



Université de Lille
École Doctorale MADIS-631 : Mathématiques, Sciences du numérique et leurs applications

---

Multi-Echelon Inventory Control Optimization Approach for Industrial Applications

---

Approche d'Optimisation de la Gestion des Stocks Multi-Échelon pour des Applications Industrielles

---

Agathe MÉTAIREAU MANCHE

Thèse préparée et soutenue publiquement le ***20 Septembre 2024*** *en vue de l'obtention du grade de* **Docteure en Informatique et Applications**

Membres du Jury:

Mr	Marius BILASCO *Professeur des Universités*	Université de Lille	Président du Jury
Mr	Bernard PENZ *Professeur des Universités*	INP Grenoble	Rapporteur
Mr	Sébastien VEREL *Professeur des Universités*	Université Littoral Côte d'Opale	Rapporteur
Mme	Corinne LUCET *Professeure des Universités*	Université Picardie Jules Verne	Examinatrice
Mme	Clarisse DHAENENS *Professeure des Universités*	Université de Lille	Directrice
Mr	Nadarajen VEERAPEN *Maître de conférences*	Université de Lille	Coencadrant
Mr	Manuel DAVY	Vekia	Coencadrant

Résumé

Dans cette thèse, nous nous intéressons à l'optimisation d'un problème d'approvisionnement multi-échelon. Cette thèse est réalisée dans le cadre d'une convention CIFRE, et le problème est motivé par un cas d'application industriel. Le problème de gestion des stocks est crucial pour le bon fonctionnement des chaines logistiques, et le réapprovisionnement en est une part importante. C'est aussi un problème avec beaucoup de variables de décision et de contraintes, ce qui le rend compliqué à résoudre, spécifiquement pour des jeux de donnée de taille industrielle. Dans la littérature, ce problème est souvent traité de façon mono-échelon. Cela veut dire que les quantités à commander sont calculées pour chacune des entités du système séparément. Cela permet de simplifier le problème, mais occulte certaines contraintes liées aux liens entre les entités du système.

Le but premier de ce travail est de développer une méthode d'optimisation qui permet de résoudre un problème d'approvisionnement multi-échelon. Cette méthode devra être adaptée à des données de taille industrielle, et être applicable pour à des cas réels. Nous nous concentrons sur un réseau avec un entrepôt et plusieurs magasins.

Nous commençons par construire un modèle mathématique en utilisant la programmation linéaire. Ce modèle est basé sur une étude de cas représentative des cas auxquels il devra s'appliquer. Ce cas d'étude nous permet d'identifier et d'expliciter les contraintes pertinentes. Ce travail nous permet d'intégrer des contraintes industrielles dans la modélisation. A partir de ce modèle, nous pouvons faire des expérimentations avec des logiciels existants de résolution exacte, en utilisant des données issues de notre cas d'étude.

Les résultats montrent que les méthodes de résolution exactes sont rapidement limitées lorsque la taille des instances augmente. Nous cherchons donc à mettre en place une méthode heuristique permettant d'obtenir des résultats satisfaisants dans un temps raisonnable. Pour ce faire, nous choisissons de développer une métaheuristique de type recherche locale. Nous mettons en place un ensemble d'opérateurs dédiés spécifiquement au problème étudié. Chaque opérateur agit sur une partie précise de la fonction coût. Etant donné que les performances individuelles des opérateurs ne sont pas satisfaisantes, nous nous orientons vers la mise en place d'une recherche locale utilisant plusieurs opérateurs. Nous mettons en place un algorithme qui permet de sélectionner les opérateurs en se basant sur un score d'efficacité qui évolue pendant la recherche. Plus un opérateur a un score élevé, plus il aura de chance d'être sélectionné pendant la recherche.

Après avoir validé notre algorithme et évalué ses performances d'un point de vue algorithmique, nous nous intéressons aux performances de la méthode d'un point de vue industriel. Dans cette partie du travail, nous cherchons d'une part à évaluer les performances de la méthode dans un contexte réaliste, et d'autre part à valider l'intérêt d'une approche multi-échelon par rapport à une approche mono-échelon. Pour ce faire, nous mettons en place un outil de simulation qui permet de reproduire le processus d'approvisionnement, et de calculer des indicateurs de performances (KPIs). Nous utilisons cet outil de simulation pour reproduire le processus sur plusieurs périodes de temps, en utilisant notre méthode d'optimisation multi-échelon. Nous reproduisons aussi l'expérience avec deux moteurs mono-échelon existant dans l'entreprise, afin de comparer leurs performances. Ces expériences montrent un avantage significatif à l'utilisation d'un moteur multi-échelon, car il permet de réduire les niveaux de stocks et de ventes perdues.

Abstract

In this thesis, we focus on the optimization of a multi-echelon supply problem. This thesis is carried out within the framework of a CIFRE agreement, and the problem is motivated by an industrial case study. The inventory management problem is crucial to the smooth operating of supply chains, with replenishment playing a vital role. It is also a problem with many decision variables and constraints, which makes it complicated to solve, especially for industrial-scale data sets. In the literature, this problem is often treated in a single-step manner. This means that the quantities to be ordered are calculated for each of the system's entities separately. This simplifies the problem, but overlooks certain constraints relating to the links and dependencies between system entities.

The primary aim of this work is to develop an optimization method for solving a multi-echelon supply problem. This method should be adapted to industrial-scale data, and be applicable to real-life cases. We focus on a network with one warehouse and multiple retailers.

We start by building a mathematical model using linear programming. This model is based on a case study representative of the contexts to which it will be applied. This case study enables us to identify and make explicit the relevant constraints. This allows us to integrate industrial constraints into the model. From this model, we can run off-the-shelf exact solvers, using data from our case study.

The results show that exact solution methods are quickly limited as the size of the instances increases. We are therefore looking to develop a heuristic method that will deliver satisfactory results in a reasonable time. To this end, we have chosen to develop a local search metaheuristic. We develop a set of operators dedicated specifically to the problem under study. Each operator acts on a specific part of the cost function. Given that the individual performances of the operators are not satisfactory, we move towards the implementation of a local search using several operators. We implement an algorithm that selects operators based on an efficiency score that evolves during the search. The higher an operator's score, the more likely it is to be selected during the search.

After validating our algorithm and evaluating its performance from an algorithmic standpoint, we now focus on the method's performance from an industrial perspective. In this part of the work, we aim to evaluate the method's performance in a realistic context, and to validate the advantages of a multi-echelon approach over a single-echelon approach. To achieve this, we develop a simulation tool that reproduces the procurement process, and calculates key performance indicators (KPIs). We use this simulation tool to reproduce the process over several time periods, using our multi-echelon optimization method. We also reproduce the experiment with two existing single-echelon engines, in order to compare their performance. These experiments show a significant advantage to the use of a multi-echelon engine, as it reduces inventory levels and lost sales.

Remerciements

A l'issue de ces trois ans – et quelques – de travail de thèse, j'aimerais prendre le temps de remercier toutes les personnes qui m'ont accompagnée, de près ou de loin, dans ce processus.

Pour commencer, je tiens à exprimer ma gratitude sincère à Bernard PENZ et Sébastien VEREL, qui ont accepté de rapporter mon manuscrit. Je souhaite aussi exprimer mes remerciements à Marius BILASCO et Corinne LUCET, qui ont accepté de faire partie de mon jury de thèse. Je vous remercie pour l'attention que vous avez portée à mes travaux, et pour vos précieuses remarques.

Je ne peux évidemment pas écrire de remerciements sans citer mes encadrants académiques, Clarisse DHAENENS et Nadarajen VEERAPEN, pour avoir dirigé et accompagné ces travaux pendant trois ans. Cette thèse n'aurait pas pu être conduite sans vos conseils, votre expertise et vos réflexions. Je pense que ces trois ans représentent la période où j'ai appris le plus de choses dans ma vie, et la plupart de ce que j'ai pu apprendre vient de nos échanges, ce pour quoi je ne peux assez vous remercier. Je tiens aussi à remercier tous les doctorants, post-doctorants et ingénieurs de l'équipe ORKAD, avec qui j'ai pu partager beaucoup de moments d'amitié, dans les rires comme dans la déprime. Trois ans de thèse peuvent paraître longs, et vous avez contribué à rendre mon séjour au laboratoire le plus agréable possible. Je tiens à remercier tout spécialement Thomas FEUTRIER et Clément LEGRAND, mes camarades du bureau S3.04, qui sont au fil du temps devenus mes amis, et qui ont été de véritables compagnons de route dans cette aventure qu'est la thèse.

Je souhaiterais aussi prendre le temps de remercier toutes les personnes que j'ai côtoyées du côté de Vekia. Tout d'abord, je tiens à remercier Manuel DAVY pour m'avoir donné cette opportunité, et accordé sa confience pour conduire ces travaux de thèse. Je souhaite aussi remercier tous mes collègues, actuels ou passés, avec qui j'ai eu le plaisir de travailler pendant ces années, car la bonne entente et l'ambiance régnant au sein de cette entreprise m'a permis de conduire mes travaux de thèse dans les meilleures conditions. Plus spécialement, je tiens à remercier mes deux encadrants industriels, Alexandre GERUSSI et Vincent KNAB, qui se sont investis dans mes travaux et ont grandement participé à faire de cette thèse ce qu'elle est aujourd'hui. Je souhaite aussi adresser un remerciement tout spécial à mes collègues de l'équipe Lab, qui m'ont écoutée raconter mes recherches toutes les semaines pendant trois ans, et m'ont fourni de précieux conseils.

Enfin, et surtout, je souhaite prendre le temps de remercier mes proches, qui m'ont accompagnée et soutenue pendant ces années, quelle que soit mon humeur, ma disponibilité ou ma fatigue. J'aimerais commencer par remrcier mes parents, Véronique et Xavier, ainsi que mon frère Théo, qui m'ont toujours fait confiance, soutenue et encouragée dans ce projet. C'était une idée que j'avais depuis longtemps, et ils n'ont jamais douté de mes capacités à accomplir ce projet. Je ne pourrai pas assez les remercier pour leur soutien sans faille. Je souhaite aussi remercier mes amies, Élodie LORRE, Audrey BAZILLIER, Morgane BOIZIER et Marine LIARD qui, malgré la distance, sont restées des amies très proches et de réels soutiens pendant ces années. Merci de m'avoir soutenue pendant ces années, d'avoir supporté mes indisponibilités, mon incapacité à vulgariser mes travaux, et mes phases de déprime. Je souhaite aussi remercier Amélie MESNIGE-MARC, mon "amie d'internet", que j'ai rencontrée au début de mes travaux, et qui a pu suivre tout le processus de cette thèse. Merci pour tes mots, toujours justes, et pour ton amitié sans faille.

Pour conclure ces remerciements, et parce que je viens de passer quelques lignes à décrire les côtés plus durs de la thèse, j'aimerais ajouter que ces trois années ont été une expérience formidable. J'ai beaucoup appris professionnellement, mais aussi personnellement, et je suis reconnaissante pour ces années, desquelles je garderai un excellent souvenir.

Contents

Introduction . . . 1

1 Problem Description and Motivations **5**
1.1 Introduction . . . 6
1.2 Industrial Context: Vekia Company . . . 7
1.2.1 Demand Forecast . . . 7
1.2.2 Inventory Control . . . 7
1.3 Replenishment Problems . . . 7
1.4 Inventory Control Policies . . . 9
1.4.1 Basic Policies . . . 10
1.4.2 Hybrid Policies . . . 12
1.4.3 Policies with Multiple Quantities . . . 15
1.4.4 Other Policies . . . 16
1.4.5 Unified Notation . . . 17
1.5 Multi-Echelon Approach for Inventory Control . . . 19
1.6 Existing Replenishment Optimization Methods at Vekia . . . 20
1.6.1 Single-Echelon, Single-Item, Single-Period Inventory Control . . . 20
1.6.2 Single-Echelon, Multi-item, Multi-period Inventory Control . . . 21
1.7 Conclusion: Motivations of our Work and Positioning . . . 22

2 Model Formulation and Exact Resolution **25**
2.1 Introduction . . . 26
2.2 Problem Definition . . . 27
2.2.1 Problem Specifications . . . 27
2.2.2 Specific Constraints and Modeling Choices . . . 28
2.3 Proposed Mathematical Model . . . 31
2.3.1 Linear Programming . . . 31
2.3.2 Data and Notations . . . 32
2.3.3 Model . . . 34
2.4 Input Data and Problem Instances . . . 36
2.4.1 Additional Elements for Data Completion . . . 37
2.4.2 Instance Construction . . . 37
2.5 Experimentations . . . 38
2.5.1 Experimental Protocol . . . 39
2.5.2 Results and Discussion . . . 40

2.6 Conclusion . . . 41

3 Heuristic Resolution **43**
3.1 Introduction . . . 44
3.2 Resolution Requirements and Method Selection . . . 45
3.3 Local Search: Overview . . . 47
3.3.1 Solution Representation . . . 48
3.3.2 Initial Solution Generation . . . 48
3.3.3 Neighborhood Operators . . . 52
3.4 Dedicated Neighborhood Operators . . . 52
3.4.1 Addition Neighborhood Operators . . . 53
3.4.2 Deletion Neighborhood Operators . . . 55
3.4.3 Rescheduling Neighborhood Operators . . . 57
3.4.4 Saturate Franco Neighborhood Operator . . . 58
3.4.5 Overview . . . 58
3.5 Use of Multiple Neighborhood Operators in the Search . . . 59
3.6 Variable Adaptive Local Search . . . 62
3.6.1 Overview of the Algorithm . . . 62
3.6.2 Score Initialization and Computation . . . 64
3.6.3 Neighborhood Operator Selection . . . 64
3.6.4 Score Evolution . . . 65
3.6.5 Termination criteria . . . 65
3.7 Experimentations . . . 67
3.7.1 Method Validation . . . 67
3.7.2 Global Performances . . . 69
3.8 Conclusion . . . 69

4 Performance Analysis in Realistic Setting **71**
4.1 Introduction . . . 72
4.2 General Description . . . 73
4.2.1 Inventory Control Process . . . 73
4.2.2 Order Engines . . . 73
4.2.3 Rolling Horizon . . . 74
4.3 Simulation Context Construction . . . 75
4.3.1 Initialization . . . 76
4.3.2 Initial Solution Computation During Simulation . . . 77
4.3.3 Environment . . . 79
4.3.4 Instance . . . 80
4.4 Simulation Process . . . 81
4.4.1 Decision Agent . . . 82
4.4.2 Sampler . . . 82
4.4.3 Applicator . . . 82
4.4.4 Monitor . . . 83
4.4.5 Shifter . . . 84
4.5 Simulator Overview . . . 84
4.6 Experimentations . . . 86

4.6.1 Experimental Protocol 86
4.6.2 Results Analysis 86
4.6.3 Conclusion on the Results 91
4.7 Conclusion 92

General Conclusion and Perspectives **95**

A Simulation Extensive Results **101**

B Alternative Variable Local Searches **107**

C Convergence curves for the experimentations with *VALS* **109**

Bibliography **130**

List of Figures

1 Inventory management process and implications. 1

1.1 Simple representation of a supply chain network. 6
1.2 Schematic representation of the inventory level evolution of an item using the Economic Order Quantity ICP. 10
1.3 Schematic representation of the inventory level evolution of an item using the Order-up-to ICP. 11
1.4 Schematic representation of the inventory level evolution of an item using the Reorder Point ICP. 12
1.5 Schematic representation of the inventory level evolution of an item using the Periodic Reorder Point ICP. 13
1.6 Schematic representation of the inventory level evolution of an item using the Reorder Point Order-Up-To ICP. 14
1.7 Schematic representation of the inventory level evolution of an item using the Periodic Reorder Point Order-Up-To ICP. 15
1.8 Schematic representation of decision process using *SPSE*. In this case, every ordering decision is made by one separate run of the algorithm. 21
1.9 Schematic representation of the decision process using MP^2. In this case, we have to run the algorithm for each entity separately, but one algorithm produces ordering decisions for all items and periods simultaneously. 22
1.10 Representation of the decision process using a multi-echelon order-engine. In this setting, the ordering decisions for all entities, items and periods of the instance are taken simultaneously. 23

2.1 Representation of the case study supply network. Solid lines represent the classic 1WNR flows, dotted lines represent the dual-sourcing flows. Active entities are in the square. 27
2.2 Schematic representation of the frozen horizon concept. 30
2.3 Representation of the elements added to an instance for the inclusion of an item supplied by an existing supplier and sold in a new store exclusively. 36
2.4 Graphical representation of the execution times in seconds obtained with CPLEX for every instance. 40

3.1 Decision tree for choosing a suitable method. We provide only the options that relate to our problem. Grey cells relate to the characteristics of our problem. 46
3.2 Schematic solution representation for a given time period t. 48

3.3 Representation of the different groups of variables. 49
3.4 Representation of a constraint violation for a given time period and a given item. All lead times are set to 0 in this example. 50
3.5 Representation of the first repair procedure for a given time period and a given item. All lead times are set to 0 in this example. 51
3.6 Representation of the second repair procedure for a given item and a given time period. We assume all lead times are set to 0, and Store 2 has been picked twice during the repair procedure. 52
3.7 Convergence curves for both algorithms: individual Local Search with *DesF* on the left, VNS-inspired algorithm on the right. For the right curve, a vertical bar represent a modification of neighborhood operator. 61
3.8 Convergence curves for both algorithms: VNS-inspired algorithm with neighborhood operators in decreasing performance order on the left, VNS-inspired algorithm with neighborhood operators in increasing performance order on the right. Vertical bars represent modifications of neighborhood operator. 61
3.9 Schematic representation of *VALS* execution. 63
3.10 Schematic representation of the evolution of the sliding window through the execution of *VALS*. 66
3.11 Boxplots for the relative gaps results of the comparison of the algorithms *U-VLS*, *VLS*and *VALS*, aggregated in all instances. 68

4.1 Description of the inventory control process on a given period with insight on the inventory level evolution. 74
4.2 Representation of the rolling horizon concept. 75
4.3 Description of the simulation protocol steps. 76
4.4 Representation of solution cost function values during the simulation experiment with *ME-LS* and information exploitation computation method. The '×' markers are for initial solutions and the '•' markers are for final solutions. Zero values are the week-ends. 78
4.5 Comparison of solution cost function values during the simulation experiment with *ME-LS* and information exploitation/step zero computation method. The '×' markers are for initial solutions and the '•' markers are for final solutions. Zero values are the week-ends. 79
4.6 Inventory levels at each simulated period averaged on all instances and seeds, for every problem size. Values are split for warehouse and store inventory levels. 87
4.7 Lost sales levels at each simulated period for all instances and seeds, for every problem size. Each curve represent an order engine. 88
4.8 Total costs at each simulated period averaged on instances and seeds, for every problem size. Costs are split in four types: Warehouse purchase and transport, Stores purchase and transport, Franco and Fixed costs. 89
4.9 Amount of orders placed by the stores at each simulated period averaged on all instances and seeds for every problem size. Orders are split between orders to the warehouse and orders to the suppliers. 90
4.10 Execution time in seconds of solvers distribution on all period of one simulation and all problem sizes with generation mode `fora`. 91
4.11 Timeline and summary of the perspectives. 99

List of Tables

1.1 Final notation and classification of Inventory Control Policies. 18

2.1 Forecast profile distribution overview. At each line, we select items that meet the forecast limit requirement for at least one period and in at least one store. 37
2.2 Overview of all generation modes for instances and their characteristics. 39
2.3 Execution time in seconds for solving instances with CPLEX, 7200 second time limit, 1% relative gap limit. The note "Time out" means the solver did not reach the gap within the given time span. 40
2.4 Relative gaps found by the solver for all instances. The notation "*Not found*" means the solver did not find any solution to compute the gap. The notation "$\sim$ 1 %" means the solver reached the gap limit within the time limit. 41

3.1 Performances of addition neighborhood operators averaged on the 10 runs for every instance. Higher values are better. 55
3.2 Performances deletion neighborhood operators averaged on the 10 runs for every instance. 56
3.3 Performances of rescheduling neighborhood operators averaged on the 10 runs for every instance. 58
3.4 Performances of neighborhood operator *Sat* averaged on the 10 runs for every instance. 59
3.5 Performances of all neighborhood operators averaged on all instances and arranged by ascending order of X^*. 60
3.6 Average gaps from the initial solution using *VALS*. 69

4.1 Description of the elements defined in the environment object of the simulation tool. 80
4.2 Description of the elements defined in the instance object of the simulation tool. . . 81
4.3 Description of the dataframes built during the instance creation process. 81

Introduction

Motivations

In today's industrial landscape, companies must make significant efforts to enhance the efficiency, resilience, and performance of their supply chain management to remain competitive. The inventory control and replenishment functions play a crucial role in the optimization of supply chains. As a matter of fact, managing the inventory is at the core of the supply process, as it directly impacts the customer satisfaction and global costs. Moreover, it can also impact other aspects of the supply chain management process, such as network design or logistics activities for instance. In Figure 1, we provide an overview of the inventory management process and its implications in supply chain management. Inventory control optimization for real-world applications is a great challenge for organisations, although the field has been widely studied in the literature. Due to the high number of business constraints and specificities existing in practice, the field still offers significant room for the development of optimization methods and recommendation systems that can be directly applied in industrial contexts.

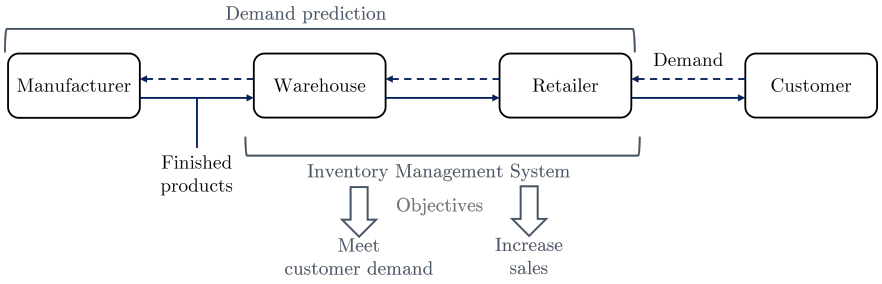


Figure 1: Inventory management process and implications.

Inventory control optimization methods aim at answering the questions faced by inventory managers on a daily basis, namely what quantity of products to order, and when to place the orders. These decisions have to be made given uncertain customer demands, lead times and sometimes even quantities supplied. Most of the research dealing with inventory control propose methods to compute relevant orders for all entities of the supply chain separately. Although this method has many benefits, one of them being the simplicity of the resulting problem to solve, it also presents many limitations. As a matter of fact, all entities of a supply chain are linked to each other, and there exist constraints that can only be treated by considering all entities jointly in the inventory

control optimization. Such an approach is called multi-echelon inventory optimization.

Context of the Thesis and Objectives

This thesis focuses on the development of optimization methods for improving the inventory control in one-warehouse multiple retailers supply chain by proposing a multi-echelon optimization method. The research presented in this dissertation has been conducted under an industrial agreement for training through research (CIFRE). This work is the result of a collaboration between the CRIStAL Laboratory (University of Lille) and the company Vekia. Vekia is a company located in Lille, that was founded based on the idea that artificial intelligence and optimization methods are powerful tools to achieve better performance through supply chain automation. The company offers a cloud-based Software as a Service (SaaS) platform to provide both a forecast for the customer demand and a recommendation system for inventory control. In the laboratory, the thesis is conducted within the ORKAD team (Operational Research, Knowledge and Data). The team focuses on simultaneously using combinatorial optimization and knowledge extraction to address optimization problems. The synergy between these two research fields offers an opportunity to enhance the performance of operational research methods through knowledge, and to efficiently solve knowledge extraction problems with operational research methods.

One of the objectives of our work is to design a new optimization method to strengthen the existing inventory control optimization methods of the company. This new optimization method will be grounded by a study of real-world inventory control contexts. As a result, the model will include realistic characteristics and constraints, which is a known research gap in the multi-echelon inventory control literature. In this research project, the company provides knowledge and data relating to the problem under study, which ensures the results of the work are applicable for the customers and their contexts. The laboratory, for its part, provides the optimization expertise and the academic knowledge. This allows us to guide our work towards suitable optimization methods for the problem, and to produce an academically relevant research. Although the main contributions of the research presented in this manuscript can be applied in a general context, the work mainly focuses on the contexts encountered by Vekia's customers.

Overall, the research we present in this dissertation aims at investigating the inventory control problem, from both an academic standpoint and for an industrial application. The main focus of the research is to develop a resolution method that can efficiently solve a complex optimization problem while being applicable to real-world contexts, based on a realistic inventory control model. The main research questions motivating this work are the following. First, we want to uncover the characteristics and the specific constraints faced by inventory managers, in order to propose a model and a resolution method that can answer their needs. Second, we want to demonstrate the benefits of a multi-echelon inventory control approach over a single-echelon one.

Outline of the Manuscript

In Chapter 1, the inventory control problem under study is presented in more detail. We introduce the classical approaches to solve such problems, and survey the literature dealing with inventory control problems. One of the best known inventory control optimization method relies on the use of Inventory Control Policies. There exist many of Inventory Control Policies, that can be applied to different contexts. However, we could not find a reference summarizing the literature around this topic. Consequently, we propose a classification for Inventory control policies, and a unified notation. We then focus on the multi-echelon inventory control specifically, and outline the main research gaps identified in the literature, one of them being the lack of models including industrial and realistic constraints. The industrial context and the expectations of the company are also detailed, highlighting the need of a multi-echelon model and resolution method. All of this leads to the conclusion of the chapter, expressing of the motivations for the work presented, and its positioning in relation to the literature and the company's expectations.

In Chapter 2, we focus on the presentation of the mathematical model developed for the research work. We detail the exact context that is studied, and list all the specific constraints and characteristics that the model must take into account. Most of these constraints are coming from the study of a real-world inventory control context, for which we provide an overview. This case-study gives us industrial data, that will be used for all experimentations throughout this research. We provide insight on this industrial data, and detail how we manipulate them to extract instances for our problem. To build the mathematical model, we choose to use Linear Programming, and we provide an overview of the history of this language, and its advantages for the case we study. Then, we introduce the notations and the mathematical model that serve as a basis for the optimization. A first experimentation is presented for the exact resolution using instances drawn from the industrial data. These instances are very small compared to the actual data volume of the case-study, and the experimentations show a clear limitation in the use of an exact resolution method, as the execution time quickly increases. These results motivate the use of an heuristic method.

In Chapter 3, we describe the heuristic resolution method designed for the inventory control problem under study. We first motivate the selection of the resolution method given the problem type and the industrial requirements, leading us to the selection of a local search algorithm. Then, the need for the development of dedicated neighborhood operators is explained. We describe the ones built for our problem, and provide an overview of their performances in a classic local search. These results are not satisfying, and lead us to the use of a more sophisticated metaheuristic, still based on the local search principles, but using several neighborhood operators. We then explain the final metaheuristic selected for the resolution, which is a local search using multiple neighborhood operators, and selecting them according to their relevance in the search. The details of the algorithm design are carefully described for every step of the optimization process. The method is then validated through experimentation, by comparing its performances to alternative algorithms. Finally, we provide performance results for the selected resolution method with instances drawn from the industrial data.

In Chapter 4, the focus is put on validating the benefits of the multi-echelon approach compared to single-echelon approaches. In this chapter, we aim at providing business-oriented performance measures for our proposed solution, and at comparing it with existing resolution methods in the company. This validation phase is achieved by developing a simulation tool, that aims at reproducing the industrial control process over a given time horizon. We first detail the motivations for using a simulation tool, and its benefits. This tool is built specifically four our problem, and is composed

of several components, that allow us to imitate the actual replenishment process and to compute key performance indicators. We detail every component of the simulation tool, how they work and how they interact with each other. We then provide experimentation results on the comparison of different inventory control optimization methods, and conclusions on the benefits and limitations of all approaches. These results show a clear benefit on the use of a multi-echelon approach, as it achieves a better inventory level versus lost sales balance in every instance of the experiment.

Finally, this manuscript is concluded by an overview of the contributions proposed in the previous chapters. This overview outlines the structure of each chapter, the research questions they address, and the significant results obtained. We also offer promising perspectives in both academic and industrial contexts. These perspectives are intended to deepen and extend the research conducted during this thesis, and are presented according to the anticipated time-horizons to achieve them.

Chapter 1

Problem Description and Motivations

Contents

1.1 Introduction 6
1.2 Industrial Context: Vekia Company 7
1.2.1 Demand Forecast 7
1.2.2 Inventory Control 7
1.3 Replenishment Problems 7
1.4 Inventory Control Policies 9
1.4.1 Basic Policies 10
1.4.2 Hybrid Policies 12
1.4.3 Policies with Multiple Quantities 15
1.4.4 Other Policies 16
1.4.5 Unified Notation 17
1.5 Multi-Echelon Approach for Inventory Control 19
1.6 Existing Replenishment Optimization Methods at Vekia 20
1.6.1 Single-Echelon, Single-Item, Single-Period Inventory Control 20
1.6.2 Single-Echelon, Multi-item, Multi-period Inventory Control 21
1.7 Conclusion: Motivations of our Work and Positioning 22

1.1 Introduction

An industry, or industrial system, can be defined as a company whose economic activities combine production factors to produce goods for the market. The supply chain is the business function that manages the physical flow of consumer goods from its entry in these industrial system until its delivery to the final customer. It comprises several classes of activities, such as demand planning, supply planning, logistics, finance or information system management. Its efficient operation directly impacts the global performance of the company in various aspects (*e.g.* economical, environmental, *etc.*). As a result, all the decisions made by supply chain managers are crucial in the industry's strategies.

In a distribution context, *i.e.* if the industrial system treats only already finished products, the supply chain's function is to drive these products from the supplier to the end consumer, and the goods pass through storage facilities, transportation modes and retailers. To describe the structure of the majority of supply chains, we can use the term of Supply Chain Network (Chopra et al. [2013]). This network includes all the physical facilities of the chain, along with the flow patterns that link these facilities together to transfer the products. We provide a simplified representation of a distribution supply chain network in Figure 1.1. To describe the different levels of a supply chain network, we refer to each one of them as an *echelon*.

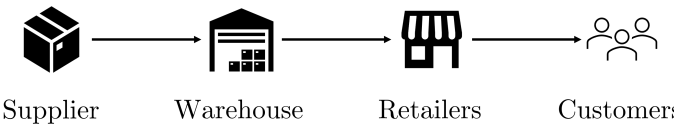


Figure 1.1: Simple representation of a supply chain network.

Supply chain management can be defined as a series of processes relating to a network of interdependent facilities that aim to improve the flow of goods, information and services from initial supplier to final customer (Simchi-Levi [2000]). It includes a large panel of various decisions that involve different actors and different entities of the industrial systems. The aim of this function is to make products available in the right quantities, at the right locations and at the right times with minimal costs and maximal customer satisfaction.

One major activity to manage this availability of products is inventory control. Inventory control, or replenishment, refer to the decisions made in order to maintain a given inventory level in the entities of the supply chain network. Moreover, inventory related decisions and operations represent a major part of supply chain costs (Sahu [2020]). In the work we present, we focus on inventory control optimization problems, and develop methods to apply in industrial contexts.

In this chapter, we first introduce the industrial context we work with and describe the company funding the thesis in Section 1.2. We then define inventory control and replenishment problems and present the literature related to these problems in Section 1.3. Inventory control policies is a traditional way to deal with replenishment problems and is widely used in the literature. In Section 1.4, we list the usual policies used in the literature and propose a classification. In Section 1.5, we focus on the multi-echelon approach for replenishment problems, and provide insights on the existing literature on this topic. In the company, there already exist optimization methods for replenishment problems. We detail them and their scope in Section 1.6. Combining all the concepts, tools and literature presented in all the sections, we conclude this chapter by providing motivations and positioning for our work in Section 1.7.

1.2 Industrial Context: Vekia Company

As this thesis is conducted with a CIFRE contract, the research presented is the result of a collaboration between a research lab and a company. The company that funds the thesis and provides context for our work is Vekia. This company, founded in 2007, provides cloud-based software as a service (SaaS) services for supply chain automation. These services are directed towards various customers from different supply chains and market types. This motivates the need to develop generic tools that can be applied to diverse contexts and that can be easily adapted. The company develops a collection of functionalities directed towards supply chain automation and decision makers support. These functionalities are integrated into a platform that is the product commercialised by the company. The functions provided by the platform are divided into two categories: demand forecasting functions, and inventory control functions.

1.2.1 Demand Forecast

The forecasting functionality is intended to provide an estimate of future external demand. This estimation is based on historical demand and exogenous data. Is is computed using time-series-based techniques and machine learning algorithms. Due to high variability in the demand and diversity in the customer's behavior, perfect demand forecast is impossible. Given this, providing a deterministic forecast, which gives a single predicted value, is limited. To answer this limitation, the demand value predicted is usually accompanied with a certain form of uncertainty. There are plenty of ways to evaluate and express uncertainty. At Vekia, the way chosen to express uncertainty on the forecast is to provide a probabilistic forecast composed of a set of likely samples of the future demand. In this thesis, we will focus on inventory replenishment optimization. As a result, we will use the already existing forecasting methods of Vekia, and consider the forecasted demand as input data for our work.

1.2.2 Inventory Control

The inventory control function aims mainly to compute the best orders to place for each time period in order to maintain a certain inventory level. These decisions are computed based on the demand forecast. The main objectives are to answer the demand as best as possible, taking into account the forecast and its uncertainty, together with minimizing the remaining inventory at the end of the period and minimizing costs. As mentioned before, this part of supply chain management is crucial and represents a major part of cost optimization. Vekia provides several tools to optimize inventory control and replenishment that we describe further in Section 1.6. In this thesis, we focus on inventory control problems, and one of our goals is to develop an order optimization algorithm that answers some limitations of the existing ones.

1.3 Replenishment Problems

As we mentioned in the previous sections, the focus of the work we present is on inventory control and replenishment problems, or as it is more frequently called in the literature, inventory control, or inventory management.

Inventory control refers to the decisions relating to the amount of items kept within the supply chain in order to satisfy the customer demand. It focuses on the trade-off between meeting this

customer demand and maximizing the profitability of operations (Ma et al. [2019]), although the demand may not be known in advance. This can also be described as decisions aimed at finding the best compromise for providing a given customer service level with a minimal cost (Bushuev et al. [2015]). To find this compromise, decision makers mainly have to make three fundamental choices (Silver [1981]):

- How often should the inventory level be observed?
- When should an order be placed?
- What quantity of items should be observed?

Optimizing inventory management aims at finding answers to these questions while optimizing a certain number of criteria. The main focus is to have the minimal cost possible. In a distribution setting, this cost usually includes the purchase and transport costs of the goods, the inventory holding cost and the shortage cost. Depending on the problem under study, other costs can be included such as disposal costs for perishable items, maintenance costs in machine repair contexts, recycling costs in case of reusable items, *etc.*

Inventory management is central in supply chain management, and has many interactions with other areas of operations management, such as production scheduling, raw material provisioning, facility location and sizing, *etc.* As a result, these decisions have a huge impact on the efficiency and profitability of industries, and should receive particular attention.

The literature dealing with optimizing the inventory management decision is extensive, initiated by the academic study of Harris [1913]. This research focuses on a simple deterministic inventory system and aims at deriving the optimal decision rule for answering the three aforementioned questions, and introduces the first known inventory control policy, the *Economic Order Quantity*. In the years following the publication of this paper, the literature related to inventory models quickly expanded to extend and adapt Harris's model to various cases, but also to develop new inventory models in different contexts and with diverse assumptions. The research around inventory control models and their applications gained attention from the 1950s to the 1990s, leading to the publication of a significant number of reviews on the topic in this time span (Whitin [1954], Veinott Jr [1966], Newberry [1960], Aggarwal [1974], Nahmias [1978], Silver [1981], Prasad [1994]), or more recently (Glock et al. [2014], Drake and Marley [2013]). In the literature, we can also find plenty of reviews examining the research conducted and identifying gaps for several given contexts of inventory management, such as spare parts inventory control (Bhalla et al. [2021]), perishable items (Bakker et al. [2012], Khanlarzade et al. [2014]), inventory management multiple supplying sources (Svoboda et al. [2021]), or transshipment assumptions (Paterson et al. [2011]). All of this research leads to a broad bibliography dealing with various themes and academic disciplines. A more comprehensive review on the literature related to inventory control models is provided in Bushuev et al. [2015].

A traditional approach to deal with inventory management decisions is to focus on separated sub-problems of the global inventory control system. In other words, researchers often provide decision support systems for single-entity and even single-item contexts. As a result, the classical approach to provide decision support system for inventory control decision makers is to provide optimization methods for a single entity. This method is intended to be replicated for all entities of the supply system. This approach can be referred to as the *mono-echelon approach*.

Moreover, a large part of the literature focuses on adapting existing decision rules to various inventory control contexts. These decision rules, referred to as *Inventory Control Policies*, are

central in inventory management research, and are the basis of most research conducted in the area. There are several Inventory Control Policies that are known and broadly used in the literature. We provide a list of these Inventory Control policies and a description of each of them in Section 1.4. All of them have the same global form, and provide a small set of parameters that direct the quantities ordered and the moments to place an order. They have the advantage of being very simple to use and to explain, but they are on the other hand very rigid and do not adapt in quickly changing contexts.

Inventory models are usually classified according to the customer demand profile that the entities under study face. This demand can be constant or variable. Moreover, inventory models are separated in two categories: stochastic and deterministic. These categories refer to the assumptions on the prior knowledge of the system's characteristic, such as demand, lead time, quantities supplied, *etc.* Deterministic models assume that these characteristics are known perfectly, while stochastic models assume a certain form of uncertainty in one or several characteristics. This leads to more complicated mathematical models, and this implies that generic mathematical conclusions are more difficult to derive, but at the same time the models are closer to the reality of supply chain management.

In most of the articles, both of these classifications of inventory systems (fixed or variable demand, deterministic or stochastic model), along with further assumptions and descriptive features of the system under study, lead to the choice of an Inventory Control Policy. The aim of the research is then to find a method to compute the parameters of the chosen policy efficiently, while optimizing a given set of criteria and satisfying the system's constraints.

Another way to approach mono-echelon inventory replenishment problems is to treat the items jointly. This type of problems is referred to as joint replenishment problems, and reviews of the academic work on such problems is provided in Goyal and Satir [1989], and more recently in Peng et al. [2022]. In this approach, the goal is to compute the best orders for several items at the same time. In such problems, in addition to optimize the classical costs that we mentioned above, the aim is also to minimize costs that specifically apply to group of items. For instance, in some cases the supplier adds a fixed cost to every order that can represent the transport cost or the preparation cost. In such situation, minimizing this cost requires computing the quantities for all items included in the order jointly.

1.4 Inventory Control Policies

As mentioned in the previous section, Inventory Control Policies are an essential part of the solutions provided in the literature for optimizing inventory management and replenishment optimization. An Inventory Control Policy, or ICP, is a set of decision rules guiding the inventory management by stating on one hand when to place orders, and on the other hand what quantity to order. These decision rules are defined by a small set of parameters, usually two or three, which indicate different cases of decision making and their corresponding choice. An ICP usually provides the following alternatives for the two aforementioned decisions.

- For the "when" question, the decision maker may order periodically with a time span given by the ICP, or review the inventory continuously and place an order when needed, which is also stated by the ICP.
- For the "how much" question, the decision maker may order a fixed quantity given by the ICP, or order a variable quantity.

Combining both of these alternatives, and the values of the ICP's parameters, leads to the complete expression of an ICP. Surveying the literature, we can observe that there exists a limited set of ICPs that are recurring and widely studied. We also note that, though these policies are frequently used in articles, they often appear under different names or notations. An overview of some ICPs used in the literature can be found in Ma et al. [2019]. This survey focuses however only on stochastic inventory control. Jackson et al. [2020] also propose a review relating to ICPs, but they focus on the methods used to find the values of their parameters, and only include a subset of all the recurring ICPs we can find in the literature.

To the best of our knowledge, there is no academic material that provides a comprehensive list of the most used ICPs in the literature. In this Section, we propose a survey of the literature dealing with ICPs, and a description of ten ICPs we identified as being recurring in this research field, and classify them into categories. Moreover, we notice that there is no consensus on the notations to describe these policies. Consequently, we also propose a unified notation process to better describe and identify ICPs.

1.4.1 Basic Policies

In this Section, we present the simple policies that we can find in the literature. These policies are basic in the sense that the ordering strategy they propose relies on a reduced set of decision rules. Despite this simplicity, they are still considered relevant in various contexts and each of them has been widely studied in the literature.

Economic Order Quantity

As mentioned before, the Economic Order Quantity is the first ICP introduced in the literature. This policy was first presented in Harris [1913], and it is intended to be used in a production context with a deterministic and constant demand. Given this context, the paper aims at finding the optimal quantity to produce and the optimal time span between two production orders given the set of costs involved in the process. The findings of this paper have been adapted to inventory management contexts, leading to the emergence of extensive research around the Economic Order quantity, its applications in the field and its extensions to address different contexts and constraints. The strategy is simple. Every **T** periods – *i.e.* days, weeks, *etc.* – we place an order of **Q** items. We provide a representation of the inventory level evolution using this ICP on Figure 1.2.

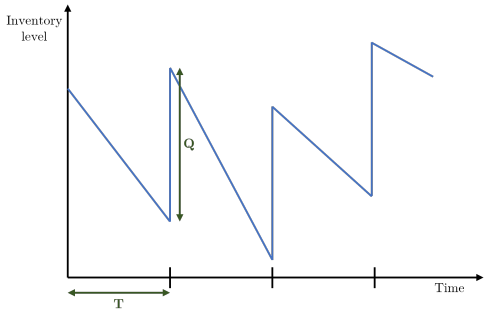


Figure 1.2: Schematic representation of the inventory level evolution of an item using the Economic Order Quantity ICP.

This ICP has been widely studied and adapted to a multitude of contexts since its introduction. According to Snyder and Shen [2019], this policy is relevant in case of deterministic demand, fixed ordering cost and lost sales. For the centenary of this paper, Andriolo et al. [2014] proposed a survey of the extensive literature dealing with the Economic Order Quantity. This study identifies the major fields in which the ICP is used, including among others imperfect quality items (Hayek and Salameh [2001], Jamal et al. [2004], Jaber et al. [2008], Khan et al. [2010]), permitted delays in payment (Teng [2002], Chung et al. [2005], Mahata [2011]), and time-varying demand (Wu et al. [2000], Teng et al. [2005], Sana [2008]). The policy has also been proven to be optimal for single-echelon inventory systems with deterministic demand (Kleinau and Thonemann [2004]).

Order-Up-To

The order-up-to policy appeared later in the literature. It has been introduced in Naddor [1956], as a periodic variant of the well-known (s, S) policy, that we detail later in Section 1.4.2. The strategy is the following. Let $\mathbf{S}$ be a fixed value representing the recompletion level. The inventory is reviewed periodically with a time span of $\mathbf{T}$ periods, in other words the decision maker checks the inventory every $\mathbf{T}$ periods to make a choice on the order to place. At the review time period, we place an order to bring the inventory level up to this recompletion level. The order value is calculated in the following manner. Let t be a time period where we have to place an order, and I_t the inventory level at this period. Then, the amount of the order q is computed as follows.

$$q = \begin{cases} 0 & if\ I_t \geqslant \mathbf{S} \\ \mathbf{S} - I_t & otherwise \end{cases} \tag{1.1}$$

In the literature, we can find this policy under the names *Order-up-To*-policy, (S, T) or (R, S). In the two last notations, S is the recompletion level, and the number of periods between two reviews is R in the first one, and T in the second. A schematic representation of the inventory level evolution of an item using the Order-Up-To policy is provided in Figure 1.3.

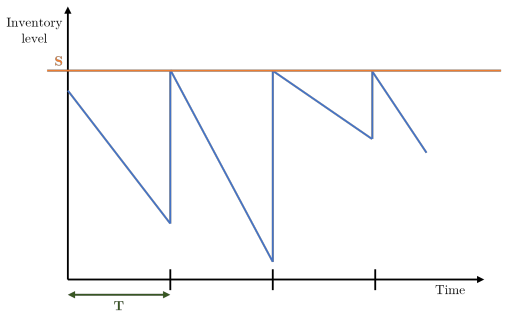


Figure 1.3: Schematic representation of the inventory level evolution of an item using the Order-up-to ICP.

The Order-Up-To policy is proven optimal for inventory systems with stochastic stationary demand, no fixed cost and no lead time (Snyder and Shen [2019]). It is also optimal for single-echelon, single-item inventory systems with stochastic demand and lost sales, when the lost sale penalty is significantly higher than the holding cost (Huh et al. [2009]).

It has been studied in various contexts, and especially for systems with perishable items (Haijema et al. [2007]) or highly perishable items (Duan and Liao [2013]). It is also proven to be optimal in systems with perishable items under several assumptions (van Zyl [1963], Nguyen and Chen [2020]).

In other fields, the policy has also been used for capacitated systems (Bertazzi et al. [2002], Veeraraghavan and Scheller-Wolf [2008], Taleizadeh et al. [2010]), capacity planning (Cheng et al. [2004]), or systems with order crossovers (Bischak et al. [2014], Riezebos and Gaalman [2009]).

Reorder Point

The reorder point policy has been introduced in Galliher et al. [1959], initially to deal with an inventory system with random demand. The ordering strategy of this policy is a bit different than the two first policies we describe, because there is no fixed time span between the orders. With this policy, the inventory system has to be reviewed continuously. Once the inventory level drops under a given value **r**, which is the reorder point, an order of size **Q** is placed. Usually, the value of **Q** is computed like in a classic EOQ policy.

In the literature, this policy is referred to as the (r, Q), or the (R, Q) policy. In both notations, Q is the quantity to be ordered, and r and R represent the value of the reorder point. We provide a schematic representation of the inventory level evolution of an item using the reorder point policy in Figure 1.4.

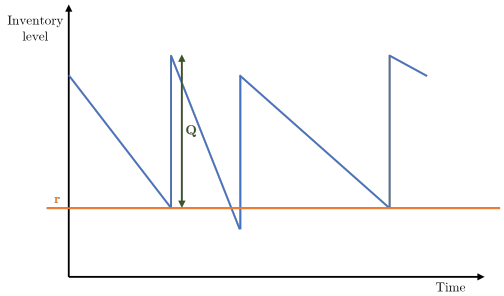


Figure 1.4: Schematic representation of the inventory level evolution of an item using the Reorder Point ICP.

This policy is known to be optimal for single-echelon inventory systems with stochastic demand. It is also proven to be optimal for an inventory system with stochastic demand, deterministic lead time, back-order and fixed cost (Snyder and Shen [2019]). In the literature, this policy is generally used in inventory systems with lead times (Taleizadeh et al. [2010], Chiang [2010], Wang [2011], Taleizadeh et al. [2013], De Smet et al. [2019]). It has also been used in contexts with perishable items (Berk and Gürler [2008], Kouki et al. [2015]), and in contexts with capacitated constraints (Zhao et al. [2012], Tamjidzad and Mirmohammadi [2015], Tamjidzad and Mirmohammadi [2017]).

1.4.2 Hybrid Policies

The ICPs we detail in this Section are more elaborate than the basic policies described above. They are built by combining several simple policies, or by adding a parameter to one of them. As a result, the set of decision rules provided by hybrid ICPs is less simple. However, this allows for addressing new contexts, such as customer waiting time, or more complex problems, such as multi-echelon inventory management or inventory systems with transshipment for instance.

Periodic Reorder Point

The Periodic Reorder Point policy has been introduced and studied first in Tempelmeier [1985], in order to develop a policy minimizing the customer waiting time in an inventory system with back-order. This ICP is a periodic extension of the Reorder Point Policy we presented above. The strategy is the following. The inventory is reviewed every **T** periods. If the inventory level is under a given value **r** at the review time period, an order of size **Q** is placed. Otherwise, no order is placed until the next inventory review time period.

In the literature, the ICP can be found with the notation (R, s, Q). In this notation form, R is the number of periods between two inventory reviews, s is the value of the reorder point, and Q is the size of the orders. A schematic representation of the inventory level evolution of an item using the Periodic Reorder Point ICP is provided in Figure 1.5.

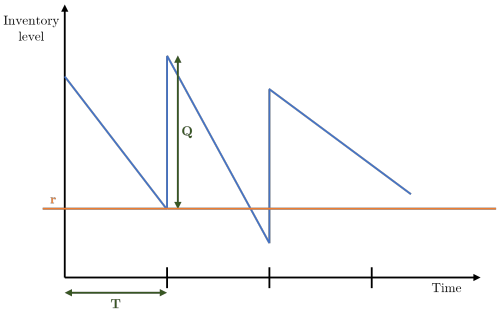


Figure 1.5: Schematic representation of the inventory level evolution of an item using the Periodic Reorder Point ICP.

After its introduction, this policy has mainly been used in the same framework it was developed for, namely inventory systems where the decision makers want to minimize the customer waiting time (Kiesmüller and de Kok [2006], Tempelmeier and Fischer [2010]). It has also been studied in various other contexts, like capacity constrained systems (Wang et al. [2011], Tempelmeier and Bantel [2015]), inventory management in medical contexts (Bijvank and Vis [2012b], Wilson et al. [2015]), systems with a minimum order quantity constraint (Kiesmüller et al. [2011]), multi-echelon systems (Tempelmeier [2013]), continuous demand (Johansen and Hill [2000]), and inventory systems with perishable items (Kouki and Jouini [2015]).

Reorder Point Order-Up-To

This policy is the second that has been introduced in the literature, and like the Economic Order Quantity ICP, it has been widely studied with diverse settings and assumptions. The seminal paper presenting this policy is Arrow et al. [1951], and aims at deriving an optimal ICP for a simple inventory system with different assumptions on the demand form. Their main concern is to provide a policy adapted for contexts with stochastic demand. The authors also provide a formula to compute the best values for the parameters of the policy using the demand distribution and the characteristics of the system. In this policy, the inventory level is reviewed continuously and the ordering decisions are determined by two parameters: the reorder point **r** and the recompletion level **S**. As long as the inventory level is higher than the reorder point level, no order is placed. Once the inventory level drops under the reorder point, an order is placed to bring the inventory

level up to the recompletion point. Let t be a time period where the inventory level $I_t \leqslant \mathbf{r}$. The, the order size $q = \mathbf{S} - I_t$.

In the literature, this policy is almost always referred to as the (s, S)-policy, where s represents the value of the reorder point, and S the value of the recompletion level. It can be viewed as a crossover policy between the Reorder Point Policy and the Order-Up-To policy, that we both described in Section 1.4.1. We provide a schematic representation of the inventory level evolution of an item using the Reorder Point Order-Up-To ICP in Figure 1.6.

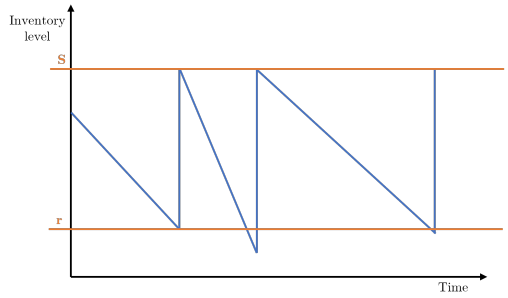


Figure 1.6: Schematic representation of the inventory level evolution of an item using the Reorder Point Order-Up-To ICP.

There is a large part of the literature using the Reorder Point Order-Up-To policy with different systems, and especially in case of stochastic demand and setup costs. It is known to be optimal for inventory systems with stochastic stationary demand, fixed ordering costs and null lead times (Snyder and Shen [2019]). In recent years, this policy has also been used for various stochastic inventory model with pricing decisions. In cases of joint inventory and pricing decisions, there exists a lot of studies that prove its optimality for different assumptions (among others Chen and Simchi-Levi [2004], Pang [2011], Chen and Sun [2012]). Among the list of fields in which this policy has been applied and studied, we can cite inventory systems with transshipment (Kukreja and Schmidt [2005], Ekren and Arslan [2020]), spare parts inventory management (Ilgin and Tunali [2007], van Jaarsveld et al. [2015], Guajardo et al. [2015]), and capacity constraints (Yang et al. [2014], and Kong et al. [2020], that prove its optimality for capacity constrained inventory systems with fixed ordering costs).

Periodic Reorder Point Order-Up-To

This policy is the periodic extension of the above described Reorder Point Order-Up-To ICP. It has been introduced by Scarf [1959], to deal with inventory systems stochastic stationary demand and more generic functions for expressing the costs. With this strategy, the inventory is reviewed periodically. Every $\mathbf{T}$ periods, the decision maker checks if the inventory level is above a given value $\mathbf{r}$, which is the reorder point. If so, no order is placed until the next review time period. Otherwise, an order is placed to bring the inventory level up to its recompletion level $\mathbf{S} > \mathbf{r}$. The decision rule set can be summarized as follows. Let t be an inventory review time period, and I_t the inventory level at this time. Then, the value of the order placed q is computed as follows.

$$q = \begin{cases} 0 & if\ I_t \geq \mathbf{r} \\ \mathbf{S} - I_t & otherwise \end{cases} \tag{1.2}$$

In the literature, this policy can be found with the notations (R, s, S) and (s, S, T). In both notations, s represents the value of the reorder point, and S the value of the recompletion level. The number of periods between two inventory reviews is indicated by T in the first notation, and by R in the second. We provide a schematic representation of the inventory level evolution of an item using the Periodic Reorder Point Order-Up-To ICP in Figure 1.7.

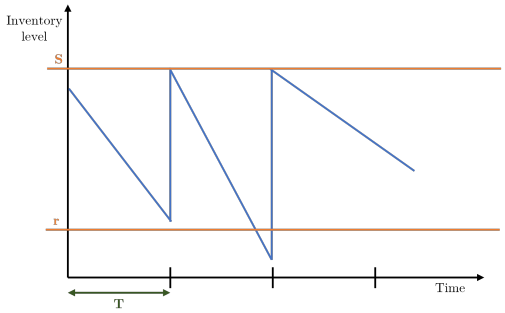


Figure 1.7: Schematic representation of the inventory level evolution of an item using the Periodic Reorder Point Order-Up-To ICP.

In the seminal paper of Scarf [1959], the policy is proven to be optimal for inventory systems with stochastic stationary demand, piecewise linear purchase cost, and convex holding lost sale costs. This result has been extended to N-echelon serial inventory systems with the same assumptions in Clark and Scarf [1960]. It has also been proven optimal for single-echelon single-period systems with stochastic stationary demand, no lead time and no supply uncertainty in Rabta and Aïssani [2005]. More recently, it has been used for inventory systems with lost sales (Esmaili et al. [2019], Bijvank and Vis [2012a]), and continuous demand (Moors and Strijbosch [2002], Gutierrez and Rivera [2021]). We can also find a significant amount of literature applying this policy in various contexts such as transshipment (Tlili et al. [2012]), multi-echelon systems (Göcken and Dosdogru [2015]), spare parts inventory management (Rodrigues and Yoneyama [2020]) or capacity constrained systems (Miranda and Cabrera [2010]).

1.4.3 Policies with Multiple Quantities

In this Section, we present two policies that have a slightly different ordering mechanism. They still provide ordering decision rules based on a reduced set of parameters, but they introduce a new type of parameter.

Reorder Point with Multiple Quantities

The idea to order a multiple of a base batch-size is first presented in Morse [1959], and it has been translated to the Reorder Point Policy with Multiple Quantities in Hadley and Whitin [1962]. The strategy is close to the Reorder Point policy, but the order value is computed with a different method. The ICP strategy is the following. Like in the reorder point strategy, the inventory is reviewed continuously. Once the inventory level drops under the reorder point $\mathbf{r}$, we place an order of size $n \times \mathbf{Q}$. n is the smallest integer such that the quantity ordered brings the inventory level above the reorder point $\mathbf{r}$.

In the literature, this policy can be found with the notation (s, nQ) or (R, nQ). In both notations, nQ represents the quantity to order. The reorder point is noted s in the first and R in the second.

This policy has mainly been studied in multi-echelon contexts (Kiesmüller et al. [2004], Li and Sridharan [2008], Shang et al. [2009], Li and Chen [2012], Berling and Marklund [2014], Berling and Farvid [2014], Xiang and Rossetti [2014]. We can also find studies that apply this policy in different contexts such as inventory systems with perishable items (Molana et al. [2012], dual sourcing (Zhou and Yang [2016]) and joint replenishment planning (Feng et al. [2011]).

Periodic Reorder Point with Multiple Quantities

This policy is the periodic version of the one we presented previously. It has also been introduced by Hadley and Whitin [1962], in the same paper. This policy is close to the Periodic Reorder Point ICP, but like its continuous review version, the order quantity is computed differently. It is intended to be used in the same situation. The strategy is the following. The inventory level is reviewed periodically, and every $\mathbf{T}$ periods, we check if the inventory level is under a value $\mathbf{r}$. If not, no order is placed until the next review time period. Otherwise, an order of size $n \times \mathbf{Q}$ is placed, with n being the smallest integer so that the inventory level is brought above the reorder point value $\mathbf{r}$.

In the literature, we can find this policy under the following notations: (R, s, nQ) or (r, nQ, T). In both notations, nQ refers to the ordering quantity. The reorder point is noted s in the first notation and r in the second, whereas the number of periods between two inventory reviews is noted R in the first notation, and T in the second.

As stated before, this ICP has been introduced in the same paper than the Reorder Point with Multiple Quantities, and with the same motive. This policy has been used also been used in various contexts, such as inventory systems with lead times (Larsen and Kiesmüller [2007], Lagodimos et al. [2012], Christou et al. [2020], inventory management with perishable items (Broekmeulen and van Donselaar [2009], van Donselaar and Broekmeulen [2012]). We can also find this ICP used in multi-echelon contexts (Shang et al. [2009]), in inventory systems with constraints on the order quantity (Zhu et al. [2015]) and in systems with packaging constraints (Lee [2009]).

1.4.4 Other Policies

The two remaining ICPs are policies that we could not classify in other categories, but that we still found in a significant portion of the literature.

Base-Stock

This policy is a specific case of the Reorder Point Order-Up-To ICP. It has been introduced in Feeney and Sherbrooke [1966], that studies the behavior of the policy with stochastic demand. The strategy is the same than for the Reorder Point Order-Up-To policy. Inventory is reviewed continuously, and once the inventory level drops under the reorder point, an order is placed to bring it up to the recompletion level. In this case, if $\mathbf{S}$ is the recompletion level, the reorder point $\mathbf{r} = \mathbf{S} - 1$.

In the literature, this policy is mostly referred to as the $(S-1, S)$-policy. In the early years after its introduction, this policy was mainly used for repairable and recoverable items (Sherbrooke [1968], Graves [1985]). More recently, this policy has been used in more diverse contexts. It has mostly been used for inventory systems with high-price low-volume items (Grahovac and Chakravarty [2001], Garg et al. [2012], Varas et al. [2019]), spare-parts replenishment contexts (Aronis et al.

[2004], Cestana et al. [2019]), and perishable items inventory management (Olsson [2010], Alizadeh et al. [2014], Mahmoodi et al. [2015]). It has also been studied for items with time-varying demand (Pinçe and Dekker [2011]) and inventory systems with lead time (Lee [2009]).

Can-Order

The can-order policy is a policy that has been defined only for joint replenishment contexts, which means we are computing the orders for a group of items simultaneously. It has been specifically designed for such contexts. This ICP has been introduced in Balintfy [1964]. The strategy is the following. Every item p has its own reorder point $\mathbf{r}_p$ and its own recompletion level $\mathbf{S}_p$. They also have another level, the can-order level $\mathbf{c}_p$. Every time the inventory level of an item drops under its reorder point, an order is placed to bring the inventory up to its recompletion level. For every other item that have an inventory level lower than their can-order level, we add an order for these items to bring their inventory level up to their recompletion level. This leads to an adaptation of the Reorder Point Order-Up-To policy, but for joint replenishment contexts. In the literature, this policy is noted (s, c, S).

1.4.5 Unified Notation

As we mentioned in the previous Sections, some policies have different notations in the literature. Moreover, even if all the policies we described are distinct, we can note that they are all based on a small set of parameters, that are recurrent. These parameters, although they refer to the same concept (reorder point value, number of periods between two orders, *etc.*), are referred to with different notations and in different orders.

In order to describe the ICPs in a more consistent way, and to clarify the notions associated to each parameter, we propose a unified notation process.

Based on the previous description of inventory control policies, there are four parameters that can be used to describe an ICP.

The first parameter is the quantity to order, in case this one is predefined. In all the policies, this parameter is noted Q, or nQ in case of multiple quantities. In order to define a unique notation for this manuscript, we note the quantity $\mathbf{Q}$.

The second parameter that have a solid notation throughout the literature is the order-up-to, or recompletion, level. This represent the inventory level we should have after an order. This parameter is noted S. For the same reason as $\mathbf{Q}$, we note this parameter $\mathbf{S}$.

The first two parameters were intended to define the quantity of items to be ordered. The next two parameters concern the ordering timing.

If the inventory is reviewed continuously, the policies define a reorder point value, that will trigger the order. In the literature, this reorder point is either noted r or s. In order to clearly distinguish the notations, and to not avoid confusion between the reorder point and the Order-Up-To level, we select the notation $\mathbf{r}$.

If the inventory is reviewed periodically, we have to define a number of periods between two orders. In the articles, this parameter is either noted R or T. For the same reasons that motivate the choice of r, we select the notation $\mathbf{T}$.

The final notation is the following: (w, x, y, z), with each component defined as follows.

$$w = \begin{cases} \mathbf{T} & \textit{if the inventory is reviewed periodically} \\ \emptyset & \textit{otherwise} \end{cases} \tag{1.3}$$

$$x = \begin{cases} \mathbf{r} & \textit{if there is a reorder point} \\ \mathbf{r}, \mathbf{c} & \textit{if there is a can order point} \\ \emptyset & \textit{otherwise} \end{cases} \tag{1.4}$$

$$y = \begin{cases} \mathbf{S} & \textit{if there is a recompletion level} \\ \emptyset & \textit{otherwise} \end{cases} \tag{1.5}$$

$$z = \begin{cases} \mathbf{Q} & \textit{if there is a predefined order quantity} \\ n\mathbf{Q} & \textit{if the order quantity is multiple} \\ \emptyset & \textit{otherwise} \end{cases} \tag{1.6}$$

We recapitulate all the policies, their classification and their final notation in Table 1.1, using the process we described above.

Table 1.1: Final notation and classification of Inventory Control Policies.

Inventory Control Policy	Notation
Classic policies	
Economic Order Quantity	$(\mathbf{T}, \emptyset, \emptyset, \mathbf{Q})$
Order-Up-To	$(\mathbf{T}, \emptyset, \mathbf{S}, \emptyset)$
Reorder Point	$(\emptyset, \mathbf{r}, \emptyset, \mathbf{Q})$
Hybrid policies	
Periodic Reorder Point	$(\mathbf{T}, \mathbf{r}, \emptyset, \mathbf{Q})$
Reorder Point Order-Up-To	$(\emptyset, \mathbf{r}, \mathbf{S}, \emptyset)$
Periodic Reorder Point Order-Up-To	$(\mathbf{T}, \mathbf{r}, \mathbf{S}, \emptyset)$
Policies with multiple quantities	
Reorder Point with Multiple Quantities	$(\emptyset, \mathbf{S}, \emptyset, n\mathbf{Q})$
Periodic Reorder Point with Multiple Quantities	$(\mathbf{T}, \mathbf{r}, \emptyset, n\mathbf{Q})$
Special cases	
Base Stock	$(\emptyset, \mathbf{S}-1, \mathbf{S}, \emptyset)$
Can Order	$(\emptyset, \mathbf{r}, \mathbf{c}, \mathbf{S}, \emptyset)$

1.5 Multi-Echelon Approach for Inventory Control

Although the research on inventory control traditionally focuses single-echelon approaches, in other words by computing the orders for one item, or for a group of items within a single entity, this is not the only approach for inventory control optimization. In fact, entities from a same supply chain are connected to each other. Moreover, some of them depend on each other, when a warehouse supplies a store for example, and some entities of a network can also share common constraints such as suppliers' requirement on the total quantity delivered for example.

All of this cannot be explicitly treated with a single-echelon model, as this approach considers every entity of the system separately. However, although the decisions made for every entity are mainly local, they still have an impact on the other entities of the system (Axsäter [2001]). For example, if a warehouse supplies a set of stores facing customer demand, the inventory level of the warehouse is critical. In addition to generating its own costs, if the inventory level of the warehouse is too low to answer all of the orders, this will create customer dissatisfaction at the store level.

Moreover, it has been shown that, except when we take into account organisational or managerial issues, if all the relevant information of the supply-chain network is centralised, multi-echelon strategies outperform single-echelon ones in terms of costs and efficiency (Hausman and Erkip [1994]). If we stick to inventory management using ICPs, there are studies proving the domination of multi-echelon adapted ICPs over single-echelon ones in different contexts (Axsäter and Zhang [1999]).

As a result, multi-echelon inventory management seems to be a promising research area for dealing with modern supply chain networks. As we already mentioned, a large part of the literature dealing with inventory control is focused on single-echelon inventory management, and especially using ICPs. However, a broad research area continues to focus on multi-echelon inventory optimization. De Kok et al. [2018] propose a literature review and classification of multi-echelon replenishment studies with stochastic demand. They also identify future research streams based of the gaps in the literature they unveil. They highlight the need for further research on more general structures, relaxing the usual restrictive assumptions that are made in the studies, *e.g.* known demand uncertainty (Grahl et al. [2016], Patriarca et al. [2016], Stenius et al. [2016]), or even known demand (Glock [2011], Sajadieh and Thorstenson [2014]), or infinite storage capacity (Moncayo-Martínez et al. [2016], Zhu [2022], Topan and van der Heijden [2020]). Moreover, there is a need for exploring more diverse constraints, such as storage capacity (Dai et al. [2021]) or capacity constrained transportation (Pazhani et al. [2016]), demand with forecast error (Boulaksil [2016]), lost sales customer behavior (Chao et al. [2021]), operational flexibility in the inventory system network (Drent and Arts [2021]). Such assumptions lead to more complex models, and that makes it more difficult to analyze them or derive mathematical properties. Nevertheless, results in models that better reflect industrial contexts, making them more applicable in real-world settings. Finally, heuristic methods and simulation are identified as promising venues to explore such contexts and propose more tractable solutions to inventory control problem.

In fact, looking more closely at the references selected in this literature review, most of them are analytic studies. 67% of the articles are aimed at finding optimal policies, optimal parameter values, or derive mathematical properties on diverse contexts. In order to be able to conduct a mathematical analysis, the authors often make very restrictive assumptions, or work with small inventory systems. This leads models that are hardly applicable in practice.

Moreover, if we concentrate on articles that do not propose an analytic study, there is still 27% of them that are aimed at finding exact methods to solve the inventory replenishment problems

they tackle. Their solving procedures mainly use dynamic programming, or classic mathematical programming methods (linear programming, non-linear programming or stochastic programming).

Svoboda et al. [2021] propose a literature review on inventory control with multiple supplying sources and supply network flexibility. One of the main research avenues they identify is to develop models and solutions tackling problems with more complex networks, and especially with multi-echelon network. This echoes the conclusions provided in De Kok et al. [2018] on the need to develop multi-echelon models tackling operational flexibility. The literature review proposed in Svoboda et al. [2021] is not focused on multi-echelon inventory control, but there still is a non negligible list of references that propose multi-echelon models. In the same way as in De Kok et al. [2018], most of the multi-echelon references cited in this paper are analytic, and aim at finding optimal policies or at deriving optimal properties, and most of the remaining references use exact methods such as linear programming. Nevertheless, there are some articles that use simulation as a comparison tool to show the added value of introducing flexibility in the supply networks (Arıkan et al. [2014], Schmitt et al. [2017]), or as an optimization tool (Wu et al. [2019], van Wingerden et al. [2019]).

To summarize all of the research streams identified for further research: there is a need for studies that tackle supply networks that are closer to the reality. This can be achieved by considering larger networks, with more echelons or more entities, or by including constraints that represent the actual inventory systems in practice, like multiple sourcing or capacity constraints for example, or by relaxing usual modeling assumption, like completely known demand uncertainty or full back-order for example.

Considering one or some of the aforementioned research direction leads to more complete models, and consequently also more complex models, with a lot of decision variables and more intricate cost functions. As a result, mathematical properties and analytic studies are more difficult to conduct in such contexts, and one has to find other techniques to find solutions for such inventory control problems.

1.6 Existing Replenishment Optimization Methods at Vekia

As mentioned in Section 1.2.2, the partner company Vekia already has algorithms to manage inventory control problems and optimize orders. There are two order engines, that are developed to tackle given replenishment contexts, and that are in use for the customers. In the following sections, we describe their scope and functioning.

1.6.1 Single-Echelon, Single-Item, Single-Period Inventory Control

If we decompose the global replenishment problem of a complete supply chain, the most simple problem is to compute orders for each entity, each item, and each planning period. This leads to a single-echelon, single-item and single-period inventory control problem. The *SPSE* (*Single-Product Single-Echelon*) order engine aims at solving such problems. We provide a schematic representation of the decision process within a supply chain using *SPSE* in Figure 1.8.

For a given entity, item and planning period length, the order engine takes as input a piecewise convex cost function, a set of predicted demand samples, and a set of inventory control constraints (*e.g.* minimum inventory level), it computes the optimal order to place in order to minimize the given cost function on the whole planning period. The optimization method is based on a dichotomic search process, which makes it very efficient.

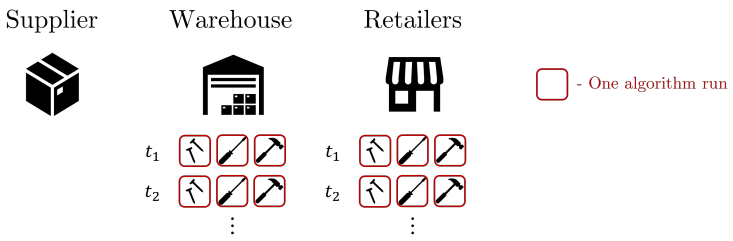


Figure 1.8: Schematic representation of decision process using *SPSE*. In this case, every ordering decision is made by one separate run of the algorithm.

One big advantage of this order engine is that, given its efficiency and the simplicity of the problem, the computation is very quick for each replenishment sub-problem. Moreover, every sub-problem is independent from one another. Therefore, we can parallelize the execution of the engine on all sub-problems and significantly reduce the total execution time for the whole supply chain. Furthermore, the algorithm is very flexible, and we can add various single-item constraints and cost elements to the model, as long as the cost function stays piecewise convex.

However, this approach is rather myopic, as it separates the whole problem in several independent sub-problems that have a reduced scope. This makes the model unable to capture the dependencies between each sub-problem. For example, in the supply network described in Figure 1.8, both stores order items from the warehouse. For a given item and time period, the sum of both orders computed for the stores may exceed the inventory level of the warehouse. In such case, we will have to add an allocation procedure that will decide how much to send to each store while respecting the inventory level of the warehouse. Then, the store will not get its optimal order, and this can incur extra cost. The nature of the algorithm makes it unable to tackle such problems, even if we have the data to prevent it, which is a significant limitation.

1.6.2 Single-Echelon, Multi-item, Multi-period Inventory Control

As we mentioned in the previous Section, the single-echelon, single-item and single-period order engine, although it is very efficient, is limited in terms of constraints and costs, as it can only treat single-item specifications. In order to tackle some of these limitations, there is a second order engine that is intended to solve single-echelon, multi-item and multi-period problems. MP^2 (*Multi-Product Multi-Period*) is an extension of *SPSE*, and computes ordering plans for groups of items that are provided by the same source within a given entity.

The optimization process relies on splitting the total planning horizon into ordering periods, that are called coverage periods, and the orders to place are represented as objects we call blocks. A block represents an order that has to answer the customer demand for a given number of coverage period. Blocks are defined by their length – the number of periods they have to cover, the quantity of the item to order, and the single-item costs associated to the order. These are computed using *SPSE* on the horizon of the number of coverage periods of the block. All the possible blocks for the planning horizon are pre-computed for all items using *SPSE*, and they are the input for a Mixed-Integer Linear program that will compute the best block combination on the planning horizon. We provide a schematic representation of the decision process within a supply chain using MP^2 in Figure 1.9.

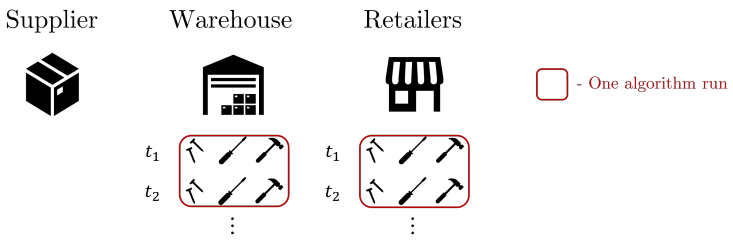


Figure 1.9: Schematic representation of the decision process using MP^2. In this case, we have to run the algorithm for each entity separately, but one algorithm produces ordering decisions for all items and periods simultaneously.

This approach has the main advantage of integrating multi-item costs and constraints while still respecting the single-item constraints ensured by *SPSE*. This is an extension of the single-item single-period and can tackle constraints that we usually encounter such as fixed ordering cost or order quantity sale.

Its main drawback is that the whole process is longer, especially the block computation phase. Moreover, this still does not answer the problem of warehouse inventory allocation, because this algorithm still computes orders for entities separately.

1.7 Conclusion: Motivations of our Work and Positioning

As it is described in Section 1.5, although the research on multi-echelon replenishment planning exists, it is mainly focused on analytic studies, and on the establishment of formulae for computing optimal parameters or solutions, or of mathematical properties or theorems for given replenishment problems. As a result, the inventory control problems that are studied remain simple, as they apply only to small supply chain and do not include many constraints, so that the problem remains mathematically tractable. This kind of study is very important to build a better understanding of the problems we tackle, uncover their specificities and identify where lies their complexity. They also provide results that are mathematically proved, and that consequently can be applied in the contexts they tackle without questioning their efficiency or relevancy.

However, this kind of study presents several downsides. First, they usually present a large number of reductive assumptions such as infinite capacity of the entities, or fully known demand variability. Moreover, they usually take only a few constraints into account, or even no constraint at all. These studies also generally focused on small supply networks, either with a small number of entities, a small number of flows, or a simple network structure. As a result, the cases that are studied are not representative of inventory control problems we usually encounter in practice. Therefore, although the results presented by these studies are solid, we can rarely apply them as is, as the problems we tackle in real-life applications are more complex. Finally, finding optimal solutions and proving their optimality is an interesting approach, because it provides the minimal cost. Nevertheless, an optimal solution only applies on a given context, and most of the time this context does not take into account the embedded variability of replenishment problem. Indeed, the variability on the demand, delivery delays, quantities delivered by suppliers, *etc.* are common in inventory control. Consequently, an optimal solution can be relevant for a given deterministic context, but may be unsuitable once the real scenario occurs, as we may observe differences with what was planned. As a result, in real-world contexts, it may be more interesting to rapidly find

sub-optimal solutions that are satisfying, although their cost may be higher, than spend time and resources into computing an optimal solution that may not cover the actual replenishment events that will occur. Therefore, we identify a need for a study that tackles problems that are more representative of real-life inventory control problems, by including industrial constraints, tackling larger supply networks, or by incorporating variability in the planned events. This may result in a mathematically intractable model, but finding a solution to such problems enables their application to practical problems, although this solution is not optimal.

On the industrial side, as we mention in Section 1.6, there already exist optimization methods to tackle replenishment problems in the company. These methods are designed with the aim of being applicable to a large range of replenishment problems and contexts. They may not return an optimal, or proven to be optimal, solution, but they provide good quality results in a reasonable time. They are also designed to be adaptable, and can include a relatively significant number of constraints if needed. This specificity is important in an industrial context where we provide software to different customers, because each of their replenishment context is unique and has its own set of costs and constraints.

Yet, both of the optimization methods already in place in the company use a single-echelon approach. This means that, even if the customer provides information on their entire supply chain, the entities will be treated separately in the order computation system. As we already mentioned earlier, the single-echelon echelon approach is interesting because it separates the complete problem and makes its complexity lower. Nevertheless, it hides the dependency between the entities and the constraints they may share. For example, let us consider a supply network where a warehouse supplies a given number of stores, that have to fulfill external demand for one given product. In this case, with the single-echelon approach, an optimization method will compute the orders originating from the stores directed at the warehouse for every store separately. Then, the warehouse will have to fulfill these orders. Although we are in a setting where we have all the information on the supply network, as we operate in a single-echelon approach, the stores do not have information on the available inventory at the warehouse, or on the orders originated from the other stores. As a result, the total number of items ordered by all stores may exceed the number of items available in the warehouse inventory. In such cases, we will have to set an inventory allocation procedure so that the total number of items sent to the stores stay within the available inventory levels, which means that the stores may not receive the item quantity they ordered, and this can lead to unplanned demand unfulfillment, and consequently to higher costs. This is one of many examples of the aspects that are impossible to handle with a single-echelon approach, and this is a serious limitation of the current optimization methods in place in the company. We provide a representation of the decision process using a multi-echelon order engine in Figure 1.10.

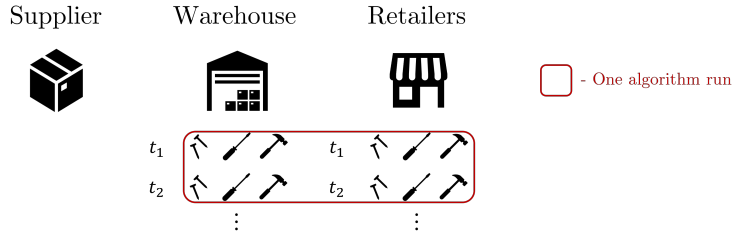


Figure 1.10: Representation of the decision process using a multi-echelon order-engine. In this setting, the ordering decisions for all entities, items and periods of the instance are taken simultaneously.

As a consequence of all the elements presented above, we can note that there is a need, both industrially and academically, for a study on inventory control problems that is aimed at treating real-life, or at least realistic, problems. In fact, this gap is made clear by the literature (De Kok et al. [2018], Svoboda et al. [2021]), as reviews state that there is still a need for research studies that either tackle larger supply chains, relax common reductive assumptions, or include constraints that are encountered in practice. On the other hand, the partner company still needs an appropriate optimization method to treat multi-echelon inventory control problems, and cannot use already existing methods, as the ones provided by the literature are hardly applicable to the customer cases.

Overall, after surveying the existing literature and the needs of the company, we can conclude on the relevance of a research study on realistic multi-echelon inventory control problems. Accordingly, the aim of the work we present in this thesis is to model a replenishment problem using a multi-echelon approach including industrial specifics to the supply configuration under study, and propose an optimization method to solve it. In this thesis, we want to answer several objectives.

- Study the industrial contexts of the customers of the company in order to identify their needs, characteristics and constraints regarding the inventory control process. With this analysis, we want to determine the key aspects that the companies face in the replenishment process, so that we can build a multi-echelon model that can better answer their requirements. This part of the work establishes a solid foundation for the optimization part, and ensures we propose a solution that will be applicable in practice.

- Propose an order computation tool that can solve a multi-echelon problem given the volume of data we have to handle, and provide good results in a reasonable time. In this phase of the work, we aim at identifying a suitable optimization method for the type of problem we want to tackle, and adapt it to answer the requirements of the model.

- Compare the designed method with the ones already in place in the company. For this part of our work, we want to highlight the advantages of using a multi-echelon approach. To do so, we want to apply the solution method to a case study drawn from a real-world application. We also want to identify its benefits over a single-echelon approach, and therefore we want to compare the method we propose with the already existing single-echelon optimization methods in place in the company.

To conclude, this thesis focuses on several aspects. First, we aim at modeling a realistic multi-echelon inventory control problem. Then, we want to propose an optimization method that can compute good quality order decisions in a reasonable time for this problem. Finally, it is important to demonstrate the effectiveness of the approach by applying it to a realistic case-study and comparing its results with the existing replenishment decision methods in the company.

In the following chapter, we provide a detailed description of the replenishment problem we want to solve. The formulation of this problem is the result of a study of industrial replenishment contexts. We describe all of the specifications drawn from this analysis, provide an application case-study for the remaining experiments, and propose a mathematical model for the specified problem.

Chapter 2

Model Formulation and Exact Resolution

Contents

2.1 Introduction 26
2.2 Problem Definition 27
2.2.1 Problem Specifications 27
2.2.2 Specific Constraints and Modeling Choices 28
2.3 Proposed Mathematical Model 31
2.3.1 Linear Programming 31
2.3.2 Data and Notations 32
2.3.3 Model 34
2.4 Input Data and Problem Instances 36
2.4.1 Additional Elements for Data Completion 37
2.4.2 Instance Construction 37
2.5 Experimentations 38
2.5.1 Experimental Protocol 39
2.5.2 Results and Discussion 40
2.6 Conclusion 41

2.1 Introduction

The main goal of the research work conducted in this thesis is to model and solve a multi-echelon inventory replenishment problem. One of the main highlights of the state of the art presented in the previous Chapter is that, although there exists research focusing on multi-echelon inventory control, there is still a lack of work taking into account realistic constraints and use-cases (De Kok et al. [2018]).

In this regard, we want to develop a model that can describe a multi-echelon replenishment problem, but that can also take into account relevant industrial constraints. Accordingly, we choose to take an existing inventory control context as a case-study for our work.

As stated in Chapter 1, research on multi-echelon systems facing stochastic demand exist, and there are plenty of studies developing exact or heuristic methods for inventory control of such systems. However, the methods provided often turn to be intractable for real-life problems due to high computation time or restrictive assumptions (Berling and Marklund [2014]).

Accordingly, using real-world problems and data as a case study for developing efficient solving method seems promising. In the literature, we can find two types of approaches for using real-life problems as case studies. Firstly, the case study is used as a validating example of a proposed method for a given research question. Such work can be found in Klosterhalfen et al. [2014], Basten et al. [2015], Ni and Shu [2015], Berling and Martínez-de Albéniz [2016]. This approach is suitable for testing a theoretically developed method and provide more practical results. A second approach is to build a model and a solving method from a given case-study, like in Moncayo-Martínez et al. [2014] and Hua and Willems [2016]. This approach enables the development of methods that are directly suited for real-life instances, and ensures the resolution the real-life problems under study.

Nevertheless, both approaches do not really counter the limitations mentioned previously. In fact, the method is either tested on or constructed for a specific example of the problem considered, and this does not ensure the tractability of the method on other datasets.

As a result, we chose to base our research on a realistic case-study drawn from a real-world dataset. To do so, we use an inventory context given by one customer of the company, and use its data. This case-study is the foundation for our model and provides data for the experimentations. This ensures that our research is grounded in real-world applications. However, in order to avoid the previously mentioned limitations, we choose a customer context that is known to be representative of the multi-echelon inventory control problems faced by the company, and we also adapt it using additional constraints that we can encounter in practice. This approach ensures that we are answering a practical research problem, while keeping our model generic enough to be applied on several cases.

In this chapter, we focus on detailing the characteristics of the problem we chose to solve. We explain the characteristics and specifications drawn from the case-study, in order to carefully define the problem we intend to address. This is presented in Section 2.2. Based on this problem description, we can build a mathematical model that will be the basis for the design of a resolution method. We present and detail this model in Section 2.3. Then, we also describe the case-study's data we use and the solving instances we build in Section 2.4. We also use the proposed model for a first study on the exact resolution of the problem using the case-study data, that we adapted for our work. This study and its results are presented in Section 2.5. Finally, we provide a conclusion of this part of our work in Section 2.6.

2.2 Problem Definition

In this section, we detail all the characteristics and specifications of the case-study we set in place for our work. As mentioned in Section 2.1, this case-study is based on a real-life industrial context. This company provides spare parts and finished products for domestic heating systems. For the spare parts category, they also provide the maintenance service to install and repair heating systems at their customers place. Throughout the manuscript, we will refer to these spare parts and finished products as *products* or *items*.

The network under study consists of a *warehouse* and a set of *retailers*, also called *stores*. These are the entities for which we manage the inventory, also referred to as *active entities*. We also consider a set of external suppliers, that provide items for the active entities. The suppliers provide goods to the warehouse, and the warehouse redistributes these goods to the retailers. Retailers face an external demand from customers and have to meet this demand without knowing it in advance. This describes a classical One-Warehouse-N-Retailers (*1WNR*) network.

In the context under study, there is an additional specification regarding the flows of items within the network. As a matter of fact, retailers can also place orders directly to suppliers, although the warehouse is the privileged ordering source. This is set in place to complement orders or replace warehouse orders when the latter does not have enough inventory to supply retailers. This property of the network is called *dual-sourcing*.

A simple representation of the network under study is provided in Figure 2.1. The complete network we work with in this thesis contains 905 items, 164 stores and 43 suppliers. In the following, we detail the constraints linked to the problem under study. These constraints are incorporated to the proposed model as case-specific constraints, or as representative constraints for the generic dual-sourcing *1WNR* industrial contexts.

We first detail the general characteristics of the inventory control problem we propose to solve in Section 2.2.1. Then, in Section 2.2.2, we detail the industrial constraints related to the case under study, and the modeling choices they incur.

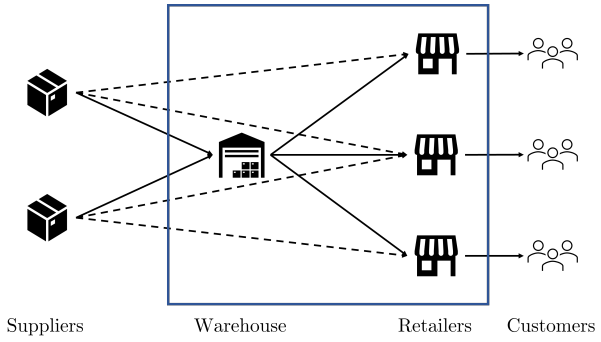


Figure 2.1: Representation of the case study supply network. Solid lines represent the classic 1WNR flows, dotted lines represent the dual-sourcing flows. Active entities are in the square.

2.2.1 Problem Specifications

Before getting to the case-study specifications and model definition, let us focus on the problem design. In this section, we detail the characteristics of the inventory control problem we want to

solve.

Multi-Echelon Approach - As stated in Section 1.6, the inventory planning system already in place in the company is a single-echelon one, and treats each entity separately. This optimization process splits the complete inventory control problem into separated entity-item couples, chooses a supplying source according to fixed decision rules for every couple, and computes the optimal order for these contexts according to the forecast. The warehouse then aggregates all orders originating from the stores and allocate its available inventory. In our approach, we want to consider the inventory control system in its whole to propose an ordering plan for all entity-item couples of the supply chain. This allows, among other things, to avoid the supplying source selection and the allocation phases.

Multi-Item Constraints - The problem we choose to tackle is also subject to multi-item costs or discounts. The first multi-item discount mechanism is the so-called *Franco penalty*. This is an extra constant cost charged by some suppliers if the total purchase price of an order does not exceed a given amount. This penalty is specific to some external suppliers and only applies to orders placed to them. The second multi-item cost is the fixed ordering cost charged to every order placed in the system. This represents the processing cost of the orders, and does not only concern the orders placed to suppliers but also the internal orders from stores to warehouse. Both of these costs (fixed and Franco) apply on complete orders, *i.e.* all the items ordered to a supplying source. A single-item order engine cannot treat these costs explicitly, because the orders are calculated for separate items. As a result, such an optimization method is "blind" to those costs, even if we can reach some thresholds by placing high orders. We want to include these constraints in the optimization and therefore we consider a multi-item problem.

Multi-Period Planning - Finally, we want to propose a plan that can tackle all the aforementioned constraints and may also anticipate some further behavior of the demand. This is motivated by two main aspects. First, we are provided with forecast on several planning period and want to take advantage of this available data. Second, we know that treating the multi-item constraints sometimes requires ordering more to reach thresholds. Multi-period planning is then interesting in such cases, because it can help the order engine to order relevant additional quantities.

As a result, we aim to solve a multi-echelon, multi-item, multi-period inventory control problem. This problem demands computing the best orders for a 1WNR-like supply chain with dual sourcing. The objective is to minimize the global inventory levels, the total costs, and to maximize the satisfied customer demand (*i.e.* minimize shortage). The model is enhanced with industry constraints and specifications that allow us to better answer the actual situation faced by inventory controllers in practice. Consequently, the model we develop is rich and complex, but more realistic and usable in practice. The constraints and specifications included in the model are detailed in the next section.

2.2.2 Specific Constraints and Modeling Choices

In this section, we focus on the case-study specificities. We detail each one of them in the following, and also the additional constraints we took into account for making the model more generic and applicable to various customer contexts. For most constraints, we need to make modeling choices

in order to translate them while staying as close as possible from the reality. These choices are also explained in the following.

Supplier Specifications - As mentioned above, suppliers are external and not included in the active entities set. This means we do not manage their inventory. In the work we present, suppliers are considered to be perfect *i.e.* we assume they have unlimited capacity, and their lead times are completely reliable. In other words, this means that there is no uncertainty on the supply, and everything that is ordered is delivered at the time planned.

Demand Forecast - The external customer demand faced by the stores is not known in advance. Still, we need some information on this demand in order to take suitable ordering decisions. To tackle the uncertainty of the external demand, we use a likely estimation of the demand for the upcoming planning period. This estimation is based on the combination of the past demand observations and of a set of temporal features, that are used by machine learning-based forecasting engines. These forecasting engines are given by the company and provide us with a probabilistic forecast for the case under study and for a given time span. As a result, we consider this probabilistic forecast to be an input data for our study, and this forecast is used to derive demand scenarii for our model. For the model, we choose to use the mean of the forecast as the demand for the planning periods. This choice may be changed without any impact on the proposed method.

Ordering Calendar - Active entities are not open every day of the planning horizon. This means that, on closed days, the entities can neither place orders nor receive past orders. On these days, external customer demand is also null. The ordering calendar is a piece of data that states which days of the planning period the entities are open or closed. Every entity has its own calendar. We use this parameter to force orders and demand to be null on closed days. Concerning the receiving process, as demand is null on closed days, considering the entities to be open or closed for receiving is mathematically equivalent for the inventory dynamic.

Shortage - When the total amount of customer demands exceeds the current inventory level of an entity, the inventory level system faces a product shortage. There are two options to model the customer behavior in such cases (ElHafsi et al. [2021]). We can consider the customer is patient and willing to wait until their demand is fulfilled, and in this case the demand is delayed to the next period. This is the *back-order* strategy. On the other hand, the customer can be impatient and will abandon their demand or search to fulfill it elsewhere, and accordingly the unanswered demand is deleted. This is the *lost sales* strategy. Both strategies incur a penalty for the system, either due to unwanted waiting-time, or to customer dissatisfaction. In our case, the customer behavior is better represented by the lost sales strategy. The penalty charged to the lost sales is modeled as a function of the sale price of items in the different entities. As the sale price for a given item can vary among the entities, this penalty will be specific to entity-item couples.

Transportation Cost - In the problem we study, we face different types of variable ordering costs. There are mainly two cost categories we encounter. The first category corresponds to the purchasing cost provided by the external suppliers. These purchasing costs are the result of different buyer-provider purchase agreements. They usually include transportation costs. The second variable cost category relates to transportation cost inside the active entities network. In

our case, this corresponds to the transportation cost of goods from the warehouse to the stores. In this situation, the products are already purchased and only the items' shipment is charged. In the model, both of these cost categories are grouped under the same cost, that we name *transport cost* or *purchase cost*.

Capacity - In practice, entities that hold inventory do not have an infinite capacity and cannot hold more than a given quantity of products. However, due to many uncertainties, they may sometimes receive more products than they can store. In such cases, the extra quantity is not thrown away, but this causes inventory management difficulties. To model this particular case, we choose to introduce an overcapacity penalty. This translates the fact that exceeding the storage capacity is accepted but not encouraged.

Frozen Horizon - In a real inventory control context, the orders placed are rarely delivered instantaneously, as the products have to be shipped from another entity. As a result, an order placed at a time period will be received after a processing and shipping time span, which is called *lead time*. The orders previously placed that still did not go through the entire lead time are referred to as the *in-progress orders*. To model these in-progress orders, we divide the time horizon modeled into two parts, as described in Boulaksil [2016]. The first part of the horizon contains the in-progress orders. As these orders are already placed, they cannot be modified. As a result, this part is called the *frozen horizon*. The second part is the actual planning horizon, where decisions are computed. We provide a schematic representation of this concept in Figure 2.2.

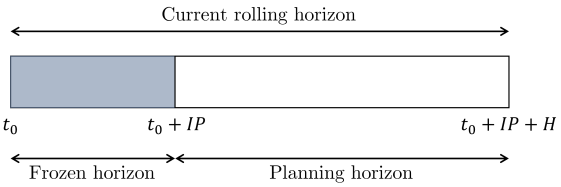


Figure 2.2: Schematic representation of the frozen horizon concept.

Objective - The problem we study contains several costs and aspects to optimize. Some of them represent "real" costs (*i.e.* monetary). These real costs include the purchase and transportation costs, the fixed costs and the Franco penalties. On the other hand, there are also costs that represent parameters used to influence the decisions on a desired behavior. For example, we want to reduce lost sales, so we charge a lost sale penalty to reflect this intention. This also includes the storage cost, to reduce the inventory level, and the exceeding capacity cost. As a result, the objective function is an aggregation of all these costs.

All of the characteristics listed and detailed above are elements that must be included in the proposed mathematical model. This model allows us to represent the described context mathematically and build a method to solve it. In the next section, we define the problem mathematically and provide the complete notations and model.

2.3 Proposed Mathematical Model

In order to develop a solving method for the previously described problem, the first step is to translate it into a mathematical model. Linear programming is a language that has been widely used in service operations, as stated in Kang et al. [2015]. Moreover, this approach has the advantage of being easy to use and to implement, and there exist many solvers that return the optimal solution for such models. As a result, we choose to model our problem by using Mixed Integer Linear Programming.

In the following, we define the Mixed-Integer Linear Program (*MILP*) corresponding to our problem. In Section 2.3.2, we list the data and notations used for the model, and define the decision variables. In Section 2.3.3, we describe the objective function and constraints that define the problem.

2.3.1 Linear Programming

Linear Programming (*LP*) and Integer Linear Programming (*ILP*) are efficient ways to model optimization problems. Linear programming was introduced by George Dantzig in 1947 in order to model and solve decision-making problems for industrial contexts. He also introduced the simplex algorithm, aimed at optimally solving linear programs. This algorithm and its generalizations have had a lot of success and dominated practical operations research for half a century (Nash [2000]). Consequently, linear programming can be found in a significant portion of the optimization literature in general, as a modeling tool as well as a solving method.

In the field of multi-echelon inventory optimization, the problems under study often lead to complex models and large instances to solve, and accordingly it is difficult to obtain solutions in a reasonable time, especially for practical contexts. Hence, a lot of multi-echelon inventory control references combine linear programming with heuristics to find good solutions within a reasonable time span.

Alvarez et al. [2015] study service differentiation in a two-echelon spare parts supply chain with back-order and emergency shipment. They model the problem with linear programming and develop a custom column generation algorithm to solve it. Fattahi et al. [2015] investigate a multi-period inventory planning problem in a two-echelon supply chain with uncertain demand. They propose a *MILP* to solve the problem in a centralized (*i.e.* make decisions for all entities jointly) and decentralized (*i.e.* make decisions for entities separately) setting. For instances where the *MILP* solver is not able to find an optimal solution, they provide a heuristic method based on evolutionary strategies. Lu et al. [2015] study assemble-to-order inventory systems and combine a linear program model with a sample path analysis to characterize the structure of the optimal ICP and find the expression of its parameters. Shu and Karimi [2009] develop a heuristic based in solving a finite series of linear programs or mixed-integer linear programs to solve a safety stock placement problem in general acyclic supply chains.

Linear Programming can be solved using the simplex method. This algorithm is very time-efficient, which makes linear programming a very interesting modeling and solving tool. However, this requires all the decision variables to be continuous. If all decision variables are integer, in an Integer Linear Program (*ILP*), of even if only a part of them are integer, in a Mixed-Integer Linear Program (*MILP*), this property is not preserved. As a matter of fact, *ILP* and *MILP* are known to be NP-hard problems. The on-the-shelf solvers for linear programming provide efficient resolution methods for *ILP* and *MILP*, usually based on several use of the simplex method in a

branch-and-cut resolution method. However, due to the complexity of the model, these resolution method, as efficient they are, can still have a significant execution time. In the case we study, we do not allow the model to order portions of items, so the ordering quantities are necessarily integers, which forces us to develop at least a *MILP* model.

2.3.2 Data and Notations

In this Section, we detail the input data used in the *MILP* mathematical model we propose for solving the problem under study. As there is a significant amount of data to include, we separate them into different categories. For the definition of the different sets, we choose to link the items to the flows they transit on or to the entities they are sold in. As a result, we propose to use a set of flow-items, that are composed of a source-entity, a target entity and an item, and a set of entity-items. For the treatment of multi-item constraints, we also have a set of flows, composed of a source entity and a target entity. This modeling choice has two advantages. First, we can take into account the fact that all entities are not linked together, and that all items are not provided in every entity. Moreover, it makes the notation lighter in the indices, and accordingly clearer.

Time Related Data

H - Planning horizon length
FH - Frozen horizon length
$T^- = [\![1, FH]\!]$ - Frozen time horizon
$T = [\![FH + 1, FH + H]\!]$ - Planning time horizon
$T^+ = [\![FH + 1, FH + H - 1]\!]$ - Reduced planning time horizon, used for the inventory dynamic formula

Set Definition

E - Set of active entities. Entities are denoted by the index e.
ST - Set of entity-items (*i.e.* [entity - item] couples). Entity-items are denoted by the index s.
FL - Set of flows (*i.e.* [source entity - target entity] couples). Flows are denoted by the index l.
F - Set of flow-items (*i.e.* [source entity - target entity - item] triplets). Flow-items are denoted by the index f.

Set Characteristics

EN_s - Entity associated to entity-item s
IT_s - Item associated to entity-item s

SF_l - Source entity associated to flow l
TF_l - Target entity associated to flow l

SE_f - Source entity associated to flow-item f
TE_f - Target entity associated to flow-item f
IT_f - Item associated to flow-item f

Cost Related Data

Ct_f - Transport and purchase cost for flow-item f
Ch_s - Holding cost for entity-item s
Cs_s - Shortage cost for entity-item s
Cf_l - Franco penalty for flow l
Ck_l - Fixed cost for flow l
Ca_e - Exceeding capacity cost for entity e

Additional Input Data

LT_f - Lead time associated to flow-item f
$O_{t,f}$ - In-progress order at time $t \in T^-$ for flow-item f
$C_{t,e}$ - Boolean stating the entity e is open at time $t \in T$
$I0_s$ - Initial inventory level for entity-item s
$D_{t,s}$ - External demand at time $t \in T$ for entity-item s
FR_l - Franco threshold of flow l
W_e - Holding capacity of entity e
M - Big M for constraint linearization

Subset Definitions

$F_s^- = \{f \in F \mid TE_f = EN_s,\ IT_f = IT_s\}$ - Set of flow-items targeting to an entity-item s
$F_s^+ = \{f \in F \mid SE_f = EN_s,\ IT_f = IT_s\}$ - Set of flow-items originating from an entity-item s
$F_l^{\cup} = \{f \in F \mid SE_f = SF_l,\ TE_f = TF_l\}$ - Set of flow-items that constitute a flow l
$IN_e = \{s \in ST \mid EN_s = e\}$ - Set of service-points that constitute entity e
$A_e = \{f \in F \mid TE_f = e\}$ - Set of flow-items targeting entity e

Variables

$q_{t,f}$ - Quantity of products transported for flow-item f at time t
$i_{t,s}$ - Inventory level for entity-item s at time t
$y_{t,s}$ - Amount of demand that is not satisfied for entity-item s at time t
$a_{t,e}$ - Amount of additional inventory at entity e and time t

$$\alpha_{t,l} = \begin{cases} 1 & \text{If an order is placed on flow } l \text{ at time } t \\ 0 & \text{Otherwise} \end{cases}$$

$$\beta_{t,l} = \begin{cases} 1 & \text{If the total amount of orders of flow } l \text{ at time } t \text{ does not exceed } FR_l \\ 0 & \text{Otherwise} \end{cases}$$

The variable q is the main decision variable of the model. Other variables are so-called intermediate variables that are required to compute costs, and are deduced from the values of q.

2.3.3 Model

Objective

$$\min \quad \sum_{f\in F}\sum_{t\in T} Ct_f q_{t,f} + \sum_{s\in ST}\sum_{t\in T}(Ch_s i_{t,s} + Cs_s y_{t,s}) + \sum_{e\in E}\sum_{t\in T} Ca_e a_{t,e} + \sum_{l\in FL}\sum_{t\in T}(Ck_l \alpha_{l,t} + Cf_l \beta_{t,l}) \tag{2.1}$$

Constraints

$$\forall t \in T^-,\ \forall f \in F \qquad q_{t,f} = O_{t,f} \tag{2.2}$$

$$\forall s \in ST \qquad i_{FH+1,s} = I0_s \tag{2.3}$$

$$\forall t \in T,\ \forall s \in ST, \qquad \sum_{f\in F_s^+} q_{t,f} \leqslant i_{t,s} + \sum_{f\in F_s^-} q_{t-LT_f,t} - (D_{t,s} - y_{t,s}) \tag{2.4}$$

$$\forall t \in T,\ \forall s \in ST, \qquad y_{t,s} \geqslant D_{t,s} - \left(i_{t,s} + \sum_{f\in A_e} q_{t-LT_f,f} \right) \tag{2.5}$$

$$\forall t \in T,\ \forall s \in ST, \qquad y_{t,s} \leqslant D_{t,s} \tag{2.6}$$

$$\forall t \in T,\ \forall e \in E, \qquad a_{t,e} \geqslant \sum_{s\in IN_e} i_{t,s} + \sum_{f\in A_e} q_{t-LT_f,f} - W_e \tag{2.7}$$

$$\forall t \in T^+,\ \forall s \in ST, \qquad i_{t+1,s} = i_{t,s} + \sum_{f\in F_s^-} q_{t-LT_f,f} - \sum_{f\in F_s^+} q_{t,f} - (D_{t,s} - y_{t,s}) \tag{2.8}$$

$$\forall t \in T,\ \forall e \in E, \qquad \sum_{f\in A_E} q_{t,f} \leqslant MC_{t,e} \tag{2.9}$$

$$\forall t \in T,\ \forall l \in FL, \qquad \sum_{f\in F_l^\cup} q_{t,f} \leqslant M\alpha_{t,f} \tag{2.10}$$

$$\forall t \in T,\ \forall l \in FL, \qquad \beta_{t,l} \leqslant \alpha_{t,l} \tag{2.11}$$

$$\forall t \in T,\ \forall l \in FL, \qquad FR \times \beta_{t,l} \geqslant FR_l \alpha_{t,l} - \sum_{f\in F_l^\cup} Ct_f q_{t,f} \tag{2.12}$$

Domains

$$\forall t \in T^{-} \cup T,\ \forall f \in F, \qquad q_{t,f} \in \mathbb{N}$$

$$\forall t \in T,\ \forall s \in ST, \qquad \{i_{t,s}, y_{t,s}\} \in \mathbb{R}^{+}$$

$$\forall t \in T,\ \forall e \in E, \qquad a_{t,e} \in \mathbb{R}^{+}$$

$$\forall t \in T, l \in E, \qquad \{\alpha_{t,l}, \beta_{t,l}\} \in \{0,1\}$$

The formula 2.1 is the objective function we want to minimize. This function includes all the different costs we take into account for the optimization. The first sum relates to the purchase and transport cost that applies on all quantities ordered. The second sum groups the inventory holding cost, applying on all the items remaining in stock at the end of each period, and the lost sale cost, applying on every unit of unsatisfied demand. The third sum refers to the exceeding capacity cost, that applies on all hold items that create stock that exceeds the capacity. The last sum relates to the multi-item costs. This includes the fixed costs, charged on every flow that orders at least one positive quantity for an item, and the Franco penalty, charged on every flow for which the order price is not null and under a given threshold. The values of the variables are defined in the constraints, that we define in the following.

Constraint 2.2 assigns the in-progress orders to q variables of the frozen horizon. This allows to compute receivings for the inventory dynamic. Constraint 2.3 assigns the initial inventory levels to the corresponding entity-items at the beginning of the planning horizon. These two constraints initialize the fixed variables of the model with the input data.

Constraint 2.4 ensures that the entities do not send more items than they hold in their inventory. In our case specifically, this only concerns the warehouse. This constraint is the only "hard" constraint of the model, as it translates a physical limitation on the quantities to be transported within the network. Other constraints are there to define the values of the variables.

Constraint 2.5 and 2.6 define the value of lost sales variables according to the external demand and available inventory. In our case-study specifically, this concerns only the stores, as the external demand for the warehouse is set to zero (and accordingly the lost sales will be null).

Constraint 2.7 defines the value of the excess inventory. This value is based on the entities total inventory considering all items combined, compared to their holding capacity.

Constraint 2.8 describes the inventory dynamic. For a given time period, it computes the resulting inventory level for the next period based on the current inventory, the amount of items delivered and the external demand satisfied.

Constraint 2.9 ensures entities do not place orders if they are closed, based on the calendar provided.

Constraint 2.10 defines the value of the boolean variable for counting the fixed costs, and constraints 2.11 and 2.12 define the value of the boolean variable for counting the Franco penalty. Both these variables are defined according to the orders aggregated on flows.

Based on the model definition and the linear program, we can compute the number of constraints and variables corresponding to given instance sets' sizes. The number of variables can be obtained with formula 2.13, and the number of constraints with formula 2.14.

$$\textit{Number of variables} = H(|F| + 2|ST| + |E| + 2|FL|) \tag{2.13}$$

$$Number\ of\ constraints = FH|F| + T(4|ST| + |E| + 4|FL|) \tag{2.14}$$

We can see that the feature that contributes the most to both of these equations is the planning horizon length H, as it is linked to every variable and almost every constraint. As a result, adding one planning period to the problem will significantly increase its complexity. Moreover, there is another feature that has a non negligible effect on the complexity. This feature is the number of items. This number does not appear directly on the formula, but has an impact on the sets' sizes. To explain this, let us assume we add an item to a given instance, that is sold only in one store of the network, and provided by one supplier. In this case, the size of the set F will be increased by three, as it creates one flow-item with this new item for the three following flows: supplier $\rightarrow$ warehouse, supplier $\rightarrow$ store and warehouse $\rightarrow$ store. Moreover, it also increases the size of the set ST by two, as it creates one entity-item for the warehouse and one for the store. If the store in question is not in the network of the original instance, it also increases the size of the sets E and FL by one each. If the supplier is not in the original instance, it also increases the size of the set FL by two – one flow for the warehouse and one flow for the store. Now if we add several items, that are all sold in several stores and provided by several suppliers, we can easily conclude that this will have a significant impact on both the number of constraints and the number of variables. We provide a representation of the increase of the sets incurred by the addition of one item that is supplied by an existing supplier and sold in a new store exclusively in Figure 2.3.

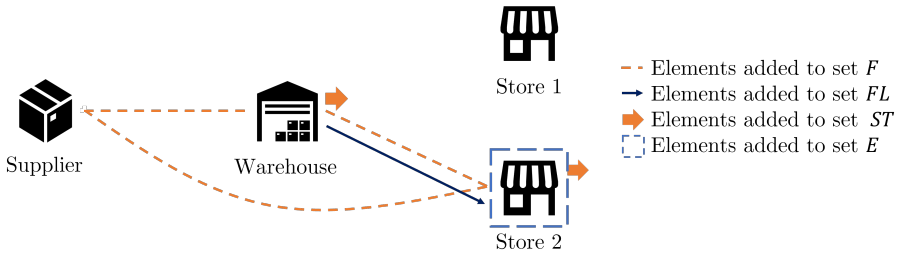


Figure 2.3: Representation of the elements added to an instance for the inclusion of an item supplied by an existing supplier and sold in a new store exclusively.

2.4 Input Data and Problem Instances

Now that we have defined the model, let us focus on the data that we will use throughout our work. As explained in Section 2.1, we use a dataset drawn from a real case to build the data for testing the proposed model and solving methods. This dataset provides a large part of the input data necessary to build the model detailed above. However, there are some necessary data that are missing from the dataset provided. This may come from additional constraint we included in the model, or from the company not providing all the necessary data. This is further discussed in Section 2.4.1.

To test our model, and later also the solving methods we propose, we need to define several instances from the complete dataset provided. We choose to select subsets of the total data to create instances. This is motivated on one hand by the complexity of the input data, and on the other hand by the need for testing diverse contexts. We detail this part in Section 2.4.2.

2.4.1 Additional Elements for Data Completion

The first type of data for which we have to make assumptions concerns the cost element that are applied to influence the model behavior (*cf.* the objective function paragraph in Section 2.2.2). As explained before, they do not represent monetary costs, and therefore their exact values are not known. For our model specifically, this concerns the holding cost, the lost sales cost, and the capacity exceeding cost. These parameters need to be calibrated according to customer preferences and are usually the result of an A/B testing procedure. A/B testing refers to the process of intuitively comparing pairs of parameter values to choose the "best" values.

The second missing type of data corresponds to the constraints we added to the case-study to make it more representative of a generic problem. Accordingly, as these constraints are not present in the initial use-case, the data describing them are not provided. As we still want to treat these problem characteristics, we choose arbitrary values for the parameters related to them. In our case specifically, this refers to fixed ordering costs and entity capacities.

Finally, the last missing type of data are due to the data not being provided. This can be caused by several factors, such as a problem in the data transmission, a lack in the company's information system, or if the process is based on manual manipulation of these data. In most cases, the company is not able to provide the missing data quickly. Consequently we have to choose an arbitrary value. In our case specifically, this corresponds to Franco penalty values.

2.4.2 Instance Construction

In this section, we describe the process we set in place to build the reduced instances drawn from the complete dataset.

The dataset associated to our case-study contains 905 different items, with various forecasted demand profiles. We want to build instances by selecting subsets of items and keep the corresponding supply network they transit through. To ensure diversity in the instances we create, we define several criteria to select the subsets of items. This is what we call *generation modes*.

Moreover, we are aware that the forecast profile of the items is very variable among the item set. We provide an overview of the forecast values distribution among the item set in Table 2.1.

Table 2.1: Forecast profile distribution overview. At each line, we select items that meet the forecast limit requirement for at least one period and in at least one store.

Forecast limit	Number of items	Percentage
≤ 1	622	68.72%
$\geqslant 1$	283	31.27%
$\geqslant 5$	40	4.42%
$\geqslant 10$	10	1.10%

Given the results presented in this table, we know that around 70% items have a very low forecast, namely the mean of the forecast is lower than 1. in all stores and for all periods of the forecasting horizon. As a result, the selection strategy used to pick items for creating the reduced instance has a high impact on the forecast profile of items in these instances, and we put several generation modes in place in order to obtain diverse instances. In the following, we describe every generation mode used to create instances.

Alphabetical Order (`alph`) - In this category, we select items in the alphabetical order of their names in the company's information system. The item's name starts with the label of the external supplier that provides them. As a result, selecting them alphabetically gives instances with a gradually expanding supply network as the number of items increases.

Random (`rand`) - In this category, we select items randomly. The first characteristic of these instances is their diversity, as their is almost no pattern in the data selected. Secondly, as most of the items of the given dataset have a low forecast, this produces instances that are rather easy to solve, especially for small instances.

Average Forecast (`fora`) - In this category, we select items that have the highest forecast averaged on all the time periods for which forecast is provided. As a result, we obtain instances with items having a high forecasted demand. This implies problems that are more difficult to solve, but the ordering plans will be more interesting.

Maximum Forecast (`for`) - In this category, we select items that have the highest maximum forecast on all time periods for which forecast is provided. Like in the previous category, this yields instances with higher forecasted demand and more interesting ordering plans. This allows us to have a second batch of difficult instances, still a bit different from the one described in the previous category.

Random Average and Maximum Forecast (`randfora`, `randfor`) - In this category, we select items randomly, but not from the whole perimeter. We first select a subset of items that have the highest forecast averaged on all time periods for which we have forecast, and extract items among this list. Like the previous category, this allows to have more difficult instances, especially for small instances. We also draw instances from a subset of items that have the highest maximum forecast on all time periods.

Given the forecast distribution presented in Table 2.1, for the arbitrary generation mode (`alph`) and random one (`rand`), there is a high probability that we only select low forecast-items. On the contrary, for the other generation modes, as we base the item selection rule on high forecast values, we know that it selects only items with higher forecast, at least for small instances. With all generation modes described above, we are able to draw a substantial set of instances that we can use for our experimentations. In the remainder of the thesis, we refer to the instances with the following naming convention. Let `gen` be the generation mode of the instance, and P the number of items, we name the instance $(\texttt{gen}, P)$. For example, if we use the average forecast generation mode for an instance with 15 items, its notation is $(\texttt{fora}, 15)$. We provide an overview of all the instances types and their characteristics in Table 2.2.

2.5 Experimentations

In order to validate the previously proposed model and to estimate the execution times with free and commercial solvers, we choose to run experiments on reduced instances of the problem. The results of these experiments are useful for two main reasons. First, running the *MILP* with different solvers allows us to have a more precise idea of the model complexity, and to confirm – or not – the need of

Table 2.2: Overview of all generation modes for instances and their characteristics.

Generation mode	Notation for P items	Selection process
`alph`	$(\texttt{alph}, P)$	Alphabetical order of item names
`rand`	$(\texttt{rand}, P)$	Random selection
`fora`	$(\texttt{fora}, P)$	Decreasing average forecast
`for`	$(\texttt{for}, P)$	Decreasing maximum forecast
`randfora`	$(\texttt{randfora}, P)$	Random selection from the 50 items with the highest average forecast
`randfor`	$(\texttt{randfor}, P)$	Random selection from the 50 items with the highest maximum forecast

a heuristic method to solve large instances of the problem. Second, the optimal solution obtained at the end of these experiments will be able to serve as baseline for the convergence performance evaluation of the heuristic method we set in place in Chapter 3. We detail the experimental protocol in Section 2.5.1. We present and discuss the results in Section 2.5.2.

2.5.1 Experimental Protocol

For the exact solving method of our *MILP*, we decide to run the experiments using several well-known solvers. First, we try to solve the model using CPLEX v22.1.1.0 (IBM ILOG [2023]). CPLEX solver is currently known to be one of the most efficient commercial solver in the literature. However, the company does not wish to acquire a CPLEX license for industrial use. In order to provide results using tools that will be available in practice, we also run the experiments using the CBC solver (Forrest and Lougee-Heimer [2005]), which is a free solver. In order to provide diverse results, we also want to run experiments with a solver that would have shorter execution times than CBC, which is known to be slow. We choose SCIP (Bestuzheva et al. [2023]), which is also a commercial solver. In the literature, SCIP is known to be an "average" alternative between CBC and CPLEX in terms of execution time.

For the resolution with CPLEX, we use IBM ILOG CPLEX Optimization Studio, the integrated development environment (IDE) provided with the solver. This IDE allows to code and solve the model. For the resolution with CBC and SCIP, we use the Python PuLP library (Mitchell et al. [2011]). This library enables to code the model using Python programming language, provides an interface to call *MILP* solvers, among others CBC and SCIP, to solve the model.

Due to the high computation time, we cannot test all instances and therefore we chose only four generation modes. We select a subset of instances to run our experiments. Among the generation modes described in Section 2.4.2, we select the following ones: $\{\texttt{alph}, \texttt{rand}, \texttt{fora}, \texttt{randfora}\}$. This choice is motivated by the will to test our model on diverse instances, and therefore we choose two generation modes giving items with low forecast, and two generation modes giving items with high forecast. As a result, this allows us to test the model on the different forecast profiles we encounter in the data. Regarding the instance sizes, we select the following item numbers for instances: $\{5, 15, 25, 30, 40\}$. In this way, we can test all of the instance sizes available for the `randfora` generation mode.

For each instance we selected, we try to solve it using the model presented above with each chosen solver. We give the solvers a 7200 seconds time limit, which is a company requirement for a solving method to be put into production. We also set a relative gap limit of 1%. The relative gap is computed by the solvers during the branch and bound process.

2.5.2 Results and Discussion

In this section, we present the results we obtained by solving the selected instances with the exact *MILP* solvers. As described in the previous section, we use three different solvers for our experimentations, and give them 7200 seconds to find the optimal solutions for the instances. In Table 2.3, we report the execution times obtained by the CPLEX solver during the experimentations. Both other solvers timed out on all instances, meaning that they do not reach the 1% gap within the time limit. A graphical representation of these results is also provided Figure 2.4

Table 2.3: Execution time in seconds for solving instances with CPLEX, 7200 second time limit, 1% relative gap limit. The note "Time out" means the solver did not reach the gap within the given time span.

Instance	`alph`	`rand`	`fora`	`randfora`
5 items	77.68	15.58	88.89	44.81
15 items	1087.28	1398.90	635.89	1621.09
25 items	1291.58	4288.03	2362.30	1941.48
30 items	5423.61	*Time out*	6274.37	5240.29
40 items	6574.59	6278.47	*Time out*	*Time out*

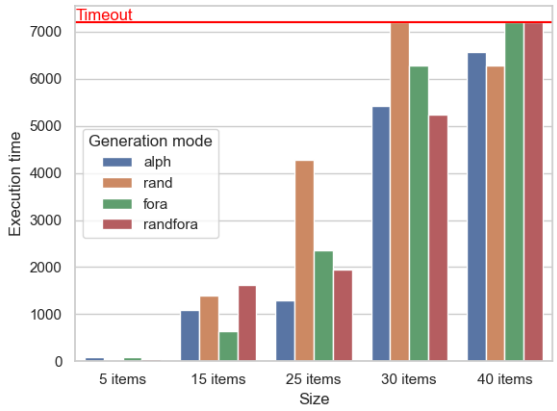


Figure 2.4: Graphical representation of the execution times in seconds obtained with CPLEX for every instance.

As we expected, we can see that the execution time quickly increases with the number of items. This is an expected result because we know that more items means more decision variables and therefore the instances are more complicated to solve.

For 40 items, which is the highest instance size we selected for this experimentation, CPLEX found a solution within around one hour and a half for only two instances. For the two remaining instances, CPLEX timed out without reaching an initial solution. Remembering that the complete problem contains 905 items, we cannot expect to find a solution with an exact solver within the given time limit.

Moreover, we can note that the two instances for which CPLEX timed out are the ones that have a higher forecast profile. And even for smaller problems, we can see that these instances are generally longer to solve than the instances with lower forecast profile. As a result, we can conclude that a higher forecast profile implies a higher complexity in the solving of the problems.

In order to have a more precise view on the solving process of each solver, we also report the relative gaps found during the resolution for each instance in Table 2.4.

Table 2.4: Relative gaps found by the solver for all instances. The notation "*Not found*" means the solver did not find any solution to compute the gap. The notation "~ 1 %" means the solver reached the gap limit within the time limit.

Instance	`alph`			`rand`		
	CBC	SCIP	CPLEX	CBC	SCIP	CPLEX
5 items	85.62 %	49.76 %	~ 1 %	50.45 %	29.15 %	~ 1 %
15 items	45.69 %	21.80 %	~ 1 %	24.07 %	27.04 %	~ 1 %
25 items	*Not found*	14.65 %	~ 1 %	20.97 %	19.42 %	~ 1 %
30 items	*Not found*	17.81 %	~ 1 %	16.21 %	10.52 %	*Not found*
40 items	*Not found*	14.92 %	~ 1 %	*Not found*	17.35 %	~ 1 %
Instance	`fora`			`randfora`		
	CBC	SCIP	CPLEX	CBC	SCIP	CPLEX
5 items	79.24 %	54.36 %	~ 1 %	79.24 %	54.51 %	~ 1 %
15 items	70.68 %	49.34 %	~ 1 %	69.13 %	45.51 %	~ 1 %
25 items	*Not found*	27.33 %	~ 1 %	*Not found*	27.35 %	~ 1 %
30 items	*Not found*	24.20 %	~ 1 %	*Not found*	24.14 %	*Not found*
40 items	*Not found*	17.73 %	~ 1 %	*Not found*	17.71 %	*Not found*

We can see that CBC is the solver having the more trouble finding solutions. In fact, the solver is unable even to compute a relative gap for instances size above 15 items, except for `rand` generation mode. As this solver is the only one we can consider to use in a real-world situation, we cannot think about exact resolution as a tractable solution to our problem.

2.6 Conclusion

In this chapter, we detail the industrial problem we want to tackle, and translate it into a mathematical model. This part of our work allowed us to formalise and describe the specificities and constraints we face with the multi-echelon replenishment problem defined by our case study. As the

problem comes from an industrial application, we have to add a significant number of constraints to the model. Some of these constraints, such as the Franco constraint for example, are rather unusual in academic problems.

In addition, we also describe supplementary specifications that make the study more representative of replenishment problems encountered in practice. This phase provides more genericity to the model, and accordingly makes it easier to use and adapt to several industrial contexts. All of this leads to a complex model with a high number of constraints and decision variables.

To model the problem, we choose linear programming, which is a widely used modeling language in the replenishment and inventory control field. We build a mixed integer linear program, or *MILP*, that enables us to transcribe the real case study into a mathematical model that is the base for the problem resolution. Furthermore, the modeling part of our work allows us to establish a better knowledge and understanding of the problem and its specific features.

As there exist several on-the-shelf solvers for Integer Linear Programming, this modeling language choice also gives us the opportunity to solve the model by finding the optimal solution. Given the size and complexity of the model we choose to draw reduced instances from the data provided for the case-study. This leads to smaller – and accordingly easier to solve – instances. Moreover, we develop several processes to create instances. This gives us the possibility to use various and diverse instances in our experimentations. We select a subset from these instances and solve them using three exact solvers: CBC, SCIP and CPLEX, giving them a time budget of two hours. As we presumed, the time to solve these instances quickly increases and every solver is unable to find a good solution for most instances with a size of 40 items. Recalling that these instances are only small subsets of the total data, we cannot consider exact resolution through *MILP* solvers as a satisfying solution to our problem.

As a result, if we want to find a solving method that can tackle both the complexity of the model and the limited amount of time to find a good solution, we have to explore other solving strategies. This solving strategy, as well as the design choices we made and the reasons motivating them, are described in Chapter 3.

Chapter 3

Heuristic Resolution

Contents

3.1 Introduction 44
3.2 Resolution Requirements and Method Selection 45
3.3 Local Search: Overview 47
3.3.1 Solution Representation 48
3.3.2 Initial Solution Generation 48
3.3.3 Neighborhood Operators 52
3.4 Dedicated Neighborhood Operators 52
3.4.1 Addition Neighborhood Operators 53
3.4.2 Deletion Neighborhood Operators 55
3.4.3 Rescheduling Neighborhood Operators 57
3.4.4 Saturate Franco Neighborhood Operator 58
3.4.5 Overview 58
3.5 Use of Multiple Neighborhood Operators in the Search 59
3.6 Variable Adaptive Local Search 62
3.6.1 Overview of the Algorithm 62
3.6.2 Score Initialization and Computation 64
3.6.3 Neighborhood Operator Selection 64
3.6.4 Score Evolution 65
3.6.5 Termination criteria 65
3.7 Experimentations 67
3.7.1 Method Validation 67
3.7.2 Global Performances 69
3.8 Conclusion 69

3.1 Introduction

In the previous chapter, we detail the specifications of the problem we aim to solve. We also provide a model to mathematically represent the problem, and better understand its constraints and complexity. We conduct experiments with the model by trying to solve it with exact solvers and with data drawn from a real-life case study. The main outcome of this work is, although our model is relevant for the problem we study, the size of the instances and the time window we are given to provide a solution make an exact resolution unthinkable in the situation we work on.

Given that, we have to find another method to provide a solution to the problem we study that is applicable to the instance size of the industrial cases we want to tackle. There exist plenty of optimization methods that can be used for various contexts. For inventory control problems specifically, we can find many methods and techniques used for such situations. As we can see in the state of the art presented in Chapter 1, there is no consensus on a single approach to solve inventory control problems. This is partly due to the diversity of contexts that replenishment problems includes. In fact, while surveying the articles, it is rare to find two identical contexts, as every study includes different aspects, constraints or assumptions, in order to deal with the effective contexts they work with. This makes the choice of a resolution method an important part to focus on, even before the algorithm design. In the process of selecting a suitable optimization method, we identify metaheuristics as an interesting solving method class.

The term metaheuristic refers to a large panel of algorithms that are widely used for optimization problems in general. As metaheuristics include a lot of algorithms, there is not a unique definition of the concept. Osman and Laporte [1996] describe metaheuristics as iterative processes that combine different concepts to explore and exploit the search space. They also characterize them as general algorithm frameworks for diverse optimization problems. Glover and Kochenberger [2006] define metaheuristics as algorithms that combine local improvement strategies aimed at finding good quality solutions with higher level strategies that intend to escape local optima. Talbi [2009] define it as a family of optimization techniques that provide acceptable solutions in reasonable time for complex problems. To summarize, this is a class of algorithms that are aimed at exploring large solution spaces by iteratively improving solutions using exploration and exploitation techniques to find good quality results in an acceptable time. Moreover, metaheuristics are known to be one of the most practical approaches for solving many real-world problems (Onar et al. [2016]).

As mentioned in Section 2.3.1, multi-echelon inventory control problems often lead to large and complex models, even for reduced instances. As a result, metaheuristics are present as solution methods when the exact methods take too long or are unable to find a solution. Noordhoek et al. [2018] study the application of a reorder-point order-up-to ICP on multi-echelon distribution networks with fill-rate constraint. They assess the performance of a scatter search for the computation of the policy's parameters, and compare this metaheuristic with another solving method. Phelan and McGarraghy [2016] develop a grammatical evolution algorithm for generating ICPs and the expression of their parameters for serial and distribution supply networks. Nepal et al. [2011] develop a multi-objective genetic algorithm to solve a supply chain configuration problem for a new product. Kleinau and Thonemann [2004] use a two-stage genetic algorithm to first find the structure of a compliant ICP for a given inventory control problem, and second to find the best expression of its parameters. They apply these methods to several supply network contexts and several assumptions on the demand.

In this chapter, we detail the complete process for developing an optimization method for the problem we face and to apply on the instance size we have to deal with. We begin with the

motivations to choose a given optimization method, leading us to select local search. We then go to the application of this method to instances of our problem. The focus is also put on the design of the actual algorithm that will be used and the validation of this design. As the outcome of this work, we want to provide a multi-echelon, multi-period and multi-item heuristic resolution method for a replenishment problem. This method has to provide good quality solutions for large instances, namely instances with more than 40 items, in a time span that is reasonable enough to respect the industrial requirements.

The Chapter is organized as follows. We first detail the reflection about the selection of the optimization method in Section 3.2. Then, we detail the functioning of the method selected and list the algorithm design choices to be made in Section 3.3. We also provide first insights on the algorithm design in this Section. In Section 3.4, we describe the neighborhood operators we designed for our problem and evaluate their performances. In Section 3.5, we explain the benefits of an algorithm using multiple neighborhood operators. This leads to the development of the final metaheuristic used in this thesis, that is described in Section 3.6. We then present an overview of the performances of the algorithm in Section 3.7, and provide a conclusion in Section 3.8.

3.2 Resolution Requirements and Method Selection

As we previously explained, there is no single preferred method for inventory control problems, nor clear guidelines to choose an optimization resolution method according to the specific context under study. In this Section, we describe all requirements and specifications of the problem we study and of the resolution method we want to design. These specifications are the motivating reasons to the selection of an optimization method that is suitable both for the problem we tackle and for the industrial conditions we have to respect in order to ensure the application of our proposed solution method. In the following, we detail different characteristics of the problem and requirements for the proposed solution method. Each element described allows to guide the selection of a suitable resolution method, and motivates the choice.

Type of Problem - In their survey about simulation-optimization for inventory replenishment problems, Jalali and Nieuwenhuyse [2015] propose a classification of resolution methods depending on the type of problems to solve. This classification aims to be used as a guiding tool to choose a resolution method class, and is presented as a decision tree. The first classifying step is whether the decision variables are discrete or continuous. In the problem we study, we work with a Mixed-Integer Linear Program, but only the intermediate variables are continuous. The main decision variables are discrete. The second decision step is whether the feasible set is finite and small or infinite or large. In our case specifically, the ordering variables are not bounded by the model, so the feasible set is infinite. Moreover, even if we try to compute an upper bound for the optimal solution, there is still a high number of variables, which means a high number of possible variable combinations for the solutions, and makes the feasible set still large. In this case, Jalali and Nieuwenhuyse [2015] recommend metaheuristics or ordinal optimization methods. In Figure 3.1, we provide a representation of the classification proposed in the article, based on the decision tree they provide.

Solution Characteristics - As the problem studied is constrained, we presume that building a feasible solution is not straightforward. Furthermore, some results of preliminary works confirm that building a solution that respects the constraints of the model through random sampling of the

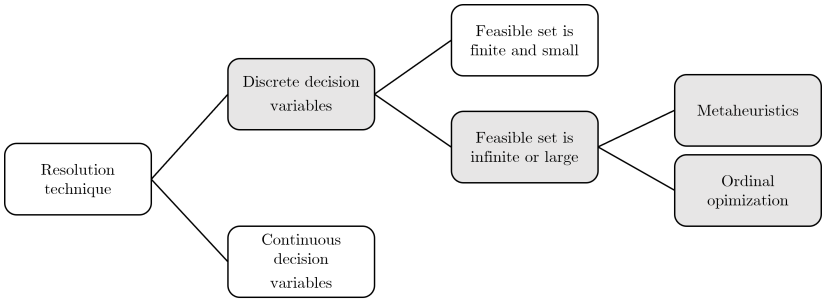


Figure 3.1: Decision tree for choosing a suitable method. We provide only the options that relate to our problem. Grey cells relate to the characteristics of our problem.

variables is not possible. In fact, we launched an experiment generating 10000 solutions through random sampling by using several upper bounds. With all the upper bounds tested, we found no feasible solution within all the generated ones. Consequently, creating a feasible solution, let alone a good quality one, requires effort and hence is time consuming. Subsequently, population-based methods such as genetic algorithms, ant colony or more generally swarming algorithms cannot be considered as a suitable optimization method, as it would require too much computing effort to generate the initial population. It also excludes ordinal optimization methods, which are based on the sampling of a significant number of solutions to find the best ones.

Industrial Requirements - The last element we have to take into account to select an adequate resolution method relates to industrial requirements we have to meet. These requirements are intended to ensure that the proposed method is applicable in real-world contexts. In Section 2.5.1, we mention a time limit of 7200 seconds of execution time for the resolution method. This requirement is given for the case-study specifically, and can be extended to most of the current contexts faced by the company. However, we may encounter other application contexts where this execution time limit is smaller, and the resolution method has to be able to still provide good quality solutions in such cases. Consequently, the resolution method has to have a specific convergence profile, which improves the solution very significantly at the beginning of the search, and may converge more slowly in the remaining execution time given. This type of convergence profile ensures that, even if we interrupt the algorithm before the initial time limit, the algorithm still returns a good quality solution. This definition corresponds to the so-called *anytime algorithms* class (Grass and Zilberstein [1996]).

Method Selected - If we combine every element and specification we detailed above, the outcome is that we want a single-solution metaheuristic algorithm with an anytime convergence profile. One standout algorithm that fills these requirements is the local search. In addition of being suited for the type of problem we aim to solve, this algorithm has many advantages, among others being very simple to understand and to set up, and being easily adaptable to many problems and contexts.

3.3 Local Search: Overview

Local search is one of the oldest metaheuristic methods known, and it is also one of the simplest (Talbi [2009]). In fact, the base principle of the algorithm is very straightforward. The algorithm starts with an initial solution, to which is associated a performance measure, or fitness value. This solution is marginally modified at each search step of the algorithm, creating neighbor solutions, and a neighbor solution with a better fitness value is chosen. The latter solution replaces the initial one, and the process is repeated until a given termination criterion is reached. The complete process is detailed in Algorithm 1.

Algorithm 1 Local Search

1: **Input:** Termination criterion TC
2: Generate initial solution S_0
3: $S = S_0$
4: **while** not TC **do**
5: Generate candidate neighbors of S
6: Choose a neighbor solution S'
7: $S = S'$
8: **end while**

Although the optimization process is simple, there are still design decisions to make in order to effectively implement it. More specifically, there are some elements that need to be determined.

Initial Solution: The initial solution is the starting point of the algorithm. Generating a good initial solution is important, as it determines the beginning of the search. As explained in Section 3.2, finding a feasible solution through random sampling is not possible in the case under study. As a result we need to develop a method to build a rather good initial solution. This is described in Section 3.3.2.

Neighborhood Operators: The neighborhood operators are the sub-processes that generate a candidate solution from a current solution. Neighborhood operators are basically functions that slightly modify a given solution to create new ones. There exist a wide list of operators in the literature, either generic or specific to a problem class. In Section 3.3.3, we introduce our thinking about the design of operators for the problem under study, and the evaluation of their performances.

Search Strategy: The search strategy relates to the order in which the neighborhood will be browsed. As the neighborhood operator modifies the solution, it creates a set of candidate neighbors. Consequently, we have to decide in which order to examine all of the candidate neighbors. For most of the operators we describe in Section 3.4, candidate neighbors are ordered randomly.

Neighbor Selection Strategy: Once the local search examined one or several neighbor solutions, we have to choose one of them to replace the current solution. In the literature, we can find three main strategies for selecting a neighborhood. The first two strategies require an exhaustive exploration of the neighborhood. Once all the possible neighbors have been generated and evaluated, we create a subset with all the neighbors having a better fitness value. Then, we can either pick the



Figure 3.2: Schematic solution representation for a given time period t.

best neighbor of the subset, following the *best improvement* strategy, or randomly pick a neighbor within the subset of improving neighbors, following the *random improvement* strategy. In our case, we work with large-size neighborhoods, and evaluating a neighbor is time consuming, so we cannot perform an exhaustive evaluation of the neighborhood. Accordingly, we choose to select the first improving neighbor we encounter while browsing the neighborhood, following the *first improvement* strategy.

Termination Criterion: The termination criterion is the condition that states when the search process is finished. There exist several termination criteria types, and choosing a suitable one is crucial. As a matter of fact, choosing a good termination criterion ensures that the algorithm provides a solution of good enough quality without wasting too much time at the end of the search. The determination of the termination criterion is discussed later in this chapter.

3.3.1 Solution Representation

Before describing the algorithm design decisions, let us focus on the solution representation. This solution representation is set for the heuristic resolution and will be used throughout this chapter. In Chapter 2, we introduced the variables of the model and classify them into two categories. There are the main decision variables, and the intermediate variables. As the intermediate variables are computed based on the main decision variables values, we choose to only include the main decision variables in the solution representation. As a result, the solution representation is composed of the following set of variables: $(q_{t,f})_{t\in H, f\in F}$. For a given pair (t, f), $q_{t,f}$ is the quantity transiting for flow-item f at time period t.

Hence, the solution is represented as a two dimension table, with the first dimension being relating to the time, and the second relating to the flow-item. In Figure 3.2, we provide an overview of the solution representation for a given time-period t.

3.3.2 Initial Solution Generation

The initial solution is the first solution that is examined and modified by the algorithm. This is the starting point of the algorithm, and its quality can impact the convergence of the search. Consequently, creating a good quality initial solution is an important phase of the algorithm design. As we already mentioned, we work with a constrained problem, which means that creating a solution that is feasible, *i.e.* which respects all the constraints, is not straightforward. Moreover, we already established that random sampling cannot be considered as a suitable generation method. Hence, we have to develop a procedure to create a feasible initial solution in a reasonable time. We describe this procedure for a given problem instance in the following.

We start from a given initial state, *i.e.* a set of in-progress orders O and a set for initial inventory levels *I0*, which serves as base for constructing a feasible solution.

The first step of our procedure is to solve the problem for the given instance using the relaxed version of the *MILP*. In this step, we use the *MILP* described in Chapter 2, but we allow all the variables to be continuous, making it a linear program. As a linear program can be solved efficiently with the simplex algorithm, we can obtain a result relatively quickly with any instance for this phase.

Once we ran the linear program and collect the results, we obtain a first set of orders, that will be used for the initial solution. However, at this stage of the procedure, the orders are not integer, as we allowed the variables to be continuous in the last step. Now, we have to round the variables to get a set of integer orders. To begin with, we split variables into three groups.

- The quantities sent from the supplier to the stores (Group 1)
- The quantities sent from the supplier to the warehouse (Group 2)
- The quantities sent from the warehouse to the stores (Group 3)

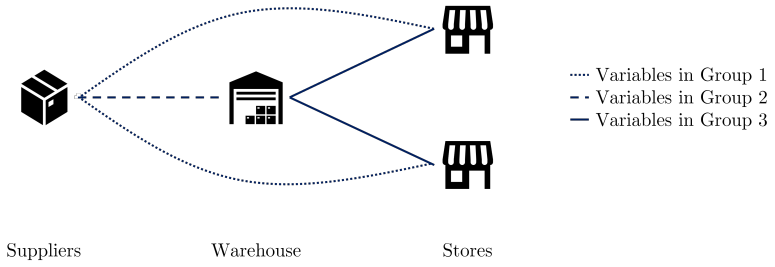


Figure 3.3: Representation of the different groups of variables.

We provide a representation of this partition in Figure 3.3. For every group of variables, we have to choose how to round them among the three following possibilities. We can either round the variable up (+1), round the variable down (−1), or round the variable to the nearest integer (0). These decisions have an impact on the construction if the initial solution, as it modifies the compliance with constraints. As a matter of fact, the main constraint we have to respect is defined in equation 2.4. This equation is in fact one of the constraints defined in the *MILP* in Chapter 2.

$$\forall s \in ST,\ \forall t \in T, \qquad \sum_{f \in F_s^+} q_{t,f} \leqslant i_{t,s} + \sum_{f \in F_s^-} q_{t-LT_f,f} - (D_{t,s} - y_{t,s}) \tag{2.4}$$

For a given entity-item s, we note F_s^- the set of flow-item that supply s and F_s^+ the set of items that are supplied by s. $q_{t,f}$ in the quantity that is transported on flow-item f at period t, $i_{s,t}$ the inventory level for entity-item s at period t, $D_{s,t}$ the demand faced by entity-item s at period t, and $y_{s,t}$ the lost sales level for entity-item s at period t. This constraint models the fact that the entities that supply other entities of the system – in our case specifically, this concerns the warehouse only – cannot send more items than the quantity they have in stock.

For the variables in Group 1, there is no constraint on the feasibility space, as we consider the suppliers have an infinite inventory. Consequently, the selection of a rounding alternative does not have influence on the global feasibility of the solution.

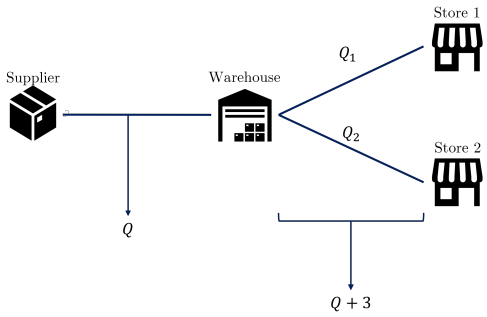


Figure 3.4: Representation of a constraint violation for a given time period and a given item. All lead times are set to 0 in this example.

For the variables in Group 2 and Group 3 however, the combination of both rounding methods chosen may influence the global feasibility of the solution, as both groups are linked by the constraint of equation 2.4. As the quantities sent by the suppliers to the warehouse are the quantities that will be redistributed by the warehouse to the stores, if we choose to round down the variables in Group 2 and round up the quantities in Group 3, there is a high chance that the constraint is not respected, as the quantities sent to the stores may be higher than the ones received by the warehouse. In such cases, we have to set up a process to repair the solution and make it respect the constraints. We describe this process in the following. In Figure 3.4, we provide a schematic representation of a situation where a constraint is broken. This schematic representation is for an artificial case with one supplier, one warehouse and two retailers. We consider a given item and time period, and assume all lead times are set to 0.

Stage 1: Let $S = (q_{t,f})_{t \in T^+, f \in F}$ be a solution we want to repair, namely a set of integer ordering quantities. We first evaluate the intermediate variables associated to the solution. Then, we list all constraints that are violated by the solution.

Stage 2: Let $\mathbf{C}_{t,s}$ be a violated constraint relating to period t and entity-item s. Let IT be the item associated to s. Moreover, the only entity that supplies other entities is the warehouse in our case. Consequently, all violated constraints concern the warehouse. We first compute the number of items necessary in the warehouse inventory to respect the constraint. Then, we choose the supplier having the lowest cost for the considered item. Let f be the flow-item with the source entity being the selected supplier, the target entity being the warehouse, and the item being IT. Let LT be the lead time associated to f. If the condition $t - LT > FH$ is verified, then we add the required quantity to the variable $q_{t-LT,f}$. The constraint is next removed from the list of violated constraints. Otherwise, the solution is not modified and stays in the list. We provide a schematic representation of this repair procedure in Figure 3.5. This example applies on an artificial network, for a given item and a given time period, assuming all lead times are set to 0.

Stage 3: Let $\mathbf{C}'_{t,s}$ be a remaining violated constraint relating to period t and entity-item s. Let IT be the item associated to entity-item s. We first compute the quantity of items lacking in the warehouse inventory to respect the constraint. We then list all the stores that are affected by the constraint, namely every store that placed a positive order of item IT to the warehouse at time

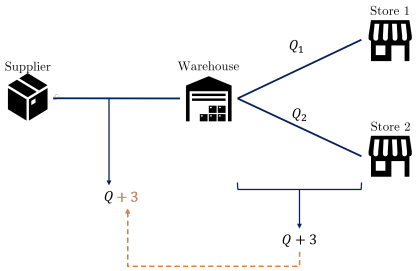


Figure 3.5: Representation of the first repair procedure for a given time period and a given item. All lead times are set to 0 in this example.

t. We then modify the variables of the solution as described in Algorithm 2. We also provide a schematic representation of this repair procedure in Figure 3.6. In this example, we consider an artificial network, a given item and a given time period. We also assume all lead times are set to 0.

At the end of this procedure, we obtain a solution with integer quantities and that respects the constraints. This means that we can use it as an initial solution for the local search.

Algorithm 2 Repair procedure for a remaining violated constraint

Input: List of stores E', Quantity to remove V, Solution S, Item IT, Period t
$v = 0$
while $v < V$ **do**
    Pick one store $e \in E'$
    Find $f \in F$ such that the source entity is the warehouse, the target entity is e and the item is IT
    if $q_{t,f} = 1$ **then**
        $q_{t,f} = 0$
        Remove e from E'
    else
        $q_{t,f} = q_{t,f} - 1$
    end if
    $v = v + 1$
end while

We still need to choose the best way to round the variables. For every variable group, there are three possibilities (round up, round down, or round to the nearest integer). As the initial solution is the starting point of the local search, we want its cost to be the lowest possible. To decide a rounding strategy, we create initial solutions for several instances with every possible combination, perform the repair procedure when needed, and choose the combination that provides the lowest fitness value on the most instances. This combination is the following. Variables of Group 1 are rounded to the nearest integer, variables of Group 2 are rounded down, and variables of Group 3 are rounded up.

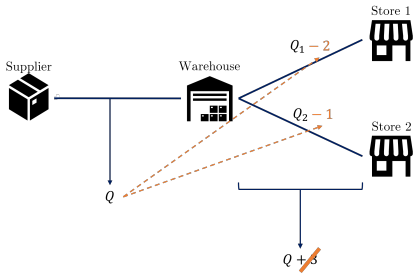


Figure 3.6: Representation of the second repair procedure for a given item and a given time period. We assume all lead times are set to 0, and Store 2 has been picked twice during the repair procedure.

3.3.3 Neighborhood Operators

The search process of a local search consists in moving through the search space by slightly modifying the current solution to find a better one. To achieve this, we need to decide how to modify the solutions.

This modification of the solutions is obtained by the use of *neighborhood operators*. A neighborhood operator is a function u that associates to a solution S a set of solutions $u(S)$. This function modifies one or several variables of the solution to create a new solution that is slightly different and that we call a *neighbor solution*.

In the literature, we can find plenty of neighborhood operators that can be used in a local search. Most of them are specifically designed for given problem classes, like the 2-opt operator for routing problems or the insertion operator for scheduling problems. There also exist neighborhood operators that are designed for solution representations, like the bit-flip operators that applies on solutions that only contains boolean values. Most of the neighborhood operators that we can find in the literature are either designed for routing problems and scheduling problems, or for continuous and boolean solution representations.

To the best of our knowledge, there is no classical neighborhood operator we can use or adapt to our problem or solution format. Moreover, repairing an unfeasible solution demands a significant computation effort, and may turn a modified solution back to the initial one. Hence, we choose to develop neighborhood operators specifically for the problem and solution we study. This allows us not only to work with neighborhood operators that are suited for our case, but also that target precise characteristics of the cost function, and finally ensure that the resulting solution stays feasible. The neighborhood operators we designed are described in the next section.

3.4 Dedicated Neighborhood Operators

In this section, we provide a list of neighborhood operators specifically designed for the optimization problem under study. For every neighborhood operator, we describe the motivations for its construction and how it works. We also provide results on its solving performances by using the following protocol.

We evaluate the performances of the neighborhood operators on the following instances: (`alph`, 15), (`rand`, 15), (`fora`, 15), (`randfora`, 15). We choose the instances that were used in the experimentations in Chapter 2. Since we already performed an exact resolution on these instances, we

can compare the optimal objective value to the results obtained with the local search. For each instance, we run a local search using the selected neighborhood operator with a termination criterion of 1500 evaluations. We then compute a performance indicator: the solution improvement ratio X. This indicator is calculated based on two values: z^0 is the objective value associated to the initial solution of the local search, and z^{LS} is the objective value of the best solution found at the end of the local search. The solution improvement ratio X is computed using equation 3.1. It is used to evaluate the progress achieved through the optimization process compared with the initial solution value. A high solution improvement ratio means that the neighborhood operator is efficient.

$$X = \frac{z^0 - z^{LS}}{z^0} \tag{3.1}$$

For every instance - operator couple, we perform 10 local searches with a random exploration of the neighborhoods, a first-improve selection strategy, and a termination criterion of 1500 evaluations. For every neighborhood operator, we report the values of X or each instance, averaged on all local search runs. In the next subsections, we describe all the neighborhood operators developed for our work and their performances. They are grouped into categories according to the type of variable change they perform, and each category is dedicated to reduce specific costs.

3.4.1 Addition Neighborhood Operators

In this subsection, we present the category of neighborhood operators that add ordering quantities to the solution. These neighborhood operators can target more or less specific variables of the solution, and they aim especially at reducing lost sale costs. There are four neighborhood operators in this category.

Random Addition

This neighborhood operator adds one ordering quantity to a flow-item from supplier to store (variables of Group 1). We refer to it as *Add*. It allows to randomly add ordering quantities without breaking the feasibility of the solution, as the variables of the Group 1 are not constrained. It randomly selects a flow-item f within Group 1, and a time period t. The corresponding variable $q_{f,t}$ is modified as follows: $q_{f,t} = q_{f,t} + 1$. We provide the results of the performance experiments in Table 3.1.

Guided Addition

This neighborhood operator, like the previous one, adds ordering quantities to the flow-items from suppliers to stores (variables of Group 1), which ensures the solution stays in the feasible space. However, it does not add ordering quantities randomly, but targets the entity-items that have a high lost sale level. We refer to this neighborhood operator as *AddS*.

First, the lost sales level variables are arranged in a descending order. To keep randomness in the neighborhood exploration, we select one variable randomly among the top-k, with k being an integer. For the experimentations, we set k to 10. Let $y_{s,t}$ be the variable selected, with s being the corresponding entity-item, and t the corresponding time period. Let V be the value of the variable. We then list all the flow-items so that the source entity is not the warehouse, the target entity is EN_s, and the item is IT_s. Let f be the flow-item of the list that has the lowest transport cost. If

$t - LT_f > FH$, then the corresponding variable $q_{t-LT_f,f}$ is modified as described in equation 3.2. Otherwise, the neighbor is rejected and deleted from the candidate neighbors list.

$$q_{t-LT_f,f} = q_{t-LT_f,f} + \lceil V \rceil \tag{3.2}$$

With *AddS*, we aim at specifically targeting the orders that result in a high lost sale level. Its performances are provided in Table 3.1.

Guided Addition with Repair

This neighborhood operator adds quantities to the flow-items from warehouse to stores (variables of Group 3), that corresponds to entity-items with a high lost sale level. This is the same idea as the previous neighborhood operator, but in this case we target variables that are constrained. As a result, in order to ensure that the solution stays feasible, we have to develop a repair procedure. We refer to this neighborhood operator as *AddSR*.

We first arrange the lost sale variables by descending order, and select one randomly among the top-k. For our experiments, we set $k = 10$. Let $y_{s,t}$ be the variable selected, with s being the corresponding entity-item and t the corresponding time period. Let V be the value of this variable. We then select the flow-item f such that the source entity is the warehouse, the target entity is EN_s and the item is IT_s. If $t - LT_F < FH$, then the neighbor is rejected and deleted from the candidate neighbors list. Otherwise, the corresponding variable $q_{t-LT_f,f}$ is modified as in equation 3.2. This modification can create an unfeasibility, as now the warehouse may send more quantities than it has in its inventory. Hence, we have to enter the repair phase. To do so, we first list all the flow-items so that the source entity is a supplier, the target entity is the warehouse, and the item is IT_s. We select the flow-item f' with the lowest transport cost among this list. Let $t' = t - LT_f$. If $t' - LT_{f'} < FH$, then the neighbor is rejected and deleted from the candidate neighbor list. Otherwise, we modify the corresponding variable $q_{t'-LT_{f'},f'}$ as described in equation 3.3.

$$q_{t'-LT_{f'}} = q_{t'-LT_{f'}} + \lceil V \rceil \tag{3.3}$$

This neighborhood operator, like the previous one, aims at reducing the lost sale costs, while specifically targeting the flow-items that lead to high lost sales. However, in this case, we try to impact the flow-items that pass through the warehouse, rather than placing orders to the warehouse directly. We provide the performance results of *AddSR* in Table 3.1.

Extension of Guided Addition with Repair

This operator is a combination of *AddS* and *AddSR*. The mechanic is very simple. We first arrange the lost sale variables by descending order, and select one variable among the top-k, with $k = 10$ for the experiments. Based on this selected variable, we try to apply *AddSR*. If the solution is rejected, we then apply*AddS*. If the solution is rejected, we delete the neighbor from the candidate neighbors list. With this neighborhood operator, we still target the lost sale costs, but we try to benefit from both supplying source for the stores. We prefer to make items pass through the warehouse, because the transport costs are lower, but we still try to order to the supplier if the lead times are too long. We refer to this neighborhood operator as *AddE*, and provide its performance results in Table 3.1.

Performances

Let us summarize the performances of addition neighborhood operators. The results are provided in Table 3.1. First, we can note that the performances are significantly variable depending on the instance. The operators seem to be more efficient on the `fora` and the `randfora` instances. This can be explained by the fact that these instances generation modes result in item groups with a high demand. As a result, lost sale costs may be high on such instances if the initial solution is not of good quality. Then, by comparing the neighborhood operator, we can note that *AddE* is the most efficient. This means that guiding the neighborhood operator and trying several options offers a significant increase in the performance for this operator category.

Table 3.1: Performances of addition neighborhood operators averaged on the 10 runs for every instance. Higher values are better.

Instance	*Add*	*AddS*	*AddSR*	*AddE*
(`alph`, 15)	0.007	0.025	0.014	0.029
(`fora`, 15)	0.048	0.050	0.088	0.101
(`rand`, 15)	0.013	0.013	0.000	0.012
(`randfora`, 15)	0.066	0.063	0.112	0.130
Average	0.034	0.038	0.054	0.068

3.4.2 Deletion Neighborhood Operators

In this section, we describe neighborhood operators that remove quantities from the orders. Just like the neighborhood operators described in Section 3.4.1, they can be more or less guided. Their main goal is to reduce transport costs and storage costs. We also describe neighborhood operators that "destroy" the orders that are placed in the solution. In contrast to the first neighborhood operators described, that remove a given quantity from the variables (*i.e.* $q \leftarrow q - 1$), these neighborhood operators remove the entire order for the selected variables (*i.e.* $q \leftarrow 0$). They can be applied on variable groups of varying sizes, and aim at reducing the transport costs and the fixed costs. There are four neighborhood operators in this category

Random Deletion

With this neighborhood operator, we target the quantities transiting from the supplier to the store, or from the warehouse to the store (variables from Group 1 and Group 3). We randomly select a strictly positive variable from one of the groups. Let f be the corresponding flow-item, and t the corresponding time period. The variable is then modified as follows: $q_{f,t} = q_{f,t} - 1$. The idea is to reduce unnecessary transport and storage costs while keeping the solution feasible. We refer to this neighborhood operator as *Del*, and provide its performance results in Table 3.2.

Deletion Supplier - Warehouse

With this neighborhood operator, the idea is to remove a quantity on a flow-item from the supplier to the warehouse (variables of Group 2). We randomly select a positive variable from this group,

and remove one quantity from the order. However, this can make the solution unfeasible, as the warehouse may not have enough inventory now to provide the planned quantities sent to the stores. In such cases, the neighbor is rejected and deleted from the candidate neighbors list. We refer to this neighborhood operator as *DelSW*. *DelSW* aims at reducing unnecessary transport and storage costs, without penalizing the corresponding flow-items from the warehouse to the stores. We provide the performance results of *DelSW* in Table 3.2.

Localized Destruction

This neighborhood operator targets flow-items from suppliers to stores or from the warehouse to the stores (variables from Group 1 and Group 3). We refer to it as the *Des* neighborhood operator. It randomly selects a variable from one of the groups and set its value to 0. *Des* aims at reducing unnecessary transport cost while keeping the solution feasible. We provide the performance results of *Des* in Table 3.2.

Flow Destruction

This neighborhood operator can be seen as an extended version of *Des*, as it works the same way but applies on more variables. We refer to it as *DesF*. Instead of selecting one flow-item variable, *DesF* randomly selects one flow originating from a supplier or from the warehouse and targeting a store, and a time period. Let l be this flow, and t the selected time period. Then, for all flow-item f such that $SE_f = SF_l$ and $TE_f = TF_l$, the ordering quantity $q_{t,f}$ are set to 0. *DesF* aims at reducing unnecessary transport costs, but also fixed ordering costs, as it removes an entire order from the solution. The performances of *DesF* are provided in Table 3.2.

Performances

The performances of deletion neighborhood operators are presented in Table 3.2. First, let us focus on the individual deletion neighborhood operators. The results are rather straightforward, as both *Del* and *DelSW* do not improve the initial solutions of any instance. Therefore, these neighborhood operators are not relevant for our problem. For the destruction operators however, we can note improvements of the initial solution. *DesF* seem to be the most efficient, and it is better suited for `alph` and `rand` instances. These generation modes result in instances with low demand, and we can assume that a non necessary ordered quantity has more impact on the solution total cost in this case.

Table 3.2: Performances deletion neighborhood operators averaged on the 10 runs for every instance.

Instance	*Del*	*DelSW*	*Des*	*DesF*
(`alph`, 15)	0.000	0.000	0.008	0.115
(`fora`, 15)	0.000	0.000	0.085	0.106
(`rand`, 15)	0.000	0.000	0.176	0.217
(`randfora`, 15)	0.000	0.000	0.019	0.030
Average	0.000	0.000	0.072	0.117

3.4.3 Rescheduling Neighborhood Operators

In this category, we describe neighborhood operators that move orders to a different time period. They can move the orders one period before, or one period later. By grouping orders together, they aim at reducing fixed ordering costs, Franco costs, and unnecessary storage costs in case of order postponing. There are five neighborhood operators in this category.

Localized Preponing

This neighborhood operator targets flow-items that originate from the warehouse and go to either the warehouse or the stores (variables of Group 1 and Group 2). The idea is to group orders while keeping the solution feasible. It randomly select one flow-item f in this group, and a time-period $t \geqslant FH+1$. Then, the ordering quantity $q_{t-1,f} = q_{t-1,f} + q_{t,f}$, and the ordering quantity $q_{t,f} = 0$. We refer to this neighborhood operators as *Pre*, and provide its performances in Table 3.3.

Flow Preponing

This neighborhood operator is an extension as the previous one, as it moves ordering quantities one period earlier, but applies to all flow-items of one flow, rather than to one flow-item only. It targets flows that originate from the supplier and go to either the warehouse or the stores (variables of Group 1 and Group 2). It randomly selects a flow l from one of both groups and a time period $t \leqslant FH+1$. Then, for all flow-items f such that $SE_f = SF_l$ and $TE_f = TF_l$, the ordering quantities are modified as follows: $q_{t-1} = q_{t-1,f} + q_{t,f}$ and $q_{t,f} = 0$. We refer to this neighborhood operator as *PreF*, and provide its performances in Table 3.3.

Period Preponing

This neighborhood operator is an extension of the previous one, as it applies to an even wider range of variables. This time, we select a time period $t \geqslant FH+1$. For every flow-item f, the corresponding variable $q_{t,f}$ is reported one time period before. We refer to this neighborhood operator as *PreP*. Like both previous neighborhood operators, it aims at reducing the fixed ordering costs and the Franco costs by grouping orders. But in addition, it also has a "perturbation" role, as it modifies a large number of variables. As a result, the resulting solution may be more different from the initial one than with other neighborhood operators. The performances of *PreP*are provided in Table 3.3.

Localized Postponing

This neighborhood operator targets the flow-items from the warehouse or the supplier to the stores (variables of Group 1 and Group 3). The idea is to group orders while keeping the feasibility of the solution. We refer it as *Pos*. It randomly selects a flow-item f from one of these groups, and a time-period $t \leqslant FH + H - 1$. The corresponding variables are modified as follows: $q_{t+1,f} = q_{t+1,f} + q_{t,f}$ and $q_{t,f} = 0$. The performances of *Pos* are provided in Table 3.3.

Flow Postponing

This neighborhood operator is an extension of the previous one, as it applies on a group of variables. It still applies on the flow-items from a supplier or the warehouse to a store (variables from Group 1 and Group 3, but this time we modify all flow-items of a flow. We randomly select a flow l and a

time period $t \leqslant FH + H - 1$. Then, for all flow-item f such that $SE_f = SF_l$ and $TE_f = TF_l$, the corresponding variable $q_{t,f}$ is reported to the next period. We refer to this neighborhood operator as *PosF*, and provide its performances in Table 3.3.

Performances

In Table 3.3, we provide the performances of rescheduling neighborhood operators. Let us first compare preponing operators and postponing operators. We can note that preponing operators have a significantly higher performance than postponing operators. We can conclude that, for our problem, it is more interesting to move orders earlier if we want to reschedule them. If we focus on preponing neighborhood operators, we can note that they are more efficient on `fora` and `randfora` instances. Moreover, *PreF* and *PreP* seem to be more efficient than *Pre*. As a matter of fact, *Pre* concerns individual orders, and therefore only impacts the storage costs and Franco costs. As *PreF* and *PreP*concern all the orders of one or several flows, they also help reduce fixed ordering costs.

Table 3.3: Performances of rescheduling neighborhood operators averaged on the 10 runs for every instance.

Instance	*Pre*	*PreF*	*PreP*	*Pos*	*PosF*
(`alph`, 15)	0.000	0.000	0.000	0.000	0.000
(`fora`, 15)	0.036	0.040	0.037	0.010	0.010
(`rand`, 15)	0.022	0.019	0.012	0.015	0.015
(`randfora`, 15)	0.035	0.039	0.039	0.005	0.005
Average	0.023	0.024	0.022	0.007	0.007

3.4.4 Saturate Franco Neighborhood Operator

This category relates to the neighborhood operator designed to reduce Franco costs specifically. We developed a single neighborhood operator in this category, that we refer to as *Sat*. It works as follows. The first step is to gather all flows that have least one strictly positive order and that do not reach the Franco threshold. Then, we randomly select on flow from this group. Let l be this flow, and t the corresponding time period. We next randomly pick out a flow-item f such that $SE_f = SF_l$ and $TE_f = TF_l$. Finally, the ordering quantity $q_{t,f}$ is increased of the exact value needed to reach the Franco threshold.

The performances of *Sat* are provided in Table 3.4. This operator seem to be efficient only for `fora` and `rand`. We can assume that both other instances either already reach Franco thresholds for all orders, or that reaching the thresholds require to order too much compared to the demand.

3.4.5 Overview

Let us summarize the performances of all neighborhood operators. In Table 3.5, we provide an overview of the performances of all neighborhood operators averaged on all instances. We provide the results for X, and add another performance indicator: the relative distance to the optimal value X^*. This operator is the counterpart of X. It allows us to evaluate how much there is still to

Table 3.4: Performances of neighborhood operator *Sat* averaged on the 10 runs for every instance.

Instance	*Sat*
(`alph`, 15)	0.000
(`fora`, 15)	0.028
(`rand`, 15)	0.025
(`randfora`, 15)	0.000
Average	0.013

optimize after the local search to reach the optimal. This indicator is calculated using the formula in equation 3.4, with z^{LS} being the cost associated to the best solution found by the local search, and z^* being the cost of the optimal solution.

$$X^* = \frac{z^{LS} - z^*}{z^*} \tag{3.4}$$

The best value of X^* we obtain is 1.283, which means we still are more than 100% away from the optimal solution. This result is not very good, and we expect to find a solution with a lower cost.

As a result, we cannot consider a classic local search with a single neighborhood operator as a definitive solution to the problem under study. However, we take advantage of the work presented in this Section to develop a more sophisticated metaheuristic, still using the search principle of a local search. We detail the motivations for this metaheuristic and its exact functioning in the next sections.

3.5 Use of Multiple Neighborhood Operators in the Search

As explained in the previous Section, the operators we developed are not very effective individually, and are not able to find a solution of good enough quality. As each one of them was developed to optimize one aspect of the cost function, we want to explore the possibility of using several operators. This can be achieved by using a Variable Neighborhood Search (*VNS*) inspired algorithm. A Variable Neighborhood Search is a metaheuristic introduced by Mladenović and Hansen [1997]. Its basic idea is to systematically change the neighborhood both within a search phase to find a local optimum or in a perturbation phase to escape a local optimum (Hansen et al. [2019]). In an inventory optimization context, VNS are mainly used for inventory routing problems (Gruler et al. [2018], Popović et al. [2012], Mjirda et al. [2012]). The main search process relies on the sequential use of different local searches with different neighborhood operators or perturbation operators to escape local optima and find better quality solutions. In this context, we aim to develop a VNS-inspired algorithm that can apply several of the operators we described in Section 3.4. We provide the pseudo-code for such an algorithm in Algorithm 3. The main idea is, given a set of operators and an initial solution, to perform a local search with the first operator of the set, replace the initial solution with the best solution found by the local search, and repeat this process for every operator of the set.

Table 3.5: Performances of all neighborhood operators averaged on all instances and arranged by ascending order of X^*.

Operator	X	X^*
DesF	0.117	1.283
Des	0.072	1.402
AddE	0.068	1.465
AddSR	0.054	1.502
AddS	0.038	1.533
Add	0.034	1.543
PreF	0.024	1.562
Pre	0.023	1.563
PreP	0.022	1.57
Sat	0.013	1.585
PosF	0.012	1.592
Pos	0.007	1.602
Del	0	1.623
DelSW	0	1.623

Algorithm 3 VNS-inspired algorithm

1: **Input:** Initial solution S, Set of operators U, Termination criterion TC
2: **for** $u \in U$ **do**
3: $S' \leftarrow LocalSearch\,(S, u, TC)$
4: $S \leftarrow S'$
5: **end for**
6: **Return:** S

For a first experiment with this algorithm, we try to compare the convergence efficiency of the VNS-inspired method with the convergence performance of the best operator from the list described in Section 3.4, which is *DesF*. To do so, we select the following subset of operators: $U = \{DesF, AddE, Add, PreF, PreP, Sat\}$, arranged by decreasing order of the performances obtained in the last section. We set the termination criterion to 250 evaluations, so that the VNS-inspired method has the same total number of evaluations than the local search with a single operator. In Figure 3.7, we provide an example of two convergence curves to illustrate this first test. On the left, there is the convergence curve of a classic local search with *DesF*, and a termination of 1500 evaluations. On the right, there is the complete convergence curve of the VNS-inspired algorithm we described above. We can see a clear improvement in the convergence curve of the VNS-inspired algorithm, as its final solution is significantly closer from the optimal value. This validates the benefit of using several neighborhood operators in the search process.

However, there are still questions we have to tackle, especially for the design of the algorithm. The main concern here is the decision of the order in which the operators are given to the local search. To illustrate the impact of this order, we provide two example convergence curves in Fig-

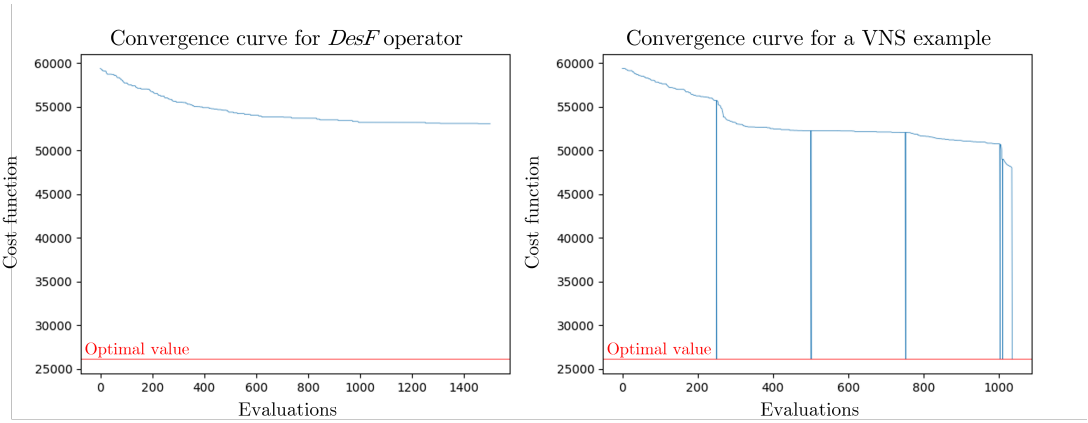


Figure 3.7: Convergence curves for both algorithms: individual Local Search with *DesF* on the left, VNS-inspired algorithm on the right. For the right curve, a vertical bar represent a modification of neighborhood operator.

ure 3.8. On the left, we reproduced the convergence curve of the VNS-inspired algorithm provided in Figure 3.7. In this case, the neighborhood operators are used in the decreasing order of their performances. On the right, we provide the convergence curve of the exact same algorithm, but with the neighborhood operators used in the increasing order of their performances. By comparing these curves, we can notice that both solutions obtained by the algorithm are significantly different in terms of cost function. We can deduce that the order in which we use the neighborhood operators has an impact on the global performance of the algorithms.

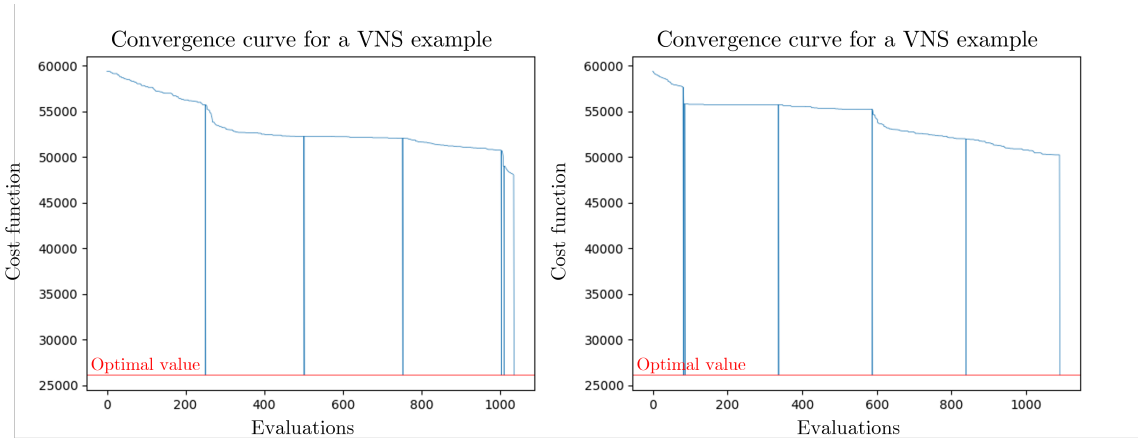


Figure 3.8: Convergence curves for both algorithms: VNS-inspired algorithm with neighborhood operators in decreasing performance order on the left, VNS-inspired algorithm with neighborhood operators in increasing performance order on the right. Vertical bars represent modifications of neighborhood operator.

Overall, the results of this experiment shows that the operators can be more or less efficient in different phases of the search, and can be more or less adapted to certain solution types. With this observation in mind, we are interested in developing a variable local search algorithm that

could take such specificities into account and use them to guide the search process. The idea is to select operators based on their recent efficiency in the search process, and to develop an adaptive way of selecting the neighborhood operators. This reflection lead to the construction of a new metaheuristic, that we describe in the next section.

3.6 Variable Adaptive Local Search

3.6.1 Overview of the Algorithm

In the last section, we emphasize the advantages of using multiple neighborhood operators and note that the effectiveness of these operators varies depending on the type of solution they are applied to. This leads to the development of a new metaheuristic, still based on the local search principle, but with a more elaborate way of selecting and using the neighborhood operators. The idea is to associate an efficiency score to every neighborhood operator, and to use this efficiency score to influence the selection of the neighborhood operators. Once a neighborhood operator is selected, we apply it until a better solution is found, or a termination criterion is reached, and use its immediate performance to make the neighborhood operator's score evolve. We call this algorithm *Variable Adaptive Local Search* (*VALS*). We provide an overview of the global search process of *VALS* in Algorithm 4. We also provide a schematic representation of the algorithm execution in Figure 3.9.

Algorithm 4 Variable Adaptive Local Search

```
1: Input: Set of neighborhood operators U, Scores of neighborhood operators G = {G_u}_{u∈U},
   Global termination criterion TC, Local termination criterion tc, Initial solution S
2: Compute selection probability distribution
3: while not TC do
4:     Select a neighborhood operator u ∈ U following the selection probabilities
5:     while not tc do
6:         Generate a candidate neighbor S'
7:         if S' is better than S then
8:             S ← S'
9:             tc ← True
10:        end if
11:    end while
12:    Compute immediate efficiency of u
13:    Update G_u
14:    Update selection probabilities
15: end while
```

This metaheuristic is inspired by the protocol for using different neighborhood operators, compute their score and making this score evolve throughout the search described in Maturana et al. [2009]. In opposition to the VNS-inspired algorithm we proposed in Section 3.5, we select a neighborhood operator as soon as a better solution is found, and do not continue the search until the neighborhood operator finds a local optimum. This allows a more dynamic exploration of the search space, as we spend less time exploring a potential local optimum. Moreover, if a neighborhood operator has a high efficiency score, its selection probability is accordingly higher, and can be selected several times in a row. This allow the algorithm to exploit the neighborhood as long as the operator

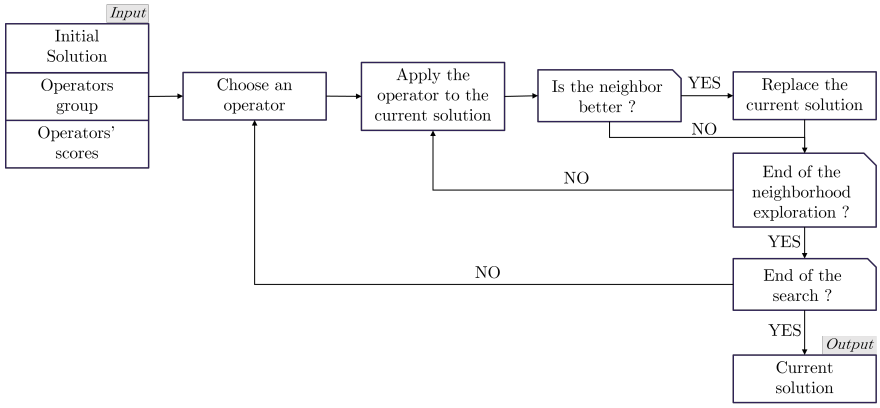


Figure 3.9: Schematic representation of *VALS* execution.

is relevant for the current solution. In order to implement *VALS*, we also have to make the following algorithm design decisions.

Score Calculation: As we mentioned above, all neighborhood operators are associated with an efficiency score. This score is very important for the algorithm, as it represents the current relevance of a neighborhood operator, and it also determines the probability of a neighborhood operator to be selected in the search process. The initialization and calculation of the scores are detailed Section 3.6.2.

Neighborhood Operator Selection: Once the scores of the neighborhood operators are computed, we need to determine how to select the neighborhood operators at the end of every neighborhood exploration. This procedure is described Section 3.6.3.

Score Evolution: Every time we select a neighborhood operator, the score associated to the neighborhood operator we have just used is updated based on its immediate performance. This process ensures that the probability of selecting a neighborhood operator is representative of the relevancy of the neighborhood operator for the current solution. This update process is detailed Section 3.6.4.

Termination Criteria: There are two termination criteria that have to be determined for the design of *VALS*. The first one is the global termination criterion, which states when the search process finishes. It has the same function as a termination criterion in a classic local search, and decides when the solution found is satisfactory enough to stop the algorithm. The second one is the local termination criterion, that is applied on the individual neighborhood searches. This local termination criterion ensures we do not spend too much search effort on a non promising neighborhood. Both of these criteria are described Section 3.6.5.

3.6.2 Score Initialization and Computation

In this section, we focus on the method for calculating the efficiency scores associated to every neighborhood operator used by *VALS*. These scores are used as a basis to choose the neighborhood operator each time we change neighborhood operators.

The score has a function of representing the efficiency of an operator at the end of the neighborhood exploration. To translate it, we balance two aspects of the neighborhood exploration. The first aspect is the gap between the initial solution, that is the input of the neighborhood search, and the final solution, that is found at the end of the neighborhood exploration. The second aspect is the time needed to achieve this gap. As the neighborhood exploration stops once we find the first improving neighbor, a short neighborhood exploration implies high efficiency, because this means the neighborhood operator is able to quickly find improving neighbors. As the execution time measure is dependent on various factors, such as the machine used to execute the algorithm or the current occupation of this machine for example, we cannot consider this measure as objective enough to use it as is. Consequently, to translate the time spent exploring a neighborhood, we use the number of neighbors evaluated during the exploration. Let us consider a single neighborhood exploration with a neighborhood operator u. With z^0 being the fitness value associated with the initial solution, z^* the fitness value of the final solution, and N_S the number of neighbors evaluated during the search, the score of the operator is computed using the formula of equation 3.5. This formula is used to compute the so-called "immediate" efficiency score of the operator. The process to compute the final score associated to the neighborhood operator before choosing a new one is further described in Section 3.6.4.

$$G = \frac{z^0 - z^*}{N_S} \tag{3.5}$$

As stated before, we calculate this score at the end of every neighborhood exploration. However, we also have to provide initial efficiency scores for the neighborhood operators as input for the algorithm. To do so, we choose to perform an individual local search for every neighborhood operator. This individual local search is applied to the initial solution computed for *VALS*, and has a termination criterion of L neighbor evaluations. This criterion L is a hyper-parameter of the algorithm. At the end of every individual local search, we compute the score associated to the operator using the same formula. At the end of every operator score initialization, we keep the best solution found as the initial one for the algorithm.

3.6.3 Neighborhood Operator Selection

In this section, we describe the selection mechanism for the neighborhood operators. This mechanism is used at the beginning of the search process, using the initial scores associated to the neighborhood operators, and at the end of every neighborhood exploration, to select the next neighborhood operator to be used in the search process.

The process we develop is very simple. Let U be the set of neighborhood operators used in the search, and G_u the score associated with the neighborhood operator u. Based on the set of scores, we compute the selection probabilities with the formula in equation 3.6. This is the most straightforward way of computing probabilities.

$$\forall u \in U, \qquad \mathbf{P}_u = \frac{G_u}{\sum_{u \in U} G_u} \tag{3.6}$$

With the calculation of probabilities for every neighborhood operator, we obtain a probability distribution for the selection of the neighborhood operators. Every time we have to select a neighborhood operator, we randomly draw a neighborhood operator according to the probability law described by $\mathbf{P}$.

3.6.4 Score Evolution

In Section 3.6.2, we describe how the performance of the operators is assessed after every neighborhood exploration. The formula we provide is used to give a representation of the immediate performance of the neighborhood operator, but we do not use this value as is to compute the new score that is used for the selection probability. Let G be the immediate score computed with the formula described in equation 3.5. This score is the most up-to-date, but it is also the result of a neighborhood exploration that selects candidate solutions randomly. As a result, this measure may not be completely representative of the neighborhood operator's performance, because it may be the result of an arbitrarily advantageous (or disadvantageous) exploration. Moreover, basing the selection probability of a neighborhood operator on a single exploration may not be relevant, as it can ignore the information obtained with the previous explorations. Nevertheless, we do not want to use the complete history of scores, as we want the method to stay adaptive.

Consequently, we want to find a way of making the scores evolve that is not too fluctuating (*i.e.* using the immediate score as is), but not too stagnating (*i.e.* using the complete history of immediate scores). To do so, we use the concept of *sliding window*, as described in Veerapen et al. [2012]. The idea is to keep a given number of immediate scores to compute the resulting score, and to make this reduced history evolve by adding the last immediate score to the list and remove the oldest one. The sliding window process operates as follows. Let K be the size of the sliding window, that represents the number of scores we keep in the history. We first fill every sliding window of the neighborhood operators with their initial performance score. We consider a neighborhood operator u, and we note its sliding window SW^u. We place ourselves at the n-th time we explore a neighborhood with u. Let $G_u(n)$ be the immediate performance score associated to this exploration. The new score associated to the operator for the selection probability is calculated using the equation 3.7.

$$G'_u = \frac{1}{K+1}\left(G_u(n) + \sum_{i=1}^{K} SW^u(i)\right) \tag{3.7}$$

We then make the sliding window evolve in the following way. If $n < K$, then $SW^u(n) = G_u(n)$. Otherwise, we remove $SW^u(0)$, and $SW^u(n) = G_u(K)$. We provide a schematic representation of the process of updating the sliding window in Figure 3.10.

Using this mechanism, we obtain a score that takes into account the recent performance of the operator without being too sensitive to the last exploration. The higher the K, the more inertia the score evolution has (*i.e.* the score evolution is slower). On the contrary, the lower the K is, the more dynamic the score evolution is (*i.e.* the immediate score has more impact). The size of the sliding window K is a hyper-parameter of the algorithm.

3.6.5 Termination criteria

As we mentioned above, there are two types of termination criteria we have to define for the design of *VALS*. First, there is the global termination criterion, that decides when the algorithm stops.

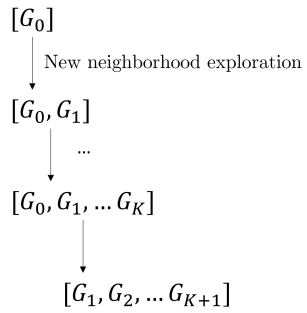


Figure 3.10: Schematic representation of the evolution of the sliding window through the execution of *VALS*.

This criterion is denoted TC. Then, there is the local termination criterion, that decides when to stop a neighborhood exploration when we did not find an improving neighbor for a certain time. This criterion is denoted tc.

The global termination criterion determines the "budget" we allow the algorithm to find a solution we consider satisfying. As we mentioned in Section 3.6.2, we do not consider the execution time as an objective measure, as it varies with the machine used. Consequently, we prefer to use the number of neighbors evaluated as a termination criterion, as it is a more objective measure of the exploration of the search space. As a result, we prefer to express TC in neighbor evaluations. Once the maximum number of evaluations is reached, the algorithm terminates its current neighborhood exploration, and returns the best solution found. However, for some experimentations, we consider an execution time limit more suited, as it allows to better control the total execution time of the experimentations. Moreover, one of the industrial requirements for the resolution method is a time limit of two hours, and we want to ensure that we find a reasonably good quality solution within this time span.

The local termination criterion determines the maximum budget allowed for the exploitation of a given neighborhood. Once a neighborhood operator is selected, the exploration of this neighborhood will continue until an improving neighbor is found, or in default until the local termination criterion is reached. This criterion is still a number of neighbors evaluated, for the same reasons as stated above, but we choose to not directly set a value. As a matter of fact, we want the exploration to stop once we evaluated a certain proportion of the neighborhood defined by the couple of the initial solution and the neighborhood operator. Once we selected a neighborhood operator, let $|u|$ be the size of the corresponding neighborhood. The number of evaluations for the local criterion tc is computed as follows: $tc = \lceil \gamma \times |u| \rceil$ evaluations allowed, with $0 < \gamma \leqslant 1$, representing the neighborhood proportion we allow the neighborhood operator to explore. If $\gamma = 1$, then we allow the algorithm to eventually explore the complete neighborhood without finding an improving solution. In such a case, the algorithm may spend a lot of its complete budget in the exploration of a not relevant neighborhood. On the contrary, the closer to 0 γ is, the least number of neighbors we allow the neighborhood operator to explore to find an improving solution. This ensures a lower probability of spending time in a not relevant neighborhood, but also increases the probability to miss a good solution if the limit is too low.

3.7 Experimentations

In this section, we present the experimentations conducted with *VALS* and the results obtained. We first validate the benefits of the method by comparing it with several variable methods in Section 3.7.1. Then, we focus on the global performances of the algorithms by applying it on several instances of the problem under study in Section 3.7.2.

Overall, the following design decisions have to be made in order to set up the *VALS* algorithm. The first decision is the set of neighborhood operators that will be used for the search. Based on the individual performances of the neighborhood operators, we select the following operator set: {*DesF*, *AddE*, *Add*, *PreF*, *PreP*, *Sat*}. Next, we have to set the hyper-parameters we described in the previous sections. They are: the number of neighbors evaluated to compute the initial scores of the neighborhood operators L, the size of the sliding window K, the local termination criterion tc and the global termination criterion TC. In the following experiments, we use the following hyper-parameter values for our experiments. We set $L = 75$ neighbor evaluations for the initialization of scores, $K = 15$ for the sliding window size, and $tc = 0.1$ for the proportion of the neighborhood we allow the neighborhood operators to explore without finding an improving solution. Regarding the total budget TC, it will vary depending on the experimentations.

3.7.1 Method Validation

In this Section, we aim at validating the metaheuristic design. We already gave insights on the benefit of using several neighborhood operators in Section 3.5. In this part, we propose to validate the two other aspects of *VALS*. The first aspect is the scoring of neighborhood operators, that results in a non-uniform probability distribution for the selection of the neighborhood operators along the search. By arbitrarily setting a uniform probability distribution for the neighborhood operators selection, *i.e.* $\mathbf{P}_u = \frac{1}{|U|}\ \forall u \in U$, we give all neighborhood operators the same chance to be used at each step of the search. In this case, the algorithm operates the exact same way as *VALS*, without the score calculation, the score evolution, and the probability evolution processes. In the following, we refer to this alternative algorithm as *Uniform Variable Local Search* (*U-VLS*). The second aspect is the adaptive function of the algorithm, that makes the selection probability evolve along the search. By initializing a probability distribution for the neighborhood operator selection based on their initial performance score, we use the fact that all neighborhood operators do not have the same performance, but assume this performance remains stable along the optimization. In this case, the algorithm operates exactly like *VALS*, but without the score evolution process. In the following, we refer to this algorithm as *Variable Local Search* (*VLS*). The pseudo-code for both alternative algorithms, *U-VLS* and *VLS*, are in Appendix B.

In order to validate both the non uniform distribution for selecting neighborhood operators, and the adaptive aspect of the algorithm, we compare the performances of our proposed solution method, *VALS*, with both algorithms we described above: *U-VLS* and *VLS*. To this end, we select a subset of instances for the problem under study (*cf.* Section 2.4), and apply the three algorithms to solve them. As the algorithms rely on several random processes (*e.g.* neighborhood operator selection, neighborhood exploration), we run each algorithm twenty times on the instances, and set an input seed for every algorithm run. To ensure a fair comparison between the algorithms, we set a common global termination criterion TC. The exact experimental protocol definition is as follows.

- Instances: All pairs of generation modes {`alph`, `for`, `fora`, `rand`} and number of items

$\{15, 25, 50, 100\}$, for a total of 16 instances.

- Number of seeds: 20
- Global termination criterion: 7200 seconds

To summarize, we run every algorithm twenty times on every instance with a fixed seed and a termination criterion of 7200 seconds, which is the time limit set by the partner company. We then compare their performances. As we do not have the optimal value for all instances of the set, we evaluate the algorithm performance by calculating their relative gap from the initial solution. The relative gap is computed the same way as described in Section 3.4. As a reminder, we recall the formula of Section 3.4 in equation 3.1, with z^0 being the cost associated to the initial solution, and z^{LS} the cost associated to the best solution found by the algorithms. This performance measure allows to assess how much the algorithm can improve the initial solution, and has the advantage to be comparable for instances with different initial solution values.

$$X = \frac{z^0 - z^{LS}}{z^0} \tag{3.1}$$

In Figure 3.11, we provide the results of the experimentations described above. These results are shown in form of boxplots. For each algorithm, we produce a boxplot for all relative gaps obtained with a single run of the algorithm. These results are aggregated for all instances of the experimental protocol. If we focus on the boxplots, we can see that the *VALS* algorithm globally obtains better results than both other algorithms. As a matter of fact, the median of its boxplot is higher than both other medians. Moreover, this algorithm is the one that achieves the highest performance, and its lowest value is also higher than both the lowest values of *U-VLS* and *VLS*.

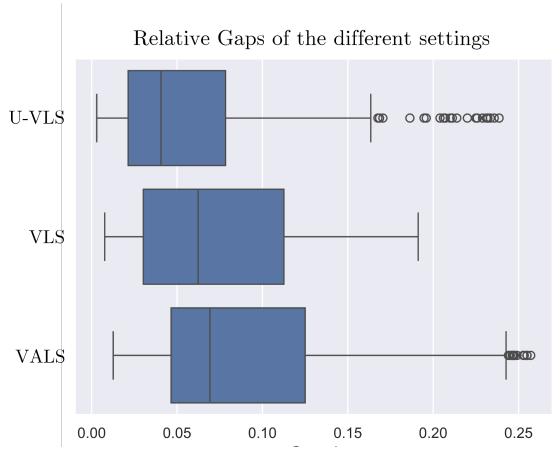


Figure 3.11: Boxplots for the relative gaps results of the comparison of the algorithms *U-VLS*, *VLS*and *VALS*, aggregated in all instances.

Overall, we can deduce that *VALS* is the algorithm that has a better chance to find the best solution among the three alternatives we consider. Consequently, we can validate the benefit of both the non-uniform distribution for the selection probability of the neighborhood operators, and the evolution of this distribution score.

3.7.2 Global Performances

In this section, we focus on the global performances of our proposed algorithm for the problem under study. To this end, we put in place the same experimental protocol, but with other parameters. The idea is to test fewer instances, but to give the algorithm a higher budget, so that it has more time to optimize the instances. Moreover, we want to give a budget in neighbor evaluations and not in execution time, so that the algorithm can have the same optimizing chance for all instances. As a matter of fact, the algorithm is slower with the increase in the instance size, as it takes more time to evaluate the cost function, so it can explore a lower number of neighbors for larger instances in the same amount of time. The exact experimental protocol definition is as follows.

- Instances: All pairs of generation modes $\{\texttt{fora}, \texttt{randfor}, \texttt{rand}\}$ and number of items $\{15, 25, 40\}$, for a total of 9 instances.
- Number of seeds: 10
- Global termination criterion: 3000 neighbor evaluations

To assess the performance of *VALS*, we compute the average relative gap from the initial solution. The formula to calculate this performance measure is provided in equation 3.8, with $\overline{z^0}$ being the average of initial solution values on all seeds, and $\overline{z^{LS}}$ the average of the best solution found values on all seeds.

$$\overline{X_1} = \frac{\overline{z^0} - \overline{z^{LS}}}{\overline{z^0}} \tag{3.8}$$

We provide the results of this experimentation in Table 3.6. We can see that, except for the `rand` instances, the algorithm achieves at least a 10% improvement of the initial solution, even for the larger instances, that are more difficult to solve. For the instance $(\texttt{fora}, 15)$, we reach an improvement of 25%, which is more than the double of the performance we could achieve with the best neighborhood operator in a simple local search (*cf.* Section 3.4). As a result, we can see a clear improvement in the resolution performances with our proposed algorithm. In Appendix C, we provide the convergence curves obtained with this experimentation phase. For instances for which we have the optimal objective value, this value is shown on the graphs.

Table 3.6: Average gaps from the initial solution using *VALS*.

Instance	`fora`	`randfor`	`rand`
15	0.25	0.21	0.09
25	0.14	0.15	0.11
40	0.11	0.10	0.05

3.8 Conclusion

In this Chapter, we propose a resolution method to solve the problem we aim to tackle in this thesis. Based on the characteristics of the problem, the specificities of the model, and the industrial

requirements we have to meet, we orient the algorithm design to a local search-like algorithm. As we do not find existing neighborhood operators that are suited for the problem under study, we develop neighborhood operators dedicated to the problem, and conduct first experimentations to assess their performance.

From this point, we note that the individual performance of the neighborhood operators is not satisfying, but also that we can achieve a better performance by using several neighborhood operators in the search. Moreover, we also note that the neighborhood operators have a varying performance depending on the moment we use them in the search. This leads us to the development of a new metaheuristic, the *Variable Adaptive Local Search* (*VALS*). This metaheuristic is designed to dynamically use a given set of neighborhood operators, by selecting a new one once an improving solution is found. It also takes into account the fact that all neighborhood operators do not have the same efficiency, by assigning them a non-uniform selection probability, and the fact that this efficiency changes along the search, by making this selection probability evolve every time a neighborhood operator is used.

The proposed algorithm is then validated by comparing its performances with two alternative variable local searches, confirming the benefit of using a non-uniform selection probability for the neighborhood operators, and the benefit of making this probability distribution evolve along the search.

Finally, we assess the global performance of the proposed algorithm by applying it on several instances of the problem under study. These experimentations show a significant improvement of the initial solution, especially by comparing these performances to the ones obtained with a simple local search. However, we know that the solution found with *VALS* are still far away from the optimal solution. Nevertheless, we know that the problem is very difficult to solve, and we also want to demonstrate the benefit of a multi-echelon inventory control approach over a single-echelon one. To this end, we propose to compare the solutions obtained with our proposed multi-echelon approach to the ones already existing in the partner company. This part of the work is detailed in the next chapter.

Chapter 4

Performance Analysis in Realistic Setting

Contents

4.1 Introduction 72
4.2 General Description 73
4.2.1 Inventory Control Process 73
4.2.2 Order Engines 73
4.2.3 Rolling Horizon 74
4.3 Simulation Context Construction 75
4.3.1 Initialization 76
4.3.2 Initial Solution Computation During Simulation 77
4.3.3 Environment 79
4.3.4 Instance 80
4.4 Simulation Process 81
4.4.1 Decision Agent 82
4.4.2 Sampler 82
4.4.3 Applicator 82
4.4.4 Monitor 83
4.4.5 Shifter 84
4.5 Simulator Overview 84
4.6 Experimentations 86
4.6.1 Experimental Protocol 86
4.6.2 Results Analysis 86
4.6.3 Conclusion on the Results 91
4.7 Conclusion 92

4.1 Introduction

One of the main questions and motivations for the work we accomplish is determining whether multi-echelon inventory replenishment is a good strategy for inventory control and comparing this approach to single-echelon strategies. By achieving this work, we expect to uncover the benefits and drawbacks of each approach, and to provide recommendations on their use in real-case settings.

A grounding motivation for developing and studying a multi-echelon inventory control approach is to show that this approach will outperform the traditional single-echelon approaches, at least on some aspects. As a result, we are trying in this chapter to determine whether our approach performs as we intend it to, which is the definition of model verification (Banks and Malavé [1984]).

In the previous chapters, we developed a realistic multi-echelon inventory control model and implemented a method to solve this model in a reasonable time. This method has been tested and evaluated on algorithmic aspects. In this chapter, we aim to evaluate our proposed algorithm on more practical aspects. We also want to study the ordering behavior of our proposed model, as well as that of single-echelon approaches. To achieve this, it seems necessary to reproduce the inventory management process of the case under study in a framework that is the closest possible to the reality. This framework will be useful to put the decision algorithms in realistic ordering situations, and to study the impacts of applying their proposed decisions.

Real-world complex systems, such as supply chains, operate in conditions depending on multiple random factors, which makes difficult to analyze them in real situations (Wiśniewski [2018]). Simulation can be defined as the imitation of operations being part of a real-world process or system over time (Banks [1999]). Simulation tools are suitable to analyze decisions in a framework more similar to the reality (Alstrøm and Madsen [1992]), and to accurately capture and explain the actual behavior of the system under study (Chu et al. [2015]). Moreover, simulation models are widely used to reproduce real-world systems and assess their performances in diverse situations or to study facets of their behaviors. In the field of supply chain management, and more specifically in inventory control, there exists some studies that use simulation as a tool to compare different inventory control strategies (Huq et al. [2006]) or to determine the impact of various parameters on the global supply chain performance (Ganeshan et al. [2001], Brown et al. [2001], Fleisch and Tellkamp [2005]).

As a result, simulation seems to be a relevant and performing solution to answer the aforementioned research question. This is why we choose to develop a simulation framework that reproduces the inventory control process of a two-echelon supply network on several time steps. This simulation tool is used as a frame that makes the decision algorithms calculate the recommended ordering plans in multiple contexts, and then apply these computed orders to verify their performances regarding practical criteria.

The remainder of this chapter is organized as follows. We first introduce key concepts for the performance analysis and recall the different order computation engines we want to compare in Section 4.2. Then, we focus on the detailed simulation process that is developed to answer the aforementioned research questions of this chapter. We focus on the simulation tool initialization and instantiating in Section 4.3. and we detail the different steps of the inventory control process reproduction phase in Section 4.4. We also provide an overview of the actual simulation tool developed as part of this work in Section 4.5. We present the experimental protocol conducted using the simulator and the results in Section 4.6. Finally, we provide a conclusion of this chapter in Section 4.7.

4.2 General Description

As mentioned in the previous section, the main objective of this chapter is to study the performances of our approach in a practical context, and to evaluate its performance from an industrial point of view. Moreover, we want to compare our approach to the ones already existing in the company.

In order to achieve this goal, we concentrate on the context in which the algorithms are used in practice. In the following, we refer to the inventory control decision algorithms as *order engines* or *decision agents*. Recalling that we work on a one-warehouse N-retailers (*1WNR*) supply chain, we want to develop a simulation model that reproduces this kind of supply chain, and especially the ordering and demand answering process of entities that manage inventory on several time periods.

In this section, we introduce the key concepts that we use during the simulation protocol. We also rapidly reintroduce the different decision agents that will be assessed in the simulation experiments.

4.2.1 Inventory Control Process

Let us first describe the inventory control process from a localized point of view. Each period of time is divided into several steps. For a given entity and item, each period starts with an initial inventory *I0*. Then, the order engine computes an order q given the state of the system and the demand previsions. From this time on, the inventory begins to evolve. First, the entity receives the orders placed before. Let t be the current period, and LT the total time needed by the supplying source to process an order once it is placed. Once the reception phase is over, the inventory level is calculated as follows.

$$I^R = I0 + q_{t-LT} \tag{4.1}$$

This inventory is the "effective" inventory that is used for the next steps of the time period, which is meeting the customer demand. We operate under a lost sales assumption, which means that every customer demand the inventory cannot meet is deleted, rather than carried forward to the next period. Let D be the external demand for the given item and period. After meeting the demand, the resulting inventory is computed as follows.

$$I^+ = \left[I^R - D\right]^+ \tag{4.2}$$

This resulting inventory I^+ is the end of period inventory. This amount of products will stay in the inventory until the beginning of the next period, and it is consequently the initial inventory of the time step $t + 1$.

These steps are detailed in the Figure 4.1, with a description of the inventory level evolution during the period. This process is applied for each entity and item of the system, at each time period of the simulation procedure.

4.2.2 Order Engines

As mentioned before, the goal here is to test several order engines (*i.e.* different methods for the order calculation phase of the process described in Section 4.2.1).

The first and main order engine we want to evaluate with the simulation framework developed in this work is the Multi-Echelon Ordering Plan (*ME-LS*) order engine. Let us quickly recall its characteristics. This decision agent, is a multi-echelon, multi-item and multi-period order engine.

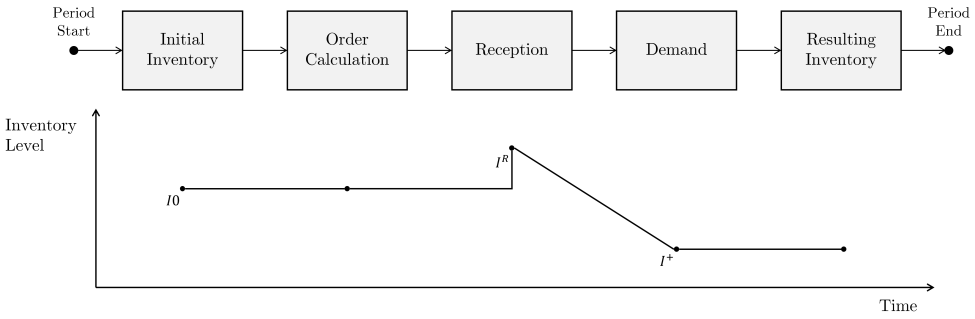


Figure 4.1: Description of the inventory control process on a given period with insight on the inventory level evolution.

The first step of the optimization process is the generation of a feasible solution. Then, a local search is applied to improve this initial solution. The decision agent is used on all entities and items of the system jointly, and computes an ordering plan for several periods. In the simulator, we use the orders computed for the first period of the solution (*cf.* Section 4.2.3 for further explanation).

The simulation protocol is used to assess the performance of *ME-LS*, but also to compare these performances to the ones of the already existing order engines in the company. These decision agents, *SPSE* and MP^2 are described in Section 1.6. As for *ME-LS*, let us rapidly recall their functioning.

Firstly, let us describe the *SPSE* (*Single-Product Single-Echelon*) decision agent. It is a single-echelon, single-item and single-period order engine. Its main feature is to compute an order for a given entity-item and for the period in which it is called. This order engine is used on every entity-item pair, for which we selected a supplying source beforehand (in the given supplying sources list, in our case the warehouse and all available external suppliers). The detailed process and optimization methods of this order engine is further explained by Sahu [2020, Chapter 5]. In the simulator, we use the calculated orders as is.

Secondly, we also work with the MP^2 (*Multi-Product Multi-Period*) decision agent. This order engine is still single-echelon, but multi-product and multi-period. Accordingly, this order engine is designed to compute the orders for an item group of a given entity and for a given number of time periods. For each item of the system, we pre-select a supplying source, in the same list as for *SPSE*, and group these items by entity and supplying source. Then, *SPSE* is used as a pre-solving tool to compute all order possibilities we want to take into account and their associated single-item cost (*i.e.* purchasing, holding and shortage cost). Then, the MP^2 order engine computes the best combination of these order possibilities on the given time horizon and returns the corresponding order planning. As for the *ME-LS* decision agent, we only use the orders computed for the first period of the planning horizon. The optimization protocol and precise functioning is described in Métaireau et al. [2022].

4.2.3 Rolling Horizon

In the multi-periodic replenishment and inventory control field, the main goal is to compute orders not for one single period but for a finite planning horizon containing several periods. One common assumption is that the order planning computation is periodical (Saracoglu et al. [2014]). This means that, for a given planning horizon length H, the orders are computed at time $t = 1$, and

strictly applied until time $t = H$. The next order planning computation happens at time $t = H+1$, and so on. This strategy is applied for many contexts, both single-echelon (Mousavi et al. [2013], Kim et al. [2015]) and multi-echelon (Aharon et al. [2009]).

In most of situations however, the external demand is an uncertain parameter, and the forecasted demand provided for the order computation is rarely the exact demand that the entities will face. As a result, we cannot be sure that the ordering plan computed at the beginning of the planning horizon will stay relevant along this number of time periods. As a matter of fact, the real inventory evolution will not necessary match the computed inventory projection used for the ordering plan due to the demand uncertainty. Then, the computed orders can be unsuitable for the real context, especially near the end of the planning horizon.

When the external demand parameter is uncertain, there exist studies that choose another planning strategy for multi-periodic contexts (Rahdar et al. [2018], Jung et al. [2004]). This strategy is the so-called *rolling-horizon* concept, introduced by Reklaitis [1982], that we describe in this section.

We use the concept of rolling horizon with the multi-period planning order engines MP^2 and *ME-LS*. In order to properly explain this concept, we need to introduce two notations. Let TH be the entire simulation time horizon. In other words, TH is the number of time periods we want to simulate during one experiment.

Then, we introduce H such that $H \ll TH$. H is the local planning horizon length for the multi-period solvers, or rolling horizon. This means that, at each simulation period t with the *ME-LS* or MP^2 order engines, their planning horizon is $[\![t, t + H]\!]$. Once the solver finishes the computation of the order planning on this horizon, only the orders supposed to be placed on time period t are saved to be applied during the simulation. The simulation step then continues until the end of the period, and the next planning horizon is $[\![t + 1, t + 1 + H]\!]$, and so forth until the last planning horizon *i.e.* until the simulation period $t = TH - H$.

We provide a representation of the rolling horizon concept on Figure 4.2.

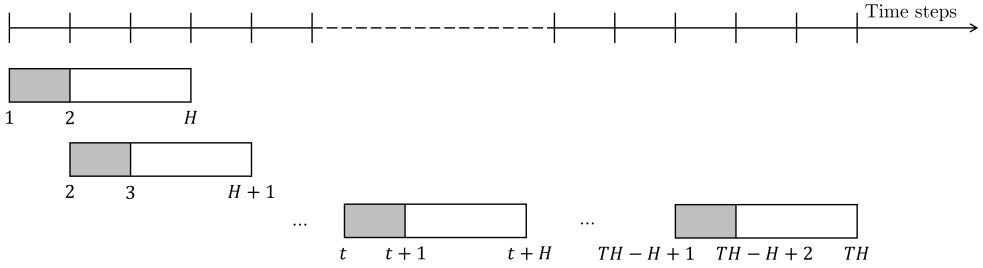


Figure 4.2: Representation of the rolling horizon concept.

4.3 Simulation Context Construction

The main goal of this chapter is to develop a simulation tool that is able to reproduce the previously described process on several time steps and for different order calculation methods (*i.e.* decision agents). In this section, we present in detail the simulation procedure. The simulator is a collection of several processes that replicate the inventory control system on one time period. These processes are called sequentially for one time step, and then the whole sequence is repeated for each time step

of the simulation horizon. To illustrate this concept, we provide a representation of the simulation protocol on Figure 4.3. In the following subsections, we describe the building of the simulation model and the definition of the context to be simulated. This corresponds to the "Environment" and "Instance" (respectively described in Sections 4.3.3 and 4.3.4) blocks displayed on the figure. We also detail the initialization phase of the simulation tool in Section 4.3.1. Finally, we provide some discussion about the initial solution computation for *ME-LS* agent in Section 4.3.2.

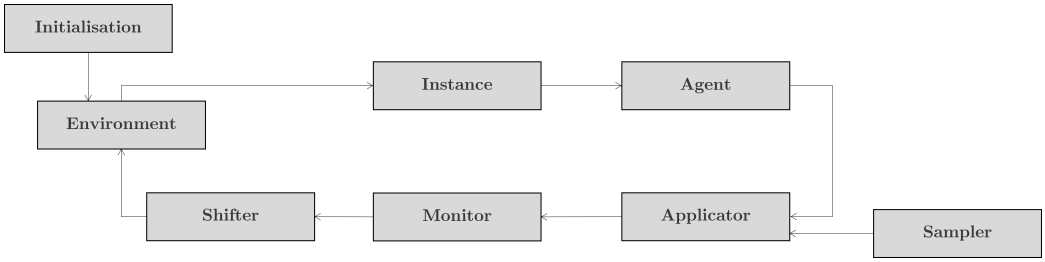


Figure 4.3: Description of the simulation protocol steps.

4.3.1 Initialization

Before entering the actual simulation process, there is an initialization process conducted only for the first simulation time step.

In this phase, we mainly create the objects that hold data during the simulation (the environment, described in Section 4.3.3, the instance, described in Section 4.3.4, and the monitor, described in Section 4.4.4).

We also create an initial state $\zeta^0 = (I0, O)$ for the simulation, which is the first context used for reproducing the inventory control process. In the data provided for our case study, we do not have information on the inventory levels and in-progress orders for the period under study. As a result, we have two options. Firstly, we can start from a so-called *null initial state*, *i.e.* we consider the initial inventory level and in-progress orders to be strictly null. Secondly, we can create a *non-zero initial state* based on the data. As the null initial state is hardly ever encountered in practice, we choose to build a non-zero state.

We want this initial state to be realistic, *i.e.* to represent a situation that may be encountered in practice. To do so, we set up an initial state computation protocol that starts from a null initial state, and mimic a simplified inventory control process during several periods to obtain credible initial inventory and in-progress orders.

The first step of this protocol is to solve the inventory control problem for the first time period of the dataset. For this part, we use the continuous relaxation of the Mixed-Integer Linear Program ($MILP$) described in Chapter 2. As mentioned in the same chapter, the $MILP$ is too slow to be used, so the Linear Program (LP) obtained after relaxation is a good alternative to obtain a basis for a solution in a reasonable time. However, this solution is not feasible because the resulting quantities are not integer. To make this solution feasible, we round the quantities and repair the obtained solution. To do so, we use the same repair protocol as the one described in Chapter 3.

The outgoing set of quantities represent a feasible ordering plan for the first periods of the total horizon available in the dataset. Let H be the planning horizon length. F is the set of flow-items of the problem considered, and ST is the set of entity-items. Let $(q_{tf})_{t\in H, f\in F}$ be this ordering plan.

Let $(i_{ts})_{s\in ST, t\in H}$ be the corresponding inventory levels. This serves as basis for the construction of a representative initial state. Let us assume that the predicted demand on this planning horizon is the exact customer demand. Let T^{-} be the frozen horizon length. We now choose an integer $N_D < T^{-}$. This integer represents the period we use to construct the initial state. In other words, we shift N_D time steps in the planning horizon. The elements of the initial state are constructed as follows.

$$\forall s \in ST, \qquad I0_s = i_{T^{-}+N_D+1,s} \tag{4.3}$$

$$\forall f \in F,\ \forall t \in [\![1, T^{-} - N_D]\!], \qquad O_{t,f} = 0 \tag{4.4}$$

$$\forall f \in F,\ \forall t \in [\![T^{-} - N_D + 1, T^{-}]\!], \qquad O_{t,f} = q_{t+N_D f} \tag{4.5}$$

We then create the initial state ζ^0, which is the set of the previously computed *I0* and O, and add it to the environment object. This is the starting point of the simulation process.

4.3.2 Initial Solution Computation During Simulation

In the Multi-Echelon Ordering Plan (*ME-LS*) procedure, we have to compute an initial solution in order to initialize the search. In Section 3.3.2, we detail a method to create this initial solution starting from scratch. This method is adequate for a context in which we do not have any prior information on a feasible solution. In the following, we refer to it as the *step zero computation method*.

In a simulation framework, however, we can have an insight on the structure of a good solution using the information gathered in the previous simulation period. As a matter of fact, for a given time step t, the solution obtained in the period $t-1$ can be slightly modified to be a compliant starting solution for the current period.

The first intuition we had was that it would be more interesting to use this previous solution for building the one for the current period. Actually, this solution is the result of a search process, so exploiting this process seems to be a good strategy to enhance the *ME-LS* optimization performances. So, we set up a method to compute an initial solution for *ME-LS* using the best solution found at the previous period. This method is described in Algorithm 5. In the following, we refer to it as the *shifting* computation method.

In the first simulation experiments we conducted, we used the shifting computation method for *ME-LS*. When analysing the results, we notice cost variations that are not compliant with our expectations, and Analyze a bit further the quality of the initial solutions. To do so, we run a simulation with *ME-LS* decision agent on several periods and record the cost function values of the initial and final solutions each period (respectively before and after the execution of the local search). The simulation is launched on the instance `fora` with 15 items and a simulation horizon $T^S = 45$. The results are displayed on Figure 4.4.

Algorithm 5 Shifting computation method

Input: Time step t
Initial solution $S = \emptyset$
if $t = 1$ **then**
    Compute initial solution using the step zero computation method
else
    Get previous solution $S^{-} = (q_{tf})_{t \in T, f \in F}$
    Create new orders table $\left(q'_{tf}\right)_{t \in T, f \in F}$
    Shift the solution one step forward
    $\forall f \in F,\ t \in [\![1, H-1]\!],\ q'_{tf} = q_{t+1f}$
    Fill the table for the last time period
    $\forall f \in F,\ q'_{Hf} = 0$
    $S \leftarrow \left(q'_{tf}\right)_{t \in T, f \in F}$
end if

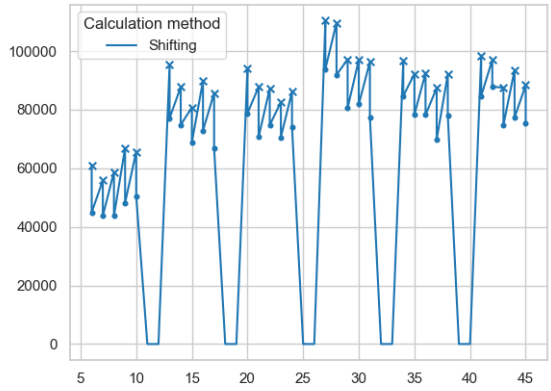


Figure 4.4: Representation of solution cost function values during the simulation experiment with *ME-LS* and information exploitation computation method. The '×' markers are for initial solutions and the '•' markers are for final solutions. Zero values are the week-ends.

We notice that, after the first simulated week, there is a significant increase in the cost of the initial solution for a given period. Moreover, the local search does not compensate this increase and the solution cost function values stay high compared to the first week. This can be explained by two assumptions. The first one is that the demand faced during the simulation process is very different from the demand forecasted. This leads to an initial state for the next period that is not resembling the state projected in the previous ordering plan. As a result, using the shifting method to compute an initial solution can lead to an inappropriate ordering plan.

As this may be a characteristic of the instance under study, we compare this curve to the one obtained with the same experiment but with the step zero computation method for all simulation time periods. The results are displayed on Figure 4.5.

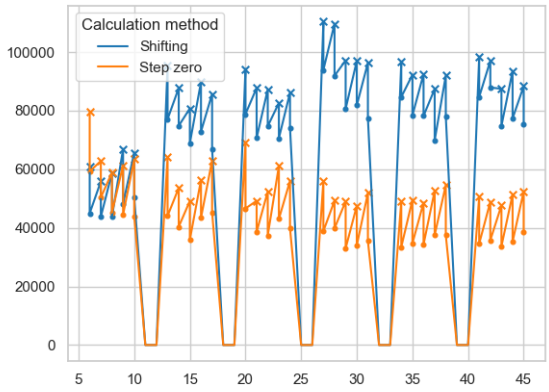


Figure 4.5: Comparison of solution cost function values during the simulation experiment with *ME-LS* and information exploitation/step zero computation method. The '×' markers are for initial solutions and the '•' markers are for final solutions. Zero values are the week-ends.

Comparing both curves, we can see that the step zero is higher for the beginning of the first week. However, for the end of this week and for all the following ones, the cost function values of solutions are stable and significantly lower than the solutions obtained with the information exploitation computation method. This means that the global algorithmic performances of the local search during the simulation are better with the step zero computation method than with the information exploitation computation method. To validate this result, we reproduced the experiment on several instances and reached the same conclusion. As a result, we choose the step zero computation method for the rest of the simulation experimentations.

4.3.3 Environment

The first phase of the protocol is the environment set-up. The environment is the simulator's element that contains all the information related to the system under study. It is the object defining the structure of the simulated context we want to build. It states which instance of the case study is used in the current simulation experiment. It also holds the hyper-parameters that generally describe the case study. For instance, elements like the storage capacity of entities W or the fixed cost value Ck are defined and stored by the environment. These elements are the additional

elements adjoined to the data to complete the inventory control model (*cf.* Section 2.4.1). To detail the information categories described above, we provide an exhaustive list of the elements contained in the environment in Table 4.1.

Table 4.1: Description of the elements defined in the environment object of the simulation tool.

Description	Elements
Instance characteristics	Generation mode of the instance `gen`
	Number of items P
Problem specifications	Entities capacity W
	Franco penalty factor ω^{Cf}
Cost specifications	Holding cost factor ω^{Ch}
	Shortage cost factor ω^{Cs}
	Fixed cost value Ck

At last, the environment also indicates the current simulation time step and contains its initial state ζ^0. The time step parameter states which time period of the simulation horizon is presently under study. The initial state is the set of current in-progress orders and the entity-items' initial inventory. Both will be updated along the simulation. As a result, the environment is the global descriptive process of the simulation protocol, and builds the general framework for the simulation.

4.3.4 Instance

The second process is the instance creation. The instance is the element that contains all practical data used for the resolution and replication phases. Based on the environment hyper-parameters and the available inputs, the instance creation process builds the suitable data structure needed to compute orders, apply them, and move one step forward. In a sense, the instance can be viewed as a tangible representation of the simulated system described by the environment.

First, it contains all the system specifications, mainly the sizes of every input set of data. Then, it contains data defining the demand and order specifications. This includes the entities' ordering calendar, *i.e.* if the entity is open at each period of the simulation horizon, and all the demand forecast for a given time horizon. As *SPSE* and MP^2 are single-echelon models, they consider the demand coming from the stores as an external demand. Accordingly, the instance holds the external demand forecast for the stores, but also the store orders forecast for the warehouse. We provide a detailed list of the elements mentioned above in Table 4.2.

Then, the instance builds the data structure describing the inventory control system and the interactions between elements, based on the informations extracted from the input dataset. Considering the tools we choose to code the simulation framework, we implement this part using dataframes. Dataframes are bi-dimensional data structures that are useful to store and manipulate data in Python, as the language provides libraries to handle them. There is one dataframe describing the entities, one describing the entity-items, one describing the flows between entities, and one describing the flow-item pairs. The elements contained by each dataframe are described in Table 4.3.

Table 4.2: Description of the elements defined in the instance object of the simulation tool.

Description	Elements
System specifications	In-progress horizon size FH
	Number of flow-items $\|F\|$
	Number of entity-items $\|ST\|$
	Number of entities $\|E\|$
	Number of flows $\|FL\|$
	Number of warehouses $\|E^-\|$
Demand and order	Forecast for stores
	Forecast for the warehouse
	Entities ordering calendar

Table 4.3: Description of the dataframes built during the instance creation process.

Dataframe	Flow-items	Entity-items	Flows	Entities
Elements	Source entity SE	Entity EN	Source entity SF	Identifier e
	Target entity TE	Item IT	Target entity TF	Capacity W
	Item IT	Holding cost Ch	Franco threshold FR	Capacity
	Lead time LT	Shortage cost Cs	Franco penalty Cf	exceeding
	Transport cost Ct		Fixed cost Ck	cost Ca

The instance is the process that holds every useful data for the remaining phases of the simulation, and especially the order calculation phase. These data are stored in different dataframes that are called and modified during the next processes. Before the beginning of the actual simulation process, the entity-items and flow-items dataframes are updated respectively with the initial inventory and in-progress orders of the initial state ζ^0.

4.4 Simulation Process

In this section, we concentrate on the different steps happening after the simulation has been correctly defined and the simulated context parameterized. In other words, the processes described in the previous sections are used to model the context we want to reproduce, and the processes we describe in this section allow to play the scenario we want to simulate. In the following subsections, we detail the processes corresponding to the "Agent", "Sampler", "Applicator", "Monitor" and "Shifter" blocks displayed on Figure 4.3.

4.4.1 Decision Agent

The next process is the order calculation agent. The order engines represent the decision making phase, in our case the order computation. Accordingly, in this phase, we call one of the agents described in Section 4.2.2, and let them compute the order (or order plan) they consider to be the best. We assume that the order recommendation is trusted by the decision maker and that the proposed orders are strictly placed without any modification. Before the optimization phase, we need to adapt the instance to the current context of the simulation procedure.

First, we need to add the time period-specific information to the dataframes. The first step is to generate the local opening calendar for the current rolling horizon and add it to the dataframe that will contain the orders. This calendar is drawn from the total horizon calendar available in the instance. The local calendar is of use for MP^2 and *ME-LS*. We then create the simulation demand scenario from the forecast. To do so, we take the mean of the probabilistic forecast for every entity-item of the instance and every period of the rolling horizon (for the stores and the warehouse), and add it to the entity-items dataframe.

The next step is to create the data structure that will effectively be used by the decision agent. For *ME-LS*, the dataframes are already compliant so we use them without any modification. For *SPSE*, we construct an independent replenishment problem for each entity-item pair of the system. Consequently, we construct a sub-instance for each entity-item (warehouse included), and pre-select a supplying source for every entity-item. We reproduce the actual decision rule of the case we study to pre-select the supplying source. For the warehouse, we select the cheapest supplier. For the stores, the rule is as follows: if the warehouse has inventory, then choose the warehouse, else, choose the cheapest supplier. For MP^2, we still manage the entities separately, but the items are grouped within these entities. We first pre-select the supplying sources for every item of the system, and them group them by entity and supplying source. Each group represents an optimization problem that will be solved by MP^2.

The next step is the order computation. Each solver computes orders for the sub-instances mentioned above and return its ordering plan. We then take the orders computed for the current period and add them to the flow-item dataframe.

4.4.2 Sampler

In this phase, we generate a demand scenario for the application of the previously computed orders. This allows to create a deterministic demand for all items and entities that face external demand, and confront the orders to a realistic scenario.

To generate this scenario, we use a Poisson distribution draw of the given forecast mean. We only use the forecast of the stores for this phase, as the "external demand" for the warehouse is the aggregation of the stores' orders.

This demand scenario is applied to the dataframe that relates to entity-items, and replaces the demand we took into account for the order engine computation phase.

4.4.3 Applicator

In this phase, we apply the computed orders and play the series of events of the simulated time period.

The first step, for *SPSE* and MP^2, is the order allocation process. As a matter of fact, as the entities are treated separately, the sum of the orders placed from the stores to the warehouse may

exceed the warehouse inventory level. In order to rectify this problem, we set up a warehouse inventory distribution process. This process changes the orders placed by the stores in order to respect the inventory level. We choose to allocate the orders proportionally. We describe how we compute the quantities effectively sent by the warehouse in the following.

Let $I_{WH}(p,t)$ be the warehouse inventory level for an item $p \in P$ at a period $t \in T^S$. Let $\mathcal{Q}(p,t)$ be the sum of orders placed by the stores for these item and period. Let $E^+ \subset E$ be the set of entities that are stores (*i.e.* the whole set of entities E except the warehouse), and let m be a member of this set E^+. We note $Q_m(p,t)$ the order placed by this store m to the warehouse for item p at period t. So, we have the following equation.

$$\mathcal{Q}(p,t) = \sum_{m \in E^+} Q_m(p,t) \tag{4.6}$$

Using all of these informations, we can calculate the quantities sent by the warehouse to the stores. These quantities are computed as follows. This enables an inventory allocation that is proportional to the orders placed by the stores and that never exceeds the warehouse inventory levels.

$$\forall p \in P,\ t \in T^S,\ m \in E^+,\ Q^+_m(p,t) = \left\lfloor \frac{Q_m(p,t) \times I_{WH}(p,t)}{\mathcal{Q}(p,t)} \right\rfloor \tag{4.7}$$

Once this allocation is computed (only for *SPSE* and MP^2), we unfold the next simulation steps, as described in Section 4.2.1. We begin by simulating the receptions of orders that covered their whole lead time and change the inventory in consequence (*cf.* equation 4.1). We then answer the demand scenario drawn by the sampler phase of the simulation process described in Section 4.4.2. Finally, we compute the new inventory values and all other intermediate variables (*e.g.* lost sales, Franco penalties). These intermediate variables allow us to calculate the different performance criteria in the monitor phase.

4.4.4 Monitor

In this phase, we compute all the performance criteria of the orders placed for the simulated period. In the following, we list them and give a quick explanation of their meaning. First, let us describe the criteria that we want to minimize with the order strategies of the decision agents.

Inventory levels: This criterion is the sum of all items remaining in the entities once the reception and demand of the period have been treated. We differentiate warehouse inventory level and store inventory levels.

Lost sale levels: This criterion is the sum of the lost sales of all items and for all stores in the period. This represents all the unmet demand.

Total costs: This criterion is the total costs induced by the orders placed in the period. We differentiate the purchase and transport cost induced by the warehouse orders, the purchase and transport cost induced by the stores' orders, the fixed costs and the Franco costs.

We also monitor one last criterion, that allows us to interpret the behaviors of the order engines. We do not want to minimize or maximize this criterion, this is only an observation to help our result analysis.

Amount of orders: This represents the total amount of orders placed by stores in the period for all items. We differentiate the orders placed to the warehouse and the orders placed to the external suppliers.

We also monitor the execution time of the agents. This is a criterion that we want to be as small as possible, but it is a more "algorithmic" feature used to compare the decision agents.

4.4.5 Shifter

This phase is the last one of the simulation process, and loops with the environment for the next time period of the simulation. In this phase, we compute the final data of the simulation for the current time period, which will be the initial data for the next one.

The end-of-period inventory levels (*cf.* equation 4.2) are the initial inventory levels for the next period. These values have already been computed in the applicator phase (Section 4.4.3), so we just have to record them for the next simulation period.

The in-progress orders set O are shifted one step forward. Thus, we create the next-step in-progress orders set O^+. For all flow-item f and time period $t \in [\![1, FH-1]\!]$, we perform the following transformation: $O^+_{tf} = O_{t+1f}$. We also incorporate the orders computed in the decision agent phase into the in-progress orders set: $O^+_{FHf} = q_{0f}$. This new set is used for the next time-period.

For the *ME-LS* agent, we also need to compute an initial solution for the next period. We chose to use the *ME-LS* initial solution computation process at each period. The initial solution for *ME-LS* is based on the resolution of the relaxed linear program of the problem for the whole horizon. The resulting quantities are then rounded to create an integer solution, which is repaired to respect the constraints. The initial solution generation process is further described in Chapter 3. As mentioned in Section 4.3.2, we also tested other methods to generate initial solutions during the simulation, but this one was the most interesting.

Then, we can change the environment's current time step from t to $t + 1$, and begin the loop from the first step for the next simulation period.

4.5 Simulator Overview

In the past sections, we decomposed the simulation tool into several sub-processes. This decomposition allowed us to describe every step of the whole simulation process, and to detail their specific functions in the simulation tool.

In this section however, we provide a more comprehensive overview of the complete simulation tool. This allows to better represent the algorithm that is used for the simulation experiment, and to describe how the different sub-processes are linked together. The pseudo-code for the complete simulation tool is provided in Algorithm 6.

The *Initialization* part of the algorithm (lines 2-9) corresponds to the processes described Section 4.3. In this part, we create the context that will be simulated during the experiment, and set-up the starting point of the simulator.

The remainder of the algorithm corresponds to the processes described in Section 4.3. It details the inventory control reproduction process for one period, which is performed for every time period of the simulation horizon T^S. It also details the steps followed in case of closing days, *i.e.* when the entities are not open, accordingly they cannot place orders and do not face demand.

The main object of interest resulting from the simulation experiment is the monitor. As it records all the evaluating criteria at every time period, this is the element that allows us to visualize and analyze simulation experiments results. On account of that, the monitor is the output of the algorithm.

Algorithm 6 Complete simulation process

Input: Specification of the simulation experiment
Initialization
Create the Monitor object M^{SIM}, Environment object E^{SIM}, and Instance object I^{SIM}
Initialize a non-zero initial state $\zeta^0 = (I0, O)$
Update the beginning period $b \leftarrow 1 + N_D$
Update the end period of the simulation horizon $end \leftarrow TH - H + 1$
Set simulation time period $t = b$
Simulation loop
while $t \leqslant end$ **do**
    Get calendar opening boolean C_t
    if C_t is `True` **then**
        Create simulation demand scenarii for stores and warehouse, respectively D_{E^+} and D_{WH}
        Assign initial state and demand scenarii to the dataframes for entity-items and flow-items
        Create the Agent object A^{SIM}
        Solve the current inventory control problem with A^{SIM}
        Assign computed orders to the flow-item dataframe
        Create demand scenario for simulation D and assign it to the entity-items dataframe
        Compute intermediate variables
        for $k \in M^{SIM}$ **do**
            Compute and record k
        end for
        Shift one step forward by calculating new $I0$ and O, assign them to ζ^0
        $t \leftarrow t + 1$
    else
        Create demand scenario and orders with only null values, assign them to dataframes
        Compute intermediate variables
        for $k \in M^{SIM}$ **do**
            Compute and record k
        end for
        Shift one step forward by calculating new $I0$ and O, assign them to ζ^0
        $t \leftarrow t + 1$
    end if
end while
Return Monitor M^{SIM}

4.6 Experimentations

This section is dedicated to the experimental part of our work on performance analysis and simulation. It includes a description of the experimental protocol we put in place and an analysis of the different results we obtained. This allows us to compare the different order engines on the quality of their proposed decisions in practice.

We conduct experimentations using the realistic instances drawn from the case-study dataset. The complete instance set we built for the thesis work is described in Chapter 2.4.2. Due to time constraints, we chose a subset of these instances to be simulated during this experimentation part. This subset of instances, as well as the experimental protocol are described in Section 4.6.1.

The results of the experimentations are presented and analyzed in Section 4.6.2.

4.6.1 Experimental Protocol

In order to assess the performances of order engines and to compare them, we launch several simulations using the realistic instances for our problem. The main goal is to compare the three decision agents under study (*cf.* Section 4.2.2) on the same contexts so that the comparison is the fairest possible. One simulation experiment is characterized by a set of parameters, and is designated as $\left(J, A^{SIM}, TH, \sigma\right)$. The first parameter is the instance J, described by its generation mode `gen` and number of items P. The instance is designated as follows: $J = (\texttt{gen}, P)$. The second parameter is the agent A^{SIM}, designated by *SPSE*, *MP*^{2} or *ME-LS*. The third parameter is the number of simulation periods *TH*. This parameter has to be smaller than the number of periods on which forecast is provided in the input data. The last parameter is a seed σ, which allows to use a fixed demand scenario for simulations experiments to be compared with each other. In other words, for a given set of instances J, number of simulation periods *TH* and seed σ, we can build a simulation experiment set $\left\{(J, \textit{SPSE}, TH, \sigma), \left(J, \textit{MP}^2, TH, \sigma\right), (J, \textit{ME-LS}, TH, \sigma)\right\}$, so that we can compare all order engines on a given context.

If we want to reproduce several weeks of inventory control for all solvers, this requires to launch all order engines a significant number of times. This means that the duration of simulations can go from several minutes for *SPSE* to around a day for *MP*^{2}. As a result, we cannot test every instance of our dataset and we select a subset of instances from all the instances drawn from the case-study data. Each instance selected is tested with a given set of seeds of size N^σ, *i.e.* we build N^σ simulation experiment sets for all instances chosen.

For the results we provide in the present work, we choose the following experimental parameters.

Instance subset: $[(\texttt{gen}, P),\ \texttt{gen} \in \{\texttt{fora}, \texttt{rand}\},\ P \in \{15, 25, 40, 100\}]$

Number of simulation periods: $TH = 45$

Number of seeds: $N^\sigma = 10$

4.6.2 Results Analysis

In this section, we present and analyze results of the experiments described in the previous section. We aggregate results by instance size, so that we can have a better overview of the order engines behavior tendencies. The results for individual instances are available in Appendix A.

During the experiments, we record the different performance criteria described in Section 4.4.4 for all order engines, and compare their values. In the remainder of this section, we display these results and provide analysis and discussion.

The first criterion we focus on is the inventory level. On Figure 4.6, we display the inventory levels for each simulated period of the experimentation. These inventory levels are averaged on all instances and seeds for every problem size, and split between warehouse inventory levels and stores inventory levels. For *SPSE* and MP^2, we can note that, even though the warehouse inventory levels are quite stable and periodic, the inventory levels for the stores tend to increase along the simulation horizon. On the other hand, inventory levels obtained with *ME-LS* agent are more stable. Moreover, global inventory levels are lower with the multi-echelon agent rather than with both single-echelon agents.

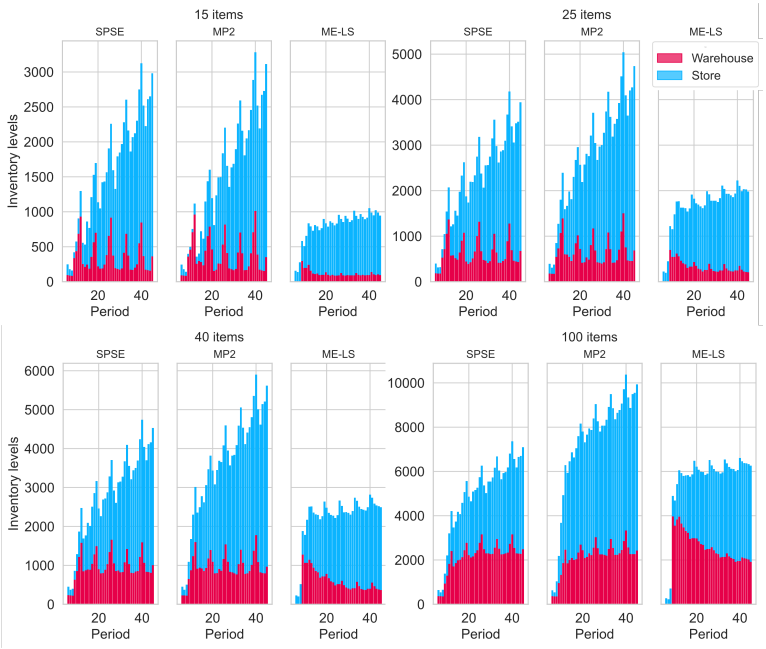


Figure 4.6: Inventory levels at each simulated period averaged on all instances and seeds, for every problem size. Values are split for warehouse and store inventory levels.

Now that we compared inventory levels, we want to confront them with the lost sales levels of the stores. We display those results on Figure 4.7. This figure represents average, minimum and maximum lost sales levels for all stores at each period of the simulation, aggregated on all generation mode and seeds. We plot one separated graphs per problem size. We notice that, for all problem sizes, although the lost sales levels of *SPSE* and MP^2 agents tend to decrease along the simulation, those obtained with *ME-LS* decision agent are globally lower. The only instance group in which *SPSE* and MP^2 manage to reach *ME-LS* inventory levels is the instance group with 15 items.

If we combine results obtained for inventory levels and lost sale levels, we can see that, with our proposed multi-echelon resolution method, we can achieve a better inventory management strategy. As a matter of fact, this method allows to better answer the demand with lower inactive inventory,

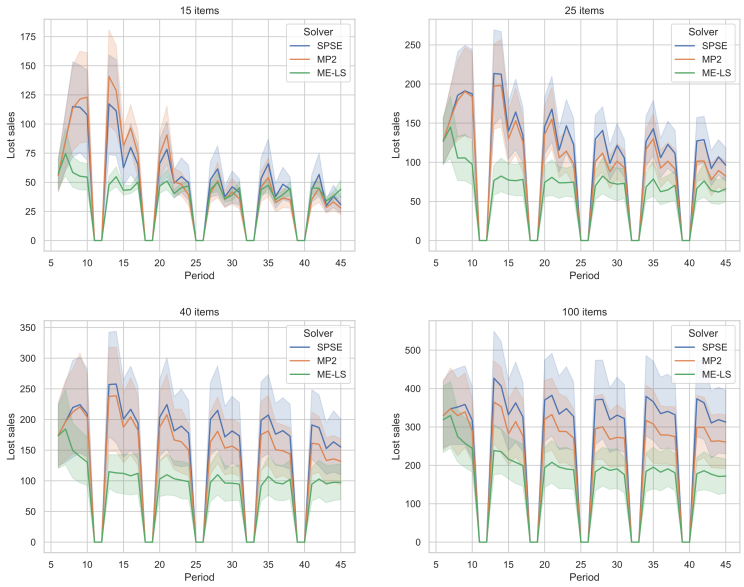


Figure 4.7: Lost sales levels at each simulated period for all instances and seeds, for every problem size. Each curve represent an order engine.

which means that the order strategy is more effective to manage the inventory / demand satisfaction trade-off.

Let us focus on the cost induced by the orders proposed by the decision agents. On Figure 4.8, we display the total costs incurred by the ordering strategies at each period of the simulation horizon. These costs are averaged on all generation modes and seeds for each problem size. In this part, we consider only the "monetary" costs, as opposed to "assumption" costs. Hence, we do not consider holding cost or capacity exceeding costs for instance. These parameters are added into the cost function of the algorithms to influence their behavior according to decision maker preferences, but they are not actually paid during the inventory control process. As a result, we split the total cost into four categories: transport and purchase cost of the warehouse, transport and purchase cost of the stores, Franco penalty cost, and fixed cost.

We observe two different cost tendencies. For *SPSE* and MP^2, we can note a high peak for all costs on the first week of simulation. In the following simulated weeks, the total cost is lower and mainly due to warehouse transport and purchase cost. As the cost of transporting products from the warehouse is very low in our case study, we can deduce the ordering behavior of the engines for these weeks. Looking at the inventory levels of the warehouse in the first periods in Figure 4.6, we notice that they are lower than for the next period. As a result, we can explain the cost behavior by the following. In the first week, the warehouse does not have many items in its inventory, and the stores consequently place a lot of orders to the external suppliers. Then, as the warehouse has more inventory, the supplying decision rule pushes the stores to order quasi-exclusively to the warehouse. In addition, regarding the Franco cost, we observe high Franco penalties on the first weeks for single-echelon engines, especially for *SPSE*. This is not the case for *ME-LS*, and this behavior can be explained by the fact that the solving method contains a neighborhood operator

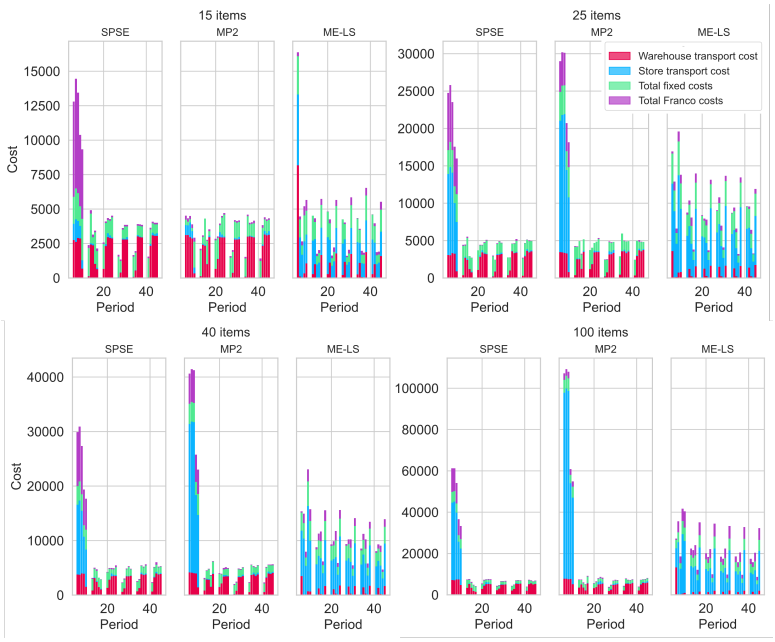


Figure 4.8: Total costs at each simulated period averaged on instances and seeds, for every problem size. Costs are split in four types: Warehouse purchase and transport, Stores purchase and transport, Franco and Fixed costs.

that aims to reduce the Franco costs.

For *ME-LS* however, the costs do not show the same tendency. In fact, the transport and purchase cost of the warehouse is lower, and the total cost is higher. In this case, and combining this observation with the inventory levels observed on Figure 4.6, we can deduce that the *ME-LS* order engine tends to keep the inventory of the warehouse lower, and to make stores order more to the external suppliers. Globally, the total cost is higher for every week except the first one.

On Figure 4.9, we display the amount of orders placed by stores at each period of the simulation horizon. These amounts are averaged on all generation modes and seeds for each problem size. We separate orders between orders placed to the warehouse and orders placed to the suppliers.

For *SPSE* and MP^2, we observe relatively high orders of the two first weeks, and then they tend to decrease quickly and to be quasi-exclusively placed to the warehouse. For *ME-LS*, the total amount of orders is smoother, even if we still observe a small peak of orders in the two first weeks. For the following weeks, the total amount of orders is higher compared to the one of both single-echelon order engines. Moreover, the orders are roughly equally split between the warehouse and suppliers. These observations tend to confirm the ordering behavior previously described in the cost analysis paragraph.

Finally, we want to monitor and compare the execution time of all order engines. For this criterion, we set up another experimentation. In fact, if we use the execution time obtained with the previously detailed experimental protocol, we observe results that are not consistent on all instances. These non consistent results are due to computation parallelization. In fact, the library we use for managing the dataframes performs "hidden" multi-threading, and this impacts the execution time of the algorithms when we launch several of them in parallel. As a result, we cannot

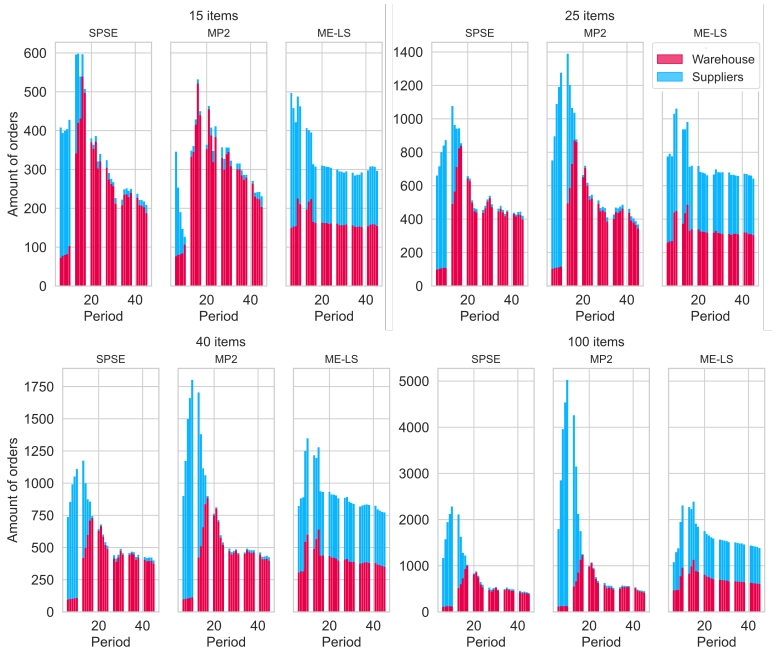


Figure 4.9: Amount of orders placed by the stores at each simulated period averaged on all instances and seeds for every problem size. Orders are split between orders to the warehouse and orders to the suppliers.

conclude on the execution time tendencies with these results.

Moreover, the decision agent's execution time is a criterion that is stable along the simulation. As a result, we can consider one simulation per decision agent to be representative for monitoring the order engine execution time, as each simulation provides several tens of values for the criterion. In this regard, we evaluate the execution time distribution on all the order computation of one simulation experiment. We focus on the instances obtained with `fora` generation mode, as these are the instances with the most demand and accordingly the most orders to compute. The results of this experiment are displayed on Figure 4.10. The Figure represents the execution time distribution for one simulation with each order engine for a fixed seed. The simulations are run with instances (`fora`, 15), (`fora`, 25) and (`fora`, 40).

The first observation we can make from this experiment is that the *SPSE* order engine is very fast compared to the two others. This can be explained by the fact that this engine is the one that answers the smallest (and accordingly easiest to solve) problems, as it separates the complete problem into single-echelon, single-period and single-item sub-problems. Moreover, this order engine has been designed to be very quick. When we compare MP^2 and *ME-LS*, we can see that the *ME-LS* order engine computation times are globally lower than the MP^2 (which treats single-echelon, multi-period and multi-item sub-problems) order engine computation times. Moreover, the difference between both seems to increase with the problem size.

As a result, our proposed method is slower than *SPSE*, which is a very efficient decision agent for its application framework. Yet, it is still quicker than MP^2, while answering a multi-echelon problem, that is more complex and treats more constraints.

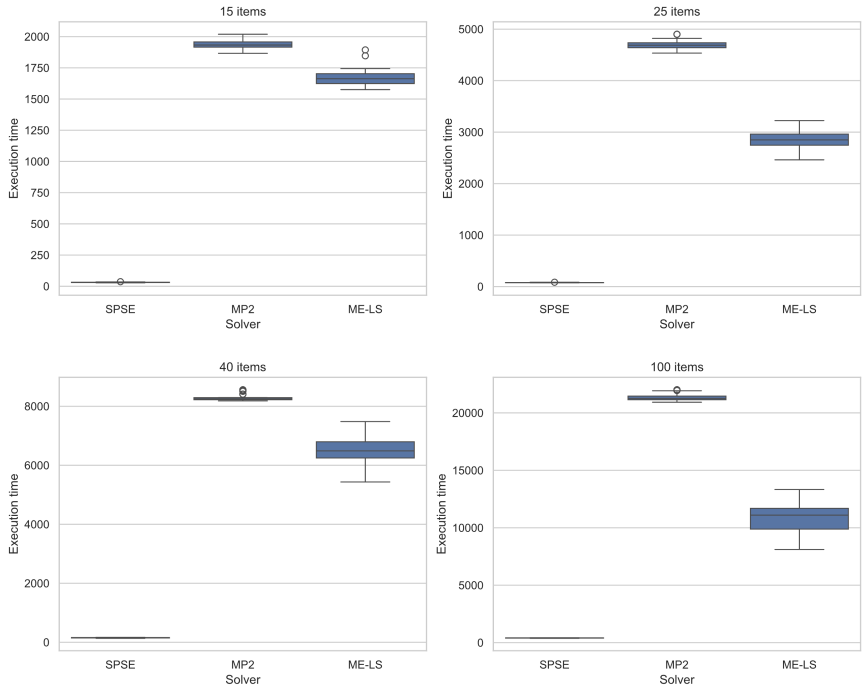


Figure 4.10: Execution time in seconds of solvers distribution on all period of one simulation and all problem sizes with generation mode `fora`.

4.6.3 Conclusion on the Results

In the previous section, we presented the experimental results on the selected performance criteria and analyzed them separately. Now, we can combine the different observations and draw several conclusions on the multi-echelon approach in regard to the two single-echelon approaches already in place.

First, the two single-echelon methods have a very similar behavior. We can observe some differences between them, but their global ordering strategies are really close. The multi-echelon approach, however, propose a different ordering strategy. If we focus on the costs and quantities ordered by the single-echelon order engines, we can see that the agents place orders from stores to suppliers mainly - and almost only - on the two first weeks. This corresponds to time periods when the warehouse inventory is lower, and can be explained by the supplying decision rules (*cf.* Section 4.4.1). Still, consequently to these supplying decision rules, the stores almost place no more orders to the suppliers once the warehouse inventory is higher. The *ME-LS* order engine, for its part, do not pre-select supplying sources and do not have to choose between warehouse and external supplier. Due to the relaxation of this decision rule, the decision agent can distribute the orders more freely. Indeed, we observe on Figure 4.9 that the amount of orders placed from the stores to the suppliers are higher, although the orders placed to the warehouse are not drastically lower.

This results in a total amount of orders placed by the stores higher with the multi-echelon approach than with the single-echelon ones. This translates into an increasing of the total costs

induced by the orders, especially regarding the stores transport and purchase cost.

If we look at the inventory level / lost sale balance, the results are also different between both approaches. As a matter of fact, the orders placed multi-echelon approach result in lower inventory levels, but also lower lost sale levels. This means that *ME-LS* decision agent achieves a better service level and a better inventory management. Consequently, the orders seem more relevant for this aspect, as the stores answer better to the demand while less saturating the inventory.

In consequence, we can say that, although the orders placed and the costs are higher, we achieve a greater service level while keeping a lower inventory level with the multi-echelon approach. This compromise between inventory level and lost sales is very important in inventory control, and is usually the main criterion to judge the performance of an ordering plan. Therefore, the results we obtain imply a better quality of the ordering strategy with regard to the customer satisfaction and inventory level balance.

Regarding those criteria, we can conclude that the multi-echelon approach we developed allows a better inventory management compared to the two single-echelon approaches presented in this research work. This can be explained by two aspects. Firstly, the multi-echelon system has a better visibility on the inventory and needs of all entities. As a result, it can adapt the stores orders in regard to the available warehouse inventory level. In a single-echelon setting, each entity has information only on its own stock and needs, so the order computed can seem very good for its point of view, but in regard to the whole system, these orders can become irrelevant because the warehouse may have to arbitrate orders. Secondly, the multi-echelon order engine has more freedom concerning the supplying source decision rules, and can balance lower stores orders to the warehouse (due to inventory constraints) by completing them with orders placed to suppliers. The single-echelon approach cannot treat this aspect, and this penalizes orders because they are conditioned by the pre-selected supplying source.

Still, one limit of our simulation tool is that it does not include a forecasting engine. In fact, in the real framework in which the single-echelon order engines are used, the forecast is recomputed at every time period. For the external customer demand, this is not a problem because the order engines does not influence the customer behavior. For the warehouse forecast however, it is computed based on the previous orders placed by stores, and so the ordering strategy can influence this prediction. In our case, we use a pre-computed forecast on a given period. If we integrate a forecasting engine to the simulation procedure, this could give a more relevant warehouse forecast for single-echelon decision agents, because this forecast would be more adapted to the simulated context, as a result the warehouse inventory could be more relevant.

4.7 Conclusion

In this chapter, we describe a simulation tool that aims to reproduce the inventory control process of a supply chain. This framework comprises several phases. It first describes an inventory control context based of the information extracted from input data. This context is the basis of the framework and represents the inventory control system we want to reproduce throughout the simulation. Then, the simulation tool computes an ordering plan using an optimization algorithm, and simulate the application of this ordering plan: orders are placed in the in-progress orders list, and the inventory projection is computed with the reception of previous orders and the satisfaction of the demand. This results in a new inventory level that will be used for the next simulation time period. This process is repeated on a given number of time periods.

The idea behind the construction of this simulation tool is to run it on several instances of our case study's dataset with several order engines. The target is twofold. First, we can evaluate our multi-echelon, multi-period and multi-item order engine (*ME-LS*) on industrial performance criteria, and conclude on the quality of its ordering decision with a more practical point of view. Second, we can also use it to monitor the same performance criteria for existing order engines in the company. These order engines are *SPSE*, that computes orders for single-echelon, single-period and single-item problems, and MP^2, that computes orders for single-echelon, multi-item and multi-period problems. By using the results obtained with these three order engines, we can compare their performances and their ordering behavior on different contexts.

We run experiments with this simulation tool using the three aforementioned order engines on several instances of the case study we work with. Results of these experimentations show that our multi-echelon order engine leads to an ordering behavior that is different from the ones obtained with the single-echelon order engines. In one hand, the amount of orders placed with *ME-LS* are higher, and consequently the costs are also higher. In the other hand however, we observe lower inventory levels and lower lost sales. This means that the multi-echelon approach provides an ordering strategy that leads to a more efficient inventory management. Moreover, the balance between inventory level and lost sales is an important criterion to judge the quality of an ordering strategy, both in the academic inventory control field and in the industrial point of view.

To conclude, the results presented in this chapter show that the multi-echelon order engine we propose in this thesis provides significant advantages for the inventory control problem we want to tackle.

General Conclusion and Perspectives

The research work presented in this manuscript is the result of a thesis focusing on the development of a new optimization method for the inventory control problem. As it is a collaboration between a research laboratory and a company, the main concern along this work was to provide a research that was interesting in an academical point of view, but also easily applicable to industrial contexts. Consequently, a special attention has been given to the problem specifications and the model definition, in order to translate the practical constraints and characteristics faced by inventory managers as best as possible. We also propose a resolution method designed to solve the problem under study, given the specificities of the model, the industrial requirements, and the data volume that real-world contexts imply. We propose a multi-echelon inventory control optimization method, which means that the proposed orders are computed for all entities of the supply system jointly. This makes the problem more difficult to solve, but allows to take into account aspects of the problem that a single-echelon approach cannot tackle.

Summary of the Contributions

Throughout this manuscript, we present the research work conducted during the thesis, and provide contributions on the research topic addressed. We organize the contributions summary according to the different chapters of the manuscript.

In Chapter 1, we focus on the detailed definition of the problem that is studied in the thesis. Based on an extended survey of the literature, we identify a major method used for single-echelon inventory control: Inventory Control Policies (ICP). Although these ICPs are widely used in inventory control optimization, and the literature about ICPs is extensive, we could not find a reference listing the existing ICPs and the contexts in which they are usually applied. To answer this gap, we propose a classification of the main ICPs used in the literature, and a unified notation procedure. We then provide an overview of the state-of-the-art relating to multi-echelon inventory control, and summarize the research gaps that are still to be addressed. The partner company and the existing optimization tools they provide are also detailed, along with the current limitations they expect this thesis to fill. Using both the literature survey and the company's current status, we provide motivations and positioning for our work, and present the research questions addressed in the manuscript.

In Chapter 2, we provide a detailed presentation of the problem specifications and of the mathematical model. We first describe the case study that serves as a basis for the specification of the problem. This results in a list of characteristics and constraints that we want to include in our model, in order to make it applicable in practice for the decision makers. The case study also provides data, that will be used for all experimentations of the thesis. We provide insight on what

the dataset contains, on the assumptions made to complete this dataset, and on the process used to draw instances from this dataset. We then provide the mathematical model built to translate the inventory control problem and the specifications previously listed. Using this model and instances built from the case-study dataset, we provide the results of experimentations on the exact resolution of the problem. These results motivate the use of a heuristic method, as the problem is too difficult to obtain a feasible solution in a reasonable time.

In Chapter 3, the focus is put on the construction of the heuristic method. Based on the problem characteristics, the industrial requirements and the data volume, we identify metaheuristics, and especially the local search, as a suited resolution method. As we do not find relevant neighborhood operators in the literature, we present a list of neighborhood operators we designed specifically for our problem. The first experimentations with these neighborhood operators do not show a satisfying performance for the classic local search. However, we note a significant improvement in the results by using several neighborhood operators in a Variable Neighborhood Search Algorithm. Additionally, first results show that the individual performance of the neighborhood operators varies depending on the moment we use them in the search. This motivates the construction of a new metaheuristic, the *Variable Adaptive Local Search* (*VALS*). This algorithm allows to dynamically change neighborhood operators once an improving solution is found. Furthermore, it uses a non-uniform selection probability for the neighborhood operators, and makes this probability evolve at each use of the neighborhood operators. In this way, the algorithm is able to take into account the variability of the neighborhood operators performances. The benefit of designing such an algorithm for our problem is validated by comparing its performances to the ones of two alternative variable local searches on instances of the case-study's dataset. We also provide the global performance results of the algorithm on the problem under study, showing a significant improvement of the initial solution.

In Chapter 4, we propose to demonstrate the benefits of a multi-echelon inventory control approach by assessing its performances with a business point of view. We identify simulation as a relevant approach for answering the objectives of this chapter, and develop a simulation tool aiming at reproducing the inventory control process as good as possible on a given period of time. With this tool, we can mimic the real-world inventory dynamic, by simulating the customer demand, the receptions, and the order computations using different inventory control optimization methods. We describe the simulation tool in details, and separate the complete simulation process into sub-processes. We then explain how every sub-process work, and how it interacts with the others. Using this simulation tool, we conduct experiments to assess the performances of our proposed method in a business point of view, and to compare these performances to the ones achieved by the already existing inventory control optimization methods in the partner company. These experiments show a significant benefit of the multi-echelon approach, as it allows to improve the balance between inventory levels and shortage levels, which is the most important aspect in inventory control optimization, although better balance implies an increase of the total costs. The main conclusion of these experiments is that the multi-echelon approach allow a better inventory management by purchasing more items, but these items are more relevant, as this approach allow to store less products and to satisfy more customers.

Perspectives

Based on the limitations we identified in the work presented, on the future research directions we consider promising, and on the future inventory control optimization functions Vekia wants to offer, we identify perspectives to extend the research conducted during this thesis. As the CIFRE contract will lead to a full time research engineer position, we will be able to continue the work on this project. After the end of the thesis, the priority will be the industrialization of our proposed resolution method, so that it can be included in the software offered by the company. In the following, we detail the other research perspectives identified, sorted according to the time scale required.

Short-Term Perspectives - The perspectives presented in this part relate mainly to technical improvements, or new experimentations. We consider them to be small enhancements of the solution presented throughout this work.

The first aspect we want to consider is the initial solution generation for the optimization algorithm. In this dissertation, we propose a protocol for generating a feasible initial solution, based on repairing a solution obtained with continuous optimization. This process is practical, and allow to build a solution by using the mathematical model proposed for the problem. However, its execution time is significant, and it results in an initial solution that can be far away from the optimal value. Moreover, we have the intuition that, if we give an initial solution that is closer from the optimal to the algorithm, it may result in an even better global performance. Consequently, we want to explore other ways of generating initial solutions, for example by using Inventory Control Policies to generate an ordering plan.

Then, we also mentioned in the dissertation that the search space of the problem is very large. This is due to the high number of decision variables, and to the large size of the industrial datasets we have to manage. However, there is in the dataset a high number of suppliers for a lot of items. In this thesis, we wanted to include the complete dataset to let the resolution method choose the best options. In practice, we know that all suppliers are not requested. Consequently, we could add a supplier selection process before the optimization process, and keep in the model only a reduced number of suppliers that are most likely to be solicited. In doing so, we can significantly reduce the number of variables, and consequently make the search space smaller. This can also reduce the execution time of the cost function calculation.

Finally, the experimentations presented in this manuscript are based on a case-study. Although this case study is representative of many real-world inventory control contexts, the dataset we use for our experiments is specific to one situation. In order to have more extensive results and conclusions, we want to use other data to validate the proposed solution. We already identified candidate contexts for this part, and we need to work on their data to build a compliant dataset for our future experiments.

The perspectives presented here are extensions of the work we present in this thesis, and they do not require changes in the structure of the resolution algorithm. Consequently, we estimate a low necessary workload to achieve them, except the time required for data management. We plan to achieve the aforementioned perspective before or during the industrialization phase of the algorithm.

Medium-Term Perspectives - In this part, we present perspectives that mainly concern extensions of the model to make it closer to real-world contexts. Throughout this thesis, we put effort in

the model definition to take into account the business constraints and characteristics faced by the inventory managers on a daily basis. This was motivated by the objective of proposing a solution that could be easily applicable in practice. However, we still had to make restrictive assumptions, or to set aside several aspects. We identify restrictions that are the most relevant for the further development of the multi-echelon order engine.

First, we mentioned that Vekia provides probabilistic forecast for the customer demand. In this thesis, we propose a model that takes deterministic input data, and use only the average of the probabilistic forecast provided. This leads to a model that is clearer, and less complex, but we know that a probabilistic forecast provides more relevant data on the estimated customer demand. As a result, a model that can use such information may propose more relevant orders. Consequently, we consider to make the model able to use and optimize probabilistic input data, so that we can exploit the complete forecast provided by Vekia. Such an evolution in the model can also enable another extension we consider relevant, which is to relax the assumption on the perfect suppliers.

Indeed, we consider in this research that the suppliers have an infinite inventory capacity, and that they deliver every order placed with no uncertainty on the delay or on the quantity delivered. We know that this assumption is rarely true in practice. Therefore, if we extend the model to enable the input of probabilistic data, we can also use it to add uncertainty on the supply, either in the lead times or in the quantity delivered. Taking this into account can also help the model to propose more relevant orders.

Finally, the model we propose computes unit quantity to order, and we know that there exist many contexts where items are only available in packs. This limitation can be addressed with a simple heuristic transforming the unit quantities recommended into a set of available packaging. However, we consider to include the packagings information directly into the model, so that the optimization can use this aspect.

All of these perspectives require a substantial amount of work, as it demands to make changes in the model, and this may affect the resolution method. However, the global structure of the problem stays the same, and consequently we estimate the workload to stay reasonable. As a result, we plan to integrate the aforementioned perspectives as updates of the industrialized version of the algorithm.

Long-Term Perspectives - Let us now propose perspectives for a more distant horizon. As we explained in the manuscript, the model and solution method we proposed are specifically designed for a One-Warehouse Multiple Retailers supply network. This kind of supply chain is the most common in the contexts treated by Vekia, and more generally in distribution systems. As one of the main motivations of our work was to produce optimisation tools that would be easily applicable in practice, this modeling decision was the most relevant in the context of the thesis.

In a more general context, there exist other supply network types, and the model we propose is not able to tackle them. A first interesting extension would be to enable the presence of several warehouses. In theory, this would require only a small amount of work to make the model compliant with such networks. Still, we know that every modification in the structure of the model requires a large amount of time to test and debug the order engine. Moreover, we do not have any data for now that we could use to test the model. Then, this would be relevant to make the model more generic and enable the presence of one, or even several, echelons to the supply network. For example, we can encounter distribution networks with a national warehouse supplying regional warehouses, that supply retailers. This kind of network is completely excluded at the current state of the model. However, including more echelons require a substantial change in the structure of the model and of

the resolution method.

Finally, the model we propose in this work includes a given set of constraints, that we selected so that the model can be applicable to the most possible cases. However, we know that each business context is unique, and may have its own set of constraints. Consequently, it would be interesting to propose a model and resolution method that could be more modular. Such a method could include a process for easily activate or deactivate constraints, and even add new ones, depending on the context. Such an extension would require a complete new reflexion on the structure of the problem, the model and the resolution method. As a result, this may require to build a whole research project around this thematic.

Both of the perspectives we mentioned above require a complete rework of the model and algorithm we proposed for this thesis. Consequently, this represents a large amount of work, and may even require the development of a new order engine. As a result, this may be part of a dedicated research project, and it is not considered for now in the roadmap.

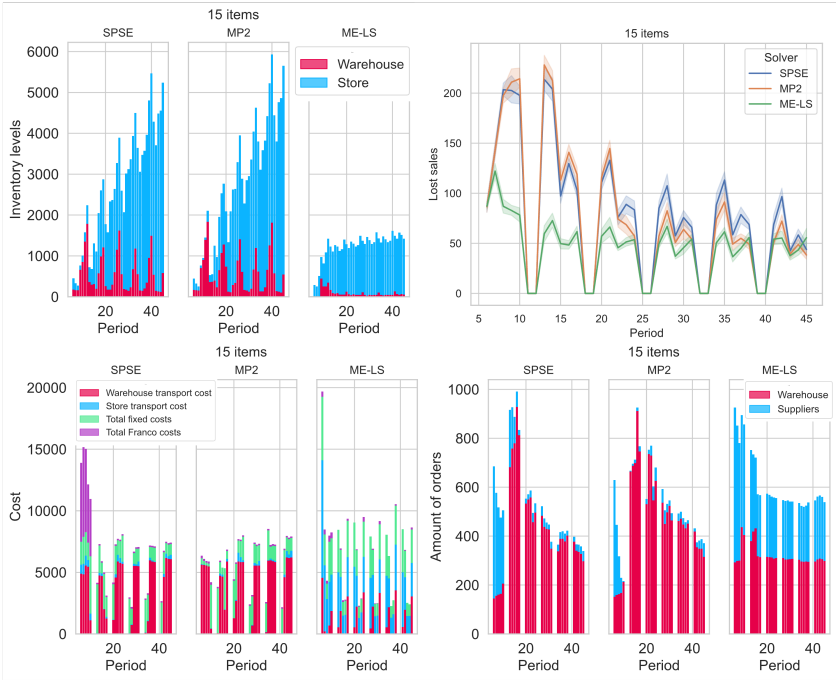
Short-Term	Medium-Term	Long-term
Extended algorithm	**Additional constraints**	**New model structure**
→ New initial solution	→ Probabilistic forecast	→ Several warehouses
→ Reducing search space	→ Supplier uncertainty	→ More echelons
→ New case study	→ Packaging constraints	→ Modular constraints

Figure 4.11: Timeline and summary of the perspectives.

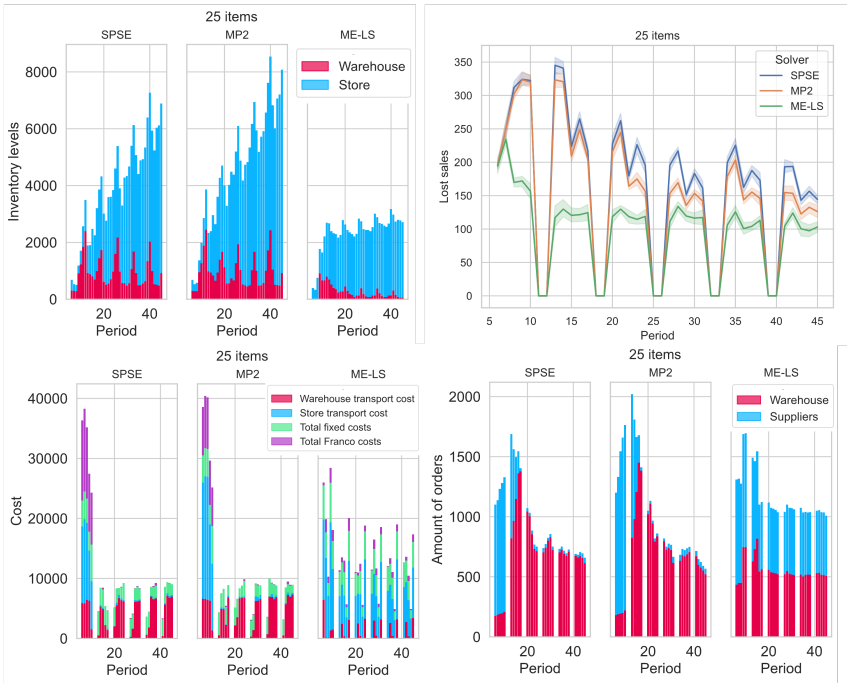
Appendix A

Simulation Extensive Results

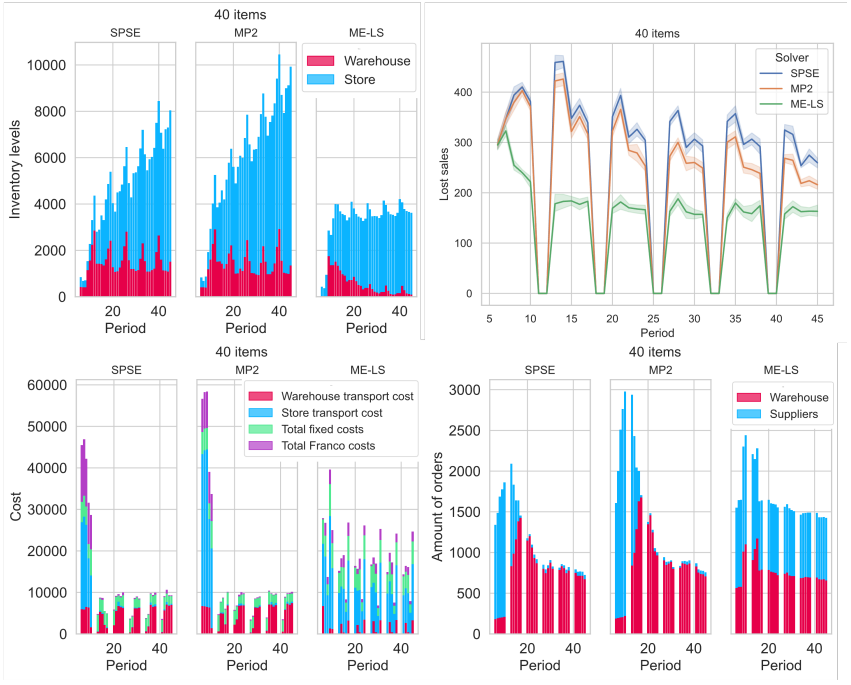
Instance: (fora, 15)



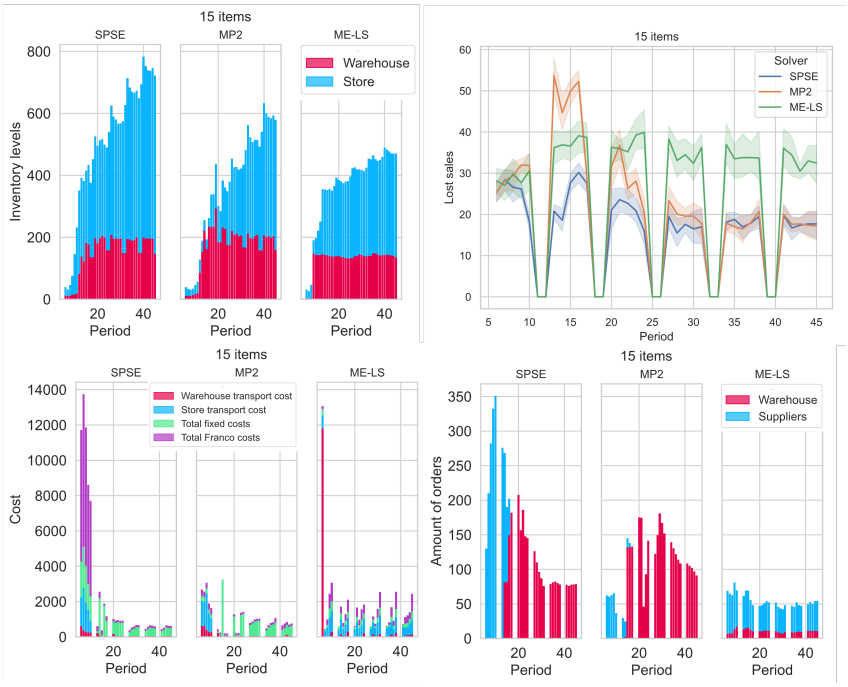
Instance: (fora, 25)



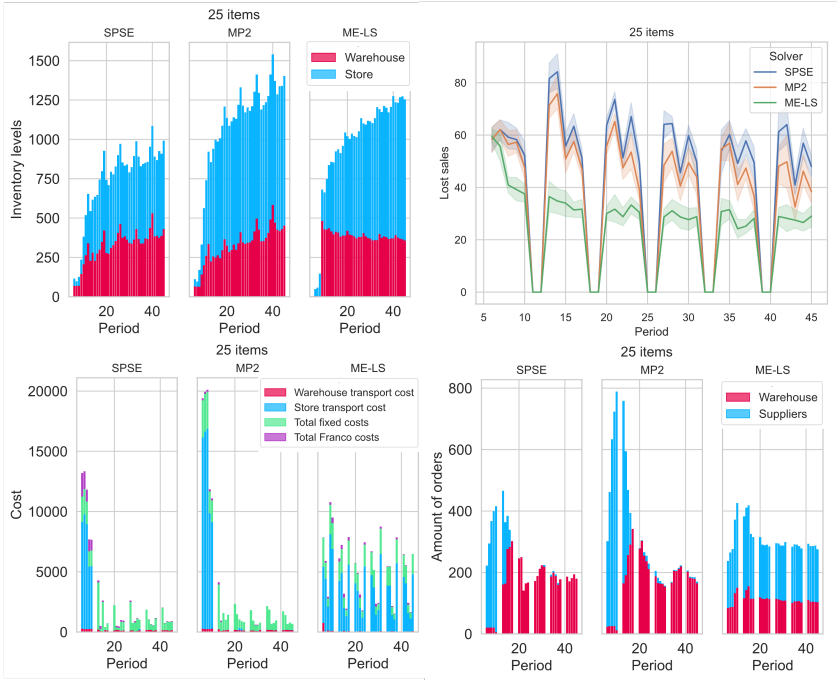
Instance: (fora, 40)



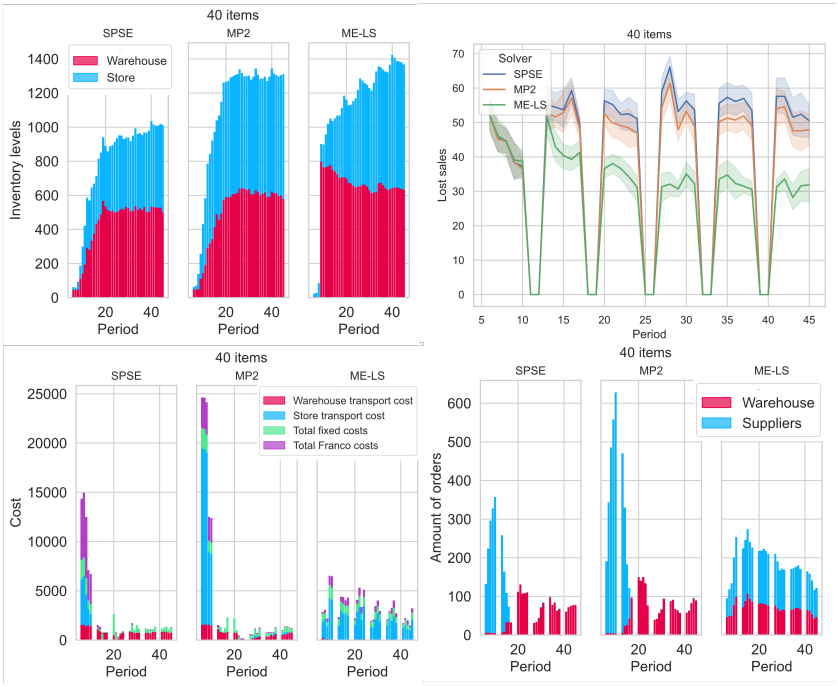
Instance: (rand1, 15)



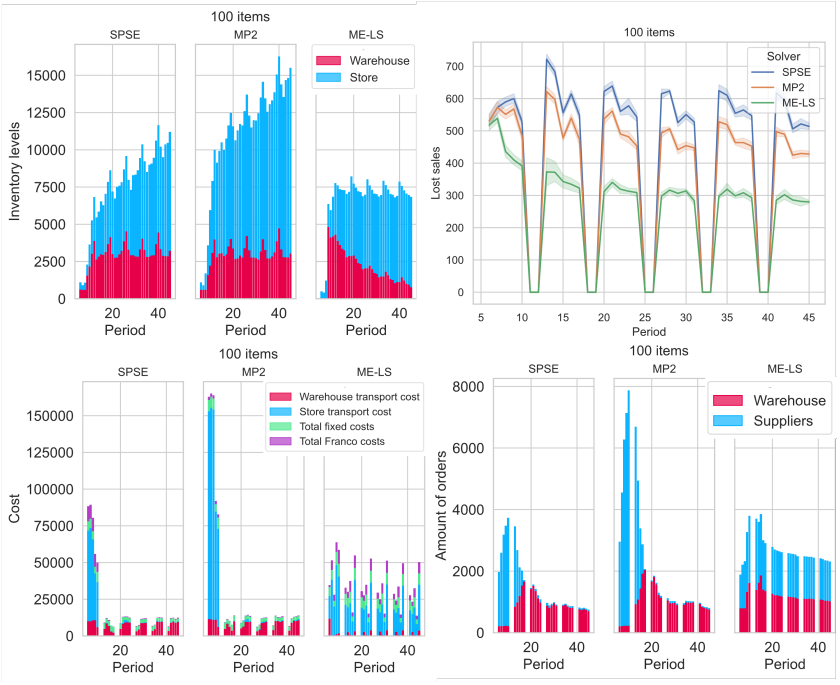
Instance: (rand1, 25)



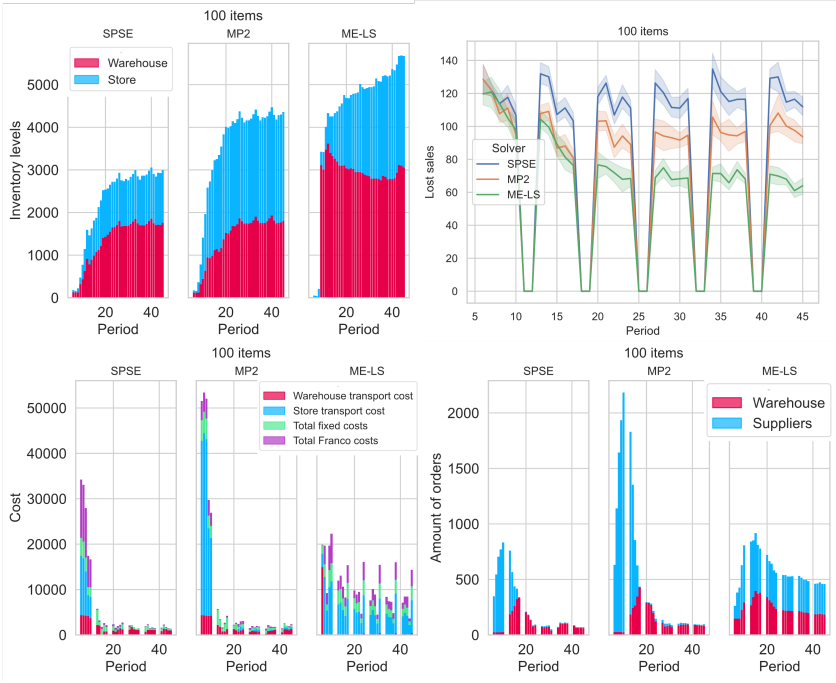
Instance: (rand1, 40)



Instance: (fora, 100)



Instance: (rand1, 100)



Appendix B

Alternative Variable Local Searches

Algorithm 7 Uniform Variable Local Search

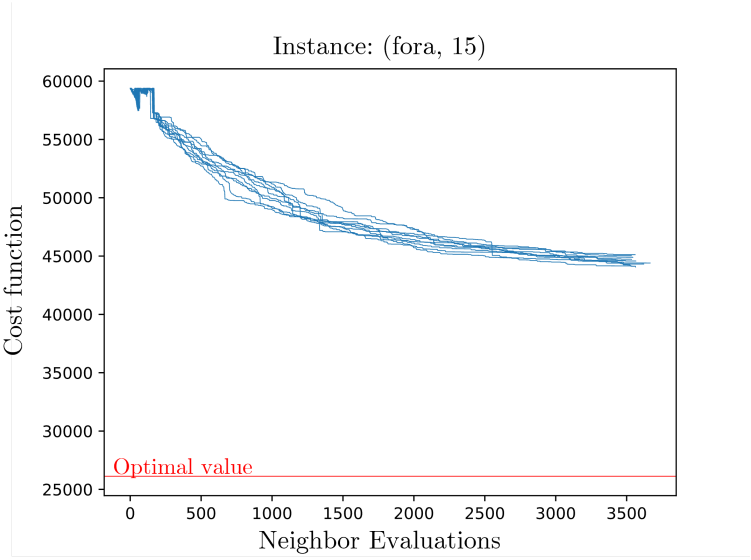
1: **Input:** Set of neighborhood operators U, Global termination criterion TC, Local termination criterion tc, Initial solution S
2: Compute selection probability distribution $\forall u \in U,\ \mathbf{P}_u = \frac{1}{|U|}$
3: **while** not TC **do**
4: Select a neighborhood operator $u \in U$ following the selection probabilities
5: **while** not tc **do**
6: Generate a candidate neighbor S'
7: **if** S' is better than S **then**
8: $S \leftarrow S'$
9: $tc \leftarrow True$
10: **end if**
11: **end while**
12: **end while**

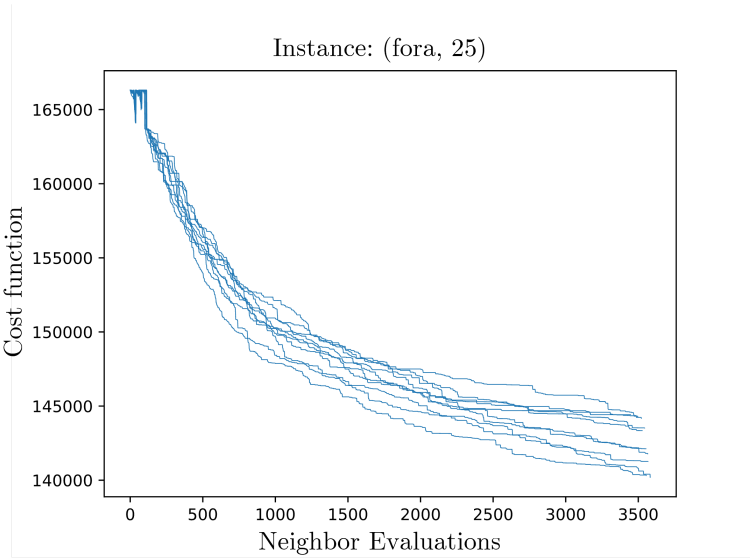
Algorithm 8 Variable Local Search

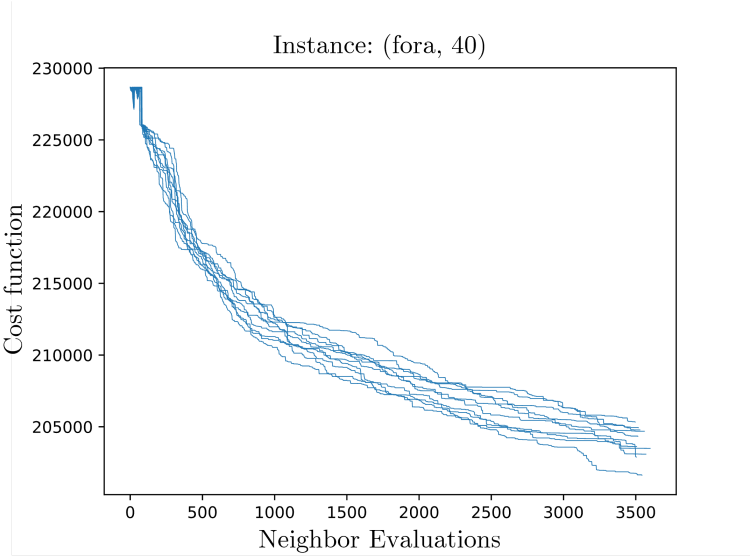
Input: Set of neighborhood operators U, Scores of neighborhood operators $G = \{G_u\}_{u \in U}$, Global termination criterion TC, Local termination criterion tc, Initial solution S
Compute selection probability distribution
while not TC **do**
    Select a neighborhood operator $u \in U$ following the selection probabilities
    while not tc **do**
        Generate a candidate neighbor S'
        if S' is better than S **then**
            $S \leftarrow S'$
            $tc \leftarrow True$
        end if
    end while
end while

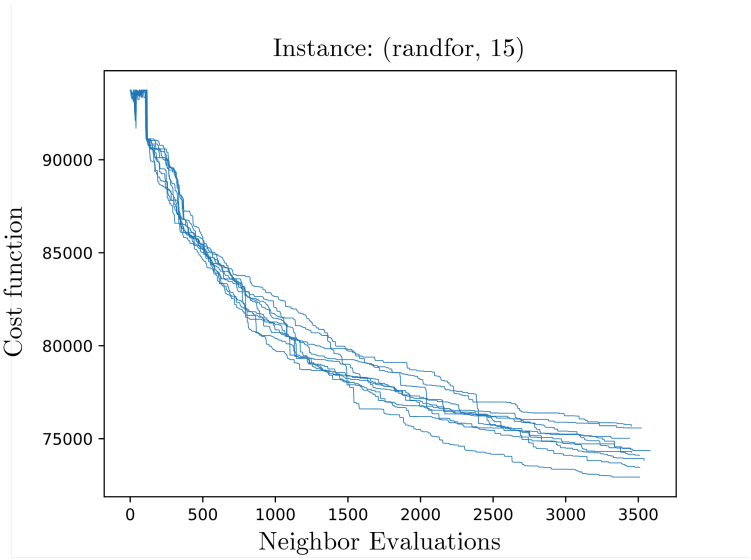
Appendix C

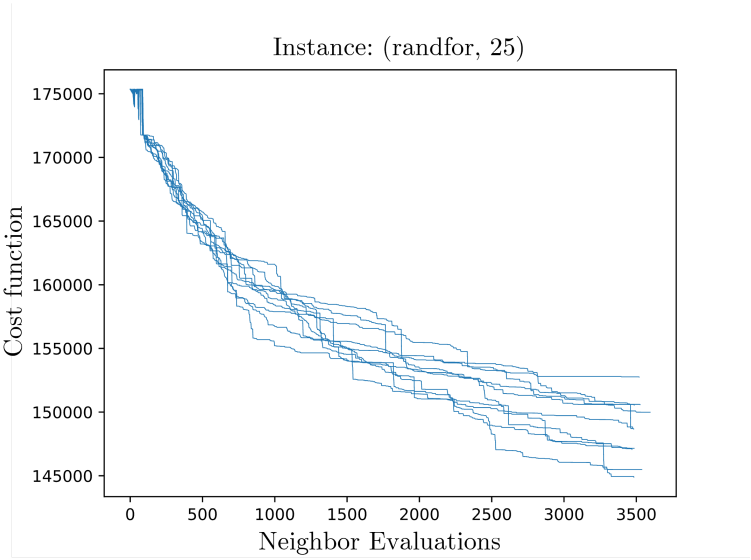
Convergence curves for the experimentations with *VALS*

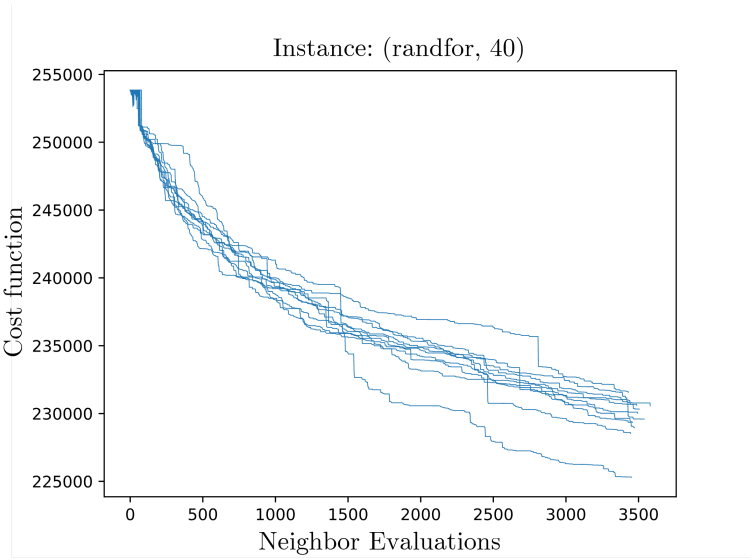


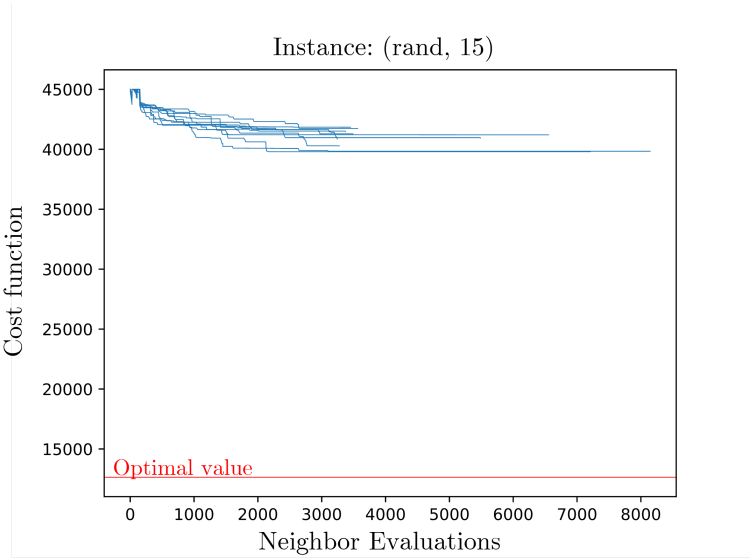

Instance: (fora, 25)
Cost function
165000
160000
155000
150000
145000
140000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations

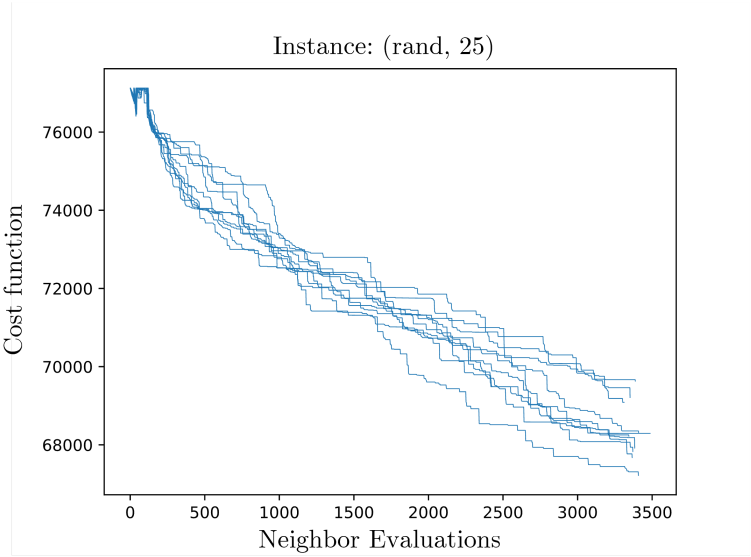

Instance: (fora, 40)
Cost function
230000
225000
220000
215000
210000
205000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations

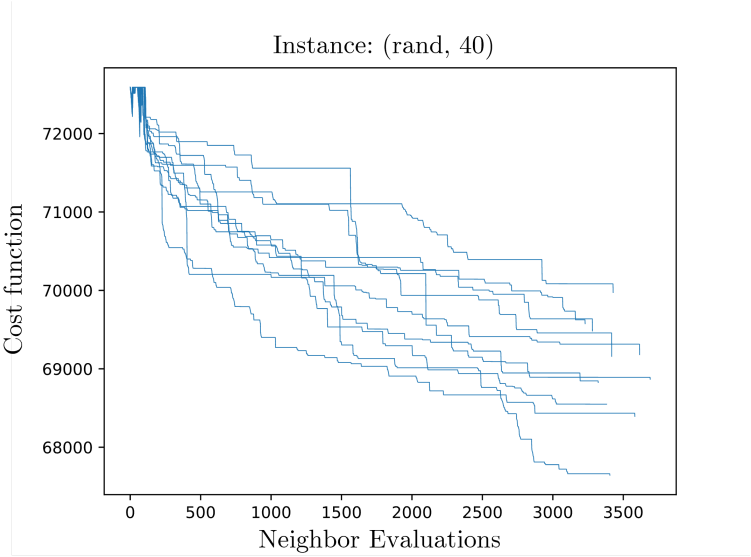

Instance: (randfor, 15)
Cost function
90000
85000
80000
75000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations


Instance: (randfor, 25)
Cost function
175000
170000
165000
160000
155000
150000
145000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations


Instance: (randfor, 40)
Cost function
255000
250000
245000
240000
235000
230000
225000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations


Instance: (rand, 15)
Cost function
45000
40000
35000
30000
25000
20000
15000
Optimal value
0
1000
2000
3000
4000
5000
6000
7000
8000
Neighbor Evaluations


Instance: (rand, 25)
Cost function
76000
74000
72000
70000
68000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations


Instance: (rand, 40)
Cost function
72000
71000
70000
69000
68000
0
500
1000
1500
2000
2500
3000
3500
Neighbor Evaluations

Bibliography

Sumer C Aggarwal. A review of current inventory theory and its applications. *International Journal of Production Research*, 12(4):443–482, 1974. URL https://doi.org/10.1080/00207547408919568.

Ben-Tal Aharon, Golany Boaz, and Shtern Shimrit. Robust multi-echelon multi-period inventory control. *European Journal of Operational Research*, 199(3):922–935, 2009. URL https://doi.org/10.1016/j.ejor.2009.01.058.

Mohammadmahdi Alizadeh, Hamidreza Eskandari, and S. Mehdi Sajadifar. A modified (s-1,s) inventory system for deteriorating items with poisson demand and non-zero lead time. *Applied Mathematical Modelling*, 38(2):699–711, 2014. URL http://doi.org/10.1016/j.apm.2013.07.014.

Poul Alstrøm and Per Madsen. Simulation of inventory control systems. *International Journal of Production Economics*, 26(1-3):125–134, 1992. URL https://doi.org/10.1016/0925-5273(92)90054-B.

Elisa M Alvarez, Matthieu C van der Heijden, and W Henk M Zijm. Service differentiation in spare parts supply through dedicated stocks. *Annals of Operations Research*, 231:283–303, 2015. URL https://doi.org/10.1007/s10479-013-1362-z.

Alessandro Andriolo, Daria Battini, Robert W Grubbström, Alessandro Persona, and Fabio Sgarbossa. A century of evolution from harris's basic lot size model: Survey and research agenda. *International Journal of Production Economics*, 155:16–38, 2014. URL https://doi.org/10.1016/j.ijpe.2014.01.013.

Emel Arıkan, Johannes Fichtinger, and Jörg M Ries. Impact of transportation lead-time variability on the economic and environmental performance of inventory systems. *International Journal of Production Economics*, 157:279–288, 2014. URL https://doi.org/10.1016/j.ijpe.2013.06.005.

Kostas Platon Aronis, Ioulia Magou, Rommert Dekker, and George Tagaras. Inventory control of spare parts using a bayesian approach: A case study. *European Journal of Operational Research*, 154(3):730–739, 2004. URL http://doi.org/10.1016/S0377-2217(02)00837-8.

Kenneth J Arrow, Theodore Harris, and Jacob Marschak. Optimal inventory policy. *Econometrica: Journal of Econometric Society*, pages 250–272, 1951. URL https://doi.org/10.2307/1906813.

Sven Axsäter. A framework for decentralized multi-echelon inventory control. *IIE Transactions*, 33 (2):91–97, 2001. URL https://doi.org/10.1023/A:1007694700789.

Sven Axsäter and Wen-Fa Zhang. A joint replenishment policy for multi-echelon inventory control. *International Journal of Production Economics*, 59(1-3):243–250, 1999. URL https://doi.org/10.1016/S0925-5273(98)00018-8.

Monique Bakker, Jan Riezebos, and Ruud H Teunter. Review of inventory systems with deterioration since 2001. *European Journal of Operational Research*, 221(2):275–284, 2012. URL https://doi.org/10.1016/j.ejor.2012.03.004.

Joseph L. Balintfy. On a basic class of multi-item inventory problems. *Management Science*, 10(2): 287–297, 1964. URL http://doi.org/10.1287/mnsc.10.2.287.

Jerry Banks. Introduction to simulation. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 7–13, 1999. URL https://doi.org/10.1145/324138.324142.

Jerry Banks and César O Malavé. The simulation of inventory systems: An overview. *Simulation*, 42(6):283–290, 1984. URL https://doi.org/10.1177/003754978404200605.

Rob JI Basten, Matthieu C Van der Heijden, Johannes MJ Schutten, and Erhan Kutanoglu. An approximate approach for the joint problem of level of repair analysis and spare parts stocking. *Annals of Operations Research*, 224:121–145, 2015. URL https://doi.org/10.1007/s10479-012-1188-0.

Emre Berk and Ülkü Gürler. Analysis of the (q, r) inventory model for perishables with positive lead times and lost sales. *Operations Research*, 56(5):1238–1246, 2008. URL https://doi.org/10.1287/opre.1080.0582.

Peter Berling and Mojtaba Farvid. Lead-time investigation and estimation in divergent supply chains. *International Journal of Production Economics*, 157(1):177–189, 2014. URL http://doi.org/10.1016/j.ijpe.2014.04.030.

Peter Berling and Johan Marklund. Multi-echelon inventory control: An adjusted normal demand model for implementation in practice. *International Journal of Production Research*, 52(11): 3331–3347, 2014. URL http://doi.org/10.1080/00207543.2013.873555.

Peter Berling and Victor Martínez-de Albéniz. Dynamic speed optimization in supply chains with stochastic demand. *Transportation Science*, 50(3):1114–1127, 2016. URL https://doi.org/10.1287/trsc.2014.0561.

Luca Bertazzi, Giuseppe Paletta, and M. Grazia Speranza. Deterministic order-up-to level policies in an inventory routing problem. *Transportation Science*, 36(1):119–132, 2002. URL https://doi.org/10.1287/trsc.36.1.119.573.

Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin

Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. Enabling research through the scip optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2), jun 2023. ISSN 0098-3500. URL https://doi.org/10.1145/3585516.

Swapnil Bhalla, Erlend Alfnes, Hans-Henrik Hvolby, and Fabio Sgarbossa. Advances in spare parts classification and forecasting for inventory control: A literature review. *IFAC-PapersOnLine*, 54 (1):982–987, 2021. URL https://doi.org/10.1016/j.ifacol.2021.08.118.

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, 2012a. URL http://doi.org/10.1016/j.ejor.2012.02.013.

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, 2012b. ISSN 14769360. URL http://doi.org/10.1057/jors.2011.52.

Diane P. Bischak, David J. Robb, Edward A. Silver, and Joseph D. Blackburn. Analysis and management of periodic review, order-up-to level inventory systems with order crossover. *Production and Operations Management*, 23(5):762–772, 2014. URL https://doi.org/10.1111/poms.12072.

Youssef Boulaksil. Safety stock placement in supply chains with demand forecast updates. *Operations Research Perspectives*, 3:27–31, 2016. URL https://doi.org/10.1016/j.orp.2016.07.001.

Rob A.C.M. Broekmeulen and Karel H. van Donselaar. A heuristic to manage perishable inventory with batch ordering, positive lead-times, and time-varying demand. *Computers and Operations Research*, 36(11):3013–3018, 2009. URL http://doi.org/10.1016/j.cor.2009.01.017.

Karen L Brown, Anthony R Inman, and James A Calloway. Measuring the effects of inventory inaccuracy in mrp inventory and delivery performance. *Production Planning & Control*, 12(1): 46–57, 2001. URL https://doi.org/10.1080/09537280150203979.

Maxim A Bushuev, Alfred Guiffrida, MY Jaber, and Mehmood Khan. A review of inventory lot sizing review papers. *Management Research Review*, 38(3):283–298, 2015. URL https://doi.org/10.1108/MRR-09-2013-0204.

Alessandro Cestana, Erica Pastore, Arianna Alfieri, and Andrea Matta. Reducing resupply time with additive manufacturing in spare part supply chain. *IFAC-PapersOnLine*, 52(13):577–582, 2019. URL http://doi.org/10.1016/j.ifacol.2019.11.220.

Xiuli Chao, Stefanus Jasin, and Sentao Miao. Adaptive algorithms for multi-warehouse multi-store inventory system with lost sales and fixed replenishment cost. *SSRN*, 2021.

Xin Chen and David Simchi-Levi. Coordinating inventory control and pricing strategies with random demand and fixed ordering cost: The finite horizon case. *Operations Research*, 52(6):887–896, 2004. URL https://doi.org/10.1287/opre.1040.0127.

Xin Chen and Peng Sun. Optimal structural policies for ambiguity and risk averse inventory and pricing models. *SIAM Journal on Control and Optimization*, 50(1):133–146, 2012. URL https://doi.org/10.1137/100791488.

Lifei Cheng, Eswaran Subrahmanian, and Arthur W. Westerberg. Multi-objective decisions on capacity planning and production-inventory control under uncertainty. *Industrial and Engineering Chemistry Research*, 43(9):2192–2208, 2004. URL https://doi.org/10.1021/ie0303404.

Chi Chiang. An order expediting policy for continuous review systems with manufacturing lead-time. *European Journal of Operational Research*, 203(2):526–531, 2010. URL http://dx.doi.org/10.1016/j.ejor.2009.08.012.

Sunil Chopra, Peter Meindl, and Dharam Vir Kalra. *Supply chain management: strategy, planning, and operation*, volume 232. MA: Pearson Boston, 2013.

Ioannis T. Christou, Konstantina Skouri, and Anathasios G. Lagodimos. Fast evaluation of a periodic review inventory policy. *Computers and Industrial Engineering*, 144, 2020. URL http://doi.org/10.1016/j.cie.2020.106389.

Yunfei Chu, Fengqi You, John M Wassick, and Anshul Agarwal. Simulation-based optimization framework for multi-echelon inventory systems under uncertainty. *Computers & Chemical Engineering*, 73:1–16, 2015. URL https://doi.org/10.1016/j.compchemeng.2014.10.008.

Kun Jen Chung, Suresh Kumar Goyal, and Yung Fu Huang. The optimal inventory policies under permissible delay in payments depending on the ordering quantity. *International Journal of Production Economics*, 95(2):203–213, 2005. URL https://doi.org/10.1016/j.ijpe.2003.12.006.

Andrew J. Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management Science*, 6(4):475–490, 1960. URL http://doi.org/10.1287/mnsc.1040.0265.

Bo Dai, Haoxun Chen, Yuan Li, Yidong Zhang, Xiaoqing Wang, and Yuming Deng. Inventory replenishment planning of a distribution system with storage capacity constraints and multi-channel order fulfilment. *Omega*, 102, 2021. URL https://doi.org/10.1016/j.omega.2020.102356.

Ton De Kok, Christopher Grob, Marco Laumanns, Stefan Minner, Jörg Rambau, and Konrad Schade. A typology and literature review on stochastic multi-echelon inventory models. *European Journal of Operational Research*, 269(3):955–983, 2018. URL https://doi.org/10.1016/j.ejor.2018.02.047.

Niels De Smet, El Houssaine Aghezzaf, and Bram Desmet. Optimising installation (r,q) policies in distribution networks with stochastic lead times: a comparative analysis of guaranteed- and stochastic service models. *International Journal of Production Research*, 57(13):4148–4165, 2019. URL https://doi.org/10.1080/00207543.2018.1518606.

Matthew J Drake and Kathryn A Marley. A century of the eoq. In *Handbook of EOQ Inventory Problems: Stochastic and Deterministic Models and Applications*, pages 3–22. Springer, 2013. URL https://doi.org/10.1007/978-1-4614-7639-9.

Melvin Drent and Joachim Arts. Expediting in two-echelon spare parts inventory systems. *Manufacturing & Service Operations Management*, 23(6):1431–1448, 2021. URL https://doi.org/10.1287/msom.2020.0888.

Qinglin Duan and T. Warren Liao. A new age-based replenishment policy for supply chain inventory optimization of highly perishable products. *International Journal of Production Economics*, 145 (2):658–671, 2013. URL http://dx.doi.org/10.1016/j.ijpe.2013.05.020.

Banu Y. Ekren and Bartu Arslan. Simulation-based lateral transshipment policy optimization for s, s inventory control problem in a single-echelon supply chain network. *International Journal of Optimization and Control: Theories and Applications*, 10(1):9–16, 2020. URL http://doi.org/10.11121/ijocta.01.2020.00789.

Mohsen ElHafsi, Jianxin Fang, and Essia Hamouda. Optimal production and inventory control of multi-class mixed backorder and lost sales demand class models. *European Journal of Operational Research*, 291(1):147–161, 2021. URL https://doi.org/10.1016/j.ejor.2020.09.009.

Nazanin Esmaili, Brian A. Norman, and Jayant Rajgopal. Exact analysis or (r,s,s) inventory control systems with lost sales and zero lead-time. *Naval Research Logistics*, 66(2):123–132, 2019. URL https://doi.org/10.1002/nav.21833.

Mohammad Fattahi, Masoud Mahootchi, S. Mohammad Moattar Husseini, Eesmail Keyvanshokooh, and Farid Alborzi. Investigating replenishment policies for centralised and decentralised supply chains using stochastic programming approach. *International Journal of Production Research*, 53(1):41–69, 2015. URL https://doi.org/10.1080/00207543.2014.922710.

George J. Feeney and Craig C. Sherbrooke. The (s-1,s) inventory policy under compound poisson demand. *Management Science*, 12(5):391–413, 1966. URL https://doi.org/10.1287/mnsc.12.5.391.

Jiejian Feng, Liming Liu, and Yat-Wah Wan. Irreducibility of joint inventory positions in an assemble-to-order system under (r, nq) policies. *Naval Research Logistics*, 59(1):18–26, 2011. URL https://doi.org/10.1002/nav.20486.

Elgar Fleisch and Christian Tellkamp. Inventory inaccuracy and supply chain performance: a simulation study of a retail supply chain. *International Journal of Production Economics*, 95(3): 373–385, 2005. URL https://doi.org/10.1016/j.ijpe.2004.02.003.

John Forrest and Robin Lougee-Heimer. Cbc user guide. In *Emerging theory, methods, and applications*, pages 257–277. INFORMS, 2005. URL https://doi.org/10.1287/educ.1053.0020.

Herdert P. Galliher, Philip M. Morse, and M. Simond. Dynamics of two classes of continuous-review inventory systems. *Operations Research*, 7(3):362–384, 1959. URL https://doi.org/10.1287/opre.7.3.362.

Ram Ganeshan, Tonya Boone, and Alan J Stenger. The impact of inventory and flow planning parameters on supply chain performance: an exploratory study. *International Journal of Production Economics*, 71(1-3):111–118, 2001. URL https://doi.org/10.1016/S0925-5273(00)00109-2.

Shashank Garg, Diatha Krishna Sundar, and Kuriakkattil Ravikumar. A periodic tabular policy for scheduling of a single stage production-inventory system. *Computers and Industrial Engineering*, 62(1):21–28, 2012. URL http://doi.org/10.1016/j.cie.2011.08.013.

Christoph H Glock. A multiple-vendor single-buyer integrated inventory model with a variable number of vendors. *Computers & Industrial Engineering*, 60(1):173–182, 2011. URL https://doi.org/10.1016/j.cie.2010.11.001.

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. URL https://doi.org/10.1016/j.ijpe.2013.12.009.

Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

Mustafa Göcken and Ayse Togba Dosdogru. (r, s, s) inventory control policy and supplier selection in a two-echelon supply chain: an optimization via simultaion. *Proceedings of the 2015 Winter Simulation Conference*, pages 2057–2067, 2015. URL https://doi.org/10.1109/WSC.2015.7408320.

Suresh K Goyal and Ahmet T Satir. Joint replenishment inventory control: deterministic and stochastic models. *European Journal of Operational Research*, 38(1):2–13, 1989. URL https://doi.org/10.1016/0377-2217(89)90463-3.

Jörn Grahl, Stefan Minner, and Daniel Dittmar. Meta-heuristics for placing strategic safety stock in multi-echelon inventory with differentiated service times. *Annals of Operations Research*, 242:489–504, 2016. URL https://doi.org/10.1007/s10479-014-1635-1.

Jovan Grahovac and Amiya Chakravarty. Sharing and lateral transshipment of inventory in a supply chain with expensive low-demand items. *Management Science*, 47(4):579–594, 2001. URL http://doi.org/10.1287/mnsc.47.4.579.9826.

Joshua Grass and Shlomo Zilberstein. Anytime algorithm development tools. *ACM SIGART Bulletin*, 7(2):20–27, 1996. URL https://doi.org/10.1145/242587.242592.

Stephen C. Graves. Multi-echelon inventory model for a repairable item with one-for-one eplenishment. *Management Science*, 31(10):1247–1256, 1985. URL http://doi.org/10.1287/mnsc.31.10.1247.

Aljoscha Gruler, Javier Panadero, Jesica de Armas, José A Moreno Pérez, and Angel A Juan. Combining variable neighborhood search with simulation for the inventory routing problem with stochastic demands and stock-outs. *Computers & Industrial Engineering*, 123:278–288, 2018.

Mario Guajardo, Mikael Rönnqvist, Ann Mari Halvorsen, and Svein Inge Kallevik. Inventory management of spare parts in an energy company. *Journal of the Operational Research Society*, 66(2):331–341, 2015. ISSN 14769360. URL http://10.1057/jors.2014.8.

Miguel Gutierrez and Francisco A. Rivera. Undershoot and order quantity probability distributions in periodic review, reorder point, order-up-to-level inventory systems with continuous demand. *Applied Mathematical Modelling*, 91:791–814, 2021. URL https://doi.org/10.1016/j.apm.2020.09.014.

George Hadley and Thomson M. Whitin. A family of inventory models. *Management Science*, 8 (4):458–469, 1962. ISSN 0025-1909. URL http://doi.org/10.1287/mnsc.8.4.458.

René Haijema, Jan van der Wal, and Nico M. van Dijk. Blood platelet production: Optimization by dynamic programming and simulation. *Computers and Operations Research*, 34(3):760–779, 2007. URL https://doi.org/10.1016/j.cor.2005.03.023.

Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A Moreno Pérez. *Variable neighborhood search*. Springer, 2019.

Ford W. Harris. How many parts to make at once. *Factory, the magazine of management*, 1913.

Warren H Hausman and Nesim K Erkip. Multi-echelon vs. single-echelon inventory control policies for low-demand items. *Management Science*, 40(5):597–602, 1994. URL https://doi.org/10.1287/mnsc.40.5.597.

Pascale A Hayek and Moueen K Salameh. Production lot sizing with the reworking of imperfect quality items produced. *Production Planning & Control*, 12(6):584–590, 2001. URL https://doi.org/10.1080/09537280175039770 7.

N Grace Hua and Sean P Willems. Analytical insights into two-stage serial line supply chain safety stock. *International Journal of Production Economics*, 181:107–112, 2016. URL https://doi.org/10.1016/j.ijpe.2015.10.010.

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. URL https://doi.org/10.1287/mnsc.1080.0945.

Faizul Huq, Kenneth Cutright, Vernon Jones, and Douglas A Hensler. Simulation study of a two-level warehouse inventory replenishment system. *International Journal of Physical Distribution & Logistics Management*, 36(1):51–65, 2006. URL https://doi.org/10.1108/09600030610642931.

CPLEX IBM ILOG. V22. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2023.

M. Ali Ilgin and Semra Tunali. Joint optimization of spare parts inventory and maintenance policies using genetic algorithms. *International Journal of Advanced Manufacturing Technology*, 34(5-6): 594–604, 2007. URL http://doi.org/10.1007/s00170-006-0618-z.

Mohamad Y. Jaber, Suresh K. Goyal, and Muhammad Imran. Economic production quantity model for items with imperfect quality subject to learning effects. *International Journal of Production Economics*, 115(1):143–150, 2008. URL https://doi.org/10.1016/j.ijpe.2008.05.007.

Ilya Jackson, Jurijs Tolujevs, and Zhandos Kegenbekov. Review of inventory control models: a classification based on methods of obtaining optimal control parameters. *Transport and Telecommunication Journal*, 21(3):191–202, 2020. URL https://doi.org/10.2478/ttj-2020-0015.

Hamed Jalali and Inneke Van Nieuwenhuyse. Simulation optimization in inventory replenishment: a classification. *IIE Transactions*, 47(11):1217–1235, 2015. URL https://doi.org/10.1080/0740817X.2015.1019162.

Abul M.M. Jamal, Bhaba R. Sarker, and Sanjay Mondal. Optimal manufacturing batch size with rework process at a single-stage production system. *Computers and Industrial Engineering*, 47 (1):77–89, 2004. URL https://doi.org/10.1016/j.cie.2004.03.001.

Søren Glud Johansen and Roger M. Hill. (r, q) control of a periodic-review inventory system with continuous demand and lost sales. *International Journal of Production Economics*, 68(3):279–286, 2000. URL http://doi.org/10.1016/S0925-5273(00)00051-7.

June Young Jung, Gary Blau, Joseph F Pekny, Gintaras V Reklaitis, and David Eversdyk. A simulation based optimization approach to supply chain management under demand uncertainty. *Computers & Chemical Engineering*, 28(10):2087–2106, 2004. URL https://doi.org/10.1016/j.compchemeng.2004.06.006.

Seong-Cheol Kang, Theodora S Brisimi, and Ioannis Ch Paschalidis. Distribution-dependent robust linear optimization with applications to inventory control. *Annals of Operations Research*, 231 (1):229–263, 2015. URL https://doi.org/10.1007/s10479-013-1467-4.

Mehmood Khan, Mohamad Y. Jaber, and Mohamed I.M. Wahab. Economic order quantity model for items with imperfect quality with learning in inspection. *International Journal of Production Economics*, 124(1):87–96, 2010. URL http://dx.doi.org/10.1016/j.ijpe.2009.10.011.

Narges Khanlarzade, B Yegane, I Kamalabadi, and Hiwa Farughi. Inventory control with deteriorating items: A state-of-the-art literature review. *International Journal of Industrial Engineering Computations*, 5(2):179–198, 2014. URL http://dx.doi.org/10.5267/j.ijiec.2013.11.003.

G. P. Kiesmüller, A. G. De Kok, and S. Dabia. Single item inventory control under periodic review and a minimum order quantity. *International Journal of Production Economics*, 133(1):280–285, 2011. URL http://doi.org/10.1016/j.ijpe.2010.03.019.

Gudrun P. Kiesmüller and Ton de Kok. The customer waiting time in an (r, s, q) inventory system. *International Journal of Production Economics*, 104(2):354–364, 2006. URL http://doi.org/10.1016/j.ijpe.2004.08.009.

Gudrun P. Kiesmüller, Ton G. De Kok, Sanne R. Smits, and Peter J.M. Van Laarhoven. Evaluation of divergent n-echelon (s,nq)-policies under compound renewal demand. *OR Spectrum*, 26(4):547–577, 2004. URL http://doi.org/10.1007/s00291-004-0172-8.

Gitae Kim, Kan Wu, and Edward Huang. Optimal inventory control in a multi-period newsvendor problem with non-stationary demand. *Advanced Engineering Informatics*, 29(1):139–145, 2015. URL https://doi.org/10.1016/j.aei.2014.12.002.

Peer Kleinau and Ulrich W. Thonemann. Deriving inventory-control policies with genetic programming. *OR Spectrum*, 26(4):521–546, 2004. URL https://doi.org/10.1007/s00291-004-0159-5.

Steffen T Klosterhalfen, Stefan Minner, and Sean P Willems. Strategic safety stock placement in supply networks with static dual supply. *Manufacturing & Service Operations Management*, 16 (2):204–219, 2014. URL https://doi.org/10.1287/msom.2013.0472.

Xiangyin Kong, Huihui Wang, Xi Chen, and Gengzhong Feng. Optimal policy for inventory systems with capacity commitment and fixed ordering costs. *Operations Research Letters*, 48(1):9–17, 2020. URL https://doi.org/10.1016/j.orl.2019.10.006.

Chaaben Kouki and Oualid Jouini. On the effect of lifetime variability on the performance of inventory systems. *International Journal of Production Economics*, 167:23–34, 2015. URL http://doi.org/10.1016/j.ijpe.2015.05.007.

Chaaben Kouki, Zied Jemaï, and Stefan Minner. A lost sales (r, q) inventory control model for perishables with fixed lifetime and lead time. *International Journal of Production Economics*, 168:143–157, 2015. URL http://dx.doi.org/10.1016/j.ijpe.2015.06.010.

Anil Kukreja and Charles P. Schmidt. A model for lumpy demand parts in a multi-location inventory system with transshipments. *Computers and Operations Research*, 32(8):2059–2075, 2005. URL https://doi.org/10.1016/j.cor.2004.01.007.

Anathasios G. Lagodimos, Ioannis T. Christou, and Konstantina Skouri. Optimal (r,nq,t) batch ordering with quantized supplies. *Computers and Operations Research*, 39(2):259–268, 2012. URL http://doi.org/10.1016/j.cor.2011.03.016.

Christian Larsen and G. P. Kiesmüller. Developing a closed-form cost expression for an (r, s, nq) policy where the demand process is compound generalized erlang. *Operations Research Letters*, 35(5):567–572, 2007. URL http://doi.org/10.1016/j.orl.2006.11.002.

Jun Yeon Lee. An incentive contract for leadtime reduction in an (s - 1, s) inventory system. *Computers and Operations Research*, 36(5):1633–1638, 2009. URL http://doi.org/10.1016/j.cor.2008.03.007.

Peng Li and Haoxun Chen. Optimal (r, nq) policies for serial inventory systems with guaranteed service. *Operations Research Proceedings 2011*, pages 439–444, 2012. URL http://doi.org/10.1007/978-3-642-29210-1_70.

Xiaoming Li and V. Sridharan. Characterizing order processes of using (R, nQ) inventory policies in supply chains. *Omega*, 36(6):1096–1104, 2008. URL http://doi.org/10.1016/j.omega.2006.06.002.

Lijian Lu, Jing-Sheng Song, and Hanqin Zhang. Optimal and asymptotically optimal policies for assemble-to-order n-and w-systems. *Naval Research Logistics (NRL)*, 62(8):617–645, 2015. URL https://doi.org/10.1002/nav.21671.

Xiyuan Ma, Roberto Rossi, and Thomas Archibald. Stochastic inventory control: A literature review. *IFAC-PapersOnLine*, 52(13):1490–1495, 2019. URL https://doi.org/10.1016/j.ifacol.2019.11.410.

Gour Chandra Mahata. Eoq model for items with exponential distribution deterioration and linear trend demand under permissible delay in payments. *International Journal of Soft Computing*, 6 (3):46–53, 2011. URL https://doi.org/10.3923/ijscomp.2011.46.53.

Anwar Mahmoodi, Alireza Haji, and Rasoul Haji. One for one period policy for perishable inventory. *Computers and Industrial Engineering*, 79:10–17, 2015. URL http://doi.org/10.1016/j.cie.2014.10.012.

Jorge Maturana, Alvaro Fialho, Frédéric Saubion, Marc Schoenauer, and Michele Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *2009 IEEE Congress on Evolutionary Computation*, pages 365–372. IEEE, 2009.

Agathe Métaireau, Rabin Kumar Sahu, Simon Delecourt, Alexandre Gerussi, and Manuel Davy. Multi-periodic joint replenishment planning method for various all-unit discounts. In *ICORES (International Conference on Operations Research and Enterprise Systems) 2022*, 2022. URL `https://hal.science/hal-03563906`.

Pablo A Miranda and Guillermo G Cabrera. Inventory location problem with stochastic capacity constraints under periodic review (r,s,s). *Proceedings ICIL 2010 - International Conference on Industrial Logistics "Logistics and Sustainability"*, pages 289–296, 2010.

Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, 65, 2011.

Anis Mjirda, Bassem Jarboui, Rita Macedo, and Saïd Hanafi. A variable neighborhood search for the multi-product inventory routing problem. *Electronic Notes in Discrete Mathematics*, 39: 91–98, 2012.

Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

Seyyed M.H. Molana, Hamid Davoudpour, and Stephan Minner. An (r, nq) inventory model for packaged deteriorating products with compound poisson demand. *Journal of the Operational Research Society*, 63(11):1499–1507, 2012. URL `http://doi.org/10.1057/jors.2011.154`.

Luis A Moncayo-Martínez, Edgar O Reséndiz-Flores, D Mercado, and Cuauhtemoc Sánchez-Ramírez. Placing safety stock in logistic networks under guaranteed-service time inventory models: An application to the automotive industry. *Journal of Applied Research and Technology*, 12 (3):538–550, 2014.

Luis A Moncayo-Martínez, Adán Ramírez-López, and Gustavo Recio. Managing inventory levels and time to market in assembly supply chains by swarm intelligence algorithms. *The International Journal of Advanced Manufacturing Technology*, 82:419–433, 2016. URL `https://doi.org/10.1007/s00170-015-7313-x`.

J. J.A. Moors and L. W.G. 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. URL `http://doi.org/10.1057/palgrave.jors.2601441`.

Philip M. Morse. Solutions of a class of discrete-time inventory problems. *Operations Research*, 7 (1):67–78, 1959. URL `http://doi.org/10.1287/opre.7.1.67`.

Seyed Mohsen Mousavi, Vahid Hajipour, Seyed Taghi Akhavan Niaki, and Najmeh Alikar. Optimizing multi-item multi-period inventory control system with discounted cash flow and inflation: two calibrated meta-heuristic algorithms. *Applied Mathematical Modelling*, 37(4):2241–2256, 2013. URL `https://doi.org/10.1016/j.apm.2012.05.019`.

Eliezer Naddor. Some models of inventory and an application. *Management Science*, 2(4):299–312, 1956. URL `https://doi.org/10.1287/mnsc.2.4.299`.

Steven Nahmias. Inventory models. *The Encyclopedia of Computer Science and Technology*, 9 (447-483):296, 1978.

John C Nash. The (dantzig) simplex method for linear programming. *Computing in Science & Engineering*, 2(1):29–31, 2000. URL https://doi.org/10.1109/5992.814654.

Bimal Nepal, Leslie Monplaisir, and Oluwafemi Famuyiwa. A multi-objective supply chain configuration model for new products. *International Journal of Production Research*, 49(23):7107–7134, 2011. URL https://doi.org/10.1080/00207543.2010.511294.

Thomas L Newberry. A classification of inventory control theory. *J. Industr. Engng*, 11(5):391–397, 1960.

Duc Huy Nguyen and Haoxun Chen. An effective approach for optimization of a perishable inventory system with uncertainty in both demand and supply. *International Transactions in Operational Research*, pages 1–23, 2020. URL https://doi.org/10.1111/itor.12846.

Wenjun Ni and Jia Shu. Trade-off between service time and carbon emissions for safety stock placement in multi-echelon supply chains. *International Journal of Production Research*, 53(22): 6701–6718, 2015. URL https://doi.org/10.1080/00207543.2015.1056319.

Marije Noordhoek, Wout Dullaert, David SW Lai, and Sander de Leeuw. A simulation-optimization approach for a service-constrained multi-echelon distribution network. *Transportation Research Part E: Logistics and Transportation Review*, 114:292–311, 2018. URL https://doi.org/10.1016/j.tre.2018.02.006.

Fredrik Olsson. Modelling two-echelon serial inventory systems with perishable items. *IMA Journal of Management Mathematics*, 21(1):1–17, 2010. URL http://doi.org/10.1093/imaman/dpp014.

Sezi Çevik Onar, Başar Öztayşi, Cengiz Kahraman, Seda Yanık, and Özlem Şenvar. A literature survey on metaheuristics in production systems. *Metaheuristics for Production Systems*, pages 1–24, 2016. URL https://doi.org/10.1007/978-3-319-23350-5_1.

Ibrahim H Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations research*, 63:511–623, 1996. URL https://doi.org/10.1007/BF02125421.

Zhan Pang. Optimal dynamic pricing and inventory control with stock deterioration and partial backordering. *Operations Research Letters*, 39(5):375–379, 2011. URL http://dx.doi.org/10.1016/j.orl.2011.06.009.

Colin Paterson, Gudrun Kiesmüller, Ruud Teunter, and Kevin Glazebrook. Inventory models with lateral transshipments: A review. *European Journal of Operational Research*, 210(2):125–136, 2011. URL https://doi.org/10.1016/j.ejor.2010.05.048.

Riccardo Patriarca, Francesco Costantino, and Giulio Di Gravio. Inventory model for a multi-echelon system with unidirectional lateral transshipment. *Expert Systems with Applications*, 65: 372–382, 2016. URL https://doi.org/10.1016/j.eswa.2016.09.001.

Subramanian Pazhani, José A Ventura, and Abraham Mendoza. A serial inventory system with supplier selection and order quantity allocation considering transportation costs. *Applied Mathematical Modelling*, 40(1):612–634, 2016. URL https://doi.org/10.1016/j.apm.2015.06.008.

Lu Peng, Lin Wang, and Sirui Wang. A review of the joint replenishment problem from 2006 to 2022. *Management System Engineering*, 1(1):9, 2022. URL https://doi.org/10.1007/s44176-022-00010-3.

Michael Phelan and Seán McGarraghy. Grammatical evolution in developing optimal inventory policies for serial and distribution supply chains. *International Journal of Production Research*, 54(1):336–364, 2016. URL https://doi.org/10.1080/00207543.2015.1085653.

Çerağ Pinçe and Rommert Dekker. An inventory model for slow moving items subject to obsolescence. *European Journal of Operational Research*, 213(1):83–95, 2011. URL http://doi.org/10.1016/j.ejor.2011.02.013.

Dražen Popović, Milorad Vidović, and Gordana Radivojević. Variable neighborhood search heuristic for the inventory routing problem in fuel delivery. *Expert Systems with Applications*, 39(18): 13390–13398, 2012.

Sameer Prasad. Classification of inventory models and systems. *International Journal of Production Economics*, 34(2):209–222, 1994. URL https://doi.org/10.1016/0925-5273(94)90037-X.

Boualem Rabta and Djamil Aïssani. Strong stability in an (r,s,s) inventory model. *International Journal of Production Economics*, 97(2):159–171, 2005. URL http://doi.org/10.1016/j.ijpe.2004.06.050.

Mohammad Rahdar, Lizhi Wang, and Guiping Hu. A tri-level optimization model for inventory control with uncertain demand and lead time. *International Journal of Production Economics*, 195:96–105, 2018. URL https://doi.org/10.1016/j.ijpe.2017.10.011.

Gintaras V Reklaitis. Review of scheduling of process operations. *AIChE Symposium Series*, 78 (214):119–133, 1982.

Jan Riezebos and Gerard J.C. Gaalman. A single-item inventory model for expected inventory order crossovers. *International Journal of Production Economics*, 121(2):601–609, 2009. URL http://dx.doi.org/10.1016/j.ijpe.2006.10.004.

Leonardo Ramos Rodrigues and Takashi Yoneyama. A spare parts inventory control model based on prognostics and health monitoring data under a fill rate constraint. *Computers and Industrial Engineering*, 148, 2020. URL https://doi.org/10.1016/j.cie.2020.106724.

Rabin Kumar Sahu. *General framework and optimization methods for stochastic replenishment planning in industrial contexts*. PhD thesis, Université de Lille, August 2020. URL https://hal.science/tel-03103384.

Mohsen S Sajadieh and Anders Thorstenson. Comparing sourcing strategies in two-echelon supply chains. *Computers & operations research*, 45:108–115, 2014. URL https://doi.org/10.1016/j.cor.2013.12.006.

Shib Sankar Sana. An economic order quantity model for seasonal goods. *International Journal of Operational Research*, 3(1-2):97–118, 2008. URL https://doi.org/10.1504/IJOR.2008.016156.

Ilkay Saracoglu, Seyda Topaloglu, and Timur Keskinturk. A genetic algorithm approach for multi-product multi-period continuous review inventory models. *Expert Systems with Applications*, 41 (18):8189–8202, 2014. URL https://doi.org/10.1016/j.eswa.2014.07.003.

Herbert Scarf. The optimality of (s,s) policies in the dynamic inventory problem. In Eytan Sheshinski and Yoram Weiss, editors, *Optimal Pricing, Inflation, and the Cost of Price Adjustment*, chapter 3, pages 49–56. The MIT Press, Cambridge, Massaschussets, 1959.

Thomas G Schmitt, Sanjay Kumar, Kathryn E Stecke, Fred W Glover, and Mark A Ehlen. Mitigating disruptions in a multi-echelon supply chain using adaptive ordering. *Omega*, 68:185–198, 2017. URL https://doi.org/10.1016/j.omega.2016.07.004.

Kevin H. Shang, Jing Sheng Song, and Paul H. Zipkin. Coordination mechanisms in decentralized serial inventory systems with batch ordering. *Management Science*, 55(4):685–695, 2009. URL http://doi.org/10.1287/mnsc.1080.0981.

Craig C. Sherbrooke. Metric: A multi-echelon technique for recoverable item control. *Operations Research*, 16(1):122–141, 1968. URL http://doi.org/10.1287/opre.16.1.122.

Jia Shu and IA Karimi. Efficient heuristics for inventory placement in acyclic networks. *Computers & Operations Research*, 36(11):2899–2904, 2009. URL https://doi.org/10.1016/j.cor.2009.01.001.

Edward A Silver. Operations research in inventory management: A review and critique. *Operations Research*, 29(4):628–645, 1981. URL https://doi.org/10.1287/opre.29.4.628.

David Simchi-Levi. *Designing and Managing the Supply Chain; Concepts, Strategies, and Case Studies*. McGraw-Hill Higher Education, 2000.

Lawrence V Snyder and Zuo-Jun Max Shen. *Fundamentals of supply chain theory*. John Wiley & Sons, 2019.

Olof Stenius, Ayşe Gönül Karaarslan, Johan Marklund, and Ton De Kok. Exact analysis of divergent inventory systems with time-based shipment consolidation and compound poisson demand. *Operations Research*, 64(4):906–921, 2016. URL https://doi.org/10.1287/opre.2016.1510.

Josef Svoboda, Stefan Minner, and Man Yao. Typology and literature review on multiple supplier inventory control models. *European Journal of Operational Research*, 293(1):1–23, 2021. URL https://doi.org/10.1016/j.ejor.2020.11.023.

El-Ghazali Talbi. *Metaheuristics: From design to implementation*. John Wiley & Sons, 2009.

Ata Allah Taleizadeh, Seyed Taghi Akhavan Niaki, Nima Shafii, Ramak Ghavamizadeh Meibodi, and Armin Jabbarzadeh. A particle swarm optimization approach for constraint joint single buyer-single vendor inventory problem with changeable lead time and (r,q) policy in supply chain. *International Journal of Advanced Manufacturing Technology*, 51(9-12):1209–1223, 2010. URL https://doi.org/10.1007/s00170-010-2689-0.

Ata Allah Taleizadeh, Seyed Taghi Akhavan Niaki, and Hui Ming Wee. Joint single vendor-single buyer supply chain problem with stochastic demand and fuzzy lead-time. *Knowledge-Based Systems*, 48:1–9, 2013. URL http://dx.doi.org/10.1016/j.knosys.2013.03.011.

Shahrzad Tamjidzad and S. Hamid Mirmohammadi. An optimal (r, q) policy in a stochastic inventory system with all-units quantity discount and limited sharable resource. *European Journal of Operational Research*, 247(1):93–100, 2015. URL http://dx.doi.org/10.1016/j.ejor.2015.05.073.

Shahrzad Tamjidzad and S. Hamid Mirmohammadi. Optimal (r, q) policy in a stochastic inventory system with limited resource under incremental quantity discount. *Computers and Industrial Engineering*, 103:59–69, 2017. URL http://dx.doi.org/10.1016/j.cie.2016.11.012.

Horst Tempelmeier. Inventory control using a service constraint on the expected customer order waiting time. *European Journal of Operational Research*, 19(3):313–323, 1985. URL http://doi.org/10.1016/0377-2217(85)90127-4.

Horst Tempelmeier. A multi-level inventory system with a make-to-order supplier. *International Journal of Production Research*, 51(23-24):6880–6890, 2013. URL http://doi.org/10.1080/00207543.2013.776190.

Horst Tempelmeier and Oliver Bantel. Integrated optimization of safety stock and transportation capacity. *European Journal of Operational Research*, 247(1):101–112, 2015. URL http://doi.org/10.1016/j.ejor.2015.05.069.

Horst Tempelmeier and Lars Fischer. Approximation of the probability distribution of the customer waiting time under an (r, s, q) inventory policy in discrete time. *International Journal of Production Research*, 48(21):6275–6291, 2010. URL http://doi.org/10.1080/00207540903317531.

Jinn-Tsair Teng. On the economic order quantity under conditions of permissible delay in payments. *Journal of the Operational Research Society*, 53(8):915–918, 2002. URL https://doi.org/10.1057/palgrave.jors.2601410.

Jinn Tsair Teng, Liang Yuh Ouyang, and Chun Tao Chang. Deterministic economic production quantity models with time-varying demand and cost. *Applied Mathematical Modelling*, 29(10):987–1003, 2005. URL https://doi.org/10.1016/j.apm.2005.02.002.

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. URL http://doi.org/10.1080/00207543.2012.671590.

Engin Topan and Matthieu C van der Heijden. Operational level planning of a multi-item two-echelon spare parts inventory system with reactive and proactive interventions. *European Journal of Operational Research*, 284(1):164–175, 2020. URL https://doi.org/10.1016/j.ejor.2019.12.022.

Karel H. van Donselaar and Rob A. C. M. Broekmeulen. Fill rate approximations for a perishable inventory system with positive lead time and fixed case pack size using a modified (r,s,nq)-replenishment policy in a lost sales environment. *SSRN Electronic Journal*, 2012. URL http://doi.org/10.2139/ssrn.1861157.

Willem van Jaarsveld, Twan Dollevoet, and Rommert Dekker. Improving spare parts inventory control at a repair shop. *Omega (United Kingdom)*, 57:217–229, 2015. URL http://doi.org/10.1016/j.omega.2015.05.002.

Erwin van Wingerden, Tarkan Tan, and Geert-Jan Van Houtum. The impact of an emergency warehouse in a two-echelon spare parts network. *European Journal of Operational Research*, 276 (3):983–997, 2019. URL https://doi.org/10.1016/j.ejor.2019.01.068.

Gideon J. J. van Zyl. *Inventory control for perishable commodities*. PhD thesis, University of North Carolina, 1963.

Mauricio Varas, Franco Basso, Armin Lüer-Villagra, Alejandro Mac Cawley, and Sergio Maturana. Managing premium wines using an (s- 1 , s) inventory policy: a heuristic solution approach. *Annals of Operations Research*, 280(1-2):351–376, 2019. URL http://doi.org/10.1007/s10479-019-03266-3.

Nadarajen Veerapen, Jorge Maturana, and Frédéric Saubion. An exploration-exploitation compromise-based adaptive operator selection for local search. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, pages 1277–1284, 2012.

Senthil Veeraraghavan and Alan Scheller-Wolf. Now or later: A simple policy for effective dual sourcing in capacitated systems. *Operations Research*, 56(4):850–864, 2008. URL https://doi.org/10.1287/opre.1080.0552.

Arthur F Veinott Jr. The status of mathematical inventory theory. *Management Science*, 12(11): 745–777, 1966. URL https://doi.org/10.1287/mnsc.12.11.745.

Feng Yu Wang, Laura Xu, Roland Lim, E. W. Lee, and Michal Zarzycki. Simulation-based operational decision analysis at decoupling point in mts-mto system. *IEEE International Conference on Industrial Engineering and Engineering Management*, pages 56–60, 2011. URL http://doi.org/10.1109/IEEM.2011.6117878.

Xiaobin Wang. Continuous review inventory model with variable lead time in a fuzzy random environment. *Expert Systems with Applications*, 38(9):11715–11721, 2011. URL http://dx.doi.org/10.1016/j.eswa.2011.03.057.

Thomson M Whitin. Inventory control research: A survey. *Management Science*, 1(1):32–40, 1954. URL https://doi.org/10.1287/mnsc.1.1.32.

Kaycee J Wilson, Rachel Hodge, and David Bivens. Reducing stockouts in a cancer center's ambulatory care clinics. *Engineering Management Journal*, 27(3):99–108, 2015. ISSN 1042-9247. URL http://doi.org/10.1080/10429247.2015.1059218.

Tomasz Wiśniewski. Simulation study of inventory management in supply chains. *Logistics and Transport*, 37:41–48, 2018.

Jie Wu, Qiang Wang, Chaolin Yang, and Yi Yang. Dual-index policies for serial systems with dual delivery modes and batch orders. *SSRN*, 2019. URL https://dx.doi.org/10.2139/ssrn.3313886.

Jong Wuu Wu, Chinho Lin, Bertram Tan, and Wen Chuan Lee. An eoq inventory model with time-varying demand and weibull deterioration with shortages. *International Journal of Systems Science*, 31(6):677–683, 2000. URL https://doi.org/10.1080/00207720050030716.

Yisha Xiang and Manuel D. Rossetti. The effect of backlog queue and load-building processing in a multi-echelon inventory network. *Simulation Modelling Practice and Theory*, 43:54–66, 2014. URL http://doi.org/10.1016/j.simpat.2014.01.006.

Baimei Yang, Yiqian Huang, and Yifan Xu. Optimal control of a finite-capacity inventory system with setup cost and lost sales. *Journal of Applied Analysis and Computation*, 4(2):197–220, 2014. URL https://doi.org/10.11948/2014009.

Xiaobo Zhao, Minmin Qiu, Jinxing Xie, and Qiming He. Computing (r, q) policy for an inventory system with limited sharable resource. *Computers and Operations Research*, 39(10):2368–2379, 2012. URL http://dx.doi.org/10.1016/j.cor.2011.12.012.

Sean X. Zhou and Chaolin Yang. Continuous-review (r, nq) policies for inventory systems with dual delivery modes. *Operations Research*, 64(6):1302–1319, 2016. URL http://doi.org/10.1287/opre.2016.1538.

Han Zhu. A simple heuristic policy for stochastic inventory systems with both minimum and maximum order quantity requirements. *Annals of Operations Research*, 309(1):347–363, 2022. URL https://doi.org/10.1007/s10479-021-04441-1.

Han Zhu, Xing Liu, and Youhua Chen. Effective inventory control policies with a minimum order quantity and batch ordering. *International Journal of Production Economics*, 168:21–30, 2015. URL http://doi.org/10.1016/j.ijpe.2015.06.008.