Robust Lot Sizing with Remanufacturing: Theory and Practice

Öykü Naz Attila
Department of Management Science
University of Strathclyde
Glasgow, UK
2019

A thesis presented in fulfilment of the requirements for the degree of Doctor of Philosophy.

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Signed: Öykü Naz Attila Date: March 2, 2020

Publications

Parts of this dissertation have been published in, submitted to for publication, or are in the process of being submitted to for publication in peer-reviewed journals and conference proceedings, as listed below:

- Attila, Ö.N., Agra, A., Akartunalı, K., Arulselvan, A. (2017). A decomposition algorithm for robust lot sizing problem with remanufacturing option. In *International Conference on Computational Science and Its Applications* (pp. 684-695). Springer, Cham.

- Attila, Ö.N., Agra, A., Akartunalı, K., Arulselvan, A., Robust formulations for economic lot-sizing problem with remanufacturing. Second round revision submitted to *European Journal of Operational Research*.

- Attila, Ö.N., Agra, A., Akartunalı, K., Arulselvan, A., A min-max decomposition approach for the robust two-level, multi-component lot sizing problem with remanufacturing, Working paper to be submitted to *INFORMS Journal on Computing* on January 2020.

Acknowledgements

I would like to show my gratitude for my supervisors Kerem Akartunalı, and Ashwin Arulselvan for their guidance and encouragement throughout the course of my PhD. Kerem and Ashwin have always been a tremendous source of support and inspiration for me. Through their supervision I have not only learned what I know about mixed integer linear programming today, but also how to become a better researcher. I am very grateful to have had their support not only while conducting the studies that led to this thesis, but also in many other aspects of academia and life. I cannot thank you enough for your incredible support!

A very special thank you to Agostinho Agra for his continuous guidance and support. I am very happy for the fruitful discussions we had on robust optimization, to which I owe a considerable part of my knowledge in this domain.

I am very grateful to Ayşe Akbalık for her continuous support. Her encouragement and guidance inspire me tremendously to become the curious researcher I aspire to be today.

I would like to express my thanks to Hakan Gültekin for introducing me to the world of mathematical programming, for his support and for having inspired me to become a researcher in this field.

I would also like to show my gratitude to the University of Strathclyde for making this PhD thesis possible by providing me with a fully-funded scholarship.

My close friends Sera, Ilkim, Ilayda and Doruk have always shown me their support, even from hundreds and thousands of miles away. A very special thank you to Shona for always being there for me and supporting me throughout my PhD journey. I would also like to thank Emma for her continuous support, and our fun weekend trips. I am very happy to have had our fun lunchtime discussions and badminton sessions with Daniel, Orion, Ross, Chirsty, Andrew, Piero, and Ben. I am very thankful to Dagmar, Gökhan, Ayşe, Esra and many other friends that I have met here at Strathclyde who were always happy to help, and have shown an incredible amount of support, both in and out of campus. I greatly appreciate their continuous encouragement, which enhanced my PhD journey vastly.

Finally, I would like to express my gratefulness to my parents, for everything they have done for me and for always being by my side. Their unconditional love and encouragement inspire me immensely. I am beyond grateful to have you.

To my parents...

Fügen & Doğan'a

Abstract

With the increasing importance of means of recovering items, practices such as reuse, recycling and remanufacturing have been gaining increasing importance in production systems. In this thesis, we aim to explore methods to obtain cost-efficient production plans for manufacturing systems that employ remanufacturing.

A crucial challenge here is regarding parameter uncertainty. Commonly, decision makers have to tackle uncertainties on aspects such as operational costs, demands and in the case of production recoveries, the number of items returned by customers. Our work aims to address these challenges through the framework of robust optimization, where we impose uncertainty on demands and/or returns.

In Chapter 1, we provide an introduction outlining the methods and formulations related to our study. In Chapter and 2, we review the literature on lot sizing problems, and the implications of remanufacturing. Following this, we introduce a robust lot sizing problem with remanufacturing in Chapter 3, where we consider uncertainties in both customer demand and return and implement a min-max decomposition approach for this problem. In Chapter 4, we propose a novel approach which employs extended reformulations for the master problem, while addressing computational challenges. In Chapter 5, we consider a different setting for the lot sizing problem with remanufacturing, where a two-level multi-component variation is considered. We consider the case where fixed costs are imposed on components and only variable costs are considered for the end-item. This is then followed by a robust formulation for and a min-max decomposition approach. We show that certain optimality properties have to be enforced to derive costs correctly. In Chapter 6, we consider the two-level multi-component problem with fixed costs on the end-item level. We show certain optimality conditions for this problem, and show how these properties can be exploited in the dynamic programming setting introduced in Teunter et al. [2006]. Throughout Chapters 4 and 5, we provide extensive computational experiments.

Finally, in Chapter 7, we provide future research directions and conclusions, where potential advantages and limitations are discussed.

Keywords: Integer Programming, Lot-Sizing, Remanufacturing, Robust Optimization, Decomposition

Abbreviations

2-MCR	Two-level Multi-component Lot Sizing with Remanufacturing
2-MCR-C	2-MCR with Fixed Costs on Component Level
2-MCR-CR	Robust 2-MCR
2-MCR-CRA	Adjustable Robust 2-MCR
AP	Adversarial Problem
DMP	Decision Maker's Problem
DMP-EFAG	Extended Aggregated Reformulation for DMP
DMP-EFAP	Approximate Extended Reformulation for DMP
EOQ	Economic Order Quantity
ERP	Enterprise Resource Planning
GLB	Global lower bound
GUB	Global upper bound
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
MRP	Material Requiriements Planning
IP	Integer Programming
LB	Lower bound
LP	Linear programming

LS	Lot-sizing
LSR	Lot-sizing with remanufacturing
LSR-D	Deterministic Lot Sizing Problem with Remanufacturing
RLSR	Robust Lot Sizing Problem with Remanufacturing
UB	Upper bound
WW	Wagner and Whitin

Notation and Symbols

$\mathbb{R}^n_+$	The n-dimensional space of non-negative real numbers.
$\mathbb{Z}^n_+$	The n-dimensional space of non-negative integer numbers.
$[0,1]^n$	The n-dimensional space of binary numbers.
$\mathcal{NP}$	Complexity class $\mathcal{NP}$.
$O()$	Big-O notation indicating problem complexity.
$Conv(P)$	Convex hull for points P.
$proj_{\mathcal{X}}(P)$	Projection of P onto $\mathcal{X}$
$\forall$	Equation/inequality applied for all indices in the indicated set.
$\exists$	Element exists in the given set.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

An everlasting challenge for production companies is to ensure cost-efficiency, despite the complex nature of production systems. A common way of accomplishing this is to find standardized procedures that are designed to assist decision makers in determining the right number of items to produce, while considering any accompanying decisions, including those that are related to inventory levels, workforce levels and scheduling of such resources. Often, achieving this involves finding methods that aim to use system resources optimally, such that the resulting decisions are able to respond to a wide and diverse range of customer demand. An important and challenging task here is to detect and enforce relevant restrictions on how and/or when the operations within the production system are allowed to be carried out. This often involves limitations such as production capacities, environmental concerns, and any other structural restrictions on resource usage, which may vary according to type of manufacturing system at hand.

Given the complex nature of production systems and the diversity of decisions involved, the variety of methods that aim to determine optimal production operations is vast. An effective methodology that is useful for determining such decisions efficiently is mathematical programming, which involves representing production systems using objective(s), constraint(s) and variable(s). Likewise, a vast number of methods and models exist within this domain, which aim to detect such optimal operational decisions. While the models within this domain may focus on decisions on the operational level, some are also commonly used to make decisions

on a wider scale, including strategic and tactical decisions in supply chains (Mula et al. [2010] for a review of mathematical models used in supply chain and transport problems). Developing and improving such methods are crucially important for decision makers, since these approaches are efficient procedures through which substantial cost savings are achieved. Moreover, the high complexity of today's production systems adds onto the importance of utilizing standardized and effective methods, such that human error is minimized while maintaining efficiency, both in terms of solution quality and time requirements.

In order to manage decisions related to such extensive variety of production operations as a whole, decision makers have been utilizing systems such as ERP (Enterprise Resource Planning), and its predecessors MRP (Material Requirements Planning, Plossl and Orlicky [1994]) and MRP II. An essential part of these systems requires mathematical programming models and methods to generate optimal solutions. This helps provide insights regarding numerous requirements for production planning and beyond. More specifically, in this thesis we will particularly examine decisions related to production sizes (to which we often refer to as "lot sizes", or "lot sizing" to define the wider range of problems that address the issue of finding optimal production sizes) using mixed integer linear programming formulations and relevant solution methods.

In addition, it is crucial to acknowledge that ERP systems have been evolving continually to answer manufacturers' changing needs (Jacobs and Weston [2007]). This calls for a need to amend mathematical models used for production planning to construct models that represent production systems today more accurately. Throughout this thesis, we will study the implications of considering the following challenges in classical lot sizing problems.

Environmental concerns

Undoubtedly, with growing concerns regarding the negative impact of production activities and waste on the environment, means of recovering used and old products have come into play in production companies in recent years. As a consequence, one of the main challenges faced by manufacturing companies today is to establish an environmentally friendly production system. This does not only imply that changes are implemented in production procedures themselves (such as incorporating reuse of old and defective products), but also that a substantial change is required in the mathemat-

ical models used for production planning purposes. As a result, developing mathematical programs that consider not only the traditional operations in production environments, but also the decisions associated with environmentally friendly means of production is of both practical and environmental importance. This, of course, is crucial to maintain cost savings, since disregarding such elements can lead to significant losses in profit, as well as inaccurate planning of resources. Throughout this thesis, we will be exploring production planning models that are able to tackle the additional decisions that come with product recovery. More specifically, we examine the option of remanufacturing, for which a formal description will be given in Sections 1.4 and 2.2. In these sections, we will also discuss the importance of remanufacturing in terms of its overall positive impact on the environment. In addition to this, we will address the challenges and changes that come with integrating decisions that are related to remanufacturing to traditional lot sizing models.

Uncertainty

With growing complexity of production systems, acquiring accurate information on the operations carried out within such systems is becoming an increasingly difficult task. As a consequence, building accurate mathematical models has become a major challenge to tackle. This is not only because of the wide range of aspects that need to be taken into account, but also because of the individual uncertainty around these aspects (such as uncertainty in lead times, workforce availability, type and level of customer demand). Estimating such fundamental elements inaccurately can lead to a great risk of deteriorating the accuracy of mathematical models, since these play a crucial rule while generating optimal production plans. Specifically, such inaccuracies can produce production plans that overuse or underuse resources (which often leads to significant problems such as: increase in operational costs, loss of profit, and inaccurate resource allocation estimations). Therefore, it is crucial to account for the uncertainty around such parameters while constructing mathematical models for production planning purposes. In this thesis, we present formulations that consider the simultaneous impact of demand and return (i.e. items that are returned to the system, which can

be used to perform remanufacturing) uncertainty in Chapters 3 and 4. In Chapter 5, we consider the case where only the number of items returned by customers are unknown. Finally, we proivde a detailed review of production planning problems where uncertainty is present, as well as relevant methods for modelling uncertainty in Chapter 2.

As discussed above, the main motivation behind this thesis is not only related to tackling the challenge of finding optimal lot sizes, but also providing formulations that are able to adapt to manufacturers' up-to-date needs. In this regard, the studies presented in this thesis are intended to address the issue of uncertainty in production systems, as well as the impact of incorporating remanufacturing in these systems.

As we will examine in further detail in the upcoming chapters, these formulations are intended to portray a general representation of production systems. Due to the favorable aspect of mathematical programs being flexible, these models can easily be altered to meet the particular needs and structure of a specific production system. Likewise, the general findings and methods presented throughout this thesis can provide useful insights while tackling similar challenges in different mathematical models.

1.2 Mixed Integer Linear Programming

Before we introduce the mathematical formulations that serve as basis for this thesis, we present the fundamental methods used to solve mixed integer linear programming problems. These methods create the very basis for the procedures used to solve the formulations that are presented throughout this thesis.

Among fundamental studies in the field of linear programming is the *Simplex Algorithm* (Dantzig [1951]). This work is of crucial importance for most methods used to solve both linear and mixed integer linear programs. Although this method has a fundamental role in solving mixed integer linear programs, it does not consider integrality restrictions on mathematical programs.

However, imposing integrality restrictions on linear programs is essential for most applications in practice. In order to comply with this, further studies were conducted, which investigate the ways through which integrality constraints may be considered in linear programs. Among preliminary studies that made a re-

markable contribution to the field of Integer Programming include the works of Gomory et al. [1958], Land and Doig [1960] and the references given in the work of Lawler and Wood [1966], where the concept of generating cuts and branching are introduced. These concepts serve as an essential part for most IP solvers today.

In the remainder of this section, we provide a general definition of Mixed Integer Linear Programs (MILPs), while defining some of the essential theoretical properties of MILPs. For a complete review of methods, formulations and properties, we refer the reader to Wolsey [1998] and Wolsey and Nemhauser [1999].

A MIP has a linear objective function and a set of constraints that need to be satisfied. In addition to this, we consider that we have different sets of decision variables. Firstly, we have the decision variables that are defined to take on continuous values, represented as $x := (x_1, x_2, \ldots, x_n)$. Another set of decision variables are those that are restricted to take on integer values, defined as $y := (y_1, y_2, \ldots, y_p)$. Under this setting, we may define a mixed integer linear programming problem as the following:

$$\min \ cx + hy \tag{MILP}$$

$$Ax + By \leq b \tag{1.1}$$

$$x \in \mathbb{R}^n, \ y \in \mathbb{Z}^p \tag{1.2}$$

which can also be shown as:

$$\min\{cx + hy : Ax + By \leq b, x \in \mathbb{R}^n, y \in \mathbb{Z}^p\} \tag{1.3}$$

Here, c and h are row vectors of size n and p, respectively. b is a column vector of size m. Likewise, we define A and B as matrices of sizes $n \times m$ and $p \times m$, respectively. Note that if the integrality restriction on variable y is relaxed, the problem becomes a linear programming (LP) program. Moreover, if all variables are restricted to only take on values 0 or 1, the problem can be classified as a "0-1 Program" (or a "Binary Program").

As we will see in the next section, a wide range of lot sizing problems can be classified as a MILP. Thus, in the remainder of this section, we particularly focus on properties of MILPs.

Definition 1.2.1. *Let $C \subseteq \mathbb{R}^n$ denote a set of points. If the convex combination of any two points in C is in set C, then C is called a convex set.*

Definition 1.2.2. *For a convex set C, Conv(C) is the convex hull of C, which is the set of all convex combinations of points in C.*

Based on the explanations given above, let us state the following definitions, through which we define the concept of feasibility for a MILP.

Definition 1.2.3. *C is called a polyhedron if the points in C satisfy a finite number of linear inequalities.*

A strong property to notice here is that a polyhedron is a convex set of points. This observation is crucial for the solution procedures for linear programming problems. Specifically, consider the following observation for MILPs:

Proposition 1. *Let $\mathcal{X}$ denote the feasible region of a MILP, and $Conv(\mathcal{X})$ denote the convex hull over $\mathcal{X}$. Then, solving this MILP over $Conv(\mathcal{X})$ is sufficient to solve this MILP.*

In other words, this property suggests that the we may find a solution in $Conv(\mathcal{X})$ using linear programming to obtain an integer optimal solution to an MILP, given that $Conv(\mathcal{X})$ is known. This is a very favorable observation, since it allows us to disregard the integrality restrictions on integer variables. However, in most cases, this property is not applicable, since identifying $Conv(\mathcal{X})$ is often a very difficult task, due to its complex structure.

Furthermore, a concept that plays a crucial role while solving MILPs is obtained by removing the integrality constraints. We formally define this as follows.

Definition 1.2.4. *Consider the MILP program:*

$$\min\{cx + hy : Ax + By \leq b, x \in \mathbb{R}^n, y \in \mathbb{Z}^p\}$$

Then, the LP relaxation to this MILP is given as:

$$\min\{cx + hy : Ax + By \leq b, x \in \mathbb{R}^n, y \in \mathbb{R}^p\}$$

where the integrality conditions on y are removed.

The concept of LP relaxations has an important role while finding optimal solutions to MILPs, as discussed further below. In addition, the optimal value of the objective function values of LP relaxations for different MILPs can provide meaningful insights while comparing the tightness of formulations.

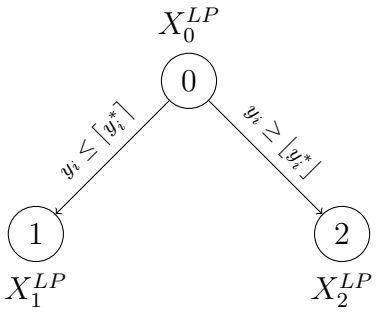

Figure 1.1: Branching decisions in the branch and bound algorithm.

A well-known method for solving MILPs is the *"Branch-and-Bound"* algorithm. The main idea behind this algorithm is to solve the LP relaxation of the MILP, while various combinations of integrality restrictions on integer variables are searched, such that the optimal objective function value improves throughout the search. More specifically, at the root node, the branch and bound algorithm solves the LP relaxation a given MILP, where:

$$X_0^{LP} : \min\{cx + hy : Ax + By \leq b, x \in \mathbb{R}^n, y \in \mathbb{R}^p\}$$

The branching process begins by considering the optimal solution for this LP relaxation. Let us indicate this solution with (x^*, y^*). Then, a *branching variable* is chosen. Let us indicate this variable with y_i^*. Based on the optimal value for the branching variable, a branching decision is made. More specifically, the algorithm eliminates fractional solutions to the branching variable. In order do this, the condition $\lfloor y_i^* \rfloor \leq y_i^* \leq \lceil y_i^* \rceil$ is imposed on the LP solution at the root node, as shown in Figure 1.1.

One way to implement this is to branch on variable y_i^* such that the algorithm seeks for the optimal values for the following LP programs:

- *Node 1:* $X_1^{LP} : \min\{cx + hy : Ax + By \leq b, y_i \leq \lceil y_i^* \rceil, x \in \mathbb{R}^n, y \in \mathbb{R}^p\}$
- *Node 2:* $X_2^{LP} : \min\{cx + hy : Ax + By \leq b, y_i \geq \lfloor y_i^* \rfloor, x \in \mathbb{R}^n, y \in \mathbb{R}^p\}$

The branch and bound algorithm continues the search in this manner until a solution is found, such that the solution is not only feasible but also optimal, including integrality restrictions.

Based on the introductory properties and methods introduced under this section, we provide an overview of production planning models that are formulated

as MILPs in the next section.

1.3 Production Planning Models

A crucial challenge that has to be tackled by production companies is to seek for efficient ways of generating effective production plans. As we have discussed in the previous section, mathematical programming models are used widely for this purpose. Throughout this section, we will provide a general introduction to such mathematical models, as well as relevant solution methods to solve these problems. The specific set of problems that we examine under this section aim to find optimal decisions related to production sizes. We call the general class of such problems "lot sizing problems", which serves as the basis for the formulations presented throughout this thesis.

Lot sizing problems are an essential part of the broad set of mathematical models used for production planning purposes. Generally speaking, lot sizing problems seek for the optimal quantity and time for producing items, while minimizing operational costs related to activities such as production, holding items in inventory, and setup of machines. As we will discuss in greater detail, an important property of lot sizing problems is that such models are able to take system requirements into consideration (e.g. restrictions on production capacities, magnitude of customer demand etc.) by considering the combinatorial structure of a vast number of relevant decisions. This is a very favorable property, since the decisions involved in production systems are hard to analyze, due to their sophisticated structure.

One of the most well-known formulations that still serves as the foundation of most lot sizing problems today is the "*Dynamic lot-sizing model*" of Wagner and Whitin [1958], which can be formulated as follows:

$$\min \sum_{t=1}^{\mathcal{T}} (h_t I_t + K_t y_t) \tag{WW}$$

$$I_{t-1} + x_t^m = I_t + D_t \qquad \forall t = 1, \ldots, \mathcal{T} \tag{1.4}$$

$$x_t^m \leq M_t y_t \qquad \forall t = 1, \ldots, \mathcal{T} \tag{1.5}$$

$$x^m, I \geq 0 \tag{1.6}$$

$$y \in \{0,1\}^{\mathcal{T}} \tag{1.7}$$

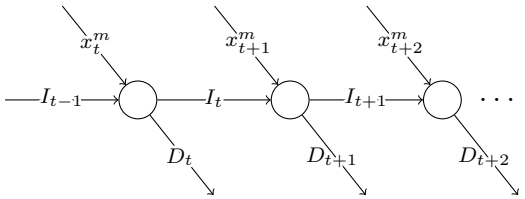

Figure 1.2: Network flow diagram for the dynamic lot sizing problem.

Our main objective in this formulation is to find the minimum total inventory and setup cost (represented as h_t and K_t, respectively, for time period t). To do this, we need to determine the optimal values of the decision variables involved in the formulation above (manufacturing, inventory and setup decisions, indicated as x_t^m, I_t and y_t for time period t, respectively). Here, we use equation (1.4) to represent the flow of items, where a customer demand of D_t has to be satisfied on time period t. Another essential element in (WW) that is often used in most lot sizing models today is the setup constraint given in (1.5). This constraint suggests that manufacturing on a given time period can only take place if a setup is carried out on the same time period. To enforce this, we define y_t as binary, where

$$y_t = \begin{cases} 0, \text{ if } x_t^m = 0 \\ 1, \text{ otherwise} \end{cases}$$

In order to obtain the optimal total cost, the aforementioned decision variables are optimized for the whole set of discrete time periods $\{1, \ldots, \mathcal{T}\}$. We refer to this set as the "planning horizon". The number of periods considered here can be altered according to the needs of the production facility (e.g. daily, weekly, monthly planning). This problem can also be represented as a network (see Figure 1.2).

Undoubtedly, the dynamic lot-sizing model has provided the foundation for a large number of lot sizing problems being studied today. Throughout the remainder of this section, we will examine several extensions to the dynamic lot-sizing model in further depth, particularly those that closely align with the fundamental structure of the models we present in this thesis.

More specifically, we focus on the following concepts:

1. *Backlogging*

   The first extension we consider is the addition of the option of backlogging, which was primarily studied in by Zangwill [1966] and Zangwill [1969]. Here, customer demand is allowed to be backlogged, which suggests that inventory levels are allowed to take on negative values (i.e. the case where the number of items present at a given time period t is insufficient to satisfy D_t). Since unsatisfied customer demand leads to loss of profit, performing backlogging incurs a penalty cost. Thus, we may write the general formulation for the dynamic lot-sizing model with backlogging as follows:

$$\min \sum_{t=1}^{\mathcal{T}} (h_t^+ I_t^+ + h_t^- I_t^- + K_t y_t)$$

$$I_{t-1}^+ + I_t^- + x_t^m = D_t + I_t^+ + I_{t-1}^- \qquad \forall t = 1, \ldots, \mathcal{T} \qquad (1.8)$$

$$x_t^m \leq M_t y_t \qquad \forall t = 1, \ldots, \mathcal{T} \qquad (1.9)$$

$$x^m, I^+, I^- \geq 0 \qquad (1.10)$$

$$y \in \{0,1\}^{\mathcal{T}} \qquad (1.11)$$

   Here, the main difference in comparison to (WW) is the new inventory variables, I_t^+ and I_t^-, which represent the positive inventory level at the end of period t, and the number of items backlogged on period t, respectively. Note that replacing the old inventory variables with the new ones results in a change in the objective function, where positive inventory and backlogging decisions have separate costs of h_t^+ and h_t^-, respectively. Considering the option of backlogging, we may update the network flow representation as given in Figure 1.3. As the figure suggests, a positive value on the newly added backlogging variables I_t^- allow customer demand to remain unsatisfied.

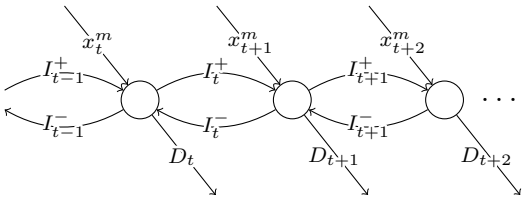

Figure 1.3: Network flow diagram for the dynamic lot sizing problem with backlogging.

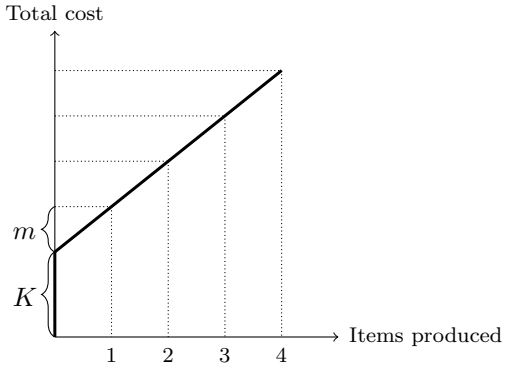

Figure 1.4: Total setup and variable costs incurred for various production quantities.

2. *Variable costs*

   Another common element that we observe in lot sizing models is the variable cost component associated with producing items. Unlike setup costs, this cost accounts for the individual costs incurred for producing items.

   As seen in Figure 1.4, the setup cost incurs a "fixed cost" that is independent from the size of production, whereas the total variable cost incurred is dependent on the number of items produced.

   The main difference with regards to including variable costs is related to the objective function, where a production cost of m_t for period t is incurred for every item manufactured on t. Thus, we may rewrite the objective function as:

$$\min \sum_{t=1}^{\mathcal{T}} (h_t^+ I_t^+ + h_t^- I_t^- + m_t x_t^m + K_t y_t) \tag{1.12}$$

   Note that the subscript t is dropped if production costs are time-independent

(i.e. production costs are equivalent across different time periods).

3. *Multi-echelon Production*

   Another crucial extension to the dynamic lot-sizing model is the concept of multi-echelon lot sizing (which is also often referred to as "multi-level lot sizing" in the literature). The general idea behind multi-level lot sizing is that there exist multiple levels of production (e.g. raw material processing, component processing, end-item processing etc.), where the output to a given level provides the input for another. This setting is relevant from a practical point of view, since in most production systems, items that are being produced in a given workstation are very likely to interact with others. Moreover, distinguishing different levels of production allows us to define specific costs for each level, which often leads to a more accurate representation of costs. Preliminary work on generating optimal production quantities in a multi-level setting include the works of Clark and Scarf [1960] and Zangwill [1969]. Taking these studies as basis, we provide a general representation of the multi-level dynamic lot sizing problem as follows:

$$\min \sum_{l=0}^{\mathcal{L}} \sum_{t=1}^{\mathcal{T}} (h_t^l I_t^l + m_t^l x_t^l + K_t^l y_t^l)$$

$$I_{t-1}^0 + x_t^0 = D_t + I_t^0 \qquad \forall t = 1, \ldots, \mathcal{T} \tag{1.13}$$

$$I_{t-1}^l + x_t^l = x_t^{l-1} + I_t^l \qquad \forall t = 1, \ldots, \mathcal{T}, \forall l = 1, \ldots, \mathcal{L} \tag{1.14}$$

$$x_t^l \leq M_t y_t^l \qquad \forall t = 1, \ldots, \mathcal{T}, \forall l = 0, \ldots, \mathcal{L} \tag{1.15}$$

$$x, I \geq 0 \tag{1.16}$$

$$y \in \{0,1\}^{\mathcal{T} \times \mathcal{L}+1} \tag{1.17}$$

   In the multi-level setting given above, we have additional variables to define the production quantities for each level in the set $\{0, \ldots, \mathcal{L}\}$, denoted as x_t^l for level l and period t. Similarly, we define inventory and setup decisions for each level as I_t^l and K_t^l, respectively. Note that we define level 0 as the level for the end-item, where customer demand needs to be satisfied (since this formulation does not include the option of backlogging). Additionally, we do not consider any independent demand on levels $\{1, \ldots, \mathcal{L}\}$. In Chapters 5

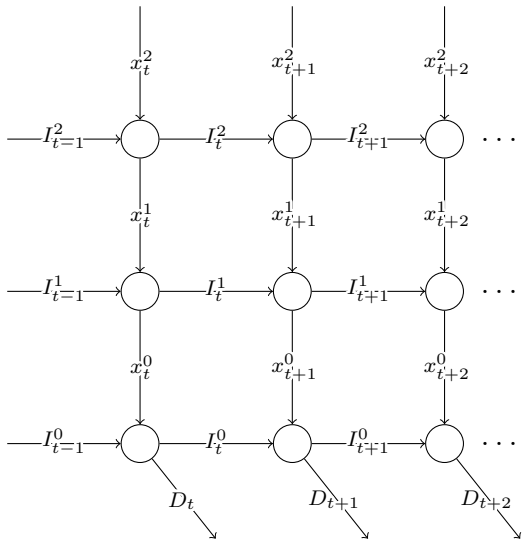

Figure 1.5: Network flow diagram for the multi-level dynamic lot sizing problem for $\mathcal{L} = 2$.

and 6, we consider special cases for the multi-level lot sizing problem, which is often classified as a *two-level lot sizing problem*. As we will further examine in detail, the *two-level lot sizing problem* has a total of two production levels (which are defined as levels 0 and $\mathcal{L} = 1$ in our work). As shown in the network flow diagram given in Figure 1.5, the output for a given level l is used as input for the previous (upper) level, $l - 1$, where $l \in \{1, \ldots, \mathcal{L}\}$.

4. *Multi-item Production*

   Another fundamental practice that is often observed in production systems is multi-item production, which has been long studied in lot sizing problems (we refer the reader to the works of Karmarkar and Schrage [1985] and Afentakis and Gavish [1986] for primary studies on multi-item models). In particular, multi-item production refers to systems where more than one type of item is produced. While these items may be produced under a single-level structure, they may also be part of multi-level production systems. Below, we introduce an introductory multi-level, multi-item formulation for a particular setting that aligns with our model assumptions (especially to those introduced in Chapters 5 and 6). More specifically, we are interested in the *two-level multi-item lot sizing problem* (which we also refer to as the

two-level multi-component lot sizing problem), where we consider multiple components, which are then assembled in order to produce a single final-item product (i.e. the item required to satisfy customer demand). Figure 1.6 demonstrates the flow of items, components and demand. Given this setting, we introduce the following mathematical formulation for the *two-level multi-component lot sizing problem*:

$$\min \sum_{t=1}^{\mathcal{T}} (h_t^0 I_t^0 + m_t^0 x_t^0 + \sum_{c=1}^{\mathcal{C}} (m_t^c x_t^c + h_t^c I_t^c + K_t^c y_t^c))$$

$$I_{t-1}^0 + x_t^0 = D_t + I_t^0 \qquad \forall t = 1, \ldots, \mathcal{T} \quad (1.18)$$

$$I_{t-1}^c + x_t^c = x_t^0 + I_t^c \qquad \forall t = 1, \ldots, \mathcal{T}, \ \forall c = 1, \ldots, \mathcal{C} \quad (1.19)$$

$$x_t^c \leq M_t y_t^c \qquad \forall t = 1, \ldots, \mathcal{T}, \ \forall c = 1, \ldots, \mathcal{C} \quad (1.20)$$

$$x, I \geq 0 \quad (1.21)$$

$$y \in \{0,1\}^{\mathcal{T} \times \mathcal{C}} \quad (1.22)$$

Here, $\{1, \ldots \mathcal{C}\}$ is the set of components that need to be produced in order to perform final assembly for the end item product. We denote the production quantities for components and end item assembly as x_t^c and x_t^0, respectively, where $c \in \{1, \ldots, \mathcal{C}\}$. Since this formulation considers two production levels, the inventory balance constraints show certain similarities to the *Multi-level lot sizing problem* introduced previously. Specifically, constraints (1.18) and (1.19) ensure the flow conversion for the end-item level, and the component level, respectively. Note that we do not consider independent demands on the component level. Since this is the case, the only "demand" that we need to satisfy for each component is equivalent to x_t^0, for a given period t. Finally, we have the traditional setup constraints defined in (1.20). Note that in this specific formulation, we do not consider setup decisions on the end-item level.

The extensions and formulations introduced above constitute for the basis for the formulations studied in this thesis. We provide a complete list of main variables presented in this section in Table 1.1, which are also used in the formulations

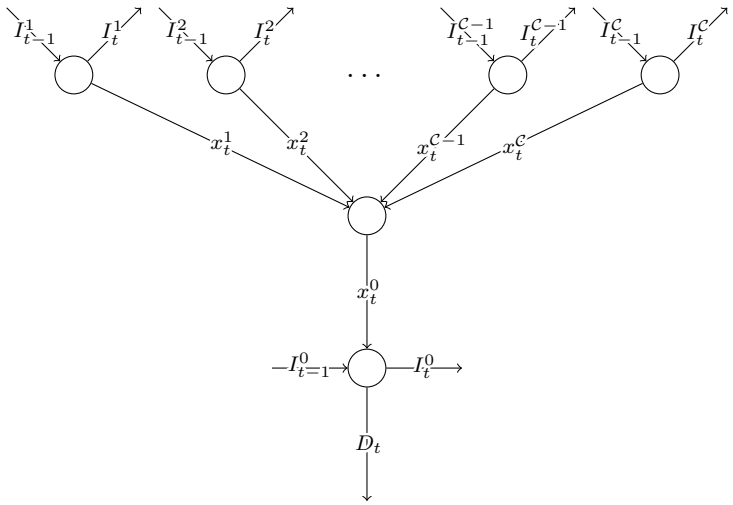

Figure 1.6: Flow diagram for the two-level multi-item (component) lot sizing problem for a given period t.

x_t^m	Number of items manufactured in t (components are not considered).
I_t	Serviceables inventory cost in t (components are not considered).
y_t	1 if setup occurrs in t, and 0 otherwise (components are not considered).
x_t^c	Number of components manufactured in t for $c = \{1, ..., \mathcal{C}\}$.
I_t^c	Serviceables inventory cost in t for component $c = \{1, ..., \mathcal{C}\}$.
y_t^c	1 if setup occurrs in t for component $c = \{1, ..., \mathcal{C}\}$.

Table 1.1: Decision variables for variations of the LSR problem.

considered in the remainder of this thesis. In the next section, we provide a detailed study on how the concept of remanufacturing can be implemented to these models. Before we explore these formulations, a general introduction to the concept of product recoveries is given in the next section, where their importance and role in production planning systems are discussed.

1.4 Product Recovery and Remanufacturing

Having introduced various traditional lot sizing models in Section 1.3, we now briefly describe the concept of product recovery, and examine how it can be incorporated into traditional lot sizing problems.

The concept of product recovery has been gaining interest in manufacturing

environments. Since production recovery reduces the need for producing items from scratch, such processes have an overall positive impact from an environmental perspective. In this sense, product recovery can be seen as a way of allowing manufacturers to reduce the total carbon emission related to production, which has been of interest to a majority of manufacturers due to increasing awareness on the need of maintaining sustainable means of production, and due to carbon emission restrictions imposed on manufacturers. Such restrictions involve carbon emission taxes and related policies, such as the carbon cap and trade policy. Undoubtedly, such practices have a considerable importance in terms of production planning, where various studies in the literature have addressed such problems (see Akbalik and Rapine [2014], Helmrich et al. [2015], Absi et al. [2016, 2013] and the references therein).

Integrating production recovery processes as a supplementary way of production to traditional manufacturing makes it very favourable for most manufacturers today, not only because it constitutes for an effective way of reducing waste, but also as it allows a reduction in the total production cost (Walsh et al. [2015]).

Among many other means of recovering products, remanufacturing is one of the practices that is being employed more frequently (Rogers and Tibben-Lembke [2001]). As a result of this, remanufacturing is practiced for a wide range of items in various sectors (De Brito et al. [2005], Agrawal et al. [2015], Guide Jr and Van Wassenhove [2009]). Such observations indicate the importance of studying methods that consider the option of integrating remanufacturing into traditional supply chain and production planning activities.

In order to do this, it is important to understand the characteristics of returned items and the process of embedding these items in traditional production environments. While there exists a considerable number of processes through which returns can be integrated into the traditional production flow (Thierry et al. [1995]), we specifically consider the option of remanufacturing throughout our study. Remanufacturing can be described as a practice that aims to recover returned products such that they are brought up to the standards of serviceable goods (i.e. items that are of as-good-as-new quality). As seen in Figure 1.7, remanufactured items are recovered such that these items are able to satisfy customer demand.

An important aspect to notice here is regarding the uncertainty around items that have been returned by customers. In most cases, the number of items returned by customers is highly random, where this can be due to the randomness

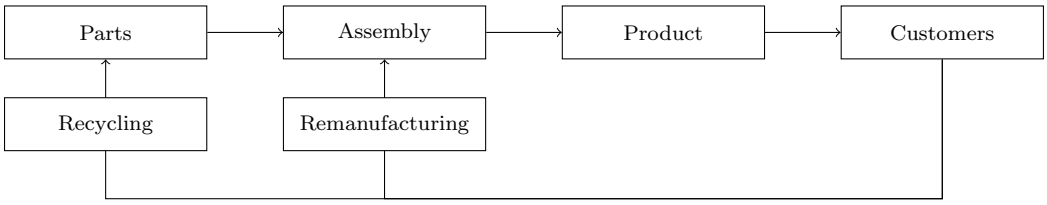

Figure 1.7: Stages of item recovery in production systems. Figure adapted from Thierry et al. [1995].

of the time of defection, or the unpredictability of the age of the product (Guide Jr and Van Wassenhove [2001], Thierry et al. [1995]). However, uncertainty around returned items is not only limited to the timing and quantity of returns, but also often related to the condition of the items returned. As a consequence, such uncertainties affect to which extent returns can be used. In the formulations presented under Chapters 3 and 4, we mainly address the issue of return uncertainty with regards to the timing and quantity of returns. In Chapter 5, we consider a slightly different setting where returns are recovered into components. The formulations presented under this chapter can be easily adapted so that they are able to handle various magnitudes of uncertainty around different components.

When it comes to integrating remanufacturing into traditional lot sizing problems, it becomes an important task to establish how remanufacturing is integrated to the traditional manufacturing process. One of the crucial aspects to determine is regarding the setup requirements for remanufacturing. In our work, we assume that a "joint setup" is carried over to produce items, where we have:

$$y_t = \begin{cases} 0, \text{ if } x_t^m + x_t^r = 0 \\ 1, \text{ otherwise} \end{cases} \tag{1.23}$$

where x_t^m indicates the number of items manufactured on a given period t, and x_t^r indicates the quantity of returns remanufactured.

It is important to note that an alternative setup setting that is also often utilized is the "separate setup" setting, where we have:

$$y_t^m = \begin{cases} 0, \text{ if } x_t^m = 0 \\ 1, \text{ otherwise} \end{cases} \tag{1.24}$$

and

$$y_t^r = \begin{cases} 0, \text{ if } x_t^r = 0 \\ 1, \text{ otherwise} \end{cases} \tag{1.25}$$

Note that the setup variables in this case are defined independently for manufacturing and remanufacturing as y_t^m and y_t^r, respectively.

In the next section, we provide details regarding different types of parameter uncertainties and relevant methods, as well as a detailed description on our assumptions regarding parameter uncertainty.

1.5 Parameter Uncertainty

Uncertainty is an inevitable challenge to tackle in most practical situations. As a result of this, uncertainty is undoubtedly a major issue to consider in mathematical programs. Ignoring the uncertainty around problem parameters can cause severe inaccuracies and infeasibilities in mathematical programs. In order to demonstrate the importance of considering parameter uncertainties, let us consider the following example from the work of Ben-Tal and Nemirovski [2000], which illustrates the impact of parameter uncertainty on the feasibility of a LP program from the `NETLIB` library:

Example 1.5.1. *Consider the following LP program:*

$$\min\{c^T x : Ax \leq b, x \in \mathbb{R}^n\}$$

and let us consider the following constraint in row i and the optimal solution to $x = x^*$:

$$A_i x^* \leq b_i$$

in the nominal case (i.e. the case where the values for A_i are known to certainty) this constraint has the following form in the optimal solution:

$$A_i x^* = 23.387405$$

In their example, Ben-Tal and Nemirovski [2000] show that imposing a negative perturbation as small as 0.1% on some of the values in A_i can lead to a significant

violation of this constraint. Doing so implies the following constraint:

$$A_i x^* = -81.5498$$

The significant impact of parameter uncertainty is evident from the large difference in the right hand side of the two cases given above, where a violation of $\frac{23.387405-(-81.5498)}{23.387405} \approx 450\%$ *is observed.*

This example is a clear indicator that even the smallest misinterpretation of problem parameters can lead to significant inaccuracies. Certainly, this is applicable to production planning problems as well, since parameters such as costs, quantity of customer demands and returns are highly impacted by uncertainty in practical cases.

A wide range of studies in the field of mathematical programming have addressed the issue of uncertainty, where various methods and assumptions around how parameter uncertainties are proposed. In order to provide an overview of how uncertainties can be modelled in mathematical programs, we provide a brief introduction to the following approaches that are commonly used to model parameter uncertainty: *stochastic optimization*, *chance-constrained optimization* and finally *robust optimization*, which forms the basis for the formulations given throughout this thesis.

Stochastic optimization

In stochastic programming, we have the following structure:

$$\min \; c^T x + E_\xi Q(x, \xi) \tag{1.26}$$

$$st. \; A_i x_i \leq b_i, \; \forall i = \{1, ..., n\} \tag{1.27}$$

$$x_i \geq 0, \; \forall i = \{1, ..., n\} \tag{1.28}$$

In the formulation above, ξ is defined as a *random variable*, and E_ξ represents the expectation associated with the outcome ξ.

The main difference between stochastic and robust optimization is with regards to how uncertainty is defined. While stochastic optimization considers distributional and probabilistic assumptions on uncertain parameters, robust optimization does not impose any distributional or probabilistic restrictions on these parameters. We refer the reader to Birge and Louveaux [2011] for

a complete review of methods and formulations used in stochastic programming.

Chance-constrained optimization

Another well-known method that is utilized to model uncertainties is chance-constrained optimization. The main idea here is to model constraints such that these are allowed to hold with a given probability. Among preliminary work in this area includes the study of Charnes et al. [1958], where we have constraints of the form:

$$P(A_i x_i \leq b_i) \geq \alpha_i \tag{1.29}$$

where α_i is defined as the *confidence coefficient* and x_i is a random variable. Under this setting, chance-constrained optimization models seek for a solution that satisfies constraints of this type with probability α_i. Birge and Louveaux [2011] provide the general structure and solution methods for such problems.

Robust optimization

In robust optimization, we consider that uncertain parameters are defined as parts of uncertainty sets, where we have the following structure:

$$\min \ \{c^T x : A_i x_i \leq b_i, (c, A, b) \in \mathcal{U}, \forall i \in \{1, ..., n\}\} \tag{1.30}$$

In this formulation, problem parameters (c, A, b) are unknown. We assume that the realization of these parameters lay in the uncertainty set $\mathcal{U}$. Given this structure, solutions to this mathematical program have to remain feasible for *any* realization in $\mathcal{U}$, where no particular information is accounted for regarding the likelihood of a given scenario occurring. This results in an approach where the "worst-case-scenario" is sought, where remaining feasible for this particular scenario implies that the scenarios in $\mathcal{U}$ are taken into account. As we will further discuss in detail in Chapter 2, a wide variety of methods in literature aim to solve problems of this nature.

The variety of methods used to model uncertainty in literature is vast and certainly not only limited to the examples provided above. The main advantage of using robust optimization over other methods listed above is with regards to

its non-probabilistic assumptions. Since robust optimization does not impose any specific distributional information on uncertain parameters, it is widely applicable in practice. This is because in most practical situations, the distributional information on uncertain parameters is unknown (or if known, difficult to validate). From the perspective of remanufacturing, robust optimization is also favourable, since the rate, timing and condition of returns is highly random, as we have discussed in Section 1.4. Since our work aligns closest with robust optimization, we particularly focus on introducing methods in this domain in Chapter 2.

Although robust optimization is favorable from the points of view discussed above, it certainly has limitations with regards to its dependency on the uncertainty sets. Accurate definition of these sets is crucial to derive correct solutions and costs. As a result, failing to identify these sets correctly can easily lead to incorrect formulations and reformulations.

1.6 Outline of the Thesis

Following the introduction provided under this chapter, a literature review on relevant mathematical models and methods is given in the Chapter 2.

In Chapter 3, the deterministic and robust lot sizing formulations with remanufacturing are formally stated. In addition to this, the structure and properties of uncertainty sets considered in these formulations are examined in detail.

Chapter 4 is dedicated to a min-max decomposition approach that can be used to solve the robust lot sizing problem with remanufacturing introduced in Chapter 3. Upon presenting this method and providing details regarding how this method can be implemented, further reformulations for the master problem are proposed. Following this, a wide range of computational tests are presented, where the performance of these formulations is compared.

Chapter 5 considers a different setting for the lot sizing problem with remanufacturing, where two production levels and multiple components are required to produce the item demanded by customers. Under this section, we present a robust formulation for this case, along with the relevant uncertainty sets. In addition, we examine a production rule that holds for this problem and discuss its importance under a min-max decomposition framework. Finally, we provide computational experiments where various observations regarding the impact of different cost structures on the problem performance are made.

Chapter 6 also focuses on the two-level multi-component case. In this section, we only provide the deterministic problem for a different variation to the problem introduced in Chapter 5, and present several optimality conditions that hold for this problem. As shown under this chapter, such conditions can be used in the dynamic programming framework proposed by Teunter et al. [2006] to derive solutions to this problem.

Finally, conclusions regarding the formulations, assumptions and computational experiments are presented in Chapter 7. In addition, future research directions and open questions are provided.

Chapter 2

Literature Review

In this chapter, we provide a literature review on production planning problems and their applications. We then provide a review on the implications of implementing remanufacturing on such traditional problems. Next, we present a review on parameter uncertainty and discuss methodologies that aim to tackle such problems, including robust optimization methods. Finally, we provide an overview of studies where robust optimization is used to address parameter uncertainty in production environments.

2.1 Classical Production Planning Problems

Finding cost-efficient production plans has been a problem of interest in production environments since early 1900s. The work of Harris [1913] is among preliminary work that tackles this problem, where the *Economic Order Quantity (EOQ) model* is introduced. Although the EOQ model is of fundamental importance, most of its assumptions are no longer applicable to many manufacturing systems today. Specifically, examples to the assumptions made in the EOQ model involve that the customer demand is stationary and time invariant order costs. Given the complexity of manufacturing environments today, such assumptions are likely to lead to an inaccurate representation of the production planning environment.

The dynamic lot-size model discussed earlier under Chapter 1 is undoubtedly among significant preliminary studies on lot-sizing models. Among other preliminary work of crucial importance is Zangwill [1966, 1969], where the dynamic lot-size model is extended to one where the option of backlogging is allowed. Moreover,

Florian et al. [1980] has further extended the classical dynamic lot sizing model, where production capacities are imposed. Their work also provides insights regarding the complexity of this problem, where the case with varying capacity levels is shown to be $\mathcal{NP}$-hard. Given the high computational requirements for solving problems of this nature, several preliminary studies have proposed heuristics to solve the lot sizing problem with capacities. Some examples to such heuristics include Diaby et al. [1992] and Kirca and Kökten [1994].

Due to the growing interest on lot sizing models, the variety of assumptions and settings considered in production planning problems today is extensive (we refer the reader to Pochet and Wolsey [2006] for a thorough review and the references therein), where the volume of studies related to production systems are continually growing Brahimi et al. [2017]. In the remainder of this review, we primarily focus on studies that consider MIP models to formulate classic lot sizing models and discuss the advancements on different extensions for the classical lot sizing problem.

As Pochet and Wolsey [2006] demonstrate in their excellent review of mathematical models developed for lot-sizing problems, the body of research devoted to the topic is extensive, covering a rich set of tools including. Among crucial methods that are utilized to solve lot sizing problems optimally include studies that tackle the problem of generating strong valid inequalities and description of the convex hull. The work of Barany et al. [1984] derives strong valid inequalities for the capacitated, multi-item lot sizing problem, as well as facet defining inequalities for the uncapacitated case when a single item setting is considered. Moreover, Agra and Constantino [1999] consider uncapacitated production with backlogging where a single item is considered with Wagner-Within costs. Other studies that consider an uncapacitated setting involve Van Hoesel et al. [1994], where a single item is considered with setup costs. In their work, Van Hoesel et al. present strong valid inequalities for this problem, which are shown to provide a complete description of the convex hull.

In more recent work that explore the polyhedral properties of lot sizing models include Akbalik and Pochet [2009], where the single item case with step-wise production costs are considered. In their work, Akbalik and Pochet present a new class of valid inequalities and define conditions that lead to facet-defining inequalities. Moreover, Küçükyavuz and Pochet [2009] provide strong valid inequalities for the uncapacitated case with backlogging, where these are shown to define the convex

hull of the feasible solutions to the problem. In addition, the work of Zhang et al. [2012] consider a different setting, where multiple echelons with independent intermediate demands are considered under an uncapacitated setting, for which strong valid inequalities are derived. A thorough survey on recent theoretical advances in single item lot sizing methods is given in Brahimi et al. [2017].

Another important aspect to obtain computationally efficient formulations is to analyze problem bounds and extended reformulations. Crucial contributions in this regard involve the work of Van Vyve et al. [2014], which considers multiple variations of the lot sizing problem with two-levels, for which strong relaxations are given. Additionally, Anily et al. [2009] consider a multi-item setting with joint setup costs, for which tight linear reformulations are given. Moreover, Krarup and Bilde [1977] introduce the facility location formulation for lot sizing problems. Among other examples that have introduced such reformulations include Eppen and Martin [1987], where a shortest path reformulation is given.

There have been numerous studies in the field that aim further provide strong properties for lot sizing problems, which can then be used to derive efficient optimization procedures for solving these problems. Such studies include the work of Wu et al. [2011], proposing formulations for the capacitated, multi-level case lot sizing problem with backlogging, where the LP relaxations obtained from these models provide good lower bounds on this problem. Furthermore, Akartunalı and Miller [2012] provide lower bounds for the multi-level case, where big bucket capacities are considered. A different setting is considered in Brahimi et al. [2006] where multiple items, capacities and time windows are taken into account, for which various relaxations of the problem are given. In addition to such methods, decomposition techniques can be used to solve lot sizing problems. For instance, the study of Jans and Degraeve [2004] employs a decomposition approach for deriving strong lower bounds for the capacitated lot sizing problem with setup times.

In addition to the exact methods used to solve such problems, numerous studies have considered non-exact methods such as heuristics Stadtler [2003], Akartunalı and Miller [2009], Wu et al. [2018] customized for use with real-world instances inherent in manufacturing systems.

Finally, we refer the interested reader to Brahimi et al. [2017] for a thorough review of single-item problems, and to Doostmohammadi and Akartunalı [2018] for a recent overview on complex multi-item lot-sizing problems. Besides exact solution methods, we refer to Jans and Degraeve [2007], providing a thorough

review on meta-heuristics that have been developed to solve lot sizing problems. Lastly, a review of modelling of lot sizing problems used in industrial applications is given in the work of Jans and Degraeve [2008].

2.2 Lot Sizing with Remanufacturing

Often, the extensions studied in production planning problems is motivated by shifts in production-related applications in practice, encouraging the studies to expand further. Remanufacturing is among the practices that have been gaining increasing interest over the recent years. Consequently, various papers have addressed the issue of considering customer returns in production planning problems.

Remanufacturing is employed in a wide range of industries. To name a few, automotive, electronics, medical equipment and white goods sectors are among examples where remanufacturing has an active role in production. As an example, a company that actively remanufactures its products is Apple. The company encourages their customers to return their used products, which can then be remanufactured into an as-good-as-new item for a much smaller cost. Other examples involve remanufacturing activities for automobile parts such as generators, starters, pumps and automatic transmissions. BMW is among companies that directly sell such refurbished parts, through which costs can be cut up to 50%. Another well-known automobile manufacturer that remanufactures parts of their products is Toyota. The company also offers repairs using remanufactured parts and components.

Remanufacturing is a type of product recovery, where items that are no longer functioning are collected from customers back into the production system. Upon the collection of these returned items (or "returns" in short), they are inspected, disassembled, reassembled and fed back into the regular production process (Thierry et al. [1995], Vlachos and Dekker [2003]). More simply, remanufacturing involves the process of recovering used products by repairing and replacing worn out components so that a product is created at least at the same quality level as a newly manufactured product, providing not only an environmentally sustainable alternative to classical manufacturing, saving tonnes of landfill every year, but also offering many industries from car engines to office copiers the potential for significant savings through the exploitation of used product inventories and many precious raw materials that are becoming scarcer (Ijomah [2009]). Remanufactur-

ing can be operated either under a dedicated system (i.e., remanufacturing only) or a hybrid system (remanufacturing combined with manufacturing), and most remanufacturing operations in European countries, as noted by Li et al. [2009], employ a hybrid system. In the context of lot-sizing, hybrid models also vary between different industries and also different products, some of which allow production setups to occur jointly for manufacturing and remanufacturing, referred to as "joint setup" systems, and others requiring separate production setups, referred to as "separate setup" system. The formulations presented in this thesis investigate a hybrid system with joint setups.

The influence of remanufacturing on lot sizing problems is very recent as its practical applications are also relatively new. The earlier works of Fleischmann et al. [1997] discussed the implications of the emerging reuse efforts with a review of the mathematical models proposed in the literature. Further preliminary work on integrating remanufacturing to mixed integer programs includes the work of Jayaraman et al. [1999]. In their work, they present a closed-loop logistics system and optimize the total costs associated with decisions regarding the transportation, remanufacturing, as well as facility opening and operating costs. On the other hand, the work of Richter and Sombrutzki [2000] and Richter and Weber [2001] are among the first studies that have specifically focused on lot sizing problems with remanufacturing, where the Wagner-Whitin algorithm is applied to remanufacturing systems. In their first study, all costs are assumed to be stationary and traditional manufacturing activities are ignored, whereas the work of Richter and Weber [2001] has extended these assumptions by considering time-variant costs and joining manufacturing with remanufacturing.

Another common practice in remanufacturing environments is to dispose of the returns for a given cost. This option was initially integrated into the lot sizing problem with remanufacturing by Golany et al. [2001], where the problem was shown to be $\mathcal{NP}$-complete for general concave costs. For the linear cost case of this problem, they provide a network flow model formulation, and a dynamic programming algorithm that runs in $O(T^3)$. Further studies on the complexity of the problem involve the work of Retel Helmrich et al. [2014], where it is shown that the problem is $\mathcal{NP}$-hard for most general configurations, including settings that consider separate and joint setups on manufacturing and remanufacturing, and time invariant separate setup costs. Additionally, a shortest path reformulation is presented for the problem, which is shown to outperform the original formulation

of the problem. An important contribution to solving LSR problems efficiently was made by Teunter et al. [2006], where a polynomial time algorithm with $O(T^4)$ time is presented for the joint setup case with time invariant costs, as well as modifications to the well-known "Silver-Meal", "Least Unit Cost" and "Part Period Balancing" heuristics. The capacitated version of the problem was studied by Pan et al. [2009], where the problem is shown to be equivalent to the traditional capacitated lot sizing problem when capacities are assumed to be constant. More recent studies on the capacitated version of the problem involve work of Akartunali and Arulselvan [2016], where two classes of valid inequalities are introduced for the problem. Further complexity results were provided by Yang et al. [2005] for the case of concave costs, by Pan et al. [2009] for constant capacities, by Akartunali and Arulselvan [2016] for special cost structures and by Pineyro and Viera [2010] for the case with a disposal option. We also remark some recent and effective heuristics for practical size problems as presented in Baki et al. [2014], Sifaleras and Konstantaras [2017], the recent polyhedral study of Syed Ali et al. [2018] comparing various reformulations and valid inequalities, and the recent work of Kilic and van den Heuvel [2019] showing optimality properties and how these can be used to decompose the problem.

Although various studies address the option of remanufacturing, studies that consider uncertainties in such settings is very scarce. The formulations considered in this thesis aim to contribute to this research gap through utilizing robust optimization methods.

2.3 Parameter Uncertainty

2.3.1 Robust Optimization

As discussed in Chapter 1, robust optimization is a widely-used and effective methodology for handling uncertainties in mathematical models. The preliminary studies in this domain date back to 1970s, with the work of Soyster [1973]. Although there have been several extensions following this study (such as the papers of Falk [1976] and Singh [1982]), the vast majority of literature in this field is relatively recent.

More specifically, there has been many advances in the field of robust optimization over the last 15 years since the papers of Ben-Tal and Nemirovski [1998, 1999],

where uncertainty sets are defined as ellipsoids. These studies are also among the first to present the robust counterpart approach, which is a deterministic problem that is used to implement the uncertainty sets to the original problem. They also show that the robust counterpart problem obtained by ellipsoidal uncertainty sets is tractable. However, robust counterpart problems of this type are non-linear, which limits the applicability of this approach greatly.

A common observation in robust optimization problems is that due to the general structure of robust problems, such formulations are defined as $\mathcal{NP}$-hard (Ben-Tal et al. [2004], Ben-Tal and Nemirovski [1998]). Generally, the complexity of these problems depends strongly on the tractability of the robust counterpart associated with the problem. This raises a fundamental challenge, since practical application of such problems are heavily dependent on their computational performance. As a consequence, there have been a vast variety of studies that have addressed methods and uncertainty sets to obtain effective robust counterparts and robust solutions. In this section, we mainly focus on approaches that have been utilized commonly within the literature. We refer the reader to Gorissen et al. [2015], Bertsimas et al. [2011], Gabrel et al. [2014], which include a thorough overview of the methods and uncertainty sets that are used within this paradigm.

The common idea behind these methods is to define uncertainty sets for unknown or semi-known problem parameters, and to find procedures to obtain meaningful solutions that remain feasible within such uncertainty sets. In order to do this, uncertainty sets are integrated into the problem of interest through ensuring that the problem remains feasible for all elements in these sets. However, by constructing a problem that is immune against all potential parameter values often worsens the optimal solution greatly for the sake of maintaining feasibility. This phenomenon is often referred to as *"conservativism"* in the literature, and numerous studies have been introduced to address this problem. The first papers that undertake the issue of over-conservativism include the work of El Ghaoui and Lebret [1997], where convex second-order cone programming is used to overcome conservative solutions. Furthermore, El Ghaoui et al. [1998] considers semidefinite programs where problem constraints remain feasible for with a high probability while generating robust solutions.

The methods and uncertainty sets proposed in the papers of Bertsimas and Sim [2004, 2003] have been widely used to define and solve robust optimization problems, where uncertainty sets are defined as budgeted polytopes to handle problems

with discrete variables more effectively. More specifically, these studies have made crucial contributions to the field by introducing a new robust counterpart approach that is formulated as a linear program. This is not only crucial from the perspective of problem tractability, but also important to overcome over-conservativism. Essentially, the idea here is to introduce new parameters, Γ, which limits the number of variables that are allowed to take extreme values, thus preventing conservative solutions. As we will further discuss in detail, the structure of uncertainty sets provided in the study of Bertsimas and Sim [2004] play an essential role for the uncertainty sets considered throughout this thesis.

One of the approaches that is used commonly in the literature is adjustable robust optimization. This involves a multi-stage procedure, where robust decision variables are modelled as functions of the revealed data. Through defining these decisions as functions of such realizations, we are able to obtain much less conservative robust counterparts while maintaining tractability Ben-Tal et al. [2004], Chen and Zhang [2009]. This setting provides a useful/meaningful structure for dynamic problems, where decisions are time independent. In this regard, applications of adjustable robust optimization include Ben-Tal et al. [2005], looking into finding optimal quantities for supplier-retailer contracts. In addition to this, the work of Atamtürk and Zhang [2007] considers a two-stage setting with uncertain demand, while showing applications to classical lot-sizing problems, including those with multiple levels.

Another interesting aspect considered in adjustable robust optimization problems is to use affine decision rules introduced in Ben-Tal et al. [2004], where its applications in supply chain problems are studied by Aharon et al. [2009]. On the other hand, methods that focus on increasing the efficiency of general classes of robust problems is the work of Koster et al. [2013], where a set of cutset inequalities are introduced. In addition, there has been interest in combining the tractability of robust optimization with the framework of stochastic optimization Fischetti and Monaci [2009].

An area that has been receiving much attention in literature recently is *distributionally robust optimization*. The main idea behind *distributionally robust optimization* is to apply the paradigm of robust optimization to a given set of distrubtions, where the *distributionally robust* solution remains feasible for the worst-case solution among a given set of given distributions. This becomes a favourable approach in practical problems, in cases where the distributional struc-

ture of the unknown parameters is not known to certainty. Fundamental studies in this domain include the work of Wiesemann et al. [2014], providing a general framework and tractable ambiguity sets, as well as Goh and Sim [2010], where formulations for such problems are provided, including multistage settings.

It is also important to note that a crucial element for constructing efficient and precise robust optimization problems is to determine tractable and precise uncertainty sets. Examples that aim to further improve the solution quality through managing conservativeness and increasing users' control over the uncertainty sets while retaining a polyhedral structure is the works of Büsing and D'Andreagiovanni [2012] and Büsing et al. [2014], where multiband uncertainty sets are introduced. Here, we consider the typical budgeted uncertainty sets in bands, where we are able to determine the exact number of robust variables in each band. In addition, methods for constructing data-based uncertainty sets have also been studied. An example is the work of Bandi and Bertsimas [2012], where central limit theorem is used to derive uncertainty sets from historical data. Other data-driven uncertainty sets involve using the coherent risk measures Bertsimas and Brown [2009], introduced by Artzner et al. [1999].

Another common approach to solve robust problems is to use cutting plane methods to eliminate solutions that are not feasible over the uncertainty sets. This approach is known to be tractable for solving noncompact problems, where in cases that utilize polyhedral uncertainty sets, such methods are known to result in a similar performance when compared to other static robust optimization approaches (Fischetti and Monaci [2012], Bertsimas et al. [2016]). More specifically, the general idea of cutting plane approaches is to generate cuts to eliminate non-robust solutions. A variation of this approach is often referred to as the "adversarial approach", where the robust problem is solved for a subset of points in a given uncertainty set. Then, an "adversarial" (or "auxiliary") problem seeks for a new point in from the original uncertainty set, which violates the optimal solution to the restricted uncertainty set. Usually the adversarial problem here consists of one where the worst-case scenario is sought. Often, the adversarial problem involves a contradictory objective to the original problem, as the main objective here is to worsen the solution to the original problem. There have been numerous studies that have addressed this approach, such as the work of Mutapcic and Boyd [2009], where pessimistic oracles for the auxiliary problem are examined in detail. Furthermore, the work of Zeng and Zhao [2013] considers a cutting plane approach

that can be used to solve two-stage robust problems, where procedures of generating optimality cuts and the corresponding complexity measures are examined. Further examples to studies that consider cutting plane approaches include Thiele et al. [2009], where a two-stage setting is examined, and a cutting-plane approach based on Kelley's algorithm is proposed.

Among crucial contributions that seek for robust solutions through generating cuts is the work of Bienstock and Özbay [2008], where an adversarial approach for computing basestock levels is introduced. Specifically, the problem considered in this paper involves unknown demands in supply chains, where the main objective of the decision maker is to minimize total costs. Since the adversarial problem seeks for the worst-case scenario, its objective is to maximize the scenario-dependent inventory and backlogging costs. This approach can be seen as a variation of *Benders' decomposition*, as it involves generating problem-specific cuts to obtain a robust solution. As we will examine in further depth in the upcoming chapters, their study plays an essential role for the formulations that we consider in this thesis.

The following section focuses on production planning problems under parameter uncertainty, where an overview of the existing literature on applications of robust and stochastic optimization are presented.

2.3.2 Production Planning Under Parameter Uncertainty

Since production planning problems are heavily dependent on information that are often not known exactly such as costs, customer demand (and in case of remanufacturing, customer returns), methods that focus on implementing parameter uncertainties have been studied extensively in literature. In this section, an overview of methods for solving production planning problems is presented.

The volume of studies that have considered parameter uncertainty in production planning problems is very extensive, especially when compared to ones that take remanufacturing into account. Among studies that have considered a stochastic setting is the work of Guan and Miller [2008b], where a polynomial time algorithm is proposed to solve the stochastic uncapacitated lot sizing problem under uncertain demand. In addition, other studies that consider a similar setting include Halman et al. [2009], where a single-item stochastic problem is considered where demands are identified as independent random variables. In their work,

the problem is shown to be $\mathcal{NP}$-hard, where approximation algorithms to solve this problem are proposed. Furthermore, Guan et al. [2006] consider a multi-stage stochastic setting integer programming setting, showing that (l, S) inequalities for the deterministic case of the problem are valid under stochastic assumptions. These inequalities are given as

$$\sum_{i \in S} x_i + \sum_{i \in \bar{S}} d_{il} y_i \geq d_{1l}$$

and the sets are given as $l \in \{1, 2, \ldots, T\}$, $S \subseteq \{1, 2, \ldots, l\}$, $\bar{S} = \{1, 2, \ldots, l\} \setminus S$ where $d_{ij} = \sum_{k=i}^{j} d_k$ is given. Following this, they present a new set of valid inequalities, which are then utilized to derive a branch-and-cut procedure for this problem.

In addition to the studies where stochastic assumptions are made on the unknown parameters themselves, See and Sim [2010] have analyzed the setting where instead of assuming exact distributional information on uncertain demands, only limited information on several characteristics of demands are given. Another similar example to this is the work of Liyanage and Shanthikumar [2005], addressing the issue of demand uncertainty for the newsvendor inventory control problem, where the distribution of demands is assumed to belong to a family of distributions.

Whereas the majority of production planning and inventory problems focus on the minimization of the total operational cost, other studies have considered a different setting, which allows the decision maker to apply other types of uncertainty measures. An example to this is the study of Ahmed et al. [2007], which considers a linear-cost inventory problem, where the objective function is defined as a coherent risk measure, and demand is defined as a random process.

In addition to the studies where complete or partial stochastic assumptions are made, various studies consider classical lot sizing and inventory problems under robust settings. The study of Bertsimas and Thiele [2006] considers a classical lot sizing problem with demand uncertainty. The uncertainty sets and methods presented in their study considers the budgeted polytope setting given in Bertsimas and Sim [2004]. Moreover, Bienstock and Özbay [2008] presents a decomposition framework to determine optimal robust basestock levels. Although the setting considered in this work is in a larger scale in comparison to lot sizing models, most of the concepts that are unique to lot sizing problems (such as setup and production

costs) can be solved using this approach. Among studies that employ this setting is the work of Agra et al. [2016], which considers the budgeted uncertainty polytope. Here, a dynamic programming approach for the adversarial problem is given. Their work also presents applications of this approach to classical lot sizing problems.

As seen from the examples given above, the number and variety of studies that consider uncertainty (in particular, uncertainty on demands) in inventory and lot sizing problems is large. While the studies on production planning problems have been studied in a robust setting (Aloulou et al. [2014] for a survey of non-deterministic lot sizing models), there are very few studies that consider remanufacturing systems under this framework. Under robust assumptions, the only study that we are aware of is Wei et al. [2011], where the robust lot sizing problem with remanufacturing is solved through employing the methods and uncertainty sets given in the work of Bertsimas and Sim [2004]. However, it is worth noting that there have been few studies that consider remanufacturing in a stochastic setting. Examples to this include the work of Macedo et al. [2016], where uncertainty is imposed on customer demands, returns and setup costs, and Hilger et al. [2016], where customer demand and return are uncertain and defined as random variables.

Under the given overview of existing literature, our main contribution is to introduce efficient robust optimization formulations to solve the robust lot sizing problem with demand and return uncertainties, where no probabilistic assumptions are made. As we further discuss in the upcoming chapters, our work closely aligns with the robust decomposition framework given in Bienstock and Özbay [2008].

2.4 Mixed-Integer Linear Programming

In this section, a brief overview of the methods and relevant concepts in the area of mixed-integer linear programming is presented. The main focus of this section is to address methods that are consulted in order to solve the problems we present in the upcoming chapters.

Following the work of Dantzig [1951], incorporating integrality constraints on classical linear programming problems have started gaining significant interest. This is due to the highly applicable nature of discrete variables on practical problems. Consequently, the volume of methods proposed to solve and improve MILPs is vast. We refer the reader to Wolsey [1998], Wolsey and Nemhauser [1999],

Schrijver [1998] for a thorough overview of the methods used to solve and improve MILPs, and Jünger et al. [2009], where the progression of fundamental methods and concepts in the domain of MILP are presented.

Preliminary studies in this domain include Gomory et al. [1958], Gomory [1960], where the idea of generating cutting planes to obtain integer solutions is presented. This concept is crucially important and widely used while solving most MILPs today, due to their highly complex nature.

The cutting plane approach involves generating *valid inequalities* that aim to eliminate fractional solutions, while ensuring feasibility for original problem constraints. In literature, there have been numerous studies that addressed the problem of generating valid inequalities. Undoubtedly, generation of such cuts is an interesting concept for production planning problems. We refer the reader to Pochet and Wolsey [2006] for a thorough discussion on the use and generation of valid inequalities for production planning problems.

Another approach that is able to generate effective solutions to MILPs is *Benders' decomposition* (Benders [1962], Geoffrion [1972]). The idea here is to decompose the original MILP problem into two subproblems, where a subset of decision variables are solved for a fixed feasible solution in the master problem. Using the solution generated to this problem, a cut is generated, and fed back into the master problem. This process is continued until convergence is achieved. Given this structure, *Benders' deocmposition* is highly applicable to various problems, and constitutes for an efficient method for finding optimal solutions to large-scale problems. Rahmaniani et al. [2017] provides a thorough review on combinatorial optimization problems that employ Benders' decomposition method. As previously discussed in Section 2.3, *Benders' decomposition* method is also utilized within the context of robust optimization by Bienstock and Özbay [2008].

Chapter 3

Implementing Uncertainty

3.1 Introduction

Although there is extensive literature on lot-sizing problems, the main focus has been on deterministic problems, where problem parameters such as demands have been assumed to be known a priori. When uncertainties are present and they cannot be sufficiently described using probability distributions (e.g., due to shortage of reliable data, or data not properly fitting into any distribution), robust optimization offers a solution that will be feasible for any realization taken in the so-called uncertainty set, i.e., the collection of all possible realizations.

The uncertainties are in particular critical in the remanufacturing setting, where both returns (numbers and product qualities) and demands are unknown, affecting both levels of the operations and their interactions in between. Although, robust approaches have been considered for classical lot-sizing problems e.g. Ben-Tal et al. [2005], Bertsimas and Thiele [2006], Bienstock and Özbay [2008], we are only aware of the work of Wei et al. [2011] in this area, who consider a lot-sizing problem with uncertain demands and returns, and use a robust linear programming formulation. However, the robust formulation is based on the static approach introduced for inventory problems by Bertsimas and Thiele [2006] which is known to produce very conservative solutions for some instances, as noticed in Bienstock and Özbay [2008].

Our recent preliminary study presented in Attila et al. [2017] proposed an exact approach, and therefore, less conservative than the one from Wei et al. [2011], which is known as adversarial approach (see Gorissen et al. [2015]) and was intro-

duced by Bienstock and Özbay [2008]. This is a decomposition approach where a robust optimization min-max problem is decomposed into a master (minimization) subproblem and an adversarial (maximization) subproblem. By exploring the properties of the adversarial problem for budgeted polytopes, Agra et al. [2016] proposed a general robust optimization dynamic programming framework that is shown to work effectively in lot-sizing problems. However, for many practical lot sizing problems, as the one considered here, the master subproblem is computationally harder to solve than the adversarial subproblem and, to the best of our knowledge, there is no work in this area studying the underlying mathematical structures for the master subproblems, such as the polyhedral characteristics or extended reformulations, which are essential ingredients in many lot-sizing algorithms developed in deterministic problems. On the other hand, it is worth to remark that stochastic programming has been used for such understanding, albeit in a very limited sense, e.g., see Guan and Miller [2008a]. Thus, there is clear potential to gain invaluable insights in the remanufacturing problems by studying their mathematical properties using robust optimization.

3.2 Problem Definition

In this section, we first introduce the deterministic lot-sizing problem with remanufacturing (LSR) option, where we introduce our notations and specify our assumptions on cost structures, inventories and handling of returned items. We then introduce the uncertainty involved in returned items and demands and formally define the robust LSR (RLSR) problem. We then provide a formulation using a min-max approach to obtain a robust production plan. A schematic diagram providing an overview of the production process can be seen in Figure 3.1.

3.2.1 Deterministic LSR Formulation

We now consider LSR presented in the preliminary study of Attila et al. [2017]. In this problem, we consider a time horizon T, a set of deterministic demands, $\mathcal{D} = \{D_1, D_2, \ldots, D_T\}$, and a set of deterministic returns, $\mathcal{R} = \{R_1, R_2, \ldots, R_T\}$, over the time horizon. Demands are to be satisfied by items that are produced, which can be achieved either by manufacturing an item from scratch or remanufacturing a returned item. We consider the setting where the costs involved are time invariant.

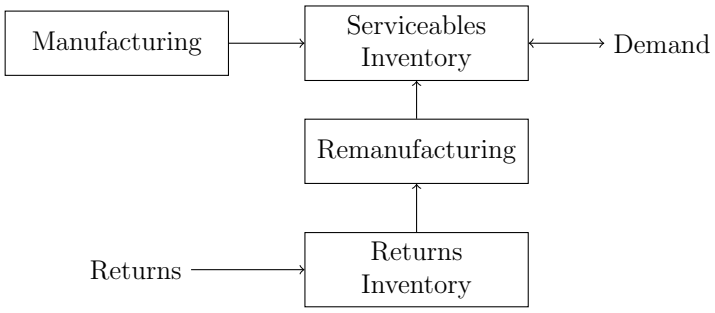

Figure 3.1: The production process with returns and remanufacturing

At every time period, we incur a variable cost of m (resp. r) per item manufactured (resp. remanufactured) and a fixed joint set up cost of K if an item was produced in that period. Note that all costs are positive. We assume that both manufactured and remanufactured items, referred to as "serviceable items", achieve the minimum quality level necessary for satisfying the demand. In a given time period, the serviceable items at hand can be in excess or short of the demand at that period. Excess serviceable items are carried over as serviceable inventory at a cost of h^s per item to the subsequent time period and can be used to satisfy future demands. Unsatisfied demands are backlogged at a cost of b per item from the subsequent time period and have to be satisfied by serviceable items that are produced at a future time period. At each time period, we have the option to remanufacture the returned items at hand or carry them over to the next time period as unprocessed returns in the "return inventory" at a cost of h^r per item, or dispose them at a cost of f per item.

The output of LSR comprises of a production plan that satisfies the demand at every time period and minimizes the overall costs involved. A production plan needs to specify the amount of manufactured and remanufactured items, return and serviceable inventory levels, amount of backlogged and disposed items for every time period over the planning horizon. Table 3.1 presents a complete list of the decision variables.

We define the vectors $x^m := (x_1^m, x_2^m, \ldots, x_T^m)$, $x^r := (x_1^r, x_2^r, \ldots, x_T^r)$, $d := (d_1, d_2, \ldots, d_T)$, $y := (y_1, y_2, \ldots, y_T)$, $s^m := (s_1^m, s_2^m, \ldots, s_T^m)$, and $s^r := (s_1^r, s_2^r, \ldots, s_T^r)$ associated with production (manufacturing and remanufacturing), disposal, setup and inventory (serviceable and return) variables, respectively. Let $\mathbf{x} := (x^m, x^r, d, y)$. Then, the objective is to minimize the total operational cost defined by:

x_t^m	Number of items manufactured in period t.
x_t^r	Number of items remanufactured in period t.
H_t^s	Serviceables inventory cost incurred in period t.
H_t^r	Returns inventory cost incurred in period t.
y_t	1 if setup occurred in period t, and 0 otherwise.
d_t	Number of returns disposed in period t.

Table 3.1: Decision variables for the deterministic LSR problem.

$$\theta^{D,R}(\mathbf{x}) + \sum_{t=1}^{T}(H_t^s + H_t^r), \tag{3.1}$$

where

$$\theta^{D,R}(\mathbf{x}) = \sum_{t=1}^{T}(Ky_t + mx_t^m + rx_t^r + fd_t) \tag{3.2}$$

W.l.o.g., we assume initial inventory levels are zero. Then, a mixed integer program (MIP) formulation for LSR is as follows:

$$\min \quad \theta^{D,R}(\mathbf{x}) + \sum_{t=1}^{T}(H_t^s + H_t^r) \tag{LSR-D}$$

st.

$$H_t^s \geq h^s \sum_{i=1}^{t}(x_i^m + x_i^r - D_i) \qquad \forall t = 1, \ldots, T \tag{3.3}$$

$$H_t^s \geq -b \sum_{i=1}^{t}(x_i^m + x_i^r - D_i) \qquad \forall t = 1, \ldots, T \tag{3.4}$$

$$H_t^r \geq h^r \sum_{i=1}^{t}(R_i - x_i^r - d_i) \qquad \forall t = 1, \ldots, T \tag{3.5}$$

$$M_t y_t \geq x_t^m + x_t^r \qquad \forall t = 1, \ldots, T \tag{3.6}$$

$$x^m, x^r, d \geq 0, \tag{3.7}$$

$$y \in \{0,1\}^T \tag{3.8}$$

The above formulation implies that demand is satisfiable through manufacturing (x^m) and/or remanufacturing (x^r), whose sum is referred to as "service-

ables". Constraints (3.3) and (3.4) determine the total holding and backlogging cost for serviceables in period t. Note that, for a given time period t, at most one of these two constraints' right hand side can be non-negative, making the other constraint redundant. Returns inventory cost is determined by constraint (3.5). Constraint (3.6) ensures a joint set up when manufacturing and/or remanufacturing takes place in a given time period t, with an appropriate choice of M_t. Finally, constraints (3.7) and (3.8) enforce nonnegativity and integrality restrictions on the variables. For a fixed y, it is easy to observe that the problem reduces to a network flow problem.

3.2.2 Robust LSR Formulation

Determining accurate values for the input parameters of the problem is often a challenging task in practice. In addition, many input parameters in realistic applications are naturally uncertain (e.g., any quantities in the future), and there is often a danger that an optimal solution may become severely infeasible or expensive even when small changes occur in input parameters (see Ben-Tal and Nemirovski [2002]). Although stochastic optimization is very effective in some cases, it makes the critical assumption that the uncertainty has a probabilistic description, which is not realistic in many applications. In such cases, robust optimization provides a suitable framework for handling parameter uncertainties by defining them as parts of predefined uncertainty sets. We refer the interested reader to Ben-Tal and Nemirovski [2002], Bertsimas et al. [2011] for a detailed discussion on general motivations for choosing a robust optimization approach for tackling input uncertainty. As noted by Bienstock and Özbay [2008], complexities in manufacturing systems varying from long production leadtimes to complex supply chains result in significant inadequacy of demand data, which often dictates the use of uncertainty sets for most effective treatment of uncertainties. Moreover, return of items for remanufacturing purposes entail further complications such as customer behavior or variation in the levels of use of the products, further motivating the case for a robust optimization framework.

A robust solution is defined as a solution that remains feasible over the entire uncertainty set. Such solution assumes that the production quantities represent "here and now" variables, corresponding to decisions taken before the uncertain parameters are revealed while the inventory levels are adjustable to the material-

ized parameters. Although such solutions provide immunity to all eventualities, considering an exhaustive number of cases may lead to solutions which may be poor for most of the reasonable scenarios. Such a solution, in order to retain feasibility, has to potentially accommodate extreme case scenarios that have a negligible chance of realisation. In order to avoid this, we employ the method introduced by Bertsimas and Sim [2004], which controls the number of scenarios in the uncertainty set using budgeted polytopes, as discussed below. Moreover, uncertainty sets of this type are known to be tractable and computationally easier to handle.

Following the approach of Attila et al. [2017], we model the uncertainty in demands and returns as budgeted polytopes. Although implementing other types of uncertainty sets (such as ellipsoids) would be relevant for the problem of interest, using budgeted polytopes is favorable since uncertainty sets of this nature are tractable. We assume the uncertainty in demands and returns are independent of each other. Assuming that demands and returns uncertainty sets are independent from one another is a favorable assumption since the dependency of returns on demands (for both the magnitude and timing of returns) is often very dependent on the product(s) and often follow a very ambiguous and diverse pattern. For each time period, $t = 1, \ldots, T$, we are provided with the nominal demands (resp. returns) $\bar{D}_t$ (resp. $\bar{R}_t$) and the maximum possible deviation from the nominal value $\hat{D}_t$ (resp. $\hat{R}_t$). In other words, the uncertain demand (resp. return) D_t (resp. R_t) in time t takes a value in the interval $[\bar{D}_t, \bar{D}_t + \hat{D}_t]$ (resp. $[\bar{R}_t, \bar{R}_t + \hat{R}_t]$). For each time period, t, we introduce the variables $z_t^D \in [0, 1]$ (resp. $z_t^R \in [0, 1]$), in order to model the proportion of deviation we have from the nominal demand (resp. return), namely $D_t = \bar{D}_t + \hat{D}_t z_t^D$ (resp. $R_t = \bar{R}_t + \hat{R}_t z_t^R$). Only positive deviations are considered as this corresponds to the worst case for demands (where the production is lower than expected leading to backlogged demands), and for returns we can show that feasibility implies that only the expected minimum number of items can be used, which allows to conclude that the case of positive deviations is equivalent to the case of negative deviations. In order to avoid over-conservative estimation of the parameters, we introduce the parameters Γ_t^D (resp. Γ_t^R) in order to constrain z_t^D (resp. z_t^R):

$$Z^D(\Gamma^D) := \{z^D \in [0, 1]^T : \sum_{i=1}^{t} z_i^D \leq \Gamma_t^D, \ \forall t = 1, \ldots, T\} \tag{3.9}$$

$$Z^R(\Gamma^R) := \{z^R \in [0,1]^T : \sum_{i=1}^{t} z_i^R \leq \Gamma_t^R,\ \forall t = 1, \ldots, T\} \tag{3.10}$$

Then, the independent uncertainty sets for demands and returns, respectively, can be defined as follows:

$$U^D(\Gamma^D) := \{D \in \mathbb{R}_+^T : D_t = \bar{D}_t + \hat{D}_t z_t^D, z^D \in Z^D(\Gamma^D)\} \tag{3.11}$$

$$U^R(\Gamma^R) := \{R \in \mathbb{R}_+^T : R_t = \bar{R}_t + \hat{R}_t z_t^R, z^R \in Z^R(\Gamma^R)\} \tag{3.12}$$

We will sometimes refer to the uncertainty set given in (3.11) and (3.12) as $U^{D+}(\Gamma^D)$ and $U^{R+}(\Gamma^R)$ to indicate we are considering positive deviations from the nominal value.

Uncertainty sets (3.11) and (3.12) limit the cumulative deviation from nominal values of demands and returns. Here, considering cumulative deviations is particularly interesting for these uncertainty sets, since this allows the consecutive time periods to be considered at once, for varying number of time periods. Although these uncertainty sets are known to be tractable, they might not be able to capture specific trends and seasonality of demands and returns.

For the number of returns, we can use the implicit balance constraints

$$s_{i-1}^r + R_i = d_i + x_i^r + s_i^r,\ \forall i = 1, \ldots, T \tag{3.13}$$

to derive the following (since $s_0^r = 0$)

$$s_t^r = \sum_{i=1}^{t} (R_i - d_i - x_i^r),\ \forall t = 1, \ldots, T. \tag{3.14}$$

Non-negativity of s_t^r implies

$$\sum_{i=1}^{t} (R_i - d_i - x_i^r) \geq 0,\ \forall t = 1, \ldots, T. \tag{3.15}$$

Under the robust setting, if the returns R belong to a given uncertainty set U then, for all $t = 1, \ldots, T$, (3.15) becomes

$$\sum_{i=1}^{t} (d_i + x_i^r) \leq \min\{\sum_{i=1}^{t} R_i | R \in U\}. \tag{3.16}$$

The following proposition establishes that there is no loss of generality in considering positive deviations. Consider the uncertainty set with negative deviations:

$$U^{R-}(\Gamma^R) := \{R \in \mathbb{R}_+^T : R_t = \bar{R}_t - \hat{R}_t z_t^R, z^R \in Z^R(\Gamma^R)\} \tag{3.17}$$

Proposition 2. *Let* $A_t = \max\{\sum_{i=1}^{t} \hat{R}_i z_i^R | z^R \in Z^R(\Gamma^R)\}$ *and let* $\bar{S}_1 = \bar{R}_1 - A_1$, $\bar{S}_t = \sum_{i=1}^{t} \bar{R}_i - A_t - \sum_{i=1}^{t-1} \bar{S}_i$ *for* $t = 1, \ldots, T$ *and* $\hat{S}_t = \hat{R}_t$ *for* $t = 1, \ldots, T$. *Then, the following equalities hold:*

a) $\min_{R \in U^{R-}(\Gamma^R)} \sum_{i=1}^{t} R_i = \min_{S \in U^{R+}(\Gamma^R)} \sum_{i=1}^{t} S_i$.

b) $\max_{R \in U^{R-}(\Gamma^R)} \sum_{i=1}^{t} R_i = \max_{S \in U^{R+}(\Gamma^R)} \sum_{i=1}^{t} S_i$.

Proof. Since the proof of (a) and (b) are similar we prove only (a).

$$\min_{R \in U^{R-}(\Gamma^R)} \sum_{i=1}^{t} R_i = \min_{z^R \in Z^R(\Gamma^R)} \sum_{i=1}^{t} (\bar{R}_i - \hat{R}_i z_i^R)$$

$$= \sum_{i=1}^{t} \bar{R}_i + \min_{z^R \in Z^R(\Gamma^R)} (-\hat{R}_i z_i^R) = \sum_{i=1}^{t} \bar{R}_i - \max_{z^R \in Z^R(\Gamma^R)} \hat{R}_i z_i^R = \sum_{i=1}^{t} \bar{R}_i - A_t$$

$$= \sum_{i=1}^{t} \bar{S}_i = \min_{z^R \in Z^R(\Gamma^R)} \sum_{i=1}^{t} (\bar{S}_i + \hat{S}_i z_i^R) = \min_{S \in U^{S+}(\Gamma^R)} \sum_{i=1}^{t} S_i$$

□

Example 3.2.1. *Consider* $T = 3$ *and* $\Gamma_i = 1, \ \forall i = 1, ..., T$. *Let* $\bar{R} = (5, 5, 5)$ $\hat{R} = (1, 2, 3)$. *Here we provide a numerical example for case a) from Proposition 2, where:*

$$A_1 = \max_{z^R \in [0,1]^1} \{z_1^R | z_1^R \leq 1\} = 1$$

$$A_2 = \max_{z^R \in [0,1]^2} \{z_1^R + 2z_2^R | z_1^R + z_2^R \leq 1\} = 2$$

$$A_3 = \max_{z^R \in [0,1]^3} \{z_1^R + 2z_2^R + 3z_3^R | z_1^R + z_2^R + z_3^R \leq 1\} = 3$$

In order to represent the minimum of a returns uncertainty set with negative deviations as one with positive deviations, we need to compute S. *From Proposition 2, we have the following:*

$$\bar{S}_1 = \bar{R}_1 - A_1 = 5 - 1 = 4$$

$$\bar{S}_2 = \sum_{i=1}^{2} \bar{R}_i - A_2 - \bar{S}_1 = (5+5) - 2 - 4 = 4$$

$$\bar{S}_3 = \sum_{i=1}^{3} \bar{R}_i - A_3 - \sum_{i=1}^{2} \bar{S}_i = (5+5+5) - 3 - (4+4) = 4$$

In this case, the minimum number of cumulative returns for the uncertainty set $S \in U^{R+}(\Gamma^R) \sum_{i=1}^{3} (\bar{S}_i + \hat{S}_i z_i^R)$ *is* $\sum_{i=1}^{3} \bar{S}_i = 12$*, in which case* $z^R = (0,0,0)$*. This is equivalent to the number of minimum cumulative returns derived from the uncertainty set where* $R \in U^{R-}(\Gamma^R)$*, which can be calculated as* $\sum_{i=1}^{3} \bar{R}_i - A_3 = (5+5+5) - 3 = 12$.

Hence we focus only on positive deviations, that is, when $U = U^{R+}(\Gamma^R)$. In this case inequalities (3.16) can be written as follows

$$\sum_{i=1}^{t} (\bar{R}_i - d_i - x_i^r) \geq 0, \ \forall i = 1, \ldots, T. \tag{3.18}$$

A favourable aspect of considering only positive deviations is that the decision maker has one absolute optimal production plan (since production variables are scenario independent). This is also favourable from a computational point of view, since the number of variables considered when we only have positive deviations is significantly less compared to the case where both negative and positive deviations are considered.

An alternative characterisation for these uncertainty sets can be provided in terms of the convex hull of its extreme points as follows, where J_D (resp. J_R) indicates the number of extreme points for demands (resp. returns):

$$Z^D(\Gamma^D) := Conv(\{z^{D^1}, z^{D^2}, \ldots, z^{D^{J_D}}\})$$

$$Z^R(\Gamma^R) := Conv(\{z^{R^1}, z^{R^2}, \ldots, z^{R^{J_R}}\})$$

and

$$U^D(\Gamma^D) := Conv(\{D^1, D^2, \ldots, D^{J_D}\})$$

$$U^R(\Gamma^R) := Conv(\{R^1, R^2, \ldots, R^{J_R}\})$$

The t^{th} component of vector D^j (resp. R^j) is given by $D_t^j = \bar{D}_t + \hat{D}_t z_t^{D^j}$ and $R_t^j = \bar{R}_t + \hat{R}_t z_t^{R^j}$, for all $j = 1, \ldots, J_D$ (resp. $j = 1, \ldots, J_R$).

Under this setting, the RLSR problem seeks a solution that is feasible for any demand $D \in U^D(\Gamma^D)$ and return $R \in U^R(\Gamma^R)$. As constraints (3.3)–(3.5) are affected by parameter uncertainty in LSR-D, we rewrite these constraints, so that a solution would be feasible for all $\tilde{D} \in U^D(\Gamma^D), \tilde{R} \in U^R(\Gamma^R)$ resulting in the following robust formulation for the RLSR problem:

$$\min \quad \theta^{D,R}(\mathbf{x}) + \pi \tag{LSR-R}$$

s.t.

$$\pi \geq \sum_{t=1}^{T}(H_t^{sj} + H_t^{ri}) \qquad \begin{array}{l}\forall j = 1,\ldots,J_D,\\ \forall i = 1,\ldots,J_R\end{array} \tag{3.19}$$

$$H_t^{sj} \geq h^s \sum_{i=1}^{t}(x_i^m + x_i^r - D_i^j) \qquad \begin{array}{l}\forall t = 1,\ldots,T,\\ \forall j = 1,\ldots,J_D\end{array} \tag{3.20}$$

$$H_t^{sj} \geq -b \sum_{i=1}^{t}(x_i^m + x_i^r - D_i^j) \qquad \begin{array}{l}\forall t = 1,\ldots,T,\\ \forall j = 1,\ldots,J_D\end{array} \tag{3.21}$$

$$H_t^{rj} \geq h^r \sum_{i=1}^{t}(R_i^j - x_i^r - d_i) \qquad \begin{array}{l}\forall t = 1,\ldots,T,\\ \forall j = 1,\ldots,J_R\end{array} \tag{3.22}$$

$$\sum_{i=1}^{t}(\bar{R}_i - d_i - x_i^r) \geq 0 \qquad \forall t = 1,\ldots,T \tag{3.23}$$

$$(3.6) - (3.8)$$

Here, the variables H_t^{sj} (resp. H_t^{rj}) correspond to the cost of serviceable inventory or backlogging (resp. return inventory) incurred at time t for the demand D^j (resp. return R^j). The variable π stores the highest cost of inventory and backlogging incurred by any demand or return. Constraint (3.23) is elaborated on earlier as it is equivalent to (3.18), and enforces feasibility of a production plan for all possible realisations of returns, ensuring we do not remanufacture or dispose more than the nominal return levels. Constraints (3.20) - (3.22)are defined for all demand and return vectors corresponding to extreme points of the budgeted uncertainty sets $U^D(\Gamma^D)$ and $U^R(\Gamma^R)$. Hence, we have exponentially many constraints in our formulation. We handle this using a decomposition approach, in a similar fashion to the approach of Bienstock and Özbay [2008], to obtain robust solutions to RLSR, which we will further discuss in the next chapter.

An important observation in (LSR-R) is that the worst costs for returns can be generated in advance of solving the robust problem. In order to find the worst-

case scenario for returns, we introduce the variable H_t^{rw}, which denotes the worst total cost associated with returns inventory. The justification for this replacement follows from the following proposition.

Proposition 3. *Let, for all* $t = 1, \ldots, T$,

$$H_t^{rw} = \max_{j=1,\ldots,J_R} h^r \sum_{i=1}^{t} (R_i^j - x_i^r - d_i) \tag{3.24}$$

and suppose z^{Rw} *is the optimal solution to*

$$\max_{z^R \in Z^R(\Gamma^R)} \sum_{t=1}^{T} (T - t + 1) \hat{R}_t z_t^R. \tag{3.25}$$

Then, $\sum_{t=1}^{T} H_t^{rw} = \sum_{t=1}^{T} h^r \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^{Rw} - d_i - x_i^r)$

Proof.

$$\sum_{t=1}^{T} H_t^{rw} = \sum_{t=1}^{T} \max_{j=1,\ldots,J_R} h^r \sum_{i=1}^{t} (R_i^j - x_i^r - d_i)$$

$$= h^r \sum_{t=1}^{T} \max_{z^R \in Z^R(\Gamma^R)} \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^R - x_i^r - d_i)$$

$$= h^r \sum_{t=1}^{T} \sum_{i=1}^{t} (\bar{R}_i - x_i^r - d_i) + h^r \max_{z^R \in Z^R(\Gamma^R)} \sum_{t=1}^{T} \sum_{i=1}^{t} \hat{R}_i z_i^R$$

Observing that

$$\max_{z^R \in Z^R(\Gamma^R)} \sum_{t=1}^{T} \sum_{i=1}^{t} \hat{R}_i z_i^R = \max_{z^R \in Z^R(\Gamma^R)} \sum_{t=1}^{T} (T - t + 1) \hat{R}_t z_t^R$$

which obtains the maximum when $z^R = z^{Rw}$ (from (3.25)) we obtain

$$h^r \sum_{t=1}^{T} \sum_{i=1}^{t} (\bar{R}_i - x_i^r - d_i) + h^r \max_{z^R \in Z^R(\Gamma^R)} \sum_{t=1}^{T} \sum_{i=1}^{t} \hat{R}_i z_i^R$$

$$= h^r \sum_{t=1}^{T} \sum_{i=1}^{t} (\bar{R}_i - x_i^r - d_i) + h^r \sum_{t=1}^{T} \sum_{i=1}^{t} \hat{R}_i z_i^{Rw}$$

$$= h^r \sum_{t=1}^{T} \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^{Rw} - d_i - x_i^r)$$

□

3.3 Concluding remarks

In this chapter, the robust lot sizing problem with remanufacturing with joint setups was formally introduced. In addition, the uncertainty sets, which are modelled as budgeted polytopes are given in detail. For returns uncertainty sets, it is shown that the worst case costs can be found in advance of solving the robust problem. In the next chapter, we provide an implementation of the min-max decomposition framework and provide reformulations for the master problem.

Chapter 4

Decomposition and Reformulations

This chapter provides further insights as to how the robust lot sizing problem with remanufacturing is solved using the min-max decomposition under the robust optimization framework (Bienstock and Özbay [2008]) based on the uncertainty sets and robust properties presented in Chapter 3. In addition to this, two additional reformulations for the master problem in the min-max framework are introduced, where the computational performance of each formulation is examined in detail.

4.1 Introduction

In this chapter, we primarily aim to extend the decomposition approach presented in the previous chapter, for a two-stage robust lot-sizing problem with remanufacturing and backlogging. We propose two extended reformulations for the computationally challenging master problem of the decomposition, where we propose first aggregating the separate production variables but extend them in the classical facility location formulation fashion, and then we propose an approximate extended reformulation. We discuss some key aspects of these reformulations (also in comparison to the basic formulation), and then present an extensive computational analysis in order to identify specific strengths and weaknesses of different formulations, as well as to support our theoretical claims. More specifically, we are extending the ideas of Van Vyve and Wolsey [2006] in order to solve the robust version of the lot sizing problem with returns and remanufacturing option. In

order to effectively handle the size of the master model which increases with the number of scenarios, we make the key observation on the formulation that the flow conservation only on the last scenario's demand is sufficient, which significantly reduces the size of the formulations. To the best of our knowledge, this is the first use of an extended formulation technique for multiple scenarios under a robust setting. In addition, we provide a thorough study of the structure associated with the returns and, in particular, we observe the equivalence between the uncertain sets with positive and negative deviations from the nominal values for the case of returns. Finally, from a computational perspective, we present comprehensive numerical results on the tightness of the extended formulations by providing a threshold value for the parameter P for various input classes, when the lower bound improvement tails off.

In the next section, we present first a deterministic formulation of the problem with a detailed explanation of the practical setting, and then propose the robust version of this. Then, in Section 4.3, we propose two extended reformulations of the robust problem, and also remark the theoretical strength of using these in comparison to the basic formulation. We then present the results of thorough computational experiments in Section 4.4, which evaluates the proposed reformulations from a number of perspectives, including computational times, lower bounds and sensitivity to input parameters. Finally, we conclude with some key remarks and potential future research directions in Section 4.5.

4.2 Min-Max Decomposition Approach

Our min-max approach involves iteratively solving a restricted version of LSR-R, which is referred to as the "Decision Maker's Problem" (DMP), where only a subset of extreme points, denoted by $\tilde{U}^D \subseteq U^D(\Gamma^D)$ and $\tilde{U}^R \subseteq U^R(\Gamma^R)$, are considered. A new demand (resp. return) point is added to the subset $\tilde{U}^D$ (resp. $\tilde{U}^R$) at every iteration by solving a certain maximization problem that we refer to as the "Adversarial Problem (AP)". Given an optimal production plan, AP seeks the demand $D \in U^D(\Gamma^D)$ and return $R \in U^R(\Gamma^R)$ vectors with the highest inventory and backlogging cost for this specific production plan. These demand and return vectors are then used to update $\tilde{U}^D$ and $\tilde{U}^R$ respectively, see also Attila et al. [2017], Bienstock and Özbay [2008], Zeng and Zhao [2013]. A schematic diagram, which details the decomposition approach is provided in Figure 4.1. Let $\tilde{J}_D$ and

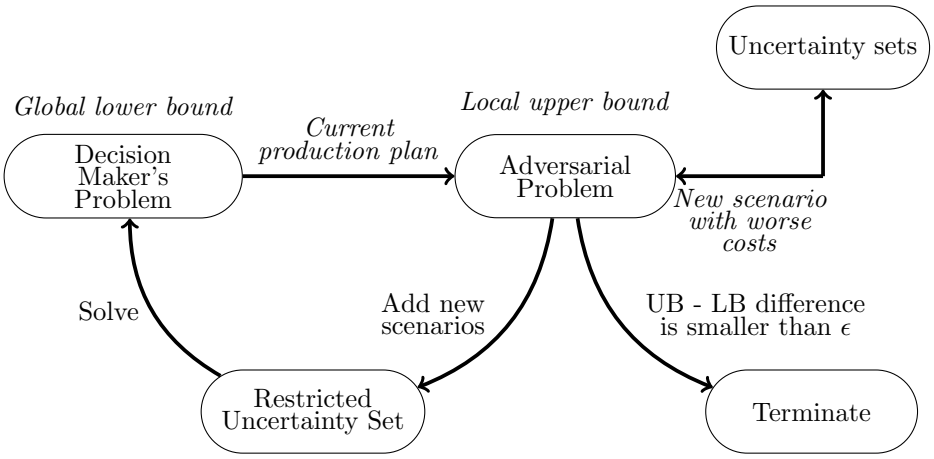

Figure 4.1: Decomposition approach.

$\tilde{J}_R$ be the number of extreme points in the sets $\tilde{U}^D$ and $\tilde{U}^R$, respectively. Then, DMP can be stated as follows:

$$\min \theta^{D,R}(\mathbf{x}) + \pi \tag{DMP}$$

$$\text{s.t. } \pi \geq \sum_{t=1}^{T} (H_t^{sj} + H_t^{rw}) \qquad \forall j = 1, \ldots, \tilde{J}_D \tag{4.1}$$

$$H_t^{sj} \geq h^s \sum_{i=1}^{t} (x_i^m + x_i^r - (\bar{D}_i + \hat{D}_i z_i^{Dj})) \qquad \begin{array}{l} \forall t = 1, \ldots, T, \\ \forall j = 1, \ldots, \tilde{J}_D \end{array} \tag{4.2}$$

$$H_t^{sj} \geq -b \sum_{i=1}^{t} (x_i^m + x_i^r - (\bar{D}_i + \hat{D}_i z_i^{Dj})) \qquad \begin{array}{l} \forall t = 1, \ldots, T, \\ \forall j = 1, \ldots, \tilde{J}_D \end{array} \tag{4.3}$$

$$H_t^{rw} = h^r \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^{Rw} - d_i - x_i^r) \qquad \forall t = 1, \ldots, T \tag{4.4}$$

$$(3.6) - (3.8), (3.23)$$

Here, the main difference between inventory and backlogging cost constraints in (LSR-R) and (DMP) formulations is that constraints (4.2) – (4.4) are written for a subset of demand and return points, rather than the complete uncertainty set. Also note that the entire constraint set (3.22) was replaced by the single constraint (4.4).

In the following discussion, we will omit the subscript D from the parameter J_D,

as we are now only enumerating the extreme points of the uncertain demand set in our formulation. Also, we let $D_i^j := \bar{D}_i + \hat{D}_i z_i^{D_j}$, for all $i = 1, \ldots, T, j = 1, \ldots, J$.

Next, we define the Adversarial Problem (AP). Here, the aim is to find a specific demand vector that implies a higher total inventory and backlogging cost for a given production plan. As such a maximum is given by (3.25) for returns, AP only seeks a new demand vector. Under this setting, the optimal production plan $u^* = (x^{m*}, x^{r*}, d^*, y^*)$ of DMP is the input to AP. For notational simplicity, let $X_t^* = \sum_{i=1}^{t}(x_i^{m*} + x_i^{r*})$. Then, AP can be defined as:

$$\max \pi \tag{AP}$$

$$\text{s.t. } \pi \leq \sum_{t=1}^{T} H_t^s \tag{4.5}$$

$$H_t^s = \max\Bigg\{ h^s(X_t^* - \sum_{i=1}^{t}(\bar{D}_i + \hat{D}_i z_i^D)), \\ - b(X_t^* - \sum_{i=1}^{t}(\bar{D}_i + \hat{D}_i z_i^D)) \Bigg\} \qquad \forall t = 1, \ldots, T \tag{4.6}$$

$$\sum_{i=1}^{t} z_i^D \leq \Gamma_t^D \qquad \forall t = 1, \ldots, T \tag{4.7}$$

$$0 \leq z_t^D \leq 1 \qquad \forall t = 1, \ldots, T \tag{4.8}$$

The optimal π value indicates the worst total inventory and backlogging costs in the uncertainty sets (3.11) and (3.17), as enforced by constraints (4.7) and (4.8), as well as (3.25). Note that the true total worst cost can be computed as $\pi + \sum_{t=1}^{T}(H_t^s + H_t^{rw})$. Since H_t^{rw} is a constant in (AP), we do not include this term in constraint (4.5). To linearize constraint (4.6), a new binary variable $s_t \in \{0, 1\},\ \forall t = 1, \ldots, T$ is introduced, which is 1 if H_t^s represents the inventory cost, and 0 in case of backlogging. Then, the following constraints are added to AP, where $D_t^{max} = \sum_{i=1}^{t}(\bar{D}_t + \hat{D}_t)$ and $D_t^{min} = \sum_{i=1}^{t} \bar{D}_t$:

$$H_t^s \leq h^s(X_t^* - \sum_{i=1}^{t}(\bar{D}_i + \hat{D}_i z_i^D)) + M_{1t}(1 - s_t) \qquad \forall t = 1, \ldots, T \tag{4.9}$$

$$H_t^s \leq -b(X_t^* - \sum_{i=1}^{t}(\bar{D}_i + \hat{D}_i z_i^D)) + M_{2t} s_t \qquad \forall t = 1, \ldots, T \qquad (4.10)$$

$$X_t^* - \sum_{i=1}^{t}(\bar{D}_i + \hat{D}_i z_i^D) \leq s_t (X_t^* - D_t^{min}) \qquad \forall t = 1, \ldots, T \qquad (4.11)$$

$$-X_t^* + \sum_{i=1}^{t}(\bar{D}_i + \hat{D}_i z_i^D) \leq (s_t - 1)(X_t^* - D_t^{max}) \qquad \forall t = 1, \ldots, T \qquad (4.12)$$

Constraints (4.11) and (4.12) ensure the correct setting of the s_t variables, and then either constraint (4.9) or (4.10) is dominated, incurring either serviceables holding or backlogging cost, respectively. We note that M_{1t} and M_{2t} can be defined as follows:

$$M_{1t} = -b(X_t^* - D_t^{max}) - h^s(X_t^* - D_t^{max}) \qquad \forall t = 1, \ldots, T \qquad (4.13)$$

$$M_{2t} = h^s(X_t^* - D_t^{min}) + b(X_t^* - D_t^{min}) \qquad \forall t = 1, \ldots, T \qquad (4.14)$$

Finally, we remark that convergence is ensured through constraint (4.1) in (DMP), which ensures that a production plan with higher total serviceables and returns inventory cost is obtained in each iteration. In such setting, the optimal value for the objective function for (DMP) determines the global lower bound (GLB). On the other hand, the optimal objective value for (AP) provides a local upper bound in each iteration. In order to find the global upper bound on a given iteration $\tilde{J}_D$, we determine $GUB = \min_{j \in \{1,\ldots,\tilde{J}_D\}} \{\pi^{*j} + \theta^{*j,D,R}(u^*)\}$, where π^{*j} indicates the optimal value for π in (AP) (for iteration j) and $\theta^{*j,D,R}(u^*)$ is the optimal production and disposal costs of (AP) solved in iteration j. We define $\epsilon = \frac{GUB - GLB}{GLB}$ to represent the magnitude of convergence.

4.3 Extended Reformulations

Although the min-max approach is an effective method for obtaining robust optimal solutions, its computational efficiency is heavily dependent on the DMP, as initially observed in the preliminary test of Attila et al. [2017] and also further discussed in Section 4.4. Therefore, in this section, we present two extended reformulations to DMP: "Aggregated Extended Formulation" (DMP-EFAG) and "Approximate Extended Formulation" (DMP-EFAP). We provide a detailed ex-

planation on the structure of both formulations, while discussing their strengths and limitations. We will empirically support our claims in the Section 4.4 and provide a detailed account of our computational experience.

4.3.1 Extended Aggregated Reformulation

We consider a facility location reformulation for DMP, which was originally proposed by Krarup and Bilde [1977]. For this purpose, we introduce the following set of decision variables:

$$\mathbf{x}^{EF} := \{\tilde{x} \in \mathbb{R}_{+}^{(T+1)\times(T+1)} : \sum_{t=1}^{T+1} \tilde{x}_{it} = x_i^m + x_i^r, \quad \forall i = 1, \ldots, T\}, \tag{4.15}$$

where the new variables, $\tilde{x}_{it}$, indicate the total amount of items that have been manufactured and remanufactured in time period i, in order to satisfy the demand in period t. Throughout the paper, we refer to this quantity as the "aggregated production" quantity. This results in $(T+1)$ new variables for each time period t, where the aggregated production in the $(T+1)^{th}$ period indicates the amount manufactured and remanufactured after the planning period and backlogged to satisfy the demand in a period inside the planning horizon. More specifically, we only need variables $\tilde{x}_{(T+1)i}$, for all i, in order to account for such backlogging. Note that we use the same objective function as in DMP, along with the original production variables x_t^m and x_t^r.

In order to keep the formulation size reasonable and effective, we consider to apply our extended variables only to one demand scenario, namely, the one introduced in the most recent iteration J. For any production plan, the idea is to create an aggregated production plan corresponding to the J^{th} scenario, while tightening the constraint (3.6) by using the aggregated decision variables. We also define separate $\tilde{H}_t^s$ and $\tilde{B}_t$ variables in order to account for the holding and backlogging costs of serviceables in the J^{th} scenario. For all other iterations $j = 1...J-1$, we preserve the structure from DMP, with serviceables inventory cost defined through the original variables. Next, we state the extended reformulation formally.

$$\min\ \theta^{D,R}(\mathbf{x}) + \pi \qquad \text{(DMP-EFAG)}$$

$$\text{s.t. } \pi \geq \sum_{t=1}^{T}(H_t^{sj} + H_t^{rw}) \qquad \forall j = 1,\ldots,J \qquad (4.16)$$

$$H_t^{sj} \geq h^s \sum_{i=1}^{t}(x_i^m + x_i^r - D_i^j) \qquad \begin{matrix}\forall t = 1,\ldots,T \\ \forall j = 1,\ldots,J-1\end{matrix} \qquad (4.17)$$

$$H_t^{sj} \geq -b \sum_{i=1}^{t}(x_i^m + x_i^r - D_i^j) \qquad \begin{matrix}\forall t = 1,\ldots,T \\ \forall j = 1,\ldots,J-1\end{matrix} \qquad (4.18)$$

$$\tilde{H}_t^s = h^s \sum_{i=1}^{t}\sum_{k=t+1}^{T+1} \tilde{x}_{ik} \qquad \forall t = 1,\ldots,T \qquad (4.19)$$

$$\tilde{B}_t = b \sum_{i=1}^{t}\sum_{k=t+1}^{T+1} \tilde{x}_{ki} \qquad \forall t = 1,\ldots,T \qquad (4.20)$$

$$H_t^{sJ} \geq \tilde{H}_t^s + \tilde{B}_t \qquad \forall t = 1,\ldots,T \qquad (4.21)$$

$$\sum_{i=1}^{T+1} \tilde{x}_{it} = D_t^J \qquad \forall t = 1,\ldots,T \qquad (4.22)$$

$$\sum_{i=1}^{T+1} \tilde{x}_{ti} = x_t^m + x_t^r \qquad \forall t = 1,\ldots,T \qquad (4.23)$$

$$\tilde{x}_{tk} \leq D_k^J y_t \qquad \begin{matrix}\forall t = 1,\ldots,T, \\ \forall k = 1,\ldots,T\end{matrix} \qquad (4.24)$$

$$H_t^{sj}, x_t^m, x_t^r, d_t \geq 0 \qquad \begin{matrix}\forall t = 1,\ldots,T \\ \forall j = 1,\ldots,J\end{matrix} \qquad (4.25)$$

$$\tilde{x}_{it} \geq 0 \qquad \forall i,t = 1,\ldots,T{+}1 \qquad (4.26)$$

$$y \in \{0,1\} \qquad (4.27)$$

$$(3.23), (4.4)$$

In the formulation above, constraints (4.17) and (4.18) indicate the serviceables holding and backlogging costs for scenarios where $j \neq J$. For iteration J, we define constraints (4.19) and (4.20), which indicate the total serviceables holding and backlogging cost for period t, respectively, and these costs are then linked to the variable H_t^{sJ} through constraint (4.21). We ensure that the demand in a given period t is satisfied through the sum of items manufactured and remanufactured (including backlogs from beyond the planning horizon) in constraint (4.22).

$\tilde{x}_{it}$	Number of items produced on i to satisfy the demand on t.
$\tilde{H}_t^s$	Serviceables holding cost incurred in t through $\tilde{x}$.
$\tilde{B}_t$	Backlogging cost incurred in t through $\tilde{x}$.
$\tilde{H}_t^{sJ}$	Total inventory and backlogging cost in t for the last scenario, J.

Table 4.1: Decision variables for (DMP-EFAG).

Constraint (4.23) is used to link the original manufacturing and remanufacturing variables with the aggregated production variable $\tilde{x}_{it}$, and finally setup periods are determined through constraint (4.24). The observation we make here is that any production plan is a feasible production plan, because we allow backlogging to the final period. Therefore, which scenario's demand is assigned to the $\tilde{x}$-variables is insignificant, as this can be realised as a different production plan for another scenario and consequently, its corresponding cost calculated. Table 4.1 presents the list of variables that are introduced in (DMP-EFAG).

We refer to the polytope corresponding to the LP relaxation of DMP (resp. DMP-EFAG) as $\mathcal{P}_J^{DMP}$ (resp. $\mathcal{P}_J^{DMP-EFAG}$), where the subset of extreme points of the uncertainty, indexed by set J, have been considered in the formulation of DMP (resp. DMP-EFAG). We slightly abuse the notation here by referring both to the index set and the index of the last scenario in the index set by J, but we could distinguish them by context easily. We let H^s to denote the vector $(H_1^{s1}, \ldots, H_T^{sJ})$. For a polytope $P := \{(u, x) \in \mathcal{U} \times \mathcal{X}\}$, where $\mathcal{U}$ and $\mathcal{X}$ are vector spaces, we define the projection of polytope P onto the x-space (or onto $\mathcal{X}$) as

$$proj_x(P) := \{x \in \mathcal{X} : \exists u \in \mathcal{U} : (u, x) \in P\}.$$

Proposition 4. *For any index set J, $proj_{\mathbf{x},d,y,H^s,\pi}(\mathcal{P}_J^{DMP-EFAG}) \subset \mathcal{P}_J^{DMP}$*

Proof. It is easy to show that for any feasible solution to $\mathcal{P}^{DMP-EFAG}$, projecting out the $\mathbf{x}^{EF}$ variables results in a feasible solution to $\mathcal{P}^{DMP}$. The first thing to note is that the only difference in the two formulations is with respect to scenario J. Thus, we need to show that any solution $(\mathbf{x}, d, y, H^s, \pi) \in proj_{\mathbf{x},d,y,H^s,\pi}(\mathcal{P}_J^{DMP-EFAG})$ satisfies constraints (4.2) and (4.3), implying $(\mathbf{x}, d, y, H^s, \pi) \in \mathcal{P}_J^{DMP}$. First, note that, for a specific t and in an extreme point solution, either $\tilde{H}_t^s$ or $\tilde{B}_t$ in constraints (4.19) and (4.20) will be zero (as this could be perceived as a flow along a negative cost cycle and hence cancelled by sending flow in the opposite direction).

Hence, for a given t, let $\tilde{B}_t = 0$. Then,

$$\begin{aligned}
H_t^{sJ} &\geq \tilde{H}_t^s + \tilde{B}_t \\
&= h^s \sum_{i=1}^{t} \sum_{k=t+1}^{T} \tilde{x}_{ik} = h^s \sum_{i=1}^{t} (\sum_{k=1}^{T} \tilde{x}_{ik} - \sum_{k=1}^{t} \tilde{x}_{ik}) \\
&= h^s \sum_{i=1}^{t} \left((x_i^m + x_i^r) - \sum_{k=1}^{t} \tilde{x}_{ik} \right) \quad \text{(due to constraint (4.23))} \\
&= h^s \left(\sum_{i=1}^{t} (x_i^m + x_i^r) - \sum_{k=1}^{t} \sum_{i=1}^{t} \tilde{x}_{ik} \right) \quad \text{(rearranged terms)} \\
&\geq h^s \left(\sum_{i=1}^{t} (x_i^m + x_i^r) - \sum_{k=1}^{t} D_k^J \right) \quad \text{(due to constraint (4.22))} \\
&= h^s \sum_{i=1}^{t} \left((x_t^m + x_t^r) - D_i^J \right)
\end{aligned}$$

The argument is analogous in the case when $\tilde{H}_t^s = 0$. Next, consider constraint (4.24). For a given t, summing up the constraint over all k, we obtain

$$x_t^m + x_t^r = \sum_{k=1}^{T+1} \tilde{x}_{tk} \leq \sum_{k=1}^{T} D_k^J y_t \leq M_t y_t \tag{4.28}$$

The first equality follows from constraint (4.23) and the last inequality follows from the fact that M_t needs to be chosen so the formulation is feasible for any scenario. This concludes the proof for $proj_{\mathbf{x},d,y,H^s,\pi}(\mathcal{P}_J^{DMP-EFAG}) \subseteq \mathcal{P}_J^{DMP}$.

In order to show that it is a proper subset, let us consider a specific feasible solution to DMP with $H_t^{sJ} = 0,\ d_t = \bar{R}_t$, i.e., we have no inventory and backlogging of serviceables at any time period for scenario J, and all nominal returns are immediately disposed. For the sake of simplicity, assume that all nominal demands are strictly positive. Then, $y_t > 0$ holds for all $t = 1 \dots T$ to maintain feasibility. Then, the following condition will hold for any solution to $\mathbf{x}$ in DMP:

$$D_t^J \leq x_t^m + x_t^r \leq \sum_{i=t}^{T} (\bar{D}_i + \hat{D}_i) y_t \quad \forall t = 1, \dots, T \tag{4.29}$$

A feasible solution satisfying this condition is when $D_t^J = x_t^m + x_t^r = \sum_{i=t}^{T} (\bar{D}_i +$

$\hat{D}_i)y_t$, which implies $y_t = D_t^J / \sum_{i=t}^{T}(\bar{D}_i + \hat{D}_i)$. This produces the feasible solution $\left(\mathrm{x}^{\mathrm{m}} = (D_1^J, \ldots, D_T^J), \mathrm{x}^{\mathrm{r}} = (0, \ldots, 0), H^{sJ} = (0, \ldots, 0), y = (D_1^J / \sum_{i=1}^{T}(\bar{D}_i + \hat{D}_i), \ldots, D_T^J/(\bar{D}_T + \hat{D}_T))\right)$ to DMP.

Solutions of this type cannot be obtained by projection of any feasible solution for DMP-EFAG because constraints (4.22) and (4.24) cannot be satisfied simultaneously. From constraint (4.22), we have

$$\sum_{i=1}^{T+1} \tilde{x}_{it} = \tilde{x}_{tt} = D_t^J \tag{4.30}$$

The first equality is due to the fact that we have no backlogging or serviceable inventory for scenario J. From constraint (4.24), we have

$$\tilde{x}_{tt} \leq D_t^J y_t = D_t^J \frac{D_t^J}{\sum_{i=t}^{T}(\bar{D}_i + \hat{D}_i)} < D_t^J \tag{4.31}$$

□

4.3.2 Approximate Extended Reformulation

Even though we are able to obtain tighter lower bounds using DMP-EFAG, the excessive number of variables can deteriorate computational performance. For this reason, preserving a relatively tight lower bound while introducing a smaller number of variables is crucial for improvement in computational times. For DMP-EFAG, one way of achieving this is to eliminate aggregated production variables that are likely to take a value of zero in the optimal solution. This is mostly the case for $\tilde{x}_{it}$ when $|i - t|$ is too high. Thus, we implement a partial formulation for DMP-EFAG, where a predefined parameter P is used to define the intervals for which $\tilde{x}_{it}$ is introduced in a similar fashion to Van Vyve and Wolsey [2006]. Ideally, we would like to choose P such that it represents an estimation for the number of periods between consecutive setup periods.

In our study, we exploit the iterative procedure involved in the min-max approach, where we derive P by tuning its value according to the structure of the optimal solutions from previous iterations. More specifically, for iteration j, let

$\mathcal{T} = \{t^1, t^2, ..., t^s\}$ be an ordered set of increasing indices with active setup periods in the solution of iteration $j-1$, i.e., $y_t = 1,\ \forall t \in \mathcal{T}$ and $t^i < t^{i+1},\ \forall i = 1 \dots s-1$. Then, we set $P = \max\{t^2 - t^1, t^3 - t^2, ..., t^s - t^{s-1}\}$ for the current iteration j. Note that for the first iteration, P is chosen arbitrarily as $P = 3$, motivated by the results given in Section 4.4.

Once P is determined, $\tilde{x}_{ti}$ is introduced for a subset of time periods, where $\forall t \in \{1, \dots, T+1\}$, and $i \in S_t^P$ such that $S_t^P = \big\{ \max\{1, t-P\}, ..., \min\{t+P, T\} \big\}$. Under these assumptions, demands in periods $i \notin S_t^P$ are not allowed to be satisfied through the aggregated production variables. Note that we will also introduce non-extended variables to allow that $D_i^j : i \notin S_t^P$ may be satisfied through production in period t, as follows:

v_t^{1s} : Number of items produced in period t to satisfy demand in any period in the interval $[t + P + 1, \dots, T]$, through keeping serviceables inventory.

v_t^{1b} : Number of items produced in period t to satisfy demand in any period in the interval $[1, \dots, t - P - 1]$, through backlogging.

v_t^{2s} : Amount of demand in period t satisfied through $v_i^{1s} : i = [1, \dots, t - P - 1]$ variables.

v_t^{2b} : Amount of demand in period t satisfied through $v_i^{1b} : i = [t + P + 1, \dots, T]$ variables.

In order to account for the inventory decisions taken through these variables, we introduce additional variables:

w_t^s : Items that are kept in serviceables inventory in period t through the use of v^{1s}, v^{2s}.

w_t^b : Items that are backlogged in period t through the use of v^{1b}, v^{2b}.

H_t^{sPJ} : The total serviceables holding and backlogging cost associated with the variables w_t^s and w_t^b.

Then, we present the formulation formally as follows, which we discuss in detail next.

$$\min\ \theta^{D,R}(\mathbf{x}) + \pi \qquad \text{(DMP-EFAP)}$$

$$\text{s.t.}\quad \pi \geq \sum_{t=1}^{T}(H_t^{sJ} + H_t^{sPJ} + H_t^{rw}) \tag{4.32}$$

$$\pi \geq \sum_{t=1}^{T}(H_t^{sj} + H_t^{rw}) \qquad \forall j = 1,\ldots,J\text{-}1 \tag{4.33}$$

$$\tilde{H}_t^s = h^s \sum_{i=a_t}^{t} \sum_{j=t+1}^{b_t} \tilde{x}_{ij} \qquad \forall t = 1,\ldots,T \tag{4.34}$$

$$\tilde{B}_t = b \sum_{i=a_t}^{t} \sum_{j=t+1}^{b_t} \tilde{x}_{ji} \qquad \forall t = 1,\ldots,T \tag{4.35}$$

$$H_t^{sPJ} = \tilde{H}_t^{sP} + \tilde{B}_t^P \qquad \forall t = 1,\ldots,T \tag{4.36}$$

$$\tilde{H}_t^{sP} = h^s(w_t^s + \sum_{i=s_t^{min}}^{s_t^{max}} v_{i+P+1}^{2s}) \qquad \forall t = 1,\ldots,T \tag{4.37}$$

$$\tilde{B}_t^P = b(w_t^b + \sum_{i=b_t^{min}}^{b_t^{max}} v_i^{2b}) \qquad \forall t = 1,\ldots,T \tag{4.38}$$

$$w_{t-1}^s + v_t^{1s} = w_t^s + v_{t+P+1}^{2s} \qquad \forall t = 1,\ldots,T\text{-}P\text{-}1 \tag{4.39}$$

$$w_{t-1}^s = w_t^s \qquad \forall t = T\text{-}P,\ldots,T \tag{4.40}$$

$$w_{t-1}^b + v_{t-P-1}^{2b} = w_t^b + v_t^{1b} \qquad \forall t = P\text{+}2,\ldots,T \tag{4.41}$$

$$D_t^J = \begin{cases} v_t^{2b} + \sum_{i=a_t}^{b_t} x_{it}, & \forall t = 1,\ldots,v_{min}^2 \\ v_t^{2s} + v_t^{2b} + \sum_{i=a_t}^{b_t} x_{it}, & \forall t = P\text{+}2,\ldots,T\text{-}P \\ v_t^{2s} + \sum_{i=a_t}^{b_t} x_{it}, & \forall t = v_{max}^2,\ldots,T \\ \sum_{i=a_t}^{b_t} x_{it}, & \text{if } P\text{+}2 > T\text{-}P, \\ & \forall t = v_{min}^2\text{+}1,\ldots,v_{max}^2\text{-}1 \end{cases} \tag{4.42}$$

$$x_t^m + x_t^r = \begin{cases} v_t^{1s} + \sum_{i=a_t}^{b_t} x_{ti} & \forall t = 1, \ldots, v_{min}^1 \\ v_t^{1b} + \sum_{i=a_t}^{b_t} x_{ti} & \forall t = v_{max}^1, \ldots, T \\ \sum_{i=a_t}^{b_t} x_{ti} & \text{if } P{+}2 > T\text{-}P\text{-}1 \\ & \forall t = v_{min}^1{+}1, \ldots, v_{max}^1\text{-}1 \\ v_t^{1s} + v_t^{1b} + \sum_{i=a_t}^{b_t} x_{ti} & \forall t = P{+}2, \ldots, T\text{-}P\text{-}1 \end{cases} \tag{4.43}$$

$$\tilde{x}_{it} \leq (\bar{D}_t + \hat{D}_t) y_i \qquad \forall i = 1, \ldots, T,\ \forall t \in S_i^P \tag{4.44}$$

$$x_t^m + x_t^r \leq M_t y_t \qquad \forall t = 1, \ldots, T \tag{4.45}$$

$$v_t^{1s} \leq \sum_{i=t+P+1}^{T} (\bar{D}_i + \hat{D}_i) y_t \qquad \forall t = 1, \ldots, T\text{-}P\text{-}1 \tag{4.46}$$

$$v_t^{1b} \leq \sum_{i=1}^{t-P-1} (\bar{D}_i + \hat{D}_i) y_t \qquad \forall t = P{+}2, \ldots, T \tag{4.47}$$

$$H_t^{rj}, x_t^m, x_t^r, x_{it}, d_t \geq 0 \tag{4.48}$$

$$y_t, \text{binary} \tag{4.49}$$

$$(3.23), (4.17) - (4.21)$$

Here, we have three different types of inventory costs. The first set arise from the original variables in DMP and hence presented again by constraints (4.17) and (4.18), through which we decide the inventory levels for demand scenarios $j = 1...J-1$, and therefore, the total serviceables inventory cost as H^{sj}.

Secondly, there is inventory cost incurred through variables $\tilde{x}_{it}$ and constraints (4.34) and (4.35), by which the inventory levels for the last scenario J are decided. These individual costs are linked to the variable H_t^{sj} by constraint (4.21).

Thirdly, we consider the inventory costs incurred through the non-extended variables, where the total inventory cost is given by H_t^{sPJ} in constraint (4.36). Constraints (4.37) and (4.38) indicate the independent serviceables inventory and backlogging costs, respectively, and we define the following measures to determine the specific variables that contribute to these costs, based on the values of t and P:

- $[\mathrm{s_t^{min}}, \mathrm{s_t^{max}}]$ where $\mathrm{s_t^{min}} = \max\{1,\ t-P\}$ and $\mathrm{s_t^{max}} = \min\{t,\ T-P-1\}$: De-

termines the interval, in which serviceables holding cost is incurred through v_t^{2s} for period t.

- $[\mathrm{b}_\mathrm{t}^{\min}, \mathrm{b}_\mathrm{t}^{\max}]$ where $\mathrm{b}_\mathrm{t}^{\min} = \max\{1,\ t-P\}$ and $\mathrm{b}_\mathrm{t}^{\max} = \min\{t,\ T-P\}$: Determines the interval, in which backlogging cost is incurred through v_t^{2b} for period t.

Here, $\mathrm{s}_\mathrm{t}^{\min}$, $\mathrm{s}_\mathrm{t}^{\max}$, $\mathrm{b}_\mathrm{t}^{\min}$, $\mathrm{b}_\mathrm{t}^{\max}$ are used in constraints (4.37) and (4.38). Constraint (4.32) is used to determine the total serviceables inventory and backlogging cost for the last scenario J, where all three types of inventory costs are summed. As the remaining demand points are handled through the constraints in DMP-EFAG, we indicate the total inventory costs for scenarios $j = 1...J-1$ through constraint (4.33).

Flow conservation of non-extended variables are achieved through constraints (4.39)-(4.41). The set of constraints in (4.42) ensure that demand is satisfied in each time period. Here, we have four different cases, depending on the specific values of t and P as aggregated production variables are only introduced for the interval $[t-P, t+P]$. Under this setting, demand can either be satisfied by both, none or only one of the approximate and aggregated production variables. In order to determine the exact intervals for each of these cases, we introduce the following:

- $v_{min}^2 = \min\{T-P,\ P+1\}$: Determines the period until which demand can be satisfied through v_t^{2b} and approximate extended production variables.

- $v_{max}^2 = \max\{T-P+1,\ P+2\}$: Determines the period from which demand can be satisfied through v_t^{2s} and approximate extended production variables.

Similarly, the flow conservation constraints given in (4.43) for production variables vary for specific combinations of t, P and T. In this case, the value of the original production variables x_t^m and x_t^r is either equal to the sum of the extended production variables, $\tilde{x}_{it}$, or slip between the sum and v_t^{1s} and/or v_t^{1b}. The sum of extended production variables is given as $\sum_{i=a_t}^{b_t} x_{ti}$, where $a_t = \max\{1,\ t-P\}$ and $b_t = \min\{t+P,\ T+1\}$. Here, a_t and b_t are used to ensure that the approximate variables at the beginning and end of the planning horizon remain in the set S_i^P. The intervals for each case is determined according to the following values:

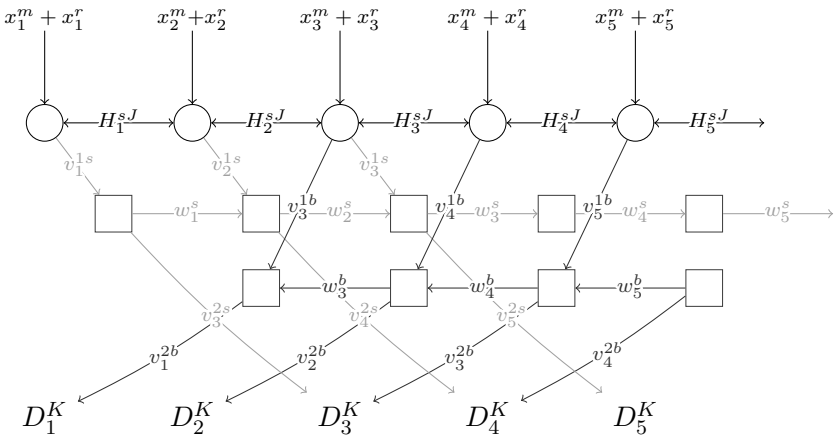

Figure 4.2: (DMP-EFAP) with $P = 1$ and $T = 5$

- $v^1_{min} = \min\{T - P - 1,\ P + 1\}$: Determines the period until which the total production (through original production variables x^m and x^r) is distributed between approximate extended production variables ($\tilde{x}_{it}$) and v_t^{1s}.

- $v^1_{max} = \max\{T - P,\ P + 2\}$: Determines the period from which the total production is distributed between approximate extended production variables and v^{1b} (until the last period T).

Figure 4.2 illustrates an example for the use of approximate variables, where $T = 5$ and $P = 1$. In contrast to DMP-EFAG, the value of the original manufacturing and remanufacturing variables are not only distributed between the aggregated production variables, but also v^{1s} and v^{1b}, where applicable. Note that Figure 4.2 only illustrates the decisions taken on the serviceables level, as decisions related to returns and remanufacturing remain unchanged.

Finally, we ensure that a joint setup cost is incurred when production takes place in constraints (4.44) - (4.47). As v_t^{1s} is only used to satisfy demand points in time periods $[t + P + 1, \ldots, T]$, we set $M_t = \sum_{i=t+P+1}^{T} (\bar{D}_i + \hat{D}_i)$ for constraint (4.46). Similarly, for constraint (4.47), as demands that are backlogged through the non-extended variables are defined as v_t^{1b} are in the interval $[1, \ldots, t - P - 1]$, we define $M_t = \sum_{i=1}^{t-P-1} (\bar{D}_i + \hat{D}_i)$.

4.4 Computational Results

The computational experiments presented in this section are conducted with datasets that have been generated and used in the study presented in Attila et al. [2017]. Moreover, we follow the same Benders' framework presented in Attila et al. [2017]. In this framework, the initial upper bound (UB) and lower bound (LB) are set to ∞ and 0, respectively. Then, LB is updated at every iteration after the DMP is re-solved with a new scenario, while UB is updated only if the AP improves UB, where the minimum of the current UB and the cost corresponding to the new scenario will be taken as the new UB.

For all datasets, the nominal demand is generated within the interval $\bar{D} = [D_{min}, D_{max}]$, where $D_{min} = 50$ and $D_{max} = 100$, and the serviceables holding cost is generated in the interval $h^s = [5, 10]$. The manufacturing and remanufacturing cost are defined as $m = m^f * h^s$ and $r = 2 * h^r$, where we refer to m^f as the *manufacturing factor*, which is set as $m^f = 2$ for all datasets used in the computational tests given below (Tables 4.2, 4.3 and 4.4). We set the backlogging cost as $b = 4 * h^s$. Note that we may set the backlogging cost for the last time period higher than b in order to account for the setup and production costs outside the planning horizon, where $b_T = kb$, such that $k > 1$. Furthermore, we identify the following key parameters and their variations in order to obtain a broad variety of problem characteristics:

- Very high, high, medium and low levels of the setup cost, $K^V = 200 * h^s * D_{max}$, $K^H = 5 * h^s * D_{max}$, $K^M = 2 * h^s * (\frac{D_{max}+D_{min}}{2})$, $K^L = 0.1 * h^s * D_{min}$, respectively.

- High, medium and low levels of nominal returns, $\bar{R}^H \in [0.7 * D_{min}, 0.7 * D_{max}]$, $\bar{R}^M \in [0.5 * D_{min}, 0.5 * D_{max}]$, $\bar{R}^L \in [0.3 * D_{min}, 0.3 * D_{max}]$, respectively.

- High, medium and low probability of constraint violation caused by Γ_t, $p^H = 0.1$, $p^M = 0.05$, $p^L = 0.01$, respectively, where the probability measures are calculated in the same fashion as proposed in Bertsimas and Sim [2004], where we use the approximation $p = 1 - \Phi(\frac{\Gamma_t - 1}{\sqrt{t}})$ to obtain the values for Γ_t^D and Γ_t^R according to desired levels of probability.

- Disposal cost, either less or greater than the remanufacturing cost, set as $f^L = \frac{r}{2}$, $f^G = 2 * r$, respectively.

	K^V			K^H		
p, d	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,G	12.0, *2.0*	11.4, *2.0*	9.1, *2.0*	–	–	–
H,L	11.3, *2.0*	8.9, *2.0*	10.9, *2.0*	9724.4, *3.0*	–	–
M,G	11.7, *2.0*	11.8, *2.0*	9.1, *2.0*	–	–	–
M,L	10.8, *2.0*	9.8, *2.0*	10.3, *2.0*	9362.9, *3.0*	–	–
L,G	14.4, *2.0*	12.7, *2.0*	10.9, *2.0*	9144.7, *3.0*	6143.3, *3.2*	8725.9, *3.0*
L,L	11.9, *2.0*	10.3, *2.0*	9.7, *2.0*	5899.9, *3.2*	8034.6, *3.3*	8959.2, *3.0*
Mean	12.0, *2.0*	10.8, *2.0*	10.0, *2.0*	9024.2, *3.1*	9030.7, *3.3*	9615.3, *3.0*
	K^M			K^L		
p, d	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,G	4808.6, *4.8*	3302.3, *5.3*	2948.0, *4.6*	45.2, *10.6*	12.2, *9.0*	12.5, *9.2*
H,L	1631.0, *4.6*	6378.9, *5.0*	4909.3, *5.0*	11.4, *8.4*	17.9, *10.6*	17.9, *10.2*
M,G	2888.4, *3.5*	1235.8, *5.0*	2578.7, *5.0*	56.4, *9.0*	15.6, *8.8*	12.3, *7.4*
M,L	3999.2, *4.3*	1407.9, *4.8*	3794.7, *5.2*	15.4, *10.4*	25.3, *10.6*	21.0, *10.4*
L,G	2707.4, *3.8*	1270.4, *4.8*	166.0, *3.8*	92.7, *10.4*	20.8, *10.0*	18.9, *9.4*
L,L	973.8, *5.0*	442.2, *4.6*	1789.0, *4.6*	16.1, *8.6*	22.1, *9.2*	17.0, *8.2*
Mean	2834.7, *4.3*	2339.6, *4.9*	2697.6, *4.7*	39.5, *9.6*	19.0, *9.7*	16.6, *9.1*

"–" indicates that time limit was reached for these instances before reaching the desired optimality gap.

Table 4.2: Average computational time (in sec.) and average number of iterations required to reach convergence (given in italic, excluding instances where the time limit is reached) for DMP with $T = 50$ for all datasets.

This experimental design resulted in 72 different combinations and hence 72 datasets were generated, with five instances in each dataset. We also note that the parameter deviations for a given period t, i.e., $\hat{D}_t$ and $\hat{R}_t$, are set as $0.1 * \bar{D}_t$ and $0.1 * \bar{R}_t$, respectively, for all datasets.

All instances were solved as MIPs using Java API for CPLEX 12.7 on an Intel Core i5, 3.30 GHz CPU, 3.29 GHz, 8 GB RAM machine. The terminating condition is met when either the time limit of 10,000s is reached, or a robust optimal solution is found, where $\epsilon = 0.01$. In order to tackle the excessive time requirements while solving the DMP, the MIP gap tolerance for earlier iterations were kept higher, while the final iteration has a relative MIP gap optimality tolerance of 1%. As the last iteration cannot be determined in advance, the MIP gap tolerance is reverted to 1% when $\epsilon \approx 0.01$ is achieved, and kept unchanged until a robust optimal solution is found.

We begin presenting the computational results for DMP, through which we highlight the strengths and weaknesses of the extended reformulations DMP-EFAG and DMP-EFAP. Before discussing detailed results, we note that a common observation for all three formulations is that an optimal solution to the adversarial

problem is achieved under a maximum of 20 seconds for all instances and datasets. On the other hand, for certain instances and datasets, a disproportionate amount of time is required to solve the decision maker's problem. For this reason, we assume that the total time requirements for the decomposition algorithm is representative of the time requirements for solving the decision maker's problem.

As the results in Table 4.2 indicate, there exists a significant difference in the computational times when setup costs vary. We first observe that the instances with very high ($K \in K^V$) and low ($K \in K^L$) setup costs are solved very quickly, whereas the computational times are significantly higher for instances with medium setup costs ($K \in K^M$) and the majority of instances with high setup costs ($K \in K^H$) even exhaust the time limit of 10,000s. When setup costs are decreased towards zero, one would naturally expect the problem to become much easier to solve, since the binary decisions become almost obsolete as one may set all or almost all of them to 1. On the other hand, increasing setup costs from very low up to a certain level naturally complicates the solution procedure, as the combinatorial nature of setup decisions becomes much more dominating as a result of the competition between such decisions. However, once setup costs are significantly increased, then the problem would again become naturally easy to solve, as setups become prohibitive and hence almost all setup variables will be set to 0. We also observe that the computational times in general decrease as the probability of constraint violation decreases from high (p^H) to low (p^L). This is not unexpected, as lower probability of constraint violation would naturally ease the search process for feasible solutions. Finally, although we observe a significant variation in times when nominal return levels vary between $\bar{R} \in \bar{R}^H$, $\bar{R}^M$ and $\bar{R}^L$ or disposal costs vary between f^L and f^G, we can not observe a clear tendency as to when the computational times would increase or decrease. However, this does not mean that we should exclude their impact, because it may be possible to observe a pattern after controlling other factors.

Next, in the same fashion, we present the computational times for the extended aggregate reformulation (DMP-EFAG) and approximate extended reformulation (DMP-EFAP) in Tables 4.3 and 4.4, respectively. In comparison to previous results, DMP-EFAG has a vast improvement on the overall time performances for datasets with $K \in K^H$, where we are now able to solve all instances except one within the time limit. Although the average time requirements remain similar for $K \in K^M$, some datasets such as those with the probability p^L are solved much

	K^V			K^H		
p, d	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,G	1.6, *2.0*	0.8, *2.0*	0.8, *2.0*	854.2, *2.8*	472.9, *2.4*	167.8, *2.4*
H,L	1.6, *2.0*	0.9, *2.0*	0.8, *2.0*	295.9, *2.2*	2.0, *2.2*	2.8, *2.2*
M,G	1.6, *2.0*	1.2, *2.0*	0.7, *2.0*	923.9, *2.4*	1.7, *2.0*	3.2, *2.4*
M,L	1.3, *2.0*	0.8, *2.0*	0.7, *2.0*	3.6, *2.4*	3.2, *2.2*	651.1, *2.6*
L,G	1.0, *2.0*	0.9, *2.0*	0.7, *2.0*	5.4, *2.4*	4.3, *2.2*	3372.1, *2.5*
L,L	0.9, *2.0*	0.8, *2.0*	0.8, *2.0*	3.7 , *2.2*	3.4, *2.2*	5.2, *2.6*
Mean	1.3, *2.0*	0.9, *2.0*	0.8, *2.0*	347.8, *2.4*	81.3, *2.2*	700.4, *2.5*
	K^M			K^L		
p, d	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,G	3140.5, *4.3*	1122.9, *3.8*	152.6, *3.2*	12.3, *6.2*	12.9, *6.4*	13.6, *6.4*
H,L	1603.3, *3.4*	620.1, *3.2*	1770.7, *3.8*	18.4, *6.4*	16.8, *7.2*	12.7, *6.0*
M,G	2646.4, *3.3*	2036.4, *3.3*	3516.4, *3.5*	15.1, *6.2*	12.9, *5.8*	29.8, *6.8*
M,L	3566.6, *3.3*	459.3, *4.2*	894.8, *3.4*	12.0, *5.2*	12.3, *6.0*	13.5, *6.4*
L,G	17.4, *3.0*	25.8, *3.2*	4.8, *3.2*	12.0, *4.8*	11.3, *5.4*	16.2, *6.8*
L,L	70.9, *3.8*	8.8, *3.4*	27.4, *3.4*	13.6, *6.0*	18.2, *7.0*	22.9, *7.6*
Mean	1840.8, *3.5*	712.2, *3.5*	1061.1, *3.4*	13.9, *5.8*	14.1, *6.3*	18.1, *6.7*

Table 4.3: Average computational time (in sec.) and average number of iterations required to reach convergence (given in italic, excluding instances where the time limit is reached) for DMP-EFAG with T=50 for all datasets.

	K^V			K^H		
p, d	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,G	3.2, *2.0*	2.4, *2.0*	2.3, *2.0*	17.4, *2.6*	1099.2, *3.0*	145.7, *2.4*
H,L	2.9, *2.0*	2.5, *2.0*	2.3, *2.0*	33.9, *2.6*	2.7, *2.4*	95.0, *2.6*
M,G	2.4, *2.0*	2.4, *2.0*	2.2, *2.0*	2.3, *2.0*	2.8, *2.6*	2.5, *2.4*
M,L	2.5, *2.0*	2.6, *2.0*	2.4, *2.0*	4.8, *2.8*	830.9, *2.4*	2001.7, *2.3*
L,G	2.7, *2.0*	2.5, *2.0*	2.3, *2.0*	4.3, *2.4*	2.4, *2.0*	2.5, *2.2*
L,L	2.5, *2.0*	2.4, *2.0*	2.3, *2.0*	4.3, *2.4*	2.4, *2.0*	2.4, *2.0*
Mean	2.7, *2.0*	2.4, *2.0*	2.3, *2.0*	11.2, *2.5*	323.4, *2.4*	375.0, *2.3*
	K^M			K^L		
p, d	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,G	6073.2, *4.5*	2407.5, *3.5*	248.7, *3.0*	11.5, *7.2*	14.7, *7.2*	25.2, *8.8*
H,L	2649.5, *4.2*	2409.3, *3.8*	8002.5, *2.0*	11.2, *7.0*	18.8, *8.2*	14.3, *7.0*
M,G	2069.1, *3.3*	1645.5, *3.2*	2069.6, *4.5*	10.7, *5.8*	18.3, *6.8*	22.7, *8.0*
M,L	2016.3, *2.8*	43.7, *3.4*	4032.7, *3.0*	11.7, *6.2*	16.0, *7.8*	19.5, *7.6*
L,G	19.6, *2.8*	12.8, *3.0*	4.2, *2.8*	11.8, *6.2*	17.4, *6.4*	12.8, *6.8*
L,L	21.1, *2.8*	14.8, *3.2*	4.8, *3.0*	13.6, *6.6*	24.7, *8.4*	15.2, *6.2*
Mean	2141.5, *3.4*	1088.9, *3.3*	2393.8, *3.1*	11.8, *6.5*	18.3, *7.5*	18.3, *7.4*

Table 4.4: Average computational time (in sec.) and average number of iterations required to reach convergence (given in italic, excluding instances where the time limit is reached) for DMP-EFAP with T=50 for all datasets, using the maximum interval approach to determine P.

more efficiently, within only 243 seconds. Similar to previous results, datasets with low values of setup costs can still be solved very fast.

As the results in Table 4.4 indicate, DMP-EFAP has even further improved the computational times for datasets with $K \in K^H$ in comparison to DMP-EFAG, where the average time requirement across datasets is now 11.2 seconds for $\bar{R} \in \bar{R}^H$, contrary to the time performance of 347.8 seconds for DMP-EFAG. One possible reason for this can be observed in Figure 4.7, where we observe that for low values of the manufacturing factor m^f, low values of P are sufficient to improve the lower bound so that DMP-EFAG does not have an advantage over the approximate extended formulation DMP-EFAP. Although the approximate extended reformulation has achieved worse times in the set $K \in K^M$ when the high probability p^H of infeasibility parameter is applied, it has a better or similar time performance in comparison to DMP-EFAG for p^M and p^L. The inferential observation from Tables 4.3 and 4.4 is that the variation in computational times with respect to varying levels of setup costs and returns is similar to DMP. Although the computational times for instances with high setup costs ($K \in K^H$) are significantly reduced, instances with medium level setup costs ($K \in K^M$) are the most challenging for extended reformulations.

Another interesting aspect to remark here is the number of scenarios needed to reach convergence. We observe that the total number of iterations mainly varies for different levels of the setup cost. As the results in Tables 4.2, 4.3 and Table 4.4 suggest, lower levels of setup costs tend to increase the number of scenarios required to reach convergence for all three formulations. All instances for $K \in K^V$ have managed to reach a robust optimal solution in 2 iterations, whereas this number is much higher for $K \in K^L$. More specifically, instances with lower setup costs require on average ≈ 9.5 iterations to converge in DMP, whereas this decreases to ≈ 6.3 for DMP-EFAG and to ≈ 7.1 for DMP-EFAP. This behavior is also observed for setup costs where $K \in K^H, K^M$. However, the difference between reformulations is less significant for these classes of datasets, where DMP requires on average ≈ 3.9 iterations, while this amount is ≈ 2.9 for DMP-EFAG and ≈ 2.8 for DMP-EFAP.

Another important consideration is with respect to the improvement in lower bounds when extended reformulations are applied. Figure 4.3 indicates the percentage improvement of the lower bounds at the root node in DMP-EFAG with respect to DMP. We can observe that the most significant gains are achieved for

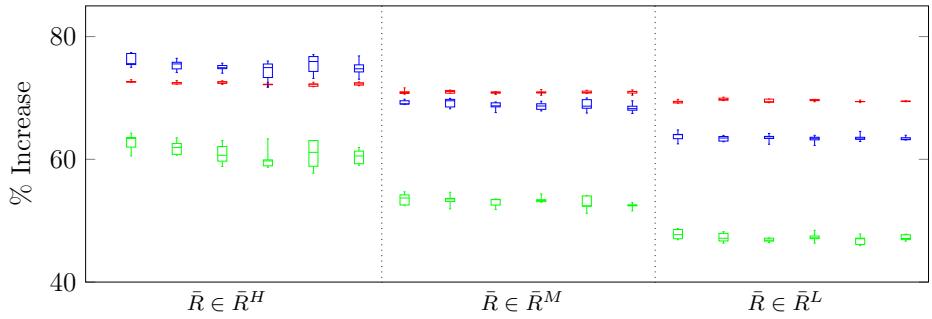

Figure 4.3: Percentage increase in the lower bound for DMP-EFAG with respect to DMP for $\tilde{J}_D = 1$, for different levels of returns ($\bar{R} \in \bar{R}^H, \bar{R}^M, \bar{R}^L$) when $K \in K^V$ (red), $K \in K^H$ (blue) and $K \in K^M$ (green).

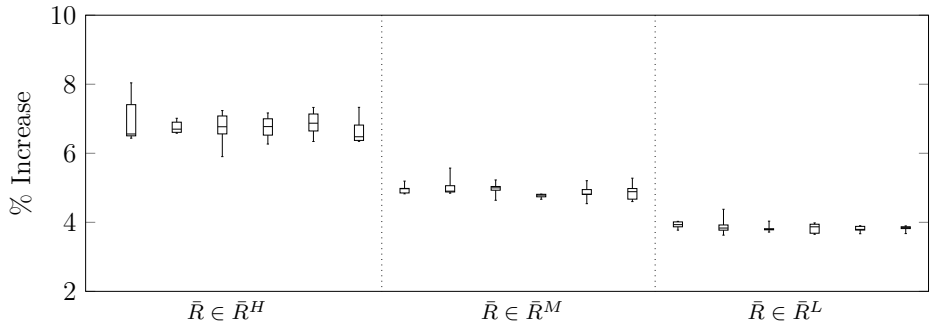

Figure 4.4: Percentage increase in the lower bound for DMP-EFAG with respect to DMP, for different levels of returns: $\bar{R} \in \bar{R}^H, \bar{R}^M, \bar{R}^L$ when $K \in K^L$

datasets with high setup costs, where we are able to obtain an improvement of $\approx$ 75% over varying levels of returns. On the other hand, the gains in lower bounds are slightly less when the setup cost levels are medium, though they are still very effective with an improvement of $\approx$ 60% over varying levels of returns.

In a similar fashion, we present the improvements in lower bounds for low setup cost datasets in Figure 4.4. The tendency of increasing improvements as returns level move towards high can also be observed in this case. However, in comparison with previous results, low setup costs result in less significant gains, with the improvements achieving at most a maximum of 8.0%. This is likely to occur as the fractionality in setup variables in the LP relaxation of DMP would not result in significant cost improvement compared with the integer solution since the setup cost associated with these variables are themselves low.

Another interesting aspect for comparison is the computational behaviour when the extended reformulations are applied, as demonstrated in Figure 4.5. With the

tolerance for the MIP gap set to 1%, we are able to obtain optimal solutions for (DMP) for 71.9% of the instances among all datasets, whereas this percentage shows a considerable increase to 97.8% and 94.4% for (DMP-EFAG) and (DMP-EFAP), respectively. In addition, we observe that only 34.8% of the instances were solved under 100 seconds for (DMP), majority of which are those with $K \in K^L$ (as seen in Table 4.2 before), as they constitute 33.3% of the total number of instances (excluding instances where $K \in K^V$). On the other hand, for (DMP-EFAG) and (DMP-EFAP), we observe a vast increase in the number of instances solved under 100 seconds, with 80.4% and 84.4% of the instances, respectively. This clearly implies a strong improvement in the overall computational performance for both reformulations. Another point to note here is that the variance among the computational time requirements for instances that are solved quickly (under 100 seconds) is very small for all formulations.

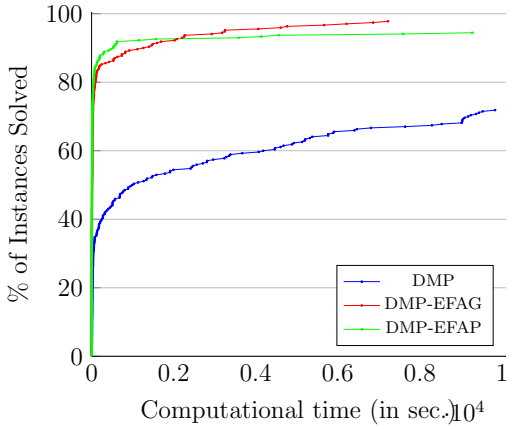

Figure 4.5: Percentage of instances (over datasets with $K \in K^H, K^M, K^L$) that are solved to optimality, where the MIP gap tolerance is set as 1%.

Although the computational time requirements for both (DMP-EFAG) and (DMP-EFAP) have shown a significant improvement, we observe that as computational time increases, (DMP-EFAG) becomes the more effective method, achieving a higher percentage of instances solved in comparison to (DMP-EFAP), which is observed when the time requirement surpasses 2079 seconds. On the other hand, for cases requiring less computational time, (DMP-EFAP) is the method of choice, achieving a higher percentage of instances solved. In addition, we remark that the choice of P plays a crucial role in the computational time performance for (DMP-EFAP), and thus has an impact on the resulting computational time performance.

Next, we analyze the impact of our choice of m^f and P in (DMP-EFAP) on the optimal objective value obtained from its LP relaxation. As we increase P, this results in an increase in the number of extended variables and constraints (4.44), which enable us to obtain tighter relaxations. However, as P increases, the LP relaxation value of (DMP-EFAP) increases towards the LP relaxation value of (DMP-EFAG). Hence, we define P^s as the P value for which this increase becomes negligible. We classify the increase as negligible when the difference in the optimal objective function value between two LP relaxations is below 0.001. The manufacturing factor m^f plays a crucial role in the value of P^s. As m^f increases, remanufacturing and backlogging naturally become more favorable. This would result in a greater number of extended variables becoming active, and hence a higher value of P^s is needed.

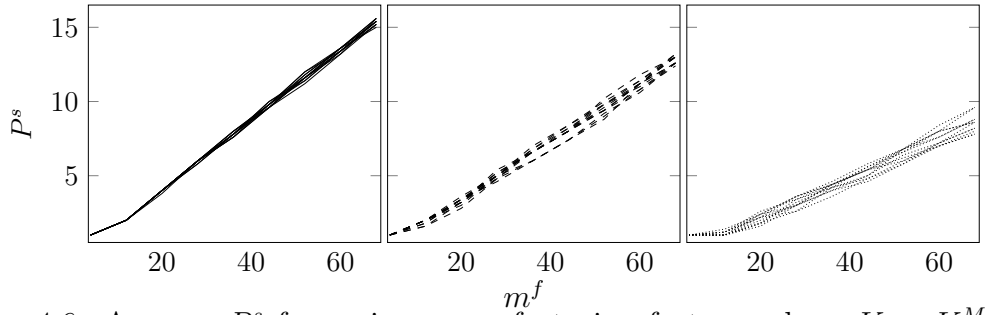

Figure 4.6: Average P^s for various manufacturing factors, where $K \in K^M, K^L$ and $\bar{R} \in R^L$ (straight), R^M (dashed), R^H (dotted).

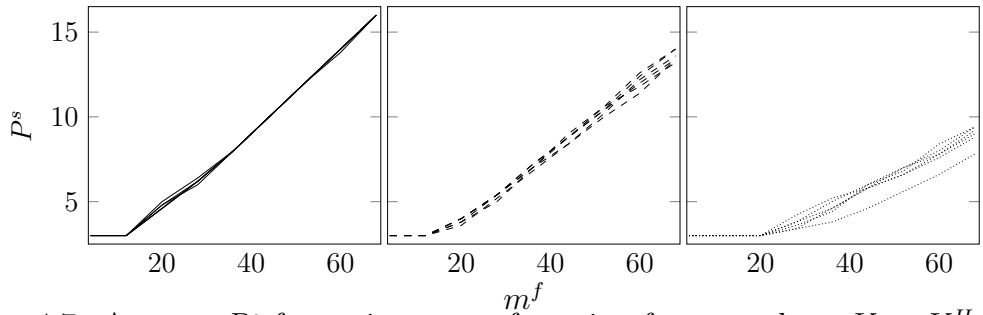

Figure 4.7: Average P^s for various manufacturing factors, where $K \in K^H$, and $\bar{R} \in R^L$ (straight), R^M (dashed), R^H (dotted).

From Figures 4.6 and 4.7, we can observe that the rate of increase in P^s varies with respect to the levels of returns and setup costs, as expected from our discussion above. As seen in Figure 4.7 for high setup costs, P^s remains fairly low for low values of m^f (till around an average value of 18.6) and it starts increasing with

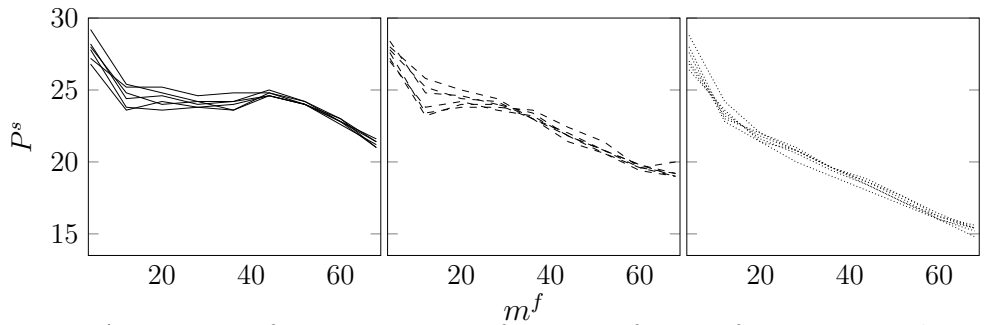

Figure 4.8: Average P^s for various manufacturing factors for $T = 50$, where $K \in K^V$, and $\bar{R} \in R^L$ (straight), R^M (dashed), R^H (dotted).

m^f. This average value of m^f till which P^s remains low drops to 10.6 for medium and low setup costs, as seen in Figure 4.6. On the other hand, since an increase in returns allows higher rates of remanufacturing rather than backlogging (and hence we can expect a decrease in extended production variables), we would expect a slower rate of increase for P^s with higher returns. This behavior can be observed in Figures 4.6 and 4.7, where it is easy to see that P^s starts increasing with m^f but with a gentler slope for datasets with $R \in R^H$ (dotted), in comparison to the those with $R \in R^M, R^L$ (dashed and straight lines, respectively). On the other hand, as seen in Figure 4.8, datasets with $K \in K^V$ have a much greater overall P^s value due to the significant increase in the setup cost. An interesting behaviour we observe here, unlike the previous cases, is the decrease in the value of P^s as m^f increases. As a result of very expensive setup costs, only a very limited number of setups is expected in the optimal solution, and we observe this often with a single setup taking place in the optimal solutions of these instances. As m^f increases, we observe that the setup periods start to split the horizon more equally in order to balance remanufacturing and backlogging costs, which in turn decreases the P^s value. We also observe that occasionally a significantly higher m^f value results in an additional setup period, again contributing to the decrease in P^s.

4.5 Concluding Remarks

In this paper, we study a lot-sizing problem with the remanufacturing option, where uncertainties exist simultaneously for demand and return parameters. Following the setting of our previous work (Attila et al. [2017]), we define parameter uncertainties in the form of polyhedral uncertainties. After a discussion of de-

terministic problem formulation, we present in detail a min-max decomposition approach. The framework iteratively solves a decision maker's problem that evaluates a limited number of scenarios to generate a production plan, and an adversarial problem that generates a scenario that has not yet been considered by the decision maker using the proposed production plan. As the computational challenge of this framework primarily lies in the decision maker's problem, we then investigate this problem further in order to improve computational performance. In particular, we propose a novel approach for formulating the robust lot sizing problem with remanufacturing, which employs two different reformulations. As detailed computational results demonstrate, these extended reformulations are capable of improving the computational performance immensely, in particular the case where setup costs are high. We also present a thorough understanding on the impact of a range of problem parameters, which we believe are invaluable to researchers not only in the area of lot-sizing but also in the broader community of robust optimization. In near future, we would like to address the complexity issues of the adversarial problem. We would like to address a few cost structures that we have not considered in this work. For instance we would like to introduce a variable costs component for our manufacturing costs and make the costs time variant.

Chapter 5

Multiple Components Case Under Uncertainty

5.1 Introduction

In this chapter, we consider a two-level, multi-component setting where a single type of end-item has to be produced to satisfy customer demand. Here, we assume that this demand is deterministic, and only belongs to a single type of item. We refer to this item as the "end-item". Throughout this chapter, we refer to the number of demands associated with this item as the "end-item demand". Furthermore, we consider the case where the end-item level has an independent demand of $D = (D_1, D_2, \ldots, D_{\mathcal{T}})$ that needs to be satisfied until a specific time period for the whole planning horizon $\{1, \ldots, \mathcal{T}\}$. Since this is the case, to satisfy the end-item demand on a given time period t, a minimum of D_t of items have to be available at the beginning of period t, since backlogging is not allowed. The multi-level structure of this problem arises from the assumption that the end-item can only be produced only after a certain set of components have been produced and assembled into the end-item. More specifically, we assume that the end-item is only available to the decision maker if the components in $\{1, \ldots, \mathcal{C}\}$ have been assembled into the end-item.

This arises additional decisions that need to be taken with regards to the availability of components. As we will be discussing in further detail, we are interested in finding optimal production and inventory quantities for these components, where backlogging is not allowed. As a result, the components that are needed to

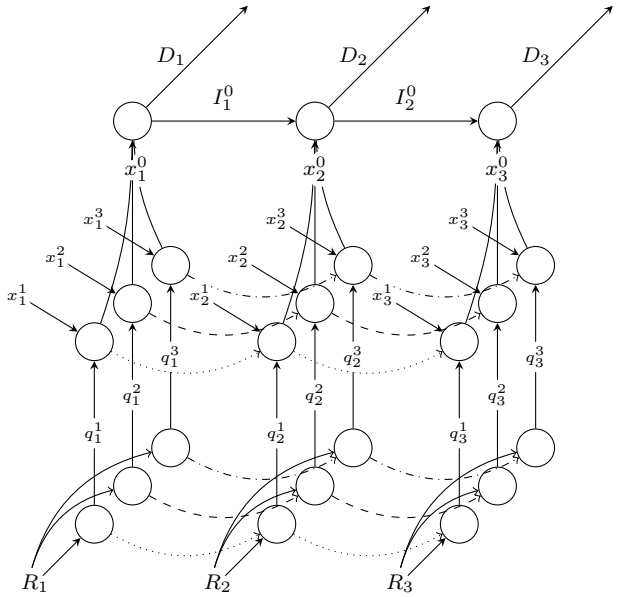

Figure 5.1: Deterministic (2-MCR) problem.

assemble the end-item have to be produced before they are required for assembly.

Component demand occurs due to the assembly decisions that are taken to meet the end-item demand. In order to meet component demand, the decision maker can either choose to manufacture items from scratch, or to remanufacture items that have been returned to the production facility by customers. Note that we do not consider independent demand on components, meaning that the only demand for components is the one that is caused by the assembly decisions for the end-item. Figure 5.1 illustrates the decisions corresponding to each level considered in the 2-MCR problem. Finally, we refer to the decisions taken with regards to meeting components' demand as ones that are taken on the "components level".

5.2 (2-MCR) with Fixed Costs on Component Level (2-MCR-C)

We firstly consider a variation of (2-MCR) where a setup has to be performed in order to produce components (2-MCR-C). In this problem, we assume that there are no setup decisions associated with the end-item level. Instead, we consider linear production costs on the end-item level.

In the following sections, we further provide details about the specific decision

variables and assumptions made for both the deterministic and the robust version of (2-MCR-C).

5.2.1 Deterministic Problem

In advance of introducing the robust setting, we first introduce the deterministic formulation for (2-MCR-C).

In order to generate an optimal production plan, we need to determine the quantities of components to manufacture, remanufacture and the number of items to keep in inventory. These amounts have to be determined for each component that is required to produce the end-item, for the whole planning horizon. We define the set $\{1,\ldots,\mathcal{C}\}$ to indicate the set of components that are required to produce a single end-item. In the remainder of this paper, we use index $c = 0$ to indicate decisions related to the end-item level. We use the set $\{1,\ldots,\mathcal{T}\}$ to define the planning horizon, which includes a finite number of discrete time periods.

Moreover, we denote our manufacturing and remanufacturing decisions with $\mathbf{x} := ((x_1^1, x_2^1, \ldots, x_{\mathcal{T}}^1), (x_1^2, x_2^2, \ldots, x_{\mathcal{T}}^2), \ldots, (x_1^{\mathcal{C}}, x_2^{\mathcal{C}}, \ldots, x_{\mathcal{T}}^{\mathcal{C}}))$ and $\mathbf{q} := ((q_1^1, q_2^1, \ldots, q_{\mathcal{T}}^1), (q_1^2, q_2^2, \ldots, q_{\mathcal{T}}^2), \ldots, (q_1^{\mathcal{C}}, q_2^{\mathcal{C}}, \ldots, q_{\mathcal{T}}^{\mathcal{C}}))$, respectively (see Figure 5.1 for an illustration of the use of these decision variables).

Our decision of producing a component on a given time period (through performing manufacturing and/or remanufacturing) is dependent on whether a setup decision has been made for this component on the given time period. More specifically, we ensure that the following condition holds in an optimal production plan:

$$y_t^c = \begin{cases} 1 & \text{if } x_t^c + q_t^c > 0, \quad \forall c = 1, ..., \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Note that the setup decisions for each component are independent from each other. Additionally, we consider a "joint setup" setting, where a single setup is performed for manufacturing and remanufacturing. In addition to decisions related to production and setup of components, various types of inventory levels need to be decided. More specifically, we consider the following inventory levels:

- *Components inventory:* Components inventory indicates the number of components held in inventory once they have been produced (i.e. once manufactured or remanufactured). We define the following to indicate the inven-

tory levels available at the end of a given time period for each component: $\mathbf{I} := ((I_1^1, I_2^1, \ldots, I_{\mathcal{T}}^1), (I_1^2, I_2^2, \ldots, I_{\mathcal{T}}^2), \ldots, (I_1^{\mathcal{C}}, I_2^{\mathcal{C}}, \ldots, I_{\mathcal{T}}^{\mathcal{C}}))$.

- *End-item inventory:* Likewise, the end-item can be held in inventory once assembled. We indicate these decisions with $I^0 := (I_1^0, I_2^0, \ldots, I_{\mathcal{T}}^0)$.

- *Returns inventory:* This refers to the number of unprocessed returns held in inventory at the end of a given time period. In the formulations given under this section, we consider that every component in $\{1, \ldots, \mathcal{C}\}$ can be recovered for remanufacturing fully. In other words, we assume that the number of remanufacturable returns is equivalent for every component.

In addition to the decisions associated with inventory levels and producing components, the optimal number of end-items to produce in each time period need to be determined. For this purpose, we define $x^0 := (x_1^0, x_2^0, \ldots, x_{\mathcal{T}}^0)$, which indicates the number of end-item level items produced on each time period. Note that there is no setup decision involved with the end-item level.

We consider that a time-invariant cost is incurred for each one of the decisions given above. Specifically, we represent these costs as $m := (m^1, m^2, \ldots, m^{\mathcal{C}})$, $r := (r^1, r^2, \ldots, r^{\mathcal{C}})$, $h := (h^1, h^2, \ldots, h^{\mathcal{C}})$, $w := (w^1, w^2, \ldots, w^{\mathcal{C}})$, $K := (K^1, K^2, \ldots, K^{\mathcal{C}})$, m^0 and h^0 for manufacturing, remanufacturing, components inventory, returns inventory, setup, end-item assembly and end-item inventory costs, respectively. We assume that all costs are time-invariant. Additionally, we make the assumption that manufacturing a given component is more expensive than remanufacturing (i.e. $m^c > r^c,\ \forall c \in \{1, \ldots, \mathcal{C}\}$). Note that the decisions on the component level have a specific cost component for each component in the set $\{1, \ldots, \mathcal{C}\}$.

Following this setting, we may write the deterministic (2-MCR-C) formulation as given below, where our objective is to minimize the total operational cost:

$$\min \sum_{t=1}^{\mathcal{T}} \Big(m^0 x_t^0 + h^0 I_t^0 + \sum_{c=1}^{\mathcal{C}} (h^c I_t^c + w^c \sum_{i=1}^{t} (R_i - q_i^c) + m^c x_t^c + r^c q_t^c + K^c y_t^c) \Big) \tag{2-MCR-C}$$

$$x_t^0 + I_{t-1}^0 = I_t^0 + D_t \qquad \begin{matrix} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{matrix} \tag{5.3}$$

$$x_t^c + q_t^c + I_{t-1}^c = I_t^c + x_t^0 \qquad \begin{matrix}\forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C}\end{matrix} \tag{5.4}$$

$$\sum_{i=1}^{t}(R_i - q_i^c) \geq 0 \qquad \begin{matrix}\forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C}\end{matrix} \tag{5.5}$$

$$x_t^c + q_t^c \leq M_t y_t^c \qquad \begin{matrix}\forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C}\end{matrix} \tag{5.6}$$

$$x, q, I, I^0 \geq 0 \tag{5.7}$$

$$y \in \{0,1\}^{\mathcal{T}\times\mathcal{C}} \tag{5.8}$$

Constraint (5.3) ensures that the flow balance is conserved for the end-item level. Similarly, constraint (5.4) is used to ensure that the flow balance for components, with a demand of x_t^0 on period t. Constraint (5.5) ensures that the returns inventory level is non-negative. Finally, constraint (5.6) is the setup constraint for components, where a joint setup cost of K^c is incurred if setup takes place for component c in a given time period.

Note that, w.l.o.g., we assume for the sake of simplicity that one unit of each component is required to assemble one unit of the end-item. This assumption can be easily relaxed by replacing the constraint (5.4) with $x_t^c + q_t^c + I_{t-1}^c = I_t^c + u^c x_t^0$, where u^c indicates number of components of type c needed to assemble one unit end-item.

5.2.2 Robust Formulation

Following the deterministic formulation, we now present a robust variation of (2-MCR-C), where we assume that the returns level is uncertain. In order to define the uncertainty around returns, we introduce the following budgeted polytope, as introduced in the work of Bertsimas and Sim [2004] and Bertsimas and Thiele [2006].

$$Z(\Gamma) := \{z \in [-1,1]^{\mathcal{T}} : \sum_{i=1}^{t} |z_i| \leq \Gamma_t,\ \forall t = 1, \ldots, \mathcal{T}\} \tag{5.9}$$

$$U(\Gamma) := \{R \in \mathbb{R}_+^{\mathcal{T}} : R_t = \bar{R}_t + \hat{R}_t z_t, z \in Z(\Gamma)\} \tag{5.10}$$

Here, we assume that the level of returns can vary in the interval $[\bar{R}_t - \hat{R}_t, \bar{R}_t + \hat{R}_t]$. In order to determine a particular scenario realization in $U(\Gamma)$, we use the variable z_i. While $z_i = 0$ implies a scenario where the level of returns is equivalent

to its nominal realization $\bar{R}_t$, cases where $|z_i| = 1$ suggest that either the positive (when $z_t = 1$) or the negative extreme (when $z_t = -1$) is realized.

We are interested in formulating the case where the decision maker has to incur an additional holding cost for scenarios where the number of returns held in inventory at a given time period is greater than that of nominal returns' inventory level. On the other hand, if the returns inventory level drops below the nominal returns' inventory level, the decision maker has to manufacture additional components in order to meet the number of items held in the component level inventory (since these returns will no longer be available in the given scenario). The latter results in an increase in costs, since we assume that remanufacturing is a cheaper alternative to manufacturing components.

Furthermore, let the following define the extreme points in the set $U(\Gamma)$. In a robust setting, we are interested in finding an optimal solution that remains feasible for the extreme points of the following set:

$$U(\Gamma) := Conv(\{R^1, R^2, \ldots, R^{\mathcal{S}}\}) \tag{5.11}$$

In order to do this, we need to ensure feasibility through the entire uncertainty set $U(\Gamma)$ for constraints where returns are present. One way of implementing this is by changing constraint (5.5) from the deterministic (2-MCR-C) formulation into one which remains feasible for the extreme points of $Conv(\{R^1, R^2, \ldots, R^{\mathcal{S}}\})$. Doing so, we obtain the following robust formulation:

$$\min \sum_{t=1}^{\mathcal{T}} \Big(m^0 x_t^0 + h^0 I_t^0 + \tag{2-MCR-CR}$$

$$\sum_{c=1}^{\mathcal{C}} (h^c I_t^c + w^c Q_t^c + m^c x_t^c + r^c q_t^c + K^c y_t^c) \Big) \tag{5.12}$$

$$x_t^0 + I_{t-1}^0 = I_t^0 + D_t \qquad \begin{array}{l} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{array} \tag{5.13}$$

$$x_t^c + q_t^c + I_{t-1}^c = I_t^c + x_t^0 \qquad \begin{array}{l} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{array} \tag{5.14}$$

$$Q_t^c \geq \sum_{i=1}^{t} (R_i^s - q_i^c) \qquad \begin{array}{l} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \\ \forall s \in \xi \end{array} \tag{5.15}$$

$$x_t^c + q_t^c \leq M_t y_t^c \qquad \begin{matrix} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{matrix} \tag{5.16}$$

$$Q, x, q \geq 0 \tag{5.17}$$

$$y \in \{0, 1\}^{\mathcal{T} \times \mathcal{C}} \tag{5.18}$$

The main difference between (2-MCR-CR) and (2-MCR-C) is regarding the returns flow balance constraint. While (2-MCR-C) is only feasible for a given value of returns, the decisions in (2-MCR-CR) have to be feasible for the returns scenario set $\xi = \{1, \ldots, \mathcal{S}\}$. For this purpose, we repeat constraint (5.15) for each extreme point in $U(\Gamma)$.

A crucial observation with regards to (2-MCR-CR) is that the production decisions in this formulation are static and cannot be adjusted to the specific scenario realized. As we will further discuss below, this results in (2-MCR-CR) to be unusable for the given uncertainty sets under this setting. This is because for each scenario in the returns uncertainty set ξ, a production rule has to be satisfied. This suggests that the production quantities need to be adjustable according to the specific scenario considered.

Thus, we define production variables to be adjustable according to the specific scenario realized, where:

$$\mathcal{W}_{t,\xi} = \max_{s \in \xi} \{ \sum_{c=1}^{\mathcal{C}} (w^c Q_t^{c,s} + m^c x_t^{c,s} + r^c q_t^{c,s}) \} \tag{5.19}$$

which indicates the maximum total production and returns inventory cost in the restricted uncertainty set. According to this, we may present the robust adjustable formulation as follows:

$$\min \ \sum_{t=1}^{\mathcal{T}} \Big(m^0 x_t^0 + h^0 I_t^0 + \mathcal{W}_{t,\xi} + \sum_{c=1}^{\mathcal{C}} (h^c I_t^c + K^c y_t^c) \Big) \tag{2-MCR-CRA}$$

$$x_t^0 + I_{t-1}^0 = I_t^0 + D_t \qquad \begin{matrix} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{matrix} \tag{5.20}$$

$$x_t^{c,s} + q_t^{c,s} + I_{t-1}^c = I_t^c + x_t^0 \qquad \begin{matrix} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{matrix} \tag{5.21}$$

$$Q_t^{c,s} = \sum_{i=1}^{t} (R_i^s - q_i^{c,s}) \qquad \begin{array}{l} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \\ \forall s \in \xi \end{array} \tag{5.22}$$

$$x_t^{c,s} + q_t^{c,s} \leq M_t y_t^c \qquad \begin{array}{l} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \\ \forall s \in \xi \end{array} \tag{5.23}$$

$$Q, x, q \geq 0 \tag{5.24}$$

$$y \in \{0,1\}^{\mathcal{T} \times \mathcal{C}} \tag{5.25}$$

Note that usually formulations with adjustable variables (such as (2-MCR-CRA)) need to satisfy non-anticipativity constraints in order to ensure consistency among scenarios. In our work, we show that the above formulation satisfies a specific production rule, through which the non-anticipativity of adjustable production variables is satisfied. This production rule is specified in Proposition 5.

In order to examine this production rule, let us rewrite the implicit return balance constraint

$$\sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^s - q_i^{c,s}) \geq 0 \tag{5.26}$$

as

$$\bar{R}_t + \hat{R}_t z_t^s - q_t^{c,s} + p_{t-1}^{c,s} = p_t^{c,s} \tag{5.27}$$

where $p_t^{c,s}$ stands for the number of returns kept in returns inventory on period t for a given scenario s. We make the observation that $\forall s \in \xi$ such that $x_t^{c,s} p_t^{c,s} = 0$ holds.

Proposition 5. *$x_t^{c,s} p_t^{c,s} = 0,\ \forall s \in \xi$ holds in an optimal solution to (2-MCR-CRA) when $m^c > r^c,\ \forall c = 1, \ldots, \mathcal{C}$.*

Proof. Let a and b indicate two consecutive setup periods in a given production plan, where $y_a = 1,\ y_b = 1,\ a < b$ and $y_t = 0,\ \forall t \in \{a+1, .., b-1\}$. We will show that there exists an alternative solution with a smaller total cost when $x_a^{c,s} p_a^{c,s} \neq 0$. Consider the following cases:

1. *Available returns on time period a are kept in returns inventory and never used for remanufacturing.*

In order to satisfy component level demand, we have $x_a^{c,s} > 0, \exists s \in \xi$ since $q_a^{c,s} = 0$. Remanufacturing an additional item on period a reduces the total returns holding cost by $w^c(\mathcal{T} - a + 1)$, and the total production cost by $(m^c - r^c)$, leading to a total cost reduction of $w^c(\mathcal{T} - a + 1) + (m^c - r^c) > 0$ when $m^c > r^c$ and $w^c > 0$. Since there is a joint setup decision, this does not affect the total setup cost incurred.

2. *Available returns on time period a are kept in inventory and used for remanufacturing on time period b.*
   We have $x_a^{c,s} > 0$ to satisfy component level demand since $q_a^{c,s} = 0$. Remanufacturing one less return on period b reduces the total returns holding cost between time periods a and b by $w^c(b-a)$. On the other hand, this increases production costs on b by $(m^c - r^c)$ since component level demand has to be satisfied. As this item can now be used to remanufacture an additional item on a, this will lead to a reduction in total production costs by $(m^c - r^c)$ on period a. As a result, the resulting difference in the total cost can be given as: $-w^c(b-a) + (m^c - r^c) - (m^c - r^c) = -w^c(b-a) < 0$, given that $b - a > 0$ and $w^c > 0$, leading to an overall cost reduction of $w^c(b - a)$.

Thus, in both cases we have an alternative solution where costs can be reduced. □

A well-known method for obtaining such extreme points while keeping track of the optimal solution for the production plan is to implement an iterative decomposition framework studied in the work of Bienstock and Özbay [2008]. This method is often referred to as the min-max approach (or the decomposition approach), where we consider two sub-problems: the Decision Maker's Problem and the Adversarial Problem. The objective of the Decision Maker's Problem is to generate an optimal production plan for a subset of extreme points in $U(\Gamma)$. On the other hand, the Adversarial Problem seeks for a new extreme that worsens the total operational cost found in the Decision Maker's problem. Once a new extreme point is found, it is fed back into the Decision Maker's Problem, where this procedure is continued until a new extreme point that worsens the total cost can not be found. In the remainder of this section, we present the Decision Maker's Problem (DMP) and the Adversarial Problem (AP).

5.2.3 Decision Maker's Problem

The decision maker's problem seeks for an optimal production plan that is feasible for a finite, restricted set of return scenarios. We indicate this set of scenarios as $\xi^R = \{1, \ldots, \mathcal{S}\}$. More specifically, manufacturing and remanufacturing variables are defined for each scenario in DMP. In the remainder of this section, we provide further details regarding this issue.

$$\min \ \left(\pi + \sum_{t=1}^{\mathcal{T}} \left(m^0 x_t^0 + h^0 I_t^0 + \sum_{c=1}^{\mathcal{C}} (h^c I_t^c + K^c y_t^c)\right)\right) \tag{DMP}$$

$$\pi \geq \sum_{t=1}^{\mathcal{T}} \sum_{c=1}^{\mathcal{C}} (w^c \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^{s*} - q_i^{c,s}) + m^c x_t^{c,s} + r^c q_t^{c,s}) \quad \forall s \in \xi^R \tag{5.28}$$

$$\sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i z_i^{s*} - q_i^{c,s}) \geq 0 \quad \begin{array}{l} \forall s \in \xi^R \\ \forall t = 1..\mathcal{T} \\ \forall c = 1..\mathcal{C} \end{array} \tag{5.29}$$

$$x_t^0 + I_{t-1}^0 \geq I_t^0 + D_t \quad \begin{array}{l} \forall t = 1..\mathcal{T} \\ \forall s \in \xi^R \end{array} \tag{5.30}$$

$$x_t^{c,s} + q_t^{c,s} + I_{t-1}^c = I_t^c + x_t^0 \quad \begin{array}{l} \forall t = 1..\mathcal{T} \\ \forall c = 1..\mathcal{C} \end{array} \tag{5.31}$$

$$x_t^{c,s} + q_t^{c,s} \leq M_t y_t^c \quad \begin{array}{l} \forall s \in \xi^R \\ \forall t = 1..\mathcal{T} \\ \forall c = 1..\mathcal{C} \end{array} \tag{5.32}$$

Here, π represents the largest total returns inventory, manufacturing and remanufacturing costs among scenarios $s \in \xi^R$. Note that z_t^{s*} is an input to DMP, where this indicates the optimal solution obtained for variables z_t in AP. Constraint (5.29) ensures that the returns inventory is non-negative for period t and scenario s. The remaining constraints are used to ensure the aforementioned conditions and flow conversions are maintained. Specifically, constraint (5.30) ensures that final-level item demand is satisfied, while constraint (5.31) is the flow balance constraint for the component level, where the demand for each component in the set $\{1, ..., \mathcal{C}\}$ has to be satisfied.

Note that we may rewrite constraint (5.32) in the following form (due to con-

straint (5.31)):

$$I_t^c + x_t^0 - I_{t-1}^c \leq M_t y_t^c \qquad \forall t = 1..\mathcal{T},\ \forall c = 1..\mathcal{C} \tag{5.33}$$

By doing so, we are able to eliminate the need for repeating constraint (5.32) for each scenario: $\forall s \in \xi^R$.

In this formulation we introduce scenario-based manufacturing and remanufacturing variables on the component level instead of using common production variables across scenarios. This is mainly because having common remanufacturing variables in constraint (5.29) suggests that the optimal remanufacturing amounts have to be limited to the smallest number of returns available to us. However, we are interested in relaxing this assumption, since we would like to evaluate the case where the decision maker will be able to remanufacture these items in scenarios where these returns are available. Below we provide an example problem with $\mathcal{T} = 1$ and $\mathcal{C} = 1$ with common manufacturing and decision variables, in order to motivate the reasons behind introducing scenario-based variables:

Example 5.2.1. *Consider the following example where* $\mathcal{T} = 1$, $\mathcal{C} = 1$, $\bar{R}_1 = 5$, $\hat{R}_1 = 5$ *and* $D_1 = 80$.

In the example shown above, we initially consider the first scenario where the nominal return $R_1^1 = 5$ is realized. Thus, this amount will be fully remanufactured (since remanufacturing is cheaper than manufacturing), which implies $q_1^1 = 5$. In the second iteration, let us assume that the return scenario generated by AP is $R_1^2 = 0$. In order to satisfy constraint (5.29), we now have to remanufacture the smallest amount of returns available to us regardless of the scenario considered, enforcing $q_1^1 = 0$ even though $q_1^1 = 5$ is still feasible for the first scenario. As a result, there exists the risk of calculating the total cost incorrectly (or suboptimally) for some scenarios. Introducing scenario-based production variables eliminates this problem through allowing an interchangable use of the manufacturing and remanufacturing variables to satisfy component-level demand. This ensures that the production plan will imply the smallest cost for the worst-case return scenario in DMP. Below we provide the same example with scenario-based variables to demonstrate the changes in costs:

As seen from Figure 5.2, introducing scenario-based production variables allows us to remanufacture all returns available to us for each given scenario. As a

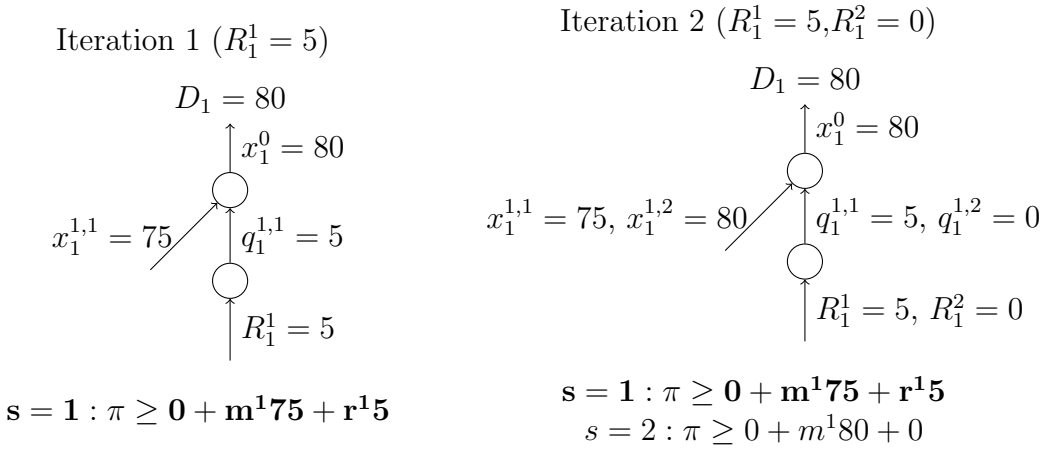

Figure 5.2: Example with $\mathcal{T} = 1$ and $\mathcal{C} = 1$

result, we are allowed to remanufacture as much as the returns available in a given scenario, regardless of the return realizations in other scenarios.

A crucial observation here is regarding the production rule given in Proposition 5, which is also satisfied in an optimal solution to (DMP).

According to the production rule given in Proposition 5, in an optimal solution, the decision maker will remanufacture as much as possible in order to meet the component demand. As a result of this, manufacturing is only performed when no returns are left on a given time period (i.e. the number of returns kept in inventory on this time period will be zero, which can be shown with $p_t^{c,s} = 0$). As we will further discuss in detail, this production rule plays a crucial rule in formulating AP.

Note that since this production rule holds in an optimal solution, it is sufficient to provide the decision maker with the optimal total number of items produced on each time period (i.e. the total amount of items manufactured and remanufactured for each component in $c \in \{1, \ldots, \mathcal{C}\}$ for the whole planning horizon). This is because the decision maker can implement this production rule based on the specific scenario realized. Thus, having different optimal values for the scenario-based production variables does not arise any problems in terms of deciding the specific production plan to provide to the decision maker. This is also a desirable property for the AP, since the input to AP can also be represented as an aggregated number of components produced in this case. As we will discuss in detail in the next section, providing an input of this type to AP requires that the production rule is accounted for in the AP.

5.2.4 Adversarial Problem

As stated previously, the Adversarial Problem (AP) generates a new return scenario that worsens the total cost in (DMP). In order to formulate (AP), we first need to identify the outputs from (DMP) that will be used in (AP) as inputs. We define these inputs as follows:

- y_t^{c*}: Optimal setup decision for component c in time period t, based on the production plan obtained in (DMP).
- x_t^{0*}: Optimal assembly quantity for time period t according to the production plan acquired from (DMP).
- I_t^{c*}: Optimal inventory level for component c on time period t, based on the optimal assembly and production decisions in (DMP).

Note that the optimal values for these inputs are equivalent across different scenarios in (DMP). For this reason, we do not need to make a decision regarding which specific scenario to feed into (AP). Instead, to determine the specific manufacturing and remanufacturing quantities in (AP), we define manufacturing and remanufacturing decisions as decision variables in (AP). However, defining manufacturing and remanufacturing levels as decision variables in (AP) does not necessarily imply that the condition given in Proposition 5 will hold in (AP). When this is the case, the optimal return scenario found in (AP) is under the risk of misinterpreting the total cost, since some feasible solutions in (AP) may disregard this production rule. In order to achieve the correct total cost and to avoid any inconsistencies between (DMP) and (AP), we impose this rule on (AP) manually through introducing new constraints and variables.

Let us define the decision variables for manufacturing and remanufacturing as:

- x_t^c: Number of components c manufactured in order to achieve the worst total production and returns inventory cost in (AP).
- q_t^c: Number of components c remanufactured in order to achieve the worst total production and returns inventory cost in (AP).

In addition, in (AP) we explicitly define returns inventory levels by introducing the decision variable $p := ((p_1^1, p_2^1, \ldots, p_{\mathcal{T}}^1), (p_1^2, p_2^2, \ldots, p_{\mathcal{T}}^2), \ldots, (p_1^{\mathcal{C}}, p_2^{\mathcal{C}}, \ldots, p_{\mathcal{T}}^{\mathcal{C}}))$.

Note that since the output for (AP) is a single return scenario (which is not dependent on any of the previous scenarios generated), we drop the subscript s from the production variables. We represent the optimal return scenario obtained from (AP) through the optimal value of the decision variable $z := ((z_1, z_2, \ldots, z_{\mathcal{T}})$, as discussed previously.

Under this setting, we write (AP) as:

$$\max\ \pi \tag{AP}$$

$$\pi = \sum_{t=1}^{\mathcal{T}} \sum_{c=1}^{\mathcal{C}} (w^c p_t^c + m^c x_t^c + r^c q_t^c) \tag{5.34}$$

$$x_t^c + q_t^c + I_{t-1}^{c*} = x_t^{0*} + I_t^{c*} \qquad \forall t = 1 \ldots \mathcal{T},\ \forall c = 1 \ldots \mathcal{C} \tag{5.35}$$

$$p_{t-1}^c + \bar{R}_t + \hat{R}_t z_t = q_t^c + p_t^c \qquad \forall t = 1 \ldots \mathcal{T},\ \forall c = 1 \ldots \mathcal{C} \tag{5.36}$$

$$M_t^2 a_t^c \geq p_t^c \qquad \forall t = 1 \ldots \mathcal{T},\ \forall c = 1 \ldots \mathcal{C} \tag{5.37}$$

$$M_t^1 (1 - a_t^c) \geq x_t^c \qquad \forall t = 1 \ldots \mathcal{T},\ \forall c = 1 \ldots \mathcal{C} \tag{5.38}$$

$$x_t^c + q_t^c \leq M_t^1 y_t^{c*} \qquad \forall t = 1 \ldots \mathcal{T},\ \forall c = 1 \ldots \mathcal{C} \tag{5.39}$$

$$-1 \leq z_i \leq 1 \qquad \forall t = 1 \ldots \mathcal{T} \tag{5.40}$$

$$\sum_{i=1}^{t} |z_i| \leq \Gamma_t \qquad \forall t = 1 \ldots \mathcal{T} \tag{5.41}$$

$$a_t^c \in \{0, 1\}^{\mathcal{T} \times \mathcal{C}} \tag{5.42}$$

In the formulation given above, our objective is to maximize the total production (i.e. manufacturing and remanufacturing) and returns inventory cost, which is denoted by π and constraint (5.34). We include constraint (5.35) to ensure that the production variables in (AP) are feasible by allowing the production and inventory decisions to be worsened for the current production plan. On the other hand, we ensure flow conversion for the returns via constraint (5.36). Likewise, constraint (5.39) ensures that the setup decisions are followed in (AP). In this constraint M_t^1 and M_t^2 are sufficiently large numbers, where we set $M_t^1 = \max\{\sum_{i=t}^{\mathcal{T}} D_i, \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i)\}$ and $M_t^2 = \sum_{i=1}^{t} (\bar{R}_i + \hat{R}_i)$. Furthermore, constraints (5.40) and (5.41) are used to define the returns uncertainty set.

An important point to note here is the use of decision variables x_t^c and q_t^c in AP. These decision variables are not acquired from DMP, nor are they used as an

input to DMP. The sole use of these production variables is to ensure that the production rule is satisfied in AP.

We introduce constraints (5.37) and (5.38) in order to implement the production rule $p_t^c x_t^c = 0,\ \forall t = 1, \ldots, \mathcal{T},\ \forall c = 1, \ldots, \mathcal{C}$. To do this, we introduce the binary decision variable a_t^c, which is used to ensure that only one of the following cases is realized in (AP) for a given component c and period t:

1. Number of returns at the beginning of t is sufficient to satisfy x_t^0, thus component c does not need to be manufactured on t i.e. $x_t^c = 0,\ p_t^c \geq 0$

2. Number of returns at the beginning of t is insufficient to satisfy x_t^0. Then, the returns available to us on period t will be depleted (i.e. $p_t^c = 0$), and manufacturing has to be performed in order to meet the demand of x_t^0, i.e. $x_t^c \geq 0$.

This ensures that variables p_t^c and x_t^c are not mutually positive in a given time period t for a specific component c. Under this setting, enforcing the original flow balance constraints is sufficient to assure that the production levels (i.e. manufacturing and remanufacturing levels) generated in (AP) does not result in an infeasible production plan to (DMP). Therefore, under this setting, the total inventory and production costs for (AP) are estimated correctly, by eliminating solutions that result in an overestimation of the total cost in (DMP).

Finally, we note that constraint (5.41) may be linearized by introducing the following constraints:

$$b_t \geq z_t \qquad \forall t = 1 \ldots \mathcal{T} \tag{5.43}$$

$$b_t \geq -z_t \qquad \forall t = 1 \ldots \mathcal{T} \tag{5.44}$$

$$b_t \leq 1 \qquad \forall t = 1 \ldots \mathcal{T} \tag{5.45}$$

and by replacing constraint (5.41) with the following:

$$\sum_{i=1}^{t} b_i \leq \Gamma_t \tag{5.46}$$

to ensure that $b_t = |z_t|$.

5.3 Computational tests

In this section, we present the results of the computational tests conducted for DMP and AP. All the runs presented under this section are restricted to a time limit of 2 hours (7200 seconds). Additionally, an iteration limit of 20 is enforced for the decomposition algorithm. Specifically, this time limit restricts the total time taken to solve DMP and AP to optimality (including all iterations). In addition, we set the UB-LB gap for the decomposition algorithm, ϵ as 0.01 for all datasets and instances.

5.3.1 Instance generation

The instances used in the computational tests were generated randomly, for various levels/assumptions for the following inputs. A total of five instances were generated in each dataset (for each combination of the levels/assumptions given below), leading to a total of 324 datasets and 1620 instances. Note that demand values have been generated randomly within the interval $D_t \in [D^{min}, D^{max}]$ where we set $D^{min} = 20$ and $D^{max} = 30$ for all datasets. In addition, let $D^{med} = \frac{D^{max}+D^{min}}{2}$, which indicates the middle value in $[D^{min}, D^{max}]$. We present the specific sets that have been generated below, including the ranges and measures used for each set.

- **Nominal returns ($\bar{R}_t$)**: low ($\bar{R}_t \in [0.3 * D^{min}, 0.3 * D^{max}] \in \bar{R}_t^L$), medium ($\bar{R}_t \in [0.8 * D^{min}, 0.8 * D^{max}] \in \bar{R}_t^M$), high ($\bar{R}_t \in [2 * D^{min}, 2 * D^{max}] \in \bar{R}_t^H$).

- **Probability of constraint violation**: low ($p^L = 0.01$), medium ($p^M = 0.05$) and high ($p^H = 0.10$). These probabilities are used to compute Γ_t, in a similar fashion to Bertsimas and Sim [2004].

- **Setup costs for components (K^c)**: low ($K^c = \lceil 0.1 * h^c * D^{min} \rceil \in K^L$), medium ($K^c = \lceil 2 * h^c * D^{med} \rceil \in K^M$) and high ($K^c = \lceil 5 * h^c * D^{max} \rceil \in K^H$).

- **Manufacturing cost for the end-item level (m^0)**: low ($m^0 = 0.1 * h^0 \in m^{0L}$) and medium ($m^0 = 2.0 * h^0 \in m^{0M}$).

- **Deviation level for returns ($\hat{R}_t$)**: low ($\hat{R}_t = \lceil 0.1 * \bar{R}_t \rceil \in \hat{R}^L$), medium ($\hat{R}_t = \lceil 0.5 * \bar{R}_t \rceil \in \hat{R}^M$) and high ($\hat{R}_t = \bar{R}_t \in \hat{R}^H$).

- **Whether the condition $h^0 > \sum_{c=1}^{\mathcal{C}} h^c$ holds.** We consider the two different cases given below. In both cases, the value for h^c is generated randomly in the intervals $h^{c,low} \in [1, 30]$, $h^{c,med} \in [50, 150]$ and $h^{c,high} \in [250, 500]$, where $h^c \in h^{c,low}$, $h^{c+1} \in h^{c,med}$ and $h^{c+2} \in h^{c,high}$, $\forall c = 1, \ldots, \mathcal{C} - 2$. By following this procedure, we are able to consider various cost ranges for different components.

  1. The end-item level inventory cost is determined as $h^0 = f \sum_{c=1}^{\mathcal{C}} h^c$, where f is a constant and $f > 1$, which implies that the condition is met. In the following computational tests, this value is set as $f = 1.2$ for datasets where this condition is met. Throughout the remainder of this section, we refer to these datasets as ones where "h^0 is restricted".
  2. The condition $h^0 > \sum_{c=1}^{\mathcal{C}} h^c$ is not necessarily satisfied, and the end-item level inventory cost is generated randomly in the interval $h^0 \in [1, 500]$. Likewise, we refer to these datasets as ones where "h^0 is unrestricted".

The remaining costs are determined as $m^c = 1.5h^c$, $r^c = 1.2h^c$ and $w^c = 0.5h^c$ for all instances.

5.3.2 Instances where h^0 is restricted

Firstly, let us consider the datasets where h^0 is restricted. As seen in Figure 5.3 and Table 5.2, the AP is solved to optimality for the vast majority of the datasets where h^0 is restricted, whereas only 83% of the instances were solved to optimatily for DMP under the time limit. This is also clear from the total time requirements for the decomposition algorithm (DMP+AP) shown in Figure 5.3, since there is not a major difference in the percentage of instances solved between DMP and DMP+AP.

As we will further investigate later on in this section, the time performance for datasets where h^0 is unrestricted vary considerably from those where h^0 is restricted. Consequently, the difference between the percentage of instances that were solved to optimality under the time limit of 7200 seconds in these two classes of datasets are considerably different, as seen in Figure 5.3. Overall, we observe that the number of instances that are solved to optimality decrease considerably when the value of h^0 is unrestricted. Specifically, the DMP is only able to solve

		K^H			K^M			K^L		
	p, m^0	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
$\hat{R} \in \hat{R}^H$	H,M	3.7,*2.0*	–	21.3,*1.0*	1.0,*2.0*	1023.2,*6.6*	0.9,*1.0*	0.1,*2.0*	0.3,*2.0*	0.1,*1.0*
	H,L	3.7,*2.0*	–	–	0.9,*2.0*	4954.3,*10.0*	1.2,*1.0*	0.1,*2.0*	0.4,*2.0*	0.4,*2.0*
	M,M	3.5,*2.0*	–	66.6,*1.4*	1.0,*2.0*	2127.8,*11.6*	0.7,*1.0*	0.1,*2.0*	0.3,*2.0*	0.1,*1.0*
	M,L	3.7,*2.0*	–	–	1.1,*2.0*	2156.9,*8.4*	1.0,*1.0*	0.1,*2.0*	0.3,*2.0*	0.4,*2.0*
	L,M	4.8,*2.4*	–	5637.4,*1.8*	1.1,*2.0*	233.0,*9.4*	0.8,*1.0*	0.1,*2.0*	0.3,*2.0*	0.2,*1.4*
	L,L	3.3,*2.0*	–	5807.4,*1.8*	0.8,*1.8*	2589.1,*10.6*	8.4,*2.4*	0.1,*2.0*	0.3,*2.0*	0.4,*2.0*
	Mean	3.8,*2.1*	–	4322.1,*1.5*	1.0,*2.0*	2180.7,*9.4*	2.2,*1.2*	0.1,*2.0*	0.3,*2.0*	0.3,*1.6*
$\hat{R} \in \hat{R}^M$	H,M	5.6,*2.0*	–	12.3,*1.0*	1.1,*2.0*	2942.1,*4.6*	0.8,*1.0*	0.1,*2.0*	0.2,*1.8*	0.1,*1.0*
	H,L	4.4,*2.0*	–	1497.3,*1.0*	1.2,*2.0*	7146.8,*4.0*	1.1,*1.0*	0.1,*2.0*	0.5,*2.0*	0.1,*1.0*
	M,M	4.6,*2.0*	–	43.9,*1.0*	1.1,*2.0*	7098.2,*6.0*	0.9,*1.0*	0.1,*2.0*	0.3,*2.0*	0.1,*1.0*
	M,L	4.2,*2.0*	–	1346.1,*1.0*	1.1,*2.0*	7187.9,*3.2*	1.3,*1.0*	0.1,*2.0*	0.4,*2.0*	0.1,*1.0*
	L,M	3.5,*2.0*	–	17.8,*1.0*	0.9,*2.0*	4840.6,*7.2*	0.8,*1.0*	0.1,*2.0*	0.3,*2.0*	0.1,*1.0*
	L,L	4.2,*2.0*	–	2898.2,*1.2*	0.9,*2.0*	7164.4,*5.0*	1.3,*1.0*	0.1,*2.0*	0.4,*2.0*	0.4,*2.0*
	Mean	4.4,*2.0*	–	969.3,*1.0*	1.0,*2.0*	6063.3,*5.0*	1.0,*1.0*	0.1,*2.0*	0.4,*2.0*	0.2,*1.2*
$\hat{R} \in \hat{R}^L$	H,M	6.6,*2.0*	–	22.0,*1.0*	1.1,*2.0*	2.3,*1.0*	1.1,*1.0*	0.1,*2.0*	0.1,*1.0*	0.1,*1.0*
	H,L	11.7,*2.0*	–	4363.5,*1.0*	1.6,*2.0*	2.7,*1.0*	1.4,*1.0*	0.1,*2.0*	0.2,*1.2*	0.1,*1.0*
	M,M	6.2,*2.0*	7060.2,*1.0*	13.5,*1.0*	0.9,*2.0*	2.3,*1.0*	0.9,*1.0*	0.1,*2.0*	0.1,*1.2*	0.1,*1.0*
	M,L	6.5,*2.0*	–	1464.7,*1.0*	1.4,*2.0*	2.7,*1.0*	1.2,*1.0*	0.1,*2.0*	0.2,*1.4*	0.1,*1.0*
	L,M	4.7,*2.0*	–	21.9,*1.0*	1.3,*2.0*	2.6,*1.0*	0.9,*1.0*	0.1,*2.0*	0.1,*1.0*	0.1,*1.0*
	L,L	7.9,*2.0*	5778.7,*1.0*	2911.4,*1.0*	1.2,*2.0*	2.8,*1.0*	1.3,*1.0*	0.1,*2.0*	0.3,*1.6*	0.1,*1.0*
	Mean	7.3,*2.0*	6939.8,*1.0*	1466.2,*1.0*	1.2,*2.0*	2.6,*1.0*	1.1,*1.0*	0.1,*2.0*	0.2,*1.2*	0.1,*1.0*

"–" indicates that the time limit was reached for these instances before reaching the desired optimality gap.

Table 5.1: Average computational time (in sec.) and number of iterations (given in italic) required to reach convergence for DMP with $\mathcal{T} = 50$ and $\mathcal{C} = 5$ for all datasets where h^0 is restricted.

only 54% for these datasets (while this amount is much higher with 83% when h^0 is restricted, as described above). Unlike DMP, the AP was solved to optimality for a vast number of instances, even for datasets where h^0 is unrestricted. More specifically, the AP was solved to optimality for 97% of the instances where h^0 is unrestricted. However, it is clear from Figure 5.3 that the overall time requirement is much higher for solving the AP for instances where h^0 is unrestricted, in comparison to those where h^0 is restricted.

5.3.2.1 Computational performance of DMP

The time requirements for DMP (where h^0 is restricted) show a considerable difference across different datasets (as shown in Table 5.1). While all the instances with $K \in K^L$ were solved under a second, part of the instances in datasets with higher setup costs (mainly $K \in K^H$) exhaust the time limit before reaching optimality. More specifically, these include instances with medium level of nominal returns ($\bar{R} \in \bar{R}^M$), where all the instances with high and medium level of returns deviation ($\hat{R} \in \hat{R}^H$ and $\hat{R} \in \hat{R}^M$) have failed to reach optimality within the time limit. Similarly, the majority of instances with low returns deviation ($\hat{R} \in \hat{R}^L$) have reached the time limit before obtaining an optimal solution, with the exception of

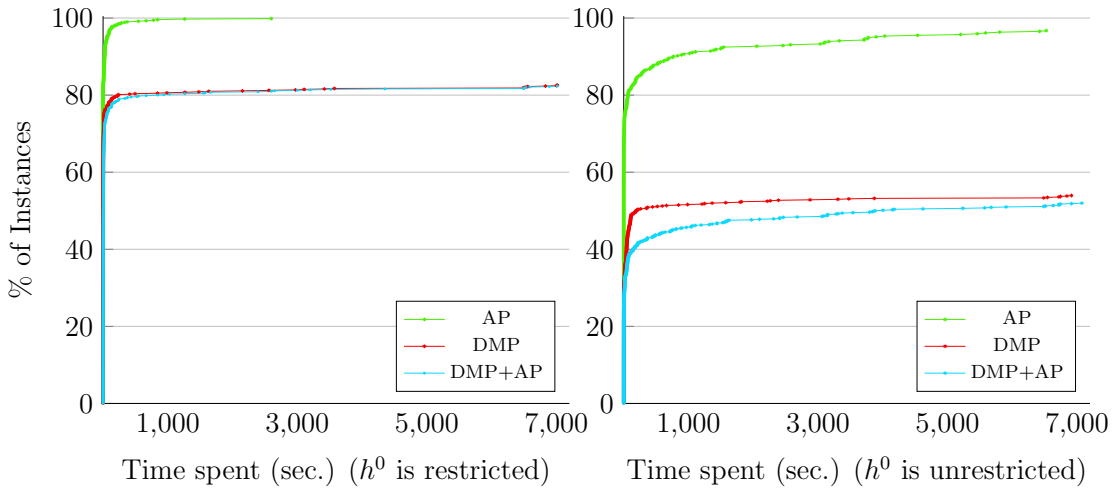

Figure 5.3: Percentage of instances solved to optimality for instances where h^0 is restricted (left) vs. unrestricted (right)

a total of 2 instances out of 30.

In addition, we make the observation that the time requirements vary significantly for instances with medium setup costs and medium nominal return level (where $K \in K^M$ and $\bar{R} \in \bar{R}^M$). This is especially the case for instances where returns deviation level is either set as high or medium ($\hat{R} \in \hat{R}^H, \hat{R}^M$), since the instances with low returns deviation level ($\hat{R} \in \hat{R}^L$) have similar time requirements, where all instances were solved under 5 seconds. The high variance of the time requirements for instances with $K \in K^M$, $\bar{R} \in \bar{R}^M$ and $\hat{R} \in \hat{R}^H, \hat{R}^M$ can be observed from Figure 5.4, where 47% of these instances were not solved to optimality under the time limit, while a total of 11% was solved as quickly as under 100 seconds.

Another interesting aspect to observe here is the impact of our choice of m^0 on the computational performance. For instances where time requirements are considerably higher, low levels of m^0 require a much longer amount of time to reach optimality in comparison to those with medium levels of m^0. Examples to this include datasets with high setup costs with medium and low levels of returns deviation ($K \in K^H$ with $\bar{R} \in \bar{R}^M, \bar{R}^L$), as well as instances with medium levels of setup costs, medium nominal returns, high and medium levels of returns deviation ($K \in K^M$ with $\bar{R} \in \bar{R}^M$ and $\hat{R} \in \hat{R}^H, \hat{R}^M$).

This trend is clear to observe from Figures 5.5 and 5.6, which both demonstrate that a greater number of instances with $m^0 \in m^{0M}$ were solved in comparison to

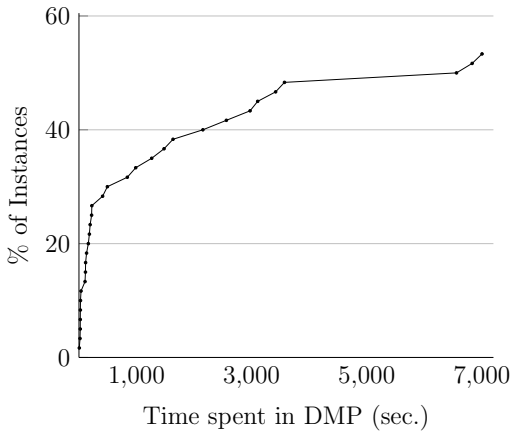

Figure 5.4: Percentage of instances solved to optimality in DMP where $K \in K^M$, $\bar{R} \in \bar{R}^M$ and $\hat{R} \in \hat{R}^H, \hat{R}^M$ under the specified time limit for datasets where h^0 is restricted.

those with $m^0 \in m^{0L}$ in the specified datasets. In the case of $K \in K^H$, while only 51% of the instances with low levels of m^0 were solved under the time limit, this number shows an increase to 65% for instances with medium levels of m^0. This trend holds for instances with $K \in K^M$ as well, where 86% of the instances were solved to optimality under the time limit when $m^0 \in m^{0L}$, which shows a substantial increase to 93% for instances with $m^0 \in m^{0M}$. Likewise, a significant portion of the instances with $m^0 \in m^{0M}$ in $K \in K^H, K^M$ were solved under 100 seconds (with 81%), whereas this amount drops to 75% for instances where $m^0 \in m^{0L}$. Overall, these percentages also demonstrate that a greater number of instances were solved to optimality for medium levels of setups, in comparison to high setup levels.

5.3.2.2 Computational performance of AP

Unlike DMP, the time performances for AP do not vary significantly across different instances for datasets where h^0 is restricted (as shown in Table 5.2). Moreover, the number of instances solved to optimality is also significantly higher for AP, where 97% of the instances were solved as quick as under 100 seconds, and only 1 of the instances was not solved to optimality under the time limit. The results shown in Table 5.2 and Figure 5.3 exclude the time requirement for solving AP for instances where the time limit is reached while solving DMP in the first iteration. Datasets where this is the case for all the instances are marked with only *. Like-

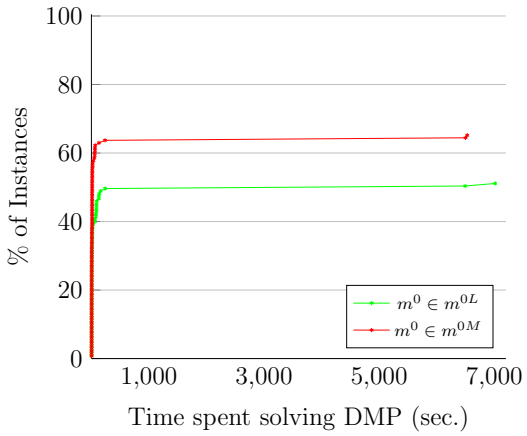

Figure 5.5: % of instances solved to optimality under the given computational time for DMP including instances with $K \in K^H$, where h^0 is restricted.

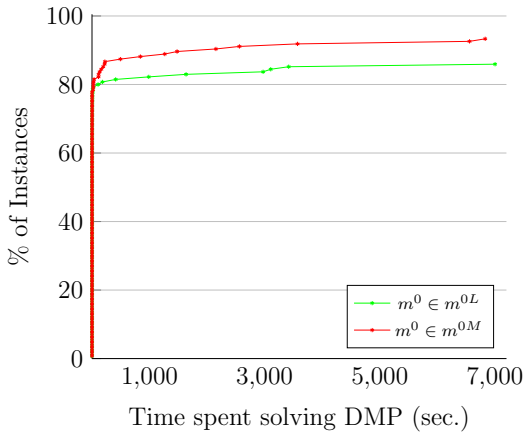

Figure 5.6: % of instances solved to optimality under the given computational time for DMP including instances with $K \in K^M$, where h^0 is restricted.

wise, datasets where the time performance is marked along with * indicate that DMP exhausted the time limit on the first iteration for only part of the instances.

As Table 5.2 suggests, all the instances where setup costs are high were solved to optimality in just a few seconds. Whereas this is the case for a great number of instances with medium and low levels of setup costs, there exists instances that require a relatively longer amount of computational time to reach optimality. More specifically, these include those with medium levels of nominal returns with high and medium return deviations. On the other hand, we do not observe a considerable difference between the computational performance of instances with different levels of manufacturing cost for the end-item ($m^0 \in m^{0M}$ and $m^0 \in m^{0L}$) unlike DMP.

An interesting aspect to examine in AP is regarding the impact of different parameter levels on the UB-LB gap at the root node. As seen in Figure 5.7, we observe different MIP gap percentages at the root node for varying levels of nominal returns, setup costs and returns deviations. Specifically, we observe that when nominal returns are set as low ($\bar{R} \in \bar{R}^L$), the majority of instances have an overall tighter initial MIP gap, where 97% of the instances have a gap percentage below 100%. Although only 35% of the instances with $\bar{R} \in \bar{R}^H$ have an initial MIP gap percentage below 100%, the majority of these instances have a much tighter MIP gap compared to those with $\bar{R}^M$, where only 2% of the instances are below

		K^H			K^M			K^L	
p, m^0	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
H,M	0.3	*	0.0	0.3	185.4	0.0	0.7	81.4	0.0
H,L	0.3	*	0.0*	0.2	84.8	0.0	0.7	25.3	0.0
M,M	0.2	*	0.0	0.2	77.6	0.0	0.8	18.0	0.0
$\hat{R} \in \hat{R}^H$ M,L	0.2	*	0.0*	0.2	37.4	0.0	0.7	19.4	0.0
L,M	0.3	*	0.0	0.2	21.7	0.0	0.8	12.2	0.0
L,L	0.2	*	0.0*	0.2	27.0	0.1	0.8	10.7	0.0
Mean	0.2	*	0.0	0.2	72.3	0.0	0.8	27.8	0.0
H,M	0.1	*	0.0	0.1	64.5	0.0	0.0	1627.9	0.0
H,L	0.1	*	0.0*	0.1	82.0	0.0	0.0	321.4	0.0
M,M	0.1	*	0.0	0.1	54.4	0.0	0.1	52.8	0.0
$\hat{R} \in \hat{R}^M$ M,L	0.1	*	0.0	0.1	57.3	0.0	0.0	856.7	0.0
L,M	0.1	*	0.0	0.1	154.9	0.0	0.0	68.9	0.0
L,L	0.1	*	0.0*	0.1	55.3	0.0	0.1	25.1	0.0
Mean	0.1	*	0.0	0.1	78.1	0.0	0.0	492.1	0.0
H,M	0.0	*	0.0	0.0	0.1	0.0	0.0	0.3	0.0
H,L	0.1	*	0.0*	0.0	0.1	0.0	0.0	0.1	0.0
M,M	0.0	0.1*	0.0	0.0	0.1	0.0	0.0	0.2	0.0
$\hat{R} \in \hat{R}^L$ M,L	0.1	*	0.0	0.0	0.1	0.0	0.0	0.1	0.0
L,M	0.1	*	0.0	0.0	0.1	0.0	0.0	0.1	0.0
L,L	0.1	0.1*	0.0*	0.1	0.1	0.0	0.0	0.3	0.0
Mean	0.1	0.1*	0.0	0.0	0.1	0.0	0.0	0.2	0.0

"*" indicates datasets where DMP has exhaused the time limit in the first iteration (i.e. AP was not solved).

Table 5.2: Average computational time (in sec.) required to reach convergence for AP with $\mathcal{T} = 50$ and $\mathcal{C} = 5$ for all datasets where h^0 is restricted.

100%. In datasets with medium nominal returns, we also observe an overall wider variety across instances. For instance, while some instances have an initial MIP gap percentage value as low as 40%, for others this amount is significantly higher with 1700%. Unlike nominal return levels, we observe that the initial MIP gap percentage for different levels of setup costs and return deviations are distributed more homogeneously across different datasets. Specifically, it is clear from Figure 5.7 that higher levels of setup costs have led to a tighter initial MIP gap percentage in the AP, while higher levels of return deviations have led to a weaker MIP gap.

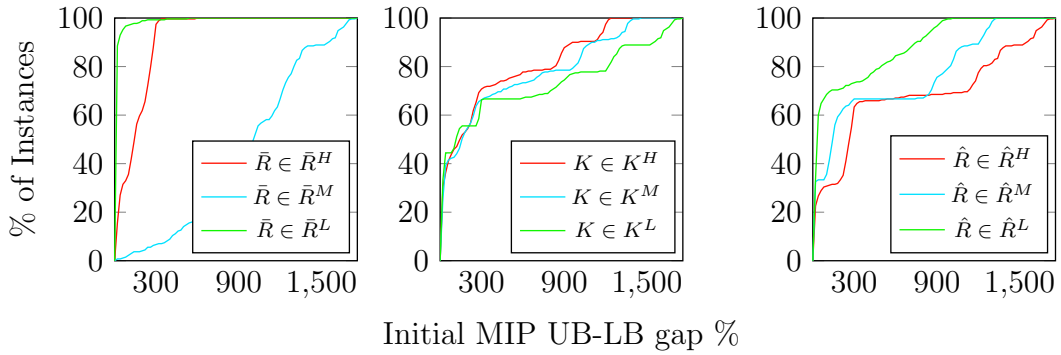

Figure 5.7: Percentage of instances with an initial MIP % value below the specified amount for AP on the first iteration, classified by the level of nominal returns (left), setup cost (middle), and return deviation (right) where h^0 is restricted.

Furthermore, we investigate the datasets with medium and high levels of nominal returns to gain further insight into the specific characteristics of datasets that lead to weaker and tighter MIP gaps for the AP. As seen in Figure 5.8, we observe that lower setup costs lead to an overall weaker initial MIP gap percentage for datasets with $\bar{R} \in \bar{R}^M$. On the other hand, the impact of the setup level on datasets with $\bar{R} \in \bar{R}^H$ is less significant in comparison to those with $\bar{R} \in \bar{R}^M$. A crucial observation we make for both $\bar{R} \in \bar{R}^M$ and $\bar{R} \in \bar{R}^H$ is regarding the impact of returns deviation level. For all levels of setup costs, we observe that higher return deviations lead to a weaker MIP gap percentage. Note that these observations align with our previous analysis of Figure 5.7.

5.3.3 Instances where h^0 is unrestricted

Next, we consider datasets where h^0 is unrestricted. In most practical situations, end-item level inventory cost is most likely to be larger than the sum of

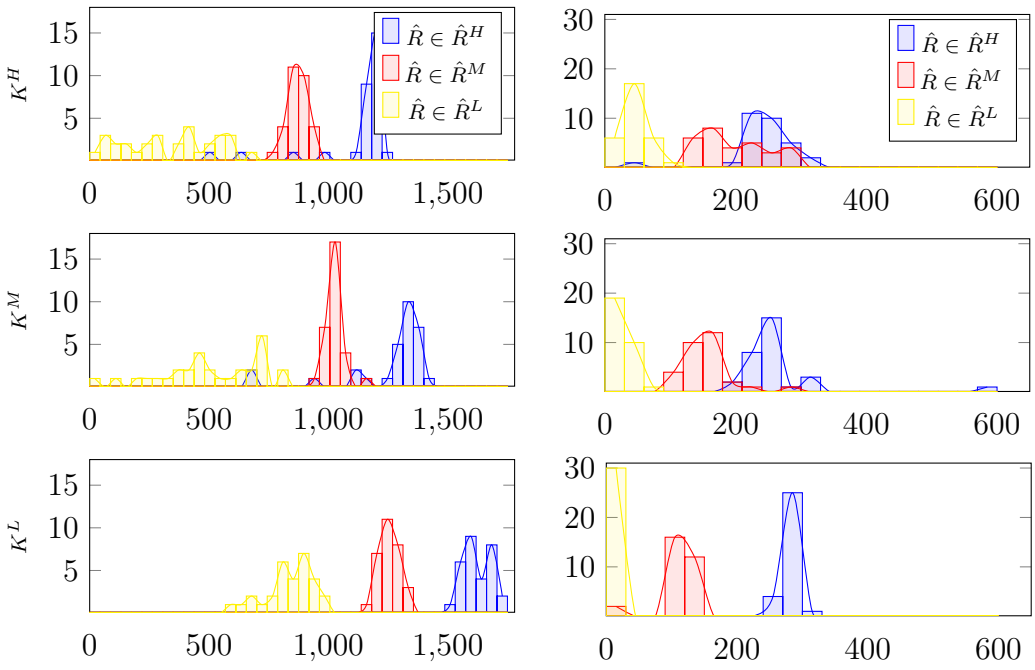

Figure 5.8: Number of instances with $\bar{R} \in \bar{R}^M$ an initial MIP % value within the specified range for AP on the first iteration, sorted by different levels of setup costs and return deviations where h^0 is restricted.

all component level inventory costs, since this item is made up of its components. Consequently, the assumption $h^0 > \sum_{c=1}^{\mathcal{C}} h^c$ is more favorable from a practical point of view. Although the datasets considered under this section do not necessarily satisfy this condition, investigating datasets of this kind is interesting from a computational point of view. This is because our choice of h^0 (as well as how it compares to components' costs) has considerable implications on the overall computational performance, as further discussed under this section.

5.3.3.1 Computational performance of DMP

As previously observed from Figure 5.3, the overall time requirements for solving DMP and AP where h^0 is unrestricted is considerably different from those instances where h^0 is restricted. The first observation we make for cases where h^0 is unrestricted is that the overall time required to solve DMP for datasets with $K \in K^H$ and $K \in K^M$ is significantly higher. Most instances that have exhausted the time limit of 7200 in DMP when h^0 is restricted have exceeded the time limit for cases where h^0 is unrestricted as well, as seen in Table 5.3. Furthermore, we observe that the overall time requirements for datasets with $\bar{R} \in \bar{R}^M, \bar{R}^L$ are significantly higher compared to those with $\bar{R} \in \bar{R}^H$. Contrary to this significant difference, datasets with low levels of setup ($K \in K^L$) are still solved more efficiently, where all instances were solved under 190 seconds.

As previously examined for datasets where h^0 is restricted, the level of m^0 has an impact on the computational performance of DMP. This is the case for instances where h^0 is unrestricted as well as shown in Figures 5.9 and 5.10. We make the observation that a greater number of instances could be solved to optimality under the time limit where $m^0 \in m^{0M}$ than those where $m^0 \in m^{0L}$ in both setup settings. For cases where $K \in K^H$, 14% of the instances with $m^0 \in m^{0L}$ were solved to optimality under 7200 seconds, while this increases to 22% when $m^0 \in m^{0M}$. Similarly, for medium setup levels these amounts show a relative increase to 41% and 48% for $m^0 \in m^{0L}$ and $m^0 \in m^{0M}$, respectively. These amounts are considerably less compared to instances where h^0 is unrestricted, where up to 65% and 93% instances could be solved to optimality for high and medium setup cost settings, respectively. Finally, Figures 5.9 and 5.10 also demonstrate that a greater number of instances with medium setups could be solved to optimality under the time restrictions in comparison to those with higher setup costs (as we may also

		K^H			K^M			K^L		
	p, m^0	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
$\hat{R} \in \hat{R}^H$	H,M	2925.4,*3.6*	–	–	485.1,*9.2*	7198.3,*1.6*	4393.3,*1.0*	40.5,⋆	94.3,⋆	0.2,*1.4*
	H,L	1564.6,*3.6*	–	–	705.4,*7.8*	7198.2,*1.8*	5761.7,*1.0*	10.1,*5.6*	75.9,*16.4*	0.4,*2.0*
	M,M	1488.9,*5.8*	–	–	372.8,*8.2*	7197.2,*1.6*	4324.4,*1.4*	7.2,*5.6*	37.3,*9.2*	0.4,*1.8*
	M,L	693.2,*4.2*	–	–	1629.9,*12.0*	–	4343.7,*1.4*	14.3,*7.8*	94.2,*12.8*	0.4,*2.0*
	L,M	2931.2,*3.6*	–	–	611.5,*9.4*	7198.8,*2.0*	5763.7,*2.2*	22.2,*11.6*	32.3,*9.2*	0.5,*2.0*
	L,L	4363.1,*8.6*	–	–	248.8,*7.0*	7199.0,*1.8*	5763.5,*1.2*	28.6,*12.8*	34.2,*9.2*	0.4,*2.0*
	Mean	2327.7,*4.9*	–	–	675.6,*8.9*	7198.6,*1.8*	5058.4,*1.4*	20.5,*10.6*	61.4,*12.8*	0.4,*1.9*
$\hat{R} \in \hat{R}^M$	H,M	4338.6,*2.2*	–	–	3544.4,*11.2*	7196.9,*1.6*	5760.3,*1.2*	25.5,*12.4*	22.4,*7.0*	0.1,*1.0*
	H,L	5795.2,*2.6*	–	–	1439.3,*4.2*	7199.6,*1.2*	5761,*1.0*	7.4,*5.6*	54.7,*12.4*	0.1,*1.0*
	M,M	3164.6,*2.4*	–	–	2725.8,*9.8*	7198.3,*1.2*	5762.9,*1.4*	0.3,*2.8*	32,*8.4*	0.1,*1.0*
	M,L	4403.0,*4.4*	–	–	2208.4,*9.0*	7198.0,*1.4*	6075.2,*1.0*	26.2,*12.8*	60.1,*15.0*	0.2,*1.4*
	L,M	4346.5,*1.4*	–	–	205.5,*7.6*	7190.7,*1.8*	4324.5,*1.2*	30.7,*16.4*	42.4,*11.2*	0.2,*1.2*
	L,L	5768.2,*6.6*	–	–	1076.2,*11.8*	–	5762.7,*1.0*	24.8,*12.8*	86.2,*16.4*	0.7,*2.0*
	Mean	4636.0,*3.3*	–	–	1866.6,*8.9*	7197.2,*1.4*	5574.4,*1.1*	19.2,*10.5*	49.6,*11.7*	0.2,*1.3*
$\hat{R} \in \hat{R}^L$	H,M	2915.9,*2.0*	–	–	13.5,*2.8*	2764.8,*1.0*	2919.2,*1.0*	1.7,*3.8*	0.1,*1.0*	0.1,*1.0*
	H,L	5773.2,*1.8*	–	–	113.8,*4.0*	5788.6,*1.0*	5763.5,*1.0*	21.7,*7.0*	0.2,*1.2*	0.2,*1.0*
	M,M	2850.3,*2.6*	–	–	30.3,*4.0*	–	1452.4,*1.0*	18.3,*7.6*	0.1,*1.0*	0.1,*1.0*
	M,L	5761.1,*1.2*	–	–	16.2,*3.0*	5764.0,*1.2*	5789.9,*1.0*	8.3,*7.8*	0.1,*1.0*	0.1,*1.0*
	L,M	2815.2,*3.0*	–	5804.1,*1.0*	1435.8,*3.6*	2885.9,*1.0*	7108.9,*1.0*	2.8,*7.0*	0.1,*1.0*	0.1,*1.0*
	L,L	5772.6,*2.4*	–	–	2867.4,*7.2*	5760.6,*1.0*	4325.0,*1.0*	19.4,*11.0*	0.4,*1.8*	0.1,*1.0*
	Mean	4314.7,*2.2*	–	6967.4,*1.0*	746.2,*4.1*	5027.3,*1.0*	4559.8,*1.0*	12.0,*7.4*	0.2,*1.2*	0.1,*1.0*

"–" indicates that the time limit was reached for these instances before reaching the desired optimality gap.
"⋆" indicates that the iteration limit was reached for these instances.

Table 5.3: Average computational time (in sec.) and number of iterations (given in italic) required to reach convergence for DMP with $\mathcal{T} = 50$ and $\mathcal{C} = 5$ for all datasets where h^0 is unrestricted.

observe from the total time requirements for DMP given in Table 5.3). Note that this is also the case for datasets where h^0 is restricted, as examined previously.

Finally, it is important to highlight that the percentages given in Figures 5.9 and 5.10 are calculated according to the number of instances that have been solved to optimality (where the MIP gap is $< 1\%$) in the corresponding datasets, with respect to the total number of instances found in these datasets. For instance, the percentages indicated in red in Figure 5.9 show the percentage of instances solved to optimality for instances that satisfy the conditions $m^0 \in m^{0M}$ and $K \in K^H$. This amount is different from the number of entries observed in Table 5.3, as these indicate the average performance for each dataset (containing five instances), instead of the performance of individual instances.

5.3.3.2 Computational performance of AP

While lower levels of setup costs result in much quicker solution times for DMP, this isn't the case for AP when h^0 is unrestricted. As seen in Table 5.4, a considerable amount of time was spent solving AP for most instances where $K \in K^L$. On the other hand, we observe that all instances with $\bar{R} \in \bar{R}^L$ were solved very quickly, where the maximum amount of time spent on these instances is as low as 0.08

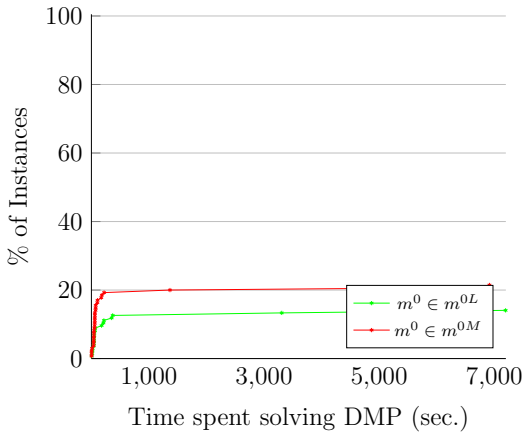

Figure 5.9: % of instances solved to optimality under the given computational time for DMP where $K \in K^H$ and h^0 is unrestricted.

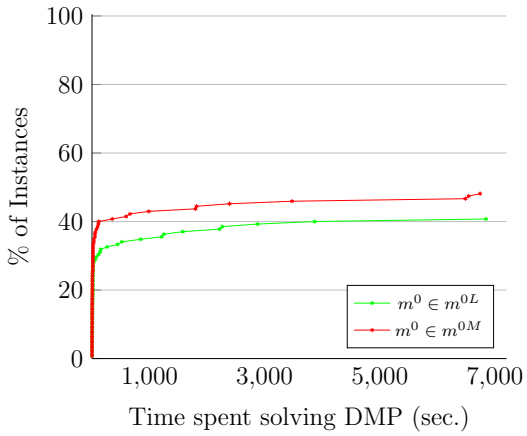

Figure 5.10: % of instances solved to optimality under the given computational time for DMP where $K \in K^M$ and h^0 is unrestricted.

seconds. Another crucial aspect to notice here is that a high number of instances have exhausted the time limit while solving DMP on the first iteration. As a result of this, AP was not solved for a significant number of instances.

Next, we examine the initial MIP percentage values for AP for instances where h^0 is unrestricted. As demonstrated in Figure 5.12, higher levels of return deviations have led to an overall increase in the initial MIP gap percentage values. Furthermore, we observe that lower levels of setups have resulted in a weaker initial MIP gap percentage. As previously examined, this is also the case for datasets where h^0 is restricted (see Figure 5.8).

One of the main differences between datasets where h^0 is restricted and unrestricted is the impact of high nominal return levels on the initial MIP gap percentage for AP. As we have previously examined from Figures 5.7 and 5.8, datasets where $\bar{R} \in \bar{R}^H$ have an overall tighter initial MIP gap in comparison to those with $\bar{R} \in \bar{R}^M$. However, this difference is not as clear for instances where h^0 is unrestricted, since these datasets have a much greater variety in terms of the initial MIP gap percentage values as seen in Figure 5.11). In addition to this, we also make the observation that return deviation levels for these datasets have less impact on the initial MIP gap percentage, as for all three deviation levels the MIP gap percentage values vary greatly. For instance in the case of high setup levels and high nominal return levels (where h^0 is unrestricted), the percentages fall within the intervals $[254.7, 1418.5]\%$, $[112.3, 1086.3]\%$ and $[31.4, 833.2]\%$ for

		K^H			K^M			K^L		
	p, m^0	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$	$\bar{R}^H$	$\bar{R}^M$	$\bar{R}^L$
$\hat{R} \in \hat{R}^H$	H,M	0.9*	*	*	40.9	15.4*	0.0*	234.0	4606.5	0.0
	H,L	0.5*	*	*	24.9	9.5*	0.0*	167.0	3710.8	0.0
	M,M	1.8	*	0.0*	123.2	8.3*	0.0*	36.9	1563.9	0.0
	M,L	0.7	*	*	64.9	*	0.1*	1589.6	263.5	0.0
	L,M	2.0*	*	0.0*	150.2	8.7*	0.1*	2262.9	1426.4	0.0
	L,L	2.1	*	*	83.6	5.7*	0.0*	1003.4	309.6	0.1
	Mean	1.3*	*	0.0*	81.3	9.5*	0.0*	882.3	1980.1	0.0
$\hat{R} \in \hat{R}^M$	H,M	1.1*	*	*	49.9	9.5*	0.0*	483.5	3026.3	0.0
	H,L	0.4*	*	*	5.6	3.8*	0.0*	23.2	4288	0.0
	M,M	0.5*	*	*	75.5	11.5*	0.0*	1.2	2560.1	0.0
	M,L	1.2	*	*	64.4	5.3*	0.0*	255.8	3857.1	0.0
	L,M	0.2*	*	0.0*	12.8	68.0*	0.0*	1110.7	3393.9	0.0
	L,L	1.4*	*	*	96.2	*	0.0*	706.7	2011.5	0.0
	Mean	0.8*	*	0.0*	50.7	19.6*	0.0*	430.2	3189.5	0.0
$\hat{R} \in \hat{R}^L$	H,M	0.2	*	*	0.9	0.1*	0.0*	15.6	0.2	0.0
	H,L	0.3*	*	*	4.7	0.0*	0.0*	766.7	0.4	0.0
	M,M	0.4*	*	*	4.3	*	0.0*	1459.4	0.1	0.0
	M,L	0.2*	*	*	0.6	0.1*	0.0*	115.8	0.1	0.0
	L,M	0.4*	*	0.0*	11.3	0.1*	0.0*	92.2	0.1	0.0
	L,L	0.2*	*	*	29.6	0.1*	0.0*	3960.8	0.2	0.0
	Mean	0.3*	*	0.0*	8.6	0.1*	0.0*	1068.4	0.2	0.0

"*" indicates datasets where DMP has exhausted the time limit in the first iteration (i.e. AP was not solved).

Table 5.4: Average computational time (in sec.) required to reach convergence for AP with $\mathcal{T} = 50$ and $\mathcal{C} = 5$ for all datasets where h^0 is unrestricted.

high, medium and low levels of return deviations, respectively. These intervals are considerably larger from those we obtain from instances where h^0 is restricted, with $[31.9, 327.5]\%$, $[120.5, 286.7]\%$ and $[0, 108.5]\%$, respectively.

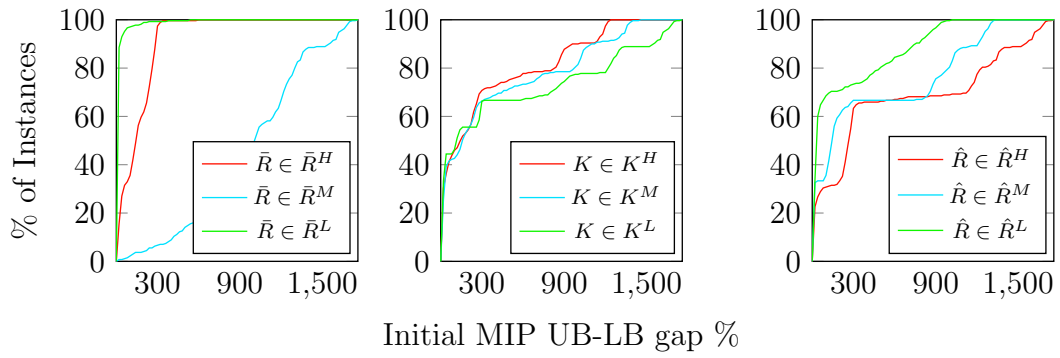

Figure 5.11: Percentage of instances with an initial MIP % value below the specified amount for AP on the first iteration, classified by the level of nominal returns (left), setup cost (middle), and return deviation (right) where h^0 is unrestricted.

5.4 Concluding remarks

In this chapter, we have presented the deterministic and robust formulations for the two-level multi-component lot sizing problem with remanufacturing. Specifically, we assume uncertainties on customer returns only, where demands are assumed to be deterministic. We make the observation that a production plan holds in an optimal solution to the adjustable case of the robust formulation. Upon this observation, the impact of such production rules on the (AP) are investigated, along with a detailed description on how such production rules can be enforced on (AP). Following this, we present thorough computational tests on a wide class of datasets, for varying levels and assumptions on problem parameters. Here, we show that the time requirements for both (DMP) and (AP) vary greatly on the structure of costs, as well as assumptions on the end-item level inventory cost, h^0. In addition to this, an analysis on the influence of various cost and parameter levels on the initial MIP gaps achieved for (AP) is provided.

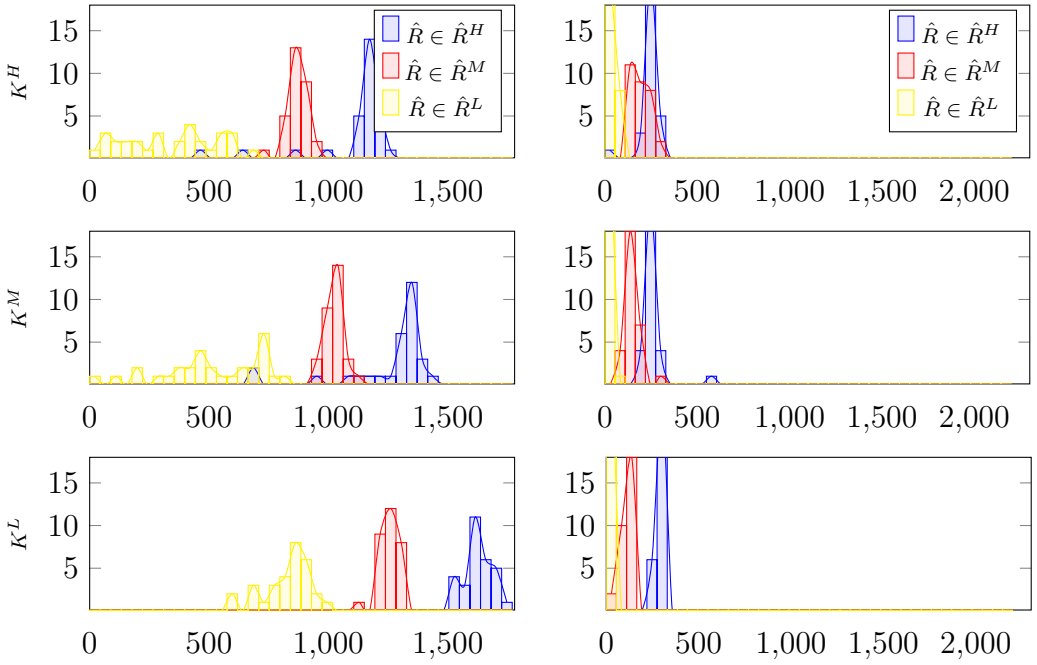

Figure 5.12: Percentage of instances with an initial MIP % value within the specified range for AP on the first iteration, sorted by different levels of setup costs, return deviations, and medium (left) and high (right) nominal return levels where h^0 is unrestricted.

Chapter 6

Deterministic Multiple Components Case

6.1 Introduction

The formulation considered in this section does not imply any uncertainty on demands or returns. Instead, we consider a deterministic setting for the two-level multi-component problem with setups on the end-item level only (i.e. no setup restrictions on the component level). Specifically, we provide certain optimality properties that hold under certain cost assumptions. Following this, we show that these properties can be used in the dynamic programming approach presented in the work of Teunter et al. [2006] to derive solutions. In their work, Teunter et al. [2006] consider a single level setting, whereas in this chapter, we are interested in exploring the two-level multi-component extension of this problem. As further discussed under this chapter, our problem has different assumptions with regards to returns inventory and production (setup, assembly, manufacturing and remanufacturing) decisions.

This setting is particularly relevant for production environments where setups are not required for every single one of the components. Although no fixed costs are considered for the component level, we do consider variable manufacturing and remanufacturing costs for components. We follow our assumption of remanufacturing being less costly than manufacturing (i.e. $r^c < m^c \ \forall c \in \{1, \ldots, \mathcal{C}\}$). In addition to this, we also make the assumption that the sum of holding costs for the components is lower than that of the assembled item (i.e. $h^0 > \sum_{c=1}^{\mathcal{C}} h^c$).

This assumption naturally stems from practice, since assembling components together for a final product adds value to the cumulative value of components, often significantly, and this value dictates holding costs.In a similar fashion, we also assume that holding unprocessed returns is cheaper than holding components (i.e. $w^c < h^c,\ \forall c \in \{1, \ldots, \mathcal{C}\}$). This assumption is also realistic from a practical perspective, since processing (i.e. remanufacturing) a returned component adds value to the component as betterment.

6.2 Problem formulation

Let us begin by introducing the deterministic two-level multi-component lot sizing problem given below.

$$\min \sum_{t=1}^{\mathcal{T}} \Big(K y_t + m^0 x_t^0 + h^0 I_t^0 + \tag{2-MCR-E}$$

$$\sum_{c=1}^{\mathcal{C}} (h^c I_t^c + w^c \sum_{i=1}^{t} (R_i^c - q_i^c) + m^c x_t^c + r^c q_t^c) \Big) \tag{6.1}$$

$$x_t^0 + I_{t-1}^0 = I_t^0 + D_t \qquad \forall t = 1, ..., \mathcal{T} \tag{6.2}$$

$$x_t^c + q_t^c + I_{t-1}^c = I_t^c + x_t^0 \qquad \begin{matrix} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{matrix} \tag{6.3}$$

$$\sum_{i=1}^{t} (R_i^c - q_i^c) \geq 0 \qquad \begin{matrix} \forall t = 1, ..., \mathcal{T} \\ \forall c = 1, ..., \mathcal{C} \end{matrix} \tag{6.4}$$

$$x_t^0 \leq M_t y_t \qquad \forall t = 1, ..., \mathcal{T} \tag{6.5}$$

$$y \in \{0, 1\}^{\mathcal{T}} \tag{6.6}$$

In the formulation above, our objective is to minimize total costs associated with assembly, production, inventory and setup. The main difference between (2-MCR-E) and (2-MCR-C) is that we no longer consider binary setup decision variables for components. Instead, we are now utilizing variables $y := (y_1, y_2, \ldots, y_{\mathcal{T}})$ to define our setup decisions for the end-item level. In this new setting, we are only able to assemble an end-item only if a setup cost of K has been incurred. Due to this modification, we rewrite the setup constraint as (6.5), which ensures that the setup decision is taken according to the assembly levels. Note that the flow

balance constraints for components and the end-item level remain unchanged.

6.3 Problem analysis and resolution

Following this formulation, we now present several optimality properties for the deterministic (2-MCR-E). Note that throughout the next section, we use $p_t^c = \sum_{i=1}^{t}(R_i^c - q_i^c)$ to define the returns inventory level for component c on period t.

6.3.1 Optimality properties

In this section, we present various optimality properties for (2-MCR-E), which are then used to derive the dynamic programming algorithm given in Section 6.3.2. Throughout this section, we make the following assumptions with regards to problem costs:

- End-item holding cost is strictly greater than that of the summation of all components in $\{1, \ldots, \mathcal{C}\}$, where $h^0 > \sum_{c=1}^{\mathcal{C}} h^c$.
- Manufacturing a component is costlier than remanufacturing a returned item, such that $m^c > r^c,\ \forall c = 1, \ldots, \mathcal{C}$.
- Similarly, the cost of holding a component in inventory is strictly greater than that of returns, where $h^c > w^c,\ \forall c = 1, \ldots, \mathcal{C}$.

Furthermore, in advance of defining the specific optimality properties, we provide the following definitions, which play a crucial role for the optimality conditions derived throughout this section.

Definition 6.3.1. Zero inventory property (ZIP) is satisfied in a solution if for each time period t, both the production decision and the inventory decision cannot be strictly positive.

Definition 6.3.2. We define (i, j) as a setup pair, which denotes the setup decision that satisfies the demands in periods $i, ..., j$, where $y_i = 1,\ y_{j+1} = 1$ and $y_t = 0,\ \forall t = i + 1, ..., j$.

Examples for feasible setup patterns for $\mathcal{T} = 10$ include: $\{(1, 3), (4, 7), (8, 10)\}$ (where setup periods are $t = 1, 4, 8$) and $\{(1, 1), (2, 2), (3, 10)\}$ (where setup periods are $t = 1, 2, 3$). Notice that we always have $y_1 = 1$, since backlogging is not allowed.

Following the definitions given above, let us introduce the first optimality property for (2-MCR-E):

Proposition 6. *In an optimal solution, zero inventory property (ZIP) holds for the end-item level, i.e.,* $x_t^0 I_{t-1}^0 = 0$ *when* $h^0 > \sum_{c=1}^{\mathcal{C}} h^c$ *and* $m^c > r^c$.

Proof. Consider the following setup pairs: $\{(t_0, t_1 - 1), (t_1, t_2 - 1), (t_2, t_3)\}$ where $t_0 < t_1 < t_2 < t_3$. Consider a solution where we produce an additional item on t_1 to satisfy the demand on t_2. We will show that an alternative solution with smaller costs exists for the solution where an additional item is assembled on t_1 in order to satisfy the demand on t_3, where ZIP is not satisfied, since $I_{t_2-1}^0 = 1$.

Assembling an additional item on t_1 implies that $x_{t_1}^0 = \sum_{i=t_1}^{t_2-1} D_i + 1$ and $x_{t_2}^0 = \sum_{i=t_2}^{t_3} D_i - 1$. This does not cause any change in the total assembly cost, since the total number of items assembled in both production plans remain the same and costs are time invariant. For this reason, we do not take the total assembly costs into account in the remainder of this proof. Note that for each case presented below, we have an additional cost of $h^0(t_2 - t_1)$ due to carrying an additional item on the end-item level inventory between periods t_1 and t_2.

Regarding the components level, consider the following cases. In order to assemble the additional item on t_1, we may decide to:

- *manufacture a set of additional components,* $A \subseteq \{1, .., \mathcal{C}\}$ *on* t_1*, which would have been remanufactured on* t_2*. Remaining components,* $B = \{1, .., \mathcal{C}\} \backslash A$ *would have been manufactured on* t_2*.*

  Since we are no longer remanufacturing the components in set A, we have a total change of $\sum_{c \in A}(m^c - r^c) > 0$ in production costs. The total production cost for components in set B remains unchanged, since manufacturing a component in t_1, instead of t_2 makes no difference in terms of production costs. Having assembled an additional end-level item, we have an overall positive holding cost of $h^0(t_2 - t_1) > 0$. Finally, components in set A have to be carried in returns inventory (since these returns are no longer remanufactured on t_2), leading to a minimum positive cost change of $\sum_{c \in A} w^c(t_3 - t_2) > 0$

- *remanufacture a set of additional components,* $B \subseteq \{1, .., \mathcal{C}\}$ *on* t_1*, which would have been manufactured on* t_2*. Remaining components,* $A = \{1, .., \mathcal{C}\} \backslash B$ *would have been remanufactured on* t_2*.*

  Remanufacturing components that belong to set B on t_1 causes a reduction

of $-m + r$ in the total production cost due to having remanufactured an additional component on t_1, and having manufactured a component less on t_2. However, for us to be able to remanufacture an additional component on t_1, an additional return has to be carried over until t_1 from an earlier time period. This implies that one less component can be remanufactured at the preceding setup period, t_0 (since this return is now carried over onto t_1). As a result, an additional component needs to be manufactured on t_0, in order to meet the demand on this time period. These changes in the production amounts on t_0 cause an increase of $m - r$ in the total production cost. As a result, the overall production cost remains unchanged, since $-m + r + m - r = 0$. In addition, having assembled an additional item on t_1, we have the extra end-item level inventory holding cost of $h^0(t_2 - t_1) > 0$. Finally, since carrying a return over in order to perform remanufacturing at a later time period implies a positive change in the returns inventory cost (as also shown in Proposition 7), we have the additional returns holding cost of $(t_1 - t_0) \sum_{c \in B} w^c > 0$. On the other hand, remanufacturing the components in set A on t_1 does not cause a change in the overall production costs. Remanufacturing these returns at an earlier time period causes a returns inventory cost reduction of $(t_2 - t_1) \sum_{c \in A} w^c$. However, since an additional item is assembled, we have an additional end-item level inventory cost of $h^0(t_2 - t_1) > 0$. In this case, the overall change in total cost is positive due to condition $h^0 > \sum_{c \in A \cup B = \{1,...,\mathcal{C}\}} w^c$ (since $h^c > w^c,\ \forall c \in \{1, ..., \mathcal{C}\}$), with a total change of $h^0 - \sum_{c \in A} w^c > 0$.

- *manufacture and/or remanufacture a set of additional components on some time period t_0, and use these components to assemble an additional item on some $t \geq t_0$, where this item is not used to satisfy any future demand.*
  Let S^m and S^r be the set of components that have been manufactured and remanufactured on t_0, respectively. Note that $S^m \cap S^r = \emptyset$ (since a component is either manufactured or remanufactured) and $S^m \cup S^r = \{1, ..., \mathcal{C}\}$ (since we require all components to be produced in order to assemble an end-item). First, let us consider the case where $t_0 < t$. Then, having produced a set of additional components in advance of time period t, we have an additional cost of $C = (t - t_0) \sum_{c=1}^{\mathcal{C}} h^c$ (due to holding these components in inventory until t). In addition, we have the additional holding cost of $D = (\mathcal{T} - t + 1)h^0$

until the end of the planning horizon, due to the additional item assembled. Since a set of these components have been remanufactured, we no longer have to carry them in the returns inventory until the end of the time horizon. This reduces the total cost by $A + B$, where $A = (t - t_0) \sum_{c \in S^r} w^c$ (returns inventory cost until t) and $B = (\mathcal{T} - t + 1) \sum_{c \in S^r} w^c$ (returns inventory cost from t, until the end of the planning horizon). Then, we have a total change in costs of $C + D - A - B$. This change is positive since $C - A > 0$, as $(t - t_0)(\sum_{c=1}^{\mathcal{C}} h^c - \sum_{c \in S^r)} w^c > 0$ and $D - B > 0$, as $(\mathcal{T} - t + 1)(h^0 - \sum_{c \in S^r} w^c) > 0$. Finally, when $t = t_0$ we have $A = 0, C = 0$ leading to a total change of $D - B > 0$. Note that in both cases, we incur an additional cost of $m^0 + \sum_{c \in S^m} m^c + \sum_{c \in S^r} r^c$ due to the extra production and assembly decisions.

□

Proposition 7. *Components are remanufactured as early as possible when $m^c > r^c$ holds. In other words, manufacturing components implies that no returns will be carried over onto the next time period, i.e. $x_t^c p_t^c = 0$.*

Proof. Consider two consecutive setup periods, $t_1 > t_2$. We will show that given a production plan and an additional return on period t_1, remanufacturing the additional return on t_1 for components $S \subseteq \{1, ..., \mathcal{C}\}$ is less costly than remanufacturing these on t_2. Consider the case where we carry this return over to t_2. Then, we have to incur an additional cost of $(t_2 - t_1) \sum_{c \in S} w^c$ in order to carry these returns over to period t_2, with a total change in production costs of $\sum_{c \in S}(r^c - m^c)$. Therefore, having this return remanufactured on t_1 implies a reduction of $(t_2 - t_1) \sum_{c \in S} w^c$ in the total cost, where the total change in production costs remain unchanged, since costs are time-invariant. □

Proposition 8. *When $h^c > w^c$, the condition $I_t^c = 0,\ \forall c \in \{1, \ldots, \mathcal{C}\},\ \forall t \in \{1, \ldots, \mathcal{T}\}$ is satisfied in an optimal solution $\forall c \in \{1, .., \mathcal{C}\}$, which implies that $x_t^c = 0$ and $q_t^c = 0$ for time periods where $y_t = 0$. Consequently, we have $x_t^c + q_t^c = x_t^0,\ \forall c \in \{1, ..., \mathcal{C}\}$ for time periods where $y_t = 1$.*

Proof. We will show that an alternative solution with better costs exists for the following cases. Consider the consecutive setup periods: $t_0 > t_1 > t_2$ and a solution where we produce additional components on t_1 to satisfy the end-item demand on t_2. Note that since the end-level assembly and inventory levels are unchanged in

the following scenarios, we will not take the costs on end-item level into account. Let us define set $S \subseteq \{1, ..., \mathcal{C}\}$ as a subset of components.

- *Manufacturing additional components in S in period t_1 in order to satisfy end-level item demand on t_2.*
  Let B indicate the set of items that would have been manufactured on t_2, and set $A = S \backslash B$ indicate those that would have been remanufactured on t_2. Manufacturing the components in set B in t_1 instead of t_2 does not cause any change in the total production costs. In this case, the only change in costs we have is regarding the component level inventory, where $(t_2 - t_1) \sum_{c \in B} h^c > 0$. As for the components in set A, we now have a positive change in total production costs (since we have swapped remanufacturing with manufacturing), where $m^c - r^c > 0$. Similarly, we also have the additional component holding cost of $(t_2 - t_1) \sum_{c \in A} h^c > 0$. Additionally, since we are no longer remanufacturing the items in set A on period t_2, this also causes a minimum increase of $\sum_{c \in A} w^c$ in the total returns inventory holding cost.

- *Remanufacturing additional components S in period t_1 in order to satisfy end-level item demand on t_2.*
  Similarly, let B indicate the set of items that would have been manufactured on t_2, and set $A = S \backslash B$ indicate those that would have been remanufactured on t_2. Then, remanufacturing the items in set B on t_1 causes a reduction of $\sum_{c \in B}(-m^c + r^c)$ in the production costs. However, in order to remanufacture additional components on t_1, an additional set of returns need to be carried over until t_1. This also means that we will need to remanufacture one less item on t_0 (and to manufacture an additional one in order to meet end-item demand), increasing the production costs by $\sum_{c \in B}(m^c - r^c)$. As a result of this, we have no cost change due to production, since $\sum_{c \in B}(-m^c + r^c + m^c - r^c) = 0$. In addition, these returned items have to be held in returns inventory until t_1, causing an additional returns inventory holding cost of $(t_1 - t_0) \sum_{c \in B} w^c$. Having produced an additional component for items in this set, we also have the additional holding cost for the component level, $(t_2 - t_1) \sum_{c \in B} h^c$. Secondly, remanufacturing the components in set A does not cause any change in the total production cost. However, remanufacturing an additional item on t_1 implies a reduction of $(t_2 - t_1) \sum_{c \in A} w^c$ (as these returns no longer have to be carried over). Contrarily, having these returns

remanufactured, we now have an additional cost of $(t_2 - t_1) \sum_{c \in A} h^c$ to incur, leading to an overall increase in costs, since $(t_2 - t_1) \sum_{c \in A} (h^c - w^c) > 0$.

□

6.3.2 Dynamic programming algorithm

Using the properties given in Section 6.3.1, we are able to compute the manufacturing, remanufacturing and inventory levels for different setup patterns. We can then calculate the total cost associated with each one of them to find the optimal production plan. The procedures detailed under this section, as well as the notations for sets and costs follow the dynamic programming framework introduced in the study of Teunter et al. [2006].

Due to Proposition 6, we are able to compute the optimal assembly levels for a given (i, j) as:

$$x^0_{i,j} = \sum_{t=i}^{j} D_t \tag{6.7}$$

Similarly, due to Propositions 7 and 8, we are able to calculate the exact manufacturing and remanufacturing amounts for each one of the components for a given value of $x^0_{i,j}$, where:

$$x^c_i(n^c) = \max\{0, x^0_{i,j} - q^c_i(n^c)\} \tag{6.8}$$

which suggests that demand is satisfied through remanufacturing first (as much as the current returns inventory allows), and any remaining items will be manufactured to satisfy the remaining demand. Then, we may calculate the total number of component c remanufactured as:

$$q^c_i(n^c) = \min\{n^c + R^c_i, x^0_{i,j}\} \tag{6.9}$$

Here, n^c is the number of returns that are carried over onto period i through returns inventory (for component c), and R^c_i is the number of returns that become available at the beginning of period i (for component c). Note that when returns are sufficient to cover the end-level item demand we have $q^c_i = x^0_{i,j}$, which implies $x^c_i = 0$ due to $x^0_{i,j} - q^c_i = 0$ (as also shown in Proposition 7).

Furthermore, let us define $n = (n^1, n^2, ..., n^{\mathcal{C}})$ to indicate the amounts of returns

available (due to returns inventory) for components $\{1, ..., \mathcal{C}\}$. We denote the c^{th} component in n as n^c, which stands for the returns available for a given component c. Then, we may calculate the number of component return inventory available at the end of period j (given a setup pair (i, j) and returns inventory level of n^c available on period $i-1$) as

$$v = (v^1, v^2, \ldots, v^{\mathcal{C}}), \text{ where } v^c = p_i^c(n^c) + \sum_{t=i+1}^{j} R_t^c \tag{6.10}$$

$$\text{and } p_i^c(n^c) = n^c + R_i^c - q_i^c(n^c), \ c \in \{1, \ldots, \mathcal{C}\} \tag{6.11}$$

here n^c stands for the returns available in the return inventory for component c from the previous time period $i-1$, and $\sum_{t=i+1}^{j} R_t^c$ is the number of returns that will be kept in inventory until the next setup period (due to Proposition 8, returns from period $i+1$ until j are not remanufactured). Note that $p_i^c(n^c)$ gives the total number of returns that will be available at the end of period i (in other words, we may describe this amount as the number of remaining returns after remanufacturing is performed on time i). Then, we can represent the complete set of resulting returns (i.e. number of returns held at the end of period j) for (i, j) as:

$$S_{i,j} = \bigcup_{n \in S_{i-1}} \{v = (v^1, v^2, \ldots, v^{\mathcal{C}}) : n = (n^1, n^2, ..., n^{\mathcal{C}})\} \tag{6.12}$$

Here, $S_{i,j}$ contains the set of returns that may be available at the end of period j for the setup pair (i, j), for each $n \in S_{i-1}$.

Furthermore, we define S_j as

$$S_j = \bigcup_{i=1}^{j} S_{i,j} \tag{6.13}$$

S_j contains the set of returns available at the end of period j (which can be derived from the setup pairs (i, j) where $i \leq j$).

Once $S_{i,j}$ is determined, the corresponding production, setup and inventory costs can be calculated as

$$f_{i,j}(v) = f_{i-1}(n) + K + m^0 x_{i,j}^0 + h^0 \sum_{t=i+1}^{j} \sum_{k=t}^{j} D_k + \sum_{c=1}^{\mathcal{C}} (m^c x_i^c(n^c) + r^c q_i^c(n^c)) +$$

$$\sum_{c=1}^{\mathcal{C}} w^c \sum_{t=i}^{j} (p_i^c(n^c) + \sum_{k=i+1}^{t} R_k^c) \quad \forall n \in S_{i-1} \tag{6.14}$$

where

$$f_j(n) = \min_{i=1,...,j} f_{i,j}(n), \ n \in S_{i,j} \tag{6.15}$$

indicates the minimum total cost that can be achieved until time period j, when the set of returns in n are realized.

Note that we define:

$$f_0(n) = 0, \ n \in S_{0,0} = \{0\} \tag{6.16}$$

Since setup has to be carried out for each setup pair (i, j), $f_{i,j}(v)$ includes K (the setup cost). As the total production in period i covers the demands in periods $\{i, ..., j\}$, a total of $h^0 \sum_{t=i+1}^{j} \sum_{k=t}^{j} D_k$ has to be incurred due to the total number of end-item level items held. Furthermore, a component production cost of $\sum_{c=1}^{\mathcal{C}} (m^c x_i^c(n^c) + r^c q_i^c(n^c))$ is incurred on setup period i. Finally, $\sum_{c=1}^{\mathcal{C}} w^c \sum_{t=i}^{j} (p_i^c(n^c) + \sum_{k=i+1}^{t} R_k^c)$ is the total returns holding cost, where $p_i^c(n^c)$ is the returns inventory at the end of period i for a given component c when n is realized in the preceding time period. Note that this amount is always non-negative due to (6.9). Additionally, we may calculate the number of returns held at the end of a given time period t where $i \leq t \leq j$ as: $p_i^c(n^c) + \sum_{k=i+1}^{t} R_k^c$ for component c. In this case, returns from periods $i + 1$ until t will only be kept in inventory, since none of these returns will be remanufactured for the setup pair (i, j) due to Proposition 8.

Algorithm 1 Dynamic programming framework for the lot sizing problem with remanufacturing (see Teunter et al. [2006])

1: Set $i, j := 1$, $S_0 := \{(0, ..., 0)\}$, $S_t := \emptyset \ \forall t = 1, ..., \mathcal{T}$, $f_0(n) = 0, \ n \in S_{0,0} = \{0\}$
2: **while** $j \leq \mathcal{T}$ **do**
3: **for** $i = 1, ..., j$ **do**
4: Compute $S_{i,j}$ and $f_{i,j}(n)$.
5: Compute S_j and $f_j(n)$.
6: $j \leftarrow j + 1$

The output for the algorithm given above are the total costs given as $f_t(n)$, where the optimal production plan can be found by determining the minimum value for $f_t(n)$ where $t = \mathcal{T}$, and $n \in S_{i,j}$.

6.4 Concluding remarks

In this chapter, we provided optimality properties for the two-level multi-component lot sizing problem with remanufacturing under certain assumptions on problem costs, where these are time invariant. Using these properties, we implement the dyanmic programming algorithm given in Teunter et al. [2006]. Furthermore, the dyanmic programming algorithm is derived for the case where the number of returns may be different for each component. This assumption can easily be relaxed, where the number of returns considered for each component is consistent across different components.

Chapter 7

Conclusion and Future Research

In this thesis, we have studied the robust lot sizing problem with remanufacturing. Initially, a general framework for MIP approaches, remanufacturing practices and its implementation in lot sizing problems are given. We have discussed potential variations of such problems, and their implications. In addition to this, we have provided detailed descriptions and examples to how parameter uncertainties affect problem feasibility and optimality. We state that uncertainty plays a crucial role where remanufacturing is considered, since the number of customer returns is often random and mostly unknown.

In Chapter 3, we formally state the robust lot sizing problem with remanufacturing and provide a detailed explanation on how demand and return uncertainty can be incorporated into the deterministic version of this problem. The specific assumptions and structure of the deterministic version of this problem are given. Following this, we state the uncertainty sets for demands and returns, which are defined as budgeted polytopes. An assumption we make here is that the uncertainty sets for demands and returns are defined independently from one another. This implies that the uncertainty set for demands is independent from that of returns. Given such independent uncertainty sets, we further investigate the characteristics of the returns uncertainty sets, and show that the worst-case costs for returns can be computed a priori. Another crucial feature here is that only positive deviations are considered to model the uncertain parameters. As part of this discussion, we show that a given returns uncertainty set with negative deviations can be written as one where only positive deviations are considered. These properties are crucial for the decomposition algorithm presented in Chapter 4, since the structure of uncertainty sets play a crucial role in formulating AP.

Following this, in Chapter 4 a min-max decomposition framework is presented for the robust lot sizing problem with remanufacturing. More specifically, here we present the Decision Maker's Problem (DMP) and the Adversarial Problem (AP), which are solved iteratively to obtain a robust optimal solution. We address some of the computational challenges while solving (DMP), and propose two extended reformulations for this problem, the Extended Aggregated Reformulation (DMP-EFAG) and Approximate Extended Reformulation (DMP-EFAP). In particular, these reformulations are implemented only for the last scenario of the restricted uncertainty set in (DMP). As discussed in detail in the computational experiments section, the extended reformulations are able to improve the computational time requirements for (DMP) significantly. Another interesting observation we make regarding the computational aspects of the decomposition algorithm is that the time requirements for solving (AP) are very low.

Chapter 5 considers a different extension of the robust lot sizing problem with remanufacturing, where two production levels and multiple components are considered. Particularly, we assume that there exists an end-item under deterministic customer demand. In order to produce this item, we make the assumption that a known, finite set of components have to be produced, which are then assembled into the final item. We assume that these components can either be manufactured from scratch or remanufactured from customer returns. The robust assumptions under this setting stem from the uncertainty in returns. Specifically, we analyze the case where excessive customer returns result in additional holding costs, whereas lack of returns causes more items to be manufactured. The latter results in an increase in overall costs, due to our assumption of manufacturing costs being costlier than those of remanufacturing. Under this setting, we provide a production rule that holds for the optimal case, and show its importance while deriving robust solutions. Particularly, we show that optimality conditions or production rules that are implied by the original problem have to be accounted for in the decomposition framework. This is important especially for (AP), since failure to do so results in under-estimation of costs. This observation is not only important for the particular case presented in this chapter, but also for future works in a more general setting (i.e. formulations where robustness is implemented through the min-max decomposition algorithm).

Finally, in Chapter 6 we consider a deterministic two-level multi-component setting. In particular, we relax the setup requirements on components, and inves-

tigate the case where setup is only imposed on the end-item level (i.e. assembly of end-items). We explore certain optimality conditions that hold for this specific case, under particular cost assumptions. These optimality properties are mostly relevant for many lot sizing problems with remanufacturing, where similar cost assumptions are made. In this sense, the optimality properties presented throughout this section provide meaningful insights as to how other variations of this problem may be solved. This does not necessarily have to involve an exact approach (since general classes of lot sizing problems with remanufacturing are known to be $\mathcal{NP}$-hard, as shown in Retel Helmrich et al. [2014]). In such cases, these properties can provide insights as to how to derive strong valid inequalities, or good lower bounds on different variations of this problem.

Although the results gathered from these studies provide meaningful insights for formulating optimal production plans in remanufacturing environments, there are certainly further future research directions to follow, which may provide valuable information about robust lot sizing problems with remanufacturing.

An important assumption that is valid for formulations presented under Chapters 3 and 4 is that customer demand can be indifferently satisfied via manufacturing and remanufacturing. An interesting aspect to consider for future research is to adjust this assumption, such that customer demand is satisfied through new products (i.e. manufacturing) while remanufacturing faces an independent demand. In practical situations, it is common to encounter such settings, thus implementing the min-max approach to this extension is undoubtedly an interesting direction for future research.

In terms of cost assumptions, an interesting direction for future research is to assume different holding costs for manufactured and remanufactured goods, since the handling costs that are associated with remanufacturing can lead to different holding costs. An interesting variation of this is the case where only manufactured items have a holding cost, causing returns to be remanufactured as quickly as possible. Specifically, this is an interesting special case of the problem, since environmentally friendly production plans can be derived using cost assumptions of this nature.

One of the essential assumptions that we are making in Chapters 3 and 4 is that returns uncertainty is only defined for positive deviations. Although this results in a formulation that is consistent and computationally favourable, it is certainly interesting to explore the option where returns deviations are defined for

both negative and positive deviations, such that $-1 \leq z_t^R \leq 1$. Further to this, investigating the specific properties of an uncertainty set of this nature is also certainly crucial to build a formulation of this nature. In this regard, the insights provided on Chapter 5 are certainly useful, as a similar uncertainty set setting is considered for the two-level multi-component lot sizing problem given in this chapter.

Another interesting direction for research here is to investigate the option of dependent uncertainty sets. More specifically, cases such as where demands and returns are dependent on one another, or uncertainty sets with time-dependency of demands and/or returns are among interesting and relevant concepts to consider. Moreover, investigating the implications of such uncertainty sets in a min-max decomposition framework can be an interesting aspect to investigate. In this sense, it would also be worthwhile to examine whether such uncertainty sets can be written in the form of another.

Moreover, our formulations have considered a hybrid system with a joint setup setting. It is certainly interesting to further extend the formulations presented in this thesis to the separate setup case for remanufacturing, where other optimality properties may hold. A general formulation of this nature can be written as:

$$\min \sum_{t=1}^{\mathcal{T}} (K_t^m y_t^m + K_t^r y_t^r + m_t x_t^m + r_t x_t^r + h_t^s I_t^s + h^r I_t^r)$$

$$I_{t-1}^s + x_t^m + x_t^r = D_t + I_t^s \qquad \forall t = 1, \ldots, \mathcal{T} \qquad (7.1)$$

$$I_{t-1}^r + R_t = x_t^r + I_t^r \qquad \forall t = 1, \ldots, \mathcal{T} \qquad (7.2)$$

$$x_t^m \leq M_t y_t^m \qquad \forall t = 1, \ldots, \mathcal{T} \qquad (7.3)$$

$$x_t^r \leq M_t y_t^r \qquad \forall t = 1, \ldots, \mathcal{T} \qquad (7.4)$$

$$x^m, x^r, I^s, I^r \geq 0 \qquad (7.5)$$

$$y^m, y^r \in \{0,1\}^{\mathcal{T}} \qquad (7.6)$$

The formulation above differs considerably from the joint setup structure considered in this thesis. Note that under such setting, the setup costs are defined independently for manufacturing and remanufacturing variables, which may result in specific optimality properties under certain cost structures. As discussed in Chapter 5, such properties are essential while formulating the robust version of this problem, and should not be disregarded. In such formulations, deciding the

cost structure of the separate setups is also crucial, since certain special conditions that provide specific optimality properties may exist in this case as well.

In the formulations presented throughout this thesis, we have considered time invariant costs. It would be an interesting future research direction to relax this assumption, so that costs are time variant. In particular, an interesting aspect to examine here would be to investigate whether any production rules hold under this case, or under special cost structures. Another assumption we make on problem costs is that they are deterministic. This is yet another interesting assumption to relax, where manufacturing/remanufacturing/inventory costs may be defined as uncertain, i.e. as parts of uncertainty sets. This is particularly interesting for the remanufacturing case, since the condition of returned product might have an impact on the cost required to recover the item, which may not be known to certainty in advance.

Another interesting aspect that is specific to remanufacturing systems is the potential dependence between demands and returns. Since returns have to be derived from past demands, it can be argued that the number of returns can be estimated from past sales. From a robust optimization perspective, it may be possible to define returns uncertainty sets as ones that are derived from past demands, by using past data that detail product characteristics such as the product's lifecycle and durability.

Finally, in Chapter 6, an interesting aspect to further investigate is regarding the size of sets $S_{i,j}$, as well as the tractability and complexity of the recursive algorithm. Although this algorithm is presented for the general setting of returns (i.e. where different return quantities are allowed to be considered for each component), this variation of the algorithm may suffer from large set sizes, especially when the number of components considered is large.

Bibliography

Nabil Absi, Stéphane Dauzère-Pérès, Safia Kedad-Sidhoum, Bernard Penz, and Christophe Rapine. Lot sizing with carbon emission constraints. *European Journal of Operational Research*, 227(1):55–61, 2013.

Nabil Absi, Stéphane Dauzère-Pérès, Safia Kedad-Sidhoum, Bernard Penz, and Christophe Rapine. The single-item green lot-sizing problem with fixed carbon emissions. *European Journal of Operational Research*, 248(3):849–855, 2016.

Panayotis Afentakis and Bezalel Gavish. Optimal lot-sizing algorithms for complex product structures. *Operations research*, 34(2):237–249, 1986.

A. Agra and M. Constantino. Lotsizing with backlogging and start-ups: The case of wagner-whitin costs. *Operations Research Letters*, 25(2):81–88, 1999.

Agostinho Agra, Marcio Costa Santos, Dritan Nace, and Michael Poss. A dynamic programming approach for a class of robust optimization problems. *SIAM Journal on Optimization*, 26(3):1799–1823, 2016.

Vishal V Agrawal, Atalay Atasu, and Koert Van Ittersum. Remanufacturing, third-party competition, and consumers' perceived value of new products. *Management Science*, 61(1):60–72, 2015.

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.

Shabbir Ahmed, Ulaş Çakmak, and Alexander Shapiro. Coherent risk measures in inventory problems. *European Journal of Operational Research*, 182(1):226–238, 2007.

Kerem Akartunali and Ashwin Arulselvan. Economic lot-sizing problem with remanufacturing option: complexity and algorithms. In *International Workshop on Machine Learning, Optimization, and Big Data*, pages 132–143. Springer, 2016.

Kerem Akartunalı and Andrew J Miller. A heuristic approach for big bucket multi-level production planning problems. *European Journal of Operational Research*, 193(2):396–411, 2009.

Kerem Akartunalı and Andrew J Miller. A computational analysis of lower bounds for big bucket production planning problems. *Computational Optimization and Applications*, 53(3):729–753, 2012.

Ayse Akbalik and Yves Pochet. Valid inequalities for the single-item capacitated lot sizing problem with step-wise costs. *European Journal of Operational Research*, 198(2):412–434, 2009.

Ayse Akbalik and Christophe Rapine. Single-item lot sizing problem with carbon emission under the cap-and-trade policy. In *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 030–035. IEEE, 2014.

Mohamed Ali Aloulou, Alexandre Dolgui, and Mikhail Y Kovalyov. A bibliography of non-deterministic lot-sizing models. *International Journal of Production Research*, 52(8):2293–2310, 2014.

Shoshana Anily, Michal Tzur, and Laurence A Wolsey. Multi-item lot-sizing with joint set-up costs. *Mathematical programming*, 119(1):79–94, 2009.

Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

Alper Atamtürk and Muhong Zhang. Two-stage robust network flow and design under demand uncertainty. *Operations Research*, 55(4):662–673, 2007.

Öykü Naz Attila, Agostinho Agra, Kerem Akartunalı, and Ashwin Arulselvan. A decomposition algorithm for robust lot sizing problem with remanufacturing option. In *International Conference on Computational Science and Its Applications*, pages 684–695. Springer, 2017.

M Fazle Baki, Ben A Chaouch, and Walid Abdul-Kader. A heuristic solution procedure for the dynamic lot sizing problem with remanufacturing and product recovery. *Computers & Operations Research*, 43:225 – 236, 2014.

Chaithanya Bandi and Dimitris Bertsimas. Tractable stochastic analysis in high dimensions via robust optimization. *Mathematical programming*, 134(1):23–70, 2012.

Imre Barany, Tony J Van Roy, and Laurence A Wolsey. Strong formulations for multi-item capacitated lot sizing. *Management Science*, 30(10):1255–1261, 1984.

Aharon Ben-Tal and Arkadi Nemirovski. Robust convex optimization. *Mathematics of operations research*, 23(4):769–805, 1998.

Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of uncertain linear programs. *Operations research letters*, 25(1):1–13, 1999.

Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical programming*, 88(3): 411–424, 2000.

Aharon Ben-Tal and Arkadi Nemirovski. Robust optimization — methodology and applications. *Mathematical Programming*, 92:453—-480, 2002.

Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer, and Arkadi Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2004.

Aharon Ben-Tal, Boaz Golany, Arkadi Nemirovski, and Jean Philippe Vial. Retailer-supplier flexible commitments contracts: A robust optimization approach. *Manufacturing & Service Operations Management*, 7(3):248–271, 2005.

Jacques F Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

Dimitris Bertsimas and David B Brown. Constructing uncertainty sets for robust linear optimization. *Operations research*, 57(6):1483–1495, 2009.

Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1-3):49–71, 2003.

Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

Dimitris Bertsimas and Aurélie Thiele. A robust optimization approach to inventory theory. *Operations Research*, 54(1):150–168, 2006.

Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.

Dimitris Bertsimas, Iain Dunning, and Miles Lubin. Reformulation versus cutting-planes for robust optimization. *Computational Management Science*, 13(2):195–217, 2016.

Daniel Bienstock and Nuri Özbay. Computing robust basestock levels. *Discrete Optimization*, 5(2):389–414, 2008.

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

N. Brahimi, N. Absi, S. Dauzère-Pérès, and A. Nordli. Single-item dynamic lot-sizing problems: An updated survey. *European Journal of Operational Research*, 263(3):838–863, 2017.

Nadjib Brahimi, Stéphane Dauzère-Pérès, and Najib M Najid. Capacitated multi-item lot-sizing problems with time windows. *Operations Research*, 54(5):951–967, 2006.

Christina Büsing and Fabio D'Andreagiovanni. New results about multi-band uncertainty in robust optimization. In *International Symposium on Experimental Algorithms*, pages 63–74. Springer, 2012.

Christina Büsing, Fabio D'Andreagiovanni, and Annie Raymond. 0–1 multiband robust optimization. In *Operations Research Proceedings 2013*, pages 89–95. Springer, 2014.

Abraham Charnes, William W Cooper, and Gifford H Symonds. Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. *Management science*, 4(3):235–263, 1958.

Xin Chen and Yuhan Zhang. Uncertain linear programs: Extended affinely adjustable robust counterparts. *Operations Research*, 57(6):1469–1482, 2009.

Andrew J Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management Science*, 6(4):475–490, 1960.

George B Dantzig. Application of the simplex method to a transportation problem. *Activity Analysis and Production and Allocation*, 1951.

Marisa P De Brito, Rommert Dekker, and Simme Douwe P Flapper. Reverse logistics: a review of case studies. In *Distribution Logistics*, pages 243–281. Springer, 2005.

Moustapha Diaby, Harish C Bahl, Mark H Karwan, and Stanley Zionts. A lagrangean relaxation approach for very-large-scale capacitated lot-sizing. *Management Science*, 38(9):1329–1340, 1992.

Mahdi Doostmohammadi and Kerem Akartunalı. Valid inequalities for two-period relaxations of big-bucket lot-sizing problems: Zero setup case. *European Journal of Operational Research*, 267(1):86 – 95, 2018.

Laurent El Ghaoui and Hervé Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal on matrix analysis and applications*, 18(4): 1035–1064, 1997.

Laurent El Ghaoui, Francois Oustry, and Hervé Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization*, 9(1):33–52, 1998.

Gary D Eppen and R Kipp Martin. Solving multi-item capacitated lot-sizing problems using variable redefinition. *Operations Research*, 35(6):832–848, 1987.

James E Falk. Exact solutions of inexact linear programs. *Operations Research*, 24(4):783–787, 1976.

Matteo Fischetti and Michele Monaci. Light robustness. In *Robust and online large-scale optimization*, pages 61–84. Springer, 2009.

Matteo Fischetti and Michele Monaci. Cutting plane versus compact formulations for uncertain (integer) linear programs. *Mathematical Programming Computation*, 4(3):239–273, 2012.

Moritz Fleischmann, Jacqueline M Bloemhof-Ruwaard, Rommert Dekker, Erwin Van der Laan, Jo AEE Van Nunen, and Luk N Van Wassenhove. Quantitative models for reverse logistics: A review. *European Journal of Operational Research*, 103(1):1–17, 1997.

Michael Florian, Jan Karel Lenstra, and AHG Rinnooy Kan. Deterministic production planning: Algorithms and complexity. *Management science*, 26(7):669–679, 1980.

Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *European journal of operational research*, 235(3): 471–483, 2014.

Arthur M Geoffrion. Generalized benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260, 1972.

Joel Goh and Melvyn Sim. Distributionally robust optimization and its tractable approximations. *Operations research*, 58(4-part-1):902–917, 2010.

Boaz Golany, Jian Yang, and Gang Yu. Economic lot-sizing with remanufacturing options. *IIE Transactions*, 33(11):995–1004, 2001.

Ralph Gomory. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

Ralph E Gomory et al. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278, 1958.

Bram L Gorissen, İhsan Yanıkoğlu, and Dick den Hertog. A practical guide to robust optimization. *Omega*, 53:124 – 137, 2015.

Yongpei Guan and Andrew J Miller. Polynomial-time algorithms for stochastic uncapacitated lot-sizing problems. *Operations Research*, 56:1172–1183, 2008a.

Yongpei Guan and Andrew J Miller. Polynomial-time algorithms for stochastic uncapacitated lot-sizing problems. *Operations Research*, 56(5):1172–1183, 2008b.

Yongpei Guan, Shabbir Ahmed, George L Nemhauser, and Andrew J Miller. A branch-and-cut algorithm for the stochastic uncapacitated lot-sizing problem. *Mathematical Programming*, 105(1):55–84, 2006.

V Daniel R Guide Jr and Luk N Van Wassenhove. Managing product returns for remanufacturing. *Production and operations management*, 10(2):142–155, 2001.

V Daniel R Guide Jr and Luk N Van Wassenhove. The evolution of closed-loop supply chain research. *Operations research*, 57(1):10–18, 2009.

Nir Halman, Diego Klabjan, Mohamed Mostagir, Jim Orlin, and David Simchi-Levi. A fully polynomial-time approximation scheme for single-item stochastic inventory control with discrete demand. *Mathematics of Operations Research*, 34(3):674–685, 2009.

Ford W Harris. How many parts to make at once. 1913.

Mathijn J Retel Helmrich, Raf Jans, Wilco van den Heuvel, and Albert PM Wagelmans. The economic lot-sizing problem with an emission capacity constraint. *European Journal of Operational Research*, 241(1):50–62, 2015.

Timo Hilger, Florian Sahling, and Horst Tempelmeier. Capacitated dynamic production and remanufacturing planning under demand and return uncertainty. *OR spectrum*, 38(4):849–876, 2016.

Winifred L Ijomah. Addressing decision making for remanufacturing operations and design-for-remanufacture. *International Journal of Sustainable Engineering*, 2(2):91–102, 2009.

F. Robert Jacobs and F.C. Weston. Enterprise resource planning (ERP) - A brief history. *Journal of Operations Management*, 25(2):357–363, 2007.

R. Jans and Z. Degraeve. Improved lower bounds for the capacitated lot sizing problem with setup times. *Operations Research Letters*, 32:185–195, 2004.

Raf Jans and Zeger Degraeve. Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches. *European journal of operational research*, 177 (3):1855–1875, 2007.

Raf Jans and Zeger Degraeve. Modeling industrial lot sizing problems: a review. *International Journal of Production Research*, 46(6):1619–1643, 2008.

Vaidyanathan Jayaraman, V Daniel R Guide Jr, and Rajesh Srivastava. A closed-loop logistics model for remanufacturing. *Journal of the operational research society*, 50(5):497–508, 1999.

Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

Uday S Karmarkar and Linus Schrage. The deterministic dynamic product cycling problem. *Operations Research*, 33(2):326–345, 1985.

Onur A Kilic and Wilco van den Heuvel. Economic lot sizing with remanufacturing: structural properties and polynomial-time heuristics. *IISE Transactions*, pages 1–14, 2019. doi: 10.1080/24725854.2019.1593555.

Ömer Kirca and Melih Kökten. A new heuristic approach for the multi-item dynamic lot sizing problem. *European Journal of Operational Research*, 75(2): 332–341, 1994.

Arie MCA Koster, Manuel Kutschka, and Christian Raack. Robust network design: Formulations, valid inequalities, and computations. *Networks*, 61(2):128–149, 2013.

Jakob Krarup and Ole Bilde. Plant location, set covering and economic lot size: An 0 (mn)-algorithm for structured problems. In *Numerische Methoden bei Optimierungsaufgaben Band 3*, pages 155–180. Springer, 1977.

Simge Küçükyavuz and Yves Pochet. Uncapacitated lot sizing with backlogging: the convex hull. *Mathematical Programming*, 118(1):151–175, 2009.

AH Land and AG Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

Jianzhi Li, Miguel González, and Yun Zhu. A hybrid simulation optimization method for production planning of dedicated remanufacturing. *International Journal of Production Economics*, 117(2):286–301, 2009.

Liwan H Liyanage and J George Shanthikumar. A practical inventory control policy using operational statistics. *Operations Research Letters*, 33(4):341–348, 2005.

Pedro Belluco Macedo, Douglas Alem, Maristela Santos, Muris Lage Junior, and Alfredo Moreno. Hybrid manufacturing and remanufacturing lot-sizing problem with stochastic demand, return, and setup costs. *The International Journal of Advanced Manufacturing Technology*, 82(5-8):1241–1257, 2016.

Josefa Mula, David Peidro, Manuel Díaz-Madroñero, and Eduardo Vicens. Mathematical programming models for supply chain production and transport planning. *European Journal of Operational Research*, 204(3):377–390, 2010.

Almir Mutapcic and Stephen Boyd. Cutting-set methods for robust convex optimization with pessimizing oracles. *Optimization Methods & Software*, 24(3): 381–406, 2009.

Zhendong Pan, Jiafu Tang, and Ou Liu. Capacitated dynamic lot sizing problems in closed-loop supply chain. *European Journal of Operational Research*, 198(3): 810–821, 2009.

Pedro Pineyro and Omar Viera. The economic lot-sizing problem with remanufacturing and one-way substitution. *International Journal of Production Economics*, 124:482–488, 2010.

George W Plossl and Joseph Orlicky. *Orlicky's material requirements planning*. McGraw-Hill Professional, 1994.

Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming*. Springer, 2006.

Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017.

Mathijn J Retel Helmrich, Raf Jans, Wilco van den Heuvel, and Albert PM Wagelmans. Economic lot-sizing with remanufacturing: complexity and efficient formulations. *IIE Transactions*, 46(1):67–86, 2014.

Knut Richter and Mirko Sombrutzki. Remanufacturing planning for the reverse wagner/whitin models. *European Journal of Operational Research*, 121(2):304–315, 2000.

Knut Richter and Jens Weber. The reverse wagner/whitin model with variable manufacturing and remanufacturing cost. *International Journal of Production Economics*, 71(1-3):447–456, 2001.

Dale S Rogers and Ronald Tibben-Lembke. An examination of reverse logistics practices. *Journal of Business Logistics*, 22(2):129–148, 2001.

Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

Chuen-Teck See and Melvyn Sim. Robust approximation to multiperiod inventory management. *Operations research*, 58(3):583–594, 2010.

A. Sifaleras and I. Konstantaras. Variable neighborhood descent heuristic for solving reverse logistics multi-item dynamic lot-sizing problems. *Computers & Operations Research*, 78:385 – 392, 2017.

C Singh. Convex programming with set-inclusive constraints and its applications to generalized linear and fractional programming. *Journal of Optimization Theory and Applications*, 38(1):33–42, 1982.

Allen L Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations research*, 21(5):1154–1157, 1973.

H. Stadtler. Multilevel lot sizing with setup times and multiple constrained resources: Internally rolling schedules with lot-sizing windows. *Operations Research*, 51(3):487–502, 2003.

Sharifah Aishah Syed Ali, Mahdi Doostmohammadi, Kerem Akartunalı, and Robert van der Meer. A theoretical and computational analysis of lot-sizing in remanufacturing with separate setups. *International Journal of Production Economics*, 203:276 – 285, 2018.

Ruud H Teunter, Z Pelin Bayindir, and Wilco Van Den Heuvel. Dynamic lot sizing with product returns and remanufacturing. *International Journal of Production Research*, 44(20):4377–4400, 2006.

Aurélie Thiele, Tara Terry, and Marina Epelman. Robust linear optimization with recourse. *Technical Report*, pages 4–37, 2009.

Martijn Thierry, Marc Salomon, Jo Van Nunen, and Luk Van Wassenhove. Strategic issues in product recovery management. *California Management Review*, 37 (2):114–136, 1995.

CPM Van Hoesel, Albert Peter Marie Wagelmans, and Laurence A Wolsey. Polyhedral characterization of the economic lot-sizing problem with start-up costs. *SIAM Journal on Discrete Mathematics*, 7(1):141–151, 1994.

Mathieu Van Vyve and Laurence A. Wolsey. Approximate extended formulations. *Mathematical Programming*, 105(2-3):501–522, 2006.

Mathieu Van Vyve, Laurence A Wolsey, and Hande Yaman. Relaxations for two-level multi-item lot-sizing problems. *Mathematical Programming*, 146(1-2):495–523, 2014.

Dimitrios Vlachos and Rommert Dekker. Return handling options and order quantities for single period products. *European Journal of Operational Research*, 151 (1):38–52, 2003.

Harvey M Wagner and Thomson M Whitin. Dynamic version of the economic lot size model. *Management Science*, 5(1):89–96, 1958.

Ben Walsh, Rachel Waugh, and Harry Symington. Remanufacturing study -— circular economy evidence building programme, summary report. Technical report, Technical Report, Zero Waste Scotland, Stirling, UK, 2015.

C. Wei, Y. Li, and X. Cai. Robust optimal policies of production and inventory with uncertain returns and demand. *International Journal of Production Economics*, 134(2):357—-367, 2011.

Wolfram Wiesemann, Daniel Kuhn, and Melvyn Sim. Distributionally robust convex optimization. *Operations Research*, 62(6):1358–1376, 2014.

Laurence A Wolsey. *Integer programming*. Wiley, 1998.

Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

Tao Wu, Leyuan Shi, Joseph Geunes, and Kerem Akartunalı. An optimization framework for solving capacitated multi-level lot-sizing problems with backlogging. *European Journal of Operational Research*, 214(2):428–441, 2011.

Tao Wu, Zhe Liang, and Canrong Zhang. Analytics branching and selection for the capacitated multi-item lot sizing problem with nonidentical machines. *INFORMS Journal on Computing*, 30(2):236–258, 2018.

Jian Yang, Boaz Golany, and Gang Yu. A concave-cost production planning problem with remanufacturing options. *Naval Research Logistics*, 52(5):443–458, 2005.

Willard I Zangwill. A deterministic multi-period production scheduling model with backlogging. *Management Science*, 13(1):105–119, 1966.

Willard I Zangwill. A backlogging model and a multi-echelon model of a dynamic economic lot size production system — A network approach. *Management Science*, 15(9):506–527, 1969.

B. Zeng and L. Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5): 457–461, 2013.

Minjiao Zhang, Simge Küçükyavuz, and Hande Yaman. A polyhedral study of multiechelon lot sizing with intermediate demands. *Operations Research*, 60(4): 918–935, 2012.

Appendix A

Codes

A.1 Decomposition Algorithm (LSR-R)

```
import java.io.*;
import java.lang.management.ManagementFactory;
import java.lang.management.MemoryUsage;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.*;
import ilog.concert.IloException;
import ilog.concert.IloIntVar;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.concert.IloNumVarType;
import ilog.concert.IloRange;
import ilog.cplex.IloCplex;

public class UNCAP_Original_SingScnrioBnd {
//Choice of dataset
public static String ReturnChoice="High";
public static String GammaChoice="High";
public static String SetupChoice="High";
public static String DisposalChoice="Greater";

public static String[] changeReturnChoice = {"","High","Med","Low"};
public static String[] changeGammaChoice = {"","High","Med","Low"};
```

```
public static String[] changeSetupChoice = {"","VHigh","High","Med","Low"};
public static String[] changeDisposalChoice = {"","Greater","Less"};

public static boolean breakit=false;
public static int flowcover=0;
public static int gomory=0;
public static int covercuts=0;
public static int MIRcuts=0;
public static int sltnindex=0;
public static double RootNodeObj_Original=0;
public static int finalnodesize=0;
public static double prev_best_obj=0;
public static double prev_gap=0;
public static double prev_optimal=0;
public static double timelim=10000;
public static String dmstatus;
public static String advstatus;
public static double initialmipgap=0.2;
public static double k_backlog=10;
public static double mipgap=initialmipgap;
public static double epsilon=0.01; //Defining convergence
public static int maxiteration=20; //Max. number of iterations in each run
public static int initial_servicables_inv=0;
public static int initial_returns_inv=0;
public static int t=50;
public static double K=0; //Setup cost
public static double m=0; //Manufacturing cost per item
public static double r=0; //Remanufacturing cost per item
//Defined as an array to test the impact of changing backlogging cost in the last time
    period
public static double[] backlog_cost = new double [t+1];
public static double dis=0; //Disposal cost per item
public static double totaladvtime=0;
public static double totaldmtime=0;
public static double avgadvtime=0;
public static double avgdmtime=0;
public static double avgiteration=0;
```

```
public static double prodcosts=0;
public static double dm_local_objective=0;
public static double dmtime_ineachrun=0;
public static double advtime_ineachrun=0;
public static double worstrettime_ineachrun=0;
public static double singlesctime_ineachrun=0;
public static double chs = 0; //Servicables' holding cost
public static double chr = 0; //Returns' holding cost
public static double[] bigM_dm_setup= new double [t+1];
public static double[] bigM1_adv = new double [t+1];
public static double[] bigM2_adv = new double [t+1];
public static double rhspi_adv=0;
public static double rhspi_dm=0;
public static double[] dbar = new double [t+1];
public static double[] rbar = new double [t+1];
public static double[] dcap = new double [t+1];
public static double[] rcap = new double [t+1];
public static double[] gammaD = new double [t+1];
public static double[] gammaR = new double [t+1];
public static double[][] zd_set = new double [t+1][maxiteration+1];
public static double[] zr_worst = new double [t+1];
public static double[] singlescenario_optimal = new double [maxiteration+1];
public static double singlesc_maximum=0;
public static double[] xm_set = new double [t+1];
public static double[] xr_set = new double [t+1];
public static double[] d_set = new double [t+1];
public static double[] b_set = new double [t+1];
public static double[] theta_set = new double [t+1];
public static double ADVobjective;
public static double DMobjective;
public static int numberofiterations=0;
public static double optimal_pi_DM=0;
public static double optimal_pi_ADV=0;
public static double gap=0;
public static double finalmipgap=0;
public static double ADVub=0;
public static double DMlb=0;
```

```
public static int addtocases=0;
public static int runmanytimes=0;
public static long dmtimestart=0;
public static long dmtimeend=0;
public static long advtimestart=0;
public static long advtimeend=0;
public static long worstrettimestart=0;
public static long worstrettimeend=0;
public static long singlesctimestart=0;
public static long singlesctimeend=0;

public static int breaknow=0;

public static void main(String[] args) throws Exception, Exception {

float countconvergence=0;
float countnonconvergence=0;

File sltns_rootnode = new
    File("Kblog_"+k_backlog+"UNCAP_Org_SinSc_RootNode_Sltns_"+t+".txt");

//Run for many instances
for (int runmnyret=1; runmnyret<=3; runmnyret++) {
   //Change returns dataset
   ReturnChoice=changeReturnChoice[runmnyret];
   for (int runmnygam=1; runmnygam<=3; runmnygam++) {
      //Change gamma dataset
      GammaChoice=changeGammaChoice[runmnygam];
      for (int runmnyset=1; runmnyset<=1; runmnyset++) {
         //Change setup cost dataset
         SetupChoice=changeSetupChoice[runmnyset];
         for (int runmnydisp=1; runmnydisp<=2; runmnydisp++) {
         DisposalChoice=changeDisposalChoice[runmnydisp];
            for (runmanytimes=1; runmanytimes<=5; runmanytimes++) {
               singlesctime_ineachrun=0;
               worstrettime_ineachrun=0;
               dmtime_ineachrun=0;
```

```
advtime_ineachrun=0;
ADVub=0;
DMlb=0;
addtocases=0;
File f = new File("C:\\Users\\npb15184\\Desktop\\LSRData\\"+ReturnChoice
    +GammaChoice +SetupChoice +DisposalChoice +"\\DataR" +ReturnChoice
    +"Gamma" +GammaChoice +"K" +SetupChoice +"Disposal" +DisposalChoice
    +runmanytimes +".txt");
BufferedReader freader = new BufferedReader(new FileReader(f));
String s;
int myt=1;
while ((s = freader.readLine()) != null && myt<=t) {
   String[] st = s.split(" ");
   ArrayList<String> results_without_blanks = new ArrayList<String>();
   for (String current_string : st) {
      if (current_string != null && !current_string.isEmpty()) {
         results_without_blanks.add(current_string);
      }
   }
   if (results_without_blanks.size()>0) {
      if (results_without_blanks.get(0).startsWith(""+myt+"")){
         dbar[myt]=Math.ceil(Double.parseDouble(results_without_blanks.get(1)));
         rbar[myt]=Math.ceil(Double.parseDouble(results_without_blanks.get(2)));
         dcap[myt]=Math.ceil(Double.parseDouble(results_without_blanks.get(3)));
         rcap[myt]=Math.ceil(Double.parseDouble(results_without_blanks.get(4)));
         K=Double.parseDouble(results_without_blanks.get(5));
         chs=Double.parseDouble(results_without_blanks.get(6));
         chr=Double.parseDouble(results_without_blanks.get(7));
         for (int time=1; time<=t; time++) {
            backlog_cost[time]=Double.parseDouble(results_without_blanks.get(8));}
         gammaD[myt]=Double.parseDouble(results_without_blanks.get(9));
         gammaR[myt]=Double.parseDouble(results_without_blanks.get(10));
         m=Double.parseDouble(results_without_blanks.get(11));
         r=Double.parseDouble(results_without_blanks.get(12));
         dis=Double.parseDouble(results_without_blanks.get(13));
         myt++;
      }
```

```
   }
}
   backlog_cost[t]=k_backlog*backlog_cost[1];

     long startTime=0;
for (int mm=1; mm<=maxiteration; mm++) {
   if (mm==1) {
      numberofiterations=0;
      zd_set = new double [t+1][maxiteration+1];
      startTime = System.currentTimeMillis();
   }
   numberofiterations++;
   if (numberofiterations==1) { //Only run this once.
      zr_worst = new double [t+1];
      returns_precompute();
      mipgap=initialmipgap; //Reset mipgap to initial mipgap
   }
   facloc_singlescenario (); //Find the opt. sltn. to single scenario case
       to potentially generate good initial MIP sltns.
   preret_decisionmakers ();
   singlesc_maximum=0; //Reset
   preret_adversarial () ;
   advtimeend=0; //Reset
   advtimestart=0;
   dmtimeend=0;
   dmtimestart=0;
   worstrettimeend=0;
   worstrettimestart=0;
   if (breakit==true) {
      breakit=false;
      break;
   }
   if (numberofiterations==1) {
      mipgap=initialmipgap; //Start from higher MIP gap tolerances for
          earlier scenarios
   }
   if (gap<= epsilon && mipgap<=0.01) {
```

```
                    mipgap=initialmipgap; //Reverse mipgap tolerance to a greater value
                    singlesctime_ineachrun=0;
                    worstrettime_ineachrun=0;
                    dmtime_ineachrun=0;
                    advtime_ineachrun=0;
                    System.out.println("Number of iterations "+numberofiterations);
                    avgiteration+=numberofiterations;
                    System.out.println("--------------");
                    countconvergence++;
                    break;
                 }
                 else
                    if (mm==maxiteration) {
                       long endTime = System.currentTimeMillis();
                       MemoryUsage heapMemoryUsage =
                           ManagementFactory.getMemoryMXBean().getHeapMemoryUsage();
                       countnonconvergence++;
                       System.out.println("Max iteration reached");
                    } else {
                       if (gap<=epsilon) {
                          mipgap=0.01;
                       }
                    }
                 if (gap>=5*epsilon && mipgap>0.01) {
                    //Gap is still too large, reduce mipgap tolerance by a greater value
                    mipgap=mipgap*0.7;
                 } else
                 { //Gap is smaller, reduce mipgap tolerance by a slightly smaller value
                    if (mipgap>0.01){
                       mipgap=mipgap*0.9;}
                 }
              }
           } //Instance
        } //Disposal options
     } //Setup costs
  } //Gamma
} //Returns
```

```
}

public static void preret_adversarial () throws FileNotFoundException {
advtimestart = System.currentTimeMillis();
double[] xm_f = new double [t+1]; //Manufactured items
double[] xr_f = new double [t+1]; //Remanufactured items
double[] d_f = new double [t+1]; //Disposed returns
//Fixed Variables for the Adversarial Problem
for (int i=1; i<=t; i++) {
   xm_f[i] = xm_set[i];
   xr_f[i] =xr_set[i];
   d_f[i]=d_set[i];
}
try {
   IloCplex cplex = new IloCplex();
   File advlog = new File("Kblog_"+ k_backlog+ "UNCAP_SingleScenario_MIPgap_"
      +initialmipgap +"_v127_ADVLOG_MILP_" + ReturnChoice+ GammaChoice+ SetupChoice+
      DisposalChoice+ t+ ".txt");
   PrintWriter advwrite = new PrintWriter(new BufferedWriter(new FileWriter("Kblog_"+
      k_backlog+ "UNCAP_SingleScenario_MIPgap_"+ initialmipgap+ "_v127_ADVLOG_MILP_"+
      ReturnChoice+ GammaChoice+ SetupChoice+ DisposalChoice+ t+ ".txt", true)));
   advwrite.println(" ");
   advwrite.print("INSTANCE "+runmanytimes+" ITERATION "+numberofiterations+" ");
   advwrite.close();
   FileOutputStream oFile = new FileOutputStream(advlog, true);
   cplex.setOut(oFile);
   cplex.setParam(IloCplex.Param.TimeLimit, (Math.max((timelim-
      (worstrettime_ineachrun+ dmtime_ineachrun+ advtime_ineachrun) /1000),0)));
   String[] ZDname = new String [t+1];
   String[] HSname = new String [t+1];
   String[] HRname = new String [t+1];
   String[] sname = new String [t+1];
   //Decision Variables
   for (int names=0; names<=t; names++) {
      ZDname[names] = "ZD_" + names;
      HSname[names] = "HS_" + names;
      HRname[names] = "HR_" + names;
```

```
   sname[names] = "s_" + names;
}
IloNumVar[] HS = cplex.numVarArray (t+1,-Double.MAX_VALUE,Double.MAX_VALUE,
    IloNumVarType.Float,HSname);
IloNumVar[] zd = cplex.numVarArray (t+1,0,Double.MAX_VALUE,
    IloNumVarType.Float,ZDname);
IloNumVar pi_adv = cplex.numVar(0, Double.MAX_VALUE, IloNumVarType.Float, "pi_adv");
IloIntVar[] s = cplex.boolVarArray (t+1,sname);
//Objective Function
IloLinearNumExpr obj = cplex.linearNumExpr();
obj.addTerm(1, pi_adv);
cplex.addMaximize(obj);
//Constraints
Arrays.fill(bigM1_adv,0);
Arrays.fill(bigM2_adv,0);
double rhs2=0, rhs3=0, rhs4=0, rhs5=0, rhs6=0, rhs7=0, rhs8=0, rhs10=0, rhs11=0;
//set big m values
for (int setms=1; setms<=t; setms++){
   if (setms==1) {
      bigM2_adv[setms]= (chs*(xm_f[setms]+ xr_f[setms]- (dbar[setms]- dcap[setms])))+
          (backlog_cost[setms]* (xm_f[setms]+ xr_f[setms]- (dbar[setms]-
          dcap[setms])));
      bigM1_adv[setms]= (-backlog_cost[setms] *(xm_f[setms]+ xr_f[setms]-
          (dbar[setms]+dcap[setms]))) -(chs*(xm_f[setms]+ xr_f[setms]-
          (dbar[setms]+dcap[setms])));
   } else {
      bigM2_adv[setms]=bigM2_adv[setms-1]+ (chs*(xm_f[setms]+ xr_f[setms]-
          (dbar[setms]- dcap[setms])))+ (backlog_cost[setms]* (xm_f[setms]+
          xr_f[setms]- (dbar[setms]- dcap[setms])));
      bigM1_adv[setms]= bigM1_adv[setms-1]+ (-backlog_cost[setms]*
          (xm_f[setms]+xr_f[setms] -(dbar[setms]+dcap[setms]))) -(chs*(xm_f[setms]+
          xr_f[setms]- (dbar[setms]+dcap[setms])));
   }
}
//-1- Pi Constraint
IloLinearNumExpr lhs1 = cplex.linearNumExpr(); //Setting LHS
for(int i=1; i <= t; i++) {
```

```
   lhs1.addTerm(-1,HS[i]);
}
lhs1.addTerm(1,pi_adv); //...more variables (LHS)
IloRange con1 = cplex.addLe(lhs1,rhspi_adv);
rhspi_adv=0;
lhs1.clear();
//HS_t first constraint (GE), case of holding inventory
IloLinearNumExpr lhs10 = cplex.linearNumExpr(); //Setting LHS
for(int i=1; i <= t; i++) {
   lhs10.addTerm(1,HS[i]);
   for(int tt=1; tt <= i; tt++) {
      lhs10.addTerm(chs*dcap[tt], zd[tt]);
      rhs10+=(chs*(xm_f[tt]+xr_f[tt]-dbar[tt])); //RHS
   }
   rhs10+=chs*initial_servicables_inv;
   IloRange con10 = cplex.addGe(lhs10,rhs10);
   rhs10=0;
   lhs10.clear();
}
//HS_t second constraint (GE), case of backlogging
IloLinearNumExpr lhs11 = cplex.linearNumExpr(); //Setting LHS
for(int i=1; i <= t; i++) {
   lhs11.addTerm(1,HS[i]);
   for(int tt=1; tt <= i; tt++) {
      lhs11.addTerm(-backlog_cost[tt]*dcap[tt], zd[tt]);
      rhs11+=(-backlog_cost[tt]*(xm_f[tt]+xr_f[tt]-dbar[tt]));
   }
   rhs11+=-backlog_cost[i]*initial_servicables_inv;
   IloRange con11 = cplex.addGe(lhs11,rhs11);
   rhs11=0;
   lhs11.clear();
}
//HS_t first constraint, case of holding inventory
IloLinearNumExpr lhs2 = cplex.linearNumExpr(); //Setting LHS
for(int i=1; i <= t; i++) {
   lhs2.addTerm(1,HS[i]);
   lhs2.addTerm(bigM1_adv[i], s[i]);
```

```
   for(int tt=1; tt <= i; tt++) {
      lhs2.addTerm(chs*dcap[tt], zd[tt]);
      rhs2+=(chs*(xm_f[tt]+xr_f[tt]-dbar[tt]));
   }
   rhs2+=chs*initial_servicables_inv+bigM1_adv[i];
   IloRange con2 = cplex.addLe(lhs2,rhs2);
   rhs2=0;
   lhs2.clear();
}
//HS_t second constraint, case of backlogging..
IloLinearNumExpr lhs5 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   lhs5.addTerm(1,HS[i]);
   lhs5.addTerm(-bigM2_adv[i], s[i]);
   for(int tt=1; tt <= i; tt++) {
      lhs5.addTerm(-backlog_cost[tt]*dcap[tt], zd[tt]);
      rhs5+=(-backlog_cost[tt]*(xm_f[tt]+xr_f[tt]-dbar[tt]));
   }
   rhs5+=-backlog_cost[i]*initial_servicables_inv;
   IloRange con5 = cplex.addLe(lhs5,rhs5);
   rhs5=0;
   lhs5.clear();
}
//HS_t third constraint
IloLinearNumExpr lhs6 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= i; tt++) {
      lhs6.addTerm(-dcap[tt], zd[tt]);
      lhs6.addTerm(-1*(xm_f[tt]+xr_f[tt]-dbar[tt]+dcap[tt]), s[i]);
      rhs6+=(-1*(xm_f[tt]+xr_f[tt]-dbar[tt]));
   }
   IloRange con6 = cplex.addLe(lhs6,rhs6);
   rhs6=0;
   lhs6.clear();
}
//HS_t fourth constraint
IloLinearNumExpr lhs7 = cplex.linearNumExpr();
```

```
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= i; tt++) {
      lhs7.addTerm(dcap[tt], zd[tt]);
      lhs7.addTerm(-1*(xm_f[tt]+ xr_f[tt]- dbar[tt]- dcap[tt]), s[i]);
      rhs7+=(dcap[tt]);
   }
   IloRange con7 = cplex.addLe(lhs7,rhs7);
   rhs7=0;
   lhs7.clear();
}
for (int i=1; i<=t; i++) { //Forall t=1..T
   cplex.addGe(zd[i],0);
   cplex.addLe(zd[i],1);
}
//Gamma constraint
IloLinearNumExpr lhs3 = cplex.linearNumExpr(); //LHS
for (int j=1; j<=t; j++) { //Forall t=1..T
   rhs3=gammaD[j];
   for (int i=1; i<=j; i++){
   lhs3.addTerm(1,zd[i]);
   }
   cplex.addLe(lhs3, rhs3);
   rhs3=0;
   lhs3.clear();
}
//Solve
if (cplex.solve()){
   int constraints=cplex.getNrows();
   int variables=cplex.getNcols();
   optimal_pi_ADV=cplex.getValue(pi_adv);
   advtimeend = System.currentTimeMillis();
   totaladvtime=totaladvtime+ advtimeend - advtimestart;
   advstatus=" "+cplex.getStatus()+" ";
   //Local UB
   System.out.println("ADV objective "+ cplex.getObjValue());
   //Global UB
   if (numberofiterations==1) {
```

```
   ADVub=(cplex.getValue(pi_adv)+prodcosts);
}
ADVub = Math.min(ADVub,(cplex.getValue(pi_adv)+prodcosts));
System.out.println("pi_ADV "+cplex.getValue(pi_adv)+" ");
if(DMlb>=ADVub) {
   gap=((ADVub-dm_local_objective)/dm_local_objective);//use local lb
} else {
   gap=((ADVub-DMlb)/DMlb);//use global lb
}
try {
   PrintWriter outcases = new PrintWriter(new BufferedWriter(new
       FileWriter("Kblog_"+ k_backlog+ "UNCAP_SingleScenario_MIPgap_"+
       initialmipgap+ "_v127_MILP_"+ ReturnChoice+ GammaChoice+ SetupChoice+
       DisposalChoice+ t+".txt", true)));

   if (numberofiterations==1 && runmanytimes==1) {
      outcases.write("RUN ");
      outcases.write("ITERATION ");
      outcases.write("GLOBAL_LB ");
      outcases.write("PI_DM ");
      outcases.write("GLOBAL_UB ");
      outcases.write("LOCAL_UB ");
      outcases.write("PI_ADV ");
      outcases.write("GAP ");
      outcases.write("FINAL_NODE_SIZE ");
      outcases.write("TIME_TAKEN_DM ");
      outcases.write("TIME_TAKEN_ADV ");
      outcases.write("TIME_TAKEN_WRZ ");
      outcases.write("TIME_TAKEN_SINGLE_SCENARIO ");
      outcases.write("FINAL_MIP_GAP ");
      outcases.println(" ");
   }
   outcases.print(runmanytimes+" ");
   outcases.print(numberofiterations+" ");
   outcases.print(DMlb+" ");
   outcases.print(optimal_pi_DM+" ");
   outcases.print(ADVub+" ");
```

```
   outcases.print((cplex.getValue(pi_adv)+prodcosts)+" ");
   outcases.print(optimal_pi_ADV+" ");
   outcases.print(gap+" ");
   outcases.print(finalnodesize+" ");
   outcases.print(dmtimeend - dmtimestart+" ");
   singlesctime_ineachrun+=singlesctimeend - singlesctimestart;
   worstrettime_ineachrun+=worstrettimeend - worstrettimestart;
   dmtime_ineachrun+=dmtimeend - dmtimestart;
   advtime_ineachrun+=advtimeend - advtimestart;
   if (gap<=epsilon){
      outcases.print(advtimeend - advtimestart+" ");
      outcases.print(worstrettimeend - worstrettimestart+" ");
      outcases.print(singlesctimeend - singlesctimestart+" ");
      outcases.print(finalmipgap+" ");
      outcases.print("DM_STATUS : "+dmstatus+" ");
      outcases.print("ADV_STATUS :"+advstatus+" ");
      outcases.print("FINAL_GAP :"+mipgap+" ");
      outcases.println(numberofiterations+" ");
   } else {
      outcases.print(advtimeend - advtimestart+" ");
      outcases.print(worstrettimeend - worstrettimestart+" ");
      outcases.print(singlesctimeend - singlesctimestart+" ");
      outcases.print(finalmipgap+" ");
      outcases.print("DM_STATUS : "+dmstatus+" ");
      outcases.println("ADV_STATUS :"+advstatus+ " ");
   }
   outcases.close();
} catch (IOException e) {}
dmtimeend=0;
dmtimestart=0;
advtimeend=0;
advtimestart=0;
worstrettimeend=0;
worstrettimestart=0;
singlesctimeend=0;
singlesctimestart=0;
for (int i=1; i<=t; i++) {
```

```
        zd_set[i][numberofiterations]=cplex.getValue(zd[i]);
      }
      cplex.end();
   }
}
catch (IloException exc) {
   exc.printStackTrace();
} catch (IOException e1) {
   e1.printStackTrace(); }}

public static void preret_decisionmakers () throws FileNotFoundException {
dm_local_objective=0;
dmtimestart = System.currentTimeMillis();
double[] zd_f = new double [t+1];
//Fixed Variables for the Decision Maker's Problem
for (int i=1; i<=t; i++) {
   zd_f[i]=zd_set[i][numberofiterations-1];
}
try {
   IloCplex cplex = new IloCplex();
   File dmlog = new File("Kblog_"+k_backlog+ "UNCAP_SingleScenario_MIPgap_"
      +initialmipgap+ "_v127_DMLOG_MILP_" +ReturnChoice+ GammaChoice+ SetupChoice+
      DisposalChoice+ t+".txt");
   PrintWriter dmwrite = new PrintWriter(new BufferedWriter(new FileWriter("Kblog_"+
      k_backlog+ "UNCAP_SingleScenario_MIPgap_"+ initialmipgap+ "_v127_DMLOG_MILP_"+
      ReturnChoice+ GammaChoice+ SetupChoice+ DisposalChoice+ t+ ".txt", true)));
   dmwrite.print("INSTANCE "+runmanytimes+" ITERATION "+numberofiterations+" ");
   dmwrite.close();
   FileOutputStream o2File = new FileOutputStream(dmlog, true);
   cplex.setOut(o2File);
   cplex.setParam(IloCplex.Param.MIP.Tolerances.MIPGap, mipgap);
   cplex.setParam(IloCplex.Param.TimeLimit,
      (Math.max((timelim-(worstrettime_ineachrun+dmtime_ineachrun+advtime_ineachrun)/1000),0)));
   String[] HSname = new String [t+1];
   String[] HRname = new String [t+1];
   String[] xmname = new String [t+1];
   String[] xrname = new String [t+1];
```

```
String[] dname = new String [t+1];
String[] yname = new String [t+1];
//Decision Variables
for (int names=0; names<=t; names++) {
   HSname[names] = "HS" + names;
   HRname[names] = "HR" + names;
   xmname[names] = "xm" + names;
   xrname[names] = "xr" + names;
   dname[names] = "d" + names;
   yname[names] = "y" + names;
}
IloNumVar[] xm = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Int,
    xmname);
IloNumVar[] xr = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Int,
    xrname);
IloNumVar[] d = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Int,dname);
IloIntVar[] y = cplex.boolVarArray (t+1,yname);

IloNumVar[][] HS = new IloNumVar[t+1][];
for (int i = 1; i <= t; i++) {
   HS[i] = cplex.numVarArray(numberofiterations+1, Double.MIN_VALUE,
       Double.MAX_VALUE);
}
IloNumVar pi_dm = cplex.numVar(0, Double.MAX_VALUE, IloNumVarType.Float, "pi_dm");
//Objective
IloLinearNumExpr obj = cplex.linearNumExpr();
obj.addTerm(1, pi_dm);
for(int i=1; i<=t; i++){
   obj.addTerm(K, y[i]);
   obj.addTerm(m, xm[i]);
   obj.addTerm(r, xr[i]);
   obj.addTerm(dis, d[i]);
}
cplex.addMinimize(obj);
//Constraints
double rhs2=0, rhs3=0, rhs4=0, rhs6=0;
//-1- Pi Constraint
```

```
IloLinearNumExpr lhs9 = cplex.linearNumExpr(); //LHS
for(int prev_iterations=1; prev_iterations <= numberofiterations; prev_iterations++)
    {
   for(int i=1; i <= t; i++) { //sum i=1 ... T
      lhs9.addTerm(-1,HS[i][prev_iterations]);
      for(int tt=1; tt <= i; tt++) { //sum 1..i
         lhs9.addTerm(chr, d[tt]);
         lhs9.addTerm(chr, xr[tt]);
         rhspi_dm+=(chr*(rbar[tt]+(rcap[tt]*zr_worst[tt]))); //RHS
      }
   }
   lhs9.addTerm(1,pi_dm); //...more variables (LHS)
   IloRange con9 = cplex.addGe(lhs9,rhspi_dm);
   rhspi_dm=0;
   lhs9.clear();
}
//-2- HS Balance Constraint - Case of holding inventory
IloLinearNumExpr lhs2 = cplex.linearNumExpr(); //LHS
for(int prev_iterations=1; prev_iterations <= numberofiterations; prev_iterations++)
    {
   for(int i=1; i <= t; i++) { //i=1 ... T
      for(int tt=1; tt <= i; tt++) { //sum 1..i
         lhs2.addTerm(-chs, xm[tt]);
         lhs2.addTerm(-chs, xr[tt]);
         rhs2+=(-chs*(dbar[tt]+(dcap[tt]*zd_set[tt][prev_iterations-1]))); //RHS
      }
      rhs2+=chs*initial_servicables_inv;
      lhs2.addTerm(1,HS[i][prev_iterations]);
      IloRange con2 = cplex.addGe(lhs2,rhs2);
      rhs2=0;
      lhs2.clear();
   }
}
//-3- HS Balance Constraint - Case of backlogging
IloLinearNumExpr lhs3 = cplex.linearNumExpr(); //LHS
for(int prev_iterations=1; prev_iterations <= numberofiterations; prev_iterations++)
    {
```

```
   for(int i=1; i <= t; i++) { //i=1 ... T
      for(int tt=1; tt <= i; tt++) { //sum 1..i
         lhs3.addTerm(backlog_cost[tt], xm[tt]);
         lhs3.addTerm(backlog_cost[tt], xr[tt]);
         rhs3+=(backlog_cost[tt]* (dbar[tt]+(dcap[tt]*
             zd_set[tt][prev_iterations-1]))); //RHS
      }
      rhs3+=-backlog_cost[i]*initial_servicables_inv;
      lhs3.addTerm(1,HS[i][prev_iterations]);
      IloRange con3 = cplex.addGe(lhs3,rhs3);
      rhs3=0;
      lhs3.clear();
   }
}
Arrays.fill(bigM_dm_setup,0);
double sumofds=0;
for(int tt=1; tt<=t; tt++){
   sumofds=sumofds+dbar[tt] + dcap[tt];
}
Arrays.fill(bigM_dm_setup,sumofds);
System.out.print("bigM = [");
for (int tt=1; tt<=t; tt++) {
   System.out.print(bigM_dm_setup[tt]);
   if (tt != t) {
      System.out.print(","); } else {
         System.out.println("]");
      }
}
//-5- Setup constraint
IloLinearNumExpr lhs5 = cplex.linearNumExpr(); //LHS
for (int j=1; j<=t; j++) {
   lhs5.addTerm(bigM_dm_setup[j], y[j]);
   lhs5.addTerm(-1, xm[j]);
   lhs5.addTerm(-1 , xr[j]);
   cplex.addGe(lhs5, 0);
   lhs5.clear();
}
```

```
//-6- Returns nonnegativity
IloLinearNumExpr lhs6 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= i; tt++) { //sum 1..i
      lhs6.addTerm(chr, xr[tt]);
      lhs6.addTerm(chr, d[tt]);
      rhs6+=(chr*(rbar[tt]));
   }
   IloRange con6 = cplex.addLe(lhs6,rhs6);
   rhs6=0;
   lhs6.clear();
}
/*Setting branching priority*/
int mypri=t;
for(int pri=1; pri<=t; pri++){
   cplex.setPriority(y[pri], mypri);
   mypri--;
}
cplex.setParam(IloCplex.Param.Emphasis.MIP, 3);
//Solve
cplex.setParam(IloCplex.IntParam.AdvInd,2);
cplex.readMIPStarts("optimal_sltn_iteration_" +(numberofiterations-1) +"_run_"
    +runmanytimes +".sol");
if (cplex.solve()){
   finalmipgap = cplex.getMIPRelativeGap();
   prodcosts = cplex.getObjValue() - cplex.getValue(pi_dm);
   prev_best_obj=cplex.getBestObjValue(); //Gets the best bound at optimal.
   prev_gap=mipgap;
   prev_optimal=cplex.getObjValue();
   int constraints=cplex.getNrows();
   int variables=cplex.getNcols();
   DMobjective = cplex.getObjValue();
   optimal_pi_DM=cplex.getValue(pi_dm);
   dmtimeend = System.currentTimeMillis();
   File mipstart_dm = new File("optimal_sltn_iteration_" +numberofiterations+
       "_run_"+ runmanytimes+ ".sol");
   mipstart_dm.delete();
```

```
   cplex.writeMIPStarts("optimal_sltn_iteration_" +numberofiterations+ "_run_"+
       runmanytimes+ ".sol");
   File mystart= new File("optimal_sltn_iteration_"+ numberofiterations+ "_run_"+
       runmanytimes+ ".sol");
   String search = "MIPStartEffortLevel=\"0\"";
   String replace = "MIPStartEffortLevel=\"2\"";
   try{
      FileReader fr = new FileReader(mystart);
      String s;
      String totalStr = "";
      try (BufferedReader br = new BufferedReader(fr)) {

         while ((s = br.readLine()) != null) {
            totalStr += s + "\n";
            if (s.length() >= 17) {
               if (s.substring(0, 17).equals(" solutionIndex")) {
                  Scanner in = new Scanner(s).useDelimiter("[^0-9]+");
                  sltnindex = in.nextInt();
               }
            }
         }
         totalStr = totalStr.replaceAll(search, replace);
         FileWriter fw = new FileWriter(mystart);
         fw.write(totalStr);
         fw.close();
      }
   }catch(Exception e){
      e.printStackTrace();
   }
   finalnodesize=cplex.getNnodes();//final node size
} else{
   breakit=true;
}
System.out.println("DM STATUS : "+cplex.getCplexStatus());
dmstatus=" "+cplex.getCplexStatus()+" ";
System.out.println("DM objective "+ cplex.getObjValue());
dm_local_objective=cplex.getObjValue();
```

```
   DMlb = Math.max(DMlb,cplex.getObjValue());
   System.out.print(" pi_DM "+cplex.getValue(pi_dm)+" ");
   totaldmtime=totaldmtime+dmtimeend - dmtimestart;
   for (int i=1; i<=t; i++) {
      xm_set[i]=cplex.getValue(xm[i]);
      xr_set[i]=cplex.getValue(xr[i]);
      d_set[i]=cplex.getValue(d[i]);
   }
   cplex.end();
}
catch (IloException exc) {
   exc.printStackTrace();
}
catch (IOException e) {
   e.printStackTrace(); }}

public static void returns_precompute () throws FileNotFoundException {
worstrettimestart = System.currentTimeMillis();
try {
   IloCplex cplex = new IloCplex();
   File worstretlog = new File("Kblog_"+ k_backlog+ "UNCAP_RetPreComp_MIPgap_"+
      initialmipgap+ "_v127_WORSTRETLOG_MILP_"+ ReturnChoice+ GammaChoice+ SetupChoice+
      DisposalChoice+ t+ ".txt");
   PrintWriter worstretwrite = new PrintWriter(new BufferedWriter(new
      FileWriter("Kblog_"+ k_backlog+ "UNCAP_RetPreComp_MIPgap_"+ initialmipgap+
      "_v127_WORSTRET_MILP_"+ ReturnChoice+ GammaChoice+ SetupChoice+ DisposalChoice+
      t+ ".txt", true)));
   worstretwrite.println(" ");
   worstretwrite.print("INSTANCE "+runmanytimes+" ITERATION "+numberofiterations+" ");
   worstretwrite.close();
   FileOutputStream o2File = new FileOutputStream(worstretlog, true);
   cplex.setOut(o2File);
   cplex.setParam(IloCplex.Param.TimeLimit, (Math.max((timelim-
      (worstrettime_ineachrun+ dmtime_ineachrun+ advtime_ineachrun)/1000), 0)));
   String[] worstretname = new String [t+1];
   //Decision Variables
   for (int names=0; names<=t; names++) {
```

```
      worstretname[names] = "wzr" + names;
   }
   IloNumVar[] wzr = cplex.numVarArray (t+1,0,1, IloNumVarType.Float, worstretname);
   //Objective
   IloLinearNumExpr obj = cplex.linearNumExpr();
   for(int i=1; i<=t; i++){
      obj.addTerm((t-i+1)*rcap[i], wzr[i]);
   }
   cplex.addMaximize(obj);
   //Constraints
   double rhs=0;
   //-1- Gamma constraint
   IloLinearNumExpr lhs = cplex.linearNumExpr();
   for(int i=1; i <= t; i++) {
      for(int tt=1; tt <= i; tt++) {
         lhs.addTerm(1, wzr[tt]);
      }
      rhs = gammaR[i];
      IloRange con = cplex.addLe(lhs,rhs);
      rhs=0;
      lhs.clear();
   }
   //Solve
   if (cplex.solve()){
      worstrettimeend = System.currentTimeMillis();
   }
   else{
      breakit=true;
    }
   for (int i=1; i<=t; i++) {
      zr_worst[i]=cplex.getValue(wzr[i]);
   }
   cplex.end();
}
catch (IloException exc) {
   exc.printStackTrace();
} catch (IOException e) {
```

```
   e.printStackTrace();
}}

public static void facloc_singlescenario () throws FileNotFoundException {
singlesctimestart = System.currentTimeMillis();
double[] zd_f = new double [t+1];
for (int i=1; i<=t; i++) {
   zd_f[i]=zd_set[i][numberofiterations-1];
}
try {
   IloCplex cplex = new IloCplex();
   File dmlog = new File("Kblog_"+ k_backlog+ "UNCAP_SingleScenario_MIPgap_"+
      initialmipgap+ "_v127_SINGLESCLOG_MILP_OurDataSets_"+ ReturnChoice+ GammaChoice+
      SetupChoice+ DisposalChoice+ t+ ".txt");
   PrintWriter dmwrite = new PrintWriter(new BufferedWriter(new FileWriter("Kblog_"
      +k_backlog +"UNCAP_SingleScenario_MIPgap_" +initialmipgap
      +"_v127_SINGLESCLOG_MILP_OurDataSets_" +ReturnChoice +GammaChoice +SetupChoice
      +DisposalChoice +t +".txt", true)));
   dmwrite.print("INSTANCE "+runmanytimes+" ITERATION "+numberofiterations+" ");
   dmwrite.close();
   FileOutputStream o2File = new FileOutputStream(dmlog, true);
   cplex.setOut(o2File);
   cplex.setParam(IloCplex.Param.TimeLimit,
      (Math.max((timelim-(singlesctime_ineachrun+worstrettime_ineachrun+dmtime_ineachrun+advtime
   String[] HSname = new String [t+1];
   String[] HRname = new String [t+1];
   String[] xmtildename = new String [t+2];
   String[] xrtildename = new String [t+2];
   String[] xmname = new String [t+1];
   String[] xrname = new String [t+1];
   String[] dname = new String [t+1];
   String[] yname = new String [t+1];
   String[] hstildename = new String [t+1];
   String[] hrtildename = new String [t+1];
   String[] btildename = new String [t+1];
   //Decision Variables
   for (int names=0; names<=t; names++) {
```

```
   HSname[names] = "HS" +names;
   HRname[names] = "HR" +names ;
   xmname[names] = "xm" + names;
   xrname[names] = "xr" + names;
   dname[names] = "d" + names;
   yname[names] = "y" + names;
   hstildename[names]= "hstilde" + names;
   hrtildename[names] = "hrtilde" + names;
   btildename[names] = "btilde" + names;
}
for (int names=0; names<=t+1; names++) {
   xmtildename[names] = "xmtilde" + names ;
   xrtildename[names] = "xrtilde" + names;
}
IloNumVar[] HS = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
    HSname);
IloNumVar[] HR = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
    HRname);
IloNumVar[] B = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
    HRname);
IloNumVar[] xmtot = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
    xmname);
IloNumVar[] xrtot = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
    xrname);
IloNumVar[] d = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Int,dname);
IloIntVar[] y = cplex.boolVarArray (t+1,yname);

IloNumVar[][] xm = new IloNumVar[t+2][];
for (int i = 1; i <= t+1; i++) {
   xm[i] = cplex.numVarArray(t+2, 0, Double.MAX_VALUE, IloNumVarType.Int,
       xmtildename);
}
IloNumVar[][] xr = new IloNumVar[t+2][];
for (int i = 1; i <= t+1; i++) {
   xr[i] = cplex.numVarArray(t+2, 0, Double.MAX_VALUE, IloNumVarType.Int,
       xrtildename);
}
```

```
IloNumVar pi_dm = cplex.numVar(0, Double.MAX_VALUE, IloNumVarType.Float, "pi_dm");
//Objective
IloLinearNumExpr obj = cplex.linearNumExpr();
obj.addTerm(1, pi_dm);
for(int i=1; i<=t; i++){
   obj.addTerm(K, y[i]);
   for (int mm=1; mm<=t; mm++) {
      obj.addTerm(m, xm[mm][i]);
      obj.addTerm(r, xr[mm][i]);}
   obj.addTerm(dis, d[i]);
}
cplex.addMinimize(obj);
//Constraints
double rhs2=0, rhs3=0, rhs4=0, rhs5=0, rhs13=0, rhs14=0;
IloLinearNumExpr lhsxmsum = cplex.linearNumExpr();
IloLinearNumExpr lhsxrsum = cplex.linearNumExpr();
lhsxmsum.clear();
lhsxrsum.clear();
for (int i=1; i<=t; i++) {
   for (int tt=1; tt<=t+1; tt++) {
      lhsxmsum.addTerm(-1, xm[i][tt]);
      lhsxrsum.addTerm(-1, xr[i][tt]);
   }
   lhsxmsum.addTerm(1, xmtot[i]);
   lhsxrsum.addTerm(1, xrtot[i]);
   cplex.addEq(lhsxmsum,0);
   cplex.addEq(lhsxrsum,0);
   lhsxmsum.clear();
   lhsxrsum.clear();
}
IloLinearNumExpr lhspi = cplex.linearNumExpr();
for(int i=1; i <= t; i++) { //sum i=1 ... T
   lhspi.addTerm(-1,HS[i]);
   lhspi.addTerm(-1,HR[i]);
   lhspi.addTerm(-1,B[i]);
}
```

```
lhspi.addTerm(1,pi_dm); //...more variables (LHS)
IloRange con9 = cplex.addGe(lhspi,0);
lhspi.clear();
//-2- HS Balance Constraint - Case of holding inventory
IloLinearNumExpr lhs2 = cplex.linearNumExpr(); //Setting LHS of our constraint..
for(int i=1; i <= t; i++) { //i=1 ... T
   for(int tt=1; tt <= i; tt++) { //sum 1..i
      for (int tt2=i+1; tt2 <= t+1; tt2++) { //sum i+1..T+1
         lhs2.addTerm(-chs, xm[tt][tt2]);
         lhs2.addTerm(-chs, xr[tt][tt2]);
      }
   }
   lhs2.addTerm(1,HS[i]);
   IloRange con2 = cplex.addEq(lhs2,0);
   con2.setName("HSTilde_LastIt_Inv_"+numberofiterations);
   lhs2.clear();
}
//-3- Returns inventory
IloLinearNumExpr lhs3 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= i; tt++) {
      lhs3.addTerm(chr, d[tt]);
      rhs3+=chr*(rbar[tt]+(rcap[tt]*zr_worst[tt]));
      for (int tt2=1; tt2 <= t+1; tt2++) {
         lhs3.addTerm(chr, xr[tt][tt2]);
      }
   }
   lhs3.addTerm(1,HR[i]);
   IloRange con3 = cplex.addEq(lhs3,rhs3);
   con3.setName("HRTilde_LastIt_"+numberofiterations);
   lhs3.clear();
   rhs3=0;
}
// -4- Backlogging
IloLinearNumExpr lhs4 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= i; tt++) {
```

```
      for (int tt2=i+1; tt2 <= t+1; tt2++) {
         lhs4.addTerm(-backlog_cost[tt2-1], xm[tt2][tt]);
         lhs4.addTerm(-backlog_cost[tt2-1], xr[tt2][tt]);
      }
   }
   lhs4.addTerm(1,B[i]);
   IloRange con4 = cplex.addEq(lhs4,0);
   con4.setName("Blog_"+numberofiterations);
   lhs4.clear();
}
// -5- Demand Satisfaction
IloLinearNumExpr lhs5 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= t+1; tt++) {
      lhs5.addTerm(1, xm[tt][i]);
      lhs5.addTerm(1, xr[tt][i]);
   }
   rhs5=dbar[i]+(dcap[i]*zd_set[i][numberofiterations-1]);
   IloRange con5 = cplex.addEq(lhs5,rhs5);
   con5.setName("Demand_Satisfaction_"+numberofiterations);
   rhs5=0;
   lhs5.clear();
}
// -6- Setup Constraint (Extended Formulation Version)
IloLinearNumExpr lhs6 = cplex.linearNumExpr(); //Setting LHS of our constraint..
for(int i=1; i <= t; i++) {
   for(int tt=1; tt <= t; tt++) {
      lhs6.addTerm(1, xm[i][tt]);
      lhs6.addTerm(1, xr[i][tt]);
      lhs6.addTerm(-(dbar[tt]+dcap[tt]), y[i]);
      IloRange con6 = cplex.addLe(lhs6,0);
      con6.setName("Setup_"+numberofiterations);
      lhs6.clear();
   }
}
/*Setting branching priorities*/
int mypri=t;
```

```
for(int pri=1; pri<=t; pri++){
   cplex.setPriority(y[pri], mypri);
   mypri--;
}
//Solve
if (cplex.solve()){
   if (numberofiterations==1) {
      File mystart= new File("optimal_sltn_iteration_" +(numberofiterations-1)
          +"_run_" +runmanytimes +".sol");
      mystart.delete();
      int index = 1;
      PrintStream fileStream = new PrintStream(mystart);
      fileStream.println("<?xml version = \"1.0\" encoding=\"UTF-8\"
          standalone=\"yes\"?>");
      fileStream.println("<CPLEXSolutions version=\"1.2\">");
      fileStream.println(" <CPLEXSolution version=\"1.2\">");
      fileStream.println(" <header");
      fileStream.println("  problemName=\"ilog.cplex\"");
      fileStream.println("  solutionName=\"m1\"");
      fileStream.println("  solutionIndex=\"0\"");
      fileStream.println("  MIPStartEffortLevel=\"0\"");
      fileStream.println("  writeLevel=\"2\"/>");
      fileStream.println(" <variables>");
      for (int myt=1; myt<=t; myt++) {
         fileStream.println(" <variable name=\"y"+myt+"\" index=\""+index+"\"
             value=\""+(int)Math.round(cplex.getValue(y[myt]))+"\"/>");
         index++;
         index++;
         index++;
         fileStream.println(" <variable name=\"d"+myt+"\" index=\""+index+"\"
             value=\""+(int)Math.round(cplex.getValue(d[myt]))+"\"/>");
         index++;
      }
      fileStream.println(" </variables>");
      fileStream.println(" </CPLEXSolution>");
      fileStream.print("</CPLEXSolutions>");
      fileStream.close();
```

```
}
else {
   File mystart= new
       File("optimal_sltn_iteration_"+(numberofiterations-1)+"_run_"+runmanytimes+".sol");
   RandomAccessFile f = new RandomAccessFile(mystart, "rw");
   byte b;
   long length = f.length() - 1;
   do {
      length -= 1;
      f.seek(length);
      b = f.readByte();
   } while(b != 10 && length>0);
   f.setLength(length+1);
   f.close();
   int index = 1;
   PrintStream fileStream = new PrintStream(new FileOutputStream(mystart, true));
   fileStream.println(" <CPLEXSolution version=\"1.2\">");
   fileStream.println(" <header");
   fileStream.println("  problemName=\"ilog.cplex\"");
   fileStream.println("  solutionName=\"mSce\"");
   fileStream.println("  solutionIndex=\""+(sltnindex+1)+"\"");
   fileStream.println("  MIPStartEffortLevel=\"0\"");
   fileStream.println("  writeLevel=\"2\"/>");
   fileStream.println(" <variables>");
   for (int myt=1; myt<=t; myt++) {
      fileStream.println(" <variable name=\"y"+myt+"\" index=\""+index+"\"
          value=\""+(int)Math.round(cplex.getValue(y[myt]))+"\"/>");
      index++;
      index++;
      index++;
      fileStream.println(" <variable name=\"d"+myt+"\" index=\""+index+"\"
          value=\""+(int)Math.round(cplex.getValue(d[myt]))+"\"/>");
      index++;
   }
   fileStream.println(" </variables>");
   fileStream.println(" </CPLEXSolution>");
   fileStream.print("</CPLEXSolutions>");
```

```
        fileStream.close();
      }
      singlescenario_optimal[numberofiterations] = cplex.getObjValue();
      singlesctimeend = System.currentTimeMillis();
      cplex.end();
   }
}
catch (IloException exc) {
   exc.printStackTrace();
} catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
}}}
```

A.2 Decomposition Algorithm (2-MCR-CRA)

```
import java.util.*;
import java.io.*;
import java.lang.management.ManagementFactory;
import java.lang.management.MemoryUsage;
import java.nio.file.Files;
import java.nio.file.Paths;
import ilog.concert.IloException;
import ilog.concert.IloIntVar;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.concert.IloNumVarType;
import ilog.concert.IloRange;
import ilog.cplex.IloCplex;

public class MLCR_MinMax_AdjustableProd {
public static String ReturnChoice="High";
public static String GammaChoice="High";
public static String SetupChoice="High";
public static String m0Choice="Med";
public static String rcapChoice="High";
public static String[] changeReturnChoice = {"","High","Med","Low"};
```

```
public static String[] changeGammaChoice = {"","High","Med","Low"};
public static String[] changeSetupChoice = {"","High","Med","Low"};
public static String[] changem0Choice = {"","Med","Low"};
public static String[] changercapChoice = {"","High","Med","Low"};
public static boolean breakit=false;
public static int flowcover=0;
public static int gomory=0;
public static int covercuts=0;
public static int MIRcuts=0;
public static int rootnode=0;
public static double RootNodeObj_Original=0;
public static int finalnodesize1=0;
public static double timelim=7200;
public static String dmstatus;
public static String advstatus;
public static double initialmipgap=0.01;
public static double mipgap=initialmipgap;
public static double firstsltnmipgap_dmp=0;
public static double firstsltnmipgap_ap=0;
public static double epsilon=0.01;
public static int maxiteration=20;
public static int initial_servicables_inv=0;
public static int initial_returns_inv=0;
public static int t=50;
public static int comp=5; //Number of components
public static double[] K = new double [comp+1]; //Setup cost
public static double[] m = new double [comp+1]; //Manufacturing cost
public static double[] r = new double [comp+1]; //Remanufacturing cost
public static double totaladvtime=0;
public static double totaldmtime=0;
public static double avgadvtime=0;
public static double avgdmtime=0;
public static double avgiteration=0;
public static double prodcosts=0;
public static double dm_local_objective=0;
public static double dmtime_ineachrun=0;
public static double advtime_ineachrun=0;
```

```
public static double chs = 0; //Servicables' holding cost
public static double m0 = 0; //Servicables' holding cost
public static double chr[] = new double [comp+1]; //Returns' holding cost
public static double chc[] = new double [comp+1]; //Returns' holding cost
public static double[] bigM_dm_setup= new double [t+1];
public static double[] bigM_adv_rets = new double [t+1];
public static double[] bigM_adv_dems = new double [t+1];
public static double rhspi_dm=0;
public static double[] dbar = new double [t+1];
public static double[] rbar = new double [t+1];
public static double[] rcap = new double [t+1];
public static double[][] gamma = new double [t+1][comp+1];
public static double[][] zr_set = new double [t+1][maxiteration+1];
public static double[] x0_set = new double [t+1];
public static double[][] hc_set = new double [t+1][comp+1];
public static double[][] y_set = new double [t+1][comp+1];
public static double ADVobjective;
public static double DMobjective;
public static int numberofiterations=0;
public static double optimal_pi_DM=0;
public static double optimal_pi_ADV=0;
public static double gap=0;
public static double finalmipgap=0;
public static double ADVub=0;
public static int runmanytimes=0;
public static long dmtimestart1=0;
public static long dmtimeend1=0;
public static long advtimestart=0;
public static long advtimeend=0;
public static int breaknow=0;

public static void main(String[] args) throws Exception, Exception {
//Run for many instances
for (int runmnyret=1; runmnyret<=1; runmnyret++) {
   //Change returns dataset
   ReturnChoice=changeReturnChoice[runmnyret];
   for (int runmnygam=3; runmnygam<=3; runmnygam++) {
```

```
//Change gamma dataset
GammaChoice=changeGammaChoice[runmnygam];
for (int runmnyset=1; runmnyset<=1; runmnyset++) {
   SetupChoice=changeSetupChoice[runmnyset];
   for (int runmnym0=1; runmnym0<=2; runmnym0++) {
      m0Choice=changem0Choice[runmnym0];
      for (int runmnyrcap=1; runmnyrcap<=1; runmnyrcap++) {
         rcapChoice=changercapChoice[runmnyrcap];
         for (runmanytimes=1; runmanytimes<=1; runmanytimes++) { //1..5
            dmtime_ineachrun=0;
            advtime_ineachrun=0;
            ADVub=0;
            Arrays.fill(bigM_dm_setup,0);
            File f = new File("C:\\ Users\\ npb15184\\ Desktop\\ MLCRData_h0urs\\"
                + ReturnChoice + GammaChoice + SetupChoice + m0Choice + rcapChoice
                + "\\MLCR_R" + ReturnChoice + "Gamma" + GammaChoice + "K" +
                SetupChoice + "m0" + m0Choice + "rcap" + rcapChoice + runmanytimes
                + ".txt");
            boolean demandstart=false;
            BufferedReader freader = new BufferedReader(new FileReader(f));
            //Reading datasets
            String s;
            int myt=1;
            double nextone=0;
            while ((s = freader.readLine()) != null && myt<=t) {
               String[] st = s.split(" ");
               ArrayList<String> results_without_blanks = new ArrayList<String>();
               if (s.equals("t Demand NominalReturn ReturnMaxDeviation Gamma "))
                   {demandstart=true;}
               for (String current_string : st) {
                  if (current_string != null && !current_string.isEmpty()) {
                     results_without_blanks.add(current_string);
                  }
               }
               if (results_without_blanks.size()>0) {
                  if (results_without_blanks.get(0).startsWith(""+myt+"") &&
                     demandstart){
```

```
            dbar[myt]=Double.parseDouble(results_without_blanks.get(1));
            rbar[myt]=Double.parseDouble(results_without_blanks.get(2));
            rcap[myt]=Double.parseDouble(results_without_blanks.get(3));
            for(int cc=1; cc<=comp; cc++){
               gamma[myt][cc]=Double.parseDouble(results_without_blanks.get(4));
            }
            myt++;
         }

         if (nextone==1) {
            nextone=0;
            m0=Double.parseDouble(results_without_blanks.get(0));
            chs = Double.parseDouble (results_without_blanks.get (1));
            for (int mycomp=1; mycomp<=comp; mycomp++) {
               chc[mycomp] = Double.parseDouble (results_without_blanks.get
                   ((5 * (mycomp-1)) + 2));
               m[mycomp] = Double.parseDouble (results_without_blanks.get
                   ((5 * (mycomp - 1)) + 3));
               r[mycomp] = Double.parseDouble (results_without_blanks.get
                   ((5 * (mycomp - 1)) + 4));
               chr[mycomp] = Double.parseDouble (results_without_blanks.get
                   ((5 * (mycomp - 1)) + 5));
               K[mycomp] = Double.parseDouble (results_without_blanks.get
                   ((5 * (mycomp - 1)) + 6));
            }
         }
         if (results_without_blanks.get(0).startsWith("m0")){
            nextone=1;}}}
   for (int mm=1; mm<=maxiteration; mm++) {
      if (mm==1) {
         numberofiterations=0;
         zr_set = new double [t+1][maxiteration+1];
      }
      numberofiterations++;
      if (numberofiterations==1) {
         mipgap=initialmipgap;
      }
```

```
                    decisionmakers ();
                    if (rootnode==0) {
                       adversarial () ; }
                    advtimeend=0;
                    advtimestart=0;
                    dmtimeend1=0;
                    dmtimestart1=0;
                    if (breakit==true) {
                       breakit=false;
                       break;
                    }
                    if (rootnode==0) {
                       if (gap<= epsilon && mipgap<=0.01) {
                          break;
                       }
                       else if (mm==maxiteration) {     System.out.println("Max iteration
                           reached");} }
                 }
              } //Instances
           } //Rcap
        } //m0
     } //Setup costs
  } //Gammas
} //Returns
}

public static void adversarial () throws FileNotFoundException {
advtimestart = System.currentTimeMillis();
double[] x0_f = new double [t+1]; //Assembled items
double[][] hc_f = new double [t+1][comp+1]; //Forward inventory
double[][] y_f = new double [t+1][comp+1]; //Setup
//Fixed Variables for the Adversarial Problem
for (int i=1; i<=t; i++) {
   x0_f[i] =x0_set[i];
   for (int cc=1; cc<=comp; cc++) {
      hc_f[i][cc] =hc_set[i][cc];
      hc_f[0][cc] = 0; // Initial inventory is zero.
```

```
      y_f[i][cc]=y_set[i][cc];
   }
}
try {
   IloCplex cplex = new IloCplex();
   //Variables
   IloNumVar pi_adv = cplex.numVar(0, Double.MAX_VALUE, IloNumVarType.Float, "pi_adv");
   IloNumVar[][] I = new IloNumVar[t+1][];
   for (int i = 0; i <= t; i++) {
      I[i] = cplex.numVarArray(comp+1, 0, Double.MAX_VALUE);
      for(int c=1; c<=comp; c++) {
         I[i][c].setName("I+"+i+","+c);
      }
   }
   IloNumVar[][] xr = new IloNumVar[t+1][];
   for (int i = 1; i <= t; i++) {
      xr[i] = cplex.numVarArray(comp+1, 0, Double.MAX_VALUE);
      for(int c=1; c<=comp; c++) {
         xr[i][c].setName("xr"+i+","+c);
      }
   }
   IloNumVar[][] xm = new IloNumVar[t+1][];
   for (int i = 1; i <= t; i++) {
      xm[i] = cplex.numVarArray(comp+1, 0, Double.MAX_VALUE);
      for(int c=1; c<=comp; c++) {
         xm[i][c].setName("xm"+i+","+c);
      }
   }
   IloNumVar[] zr = cplex.numVarArray(t+1, -1, 1);
   IloNumVar[] b = cplex.numVarArray(t+1, 0, 1);
   IloIntVar[][] a = new IloIntVar[t+1][];
   for (int i = 1; i <= t; i++) {
      a[i] = cplex.boolVarArray(comp+1);
      for(int c=1; c<=comp; c++) {
         a[i][c].setName("a"+i+","+c);
      }
   }
```

```
//Objective Function
IloLinearNumExpr obj = cplex.linearNumExpr();
obj.addTerm(1, pi_adv);
cplex.addMaximize(obj);
//Constraints
Arrays.fill(bigM_adv_rets,0);
Arrays.fill(bigM_adv_dems,0);
double rhs4=0;
for (int i=1; i<=t; i++) {
   bigM_adv_rets[i]=bigM_adv_rets[i-1]+rbar[i]+rcap[i];
   bigM_adv_dems[i]=bigM_dm_setup[i];
}
System.out.print("Bigmadvs= ");
for (int tm=1; tm<=t; tm++) {
   System.out.print(bigM_adv_rets[tm]+",");
}
//-1- Pi Constraint
IloLinearNumExpr lhs1 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int c=1; c<= comp; c++) {
      lhs1.addTerm(-chr[c],I[i][c]);
      lhs1.addTerm(-m[c],xm[i][c]);
      lhs1.addTerm(-r[c],xr[i][c]);
   }
}
lhs1.addTerm(1,pi_adv); //...more variables (LHS)
IloRange con1 = cplex.addEq(lhs1,0);
con1.setName("Pi_Constraint");
lhs1.clear();
//-2- Production quantities should equal "demand from upper level"
IloLinearNumExpr lhs2 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int c=1; c<= comp; c++) {
      lhs2.addTerm(1, xr[i][c]);
      lhs2.addTerm(1, xm[i][c]);
      IloRange con2 = cplex.addEq(lhs2,x0_set[i]+hc_set[i][c]-hc_set[i-1][c]);
      con2.setName("Production_plan_"+i+"_"+c);
```

```
      lhs2.clear();
   }
}
lhs2.clear();
//-3- Initial inventories are zero
IloLinearNumExpr lhs3 = cplex.linearNumExpr();
for(int c=1; c<= comp; c++) { //sum 1..comp
   lhs3.addTerm(1, I[0][c]);
}
IloRange con3 = cplex.addEq(lhs3,0);
con3.setName("Initial_inventories_are_zero");
lhs3.clear();
//-4- Balance constraint
IloLinearNumExpr lhs4 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) { //sum i=1 ... T
   for(int c=1; c<=comp; c++) {
      lhs4.addTerm(1,I[i-1][c]);
      lhs4.addTerm(rcap[i],zr[i]);
      lhs4.addTerm(-1,I[i][c]);
      lhs4.addTerm(-1,xr[i][c]);
      rhs4+=-rbar[i];
      IloRange con4 = cplex.addEq(lhs4,rhs4);
      con4.setName("InvBalance_"+i);
      rhs4=0;
      lhs4.clear();
   }
}
IloLinearNumExpr lhs7 = cplex.linearNumExpr();
for(int i=1; i <= t; i++) {
   for(int c=1; c<=comp; c++) {
      lhs7.addTerm(bigM_adv_rets[i],a[i][c]);
      lhs7.addTerm(-1,I[i][c]);
      IloRange con7 = cplex.addGe(lhs7,0);
      con7.setName("act=1_"+i);
      lhs7.clear();
   }
}
```

```
IloLinearNumExpr lhs8 = cplex.linearNumExpr();
for(int c=1; c<=comp; c++) {
   for(int i=1; i <= t; i++) {
      lhs8.addTerm(-bigM_adv_dems[i],a[i][c]);
      lhs8.addTerm(-1, xm[i][c]);
      IloRange con8 = cplex.addGe(lhs8,-bigM_adv_dems[i]);
      con8.setName("act=0_"+i);
      lhs8.clear();
   }
}
IloLinearNumExpr lhs9_1 = cplex.linearNumExpr();
for (int c=1; c<=comp; c++) {
   for (int j=1; j<=t; j++) {
      for (int i=1; i<=j; i++){
         lhs9_1.addTerm(1,b[i]);
      }
      cplex.addLe(lhs9_1, gamma[j][c]);
      lhs9_1.clear();
   }
}
IloLinearNumExpr lhs9_2 = cplex.linearNumExpr();
for (int j=1; j<=t; j++) {
   lhs9_2.addTerm(-1,b[j]);
   lhs9_2.addTerm(1, zr[j]);
   cplex.addLe(lhs9_2, 0);
   lhs9_2.clear();
}

IloLinearNumExpr lhs9_3 = cplex.linearNumExpr();
for (int j=1; j<=t; j++) {
   lhs9_3.addTerm(-1,b[j]);
   lhs9_3.addTerm(-1, zr[j]);
   cplex.addLe(lhs9_3, 0);
   lhs9_3.clear();
}
IloLinearNumExpr lhs10 = cplex.linearNumExpr();
for (int c=1; c<=comp; c++) {
```

```
   for (int i=1; i<=t; i++) {
      lhs10.addTerm(1,xm[i][c]);
      lhs10.addTerm(1,xr[i][c]);
      cplex.addLe(lhs10, bigM_adv_dems[i]*y_f[i][c]);
      lhs10.clear();
   }
}
cplex.setParam(IloCplex.Param.MIP.Limits.Solutions,1);
firstsltnmipgap_ap=0;
cplex.setOut(null);
if (cplex.solve()){
   double bestobj_ap=cplex.getBestObjValue();
   double bestint_ap=cplex.getObjValue();
   firstsltnmipgap_ap=round(((bestobj_ap-bestint_ap)/bestint_ap)*100,2); }
cplex.setParam(IloCplex.Param.MIP.Limits.Solutions,
    cplex.getDefault(IloCplex.Param.MIP.Limits.Solutions)); //Back to default
File advlog = new File("MIPgap_" + initialmipgap + "_MLCR_ADVLOG_" + ReturnChoice +
    GammaChoice + SetupChoice + m0Choice + rcapChoice + t + "_" + comp + ".txt");
PrintWriter advwrite = new PrintWriter(new BufferedWriter(new FileWriter("MIPgap_" +
    initialmipgap + "_MLCR_ADVLOG_" + ReturnChoice + GammaChoice + SetupChoice +
    m0Choice + rcapChoice + t + "_" + comp + ".txt" , true)));
advwrite.println(" ");
advwrite.print("INSTANCE "+runmanytimes+" ITERATION "+numberofiterations+" ");
advwrite.close();
FileOutputStream oFile = new FileOutputStream(advlog, true);
cplex.setOut(oFile);
cplex.setParam(IloCplex.Param.TimeLimit,
    (Math.max((timelim-(dmtime_ineachrun+advtime_ineachrun)/1000),0)));
if (cplex.solve()){
   optimal_pi_ADV=cplex.getValue(pi_adv);
   advtimeend = System.currentTimeMillis();
   totaladvtime=totaladvtime+ advtimeend - advtimestart;
   advstatus=" "+cplex.getStatus()+" ";
   if (numberofiterations==1) {
      ADVub=(cplex.getValue(pi_adv)+prodcosts);
   }
   ADVub = Math.min ( ADVub , (cplex.getValue(pi_adv) + prodcosts ));
```

```
double localgap=0, globalgap=0;
localgap=(((cplex.getValue(pi_adv)+prodcosts)-dm_local_objective)/dm_local_objective);
globalgap=((ADVub-dm_local_objective)/dm_local_objective);
try {
   PrintWriter outcases = new PrintWriter(new BufferedWriter(new
       FileWriter("MIPgap_" + initialmipgap + "_" + ReturnChoice + GammaChoice +
       SetupChoice + m0Choice + rcapChoice + t + "_" + comp + ".txt", true)));

   if (numberofiterations==1 && runmanytimes==1) {
      outcases.write("RUN ");
      outcases.write("ITERATION ");
      outcases.write("DMP_OBJ ");
      outcases.write("GLOBAL_UB ");
      outcases.write("LOCAL_UB ");
      outcases.write("PI_ADV ");
      outcases.write("GAP ");
      outcases.write("FINAL_NODE_SIZE ");
      outcases.write("TIME_TAKEN_DM ");
      outcases.write("TIME_TAKEN_ADV ");
      outcases.write("FINAL_MIP_GAP ");
      outcases.write("FRSTSLTN_MIP_GAP(DMP) ");
      outcases.write("FRSTSLTN_MIP_GAP(AP) ");
      outcases.write("DM_STATUS ");
      outcases.write("ADV_STATUS ");
      outcases.println(" ");}
   outcases.print(runmanytimes+" ");
   outcases.print(numberofiterations+" ");
   outcases.print(dm_local_objective+" ");
   outcases.print(ADVub+" ");
   outcases.print((cplex.getValue(pi_adv)+prodcosts)+" ");
   outcases.print(optimal_pi_ADV+" ");
   outcases.print(gap+" ");
   outcases.print(finalnodesize1+" "); //Print final node size on DM.
   outcases.print(((double)dmtimeend1 - (double)dmtimestart1)/1000+" ");
   outcases.print((((double)advtimeend - (double)advtimestart)/1000)+" ");
   outcases.print(finalmipgap+" ");
   outcases.print(firstsltnmipgap_dmp+" ");
```

```
            outcases.print(firstsltnmipgap_ap+" ");
            outcases.print(dmstatus+" ");
            outcases.print(advstatus+" ");
            advtime_ineachrun+=advtimeend - advtimestart;
            outcases.print("FINAL_GAP :"+mipgap+" ");
            outcases.println(numberofiterations+" ");
            outcases.close();
         } catch (IOException e) {}
         dmtimeend1=0;
         dmtimestart1=0;
         advtimeend=0;
         advtimestart=0;
         for (int i=1; i<=t; i++) {
            zr_set[i][numberofiterations]=cplex.getValue(zr[i]);
         }
         cplex.end(); } else {
         System.out.println("InfeasibleADV"); }}
   catch (IloException exc) {
      exc.printStackTrace(); }
   catch (IOException e1) {
      e1.printStackTrace(); }}

   public static void decisionmakers () throws FileNotFoundException {
   dm_local_objective=0;
   try {
      IloCplex cplex = new IloCplex();
      String[] HSname = new String [t+1];
      String[] x0name = new String [t+1];
      //Decision Variables
      for (int names=0; names<=t; names++) {
         HSname[names] = "HS" + names;
         x0name[names] = "x0" + names;
      }
      IloNumVar[] x0 = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
          x0name);
      IloNumVar[] HS = cplex.numVarArray (t+1,0,Double.MAX_VALUE, IloNumVarType.Float,
          HSname);
```

```
IloNumVar pi_dm = cplex.numVar(0, Double.MAX_VALUE, IloNumVarType.Float, "pi_dm");
IloNumVar[][][] xm = new IloNumVar[t+1][comp+1][];
for (int tt = 1; tt <= t; tt++) {
   for (int c=1; c<= comp; c++) {
      xm[tt][c] = cplex.numVarArray(numberofiterations+1, 0, Double.MAX_VALUE,
          IloNumVarType.Float);
      for (int s=0; s<=numberofiterations-1; s++) {
         xm[tt][c][s].setName("xm"+tt+","+c+","+s);
      }
   }
}
IloNumVar[][][] xr = new IloNumVar[t+1][comp+1][];
for (int tt = 1; tt <= t; tt++) {
   for (int c=1; c<= comp; c++) {
      xr[tt][c] = cplex.numVarArray(numberofiterations+1, 0, Double.MAX_VALUE,
          IloNumVarType.Float);
      for (int s=0; s<=numberofiterations-1; s++) {
         xr[tt][c][s].setName("xr"+tt+","+c+","+s);
      }
   }
}
IloNumVar[][] y = new IloNumVar[t+1][];
for (int tt = 1; tt <= t; tt ++) {
   y[tt] = cplex.boolVarArray(comp+1);
   for (int cc=1; cc<=comp; cc++) {
      y[tt][cc].setName("y"+tt+","+cc);
   }
}
IloNumVar[][] HC = new IloNumVar[t+1][];
for (int tt = 0; tt <= t; tt ++) {
   HC[tt] = cplex.numVarArray(comp+1, 0, Double.MAX_VALUE);
   for (int cc=1; cc<=comp; cc++) {
      HC[tt][cc].setName("HC"+tt+","+cc);
   }
}
//Objective
IloLinearNumExpr obj = cplex.linearNumExpr();
```

```
obj.addTerm(1, pi_dm);
for(int i=1; i<=t; i++){
   obj.addTerm(m0, x0[i]);
   obj.addTerm(chs, HS[i]);
   for(int lv=1; lv<=comp; lv++) {
      obj.addTerm(K[lv], y[i][lv]);
      obj.addTerm(chc[lv], HC[i][lv]);
   }
}
cplex.addMinimize(obj);
cplex.addEq(HS[0], 0);
for (int lvls=1; lvls<=comp; lvls++) {
   cplex.addEq(HC[0][lvls],0); }
//-1- Pi constraint
IloLinearNumExpr lhs1 = cplex.linearNumExpr();
double rhs1=0;
for(int sc=0; sc<=numberofiterations-1; sc++) {
   for (int tt=1; tt<=t; tt++) {
      for (int c=1; c<=comp; c++) {
         for (int i=1; i<=tt; i++) {
            lhs1.addTerm(chr[c], xr[i][c][sc]);
                rhs1=rhs1+(chr[c]*(rbar[i]+(rcap[i]*zr_set[i][sc]))); }
         lhs1.addTerm(-m[c], xm[tt][c][sc]);
         lhs1.addTerm(-r[c], xr[tt][c][sc]); }}
   lhs1.addTerm(1, pi_dm);
   IloRange con1 = cplex.addGe(lhs1, rhs1);
   lhs1.clear();
   rhs1=0;
}
//-2- Inventory balance for components
IloLinearNumExpr lhs2 = cplex.linearNumExpr();
for(int lvls=1; lvls<=comp; lvls++) {
   for (int i=1; i<=t; i++) {
      for (int sc=0; sc<=numberofiterations-1; sc++){
         lhs2.addTerm(1, HC[i-1][lvls]);
         lhs2.addTerm(1, xr[i][lvls][sc]);
         lhs2.addTerm(1, xm[i][lvls][sc]);
```

```
            lhs2.addTerm(-1, x0[i]);
            lhs2.addTerm(-1, HC[i][lvls]);
            IloRange con2 = cplex.addEq(lhs2, 0);
            lhs2.clear();
         }
      }
   }
   //-3- Inventory balance for end item
   IloLinearNumExpr lhs3 = cplex.linearNumExpr();
   for (int i=1; i<=t; i++) {
      lhs3.addTerm(1, HS[i-1]);
      lhs3.addTerm(1, x0[i]);
      lhs3.addTerm(-1, HS[i]);
      IloRange con3 = cplex.addEq(lhs3, dbar[i]);
      lhs3.clear();
   }
   double sumofdemands=0;
   for (double j : dbar) {
      sumofdemands=sumofdemands+j; }
   double sumofrets=0;
   for (int tt=1; tt<=t; tt++){
      sumofrets=sumofrets+rbar[tt]+rcap[tt];
      bigM_dm_setup[tt]=Math.max(sumofrets, sumofdemands);
      sumofdemands=sumofdemands-dbar[tt]; }
   //-4- Setup constraint
   IloLinearNumExpr lhs4 = cplex.linearNumExpr();
   for (int j=1; j<=t; j++) {
      for (int c=1; c<=comp; c++) {
         for (int sc=0; sc<=numberofiterations-1; sc++){
            lhs4.addTerm(bigM_dm_setup[j], y[j][c]);
            lhs4.addTerm(-1, xm[j][c][sc]);
            lhs4.addTerm(-1 , xr[j][c][sc]);
            cplex.addGe(lhs4, 0);
            lhs4.clear(); }}}
   //-5- Remanufacturing
   IloLinearNumExpr lhs5 = cplex.linearNumExpr();
   double rhs5=0;
```

```
for(int sc=0; sc<=numberofiterations-1; sc++) {
   for (int j=1; j<=t; j++) {
      for (int c=1; c<=comp; c++) {
         for (int ii=1; ii<=j; ii++) {
            lhs5.addTerm(1,xr[ii][c][sc]);
                rhs5+=(rbar[ii]+(rcap[ii]*zr_set[ii][sc]));
         }
         cplex.addLe(lhs5, rhs5);
         rhs5=0;
         lhs5.clear(); }}}
/*Setting branching priorities*/
int mypri=t*comp;
for(int pri2=1; pri2<=comp; pri2++){
   for(int pri=1; pri<=t; pri++){
      cplex.setPriority(y[pri][pri2], mypri);
      mypri--; }}
cplex.setParam(IloCplex.Param.Emphasis.MIP, 3);
if (rootnode == 0) {
   firstsltnmipgap_dmp=0;
   cplex.setOut(null);
   cplex.setParam(IloCplex.Param.MIP.Limits.Solutions,1);
   if (cplex.solve()){
      double bestobj_dmp=cplex.getBestObjValue();
      double bestint_dmp=cplex.getObjValue();
      firstsltnmipgap_dmp=round(((-bestobj_dmp+bestint_dmp)/bestint_dmp)*100,2);
   }
   dmtimestart1 = System.currentTimeMillis();
   if (numberofiterations>1) {cplex.readMIPStarts ( "dmpsltn_" +
       (numberofiterations-1) + "_run_" + runmanytimes + ".sol");}
   File dmlog = new File("MIPgap_" + initialmipgap + "_DMLOG_" + ReturnChoice +
       GammaChoice + SetupChoice + m0Choice + rcapChoice + t + "_" + comp + ".txt");
   PrintWriter dmwrite = new PrintWriter(new BufferedWriter(new FileWriter("MIPgap_"
       + initialmipgap + "_DMLOG_" + ReturnChoice + GammaChoice + SetupChoice +
       m0Choice + rcapChoice + t + "_" + comp + ".txt" , true)));
   dmwrite.println(" ");
   dmwrite.print("INSTANCE "+runmanytimes+" ITERATION "+numberofiterations+" ");
   dmwrite.close();
```

```
FileOutputStream o2File = new FileOutputStream(dmlog, true);
cplex.setOut(o2File);
cplex.setParam(IloCplex.Param.MIP.Tolerances.MIPGap, mipgap);
cplex.setParam(IloCplex.Param.TimeLimit,
    (Math.max((timelim-(dmtime_ineachrun+advtime_ineachrun)/1000),0)));
cplex.setParam(IloCplex . Param . MIP . Limits . Solutions , cplex . getDefault (
    IloCplex . Param . MIP . Limits . Solutions)); //Back to default
if (cplex.solve()){
   finalmipgap = cplex.getMIPRelativeGap();
   prodcosts = cplex.getObjValue()-cplex.getValue(pi_dm);
   DMobjective = cplex.getObjValue();
   optimal_pi_DM=cplex.getValue(pi_dm);
   dmtimeend1 = System.currentTimeMillis();
   dmtime_ineachrun+=dmtimeend1 - dmtimestart1;
   File mipstart_dm = new File("dmpsltn_" + (numberofiterations) + "_run_" +
       runmanytimes + ".sol");
   mipstart_dm.delete();
   cplex.writeMIPStarts("dmpsltn_" + (numberofiterations) + "_run_" + runmanytimes
       + ".sol");
   File mystart= new File("dmpsltn_" + (numberofiterations) + "_run_" +
       runmanytimes + ".sol");
   String search = "MIPStartEffortLevel=\"0\"";
   String replace = "MIPStartEffortLevel=\"2\"";
   try{
      FileReader fr = new FileReader(mystart);
      String s;
      String totalStr = "";
      try (BufferedReader br = new BufferedReader(fr)) {
         while ((s = br.readLine()) != null) {
            totalStr += s + "\n";
         }
         totalStr = totalStr.replaceAll(search, replace);
         FileWriter fw = new FileWriter(mystart);
         fw.write(totalStr);
         fw.close();
      }
   }catch(Exception e){
```

```
            e.printStackTrace();
         }
         finalnodesize1=cplex.getNnodes();//final node size
      }else{
         System.out.println("InfeasibleDM");
         breakit=true; }
      dm_local_objective=cplex.getObjValue();
      totaldmtime=totaldmtime+dmtimeend1 - dmtimestart1;
      for (int i=1; i<=t; i++) {
         x0_set[i]=cplex.getValue(x0[i]);
         for (int cc=1; cc<=comp ; cc++) {
            hc_set[i][cc]=cplex.getValue(HC[i][cc]);
            y_set[i][cc]=cplex.getValue(y[i][cc]);}}
      cplex.end();}}
catch (IloException exc) {
   exc.printStackTrace();}
catch (IOException e) {
   e.printStackTrace();}}
private static double round (double value, int precision) {int scale = (int)
    Math.pow(10, precision);
return (double) Math.round(value * scale) / scale;}}
```

A.3 Instance Generation (LSR-R)

```
import java.util.*;
import java.io.*;
import org.apache.commons.math3.distribution.*;
public class Generate {
public static void main(String[] args) throws Exception, Exception {
List<String> ReturnList = Arrays.asList("High", "Med", "Low");
List<String> GammaList = Arrays.asList("High", "Med", "Low");
List<String> SetupList = Arrays.asList("VHigh", "High", "Med", "Low");
List<String> DisposalList = Arrays.asList("Greater", "Less");
String ReturnChoice="", GammaChoice="", SetupChoice="", DisposalChoice="";
for (int mm=1; mm<=ReturnList.size(); mm++){
   for (int j=1; j<=GammaList.size(); j++){
      for (int k=1; k<=SetupList.size(); k++){
```

```
for (int l=1; l<=DisposalList.size(); l++){
   ReturnChoice=ReturnList.get(mm-1);
   GammaChoice=GammaList.get(j-1);
   SetupChoice=SetupList.get(k-1);
   DisposalChoice=DisposalList.get(l-1);
   int T=50;
   double[] dbar = new double [T+1]; //Nominal demand (d bar)
   double[] rbar = new double [T+1]; //Nominal return (r bar)
   double[] dcap = new double [T+1]; //Max. deviation of demand (d cap)
   double[] rcap = new double [T+1]; //Max. deviation of return (r cap)
   double[] d_gamma = new double [T+1]; //Gamma for demands
   double[] r_gamma = new double [T+1]; //Gamma for returns
   double K,chs,chr,b,d,m,r; //Costs
   //Change these parameters to redefinine Low, Med, High
   double demand_upper=100, demand_lower=50; //Upper/lower limits for dbar
   double demand_deviation=(demand_upper-demand_lower)/2;
   double demand_nominal=demand_lower+demand_deviation;
       double K_Vupper_percentage=200, K_upper_percentage=5, K_med_percentage=2,
          K_lower_percentage=0.1;
   double Gamma_e_upper=0.1;
   double Gamma_e_med=0.05;
   double Gamma_e_lower=0.01;
   double r_percentage_upper=0.7; //Upper value for r_percentage
   double r_percentage_med=0.5; //Med value for r_percentage
   double r_percentage_lower=0.3; //Lower value r_percentage

   double chs_upper=10; //Upper limit for servicables' holding cost (hs)
   double chs_lower=5; //Lower limit for servicables' holding cost (hs)
   double d_cap_percentage=0.1; //What % of nominal demand is going to be dcap?
   double r_percentage=0; //What % of demand should be considered as those of
      returns?
   double m_percentage=2; //m=hs*m_percentage
   double rmf_percentage=2; //r=hr*rmf_percentage
   double dis_percentage=2; //if DisposalChoice = "Greater" d=r*dis_percentage
      else, d=r/dis_percentage
   double returns_holding_cost_percentage=0.1;
      //hr=hs*returns_holding_cost_percentage
```

```
double backlog_cost_percentage=4; //b=hs*backlog_cost_percentage
double r_cap_percentage=0.1; // Percentage of returns' deviation
double e_demand=0;
double e_return=0;
double z_demand=0;
double z_return=0;

if(ReturnChoice=="Low") {
   r_percentage=r_percentage_lower; }
if(ReturnChoice=="Med"){
      r_percentage=r_percentage_med; }
if(ReturnChoice=="High"){
         r_percentage=r_percentage_upper;
      }

if(GammaChoice=="Low") {
   e_demand=Gamma_e_lower;
   e_return=Gamma_e_lower;}
if(GammaChoice=="Med"){
      e_demand=Gamma_e_med;
      e_return=Gamma_e_med;}
if(GammaChoice=="High"){
         e_demand=Gamma_e_upper;
         e_return=Gamma_e_upper; }

for (int generatemany=1; generatemany<=5; generatemany++) {
   //Find gamma
   NormalDistribution normdist = new NormalDistribution();
       z_demand=normdist.inverseCumulativeProbability(1-e_demand);
       z_return=normdist.inverseCumulativeProbability(1-e_return);
   for (int t=1; t<=T; t++){
       d_gamma[t]=Math.min(t,Math.ceil((z_demand*Math.sqrt(t))+1));
       r_gamma[t]=Math.min(t,Math.ceil((z_return*Math.sqrt(t))+1)); }
   for (int i=1; i<=T; i++) {
      Random rdemand =new Random();
      Random rreturn =new Random();
```

```
   dbar[i] =demand_lower + (demand_upper - demand_lower) *
       rdemand.nextDouble();
   rbar[i] =(demand_lower*r_percentage) + (r_percentage*(demand_upper -
       demand_lower)) * rreturn.nextDouble();
   dcap[i]=d_cap_percentage*dbar[i];
   rcap[i]=r_cap_percentage*rbar[i];
}
Random storagecost =new Random();
chs=(chs_lower) + (chs_upper - chs_lower)*storagecost.nextDouble();
    chr=chs*returns_holding_cost_percentage;
b=chs*backlog_cost_percentage;
m=chs*m_percentage;
r=chr*rmf_percentage;
if(DisposalChoice=="Greater") {
   d=r*dis_percentage; }
else {
   d=r/dis_percentage;}
if(SetupChoice=="Low") {
    K=Math.ceil(chs*(demand_nominal-demand_deviation)*K_lower_percentage);}
      if(SetupChoice=="Med"){
          K=Math.ceil(chs*(demand_nominal)*K_med_percentage);}
      if(SetupChoice=="High"){ K=Math.ceil(chs * (demand_nominal +
          demand_deviation) * K_upper_percentage); }
      if(SetupChoice == "VHigh") { K=Math.ceil(chs * (demand_nominal +
          demand_deviation) * K_Vupper_percentage);}

for (int t=1; t<=T; t++) {
   try {
      File file = new File("C:\\ Users \\npb15184 \\Desktop \\LSRData_Rev
          \\"+ ReturnChoice +GammaChoice +SetupChoice +DisposalChoice
          +"\\DataR" +ReturnChoice +"Gamma" +GammaChoice +"K" +SetupChoice
          +"Disposal" +DisposalChoice +generatemany +".txt");
      file.getParentFile().mkdirs();
      PrintWriter outcases = new PrintWriter(new BufferedWriter(new
          FileWriter(file,true)));
      if (t==1) {     outcases.println("Dataset for LSR problem with joint
          setup costs and T=200");
```

```
outcases.println("(Uniformly distributed)
    Demand=["+demand_lower+","+demand_upper+"]");
outcases.println("(Uniformly distributed)
    Returns=["+r_percentage+"*"+demand_lower+" ,
    "+r_percentage+"*"+demand_upper+"] ("+ReturnChoice+")");
   outcases.println("K = "+K+" ("+SetupChoice+")");
outcases.println("(Uniformly distributed) chs =
    ["+chs_lower+","+chs_upper+"]");
outcases.println("chr = "+returns_holding_cost_percentage+"*chs");
outcases.println("b = "+backlog_cost_percentage+"*chs");
   outcases.println("m = "+m_percentage+"*chs");
   outcases.println("r = "+rmf_percentage+"*chr");
   outcases.println("Disposal cost (d) is "+DisposalChoice+" than
      the remanufacturing cost.");
   outcases.println("Disposal cost is calculated as
      d=r*"+dis_percentage+ " (when disposal costs are greater)
      "+"d=r/"+dis_percentage+" otherwise.");
   outcases.println("Dcap and Rcap are "+d_cap_percentage+" and
      "+r_cap_percentage+" times the nominal demand value
      respectively (for each time period)");
   outcases.println("Gamma_D and Gamma_R have a probability of
      "+e_demand+" for generating a better solution than the actual
      cost realization ("+ GammaChoice + ")");
   outcases.println(" ");
   outcases.write("t ");
   outcases.write("NominalDemand ");
   outcases.write("NominalReturn ");
   outcases.write("DemandMaxDeviation ");
   outcases.write("ReturnMaxDeviation ");
   outcases.write("SetupCost ");
   outcases.write("Chs ");
   outcases.write("Chr ");
   outcases.write("BackloggingCost ");
   outcases.write("GammaD ");
   outcases.write("GammaR ");
   outcases.write("ManufacturingCost ");
   outcases.write("RemanufacturingCost ");
```

```
            outcases.write("DisposalCost ");
            outcases.println(" ");
        }
             outcases.print(t+" ");
        outcases.print(dbar[t]+" ");
        outcases.print(rbar[t]+" ");
        outcases.print(dcap[t]+" ");
        outcases.print(rcap[t]+" ");
        outcases.print(K+" ");
        outcases.print(chs+" ");
        outcases.print(chr+" ");
        outcases.print(b+" ");
        outcases.print(d_gamma[t]+" ");
        outcases.print(r_gamma[t]+" ");
        outcases.print(m+" ");
        outcases.print(r+" ");
        outcases.println(d+" ");

        outcases.close();
    } catch (IOException e) {}}}}}}}}
```