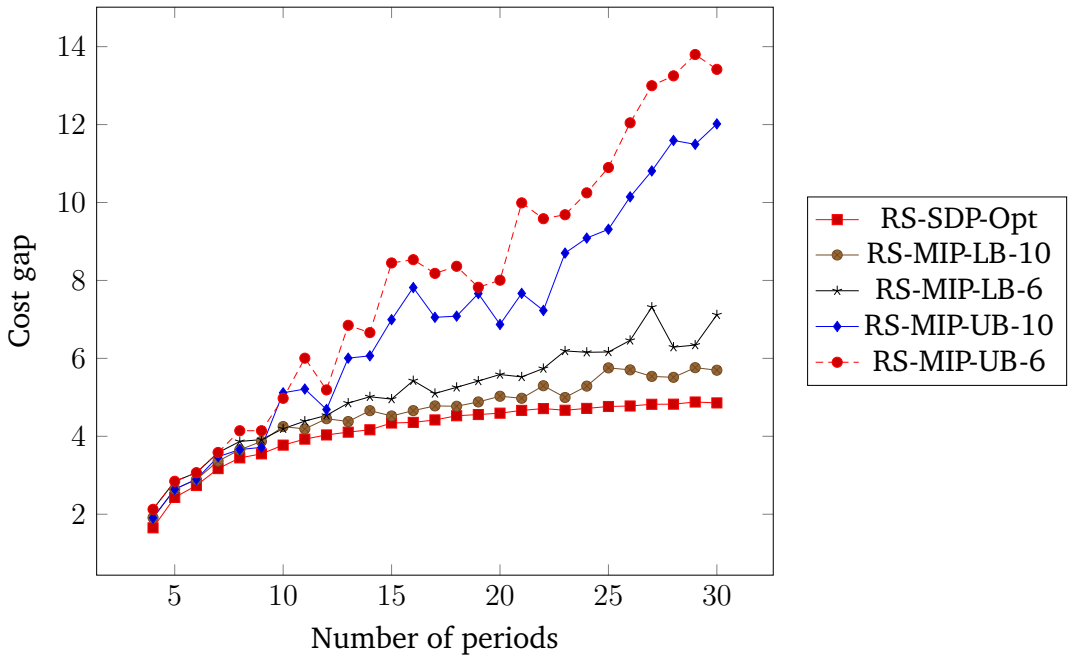

Figure 4.6: Optimality gap of the simulated cost over the number of periods for $\sigma = 0.3$

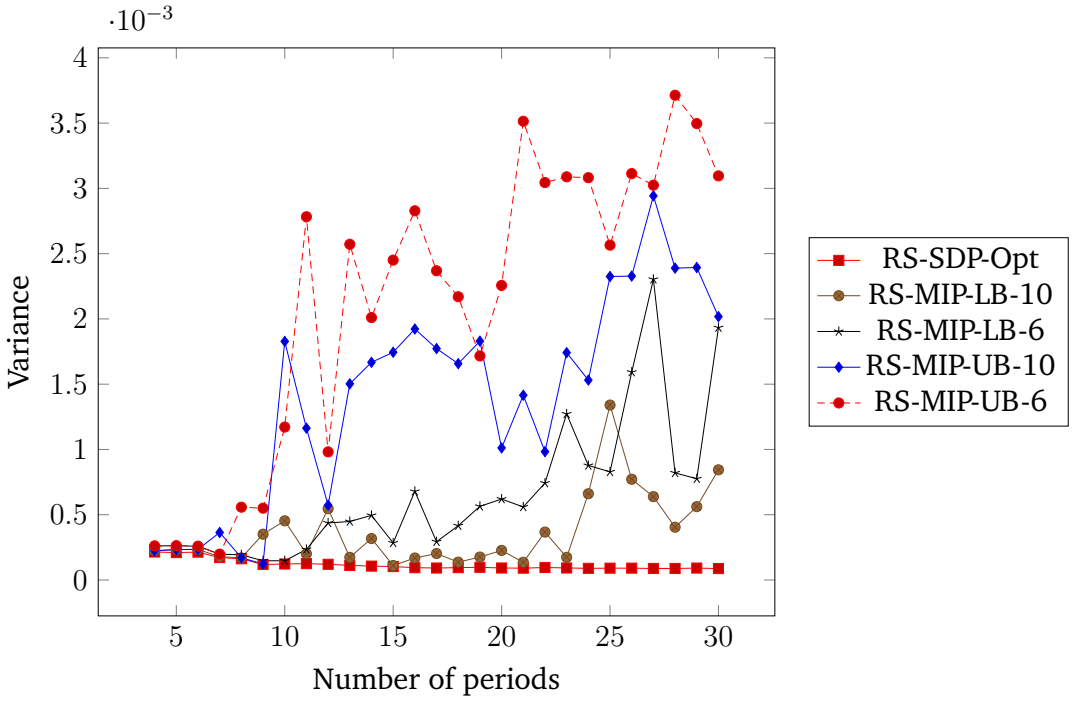

Figure 4.7: Variance of the simulated cost optimality gap over the number of periods for $\sigma = 0.3$

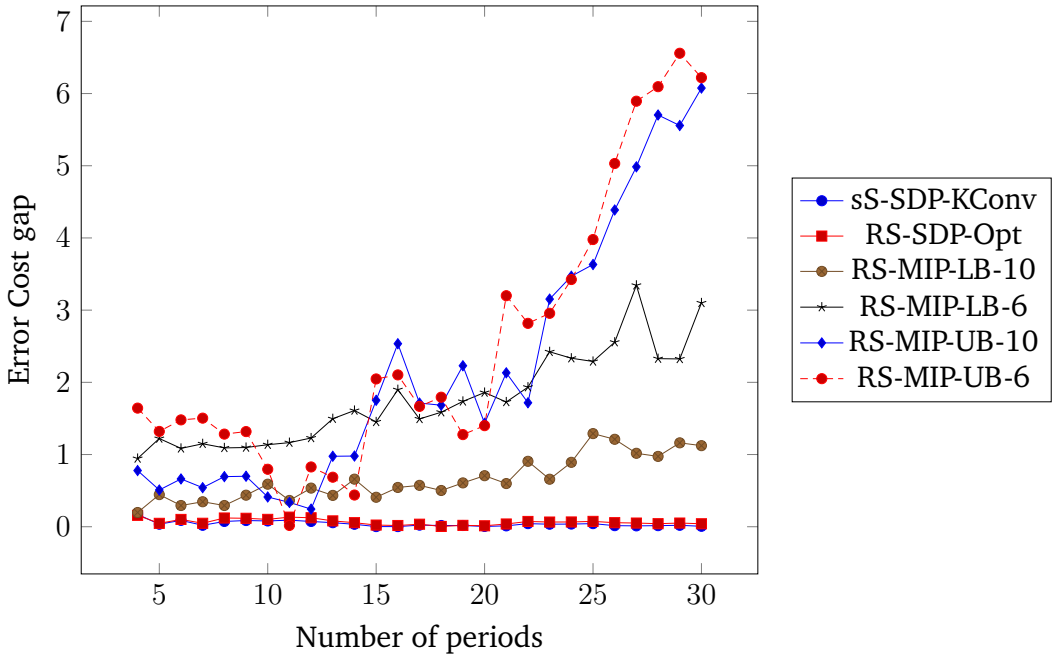

Figure 4.8: Expected cost error over the number of periods for $\sigma = 0.3$

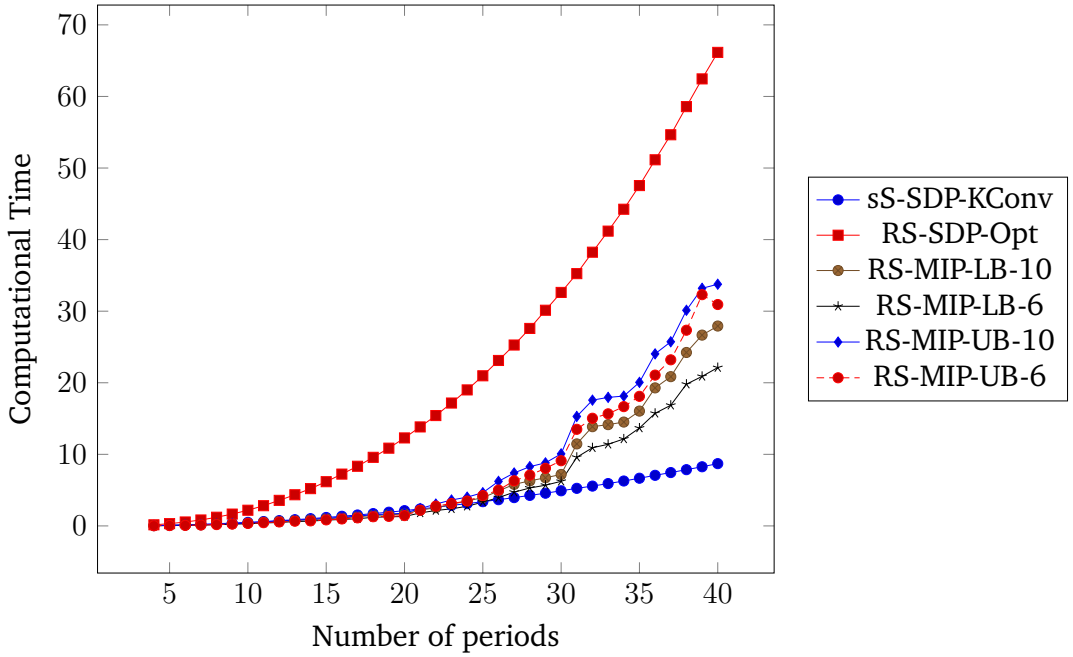

Figure 4.9: Computational time in seconds for $\sigma = 0.1$

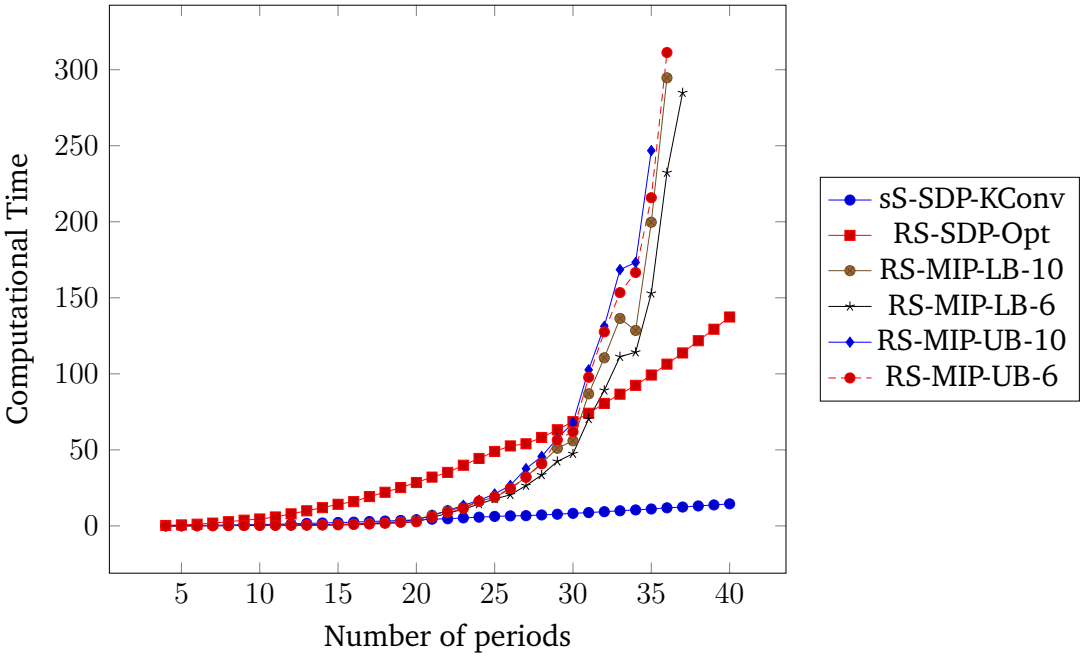

Figure 4.10: Computational time in seconds for $\sigma = 0.3$

4.5 Conclusions

In this chapter, we describe the widely known SDP for computing the optimal (s, S) policy parameters under Scarf's setting, widely used in the next chapters. We introduce the K-convexity property, and we provide instructions for its deployment. We then present the main contribution of this chapter, the first purely SDP algorithm to compute optimal (R, S) policy parameters. Three different approaches are deployed to improve its performances: a memoisation approach used to avoid recomputations, a filtering technique to prune the sub-optimal search space and a binary search on the optimal order-up-to-level. This approach strongly differs from the extensive literature on the topic that Section 3.4.4 surveys. The approach is optimal, and it does not make any assumption on the demand type. We provide the algorithm's pseudocode to ease the implementation of the solution; in Python, it can be coded in less than 50 lines of code.

Interesting findings are presented in the extensive computational study, two of them regarding techniques already available in the literature:

- We quantify the improvement that the K-convexity brings to the SDP computational time. To the best of our knowledge, this analysis is not present in the literature, and some recent works deploy the technique without using the K-convexity property.
- The MIP models presented by [RKT15] performs considerably better using the lower bound approximation compared to the upper bound one.

Regards our technique and the optimal static-dynamic policy, we discovered that:

- The computational complexity of the technique is particularly high. It can solve only small-medium instances in a reasonable time. However, the optimisation techniques deployed reduce the computational effort considerably, making it able to solve real-world size instances.
- The approach has performances similar to Scarf's SDP. The optimality gap is stable, and it seems to converge for larger size instances.
- The expected cost computed by the policy is particularly small. The expected cost error is minimally higher than the (s, S) one. We justify this difference by considering the extra holding cost not considered when

the order-up-to-level is smaller than the current inventory during a review period.

- Compared to the best algorithm for the determination of (R, S) analysed in the recent computational study of [DSRKT19], the policies computed by the solution presented herein have a lower cost, a more accurate expected cost and a stable optimality gap.
- The SDP solution is competitive with the state-of-the-art in terms of computational effort required to compute a solution. Additionally, with memoisation, the SDP does not require the convolution of the demand of the replenishment cycles.

Overall, the performances and the easy implementation of this new technique mean that it fills a gap in the literature.

Chapter 5

Branch-and-Bound solution for (R, s, S)

In this chapter, we propose an efficient BnB approach for computing optimal (R, s, S) policy parameters for the stochastic lot sizing with non-stationary demand, backlogging of excessive demand, linear holding and penalty cost, fixed ordering and review cost.

The (R, s, S) policy is a generalisation of the (s, S) and (R, S). In absence of a review cost, the policy is equivalent to the (s, S) one. If we fix $s = S - 1$ an order is placed at every review moment, so it becomes equivalent to the (R, S) policy.

According to [SPT16], this policy is widely used in practice. However, the computational effort for determining its optimal parameters is considered prohibitive. Therefore, in many practical applications, simpler techniques are deployed even if less cost performing. This policy becomes particularly interesting when we take into account a system cost associated with reviewing the inventory. In this case, the (s, S) policy is not cost-optimal anymore. The literature surveyed in Section 3.4.5 does not contain any ad hoc method to compute (R, s, S) parameters for such problem setting.

We present a first algorithm that extends Scarf's SDP presented in Section 4.1 to compute optimal (R, s, S) parameters. This approach is a simple brute force application of the SDP for all the possible replenishment cycles. We consider it as the baseline and as the current state-of-the-art.

The main solution proposed in this chapter is based on a hybrid approach; it

exploits tree search to compute the optimal replenishment cycles and stochastic dynamic programming to compute s and S levels for a given replenishment cycle. To speed up the computations, we applied a BnB technique. Bounds, computed using DP, are generated via an efficient dynamic programming algorithm; these bounds allow the BnB to prune of up to 99.5% of the search tree without compromising optimality.

The next section describes the baseline technique. The core technique of this chapter is presented in Section 5.2. Section 5.3 contains an extensive computational study of the techniques introduced in this chapter. Finally, Section 5.4 concludes.

5.1 Baseline

In this section, we provide a simple technique to compute the optimal (R, s, S) policy parameters. Since no technique to compute optimal parameters in the presence of stochastic non-stationary demand is available in the literature, it can be considered state-of-the-art. Moreover, it constitutes the basis of the BnB technique presented in the next section. This approach is based on the SDP used to compute optimal parameters of the (s, S) policy described in Section 4.1.

If the review cycles are fixed, the (R, s, S) is reduced to the well known (s, S) policy. It is possible to model this problem as the SDP formulation and solve it to optimality. The idea behind this approach is to compute the optimal (s, S) policy for all the possible review plans; a brute force application of the SDP.

Consistently with the models presented in Section 2.2, we represent the replenishment moments with the binary variables γ_t, for $t = 1, \ldots, T$, which takes value 1 if a review is placed in period t and 0 otherwise. R_t is the number of periods before the next review moment. So,

$$R_t = min(\{l | t < l \leq T, \gamma_l = 1\}) - t \tag{5.1}$$

To be consistent with this formulation, we assume $S_i = -\infty$ and $s_i = -\infty$ if $\gamma_i = 0$. We assume $Q_t = 0$ if $\gamma_i = 0$. Therefore, no order will be placed outside a review moment. The optimal (R, s, S) policy for our problem is represented by the parameters γ_t, s_t, S_t that minimize the expected total cost. The values of δ_t,

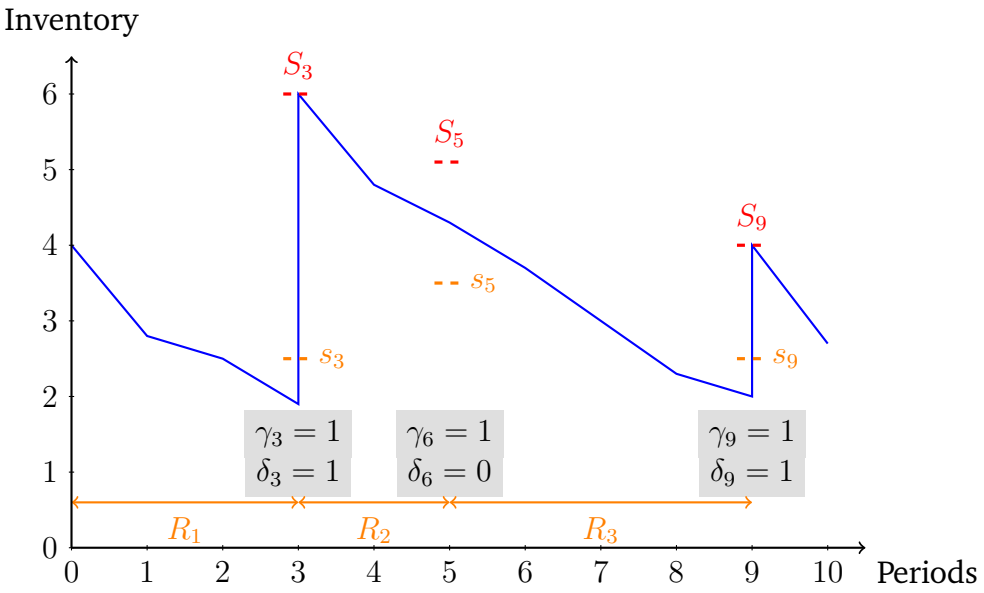

Figure 5.1: Example of a (R, s, S) policy.

the binary variable that takes value 1 if an order is placed on period t, are not fixed a priori since in this policy the decision of placing an order is taken after observing the previous demand. Figure 5.1 shows an example application of this policy. A review is scheduled at period 5 ($\gamma_5 = 1$), since the inventory level is higher than s_5 the order is not placed($\delta_5 = 0$). All the non-specified values of δ_i and γ_i are equal to zero.

We consider an arbitrary review cycle plan. Therefore, γ_t is used as parameter and not as a decision variable. Each stage of the dynamic programming formulation represents a period of the planning horizon.

The structure of the SDP is equivalent to the one presented in the previous chapter. The only difference is in the functional equation. Let $C_t(I_{t-1})$ denote the expected total cost of an optimal policy over periods $t, \ldots, T$ associated with state $N_t = I_{t-1}$. Then, $C_t(I_{t-1})$ can be written as:

$$C_t(I_{t-1}) = \min_{0 \leq Q_t \leq M\gamma_t} (f_t(I_{t-1}, Q_t) + E[C_{t+1}(I_{t-1} + Q_t - d_t)]) \tag{5.2}$$

where M is a sufficiently large number. The constraint $0 \leq Q_t \leq M\gamma_t$ forces the order quantity to 0 when the period is not a review moment.

$C_1(I_0)$, where I_0 is the initial inventory, contains the expected cost for the optimal (s, S) policy associated with the γ assignment.

Let $\hat{C}_1(I_0)$ represent the expected total cost of the optimal (R, s, S) policy, given

the initial inventory level I_0 at period 1. We can define it as:

$$\hat{C}_1(I_0) = \min_{\gamma_1,\dots,\gamma_T}(C_1(I_0)) \tag{5.3}$$

evaluating the optimal (s, S) policy for all the possible assignments of $\gamma_1, \dots, \gamma_T$ leads to the optimal (R, s, S) policy.

5.1.1 Time complexity

The time complexity of the K-convexity (s, S) SDP is $\mathcal{O}(D^2T^2)$, as states Section 4.1.4. The baseline computes it for all the possible review plans. A review plan is an assignment of the γ_t binary variables. All the possible assignments of T binary variables are 2^T. The overall complexity of the algorithm is:

$$\mathcal{O}(2^T D^2 T^2) \tag{5.4}$$

Due to the exponential complexity, the algorithm can solve only small instances.

Example

We shall consider a simple example in detail, to show how, in practice, it is possible to apply the procedures described herein. A single problem over a 3-periods planning horizon is considered. We assume an initial null inventory level and a Poisson distributed demand for each period with averages $\overline{d} = [20, 30, 40]$. We consider an ordering cost value $K = 30$, a review cost $W = 10$, holding cost and penalty cost respectively $h = 1$ and $b = 10$ per unit per period.

The algorithm's goal is to compute the replenishment moments $\gamma = [\gamma_1, \gamma_2, \gamma_3]$ that minimise the expected cost of the policy. Table 5.1 shows the expected cost of each (s, S) policy computed with different review periods. The optimal solution is $\gamma = [1, 0, 1]$ with expected cost 142.7.

5.2 Branch-and-Bound

In this section, we present a branch-and-bound technique used to compute cost-optimal parameters for the (R, s, S) policy. The search tree is defined in Section 5.2.1. The subproblems associated to the nodes are defined in Section 5.2.2.

Table 5.1: Expected cost for the 3-periods numerical example for each possible replenishment plan.

γ_1	γ_2	γ_3	Expected cost
0	0	0	1600.0
0	0	1	751.8
0	1	0	304.7
0	1	1	302.0
1	0	0	185.0
1	0	1	**142.7**
1	1	0	153.1
1	1	1	150.4

Section 5.2.3 introduces the pruning condition and lower bound computed with dynamic programming. Finally, Section 5.2.4 presents the nodes resolution process.

5.2.1 Search tree

The approach previously presented repeats multiple times the same computations. Consider two assignments of γ: γ^a and γ^b. Let C_t^a and C_t^b be the respective expected costs computed with SDP.

Proposition 1. *If there exists a period j for which:*

$$\gamma^a = \gamma^b \quad \forall t \in [j, n] \tag{5.5}$$

then:

$$C_t^a(I) = C_t^b(I) \quad \forall t \in [j, n], \forall I \tag{5.6}$$

Since the SDP is computed backwards, the cost of each period t depends on the periods j with $j > t$. If two different review plans share a common final part of length l, then the last l periods have the same expected costs. In the baseline approach presented above, these computations are repeated multiple times since the SDP state space is computed entirely for all the assignments.

The BnB goal is to avoid these recomputations and to find the review plan with the minimum expected cost. In the branching, a γ_t value is fixed to 1 or 0. The search tree has $T + 1$ levels, the branching in the root fix the value of γ_T. At level l the branching involves the variable γ_{T-l+1}. The path from the root to a node in level l represents a fixed assignment of the suffix $[\gamma_{T-l+2}, \ldots, \gamma_T]$. A

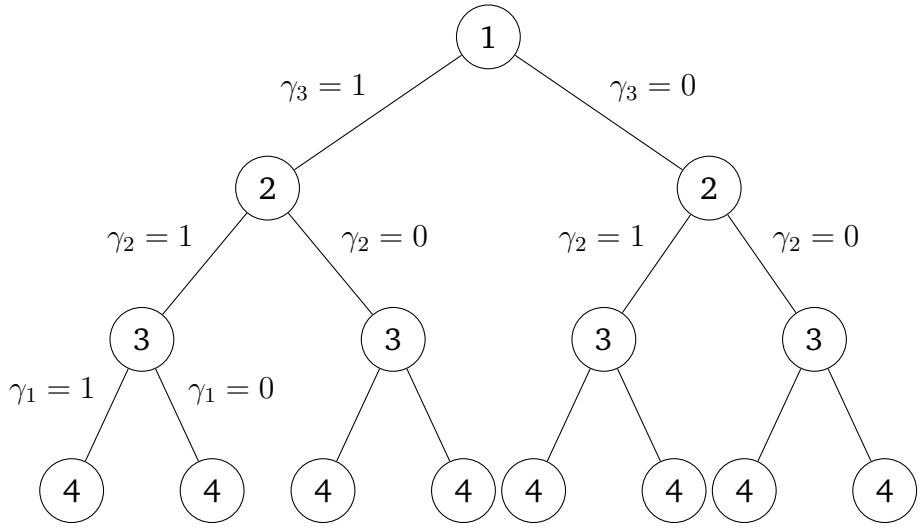

Figure 5.2: Search tree associated with a 3-periods instance, the nodes contains the level number.

leaf node represents a complete assignment of the γ values. The search tree is visited in a depth-first search (DFS).

The baseline computes all the periods for all the possible combinations; the total number of replenishment cycles is 2^T. The total number of computed periods is $T \times 2^T$. The number of nodes in a complete binary tree is $2 \times 2^T - 1$. So, the binary tree restructuring should reduce the computational effort by a factor of $T/2$, reducing the overall complexity to:

$$\mathcal{O}(2^T D^2 T) \tag{5.7}$$

Example

Figure 5.2 shows the search tree of a 3-periods problem, like the example presented in the previous section.

5.2.2 Subproblems

Given the period t and the partial assignment of a suffix of the review moments $[\gamma_t, \ldots, \gamma_T]$, the problem at a node is to find the $[\gamma_1, \ldots \gamma_{t-1}]$ that minimize the expected cost of the optimal policy. We denote this problem as BnB-SDP(t,$[\gamma_t, \ldots, \gamma_T]$). For each subproblem, using Equation 4.2, we can compute the expected cost of the optimal policy starting at period t with inventory level i; this is possible because all the review moments that take place after period t

are fixed and thanks to the dynamic programming stage structure presented in Section 4.1.

5.2.3 Bounds and pruning

If it is possible to prove that all the solutions present in the subtree rooted in a node are not optimal, we can prune the tree without compromising the optimality.

Proposition 2. *Given a fixed assignment of γ:*

$$\min_I(C_t(I)) \geq \min_I(C_{t-1}(I)) \tag{5.8}$$

Considering the functional equation (Equation 5.2), C_t is equal to the expected value of C_{t+1} plus some non-negative costs. Then, the minimum cost in each stage is monotonically increasing while descending the tree.

Let $\bar{C}$ be an upper bound on the expected cost of the optimal policy. We store in $\bar{C}$ the expected cost of the best policy computed so far, the minimum $C_1(I_0)$ among all the leaves already computed.

Considering the subproblem BnB-SDP(t,$[\gamma_t, \ldots, \gamma_T]$), with the associated $C_t(i)$ expected costs:

Proposition 3. *If*

$$\min_i(C_t(i)) \geq \bar{C} \tag{5.9}$$

then, due to the monotonicity of the cost function (5.8):

$$\min_i(C_1(i)) \geq \bar{C} \tag{5.10}$$

finally, since the expected cost associated with a policy ($C_1(I_0)$) is part of C_1:

$$C_1(I_0) \geq \bar{C} \tag{5.11}$$

If (5.9) is true the subproblem BnB-SDP(t,$[\gamma_t, \ldots, \gamma_T]$) is not part of an optimal solution and the search tree can be pruned.

However, this pruning condition does not consider the costs faced on periods $1, \ldots, t-1$. A lower bound on the costs faced in those periods leads to a more

effective pruning.

Let $MC_t(I_t)$ represent a lower bound of the cost faced in periods $1, \ldots, t$ with a closing inventory of I_t in period t. The pruning condition (5.9) can be updated as:

$$\min_{I_t}(C_t(I_{t-1}) + MC_{t-1}(I_{t-1})) \geq \bar{C} \tag{5.12}$$

Having a bound independent from the review plan allows to compute it only once before the BnB algorithm.

The bounds can be computed with a dynamic program with stages and states equivalent to the one presented in Section 4.1 and functional equation:

$$MC_t(I_t) = \min \begin{cases} f_t(I_t, 1) + \min\limits_{j<I_t}(MC_{t-1}(j)) \\ f_t(I_t, 0) + \min\limits_{j\geq I_t}(MC_{t-1}(j)) \end{cases} \tag{5.13}$$

where I_t is the current inventory level, and $f_t(I_t, Q_t)$ is the ordering-holding-penalty cost. The bound takes the minimum value between placing an order in $t-1$ or not. In the first case, if an order has been placed then the previous period inventory level was lower or equal to the current one. In the second case, an order has not been placed on the period t, therefore the inventory level had to be higher or equal than the current one. The boundary condition is:

$$MC_1(I_1) = \begin{cases} W + K + f_1(I_1) & \text{if } I_1 > I_0 \\ f_1(I_1) & \text{if } I_1 \leq I_0 \end{cases} \tag{5.14}$$

where I_0 is the initial inventory. This dynamic program can compute the bounds in polynomial time.

5.2.4 Nodes computation

The pseudocode for the BnB procedure is presented in Algorithm 6. The resolution of each search tree node (RsS-BnB(t,$[\gamma_t, \ldots, \gamma_T]$)) involves three phases:

- **Preprocessing** - in line 1, the stochastic dynamic programming stage t is solved.
- **Pruning** - the pruning condition is evaluated in line 7; if a pruning occurs the branching phase is skipped.

- **Branching** - in lines 8 and 9, the algorithm continues with the depth first search of the tree.

Lines 3-6 involve leaf nodes. If the policy represented by the leaf is better than the best so far, the value of $\bar{C}$ is updated. The solving starts by invoking RsS-BnB($T+1$,$\emptyset$). At the end of the computations, the expected cost of the optimal policy is contained in $\bar{C}$.

The performance of the algorithm can be improved with randomisation. The algorithm always branches by assigning first $\gamma_t = 0$. If during each branching phase, we decide the order of lines 8-9 randomly, we can obtain a better solution earlier. This leads to a stronger pruning of the search tree. We evaluate the effect of randomisation in Section 5.3.1.

A more informed heuristic for selecting which branch to explore finds a near-optimal solution faster than a randomised search. The heuristic needs to decide if, given the replenishment plan for future periods, a review should be scheduled in the current period. Due to the backward computation of the SDP the expected level position at a given period is hard to estimate during the tree search. In Chapter 6, we use a (R, s, S) heuristic to precompute a near-optimal review plan. Then, we use this plan to guide the first descend of the tree. A tight upper bound on the policy cost is found at the first descend of the tree, and the search continues in similar review plans.

Algorithm 6 RsS-BnB(t,$[\gamma_t, \ldots, \gamma_T]$)

Data: the current upper bound $\bar{C}$, the $C_{t+1}(i)$ computed at the parent node, the bounds $MC(i)$.

1: Compute C_t using Equation 5.2
2: **if** $t = 1$ **then**
3: **if** $C_1(I_0) < \bar{C}$ **then**
4: $\bar{C} \leftarrow C_1(I_0)$
5: Save $[\gamma_1, \ldots, \gamma_T]$ as best so far review plan
6: **else**
7: **if** $\min(C_t(i) + MC_{t-1}(i)) \geq \bar{C}$ **then return**
8: BnB-SDP($t-1$,$[0, \gamma_t, \ldots, \gamma_T]$)
9: BnB-SDP($t-1$,$[1, \gamma_t, \ldots, \gamma_T]$)

To calculate the impact on the problem complexity, we need to evaluate the portion of the state space pruned. The computation of an average-case value requires an indicator random variable [CLRS09]. This variable represents the probability of a pruning occurring at a specific node. This is strongly dependent

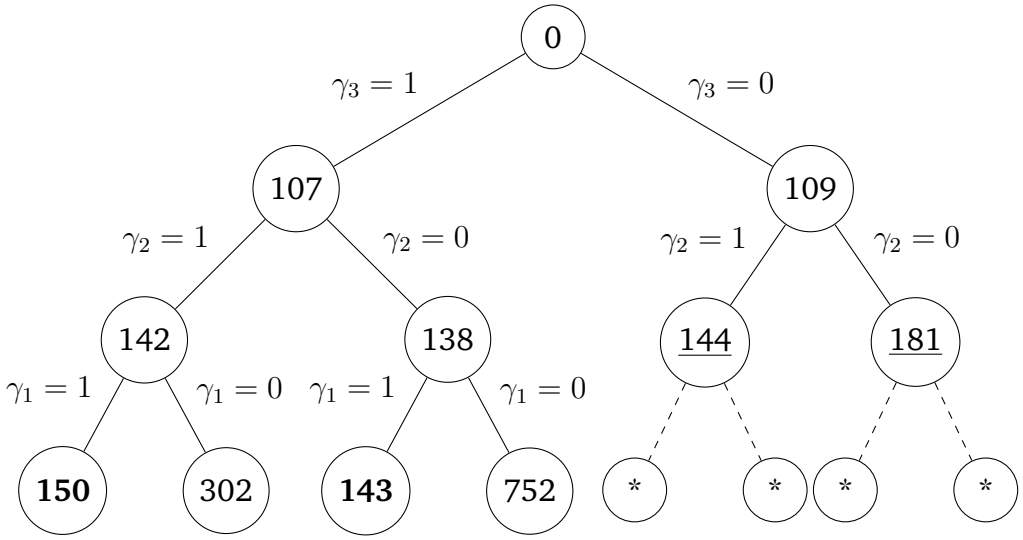

Figure 5.3: BnB technique applied to the toy problem

on the instance parameters (cost factors, demand average and type, instance size). To the best of our knowledge, it is not possible to compute such an estimate, and its computation is beyond the scope of this thesis. However, we measure it empirically in the experimental section. We define as pruning percentage the percentage of nodes that are proved to be not optimal by the pruning condition during the tree visit. Knowing the pruning percentage allows estimating the computational time. The empirical analysis of it enables us to understand the impact of the pruning on the state space.

Example

The search tree with the DP bounds for the example is represented in Figure 5.3. Each internal node contains the value of the pruning condition with the dynamic programming bounds (5.12). An internal node is underlined if the pruning occurs in that node. Each leaf is emboldened if it contains an improvement compared to the previous best solution, $\bar{C}$. Prune nodes are represented by an asterisk '*'.

In this example, the number of computed nodes is 10, and 4 nodes have been pruned, so the pruning percentage is $4/14 = 28.57\%$.

5.3 Experimental results

In this section, we evaluate empirically the new methods introduced in this chapter, including an assessment of the effects of branching randomisation and problem parameters (costs). We conduct two sets of experiments as follows. In Section 5.3.1, we analyse the scalability of the new approaches by increasing the number of periods until no method is able to solve the problem within a 10 hours time limit consistently. In Section 5.3.2, we fix the planning horizon to 10 and 20 periods and vary the cost parameters. For the experiments, we use three (R, s, S) policy solvers:

- **RsS-Base**, the application of the SDP technique for all the possible replenishment cycles described in Section 5.1 which we consider is the current state-of-the-art.
- **RsS-BT**, the binary tree restructure of the computations.
- **RsS-BnB**, the BnB solution introduced in Section 5.2.
- **RsS-BnB-Rand**, BnB with randomised branching.

We compare these in terms of computational time, pruning percentage and the average number of review periods. Since all the techniques presented in this chapter compute the optimal (R, s, S) policy with the same expected and simulated cost, our investigation does not involve analysis on the cost metrics. All experiments are executed on an Intel(R) Xeon E5640 Processor (2.66GHz) with 12 Gb RAM.

5.3.1 Scalability

This test aims to assess how scalable are the approaches proposed herein. The testbed is similar to the one used in Section 4.4.1, we add the review cost modelled as a random uniform variable $W \in [80, 320]$, the other cost parameters and the demand are the same. We extend the planning horizon until no technique can solve all the instances in less than one hour. For each length, we generate 100 different instances.

Figure 5.4 and Figure 5.5 show the results of this test, in the second one the y-axis is scaled logarithmically. The exponential behaviour of the solutions is evident. However, the new solution is able to solve instances almost twice as

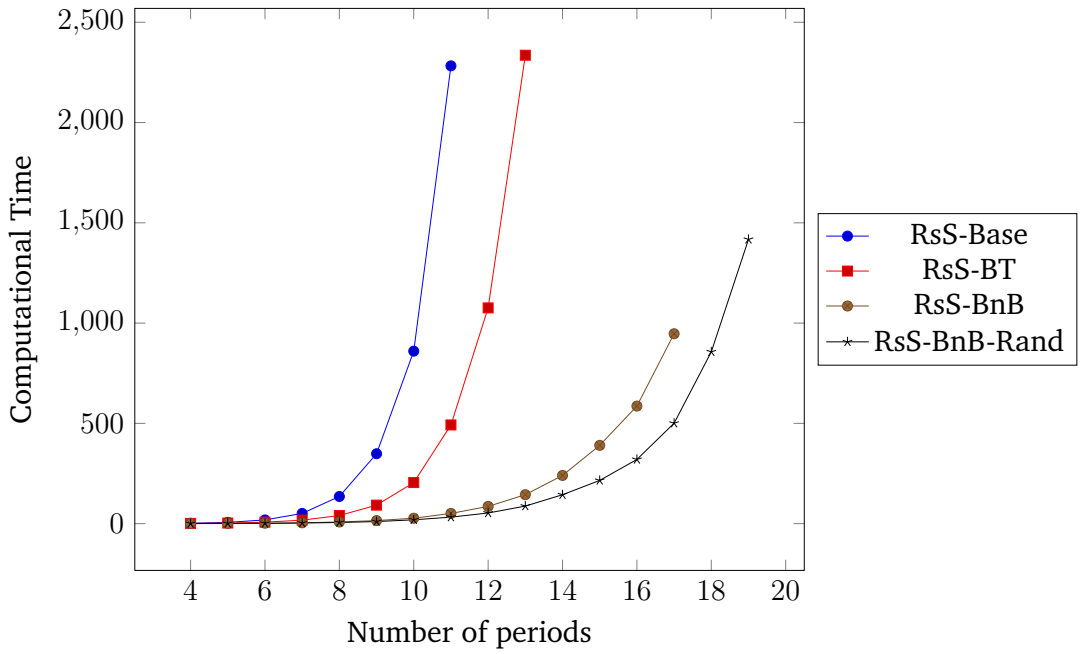

Figure 5.4: Computational time over the number of periods. Time limit 1 hour.

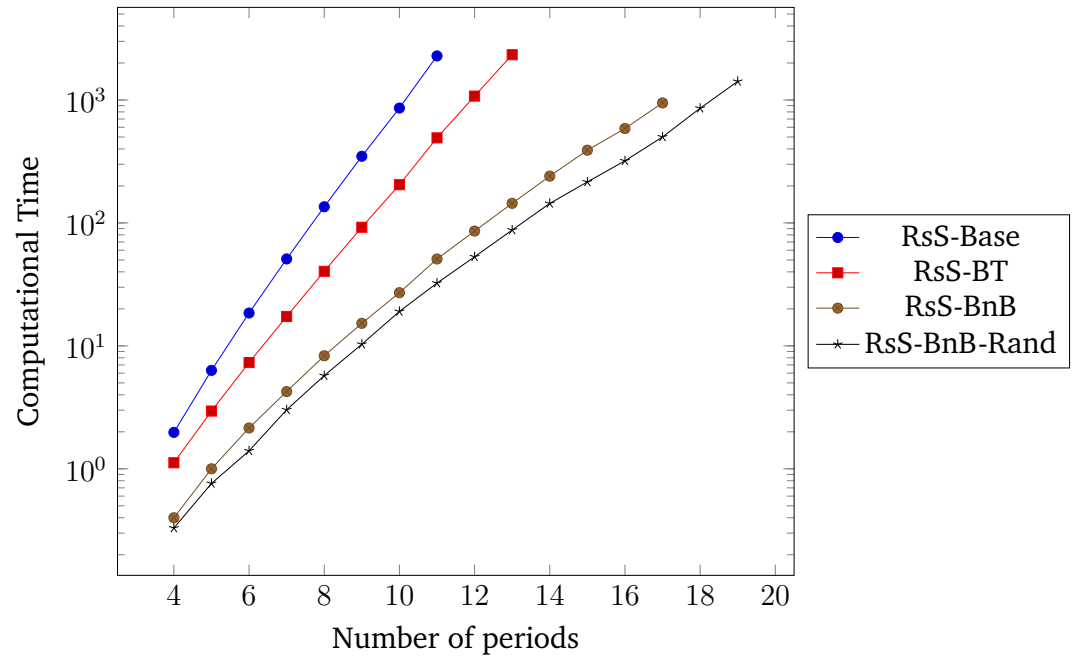

Figure 5.5: Computational time over the number of periods. Time limit 1 hour.

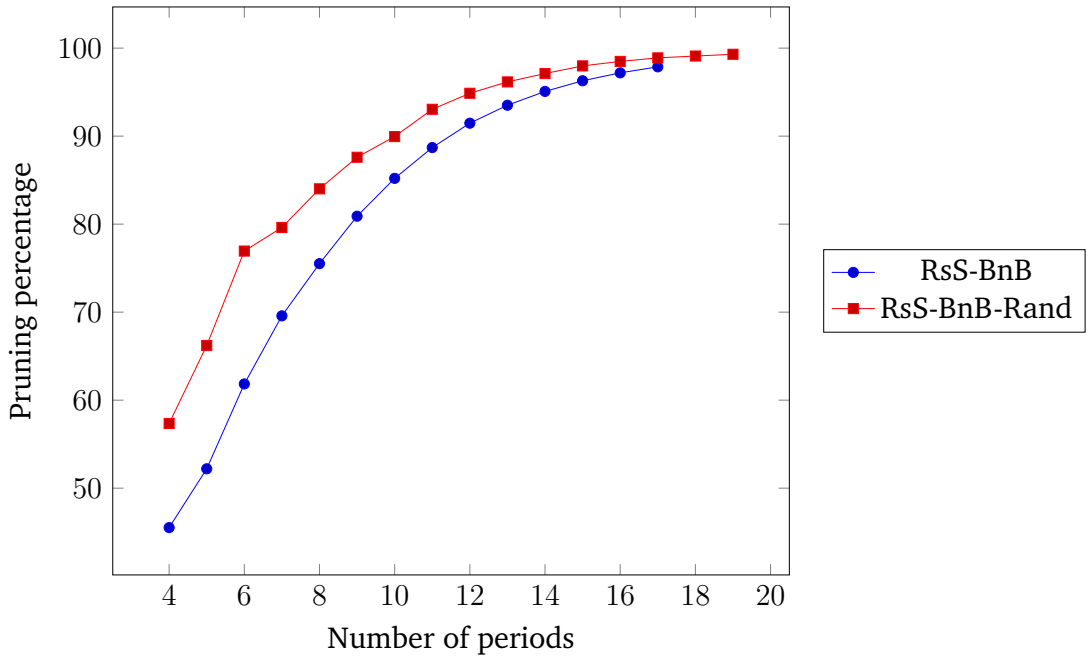

Figure 5.6: Pruning percentage over the number of periods.

big in a reasonable time. Even if the new solution has an exponential behaviour, its slope on the logarithmically scaled plot is considerably smaller than the SDP one.

Figure 5.6 shows the pruning percentage of the BnB with and without random descent of the tree. The pruning becomes more effective for longer planning horizons. Using randomness, a better solution is found earlier in the search; this allows a stronger pruning of the search tree.

5.3.2 Instance type analysis

The second set of experiments aims to investigate how the instance parameters affect the performances. For the instance type analysis, we consider a testbed which includes 324 instances. To generate the average demand values, we use seasonal data with different trends:

- **(STA)** stationary case: $\tilde{d}_t = 50$
- **(INC)** positive trend case: $\tilde{d}_t = \lceil 100t/(n-1) \rceil$
- **(DEC)** negative trend case: $\tilde{d}_t = \lceil 100 - 100t/(n-1) \rceil$
- **(LCY1)** life-cycle trend 1 case: this pattern is a combination of the first

3 trends. The first third of positive trend up to an average demand of 75, a central stationary one and the last negative third. If the number of periods is not a multiple of 3, the central period is extended.

- **(LCY2)** life-cycle trend 2 case: this pattern is a combination of INC and DEC trends. Positive trend for the first half of the planning horizon and negative trend for the second half.
- **(RAND)** erratic: $\tilde{d}_t = \lceil U(0, 100) \rceil$

all the patterns have an average demand of 50 per period. These patterns have been originally proposed in [Ber72] and are widely used in the literature, e.g. [RTHP08, DSKTR16, XRMBT18].

For the cost parameters, we use all the possible combinations of ordering cost values $K \in \{80, 160, 320\}$, review costs $W \in \{80, 160, 320\}$, holding cost fixed $h = 1$, penalty costs $b \in \{4, 8, 16\}$. We tested all the combinations of cost parameters and the six patterns presented above. We analyse the results for 10-periods and 20-periods instances.

Since the baseline is too computationally expensive, it takes approximately 45 days to solve a 20 periods instance; we replace it with an estimate in the 20-periods instances. The estimate is computed by solving 100 times the stochastic dynamic programming for different γ assignments and averaging it over all the possible assignments.

Table 5.2 and Table 5.3 give an overview of the computational time, the pruning percentage and the average number of reviews of the methods discussed in this study, respectively related to the 10 and 20-periods experiments. The computational time of the BnB solutions depends on the complexity of the SDP and the pruning efficacy.

The SDP is not strongly affected by the cost parameters. The patterns make the main difference. This is due to the maximum average demand per period being lower for the patterns STA, LCY1 and RAND. The stationary is faster to compute since its maximum is 50, the second one is the first life cycle with a maximum of 75, finally the erratic pattern. All the other patterns have a maximum of 100.

The pruning percentage gives an indication of the efficacy of the BnB. The algorithms proposed herein perform particularly well for high review costs. For instance, with 20 periods and $W = 320$ the pruning percentage reached an impressive average of 99.57% for the BnB with randomised visit of the tree,

solving one of these instances in half an hour; while the baseline is expected to solve them in a month and a half. Without randomisation, the percentage is 99.27%, so it has to compute twice the nodes compared to the randomised version. We can notice that the penalty cost affects the performances as well. In this case, a higher penalty cost leads to a weaker pruning.

We also assess the average number of review moments of the optimal policy. The results show that these reduce when the ordering and the review increase. Additionally, a higher penalty cost leads to more frequent reviews; this reduces the probability of having an excess of demand and mitigate the uncertainty of the inventory level. We observe that the decreasing pattern requires fewer review periods than the others, due to its decreasing final tail that reduces the number of orders needed.

On average, the best solution proposed herein outperforms the baseline by 40 and 820 times, respectively on 10 and 20-periods instances.

5.4 Conclusion

In this chapter, we present the first algorithm to compute optimal (R, s, S) policy parameters. This policy has a high practical value, but the computation of optimal or near-optimal parameters has been considered extremely difficult. The technique described herein is a hybridisation of branch-and-bound and stochastic dynamic programming, enhanced by ad-hoc bounds computed with dynamic programming. We improve the method using a randomised depth-first visit of the search tree.

We conduct an extensive numerical study. We first investigate the scalability of the technique at increasing time horizon, analysing both computational time and the efficacy of the pruning technique. We then test the performances of the method for different cost parameters. Our technique performs better for low penalty cost and high review cost. On 20 periods instances, our approach beats the baseline by three orders of magnitude.

This technique opens up multiple research directions on the determination of (R, s, S) policy parameters. It can lead to new optimal solutions for the same problem, and it can be improved with tighter bounds. It is also useful for computing optimality gaps of new heuristics. The next chapter explores some of these possible directions and improves the pruning performances by using

Table 5.2: Computational times (in minutes), pruning percentage and number of reviews for 10-periods instances

		Computational time			Pruning %		
		Base	BnB	BnB-Rand	BnB	BnB-rand	Nr. reviews
K values	80	14.62	0.55	0.36	82.15	88.7	3.0
	160	14.83	0.56	0.36	81.79	88.9	2.56
	320	14.97	0.61	0.39	80.31	87.94	2.06
W values	80	14.83	0.68	0.51	78.36	84.45	3.0
	160	14.79	0.56	0.35	81.94	89.13	2.56
	320	14.8	0.48	0.25	83.96	91.97	2.06
b values	4	14.89	0.57	0.36	81.48	88.97	2.39
	8	14.81	0.57	0.37	81.69	88.71	2.56
	16	14.71	0.58	0.39	81.09	87.85	2.67
Pattern	STA	10.76	0.37	0.25	82.87	89.46	2.63
	INC	17.3	0.69	0.47	81.27	87.79	2.7
	DEC	17.39	0.66	0.41	81.5	89.07	2.33
	LCY1	15.03	0.65	0.43	78.61	86.45	2.59
	LCY2	16.56	0.71	0.49	78.99	86.07	2.48
	RAND	11.79	0.35	0.19	85.27	92.24	2.48
Average		14.81	0.57	0.37	81.42	88.51	2.54

a heuristic to guide the tree search. As future work, we plan to investigate different heuristics for the search.

Table 5.3: Computational times (in minutes), pruning percentage and number of reviews for 20-periods instances

		Computational time			Pruning %		
		Base	BnB	BnB-Rand	BnB	BnB-rand	Nr. reviews
K values	80	65366.67	105.12	76.69	98.56	98.96	6.04
	160	65470.02	109.35	80.25	98.53	98.93	5.17
	320	66070.17	115.98	84.1	98.47	98.89	4.13
W values	80	66737.03	181.66	142.66	97.61	98.12	6.04
	160	64772.93	96.12	66.74	98.68	99.09	5.17
	320	65396.9	52.67	31.64	99.27	99.57	4.13
b values	4	65851.88	96.59	67.47	98.7	99.1	4.78
	8	65847.45	108.37	80.13	98.56	98.94	5.2
	16	65207.52	125.49	93.45	98.3	98.74	5.35
Pattern	STA	43447.11	73.24	56.66	98.51	98.85	5.3
	INC	72449.66	110.73	86.29	98.69	98.98	5.41
	DEC	72706.98	141.49	95.02	98.29	98.86	4.7
	LCY1	62607.87	139.2	101.02	98.05	98.59	5.19
	LCY2	69243.25	141.35	100.81	98.22	98.74	5.04
	RAND	73358.85	54.88	42.29	99.36	99.51	5.04
Average		65635.62	110.15	80.35	98.52	98.92	5.11

Chapter 6

Stochastic Dynamic Programming Based Heuristics for the (R, s, S) policy parameters computation

In this chapter, we introduce two new algorithms to compute the (R, s, S) policy parameters for the non-stationary stochastic lot sizing problem with backlogging of the excessive demand, fixed order and review cost, linear holding and penalty cost. The SDP formulations of inventory problems are particularly important. They allow a better understanding of the problem structure, and they do not require any external software for their deployment. Scarf's SDP model for computing the (s, S) policy is the most used and cited stochastic lot sizing technique. It is the first solution to efficiently compute the optimal parameters for the (s, S) policy. We present the first formulation of the (R, s, S) problem as functional equation of an SDP model. This formulation is a hybridisation of Scarf's (s, S) SDP and the (R, S) SDP presented in Chapter 4. A simple implementation of the model requires a prohibitive computational effort to compute the parameters, worse than the baseline introduced in Section 5.1. However, we can speed up the computations by using K-convexity property and memoisation techniques. The resulting algorithm is considerably faster for big instances. However, this algorithm is sub-optimal. In a few instance types, there is a low probability of computing a near-optimal policy. In the experimental section, we had to create ad-hoc testbed to have a non-optimal policy.

The second contribution, presented in Section 6.2, is the introduction of a simple heuristic for the same problem. This heuristic deploys sequentially an

(R, S) and an (s, S) algorithms to compute the (R, s, S) policy parameters. Depending on the technique used, this approach can be considerably faster than the other techniques, keeping a high quality of the computed policies. This approach can be used as bound for the BnB solution, leading to an improvement of the computational time.

In Section 6.3, we conduct an extensive computational analysis of these techniques. We assess the time required to compute the parameters and the optimality gap of these parameters. In Section 6.3.3, we analyse the impact of the heuristics on the pruning percentage of the BnB algorithm.

Finally, Section 6.4 concludes the chapter.

6.1 Stochastic Dynamic Program for the (R, s, S) policy

This section introduces the SDP formulation for the (R, s, S) policy. The (R, s, S) policy is a generalization of the (s, S) and the (R, S) one. It is possible to merge ideas from the SDP models for (s, S) and (R, S) presented in Chapter 4 to provide a pure SDP formulation to the problem of computing (R, s, S) policy parameters. The model herein follows the same structure used in the previous chapters.

6.1.1 Model

The SDP formulation, consistent with the ones of Chapter 4, is:

1. **Stage.** A stage represents a time period $t = 1, \ldots, T$ for a T-period stochastic lot-sizing problem.
2. **State.** We define N_t as the state of the system at the beginning of period t before replenishment. State $N_t = I_{t-1}$ includes the opening inventory level of period t.
3. **Action.** An action is represented by the pair (Q_t, R_t). They represents the scheduling of an order with quantity Q_t at the beginning of period t that aims to cover the demand of the next R_t periods.

4. **Immediate cost.** Let $f_t(I_{t-1}, Q_t, R_t)$ be the expected immediate cost compromising ordering, holding and penalty cost for periods $t, \ldots, t+j-1$, given the state $N_t = I_{t-1}$ and action (Q_t, R_t).

$$f_t(I_{t-1}, Q_t, R_t) = \quad K\mathbb{1}\{Q_t > 0\} + W + \sum_{i=1}^{R_t} E[h \max(I_{t-1} - d_{t,t+i} + Q_t, 0) + b \max(-I_{t-1} - Q_t + d_{t,t+i}, 0)] \tag{6.1}$$

   where E denotes the expected value with respect to the random variable $d_{t,t+i}$ and $\mathbb{1}$ is the indicator function.

5. **Objective function.** Let $C_t(I_{t-1})$ denote the expected total cost of an optimal policy over periods $t, \ldots, T$ associated with state $N_t = I_{t-1}$. Then, $C_t(I_{t-1})$ can be written as:

$$C_t(I_{t-1}) = \min_{R_t}(\min_{Q_t}(f_t(I_{t-1}, Q_t, R_t) + E[C_{t+R_t}(I_{t-1} + Q_t - d_{t,t+R_t})])) \tag{6.2}$$

   The boundary condition is:

$$C_{T+1}(I_T) = 0 \tag{6.3}$$

As in the previous SDP formulations, $C_1(I_0)$ contains the expected cost for the optimal parameters. For a list of all the symbols used we refer to 8.1.

The formulation shares the stages with the previous ones. The state space is the same as the (s, S) SDP; a state is represented by the period and the closing inventory level. The action involves the order quantity and the replenishment(review) cycle length as respectively the (s, S) and the (R, S) formulation. The functional equation resembles the (R, S) one, where the state space is explored testing the possible length of the replenishment cycle and order-up-to-level.

This formulation is more complex compared to the ones of Chapter 4, making the computational effort required to solve it prohibitive. This is mainly due to the immediate cost structure; its computation involves three variables in each period: current inventory, order size and length of the replenishment cycle. The deployment of search reduction and memoisation techniques has a crucial impact on the applicability of this model. These two enhancements are inherited from the (R, S) SDP. Section 6.1.3 applies the K-convexity property

and Section 6.1.4 adapt the memoisation presented in Section 4.2.4.

This SDP does not compute the optimal policy for all the instances. The optimal order-levels and order-up-to-levels can not be computed considering the replenishment cycles independently. However, as the experimental part of this chapter shows, the algorithm computes near-optimal policies only for sporadic cases.

6.1.2 Pseudocode

Algorithm 7 shows the procedure to compute the SDP backwards. Lines 1-2 contains the boundary condition. Line 4 search through all the possible starting inventory levels, line 6 through all the possible replenishment cycles and line 7 through all the possible order quantities.

Algorithm 7 RsS-SDP()

```
1:  for i from min_inventory to max_inventory do
2:      C_{T+1}(i) = 0
3:  for t from T down to 1 do
4:      for i from min_inventory to max_inventory do
5:          C_t(i) ← ∞
6:          for r from 1 to T − t + 1 do
7:              for q from 0 to max_order do
8:                  expected_cost ← f_t(i, q, r) + E[C_{t+r}(i + q − d_{t,t+r})]
9:                  if expected_cost < C_t(i) then
10:                     C_t(i) ← expected_cost
```

For clarity and for the sake of the future enhancements, we separate the computation of the immediate cost. Let $\zeta_{t,t+j}$ be a value of the random variable $d_{t,t+j}$ and $P(\zeta_{t,t+j})$ be the probability of assuming that value. Algorithm 8 computes Equation 4.10.

6.1.3 K-convexity

Similarly to the computation of the (s, S) policy, we can use the K-convexity property described in Section 4.1.4. Considering the functional equation (Equation 6.2), for a fixed R_t the problem is reduced to an (s, S) one with the next $R_t - 1$ periods in which an order can not be placed.

Algorithm 8 $f_t(i, q, r)$

```
cost ← W
if q > 0 then
    cost ← cost + K
for j from 1 to r do
    for each ζ_{t,t+j} value of d_{t,t+j} do
        close_inv ← i + q − ζ_{t,t+j}
        if close_inv ≥ 0 then
            cost ← cost + h close_inv P(ζ_{t,t+j})
        else
            cost ← cost − b close_inv P(ζ_{t,t+j})
return cost
```

The application is equivalent, it strongly reduces the computational time since it removes the search for the optimal Q_t for a fixed R_t. Algorithm 9 shows the pseudocode of the enhanced SDP. It is equivalent to Algorithm 2 nested inside the search for the optimal replenishment cycle.

Algorithm 9 RsS-SDP-KConv()

```
for i from min_inventory to max_inventory do
    C_{T+1}(i) = 0
for t from T down to 1 do
    best_cost ← ∞
    for j from 1 to r do
        best_cost_cycle ← ∞
        for i from max_inventory down to min_inventory do
            C_t^{cycle}(i) ← f_t(i, 0, r) + E[C_{t+1}(I_{t−1} + Q_t − d_t)]
            if C_t^{cycle}(i) < best_cost_cycle then
                best_cost_cycle ← C_t(i)
                S_t^{cycle} ← i
            if C_t^{cycle}(i) > best_cost_cycle + K then
                s_t^{cycle} ← i
                break for
        for i from min_inventory to s_t^{cycle} do
            C_t^{cycle}(i) ← C_t^{cycle}(s_t)
        if best_cost_cycle < best_cost then
            best_cost ← best_cost_cycle
            C_t ← C_t^{cycle}
```

6.1.4 Immediate Cost Memoisation

To speed up the computation of the immediate cost, we can deploy a technique similar to the one presented in Section 4.2.4. Let $l_t(I_t, R_t)$ be the function defined in Equation 4.14 that computes the holding and penalty expected cost of starting at the end of period t with closing inventory I_t and with the next review moment in R_t periods. Equation 6.1 can be rewritten as:

$$f_t(I_{t-1}, Q_t, R_t) = K\mathbb{1}\{Q_t > 0\} + W + l_t(I_{t-1} - d_{t,t+i} + Q_t, R_t) \tag{6.4}$$

The $l_t(I_t, R_t)$ function can be computed in a recursive way following the same SDP described in Section 4.2.4. The pseudocode of Algorithm 5 can be applied to the (R, s, S) problem as well.

6.1.5 Time complexity

The number of states is $2DT^2$, the same of the (s, S) SDP in Section 4.1.3. For each state, we need to evaluate all the possible replenishment cycle lengths T (line 6 of Algorithm 7) and order quantity DT (line 7). The immediate cost is computed using Algorithm 8 and has complexity of $\mathcal{O}(DT)$, while the expected cost of future periods requires $\mathcal{O}(D)$. The upper bound of the worse case complexity is:

$$\mathcal{O}(D^3T^5) \tag{6.5}$$

The complexity is considerably higher compared to the SDP solutions of Chapter 4. In the experimental section, we show that without using computational enhancements, the algorithm can solve only minimal instances.

With the K-convexity property, we avoid the search for the optimal order quantity. This reduces the complexity in Algorithm 9 to:

$$\mathcal{O}(D^2T^4) \tag{6.6}$$

we are not able to quantify the worse case impact of the memoisation since it strongly depends on the instance type.

6.2 Heuristic based on the combination of (s, S) and (R, S) algorithms

In this section, we present a new heuristic that uses an (R, S) algorithm to compute the replenishment plan and then compute the optimal (s, S) parameters for that fixed plan. The algorithm presented in Chapter 5 has some similarities with this approach since it uses an SDP to compute the order-levels and order-up-to-levels while using BnB to find the optimal replenishment schedule. The approach used herein can be seen as a heuristic to find a near-optimal leaf of the search tree.

The approach of fixing replenishment cycles and then computing the rest of the policy parameters is widely used in inventory control heuristics. For example, it has been using in algorithms for computing (s, S) policy parameters by [Ask81, BM99] and for the (R, S) policy by [WW58].

All the (R, S) solutions can compute a replenishment plan; however, not all the (s, S) algorithms can compute the s and S parameters for a fixed replenishment schedule. The dynamic solution algorithm needs to be able to have some periods in which the orders are forbidden. For this task, the SDP presented in Section 4.1 is an ideal candidate. It quickly computes the optimal policy, and it offers the possibility of considering a fixed replenishment plan, e.g. the baseline for (R, s, S).

Regarding the replenishment plan, we decided to deploy two alternatives:

- The SPD algorithm presented in Section 4.2. This solution offers an optimal policy with a low degree of uncertainty on the optimality gap.
- The shortest path solution presented in [RTHP11]. While this algorithm is designed for the service level problem, the computational effort needed to compute it is negligible compared to the SDP one. In the next section, we briefly describe this algorithm.

A heuristic allows computing a near-optimal policy. We can use the expected cost of this policy as upper bound for the pruning condition of the BnB technique presented in the previous chapter. A tighter bound allows a higher pruning percentage and a lower computational time.

6.2.1 Dynamic Programming for static-dynamic inventory

In this section, we describe the Rossi et al. [RTHP11] DP algorithm used to compute the static-dynamic uncertainty policy parameters under service level constraint. The literature review in Section 3.4.4 present their work. The goal of this section is to give an insight into the algorithm to understand better and justify the finding of the experimental section. While this algorithm focuses on the α service level (see Section 2.2), it computes a good policy almost instantly, even for big instances. It provides a faster solution for the (R, s, S) policy computation.

They extend the state space relaxation concept presented by [Tar96]. Their contribution is a filtering procedure and an augmentation procedure for the state space graph. This approach achieves a significant computational efficiency, solving any relevant size instance in trivial time. This approach has some similarities with the (R, S) SDP. In this section, we briefly explain the technique. We suggest the original work for a more detailed description.

The approach is designed for a α service level constraint, so that the non-stockout probability has to be at least α. Solving the problem under penalty cost is more complicated since the cost function for each cycle is non-convex. This is a different approach compared to the penalty cost used in this thesis; however, it is possible to connect these two models through an approximation. The so-called critical ratio can be used to compute an approximate service level, starting from holding and penalty cost.

$$\alpha_{approx} = \frac{b}{b+h} \tag{6.7}$$

The two models are not equivalent; an optimal penalty cost model is not able to compute an optimal service level policy using the critical ratio. However, the solution deployed with the critical ratio can produce a reasonable policy in trivial time. Moreover, we use only the replenishment plan and not the order-up-to-levels.

They define as cycle buffer stock $b(i, j)$ the minimum expected buffer stock level required to satisfy the required non-stockout probability for a replenishment cycle starting in period i and finishing in period $j + 1$, $j \geq i$. They define $b(i, j)$

as:

$$b(i,j) = G_{d_{ij}}^{-1}(\alpha) - d_{ij} i = 1, \ldots, T \text{ for } j = i, \ldots, T \tag{6.8}$$

where $G_{d_{ij}}$ is the cumulative probability distribution function of d_{ij}. It is assumed that G is strictly increasing, like in our case, so that G^{-1} is uniquely defined. Considering the cycle buffering cost is possible to compute the cycle cost $c(i,j)$. It can be expressed as:

$$c(i,j) = K + h(j-i+1)b(i,j) + h\sum_{t=i}^{j}(t-i)\tilde{d} \tag{6.9}$$

the cycle cost is divided in three components: the fixed ordering cost, the holding cost for the buffer stock charged all the periods and the holding cost for the items used to satisfy the expected demand.

The DP algorithm shares the same stages and states structures of the one presented in 4.2.1. They assume that the cycles are independent. Each state C_t contains the expected cost of the optimal policy that satisfies the service level constraint starting from period t. The functional equation is:

$$C_t = \min_{j \geq t}(c(t,j) + C_j)) \tag{6.10}$$

The boundary condition is:

$$C_{T+1} = 0 \tag{6.11}$$

It can be seen as a graph where the states are the nodes, and the length of the edges is represented by the cycle buffering cost.

They introduce a filtering technique to remove sub-optimal edges and a state augmentation procedure to provide feasible scheduling when a negative order is planned.

6.3 Experimental Results

In this section, we conduct an extensive computational study of the heuristics presented in this chapter. In Section 6.3.1, we assess the computational effort they require to compute a policy and the quality of the policy itself under an increasing time horizon. An analysis of the heuristics behaviour under different

demand patterns and cost parameters is presented in Section 6.3.2. Finally, we evaluate the impact of the heuristics on the BnB algorithm in Section 6.3.3.

For the experiments, we use the solvers presented in the previous chapter as a comparison and to compute the optimality gap. The new solvers described in this chapter are:

- **RsS-SDP**, the basic implementation of the SDP model presented in Algorithm 7. We include this to appreciate the impact of the optimisation techniques deployed.
- **RsS-SDP-Opt**, the SDP implementation deployed using the K-convexity property (Algorithm 9) and the immediate cost memoisation.
- **RsS-Comb-SDP**, the heuristic that combines (R, S) and (s, S) algorithms to compute an (R, s, S) policy. The SDP algorithm presented in Algorithm 3 is used for computing the replenishment cycle.
- **RsS-Comb-SP**, similar to the previous approach, the difference is that the replenishment cycle is computed using Rossi et al. [RTHP11] shortest path approach.

All experiments are executed on an Intel(R) Xeon E5640 Processor (2.66GHz) with 12 Gb RAM.

6.3.1 Scalability and quality of the solution

The testbed used in this section is equivalent to the one used in Section 5.3.1. A fixed holding cost per unit $h = 1$. The other cost factors are sampled from uniform random variables: fixed ordering cost $K \in [80, 320]$, fixed review cost $W \in [80, 320]$ and linear penalty cost $b \in [4, 16]$. The demand is modelled as a series of Poisson random variables. A uniform random variable draws the average demands per period with range 30 to 70. We generate 100 different instances. We replicate the experiments for increasing values of the number of periods.

Figure 6.1 and Figure 6.2 show the average computational time over the 100 instances in comparison with the fastest technique from the previous chapter. The simple implementation of the SDP is barely able to solve tiny instances before the time limit, making it useless for every practical use. Its performances are considerably worse than the baseline presented in Section 5.1. The

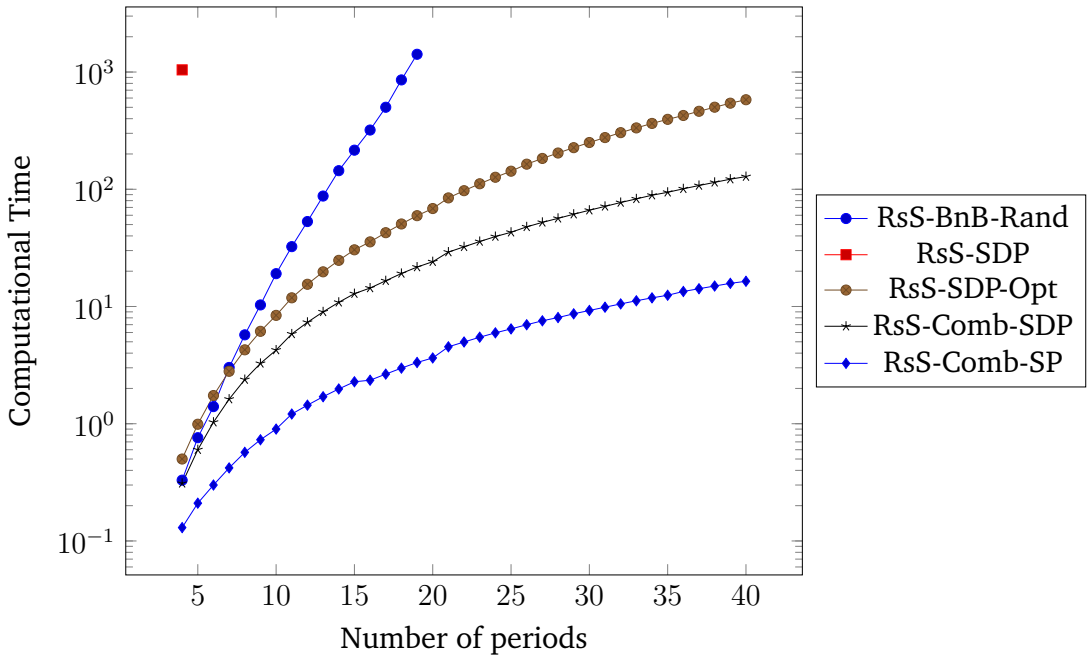

Figure 6.1: Computational time of the (R, s, S) heuristics over the number of periods, time limit 1 hour

reduction of computational effort provided by K-convexity and memoisation is massive. The randomised BnB slightly outperforms the optimised SDP for small instances up to 7 periods, then the gap between the two strongly increases; making it able to solve instances more than twice as big in the same amount of time. The K-convexity performances improvement is more significant than the memoisation one. The K-convexity reduces the complexity considerably. Moreover, it generally avoids the computation of all the DP states associated with a negative inventory (line 13 of Algorithm 9 since the occurrence of a negative optimal s_t is really rare. The memoisation offers a great speed up in the computational times, that is more significant in bigger instances. For bigger instances, the physical memory needed grows to require the usage of memory swap and a slow down in performances.

The two heuristics based on the sequential deployment of different algorithms are considerably faster than the other approaches. From the computational effort point of view, RsS-Comb-SP requires the same amount as Scarf's SDP since the time required to compute the replenishment plan with the shortest path is negligible in comparison; while RsS-Comb-SDP is the sum of the two SDP techniques from Chapter 4.

The analysis of the optimality gap computed using Equation 4.29 is displayed in

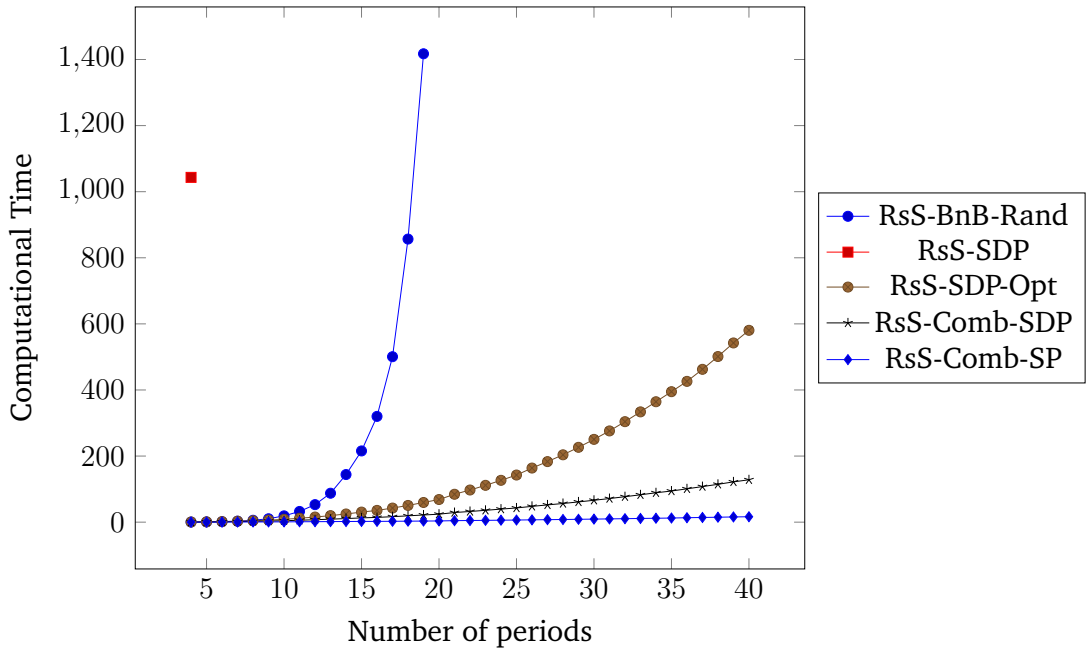

Figure 6.2: Computational time of the (R, s, S) heuristics over the number of periods, time limit 1 hour

Figure 6.3. We can compute the optimality gap only for the instances solved by the BnB technique as well. RsS-SDP and RsS-Comb-SDP calculate the optimal policy for all the instances of this experiment, with a null optimality gap. The solution based on the shortest path displays a rather stable optimality gap. In the next section, we design instances with a higher level of uncertainty to understand where these heuristics fail to compute the optimal policy.

6.3.2 Instance type analysis

These experiments aim to analyse the performances of the heuristics under different instance parameters. In the first part, we use a testbed equivalent to the one presented in Section 5.3.2. Since one heuristic solved to optimality all the instances with Poisson distributed demand, we evaluate a similar setting but with normally distributed demand in the second part.

We consider six different demand patterns: stationary (STA), positive trend (INC), negative trend (DEC), two life-cycle trends (LCY1, LCY2) and an erratic one (RAND). For the cost parameters, we use all the possible combinations of ordering cost values $K \in \{80, 160, 320\}$, review costs $W \in \{80, 160, 320\}$, holding cost fixed $h = 1$, penalty costs $b \in \{4, 8, 16\}$. Finally, the length

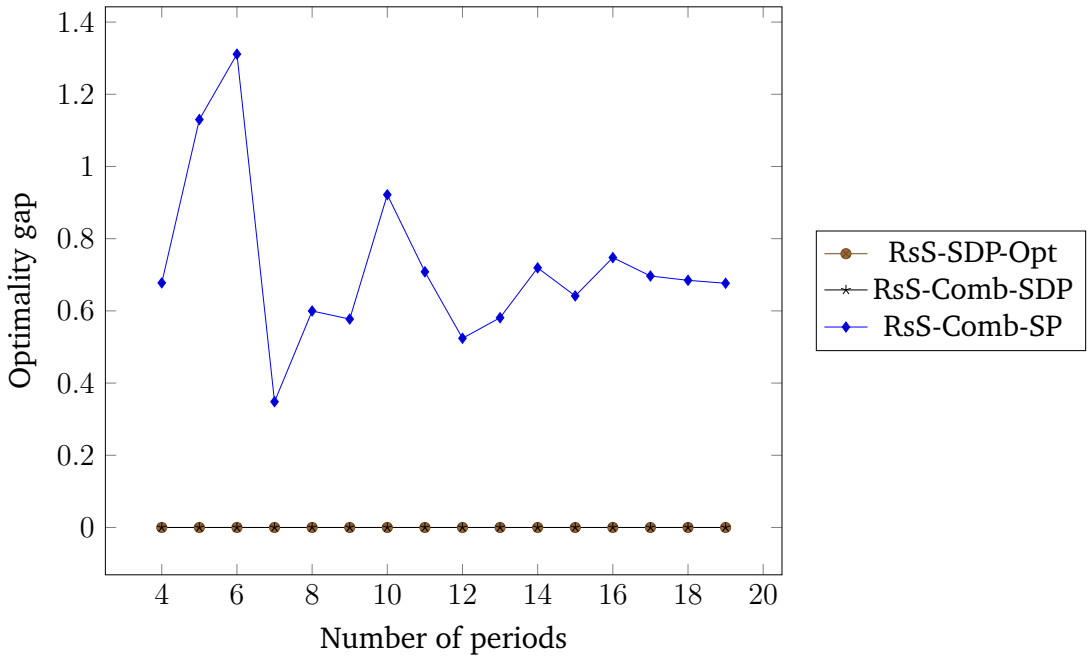

Figure 6.3: Optimality gap over the number of periods.

of the planning horizon $T \in \{10, 20\}$. We test all the combinations of these parameters, for a total of 324 instances. We assess the computational time and the simulated optimality gap for all the techniques, and we assess the effect on the BnB pruning percentage provided by the faster heuristics.

Table 6.1 and Table 6.2 shows the results for the 10 and 20-period instances. For this instance setting, the SDP and Comb-SDP solutions always compute the optimal policy. The heuristic-based on [RTHP11] display a higher optimality gap; the differences are due to different review plans.

Using the heuristics' results as upper bound for the BnB algorithm further improves the pruning percentage. On average, the best bound can reduce the total number of nodes to compute almost by half, reaching the 99.86% on instances with a high review cost. In the 10-period instances, the two best heuristics have a small optimality gap. They minimally outperform the optimal solution in a single instance, where they compute a different policy but with the same expected cost. In the simulation, the heuristic policy has a slightly lower cost due to the randomness.

We created a different testbed to appreciate the differences between heuristics. We use normally distributed demand. In the normally distributed demand, we can increase the value of the standard deviation (σ) to have a higher uncertainty

Table 6.1: Optimality gap and pruning percentage for the techniques for 10-period instances

		Optimality gap %			BnB Pruning %		
		SDP	Comb-SDP	Comb-SP	BnB-Rand	Comb-SDP	Comb-SP
K values	80	0.0	0.0	1.96	88.7	92.24	91.51
	160	0.0	0.0	1.75	88.9	92.78	92.06
	320	0.0	0.0	2.7	87.94	92.43	91.06
W values	80	0.0	0.0	1.96	84.45	88.28	86.94
	160	0.0	0.0	1.76	89.13	93.27	92.48
	320	0.0	0.0	2.7	91.97	95.9	95.2
b values	4	0.0	0.0	3.41	88.97	93.25	92.06
	8	0.0	0.0	1.64	88.71	92.65	91.89
	16	0.0	0.0	1.61	87.85	91.56	90.68
Pattern	STA	0.01	0.01	1.14	89.46	92.17	91.58
	INC	0.0	0.0	3.37	87.79	89.73	88.67
	DEC	0.0	0.0	0.78	89.07	94.06	93.7
	LCY1	0.0	0.0	2.19	86.45	92.12	90.72
	LCY2	0.0	0.0	3.92	86.07	92.48	90.57
	RAND	0.0	0.0	1.56	92.24	94.35	94.01
Average		0.0	0.0	2.19	88.51	92.49	91.54

on the expected demand. We examine instances with $\sigma \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, the maximum value used in the lot sizing literature that Chapter 3 surveys is 0.3. In this experiment, we fix the penalty cost to 10 and we extend the possible values of review and ordering costs; $K, W \in \{20, 40, 80, 160, 320\}$. We also include the (R, S) SDP algorithm to compare the heuristics with the optimal static-dynamic policy; moreover, it shares the same replenishment plan with RsS-Comb-SDP.

Table 6.3 shows the results for the 10-period instances. The patterns are clear. Optimality gap increases with high demand uncertainty, high fixed ordering cost and low review cost. The reason behind these results is that the heuristics have some similarities with the (R, S) policy computation. They perform poorly where a (R, S) policy performs worse than a (R, s, S) one. Intuitively, the (R, s, S) is suggested in situations with high uncertainty, low review cost and high ordering cost. Reviewing the inventory more frequently allows it to damp

Table 6.2: Optimality gap and pruning percentage for the techniques for 20-period instances

		Optimality gap %			BnB Pruning %		
		SDP	Comb-SDP	Comb-SP	BnB-Rand	Comb-SDP	Comb-SP
K values	80	0.0	0.0	1.27	98.96	99.42	99.34
	160	0.0	0.0	1.38	98.93	99.43	99.33
	320	0.0	0.0	1.45	98.89	99.46	99.32
W values	80	0.0	0.0	1.27	98.12	98.86	98.67
	160	0.0	0.0	1.38	99.09	99.58	99.5
	320	0.0	0.0	1.45	99.57	99.86	99.83
b values	4	0.0	0.0	2.05	99.1	99.57	99.45
	8	0.0	0.0	1.36	98.94	99.46	99.35
	16	0.0	0.0	0.77	98.74	99.28	99.21
Pattern	STA	0.0	0.0	0.52	98.85	99.27	99.23
	INC	0.0	0.0	2.23	98.98	99.33	99.18
	DEC	0.0	0.0	1.0	98.86	99.49	99.43
	LCY1	0.0	0.0	1.48	98.59	99.3	99.13
	LCY2	0.0	0.0	2.01	98.74	99.44	99.3
	RAND	0.0	0.0	1.02	99.51	99.77	99.74
Average		0.0	0.0	1.38	98.92	99.43	99.33

uncertainty. This justifies the bad performance for the heuristics in the DEC pattern, where the higher demand occurs in the first periods, a higher average leads to stronger outliers. On the other side, when the review cost is particularly high, an order is placed almost in all the review moments, in this situations the (R, S) policy is almost optimal. The RsS-SDP review plan computation has some similarities with the (R, S) SDP formulation, and the other two heuristics directly use a static-dynamic algorithm.

Non optimality of the (R, s, S) SDP

This empirical analysis allows us to understand better why the RsS-SDP is not optimal in some particular situations. When computing the solution, it considers only the expected demand for future periods. The BnB approach behaves similarly; however, during the search process, it tests all the possible

Table 6.3: Optimality gap and pruning percentage for the techniques for 10-period instances

		Optimality gap %			
		SDP	Comb-SDP	Comb-SP	RS-SDP
K values	20	0.0	0.02	0.46	0.06
	40	0.0	0.05	0.48	0.22
	80	0.02	0.11	0.62	0.28
	160	0.07	0.18	0.76	0.4
	320	0.17	0.43	1.16	0.61
W values	20	0.36	0.72	1.11	1.23
	40	0.08	0.3	0.71	0.63
	80	0.01	0.13	0.65	0.28
	160	0.0	0.04	0.65	0.08
	320	0.0	0.0	0.74	0.01
σ values	0.1	0.0	0.0	0.14	0.0
	0.2	0.01	0.02	0.32	0.02
	0.3	0.05	0.14	0.51	0.23
	0.4	0.12	0.33	0.93	0.62
	0.5	0.19	0.52	1.08	1.02
Pattern	STA	0.07	0.11	0.64	0.16
	INC	0.0	0.01	1.3	0.03
	DEC	0.19	0.38	0.88	0.61
	LCY1	0.05	0.18	0.59	0.36
	LCY2	0.04	0.3	0.72	0.49
	RAND	0.07	0.18	0.35	0.54
Average		0.07	0.19	0.75	0.36

Table 6.4: (R, s, S) policy parameters for the $K = 320$, $W = 20$, $\sigma = 0.5$, DEC pattern instance.

Period		1	2	3	4	5	6	7	8	9	10	Policy cost
RsS-SDP	γ_t	1	0	0	1	0	1	0	1	0	0	1903
	S_t	335	-	-	235	-	142	-	59	-	-	
	s_t	225	-	-	96	-	66	-	26	-	-	
RsS-BnB	γ_t	1	0	1	1	1	1	0	1	0	0	1871
	S_t	303	-	245	235	189	142	-	59	-	-	
	s_t	159	-	64	51	44	66	-	26	-	-	

replenishment combinations of the previous periods. Not considering the previous demands means ignoring the possibility of having such a low demand that at a period t the opening inventory level I_t is higher than S_t, an improbable event. This difference makes the heuristics performances worsening for high values of uncertainty and the decreasing pattern (DEC). In these instances, the high demand with high uncertainty at the beginning of the time horizon makes the occurrences of unexpected high inventory levels at a replenishment moment more likely. In this situation, the BnB solution adds more review moments (especially when the cost associated W is low) to assess the inventory level and react to the uncertainty.

For example, considering the instance of Table 6.3 with $K = 320$, $W = 20$, $\sigma = 0.5$ and decreasing demand pattern. The policies computed by the two approaches are presented in Table 6.4. The BnB approach considers the higher uncertainty at the beginning of the time horizon, and review the inventory level at period 3 and period 5 as well. While these reviews add an extra cost and in an almost deterministic system, they allow a better reaction to unexpected demand. At the end of the time horizon, the uncertainty on the inventory level is lower, and the two policies are identical from period 6 on.

The BnB policy has an expected cost of 32 (1871 against 1903) inferior compared to the SDP one.

6.3.3 Using the heuristics as bound

Many BnB approaches use a heuristic to compute bounds, e.g. MIP solvers use linear programming relaxation to compute bounds. In the BnB algorithm for (R, s, S), we can use the expected cost of the policy computed by a heuristic as the upper bound of Equation 5.12 (the pruning condition). A tighter bound leads to a higher pruning percentage. In this experiment, we asses the impact

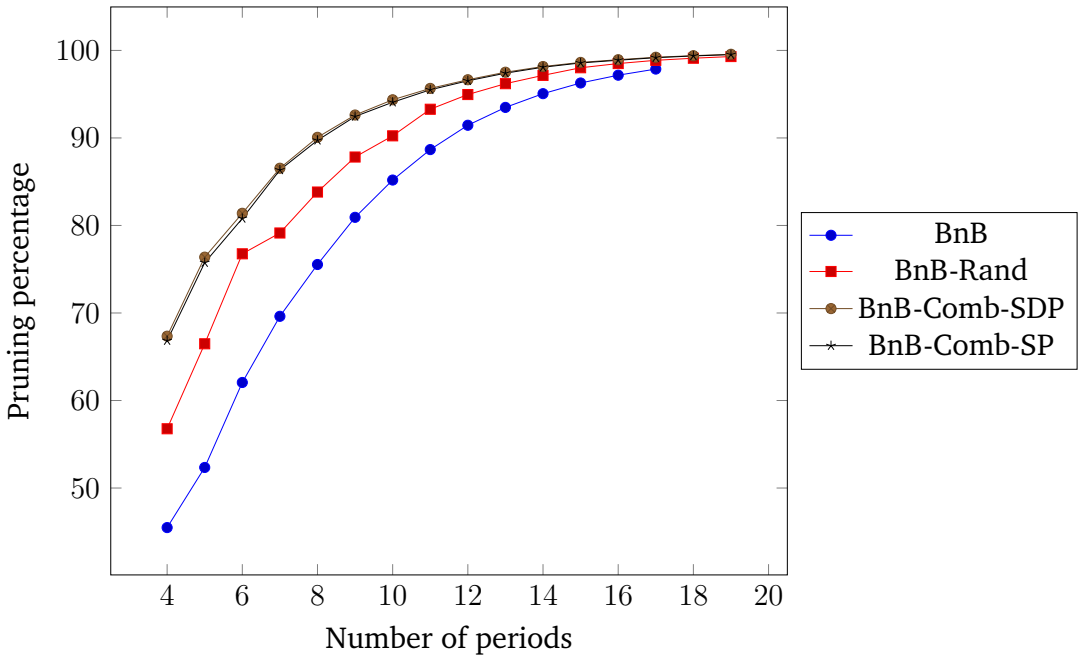

Figure 6.4: Pruning percentage over the number of periods for the BnB algorithms.

of the heuristics bounds on the performances of the RsS-BnB-Rand.

Figure 6.4 adds the new techniques to the results of Figure 5.6. The bounds computed with the heuristics improve the pruning percentage considerably. Since the combination of the two SDP generally provides a better policy, the pruning percentage is slightly higher compared to the shortest path one. However, the difference between the two is minimal. Compared to the randomised version, the new bounds reduce the number of nodes that have to be computed by more or less $\frac{1}{3}$.

Figures 6.5 shows the computational effort performances. For small instances, the effort to compute the bounds is higher than the improvement provided by a higher pruning percentage. This gap inverts for bigger instances. The shortest path heuristic bound is more efficient than the SDP one for most of the instances, since its computation takes considerably less time. For bigger instances, the higher quality of the policy computed through SDP gives that approach the upper hand.

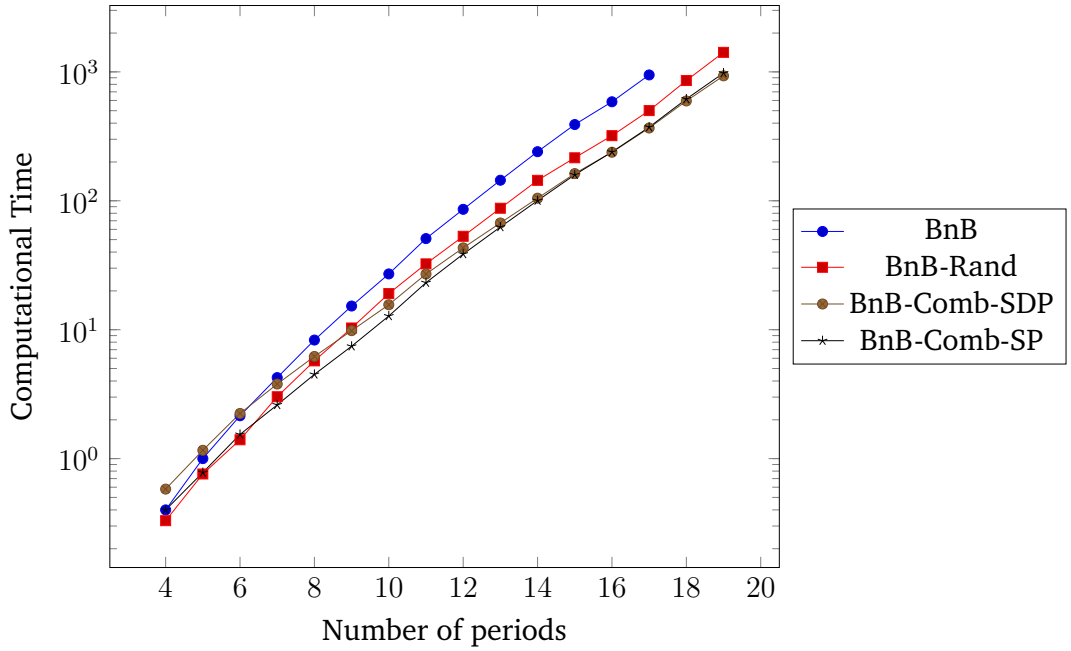

Figure 6.5: Computational time over the number of periods.

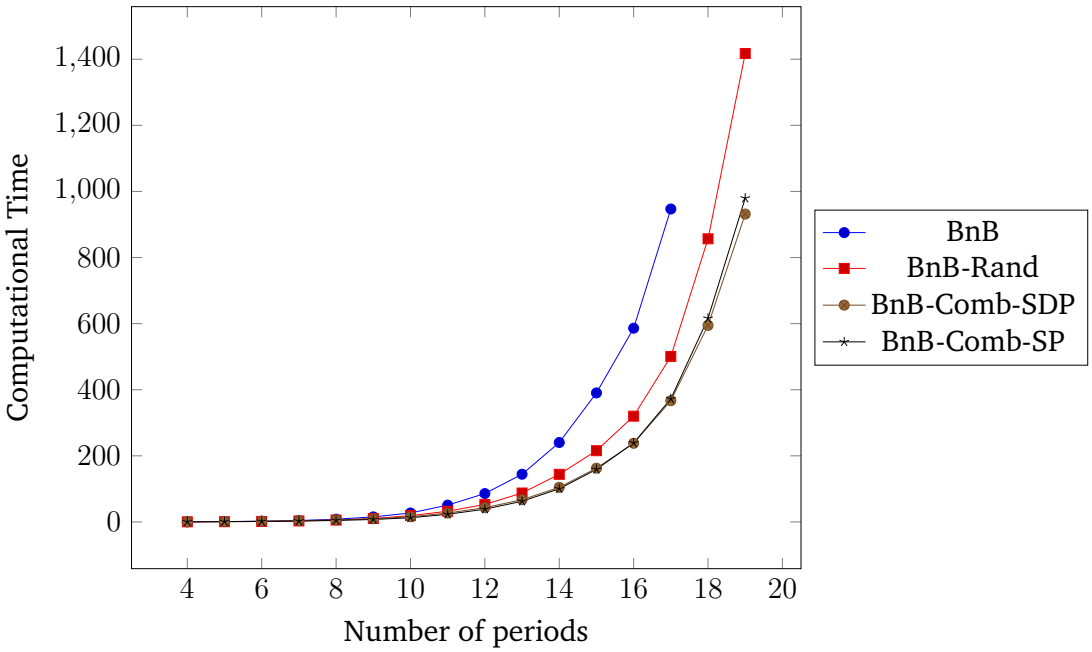

Figure 6.6: Computational time over the number of periods. Time limit 1 hour.

6.4 Conclusions

In this chapter, we present a set of heuristics for the (R, s, S) policy parameters computation problem. The computation of such parameters has been considered extremely resource demanding.

The first solution is a pure SDP formulation for the (R, s, S) policy. It shares the same state space of the (s, S) SDP and can be enhanced by the implementation of the K-convexity property. The common features with the (R, S) SDP are the way to compute the replenishment plan and the memoisation technique.

The second approach deploys an (R, S) and an (s, S) solver sequentially. The first one fixes the replenishment periods; the second computes the order-levels and order-up-to levels.

We conduct an extensive numerical study. We first investigate the scalability and the quality of the computed policies for increasing planning horizons. We then assess the performances under different types of instances. The new heuristics perform better when there is less uncertainty on the demand, and for high review, low fixed ordering cost instances. These performances are similar to the (R, S) policy ones. Finally, we focus on the analysis of the impact on the pruning percentage of using the faster heuristics as bound. The heuristics further reduce the state space computed by the BnB algorithm.

The algorithms display very low optimality gaps. The pure SDP formulation computes the optimal policy in all the situations in which the uncertainty over the demand is low and in most of the other instances.

Chapter 7

Model Dynamic Programming in Constraint Programming

In this chapter, we introduce an innovative technique used to modelling dynamic programming in constraint programming.

As seen in Chapters 5 and 6, the computation of (R, s, S) policy parameters can be divided in two parts: fixing the review plan, and solving the remaining (s, S) problem. In all the algorithms presented herein, an edited version of Scarf's SDP is used to compute the order levels and order-up-to-level. The difference between the approaches is how to compute the replenishment plan:

- The baseline presented in Section 5.1 evaluates all replenishment plans. A "brute force" application of Scarf's SDP.
- In Section 5.2, we introduce a binary tree search that aims to reuse the SDP state space. We then applied a BnB approach to prune the tree.
- In Section 6.2, we compute the replenishment plan using an (R, S) algorithm.

The only method that computes all the parameters at the same time is the full SDP formulation presented in Section 6.1.

In this chapter, we aim to use CP to compute the optimal review plan. Section 3.3 gives a brief introduction to this paradigm. CP has been previously used in the computation of policy parameters, e.g. [RTHP08,TS08]. CP solvers use BnB techniques to explore the search space. They use optimised techniques to infer tight bounds to prune the search tree effectively. The constraint propagation

might prove more efficient in pruning the search tree than the technique of Chapter 5. Moreover, a CP model of the parameters' computation allows to deploy further constraints easily. Each solver provides a wide constraint catalogue; these constraint can involve different problem variables and even the states of the DP.

The main issue is to model the calculation of the s_t and S_t parameters. There is no standard procedure to encode a DP model into CP. If part of a problem requires a DP-based constraint that is not provided by the solver used, the modeller has to either to write the global constraint or to change solver. Also, a global constraint does not allow the interaction with the individual states of the DP problem. These limitations restrict the usefulness of DP and SDP in CP.

We propose the first generic technique used to model some DP and SDP algorithms in CP. This encoding can model the DP problems that can be represented as a shortest path problem on a directed acyclic graph. This class of problem is particularly wide, e.g. knapsack problem, longest common subsequence, pattern matching.

We name this technique dynamic programming encoding (DPE). When the underling solver propagator guarantees arc consistency, a DP model can be solved by pure constraint propagation with the DPE. DPEs can form part of a larger CP model, and provide a general way for CP users to implement DP-based global constraints.

In this chapter, we provide a formalisation of the DPE that allows a one-to-one correspondence with a generic DP approach. Section 3.1.1 contains examples of DP solutions; we use these examples to show how the DPE can be deployed on different problems. We encode one of the most famous DP solutions as well, the knapsack problem. This problem is widely studied and used in many real-world situations. We present an algorithm based on the DPE to compute optimal parameters for the (s, S) and (R, s, S) policy. Section 7.2 presents a widely known variable redefinition technique for modelling DP in MIP [Mar87] called flow formulation. We use it as a competitor for our encoding. In the experimental section, we show how to use the encoding in MiniZinc, a successful application in the knapsack problem with a state reduction technique and we analyse its application on the (s, S) policy computation.

The chapter differs from the previous ones; we added introductory material to define DPE. We do not tackle only lot sizing problems for two reasons:

the DPE can be applied to a wider variety of problems and examples help to understand the encoding fully, and the computation of inventory policies has some particularities that make it a negative case for DPE. We show how the encoding can be valuable for other problems, such as the Knapsack one, and we investigate why it is computationally too demanding in the inventory lot sizing problem.

7.1 DPE Method

In this section, we define and formalise the DPE. The main idea is to model each DP state as a CP variable and use constraints to describe the way the functional equation connects them. In this way, we can implement a DP as a constraint satisfaction problem (CSP), which we call a DPE. This model is solved by constraint propagation, without backtracking. It can be part of a larger constraint model, allowing DP to be seamlessly integrated within CP. This technique can be used to model SDP as well.

To better describe the DPE, we decided to use the shortest path problem directly. One of the most famous DP-like algorithms is used to solve this problem: Dijkstra's algorithm. Figure 7.1 helps the visualisation of the problem.

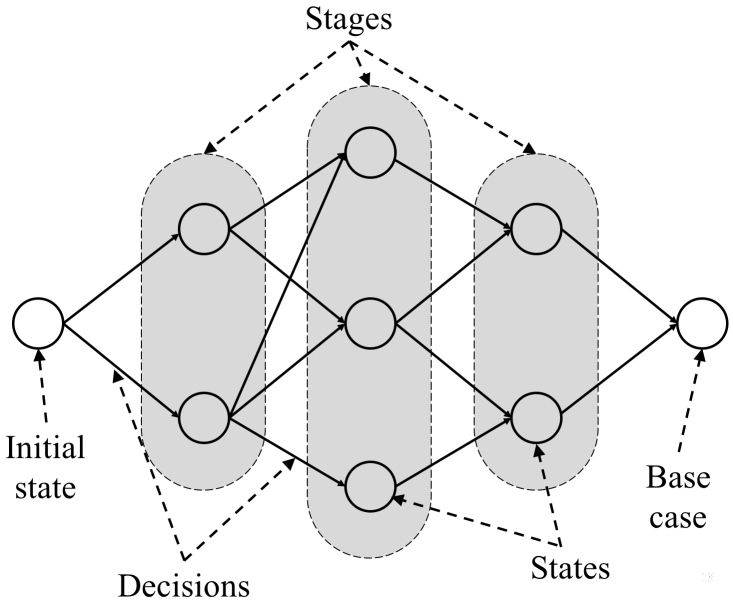

Figure 7.1: Graph relative to a generic shortest path problem.

We follow the same structure presented in Section 3.1 and used in the previous

chapters to define DPs algorithms. Most DPs can be described by their five most important elements: *stages*, *states*, *actions*, *immediate cost* and *objective function*. These elements are modelled as:

1. **Stage.** The fundamental feature of the DP approach is the structuring of optimisation problems in stages, which are solved sequentially. The solution of each stage is used in the computation of the next stage problem. In Figure 7.1, the stages are represented in grey. In the DPE the stages are simply represented as groups of states. The solver discovers the order of the stages by suspending the constraints involving variables non-fixed values.

2. **State.** Each stage of the problem is associated with one or more states. These contain enough information to make future decisions. In the DPE, the states are represented by CP variables. They contain the optimal value for the related subproblem. In the graphical representation, they are the nodes of the graph. In Dijkstra's algorithm, these variables contain the length of the shortest path from that node to the sink. We can identify two particular type of stages: the initial state that contains the optimal solution for the whole problem and no other state uses its value. And the boundary condition that is the cost of the smallest problems, their solution does not depend on any other state. In Figure 7.1, they are represented by the source and the sink of the graph.

3. **Action.** For every state, we have a set of feasible actions that can lead to a state of the next stages, which in the graph are represented by the edges leaving the associated node. If the edge is part of the shortest path, it means that the action is taken. In the DPE, the possible actions of a state are represented by the future stages states involved in its functional equation.

4. **Immediate cost.** An immediate cost is associated with each action taken from a state. In Figure 7.1 these costs are represented by the weights of the involved edges.

5. **Objective function.** The last general characteristic of the DP approach is the recursive optimisation procedure. The goal of this procedure is to build the overall solution by solving one stage at the time. The procedure has numerous components that are modelled as a functional equation. The functional equation links the optimal solution to each state to the

subsequent's states solution. In the DPE the equation is applied to every state via a constraint. This constraint binds the state's optimal value to the solution of the next stages involved. The constraint considers the immediate cost associated with each decision action; which is the value that is added to or subtracted from the next stage state variables. In the shortest path problem, the constraint applied to each (non-sink) state assigns to the node's CP variable the minimum of the reachable with one edge node's CP variables, plus the edge cost.

The correctness of the DP formulation guarantees the optimality of the solution.

Modelling DP as CP also has a software development advantage: it makes available for DP development a variety of CP tools for specification, tracing, debugging and visualisation.

7.2 MIP flow formulation

We compare the new encoding against the well-known flow formulation introduced in [Mar87]. The flow formulation can encode the shortest path problem in Figure 7.1. Binary variables represent the edges of the DAG; the variables assume value 1 if the edge is used. Each node is represented by a constraint that balances its flow.

The difference between the encodings is the order in which the states are explored and resolved. In DP, the state computations are ordered by the stage structure. The computation of a state's value occurs when all the information need is available. The DPE preserves this structure; for this reason, it can solve problems without search, only with constraint propagation. In the MIP flow formulation, it is completely replaced by a search on the binary variables. The DPE approach is more robust than search; which in the worst case, can spend significant time exploring a subtree.

We should also note that our DPE can, in principle, be used in MIP. However, this requires a large number of big-M constraints causing inefficiency. CP is a much more suitable technology for the DPE because of its greater expressiveness.

The flow formulation without side constraints works particularly well in MIP solvers because they can take advantage of the total unimodularity of the

matrix. This allows the Simplex method to find an integer solution, and it is highly exploited by modern MIP solvers.

7.3 Examples

In this section, we present some of DPE's possible applications. DPE is a paradigm, not a specific algorithm. An exhaustive definition of DP that covers all its possible applications is complex to achieve. We start by encoding the problems described in Section 3.1.1, then we apply the approach to the well-known knapsack problem.

7.3.1 Fibonacci numbers

Suppose that as part of a CP model, we need to compute the n^{th} Fibonacci number.

To find the n^{th} Fibonacci number we need n states $x_1, l \ldots, x_n$. The boundary conditions are:

$$x_1 = 1 \tag{7.1}$$

$$f_2 = 1 \tag{7.2}$$

and functional equation:

$$f_i = f_{i-1} + f_{i-2} i = 3, \ldots, n \tag{7.3}$$

We create a CP model with n variables x_i in a CP system. The boundary conditions are constants. If we post the functional equation as a series of constraint that bounds the value of each variable to the previous ones, all the f_i values are computed through constraint propagation. There is no need to search the solution space created by the cartesian product of the variables domains.

The resulting CSP is not recursive; it simply relates a set of variables by constraints. Each value is computed only once. This is equivalent to solving a DP by forward recursion. DPEs are not equivalent to memoisation in CP. In the second one, recursive solutions are stored and reused in a DP-like way. In DPE, the state variables can interact with other problem variables via constraints.

7.3.2 Change problem

We have to find the minimum number of coins adding up to a given total of Y. The problem can be modelled in CP using the following mathematical formulation:

$$\min \sum_i x_i \tag{7.4}$$

$$\text{s.t. } \sum_i x_i c_i = Y \tag{7.5}$$

where the x_i are finite domain variables denoting the number of coins from denomination i. This formulation is solved through search on the x_i domains. However, the problem can be solved in pseudo-polynomial time by the DP described in Section 3.1.1.

We now use the DPE to model this DP as a CSP, leading to a backtrack-free CP model. First, create a CP variable for each DP state. Then the seed values and recurrence relation can be written using arithmetic constraints:

$$N_0 = 0$$
$$N_i = \min\{N_{i-d_k} + 1 \,|\, k = 1, \ldots, D\}$$

The result of the DP computation is the value of N_Y. Depending on the solver, additional variables might be needed. For example, it might be necessary to create a variable j to store $i - d_1$ to access N_j then.

7.3.3 Knapsack Problem

The knapsack problem is one of the most famous problems in combinatorial optimisation. Variations of the knapsack problem are widely used in inventory control. We consider the most common version in which every item can be packed at most once, also known as the 0-1 knapsack problem [PN14]. Research on this problem is particularly active [MPT00] with many applications and different approaches.

The problem consists of a set of items I with volumes v_i and profits p_i. The items can be packed in a knapsack of capacity K. The objective is to maximize the total profit of the packed items without exceeding the capacity of the knapsack. The binary variables x_i represent the packing scheme; x_i is equal to 1 if the item

is packed, 0 otherwise. The mathematical formulation is:

$$\max \sum_{i=1}^{I} x_i p_i \tag{7.6a}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} x_i v_i \leq K \tag{7.6b}$$

$$x \in \{0, 1\} \tag{7.6c}$$

This model can be directly implemented in CP or in MIP. To solve the binary knapsack problem, we use the well known DP-like algorithm described in [Mar90].

The problem can be formulated as DP as:

1. **Stage.** There are I stages, each one is associated to an item. The items have a defined order. In each stage, we decide if the associated item is packed or not.
2. **State.** We define S_i as the state of the system at stage i. State $S_i = \{V_i\}$ includes V_i the unused volume of the knapsack.
3. **Action.** An action is represented by the packing of the item x_i.
4. **Immediate cost.** Let $f_i(x_i)$ be the immediate cost given the state S_i and action x_i.

$$f_i(x_i) = x_i p_i \tag{7.7}$$

5. **Objective function.** Let $C_i(V_i)$ denote the optimal profit for packing items $i, \ldots, I$ in a knapsack of size V_i. Then, $C_t(I_{t-1})$ can be written as:

$$C_i(V_i) = \max_{x_i}(f_i(x_i) + C_{i+1}(V_i - x_i v_i)) \tag{7.8}$$

with the condition that if $V_i - x_i v_i < 0$ then $x_i = 0$. The boundary condition is:

$$C_{I+1}(V_{I+1}) = 0 \tag{7.9}$$

$C_1(K)$ contains the profit of the optimal packing scheme.

A rooted DAG can represent this DP; in this case, a tree with state $C_1(K)$ as root and nodes $C_{I+1}(V_{I+1})$ as leaves. For every internal node, $C_i(V_i)$ the leaving arcs represent the action of packing the item i, and their weight is the profit

obtained by packing the i-th item. A path from the root to a leaf is equivalent to a feasible packing. The longest path of this graph is the optimal solution. If we encode this model using a DPE, creating all the CP variables representing the nodes of the graph, then it is solved by pure constraint propagation with no backtracking.

We use this problem to show the potential for speeding up computational times. Some states can be removed by the state space a priori, if they are proved to not be part of an optimal solution. With the DPE implementation, we can use a well-known and straightforward technique to reduce the state space without compromising the optimality. If at state $C_i(V_i)$ volume V_i is enough to contain all items from i to I (all the items we might pack in the next stages); then we know that the optimal solution contains all of them, as their profit is a non-negative number. This pruning can be made more effective by sorting the items in decreasing order of size, so the pruning will occur closer to the root and further reduce the size of the state space.

In the experimental section of this chapter, we use the knapsack problem as a benchmark to compare the DPE with the flow formulation.

7.4 (R, s, S) policy computation

With DPE, we can create an equivalent CP model for the SDP algorithms introduced in the previous chapters.

We can model Scarf's (s, S) SDP in CP. The approach is not different from the one used in the previous section for the knapsack problem. Then delegate to the CP search the computation of the review plan. This leads to a monolithic CP model for the computation of optimal (R, s, S) parameters.

The DPE associated uses CP variables to model the states of the SDP model presented in Section 4.1, so one for every possible combination of inventory level and period. Each variable contains the cost of the expected cost of the optimal policy deployed in that period and using the inventory level as starting stock. CP is generally working with variables with a finite domain; in this case, discretisation is used to model these real variables. Another set of variables is needed to represent the replenishment plan, each γ_t is represented by a binary variable. Depending on the solver used, additional variables might be needed to model the problem.

In each state, the functional equation is deployed through a series of constraints that implement Equation 5.2. The expected cost is modelled as a weighted sum of next period state variables, the immediate cost can be easily computed, and the min constraint is available in all the solvers.

Such a model can solve the problem by branching only on the γ_t variables and obtaining the value of the state variables by constraint propagation. This is an example of a parameterised SDP problem that can be solved through DPE.

DPE allows the deployment of side constraints from the CP solver catalogue. It gives a considerable modelling advantage compared to a bespoke implementation. For example, limitations of order patterns are easy to deploy in a DPE model; regular expressions can be used to force a minimum distance between orders or forcing a maximum number of orders per week/time horizon. However, the added complexity makes it impossible for the DPE the solving of small/medium instances in reasonable time; making it not usable in a real-world case. The encoding of a problem into a general-purpose solver as a CP one is generally less efficient compared to a bespoke implementation; due to the variable creation overhead and the inference and search processes. A CP In the experimental section, we analyse a DPE encoding of the (s, S) SDP that is the base of the (R, s, S) policy computation. We show that it adds a consistent overhead, and it does not allow the deployment of the K-convexity enhancement; these and other reasons explained in the experimental section make the policy computation a negative case for the DPE.

Complexity

We use the same notation available in Section 4.1.3. We consider D the maximum demand per period and T the length of the planning horizon. The maximum inventory level is DT and the minimum $-DT$. The CP model generated by the DPE for the computation of the (s, S) policy requires $\mathcal{O}(D^2T^4)$ variables and $\mathcal{O}(D^2T^3)$ constraints. The number of variables needed is higher than the state space dimension because additional variables are needed to store partial results of the expected cost computation.

7.5 Experimental Results

This experimental section is more diversified that the previous chapters' ones. Since we introduced a new encoding, we aim to assess the general applicability of the DPE, not only in the inventory control problem.

In Section 7.5.1, we compare the DPE and the flow formulation on the shortest path problem. We implement the models in Minizinc; we solve them using a CP and a MIP solver.

Experiments in Section 7.5.2 focus on the knapsack problem. We compare the DPE solved by a CP solver with the flow formulation solved by different MIP solvers. Moreover, we assess the impact of the state reduction technique.

Finally, Section 7.5.3 analyses the DPE solution for the (R, s, S) policy computation.

Environment

In the first experiment, we use MiniZincIDE 2.1.7, while the second part is coded in Java 10. We used 3 CP solvers: Gecode 6.0.1, Google OR-Tools 6.7.2 and Choco Solver 4.0.8. We use as MIP solvers: COIN-OR branch and cut solver, IBM ILOG CPLEX 12.8 and Gurobi 8.0. All experiments are executed on an Ubuntu system with an Intel i7-3610QM, 8GB of RAM and 15GB of swap memory.

7.5.1 Shortest path in MiniZinc

MiniZinc is a standard modelling language. It provides a set of standard constraints, with ways of decomposing all of them to be interpreted by a wide variety of solvers. To achieve standardisation, the MiniZinc constraints catalogue is very limited. It includes only constraints that are available in all its solvers, or that can be decomposed into simpler. The decomposition is generally done in a naive way, causing poor performances. The DB-based constraints are an example of this.

In this section, we focus on applications of the DPE in MiniZinc. We aim to apply this new technique to the shortest path problem and solve it with the DPE of the Dijkstra algorithm. The shortest path is one of the benchmarks of

```
forall( i in 1..N ) (
    if i = Start then
            % outgoing flow
        sum(e in Edges where Edge_Start[e] = i)(x[e]) -
            % incoming flow
        sum(e in Edges where Edge_End[e] = i)(x[e]) = 1
    elseif i = End then
        sum(e in Edges where Edge_Start[e] = i)(x[e]) -
        sum(e in Edges where Edge_End[e] = i)(x[e])
        = -1
    else
        sum(e in Edges where Edge_Start[e] = i)(x[e]) -
        sum(e in Edges where Edge_End[e] = i)(x[e])
        = 0
    endif
);
```

Figure 7.2: Minizinc flow formulation of the shortest path.

```
forall( i in 1..N ) ( % for all the nodes
    if i = End then
        x[i] = 0     % special case for the sink
    else
        x[i] = min(e in Edges where Edge_Start[e] = i)
        (x[Edge_End[e]] + L[e])
    endif
);
```

Figure 7.3: Minizinc DPE of the shortest path.

one the past MiniZinc challenges [SFS+14]. Analyzing the model (see Figure 7.2) we can see that the current reduction is based on a flow formulation on the nodes of the graph, which regulates the flow over each node and requires a binary variable x for each edge indicating whether an edge is used or not. This is the encoding proposed by Martin [Mar87].

Our implementation is based on Dijkstra's algorithm: every decision variable contains the shortest distance to the sink node. The formulation (Figure 7.3) is shorter and more intuitive than the previous one.

We compare the methods on the ten available benchmark instances. We use the MiniZincIDE and Gecode as the solver, with 20 minutes as a time limit. Table 7.1 shows the results of the computations. When the flow formulation finds a good or optimal solution quickly, the DPE is approximately twice as fast. However, the flow formulation requires search that can take exponential time, and it is

Table 7.1: Time required to complete the computation of the 10 benchmark instances in Gecode. '-' represents a timeout.

CP solver	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Dijkstra	23 ms	19 ms	18 ms	17 ms	24 ms	20 ms	25 ms	23 ms	20 ms	29 ms
Flow formulation	-	50 ms	60 ms	571 ms	46 ms	-	47 ms	1 182 s	4 504 ms	-

unable to find a solution before the timeout occurs. The most interesting result is that, by using only constraint propagation, DPE performance is robust and only marginally affected by the structure of the instances. In some cases, e.g. instance 7, the flow formulation quickly finds an optimal solution but takes a long time to prove optimality, in which case the DPE is more than four orders of magnitude faster.

The DPE requires a smaller number of variables since it requires only one for each node. On the contrary, the flow formulation requires a variable for each edge. This is without taking into account the number of additional variables created during the decomposition.

The DPE cannot rival a state-of-the-art shortest path solver in terms of performance for standard problems. However, the DPE allows a more flexible model than a specific global constraint and a more efficient model in MiniZinc. This is useful in the case of parameterised shortest path problems, in which other constraints influence the costs of the edges. An example of a parameterised shortest path problem is the network interdiction problem [IW02] that can be modelled using DPE.

We repeated the above experiment using a MIP solver instead of CP. Table 7.2 contains the results of the ten instances solved using COIN-OR branch and cut solver. Interestingly, the situation is inverted: the flow formulation performs efficiently while the DPE fails to find an optimal solution in many cases. This is due to the high number of auxiliary discrete variables needed by the MIP decomposition of the *min* constraint. The DPE loses one of its main strengths: DP computation by pure constraint propagation. Moreover, the MIP can take advantage of the unimodularity of the matrix, as mentioned before. We, therefore, recommend the usual flow-based formulation for MIP and the DPE for CP.

Table 7.2: Time required to complete the computation of the 10 benchmark instances in CBC COIN. '-' represents a timeout.

MIP solver	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Dijkstra	375 s	64 ms	-	-	-	20 667 ms	61 ms	-	138 ms	303 ms
Flow formulation	31 ms	39 ms	34 ms	40 ms	46 ms	35ms	36 ms	40ms	37 ms	53ms

7.5.2 Knapsack Problem

We now apply the DPE to the knapsack problem [Mar90] because it is a widely known NP-hard problem, it has numerous extensions and applications in supply chain systems, there is a reduction in MiniZinc for this constraint, and it can be modelled with the technique proposed by [Mar87]. Variations of the knapsack problem are widely used in inventory control. We consider the model previously presented in this chapter.

We compare the DPE's performance with several other decompositions of the constraint:

- **Naive CP**, a CP model that uses the simple scalar product of the model (7.6a) - (7.6b). The MiniZinc encoding of the knapsack constraint uses the same decomposition.
- **Global constraint**, the knapsack global constraint available in Choco. This constraint is implemented with scalar products. The propagator uses Dantzig-Wolfe relaxation [DW60].
- **DPE**, a CP formulation of the encoding proposed in this chapter solved using Google OR tools.
- **DPE-sr**, the previous solution with the addition of a state space reduction technique.
- **DPE-sr-sorting**, the previous solution with the items ordered for decreasing volume.
- **Flow model {Solver}**, the MIP flow formulation solved by three different MIP solvers: COIN CBC, CPLEX and Gurobi.

No restriction on the multithreading is imposed on the MIP solvers.

As a testbed, we use Pisinger's instances [Pis97]. The volume of each item (v_i) is sampled from a uniform variable in the range $[1, 100]$. Four different types of instances are defined, in decreasing correlation between items' weight and profit order:

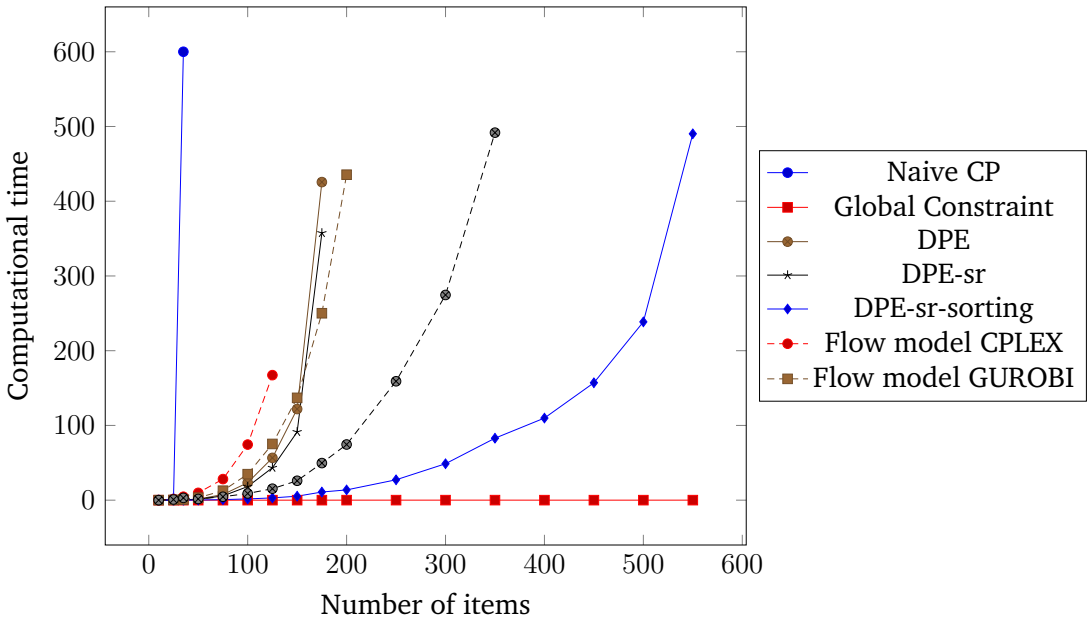

Figure 7.4: Average computational time for subsetsum instances

- *Subsetsum*, for all the items the profit is equivalent to the volume $p_i = v_i$.
- *Strongly correlated*, the profit of each item is equal of its volume plus ten, $p_i = v_i + 10$.
- *Strongly correlated*, the profit of each item is equal of its volume plus ten, $p_i = v_i + 10$.
- *Weakly correlated*, the profit of each item is in the close range of its volume, p_i is sampled from a uniform variable with range $[v_i - 10, v_i + 10]$.
- *Uncorrelated*, the profit of each item is sampled from a uniform variable with range $[1, 100]$.

the capacity of the knapsack is the half of the volume of the sum of all items. We increase the size of the instances (I) until all the DP encodings fail to find the optimal solution before the time limit. A time limit of 10 minutes is imposed on the MIP and CP solvers, including variable creation overhead.

Figures 7.4, 7.5, 7.6 and 7.7 shows the computational time in relation with the instance size for respectively subsetsum, strongly correlated, weakly correlated and uncorrelated instances.

We can see that DPE clearly outperforms the naive formulation in CP and the flow formulation solved with an open-source tool, CBC. Basic DPE computed by an open-source CP solver is computationally comparable to the flow formu-

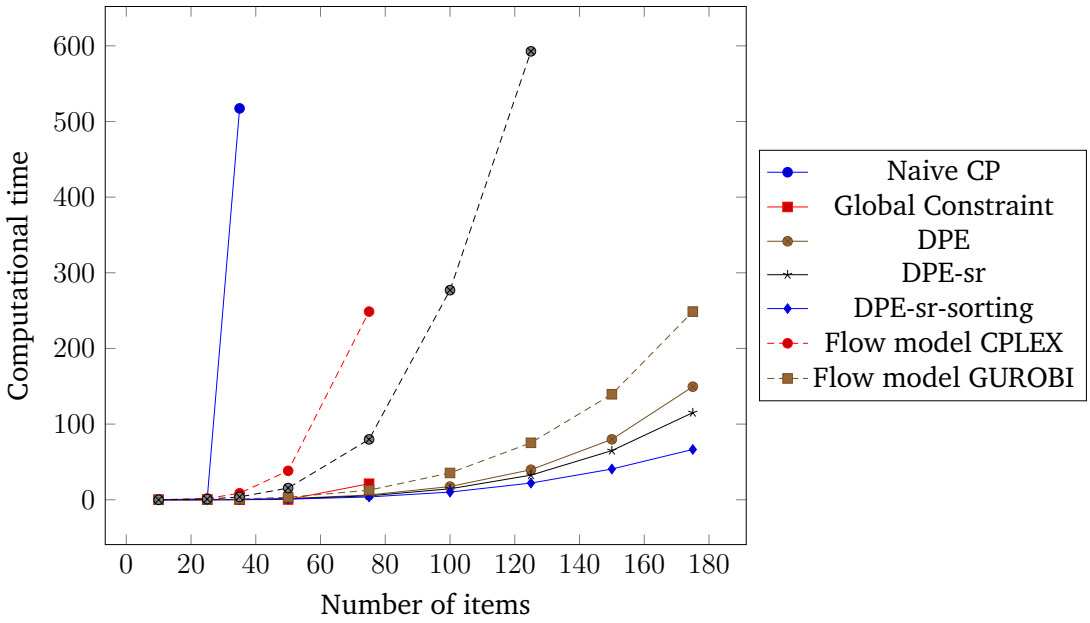

Figure 7.5: Average computational time for stronglycorrelated instances

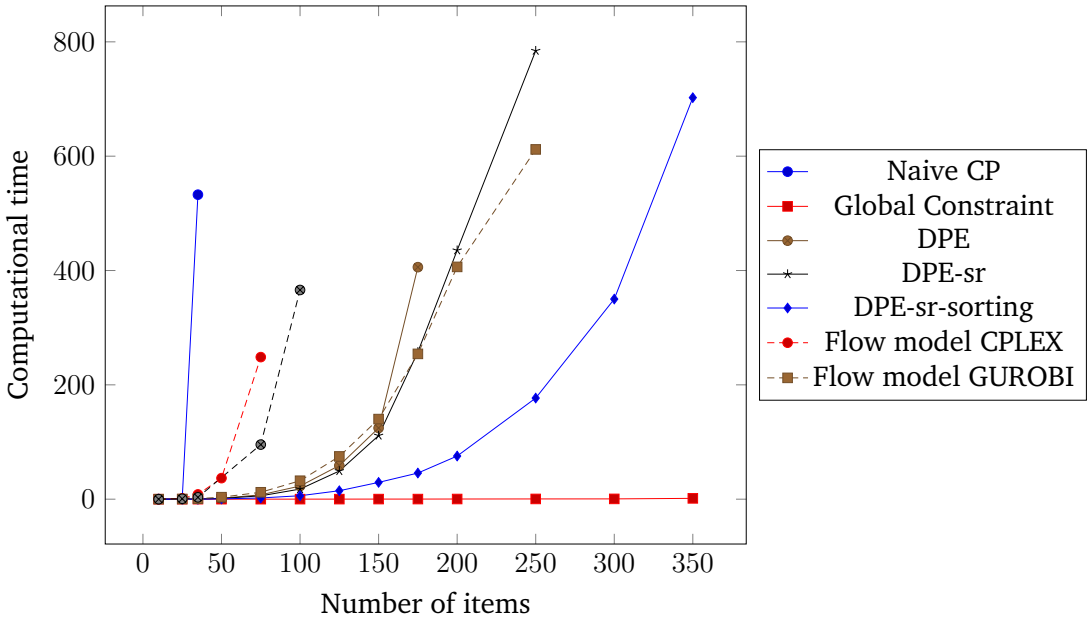

Figure 7.6: Average computational time for weaklycorrelated instances

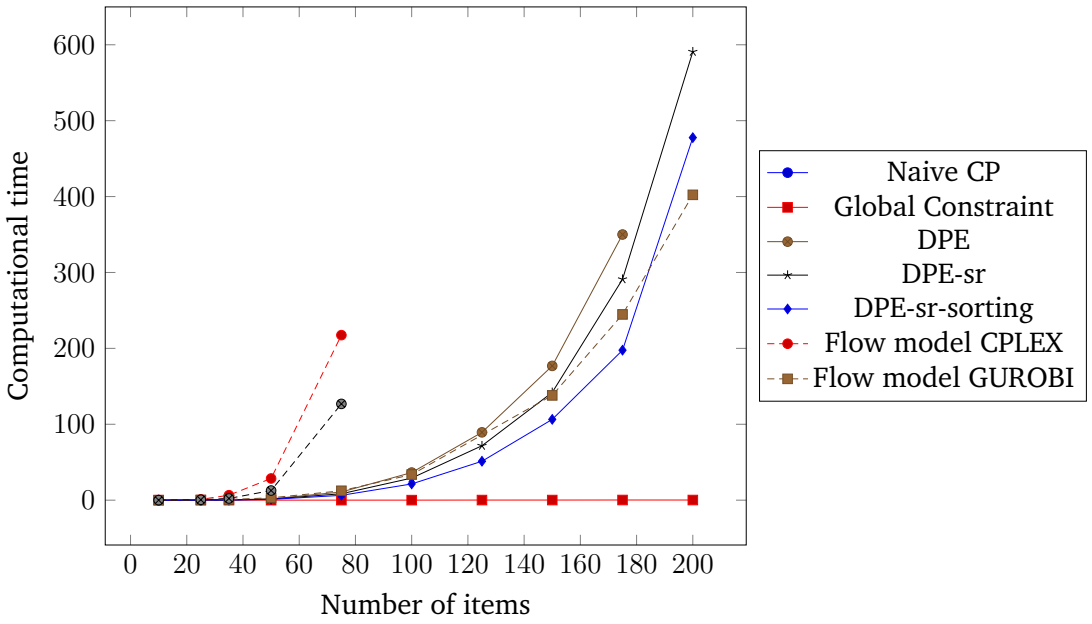

Figure 7.7: Average computational time for uncorrelated instances

lation implemented in CPLEX, and it outperforms the Gurobi's one in instances with low correlation between weight and profit of the items.

The DPE outperforms the variable redefinition technique in MIP, because of the absence of search. It is also clearly better than a simple CP model of the problem definition, which is equivalent to the Minizinc's decomposition. The Choco constraint with ad-hoc propagator outperforms the DPE in most of the cases, confirming that a global constraint is faster than a DPE. An exception occurs in the strongly correlated instances; in this case, the global constraint fails to find the optimal solution in many test instances, even with a small number of items. We assume that the search focuses on non-optimal branches of the search tree. When the DPE constraint has to be called multiple times during the solving of a bigger model, it can outperform a global constraint since the overhead to create all the variables is not repeated.

The state reduction technique provides a considerable improvement. The basic DPE can solve instances up to 200 items. Its limits are related to the memory necessary to store the state space. A CP variable has an intrinsic overhead. The state space grows so rapidly that heavy usage of the SWAP memory is needed. However, this effect is less marked when a state reduction technique is applied. On the contrary, it is stronger when the correlation between item profits and volumes is high.

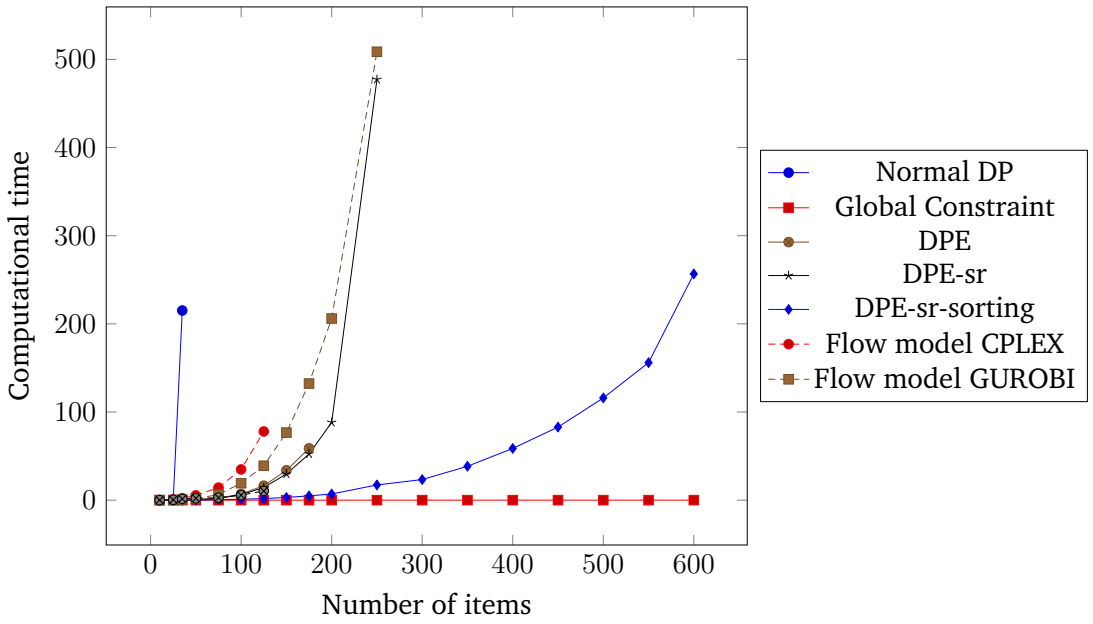

Figure 7.8: Average computational time for subsetsum instances with a larger knapsack

The reduction technique performance improves when we increase the number of items needed to fill the knapsack since the pruning occurs earlier in the search tree. Figure 7.8 shows the results for a knapsack with extended volume.

This experiment demonstrates the potential of DPE with state space reduction. With a simple and intuitive reduction technique, we can solve instances ten times bigger than with a simple CP model. We can see that the behaviour of DPE is stable regardless of the type of the instance; on the contrary, the performance of the space reduction technique strongly depends on the instance type and the volume of the knapsack.

Clearly, we can not outperform a pure DP implementation. This is mainly due to the time and space overhead of creating CP variables. The DPE requires more time to create the CP variables than to propagate the constraints.

7.5.3 (s, S) policy computation

The performances of the DPE model of Section 7.4 solved using OR tools make the (R, s, S) policy computation a negative case for the DPE.

In this section, we compare the DPE encoding of the (s, S) SDP with the

Table 7.3: Time required to compute a policy in seconds.

Periods	**3**	**4**	**5**	**6**	**7**
(s, S) DPE	469	966	1863	2919	5454
(s, S) SDP	20	44	86	166	218
(s, S) SDP + K-Convexity	0.04	0.07	0.11	0.2	0.24
(R, s, S) Baseline	0.4	1.2	3.7	11	29

standard DPs and the (R, s, S) baseline brute force. We tested the policies on the instances for testing the scalability of the solutions described in Section 4.4.1. We use as CP solver Google OR Tools. Since CP solvers need discrete variables, we discretised the variables containing costs by multiplying all the cost factors by 100 and then scaling back the final policy cost.

Table 7.3 shows the results of the experiments. The first two lines of the table compare the same algorithm, the first one encoded with DPE and the second one directly coded in Python. Solving the DP with a CP solver takes 20 times more computational effort than a bespoke implementation. A consistent part of the computational time is required to create the variables and constraints; this part takes more time than the solving of the model itself. This is to be expected, a bespoke implementation of a particular algorithm is generally faster than an equivalent model passed to a solver. A CP solver has to find in which order the constraints have to be solved, and the updates of its variables are more time demanding.

The K-convexity has a massive impact on the computational performances, see Section 4.4. The time required to solve the (s, S) SDP through DPE is considerably higher than the time needed for the baseline to compute the optimal (R, s, S) policy. So, it is faster to calculate all the policies for all the possible replenishment plans faster than the time that DPE solves one.

Understanding the reasons behind this poor performance is useful to understand DPE better. These reasons are:

- *Variables creation overhead*. As for the knapsack problems, the variable creation requires a considerable amount of time. The (s, S) SDP has a large state space. A CP variable represents each state. It includes a domain and the references to the constraints involving that variable. Creating a CP variable requires more computational effort and memory compared to a normal one. Moreover, additional variables are needed to store partial

results of the cost computation.

- *The discretisation of the costs*.CP solvers work with variables with finite domains. The cost stored in each state is a real number. CP solvers generally include discretisation techniques to deal with floating-point variables. Higher accuracy is associated with bigger domains and slower performances.
- *Constraint propagation*. In SDP the costs associated with the states are computed with the functional equation, in the DPE models through constraint propagation. While constraint propagation is generally faster than a search process, it is considerably slower than the computation of the functional equation.
- *K-convexity*. The K-convexity considerably reduces the computational effort required by the (s, S) SDP. However, it is not possible to model it through DPE.

7.6 Conclusions

This chapter presents an innovative technique for mapping DP into CP, called the dynamic program encoding. Through DPE, a DP algorithm can be seamlessly included in a CP model and solved by pure constraint propagation; without search or backtracking.

We provide a standard way to model a DP into DPE and some examples of its application. We demonstrate the potential of the DPE in constraint modelling in several ways: we compare it with another DP-encoding technique using CP and MIP solvers; we show how to use state reduction techniques to improve its performance; we show that it outperforms a well-known DP encoding technique, and greatly outperforms non-DP-based CP approaches to the knapsack problem. We apply the DPE to MiniZinc benchmarks, showing how its performance is faster and more robust than existing CP techniques. The experimental results prove that DPE is unsuitable for use in MIP, where standard methods are much better.

We design a new model that uses the DPE to compute optimal (R, s, S) policy parameter by searching the replenishment plans solution space using CP. This is a negative result for DPE; the main reasons are the large state space and the

overhead introduced by discretisation.

To summarise, DPE can be used when:

- a DP-based constraint is needed, but also other constraints can affect states inside the DP. For example, bilevel interdiction problems, see [PRTV18];
- the respective DP global constraint is not implemented in the specific solver;
- DP approaches are needed in MiniZinc as starting approach to decompose more complex problems in simpler instructions.

Chapter 8

Conclusion

Stochastic lot sizing is an active branch of operational research. This thesis introduces a set of novel algorithms to compute policy parameters for the single-item, single-location stochastic inventory control problem with backlogging of excessive demand. These algorithms can compute optimal parameters for the (R, S) policy and optimal and near-optimal parameter for the (R, s, S) one. No restrictions on the demand type are needed. They can be easily implemented without the need for external solvers. The common thread between the elements of the set is the utilisation of Stochastic Dynamic Programming approaches.

The (R, s, S) policy has a high practical value. It comprises the better cost performances of the (s, S) policy and the reduced nervousness of the (R, S) replenishment plan. However, its parameters' computation is considered too complex by the literature; especially in the presence of non-stationary stochastic demand. Classical inventory control literature does not model a cost factor related to stock taking or order cancellation.

Chapter 4 presents the first optimal SDP formulation for the (R, S) policy computation with no assumption of the demand distribution type. To tackle larger instances, we enhance the model computation with a memoisation approach, a filtering technique and binary search. This algorithm can be easily implemented without using external solvers. Extensive computational experiments show that this new approach has a small optimality gap and an accurate expected cost, outperforming the current state-of-the-art.

Chapter 5 introduces the first algorithm for the computation of optimal (R, s, S) parameters. This solution is a hybrid of BnB and SDP. Ad-hoc bounds computed

through DP strongly prune the search tree without compromising the optimality. Having an optimal solution is fundamental for future research on this policy since it allows to compute the optimality gap. In our experimental results, this approach can prune up to 99.8% of the search tree. The algorithm can solve real-world problems in a reasonable time.

Chapter 6 contributes by presenting a set of heuristics for the (R, s, S) problem. These algorithms compute near-optimal policies in a fraction of the time required by the optimal solution. The first one is a pure SDP formulation for the (R, s, S) problem. We enhance this model using the K-convexity property and a memoisation approach. The second is a sequential application of an (R, S) and an (s, S) policy computation algorithms. The heuristics can be used as upper bound for the optimal policy cost in the branch-and-bound solution, further improving the pruning percentage. The experimental section shows that these solutions have a low optimality gap and fail to compute the optimal policy in rare cases. On average, the monolithic SDP has an optimality gap of 0.07 %.

Finally, Chapter 7 presents the dynamic programming encoding (DPE). A novel technique to model DP in CP. We describe the correspondences between the components of a DP algorithm and the elements of a CP model. A set of examples illustrate the deployment of this technique. In the experimental analysis, we compare this approach with a similar encoding designed for MIP solvers. The results show that the DPE is more suitable for CP solvers compared to the encoding available in the literature. This encoding allows to monolithically model the computation of (R, s, S) policy parameters in CP, without ad hoc global constraints. However, this is a negative example for the DPE due to some intrinsic characteristics of the stochastic lot sizing problem, making it uncompetitive with the approaches aforementioned.

In summary, this work contributes to the literature of stochastic inventory control by providing novel approaches to compute optimal and near-optimal policies for a range of widely known and used problems. This thesis covers research questions that are unanswered in the literature, opening to more research path that can be explored. In the next section, we present some of these possible novel contributions to the literature.

8.1 Future works

This thesis opens many different possible paths for future research. We hope that this work will stimulate further investigation on techniques to compute (R, s, S) parameters.

The algorithms presented herein can be used as a comparison to newly developed techniques:

- The (R, S) SDP technique can be used to compute the optimality gap for new (R, S) heuristics.
- The (R, s, S) BnB can be used to compute the optimality gap for new (R, s, S) heuristics that can be compared to the existing heuristics proposed in Chapter 7.

The problem frame can be modelled to consider different problem configurations. Inventory control systems act in a wide variety of real-world settings. A mathematical formulation that closely resembles reality can lead to consistent savings. Possible variations on the problem settings are:

- **Capacitated lot sizing**. When some constraints limit the decisions. For example, establishing a maximum inventory level. In real-world applications, the warehouses have limited capacity. Alternatively, a fixed extra cost is charged when the inventory level exceeds the capacity limit, the cost of renting additional space. Another capacity constraint can regard the order size. There might be a minimum order size; this is common in many B2B environments. The ordering cost can involve a fixed charge for exceeding a container capacity, e.g. [vNvdV05,MMDA16]. It can represent a truck size limitation, every time a truckload limit is exceeded even by a single item another truck has to be used. For example, Chen and Sarker [CS14] deploy a lot sizing solution alongside a vehicle routing problem. A review of the relevant literature can be found in [KGW03].
- **Different cost structure**. Include a quantity discount on the per unit cost. Many businesses offer a lower per item price for bigger orders; since the administrative and preparation costs do not grow linearly with the order size, e.g. [LKL13]. The per unit ordering cost can be modelled as a stochastic non-stationary variable as well. For example, a jewellery producer has to acquire gold from the metal market where the prices

change constantly. The holding cost can change over time. Part of it is represented by the lost investment opportunity that is not constant.

- **Lead time**. Adapt the algorithms to model the lead time. To further generalise the problem, the lead time can be modelled as a stochastic variable, e.g. [RTHP10].
- **Using service level constraints**. No technique is currently available to compute optimal (R, s, S) policy parameters with a service level constraint.
- **Inventory deterioration**. We assume that the inventory can be held in the inventory for an indefinitely amount of time. A false assumption for perishable products. Other items with similar behaviour are goods with quick obsolescence. An example of inventory deterioration is Gunpinar and Centeno [GC15] work on blood supply. Surveys on this particular type of inventory control system can be found in [Raa91, GG01].
- **Remanufacturing**. When the inventory control system manages the production, many real-world settings include remanufacturing. Traditional inventory control models do not take into consideration that the end-user may return the goods. These items can be refurbished or can be disassembled to reuse their components. Recovering these items is strongly beneficial from the environmental point of view. An extensive literature is available for the inventory control problem with remanufacturing, e.g. Van der Lan et al. propose a heuristic to compute (s, Q) policy parameters for such a problem.
- **Multi-item**. In most of the businesses, the warehouses store more than one type of goods. When dealing with multiple products, a supplier can provide more than one. In that case, placing orders that include multiple products can lead to saving on shipping and ordering costs. The problem of managing the inventory of a multi-item system with joint replenishment is not novel in the literature; recent surveys of these efforts can be found in [KG08, BMN$^+$17]. Different policy computation algorithms are available for such inventory system, e.g. [AI88, NL05, ÖGB06]. However, the literature does not contain any (R, s, S) policy computation algorithm considering joint replenishment.
- **Multi-supplier**. We can consider multiple suppliers for the same product. Different suppliers can have different linear and fixed ordering cost. For

example, Janssen and Kok [JdK99] consider a two suppliers situation in which one is managed with a (R, Q) policy and the other with a (R, S) one.

- **Specific type of demand**. The techniques presented in this thesis do not assume the demand type. As showed in Chapter 3, most of the techniques in the literature assume that the demand is normally or Poisson distributed. This assumption limits their applicability. A problem that practitioners often tackle includes a bimodally distributed demand. Scenario analysis uses it. An example can be a company that export goods from China to the US; the forecasted demand strongly depends on the tariffs. If the tariffs are lifted, we expect a normal distributed demand with higher mean compared to the situation with tariffs. We can estimate the probability that the tariffs are lifted and consequentially modelled the demand as a bimodal one.
- **Correlated demand**. Classical stochastic lot sizing models assume that the variables representing the demand are independent. In real-world situations, external factors can affect the demand for multiple periods. Demand correlation can be taken into account in inventory control systems. Various models that model this particular lot sizing are available in the literature, e.g. [SZ93, CS99, DL03].

This is a limited set of examples of the many possible extensions for the inventory control techniques presented in this work. Business challenges have strong diversity and new variants of the problem will arise in the future.

The DPE approach opens a new set of entirely different research challenges. It makes it possible to model some bilevel problems with a compact CP model. Bilevel problems involve two decision-makers that act sequentially pursuing different objectives. A pioneering work in this area is a bilevel scheduling problem, [KK11]. They claim that "no direct encoding of discrete bilevel problems into CP or MIP can be expected". In [PRTV18] we show that the bilevel network interdiction problem [IW02] can be modelled through DPE. This extends to more bilevel problems in which the second decision-maker can be solved in DP; for example, the bilevel knapsack interdiction problem. Another direction is to make available models of widely known constraints solvable through DP, as the grammar constraint [QW07] or the regular constraint [Pes04].

References

[ABKV20] Jose A Lopez Alvarez, Paul Buijs, Onur A Kilic, and Iris FA Vis. An inventory control policy for liquefied natural gas as a transportation fuel. *Omega*, 90:101985, 2020.

[ABTM08] Amir Azaron, Kenneth Brown, S Armagan Tarim, and Mohammed Modarres. A multi-objective stochastic programming approach for supply chain design considering risk. *International Journal of Production Economics*, 116(1):129–138, 2008.

[AHM51] Kenneth J Arrow, Theodore Harris, and Jacob Marschak. Optimal inventory policy. *Econometrica: Journal of the Econometric Society*, pages 250–272, 1951.

[AI88] Derek R Atkins and Paul O Iyogun. Periodic versus "can-order" policies for coordinated multi-item inventory systems. *Management Science*, 34(6):791–796, 1988.

[Ake78] Sheldon B Akers. Binary decision diagrams. *IEEE Transactions on computers*, C-27(6):509–516, 1978.

[AMP18] Taher Ahmadi, Masoud Mahootchi, and Kumaraswamy Ponnambalam. Optimal randomized ordering policies for a capacitated two-echelon distribution inventory system. *Computers & Industrial Engineering*, 124:88–99, 2018.

[AN87] Yash Aneja and A Hamid Noori. The optimality of (s, S) policies for a stochastic inventory problem with proportional and lump-sum penalty cost. *Management Science*, 33(6):750–755, 1987.

[Ask81] Ronald G Askin. A procedure for production lot sizing with probabilistic dynamic demand. *Aiie Transactions*, 13(2):132–137, 1981.

[ASMPB18] Claudio Araya-Sassi, Pablo A Miranda, and Germán Paredes-Belmar. Lagrangian relaxation for an inventory location problem with periodic inventory control and stochastic capacity constraints. *Mathematical Problems in Engineering*, 2018, 2018.

[AW06] Krzysztof R Apt and Mark Wallace. *Constraint logic programming using ECLiPSe*. Cambridge University Press, 2006.

[BADPN17] Nadjib Brahimi, Nabil Absi, Stéphane Dauzère-Pérès, and Atle Nordli. Single-item dynamic lot-sizing problems: An updated survey. *European Journal of Operational Research*, 263(3):838–863, 2017.

[BBB+08] Ralph Becket, Sebastian Brand, Mark Brown, Gregory J Duck, Thibaut Feydy, Julien Fischer, Jinbo Huang, Kim Marriott, Nicholas Nethercote, Jakob Puchinger, et al. The many roads leading to rome: Solving zinc models by various solvers. In *7th International Workshop on Constraint Modelling and Reformulation*, 2008.

[BCR12] Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global constraint catalog, (revision a), 2012.

[BCvH14] David Bergman, Andre A Cire, and Willem van Hoeve. MDD propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50:697–722, 2014.

[BCVHH16] David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

[BDPNN06] Nadjib Brahimi, Stéphane Dauzere-Peres, Najib M Najid, and Atle Nordli. Single item lot sizing problems. *European Journal of Operational Research*, 168(1):1–16, 2006.

[Bel54] Richard Bellman. The theory of dynamic programming. Technical report, RAND Corp Santa Monica CA, 1954.

[Bel66] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[Ber72] William L Berry. Lot sizing procedures for requirements planning systems: A framework for analysis. *Production and Inventory Managements*, 13(2):19–34, 1972.

[BF98] Sridhar Bashyam and Michael C Fu. Optimization of (s, S) inventory systems with random lead times and a service level constraint. *Management Science*, 44(12-part-2):S243–S256, 1998.

[BGJK15] Maxim A Bushuev, Alfred Guiffrida, M Y Jaber, and Mehmood Khan. A review of inventory lot sizing review papers. *Management Research Review*, 38(3):283–298, 2015.

[BGP14] Giuseppe Bruno, Andrea Genovese, and Carmela Piccolo. The capacitated lot sizing model: A powerful tool for logistics decision making. *International Journal of Production Economics*, 155:380–390, 2014.

[BHM77] Stephen P Bradley, Arnoldo C Hax, and Thomas L Magnanti. Applied mathematical programming. 1977.

[Bij14] Marco Bijvank. Periodic review inventory systems with a service level criterion. *Journal of the Operational Research Society*, 65(12):1853–1863, 2014.

[BM99] Srinivas Bollapragada and Thomas E Morton. A simple heuristic for computing nonstationary (s, S) policies. *Operations Research*, 47(4):576–584, 1999.

[BMN+17] Leonardo dos Santos Lourenço Bastos, Matheus Lopes Mendes, Denilson Ricardo de Lucena Nunes, André Cristiano Silva Melo, and Mariana Pereira Carneiro. A systematic literature review on the joint replenishment problem solutions: 2006-2015. *Production*, 27, 2017.

[Bro82] Robert Goodell Brown. *Advanced service parts inventory control*. Materials management systems, 1982.

[BSH18] Esra I Büyüktahtakın, J Cole Smith, and Joseph C Hartman. Partial objective inequalities for the multi-item capacitated lot-sizing problem. *Computers & Operations Research*, 91:132–144, 2018.

[BST10] M Zied Babai, Aris A Syntetos, and Ruud Teunter. On the empirical performance of (T, s, S) heuristics. *European Journal of Operational Research*, 202(2):466–472, 2010.

[BSvDK17] Rob Broekmeulen, Michael G Sternbeck, Karel H van Donselaar, and Heinrich Kuhn. Decision support for selecting the optimal product unpacking location in a retail supply chain. *European Journal of Operational Research*, 259(1):84–99, 2017.

[BT88] James H Bookbinder and Jin-Yan Tan. Strategies for the probabilistic lot-sizing problem with service-level constraints. *Management Science*, 34(9):1096–1108, 1988.

[Büy11] Esra I Büyüktahtakın. Dynamic programming via linear programming. *Wiley Encyclopedia of Operations Research and Management Science*, 2011.

[BV11] Marco Bijvank and Iris F A Vis. Lost-sales inventory theory: A review. *European Journal of Operational Research*, 215(1):1–13, 2011.

[BV12a] Marco Bijvank and Iris F A Vis. Inventory control for point-of-use locations in hospitals. *Journal of the Operational Research Society*, 63(4):497–510, 2012.

[BV12b] Marco Bijvank and Iris F A Vis. Lost-sales inventory systems with a service level criterion. *European Journal of Operational Research*, 220(3):610–618, 2012.

[CC59] Abraham Charnes and William W Cooper. Chance-constrained programming. *Management science*, 6(1):73–79, 1959.

[CCLW16] Alberto Caprara, Margarida Carvalho, Andrea Lodi, and Gerhard J Woeginger. Bilevel knapsack with interdiction constraints. *INFORMS Journal on Computing*, 28(2):319–333, 2016.

[CDLBS12] Geoffrey Chu, Maria Garcia De La Banda, and Peter J Stuckey. Exploiting subproblem dominance in constraint programming. *Constraints*, 17(1):1–38, 2012.

[CL09] Yee Ming Chen and Chun-Ta Lin. A coordinated approach to hedge the risks in stochastic inventory-routing problem. *Computers & Industrial Engineering*, 56(3):1095–1112, 2009.

[Cla99] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[CLRS09] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[CMC+13] Guillermo Cabrera, Pablo A Miranda, Enrique Cabrera, Ricardo Soto, Broderick Crawford, Jose Miguel Rubio, and Fernando Paredes. Solving a novel inventory location model with stochastic constraints and inventory control policy. *Mathematical Problems in Engineering*, 2013, 2013.

[CMS05] Benoît Colson, Patrice Marcotte, and Gilles Savard. Bilevel programming: A survey. *4or*, 3(2):87–107, 2005.

[CO10] Hadrien Cambazard and Barry O'Sullivan. Propagating the bin packing constraint using linear programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 129–136. Springer, 2010.

[CS99] Feng Cheng and Suresh P Sethi. Optimality of state-dependent (s, S) policies in inventory models with markov-modulated demand and lost sales. *Production and operations management*, 8(2):183–192, 1999.

[CS09] Geoffrey Chu and Peter J Stuckey. Minimizing the maximum number of open stacks by customer search. In *International Conference on Principles and Practice of Constraint Programming*, pages 242–257. Springer, 2009.

[CS14] Zhixiang Chen and Bhaba R Sarker. An integrated optimal inventory lot-sizing and vehicle-routing model for a multisupplier single-assembler system with jit delivery. *International journal of production research*, 52(17):5086–5114, 2014.

[Dec99] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[DKI97] Ton De Kok and Karl Inderfurth. Nervousness in inventory management: comparison of basic control rules. *European Journal of Operational Research*, 103(1):55–82, 1997.

[DL03] Lingxiu Dong and Hau L Lee. Optimal policies and approximations for a serial multiechelon inventory system with time-correlated demand. *Operations Research*, 51(6):969–980, 2003.

[Dol87] Robert J Dolan. Quantity discounts: Managerial issues and research opportunities. *Marketing Science*, 6(1):1–22, 1987.

[DSKTR16] Gozdem Dural-Selcuk, Onur A Kilic, S Armagan Tarim, and Roberto Rossi. A comparison of non-stationary stochastic lot-sizing strategies. *arXiv preprint arXiv:1607.08896*, 2016.

[DSRKT19] Gozdem Dural-Selcuk, Roberto Rossi, Onur A Kilic, and S Armagan Tarim. The benefit of receding horizon control: Near-optimal policies for stochastic inventory control. *Omega*, page 102091, 2019.

[Duc93] Leslie K Duclos. Hospital inventory management for emergency demand. *International Journal of Purchasing and Materials Management*, 29(3):29–38, 1993.

[DW60] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[Edg88] Francis Y Edgeworth. The mathematical theory of banking. *Journal of the Royal Statistical Society*, 51(1):113–127, 1888.

[EHVW14] JCF Ehrenthal, Dorothée Honhon, and Tom Van Woensel. Demand seasonality in retail inventory management. *European Journal of Operational Research*, 238(2):527–539, 2014.

[EL13] Deborah K Elms and Patrick Low. *Global value chains in a changing world*. WTO/FGI/TFCTN, 2013.

[EM87] Gary D Eppen and R Kipp Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.

[FLH05] John Forrest and Robin Lougee-Heimer. CBC user guide. In *Emerging theory, methods, and applications*, pages 257–277. INFORMS, 2005.

[FM01] Filippo Focacci and Michela Milano. Connections and integrations of dynamic programming and constraint programming. In *Inter-*

national Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2001.

[Fre97] Eugene C Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, 1997.

[Fre18] Eugene C Freuder. Progress towards the holy grail. *Constraints*, 23(2):158–171, 2018.

[Frü09] Thom Frühwirth. *Constraint handling rules*. Cambridge University Press, 2009.

[GBDG15] Mustafa Göçken, Asli Boru, Ayşe Tuğba Dosdoğru, and Faruk Geyik. (R, s, S) inventory control policy and supplier selection in a two-echelon supply chain: an optimization via simulation approach. In *Proceedings of the 2015 Winter Simulation Conference*, pages 2057–2067. IEEE Press, 2015.

[GC15] Serkan Gunpinar and Grisselle Centeno. Stochastic integer programming models for reducing wastages and shortages of blood products at hospitals. *Computers & Operations Research*, 54:129–141, 2015.

[GCB02] Thomas W Gruen, Daniel S Corsten, and Sundar Bharadwaj. Retail out of stocks: A worldwide examination of extent, causes, and consumer responses. 2002.

[GG65] P C Gilmore and Ralph E Gomory. Multistage cutting stock problems of two and more dimensions. *Operations research*, 13(1):94–120, 1965.

[GG01] Suresh Kumar Goyal and Bibhas Chandra Giri. Recent trends in modeling of deteriorating inventory. *European Journal of Operational Research*, 134(1):1–16, 2001.

[GGR14] Christoph H Glock, Eric H Grosse, and Jörg M Ries. The lot sizing problem: A tertiary study. *International Journal of Production Economics*, 155:39–51, 2014.

[Har13] Ford W Harris. How many parts to make at once. 1913.

[Hei98] Gerald Heisig. Planning stability under (s, S) inventory control rules. *Operations-Research-Spektrum*, 20(4):215–228, 1998.

[Hei99] Susanne Heipcke. Comparing constraint programming and mathematical programming approaches to discrete optimisation—the change problem. *Journal of the Operational Research Society*, 50(6):581–595, 1999.

[Hei01] Gerald Heisig. Comparison of (s, S) and (s, nq) inventory control rules with respect to planning stability. *International Journal of Production Economics*, 73(1):59–82, 2001.

[HJMR09] Woonghee Tim Huh, Ganesh Janakiraman, John A Muckstadt, and Paat Rusmevichientong. Asymptotic optimality of order-up-to policies in lost sales inventory systems. *Management Science*, 55(3):404–420, 2009.

[Hoo02] John N Hooker. Logic, optimization, and constraint programming. *INFORMS Journal on Computing*, 14(4):295–321, 2002.

[HOOS09] Tarik Hadzic, Eoin O'Mahony, Barry O'Sullivan, and Meinolf Sellmann. Enhanced inference for the market split problem. In *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 716–723. IEEE, 2009.

[Hop04] Wallace J Hopp. Ten most influential papers of management science's first fifty years. *Management Science*, 50(12_supplement):1763–1763, 2004.

[HRTP11] Brahim Hnich, Roberto Rossi, S Armagan Tarim, and Steven Prestwich. A survey on CP-AI-OR hybrids for decision making under uncertainty. In *Hybrid Optimization*, pages 227–270. Springer, 2011.

[HSC07] Johnny C Ho, Adriano O Solis, and Yih-Long Chang. An evaluation of lot-sizing heuristics for deteriorating inventory in material requirements planning systems. *Computers & Operations Research*, 34(9):2562–2575, 2007.

[HVH18] John N Hooker and Willem-Jan Van Hoeve. Constraint programming and operations research. *Constraints*, 23(2):172–195, 2018.

[HVHH10] Samid Hoda, Willem-Jan Van Hoeve, and John N Hooker. A systematic approach to mdd-based constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 266–280. Springer, 2010.

[HYHX09] Zhongsheng Hua, Jerry Chih-Yuan Yang, Fu-ling Huang, and Xin Xu. A static-dynamic strategy for spare part inventory systems with nonstationary stochastic demand. *Journal of the Operational Research Society*, 60:1254–1263, 2009.

[Igl61] Donald L Iglehart. Dynamic programming and stationary analyses of inventory problems. Technical report, Stanford University, 1961.

[Igl63] Donald L Iglehart. Optimality of (s, S) policies in the infinite horizon dynamic inventory problem. *Management science*, 9(2):259–267, 1963.

[IW02] Eitan Israeli and R Kevin Wood. Shortest-path network interdiction. *Networks: An International Journal*, 40(2):97–111, 2002.

[JdK99] Fred Janssen and Ton de Kok. A two-supplier inventory model. *International journal of production economics*, 59(1-3):395–403, 1999.

[KG08] Moutaz Khouja and Suresh Goyal. A review of the joint replenishment problem literature: 1989–2005. *European Journal of Operational Research*, 186(1):1–16, 2008.

[KGW03] Behrooz Karimi, S M T Fatemi Ghomi, and JM Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378, 2003.

[KH06] Wook Hyun Kwon and Soo Hee Han. *Receding horizon control: model predictive control for state models*. Springer Science & Business Media, 2006.

[Kje14] Hakan Kjellerstrand. Picat: A logic-based multi-paradigm language. 2014.

[KK11] András Kovács and Tamás Kis. Constraint programming approach to a bilevel scheduling problem. *Constraints*, 16(3):317–340, 2011.

[KS08] Serdar Kadioglu and Meinolf Sellmann. Efficient context-free grammar constraints. In *AAAI*, pages 310–316, 2008.

[KST94] Paul Katz, Amir Sadrian, and Patrick Tendick. Telephone companies analyze price quotations with bellcore's pdss software. *Interfaces*, 24(1):50–63, 1994.

[KT11] Onur A Kilic and S Armagan Tarim. An investigation of setup instability in non-stationary stochastic inventory systems. *International Journal of Production Economics*, 133(1):286–292, 2011.

[LKL13] Amy HI Lee, He-Yau Kang, and Chun-Mei Lai. Solving lot-sizing problem with quantity discount and transportation cost. *International journal of systems science*, 44(4):760–774, 2013.

[LL06] Jianli Li and Liwen Liu. Supply chain coordination with quantity discount policy. *International Journal of Production Economics*, 101(1):89–98, 2006.

[Mar87] R Kipp Martin. Generating alternative mixed-integer programming models using variable redefinition. *Operations Research*, 35(6):820–831, 1987.

[Mar90] Silvano Martello. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion*, 1990.

[MM10] A Ridha Mahjoub and S Thomas McCormick. Max flow and min cut with bounded-length paths: complexity, algorithms, and approximation. *Mathematical programming*, 124(1-2):271–284, 2010.

[MMDA16] Flavio Molina, Reinaldo Morabito, and Silvio Alexandre De Araujo. Mip models for production lot sizing problems with distribution costs and cargo arrangement. *Journal of the Operational Research Society*, 67(11):1395–1407, 2016.

[MPT00] Silvano Martello, David Pisinger, and Paolo Toth. New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research*, 123(2):325–332, 2000.

[MRA19] Xiyuan Ma, Roberto Rossi, and Thomas Archibald. Stochastic inventory control: A literature review. *IFAC-PapersOnLine*, 52(13):1490–1495, 2019.

[MRC90] R Kipp Martin, Ronald L Rardin, and Brian A Campbell. Polyhedral characterization of discrete dynamic programming. *Operations research*, 38(1):127–138, 1990.

[MRF] Ahmed Magdy, Frank Raiser, and Thom Frühwirth. Implementing dynamic programming recurrences in constraint handling rules with rule priorities. Citeseer.

[MS02] JJA Moors and LWG Strijbosch. Exact fill rates for (R, s, S) inventory control with gamma distributed demand. *Journal of the Operational Research Society*, 53(11):1268–1274, 2002.

[MSDKS10] K Marriot, Peter J Stuckey, Leslie De Koninck, and Horst Samulowitz. An introduction to MiniZinc. *University of Melbourne G*, 12:2012, 2010.

[MSVH08] Yuri Malitsky, Meinolf Sellmann, and Willem-Jan Van Hoeve. Length-lex bounds consistency for knapsack constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 266–281. Springer, 2008.

[MY08] Chumpol Monthatipkul and Pisal Yenradee. Inventory/distribution control system in a one-warehouse/multi-retailer supply chain. *International Journal of Production Economics*, 114(1):119–133, 2008.

[NL05] Christina Nielsen and Christian Larsen. An analytical study of the Q (s, S) policy applied to the joint replenishment problem. *European Journal of Operational Research*, 163(3):721–732, 2005.

[ÖDT12] Ulaş Özen, Mustafa K Doğru, and S Armagan Tarim. Static-dynamic uncertainty strategy for a single-item stochastic inventory control problem. *Omega*, 40(3):348–357, 2012.

[ÖGB06] Banu Yüksel Özkaya, Ülkü Gürler, and Emre Berk. The stochastic joint replenishment problem: A new policy, analysis, and insights. *Naval Research Logistics (NRL)*, 53(6):525–546, 2006.

[OPL10] Efficient modeling with the IBM ILOG CPLEX optimization studio. Technical report, IBM Corporation, 2010. White paper.

[Pes04] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *International conference on principles*

and practice of constraint programming, pages 482–495. Springer, 2004.

[Pis97] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[Pis99] David Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47(4):570–575, 1999.

[PN14] Gérard Plateau and Anass Nagih. 0–1 knapsack problems. *Paradigms of Combinatorial Optimization: Problems and New Approaches*, pages 215–242, 2014.

[Pow07] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[PRTV18] Steven Prestwich, Roberto Rossi, S Armagan Tarim, and Andrea Visentin. Towards a closer integration of dynamic programming and constraint programming. In *4th Global Conference on Artificial Intelligence*, pages 202–214, 2018.

[PSS10] Margarita Protopappa-Sieke and Ralf W Seifert. Interrelating operational and financial performance measurements in inventory control. *European Journal of Operational Research*, 204(3):439–448, 2010.

[PTR16] Steve Prestwich, S Armagan Tarim, and Roberto Rossi. Constraint problem specification as compression. In *Global Conference on Artificial Intelligence*, pages 280–292, 2016.

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[QW06] Claude-Guy Quimper and Toby Walsh. Global grammar constraints. In *International conference on principles and practice of constraint programming*, pages 751–755. Springer, 2006.

[QW07] Claude-Guy Quimper and Toby Walsh. Decomposing global grammar constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 590–604. Springer, 2007.

[RA05] Boualem Rabta and Djamil Aïssani. Strong stability in an (R, s, S) inventory model. *International Journal of Production Economics*, 97(2):159–171, 2005.

[Raa91] Fred Raafat. Survey of literature on continuously deteriorating inventory models. *Journal of the Operational Research society*, 42(1):27–37, 1991.

[Raf99] John F Raffensperger. The marriage of dynamic programming and integer programming. In *PIOCS. ORSNZ 34th Annual Conference, Waikato*, page 4, 1999.

[Rég11] Jean-Charles Régin. Global constraints: A survey. pages 63–134, 2011.

[RF] Asma Rakiz and Pierre Fenies. An integrated multi level lot sizing problem with a single railway track transport problem. In *International Workshop on Lot-Sizing-IWLS'2019*, page 30.

[RKT15] Roberto Rossi, Onur A Kilic, and S Armagan Tarim. Piecewise linear approximations for the static–dynamic uncertainty strategy in stochastic lot-sizing. *Omega*, 50:126–140, 2015.

[RR05] N R Srinivasa Raghavan and Debjit Roy. A stochastic petri net approach for inventory rationing in multi-echelon supply chains. *Journal of Heuristics*, 11(5-6):421–446, 2005.

[RTHP08] Roberto Rossi, S Armagan Tarim, Brahim Hnich, and Steven Prestwich. A global chance-constraint for stochastic inventory systems under service level constraints. *Constraints*, 13(4):490–517, 2008.

[RTHP10] Roberto Rossi, S Armagan Tarim, Brahim Hnich, and Steven Prestwich. Computing the non-stationary replenishment cycle inventory policy under stochastic supplier lead-times. *International Journal of Production Economics*, 127(1):180–189, 2010.

[RTHP11] Roberto Rossi, S Armagan Tarim, Brahim Hnich, and Steven Prestwich. A state space augmentation algorithm for the replenishment cycle inventory policy. *International Journal of Production Economics*, 133(1):377–384, 2011.

[RTHP12] Roberto Rossi, S Armagan Tarim, Brahim Hnich, and Steven Prestwich. Constraint programming for stochastic inventory systems under shortage cost. *Annals of Operations Research*, 195(1):49–71, 2012.

[RTPH14] Roberto Rossi, S Armagan Tarim, Steven Prestwich, and Brahim Hnich. Piecewise linear lower and upper bounds for the standard normal first order loss function. *Applied Mathematics and Computation*, 231:489–502, 2014.

[RVBW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[SBF10] Peter J Stuckey, Ralph Becket, and Julien Fischer. Philosophy of the MiniZinc challenge. *Constraints*, 15(3):307–316, 2010.

[Sca59] Herbert Scarf. The optimality of (s, S) policies in the dynamic inventory problem. 1959.

[SFS+14] Peter J Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The MiniZinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, 2014.

[Sil73] Edward A Silver. A heuristic for selecting lot size quantities for the case of a deterministic time-varying demand rate and discrete opportunities for replenishment. *Production and inventory management*, 2:64–74, 1973.

[Sil78] Edward A Silver. Inventory control under a probabilistic time-varying, demand pattern. *Aiie Transactions*, 10(4):371–379, 1978.

[Sil81] Edward A Silver. Operations research in inventory management: A review and critique. *Operations Research*, 29(4):628–645, 1981.

[Sil08] Edward A Silver. Inventory management: An overview, Canadian publications, practical applications and suggestions for future research. *INFOR: Information Systems and Operational Research*, 46(1):15–27, 2008.

[SK97] Babangida Sani and Brian G Kingsman. Selecting the best periodic inventory control and demand forecasting methods for

low demand items. *Journal of the Operational Research Society*, 48(7):700–713, 1997.

[SM+02] Leonardus Wilhelmus Gerardus Strijbosch, Johannes Josephus Antonius Moors, et al. *Simulating an (R, s, S) inventory system*. Tilburg University, 2002.

[SMFD14] Ankur Sinha, Pekka Malo, Anton Frantsev, and Kalyanmoy Deb. Finding optimal strategies in a multi-period multi-leader–follower stackelberg game using an evolutionary algorithm. *Computers & Operations Research*, 41:374–385, 2014.

[Sox97] Charles R Sox. Dynamic lot sizing with random demand and non-stationary costs. *Operations Research Letters*, 20(4):155–164, 1997.

[SPP+98] Edward A Silver, David F Pyke, Rein Peterson, et al. *Inventory management and production planning and scheduling*, volume 3. Wiley New York, 1998.

[SPT16] Edward A Silver, David F Pyke, and Douglas J Thomas. *Inventory and production management in supply chains*. CRC Press, 2016.

[SR91] Helmut Schneider and Dan B Rinks. Empirical study of a new procedure for allocating safety stock in a wholesale inventory system. *International Journal of Production Economics*, 24(1-2):181–189, 1991.

[SRK95] Helmut Schneider, Dan B Rinks, and Peter Kelle. Power approximations for a two-echelon inventory system using service levels. *Production and Operations Management*, 4(4):381–400, 1995.

[SSBJ11] Leo WG Strijbosch, Aris A Syntetos, John E Boylan, and Elleke Janssen. On the interaction between forecasting and stock control: The case of non-stationary demand. *International Journal of Production Economics*, 133(1):470–480, 2011.

[SZ93] Jing-Sheng Song and Paul Zipkin. Inventory control in a fluctuating demand environment. *Operations Research*, 41(2):351–370, 1993.

[Tal13] El-Ghazali Talbi. Metaheuristics for bi-level optimization. *Studies in Computational Intelligence*, 482, 2013.

[Tar96] S Armagan Tarim. *Dynamic lotsizing models for stochastic demand in single and multi-echelon inventory systems*. PhD thesis, PhD thesis, Lancaster University, 1996.

[TBS10] Ruud H Teunter, M Zied Babai, and Aris A Syntetos. ABC classification: service levels and inventory costs. *Production and Operations Management*, 19(3):343–352, 2010.

[TDÖR11] S Armagan Tarim, Mustafa K Dogru, Ulaş Özen, and Roberto Rossi. An efficient computational method for a stochastic dynamic lot-sizing problem under service-level constraints. *European Journal of Operational Research*, 215(3):563–571, 2011.

[THRP09] S Armagan Tarim, Brahim Hnich, Roberto Rossi, and Steven Prestwich. Cost-based filtering techniques for stochastic inventory control under service level constraints. *Constraints*, 14(2):137–176, 2009.

[TK04] S Armagan Tarim and Brian G Kingsman. The stochastic dynamic production/inventory lot-sizing problem with service-level constraints. *International Journal of Production Economics*, 88(1):105–119, 2004.

[TK06] S Armagan Tarim and Brian G Kingsman. Modelling and computing (Rn, Sn) policies for inventory systems with non-stationary stochastic demand. *European Journal of Operational Research*, 174(1):581–599, 2006.

[TKTE11] Huseyin Tunc, Onur A Kilic, S Armagan Tarim, and Burak Eksioglu. The cost of using stationary inventory policies when demand is non-stationary. *Omega*, 39(4):410–415, 2011.

[TKTE13] Huseyin Tunc, Onur A Kilic, S Armagan Tarim, and Burak Eksioglu. A simple approach for assessing the cost of system nervousness. *International Journal of Production Economics*, 141(2):619–625, 2013.

[TKTE14] Huseyin Tunc, Onur A Kilic, S Armagan Tarim, and Burak Eksioglu. A reformulation for the stochastic lot sizing problem with service-level constraints. *Operations Research Letters*, 42(2):161–165, 2014.

[TKTR18] Huseyin Tunc, Onur A Kilic, S Armagan Tarim, and Roberto Rossi. An extended mixed-integer programming formulation and dynamic cut generation approach for the stochastic lot-sizing problem. *INFORMS Journal on Computing*, 30(3):492–506, 2018.

[TMC12] Mounira Tlili, Mohamed Moalla, and Jean-Pierre Campagne. The trans-shipment problem in a two-echelon, multi-location inventory system with lost sales. *International Journal of Production Research*, 50(13):3547–3559, 2012.

[Tri03] Michael A Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. *Annals of Operations Research*, 118(1-4):73–84, 2003.

[TS08] S Armagan Tarim and Barbara M Smith. Constraint programming for computing non-stationary (R, S) inventory policies. *European Journal of Operational Research*, 189(3):1004–1021, 2008.

[Var09] Vicente Vargas. An optimal solution for the stochastic version of the Wagner–Whitin dynamic lot-size model. *European Journal of Operational Research*, 198(2):447–451, 2009.

[VdLDSR96] Erwin Van der Laan, Rommert Dekker, Marc Salomon, and Ad Ridder. An (s, q) inventory model with remanufacturing and disposal. *International journal of production economics*, 46:339–350, 1996.

[VJ66] 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.

[VJW65] Arthur F Veinott Jr and Harvey M Wagner. Computing optimal (s, S) inventory policies. *Management Science*, 11(5):525–552, 1965.

[vNvdV05] Linda van Norden and Steef van de Velde. Multi-product lot-sizing with a transportation capacity reservation contract. *European Journal of Operational Research*, 165(1):127–138, 2005.

[Wal02] Toby Walsh. Stochastic constraint programming. In *European Conference on Artificial Intelligence*, volume 2, pages 111–115, 2002.

[War92] David S Warren. Memoing for logic programs. *Communications of the ACM*, 35(3):93–111, 1992.

[Whi69] Douglas John White. Dynamic programming. Technical report, 1969.

[WW58] Harvey M Wagner and Thomson M Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1):89–96, 1958.

[Xia19] Mengyuan Xiang. Mathematical programming heuristics for nonstationary stochastic inventory control. volume PhD Thesis. The University of Edinburg, 2019.

[XRMBT18] Mengyuan Xiang, Roberto Rossi, Belen Martin-Barragan, and S Armagan Tarim. Computing non-stationary (s, S) policies using mixed integer linear programming. *European Journal of Operational Research*, 271(2):490–500, 2018.

[XRMBT19] Mengyuan Xiang, Roberto Rossi, Belen Martin-Barragan, and S Armagan Tarim. Joint replenishments optimization for the (Rn, Sn) policy. *arXiv preprint arXiv:1902.11025*, 2019.

[ZF91] 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.

[Zhe91] Yu-Sheng Zheng. A simple proof for optimality of (s, S) policies in infinite-horizon inventory systems. *Journal of Applied Probability*, 28(4):802–810, 1991.

[Zip00] Paul Herbert Zipkin. *Foundations of inventory management*. 2000.

[Zip08] Paul Zipkin. Old and new methods for lost-sales inventory systems. *Operations Research*, 56(5):1256–1263, 2008.

[ZKF15] Neng-Fa Zhou, Håkan Kjellerstrand, and Jonathan Fruhman. *Constraint solving and planning with Picat*. Springer, 2015.

[ZKS10] Neng-Fa Zhou, Yoshitaka Kameya, and Taisuke Sato. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 213–218. IEEE, 2010.

List of Symbols

The next list describes several symbols used in the document

Knapsack Problem

C_i optimal profit for packing items $i, \ldots, I$ in a knapsack of size V_i

f_i immediate cost for stage i

I number of items

K capacity of the knapsack

p_i profit of item i

S_i state of the system at stage i

V_i volume of the knapsack not utilised at stage i

v_i volume of item i

x_i binary variable which is set to one if an item i is packed

Lot Sizing

α service level percentage

δ_t binary variable which is set to one if an order is placed in period t

γ_t binary variable which is set to one if the inventory is reviewed in period t

$\hat{C}$ upper bound of the expected cost of the optimal policy

ζ_t value of a random variable d_t

ζ_{tj} value of a random variable d_{tj}

b penalty cost per unit per period

C_t expected total cost of an optimal policy over periods $t, \ldots, T$

d_t random variable representing the demand in period t

d_{tj} random variable representing the demand over periods $t, \ldots, j$. $d_{tj} = d_t + \cdots + d_j$

f_t immediate cost for period t

$G(\cdot)$ cumulative distribution function

$g(\cdot)$ probability density function

h holding cost per unit per period

K fixed ordering cost

MC_t lower bound of the cost of the optimal policy over periods $1, \ldots, t$

Q_t quantity of order placed in period t

R_t number of periods covered by an order placed in period t

S_t order-up-to-level of period t

S_t state of the system at the beginning in period t

T periods in the planning horizon

v ordering cost per unit

W fixed ordering cost

Appendix A

Experiment plots Chapter 4

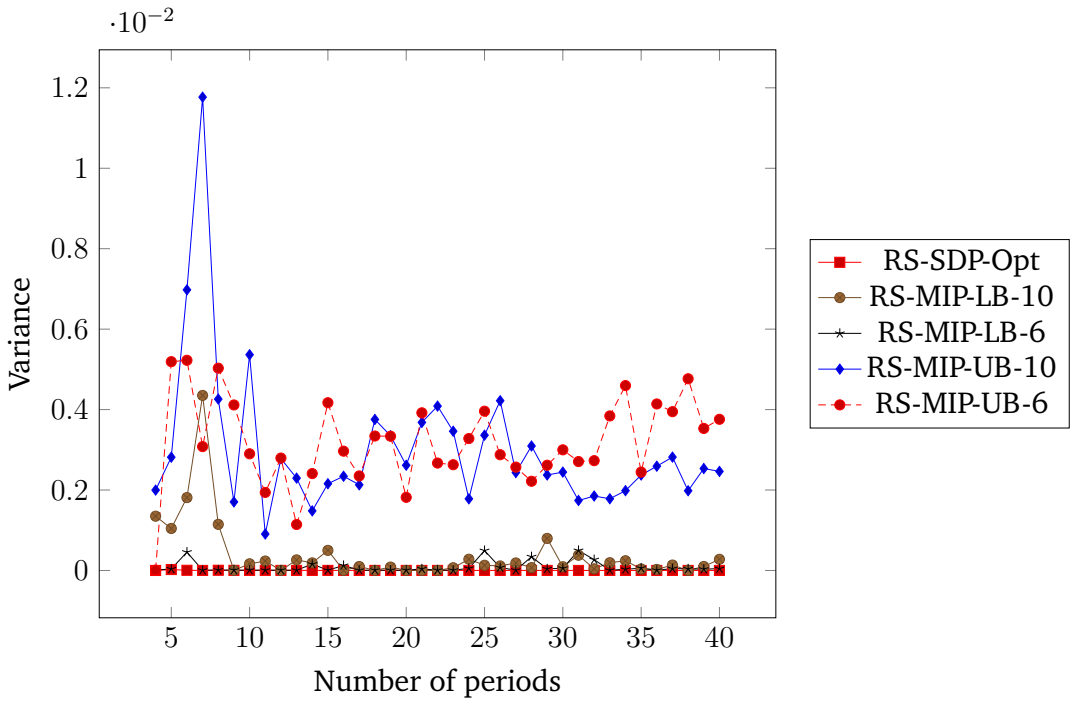

Figure A.1: Variance of the simulated cost optimality gap for $\sigma = 0.1$

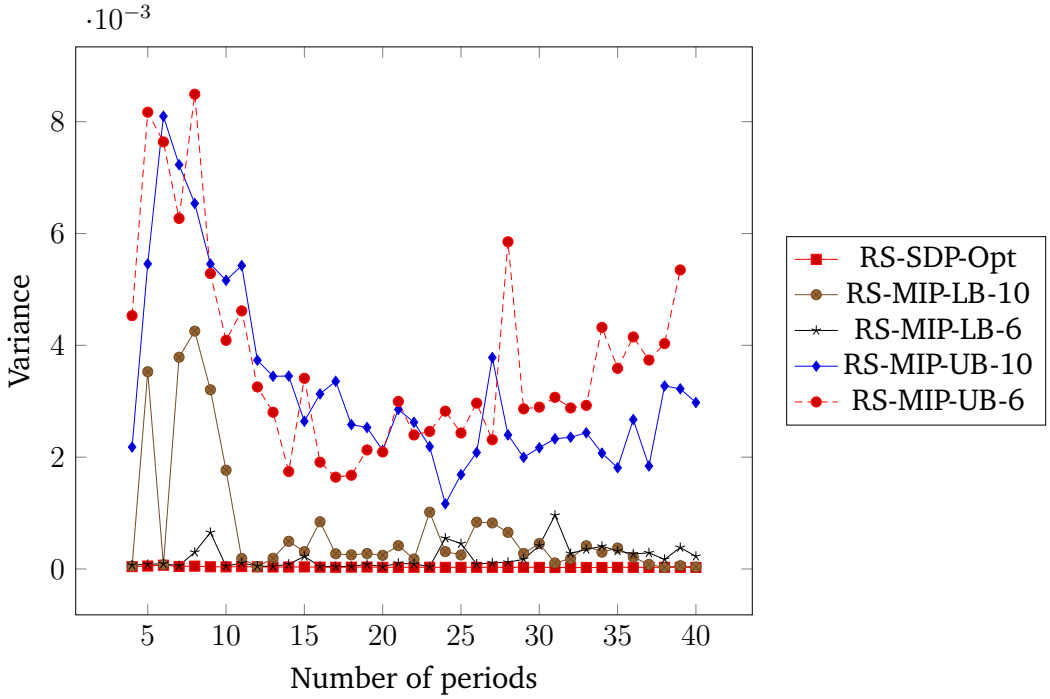

Figure A.2: Variance of the simulated cost optimality gap for $\sigma = 0.2$

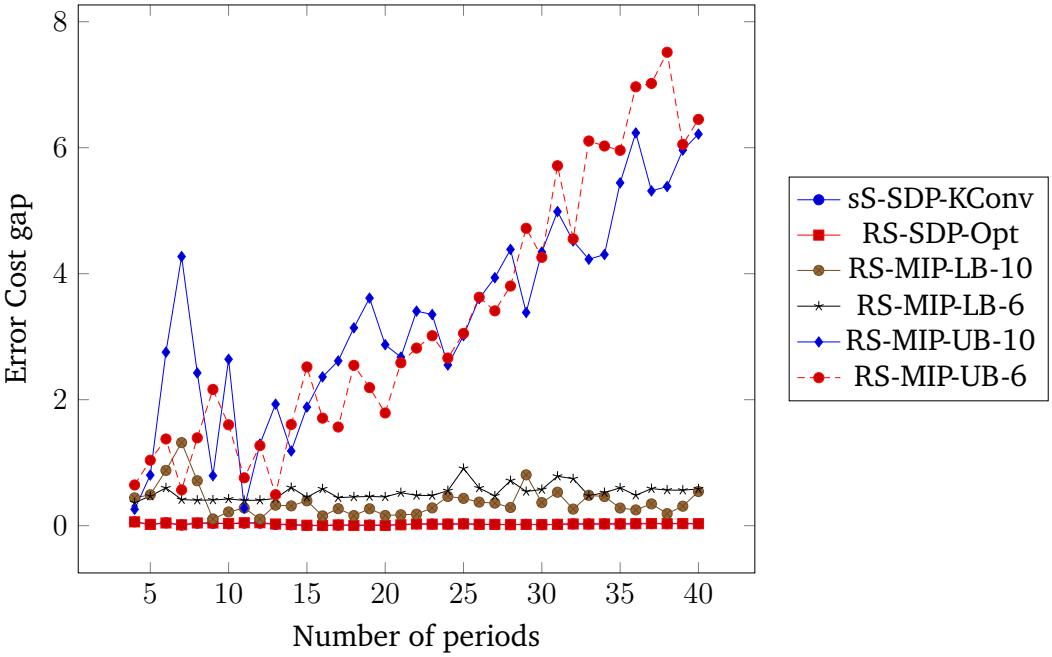

Figure A.3: Expected cost error over the number of periods for $\sigma = 0.1$

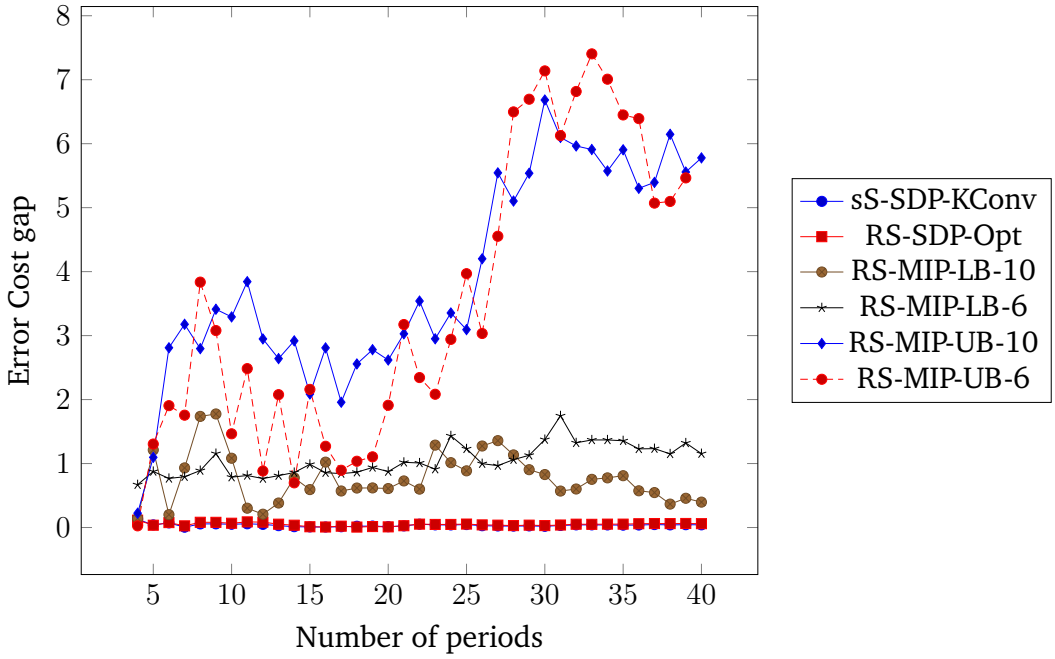

Figure A.4: Expected cost error over the number of periods for $\sigma = 0.2$

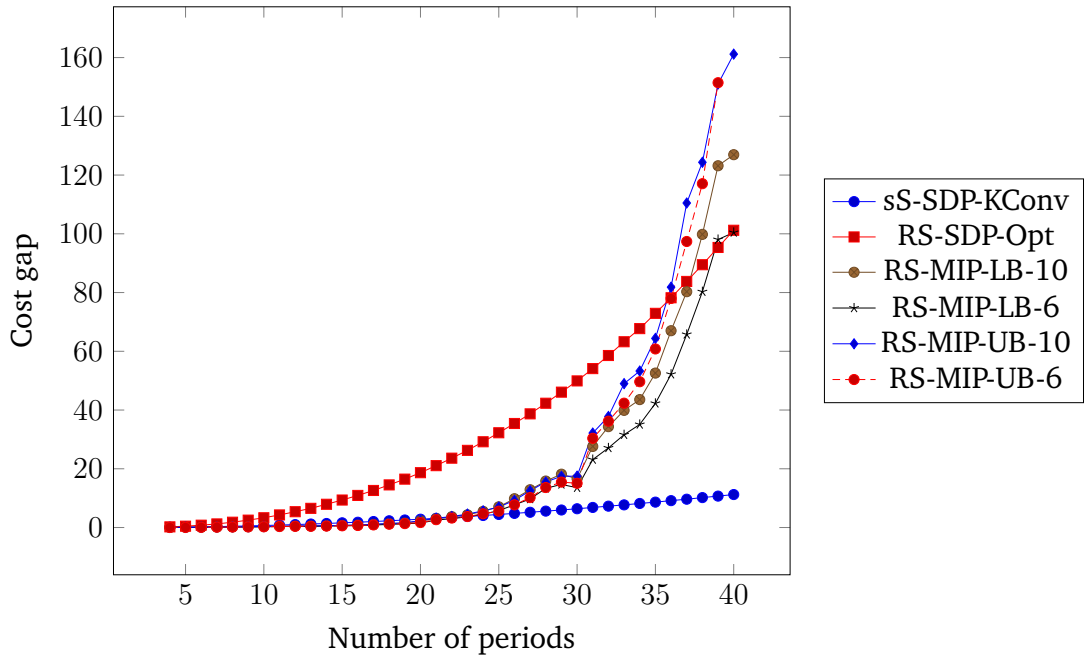

Figure A.5: Computational time in seconds over the number of periods for $\sigma = 0.2$