POLITECNICO DI TORINO

III Facoltà d'Ingegneria

Mathematical Engineering

Master's Thesis

Coordinated Lot-Sizing Problem under Demand Uncertainty

(Stochastic Programming approach)

Director:

Prof. Paolo Brandimarte

Candidate:

Faezeh Seyri

December 2018

Dedication

To my Father...

Abstract

Lot-sizing problems are production/procurement planning problems with different types of setups between production lots. These setups are often too costly so it is not reasonable to produce in every period. On the other hand, having fewer setups by producing/ordering large quantities to satisfy future demands will result in a high inventory holding costs. Thus, the objective is to determine the periods where production/order should take place, and the quantities to be produced/ordered to satisfy demands while minimizing setup and inventory holding costs.

In this project we study an extension version, the Coordinated Lot-Sizing Problem (or Joint Replenishment Problem), where a family of different items shares a common production line, supplier, or a transportation mode etc. Hence a joint cost is incurred whenever at least one item of the family is replenished. We consider an uncapacitated variant of the problem, but the main and important issue we deal with in our study is uncertainty in items demand, so that planning decisions must be taken knowing only the demands in the first time period, while future demands have some degree of uncertainty.

We firstly described the deterministic version of the problem, and evaluated the performance of one of the well-known heuristic solving procedures, Forward-Pass, in both static and rolling-horizon framework. Then by adding uncertainty, we implemented our problem as a mixed-integer multi-stage Stochastic Programming optimization model, taking into account demand uncertainty explicitly through a scenario tree.

First we tried to solve the Deterministic Equivalent of the Stochastic Programming model by *Gurobi*, a state-of-the-art optimization solver, to compare the performance of different structure scenario trees and also to explore the maximum size tree (in both branching structure and number of items in the family) that is solvable in a reasonable amount of CPU time. We observed that even the best and most mature solvers will struggle even with not very large-size trees, so we may rely on alternative solving strategies such as scenario decomposition that enables large-scale problems to be efficiently solved. We applied the Progressive Hedging Algorithm (PHA) which is a general scenario-based decomposition method to cope with hard mixed-integer multi-stage stochastic programs, to our stochastic coordinated lot-sizing model. Our computational results show that the PH-based method we applied, is efficient and its solution quality is comparative to the solution obtained by *Gurobi* optimizer.

Finally, we assessed the convenience and any competitive advantage of stochastic programming approach with respect to using the methods based on the approximated values or expected value of uncertain demand.

Table of Contents

Introduction

One of the critical issues in MRP/ERP systems is the support in developing a good Master Production Schedule (MPS). In principle, Mixed-Integer Linear Programming (MILP) models, like variants of the lot-sizing problems could be helpful (Brandimarte 2006).

Lot-sizing Problems (LSP) have been an active research area starting from the seminal work of Wagner & Whitin (1958). They refer to the production planning problems with different types of setups between production/order lots. Because of these setups, it is often too costly to produce/purchase a given product in every period; on the other hand, having fewer setups by producing large quantities to satisfy future demands will result in a high inventory holding costs. Obtaining a cost-efficient plan, balancing the trade-off between setups and inventory costs –lot-sizing– has been extensively studied since the beginning of industrialization. Thus, the LSP objective is to find an optimal production/purchase plan that determines the periods where production/order should occur, and the quantities to be produced/ordered to satisfy demands while minimizing the total system costs, which includes in its general form:

- Setup cost: a fixed charge for each product that is generated every time production/order takes place.
- Inventory holding cost: the cost of holding a unit of each product per period.
- Penalty cost: a punishment for lost sales.

There are several ways to classifying the lot-sizing problems and certainly the complexity of a LSP depends on the features taken into account by the model. Karimi *et al*. (2003) explained the characteristics affect the classifying, modelling and the complexity of lot-sizing decisions, including:

- Planning horizon: which is the time interval on which the master production schedule extend into the future;
- Number of levels: whether single-level or multi-level;
- Number of products;
- Capacity or resource constraints: include manpower, equipments, machines, budget, etc.;
- Determination of items that influence the restrictions in the inventory holding time;
- Demand nature;
- Setup structure.

In general, the lot-sizing problems (LSP) are classified according to three factors:

- Single or multiple items;
- Capacitated or uncapacitated replenishment quantities;

— Joint or independent setup cost structure.

Then several variations of each category can be generated depending on how demand is characterized. Static demand problems assume a *stationary* or constant demand pattern while *dynamic* demand problems permit demand to vary during the time. If all demand values are known for the duration of the planning horizon the demand stream is defined *deterministic*, otherwise, is considered *stochastic.*

The single-item uncapacitated lot-sizing problem is the quite simplest version of LSP. For finite-horizon problems with deterministic demand, the optimal method was first developed by Wagner & Whitin (1958) that is well-known as the classical dynamic lot-sizing algorithm. They developed a dynamic programming algorithm of $O(T^2)$ for an uncapacitated model where T is the length of the planning horizon, when the production and holding costs are linear, and the unit production costs, unit holding costs and the setup costs are the same in all periods. The algorithm can give optimal solution in reasonable amount of run times when the number of periods is not large. However, the WW algorithm is quite limited when the finite horizon is long; because a dynamic programming based algorithm requires a tremendous amount of central processing unit (CPU) time. To fix, this problem, several dynamic lot size heuristics are developed.

Coordinated Lot-sizing Problem (CLSP) or Joint Replenishment Problem (JRP) is a variant of the classical lot-sizing problem and it is referred to the decision planning problems when a family of items shares a common production line, supplier, or a mode of transportation, etc. In these situations the coordination of shared and often limited, resources across items is economically attractive because it is often too costly to produce/order a given product family in every period. So a firm's replenishment policy is a major driver of its operational investments and costs and the ability to meet its customer service objectives. In the joint replenishment problems a major setup cost is incurred each time one or more items in the product family are replenished, and a minor setup cost is charged for each item replenished. Because of their practical importance, replenishment lot-sizing problems are among the most widely researched areas in operations and supply chain management. However, due to their computational complexity, several important problem classes remain unsolved.

In this work, we focus on the *uncapacitated* version of CLSP. Considering a single-family of different item types, we cope the problem that aims in its simplest form, to find the time-phased replenishment schedule that minimizes replenishment, inventory, and penalty costs, while it is formulated in a mixed-integer optimization problem. Although Imposing capacity constraints will make the model better fit to a manufacturing environment, we assume that there is no capacity restrictions; in such way, the family order replenishments can take place whenever we want having whatever amount of each item type.

The main issue we deal with in CLSP under consideration is *demand uncertainty*. Using the notation same as in Manerba *et al*. (2017), we denote by CLSP_{ud} our Stochastic Coordinated Lot-Sizing Problem, where "ud" refers to demand uncertainty. Our objective is to approach the problem by Stochastic Programming (SP) techniques, and then to assess the convenience and any

competitive advantage of our stochastic model with respect to use the simpler methods based on the approximation or expected value of uncertain demands.

When dealing with Stochastic Programming (SP) models, the most important point is how to represent the future uncertainty of the parameters. In the present study, demand uncertainty is explicitly modeled through a *scenario tree* which results in a multi-stage mixed-integer stochastic programming model. While some efficient solution methods for multi-stage linear programming SPs have been developed in the stochastic programming literature, most of these solution methods are not efficient when applied to the multi-stage stochastic *integer* programming problem. This is because of the non-convexities caused by the integer decision variables (Solak 2007).

Recent works have shown the importance of explicitly incorporate demand uncertainty in the economic setting, along with the evidence about the computational burden of solving the relative stochastic programming models for a sufficiently large number of scenarios (Manerba *et al*. 2017). Although with the latest optimization algorithms realistic sized deterministic optimization problems can be successfully solved, however it is still a challenge to solve large scale stochastic programming problems when the number of scenarios is large and the underlying deterministic problem is already large scale and *NP*-complete. Since in order to determine the size of a SP model, the mathematical model dimensions and the number of scenarios need to be considered, it might be interesting to assess a $CLSP_{ud}$ in both aspects and find the practical sized SP models that are solvable in a reasonable amount of time by an optimization solver. To this aim, we have experienced and compared different $CLSP_{ud}$ models in terms of number of items in the family and the branching structure of the scenario tree, using a state-of-the-art optimization solvers, *Gurobi*. Our computational experiments showed that the time needed to solve the model depends strictly on the joint cost value relative to other costs and unfortunately, for some relative values of joint cost, even small-medium size models are not efficiently solved by *Gurobi*. This fact encouraged us to seek an efficient heuristic approach to overcome the computational burden of our stochastic mixed-integer model. To this aim, we tried to apply a progressive hedging-based algorithm, as a heuristic method to our $CLSP_{ud}$ model. Progressive Hedging Algorithm (PHA) is a scenario-based decomposition method to tackle with large scale SPs. It has been used to devise heuristics for hard mixed-integer multi-stage stochastic programs. The PHA decomposes the stochastic model by scenarios, and uses an Augmented Lagrangean Relaxation method to relax the complicated Non-Anticipativity (NA) constraints. Since, one of our objectives in the present study, is to evaluate the value of applying a stochastic model with respect to the simpler methods, the heuristic approach we applied to the stochastic model, must result in low optimality gap. Our computational results show that the PH-based algorithm is also effective.

The contents of this work are structured as follows:

— Chapter 1 starts out with the deterministic version of the problem, to acquaint the reader with the problem notation and many of the concepts to be used later on. We present a mixed-integer formulation which is the underlying formulation in our stochastic models in the subsequent chapters. After mentioning some interesting characteristics of the problem, we apply a well-known heuristic procedure to the deterministic problem and we assess its performance in both

static and rolling horizon frameworks. In this way, the reader may become familiar with rolling horizon strategy that will be used in the simulations of stochastic programming models in the next chapters. The other objective is to compare the performance of an optimal procedure to a heuristic one for the CLSP, in the rolling horizon environment.

— In chapter 2, by adding uncertainty to the problem, we firstly provide a short introductory to the Stochastic Programming models with recourse; we explain how to model the demand uncertainty explicitly; and then we present the stochastic coordinated lot-sizing problem ($CLSP_{ud}$) formulation with recourse. Finally we describe our scenario generation method.

— Chapter 3, contains the stochastic programming solution methods to our $CLSP_{ud}$. Firstly, we try to solve the Deterministic Equivalent (DE) of the $CLSP_{ud}$ optimally by the solver. We report different computational experiments to explore the maximum size problem instance and scenario tree which can be solved in a reasonable amount of running time by *Gurobi*. We also test the performance of SP models using different scenario tree structures (in terms of branching factor and time horizon length), running several simulations in a rolling horizon framework, by means of a history of true values for item demands (out-of-sample scenarios). Then we describe the progressive hedging based algorithm; we explain in details how we applied it to $CLSP_{ud}$. Eventually the computational results are presented to show the efficiency and the solution quality obtained by the PH-based algorithm.

— In chapter 4, we present a brief analysis to assess the convenience of our stochastic model, with respect to use the simpler methods based on the expected value of demands.

— We finish the present study by providing a brief recap of the main results.

Chapter 1

Deterministic Programming models

Deterministic programming is in fact the counterpart of the stochastic programming. Although our main objective in the present work, is to study the impact of demand uncertainty on the joint replenishment problem by means of Stochastic Programming (SP) models, we start this study by presenting some of the problem characterizations without considering any uncertainty in problem parameters, particularly in item demands.

In general, when demands are known for all periods in problem time horizon, we deal with the problem in a *static* planning framework. However, in practice the value of future demand is not always known for the whole length of the long-term activity but it reveals itself over time as customer orders firmed up (Narayanan *et al*. 2010). In this situations, one can make decisions using the rolling-horizon strategy. In the rolling horizon framework, solution will be a series of linked static planning problems implemented rolling through time (Blackburn and Millen 1982). That means, the replenishment schedules are determined using the available demand information and a subset of the earliest decisions are implemented. When it is time for the next planning cycle, the analyst incorporates recently updated demand and solves the replenishment schedule again. In this manner replenishment schedules are updated rolling through time. Indeed, rolling schedule procedures provide a mechanism for dealing with incomplete but deterministic demand data. The difference between optimal solution for the static problem, which assumes demand is known for the entire problem duration and the rolling schedule solution, provides a measure for the cost of limited demand information (Narayanan *et al*. 2010).

Working on a static or rolling-horizon planning framework, a natural division of the deterministic demand literature is based on whether demand is assumed to be constant (static) during the time, or allow them to be varied (dynamic) in each period.

In this chapter we study the coordinated ordering of different items of a single-family considering deterministic but time-varying demands for each item products, where there is a common or joint ordering cost for the family whenever one or more of these items are ordered, in addition to individual products ordering costs.

We firstly describe the Coordinated Lot-sizing Problem (CLSP) in its general form, then we present a Mixed-Integer Linear Programming (MILP) formulation which is the classical or natural formulation of the problem. After mentioning some properties of the model, one of the well-known heuristic methods, Forward-Pass, will be presented and its performance will be evaluated in both static and rolling horizon frameworks. Indeed, this deterministic procedure would be useful later, when we rely on scenario decomposition methods to solve our stochastic programming model. We will see in the progressive hedging which is a scenario-based decomposition methods, each scenario does not need to be solved to optimality and therefore, a good and quick deterministic heuristic procedure might be helpful. In addition, working in a rolling horizon environment, the reader will become familiar with this framework which is the underlying idea in the simulation process of the stochastic models in the subsequent chapters. Moreover, our objective is to see how well a heuristic procedure for CLSP performs in a rolling horizon framework, with respect to an optimal procedure.

1.1 Coordinated Lot-sizing Problem

The joint, also known as coordinated replenishment problems are commonly encountered in supply chain contexts such as when a family of items shares a production facility or a transportation shipment etc. In these cases, a joint fixed charge is incurred when any member of product family is replenished and a separate fixed cost is applied for each different item type replenished.
In Silver (1979) several examples of joint replenishment are provided, including scheduling a packaging line to produce various sizes and types of containers (production), purchasing multiple items from a common supplier (procurement), and shipping items that share a common mode of transportation (transportation). Flexible manufacturing system can be a great example where a joint setup cost is incurred when changing form one family to another and a tool adjustment cost for changing between item types within the product family. Robinson *et al*. (2004) describes the production of industrial lubricants, which requires a major setup cost each time a product family is made and a packaging line changeover cost for each different item produced. In these situations cost saving can be achieved by coordinating the replenishment of different items.
In the single-family Coordinated Lot-Sizing Problem (CLSP) or Joint Replenishment Problem (JRP), N different item types must be replenished to satisfy the demands in T different time periods. At each order, a subset of item types are jointly replenished and it generates two types of fixed costs:

- Major setup cost: a common ordering cost, associated with the coordinating activity;
- Minor setup costs: the individual ordering costs associated with each item type.

Hence, it makes economic sense to coordinate the individual item lot-size schedules such that the total replenishment costs are minimized.
In this work, we study the quite simplest version of this category of lot-sizing problems by considering the coordinated *ordering* of a single-family including different item types and infinite

capacity. Although adding capacity constraints based on labor, equipment, storage space, and other limiting factors extends the model to consider more practical contexts, we assume that no limit is imposed on order sizes or inventory levels. So in the present study we are dealing with a Single-family Coordinated Uncapacitated Lot-sizing Problem (CULSP), but we omit the notation related to uncapacitated and refer to the problem simply as CLSP. In addition, let us remark some notable assumptions:

- The replenishment orders are made at the beginning of each time period, in such way, no holding inventory cost is incurred for items consumed during each period;
- The lead times are negligible in such way the orders are delivered immediately;
- All kinds of costs are constant during the problem time horizon; but they may vary along different item types.
- Demand values and lot-size amounts express a magnitude that need not to be integer and they might be multiplied by any factor in the real practical situations.

The Coordinate Uncapacitated Lot-Sizing Problem is proved to be *NP*-complete by Arkin, *et al*. (1989) and Joneja (1990), implying that a polynomial-bound algorithm for solving the problem to optimality is unlikely to exist. However, numerous researchers seek to exploit the specialized mathematical structure of CLSP with specialized exact algorithms. These can be broadly classified into Branch-and-Bound algorithms and dynamic programming algorithms in which the computational times increases exponentially with problem size. Other specialized approaches include Branch-and-Cut algorithms and Dantzig Wolfe decomposition. Since the Exact algorithms of joint replenishment problem have not attained widespread applications because of their computational complexity and so inability to efficiently solve the real large-scale problems encountered in industry, try to develop the efficient and effective heuristic solution approaches is always well-justified. Only the modest size instances can be solved by means of exact algorithms. Among all heuristic and metaheuristic methods introduced in the literature for CLSP, we decided to implement the Forward-Pass heuristic which can be applied as a stand-alone solver to the deterministic CLSP. We tried to assess the performance of FP heuristic when implemented it in both static and rolling schedule environment.

1.2 Linear Programming formulation

Coordinated Lot-sizing or Joint Replenishment problem can be formulated in many different ways, we base our work on the classical or natural mathematical formulation, since it is readily comprehensible and we use it as the underlying structure in our stochastic programming models in the succeeding chapters. This math program formulation simply expresses the optimal balance between the individual and joint setup costs and the inventory holding costs over a static planning horizon. Item demands are assumed to be dynamic (time-varying) but deterministic over planning

horizon and they must be met without considering the possibility of backorders or lost sales. In addition, it assumes that an initial inventory is available for each item product in the family.
For each item $i;\ i \in \{1, 2, \dots, N\}$ and each time period $t;\ t \in \{1, 2, \dots, T\}$, we define the following parameters:

S_t Major setup cost, incurred if at least one unit of any item type is ordered in period t;
$s_{t,i}$ Minor setup costs, incurred if at least one unit of item i is ordered in period t;
$h_{t,i}$ Per-unit inventory holding cost for item type i during period t;
$d_{t,i}$ Deterministic time-varying demand for item type i in period t;
$I_{0,i}$ Initial inventory level for item type i;
T Time horizon length;
N Number of item products in the family.

Where the decision variables are:
$x_{t,i}$ Replenishment quantity (lot-size) of item type i at the beginning of period t;
$I_{t,i}$ Inventory level of item type i at the end of the period t (ending inventory level);
$y_{t,i}$ Binary variables = 1 if and only if item type i is replenished at the beginning of period t, i.e. $y_{t,i} = 1$ if $x_{t,i} > 0$;
z_t Binary variables taking the value 1 if an order is placed in period t.

So having denoted the order and inventory quantity decisions by non-negative continuous variables, and the setup decisions by binary type variables, the classical Coordinated Lot-Sizing Problem can be simply stated as a Mixed-integer programming formulation as the following:

$$\text{(CLSP)} \qquad \min \sum_{t=1}^{T}\left[S_t z_t + \sum_{i=1}^{N}\left(s_{t,i}y_{t,i} + h_{t,i}I_{t,i}\right)\right] \tag{1.2.1}$$

s.t.

$$I_{(t-1),i} + x_{t,i} - d_{t,i} = I_{t,i}, \qquad i = 1, \dots, N, \quad t = 1, \dots, T; \tag{1.2.2}$$

$$x_{t,i} \le M y_{t,i}, \qquad i = 1, \dots, N, \quad t = 1, \dots, T; \tag{1.2.3}$$

$$\sum_{i=1}^{N} y_{t,i} \le N z_t, \qquad t = 1, \dots, T, \tag{1.2.4}$$

$$x_{it}, I_{t,i} \ge 0, \qquad i = 1, \dots, N, \quad t = 1, \dots, T; \tag{1.2.5}$$

$$z_t, y_{t,i} \in \{0, 1\}, \qquad i = 1, \dots, N, \quad t = 1, \dots, T. \tag{1.2.6}$$

The objective function (1.2.1) computes the sum of order setups and inventory holding costs during the model time horizon. Constraints (1.2.2) ensure demand satisfaction for each item type at each

time period: the ending inventory level of each item at the end of a period should be equal to the ending inventory level at the previous time period, plus the item order lot-size, minus the demand of this item. Constraints (1.2.3) that link lot-sizing variables ($x_{t,i}$) to the setup binary variables $y_{t,i}$, state that the replenishment quantity of an item type can be positive only if that item type is replenished, while constraints (1.2.4) mean that individual item types can only be included in a joint replenishment if that replenishment is made. Note that constraints (1.2.3) can be tightened if M is replaced with $B_{t,i}$, where M is a sufficiently large number; and the $B_{t,i}$ are the sum of demands for item type i from period t to last period T. The initial inventories of each item type may be dealt with by netting off the demand.

General properties

The following properties are valid due to Wagner & Within (1958) for the joint replenishment optimal solutions:

- *Property 1*: Any optimal solution is such that: $x^*_{t,i} \times I^*_{(t-1),i} = 0 \quad (i = 1, \dots, N, t = 1, \dots, T)$. In other words, if item type i is replenished at the beginning of period t, it does not pay to hold this item type in stock during the time period $t-1$.
- *Property 2*: The optimal order $x^*_{t,i}$ takes one of the values: $d_{t,i}, d_{t,i} + d_{(t+1),i}, \dots, \sum_{q=t}^{T} d_{qi}$.
- *Property 3*: The optimal inventory level $I^*_{(t-1),i}$ takes one of the values: $0, d_{ti}, d_{ti} + d_{(t+1),i}, \dots, \sum_{q=t}^{T} d_{q,i}$.

The next one is due to Silver (1979):

- *Property 4*: if $d_{q,i} \sum_{r=t}^{q-1} h_{r,i} > S_q + s_{q,i}$ for any $q > t$ then it is not optimal to replenish the demand of item type i for period q at the beginning of period t.

Actually, there are other formulations which are more compact, based on the properties mentioned above and maybe faster in computational operations, but we use the classical formulation as the underlying structure in our models because of its comprehensibility.

1.3 Forward-Pass Heuristic

In the coordinated lot-sizing problems when the problem size increases, the computational time of optimal solution procedures may become a limiting factor particularly on micro-computers, and only the modest size instances can be solved to optimality in a reasonable amount of CPU time. Hence it would certainly be worthwhile to develop an efficient and effective heuristic algorithm. Forward-Pass heuristic is a technique that attempts to reduce total schedule costs by shifting replenishment orders into the earlier time periods when the setups cost saving exceeds the increase in inventory holding costs (Robinson *et al*. 2006). In other words, the procedure simply evaluates whether it is affordable or not, to meet demands of the future time periods by a production/order made at the current time period. It builds the production/order replenishment schedule beginning

from the period 1 as the current period, and then rolls forward through the time horizon. Two Forward-Pass heuristic procedures are proposed and they are applied to the joint replenishment problem, one algorithm is based on the Eisenhut (1975) decision criteria and the other one applies a modified Lambrecht and Vanderveken (1975) decision criterion. We based our computational experiments according to the first decision criterion that extends the concept proposed by Eisenhut for solving the multi-item capacitated lot-size problem to the coordinated lot-sizing problem. We describe in brief the algorithm here; the reader may refer to Robinson *et al*. (2006) for more details.

Starting from period 1, as the current time period, the cost function $C(t)$ is defined for the family, as the average cost per time period, for serving all item types demands through time t by an order made at time period 1, as the following:

$$C(t) = \frac{S}{t} + \sum_{i=1}^{N} C_i(t) \qquad (1.3.1)$$

Where *S* is the joint cost value, *N* is the number of items in the family and the $C_i(t)$, is defined as:

$$C_i(t) = \frac{(s_i + I_i(t))}{t} \qquad t \leq T \quad i = 1, \dots, N \qquad (1.3.2)$$

The average cost function introduced by Silver and Meal (1973) for each item type i in the family; where s_i is the minor setup cost for item i, and $I_i(t)$ is the sum of inventory holding costs incurred by item type i. Then by defining an extension of the Eisenhut's decision criterion to the family of items, it calculates per unit saving coefficient that is based on the first derivative of (1.3.1), for the future time periods as follows:

$$U(t) = \frac{(S/t^2 + \sum_i U_i(t) d_{it})}{\sum_i d_{it}} \qquad (1.3.3)$$

The positive saving coefficient at time period t, implies that the total system cost can be reduced by shifting *all* family orders in periods $2, 3, \dots, t$ into the current period 1. After assessing all future periods, the procedure rolls the current period forward to the next period and so on. Detailed steps of the heuristic procedure are provided in Robinson *et al.* (2006).

1.3.1 Computational experiments

This section is dedicated to evaluate the performance of Forward-Pass heuristic for deterministic CLSP. The Forward-Pass procedure we use is based on the Eisenhut decision criterion. We tested the performance of FP with respect to the optimal solution, in both static and rolling horizon environments. Therefore, we firstly assume that items demand values are known for all periods in problem time horizon (static planning framework) and then as is the case in the real life production/order planning schedules, we consider a long time horizon problem in which demands are not always available for the whole length of the long-term activity, but they are known only for a limited portion of time called planning horizon. In this situations, decisions are made by

solving a series of linked static planning problems implemented rolling through time (rolling-horizon framework).

In fact, one of our objectives of testing the FP in rolling horizon framework is due to the fact that the optimal methods over the short horizon may be suboptimal over the long run, whereby new time buckets are added to the current planning horizon. This comes from planning errors known as *end-effects*. A common end-effect in production scheduling is to set the level of inventory at the end of the planning horizon to zero, which is a policy that can result in excessive setup costs or stockouts in the long run (Brandimarte 2006). Therefore an optimal solution when applied within a rolling horizon framework, may be outperformed by heuristic strategies, as it is proved in the case of single-item lot-sizing problem that heuristic methods can outperform the WW optimal solution, in the rolling horizon environment

To evaluate the heuristic performance we need an efficient method to obtain the performance benchmark, like the optimal solution and its run time to achieve it. The metrics used are the percentage of gap optimality, defined by 100*(FP objective function minus the Optimal function value)/Optimal function value, and their running times. In both static and rolling horizon frameworks the optimal solution of each problem instance is obtained by the *Gurobi* optimizer version 7.5.2., which is a state-of-the-art math programming solver. To this aim, we considered the optimal solution of the mixed-integer programming formulation discussed in the previous section (1.2) as the performance benchmark solution. Indeed we modeled our mixed-integer problem in CVX modeling system, using Matlab and chose *Gurobi* as an active solver for it.

We implemented and solved the FP procedure also in Matlab. The computer used to run the both heuristic and optimal procedures is a Lenovo Yoga notebook PC, with an Intel(R) core(TM) i3-606U @ 2.00 GHz processor.

1.3.1.1 Finite time horizon

Considering a static or fixed horizon problem, the demand information is available for the entire horizon of the problem.

Our experiment design is similar to Erenguc (1988) which is based on the Kao (1979)'s random problem generation scheme. We modeled the experiment in 5 problem sets based on the number of item types in the family where $N \in \{2, 4, 6, 10, 20\}$ in order to study the effect of the number of items in the family on the computational efforts. The length of the planning horizon was fixed to 12 time periods ($T = 12$) in all problem sets. The inventory holding cost for each item $i \in \{1, 2, \dots, N\}$ was set to one unit ($h_{ti} = 1$) constant during the time, and we assumed the level of inventory is set to zero, for all items at the beginning of the planning horizon ($I_{0i} = 0\ \forall i \in \{1, 2, \dots, N\}$) in all problem instances. Then we generated diverse versions of CLSP by assigning different values to the joint setup cost, minor setup costs and items demand values assuming them to follow the *normal* distribution as the following:

$$\begin{aligned} &\text{Major setup cost:} && S_t \sim \mathcal{N}(\mu_S, 36), && \mu_S \in \{120, 480, 960\}, \\ &\text{Minor setup costs:} && s_{t,i} \sim \mathcal{N}(60,18), \end{aligned}$$

Demand values for even-numbered items: $d_{t,i} \sim \mathcal{N}(100,20)$,
Demand values for odd-numbered items: $d_{t,i} \sim \mathcal{N}(50,20)$.

The individual setup cost varies across item types, but it is assumed constant during the time, for a specified item, within a test problem. The joint setup cost is also considered constant within the planning time horizon of each test problem. The deterministic demand values for each item type per time periods are randomly generated from a *normal* distribution. For odd-numbered item types the mean value of demands is considered 50, while for even-numbered it is set to 100 in all test problems. In each test set, 5 problems were randomly generated, so on the whole, the Forward-Pass procedure was tested on 75 different test problems.

As introduced in Robinson *et.al.* (2006) the *mean setup cost ratio*, $\sum_i s_{t,i} / S_t$ can be considered as a measure in the joint replenishment problems to assess the performance of the heuristic procedures and their required computational times. It is defined as the ratio of mean value of minor setup costs to the mean value of joint setup cost. We calculated this ratio for each problem set (each row of the table) representing them in column 3 of the Table 1.

The percentage of Optimality Gap was calculated as: 100*(FP objective function value – Optimal objective value) / Optimal objective value. The principal results are summarized in Table 1.

Joint Cost	*Size (N x T)*	*Mean setup cost Ratio*	**Percentage Optimality Gap**			**No. of problems with Optimal solution**	**Average CPU time**	
			Average	*Min.*	*Max.*		*Optimal Solution*	*FP Heuristic*
120	2 × 12	*1*	1.69	0.00	4.28	2	0.53	0.0000
	4 × 12	2	14.11	5.77	18.18	0	0.52	0.0000
	6 × 12	3	10.34	4.65	15.98	0	0.57	0.0000
	10 × 12	5	8.36	5.90	11.88	0	0.54	0.0000
	20 × 12	10	8.90	6.57	10.76	0	0.61	0.0000
480	2 × 12	0.25	0.73	0.00	2.74	3	0.59	0.0020
	4 × 12	0.50	0.74	0.00	1.35	1	0.54	0.0024
	6 × 12	0.75	0.00	0.00	0.00	5	0.47	0.0024
	10 × 12	1.25	19.61	18.09	21.12	0	0.48	0.0024
	20 × 12	2.50	7.91	5.07	9.53	0	0.55	0.0034
960	2 × 12	0.12	5.17	0.00	12.80	2	0.56	0.0000
	4 × 12	0.25	0.00	0.00	0.00	5	0.53	0.0000
	6 × 12	0.37	2.12	0.39	2.76	0	0.57	0.0008
	10 × 12	0.62	0.00	0.00	0.00	5	0.55	0.0008
	20 × 12	1.25	18.39	14.17	20.27	0	0.61	0.0012

Table 1: Summary of computational results of 75 problem instances for Forward-Pass Heuristic.

We can easily observe that when the joint cost was 120, the Forward-Pass heuristic was able to find the optimal solution in 2 of the 25 test problems, while in the cases of joint cost 480 and 960, the number of problems with optimal solution found by heuristic increases to 9 and 12 respectively. The average gaps optimality (percentage difference between the heuristic and the optimal solution) were 8.68%, 5.79% and 5.13% when the joint cost was 120, 480 and 960 respectively. It varies between [0.00%, 21.12%] among all 75 test problems. However, in 27 of 75 problem solved, the percentage gap optimality was less than 1% and in 39 test problems this percentage was less than 5%. The average CPU time required by *Gurobi* optimizer was [0.47, 0.61] seconds, while the computational effort required by Forward-Pass heuristic is always less than 0.0024 second in all 75 problems.

As reported in the third column of Table 1, the mean setup cost ratio ranges from 0.12 to 10 in our experimental design. In general the problem sets with lower mean setup cost ratio in particular when the ratio is less or equal than one, perform substantially better than ones with higher ratio, based on the average gap optimality and the number of optimal solutions reported in columns 4 and 7. It means that in general when a problem becomes more joint cost dominant and consequently the time between family orders increases, the problem gets easier to be solved by Forward-Pass heuristic procedure.

In contrast, At each level of joint cost value, when fixing other costs in the problem, as the number of item types increases and consequently the mean setup cost ratio increases, we cannot say that the problem becomes difficult to be solved; as an example the solution quality in problem set with ratio=10 is better than the problem set with the ratio=2. In fact comparing the problem sets with ratio greater than one, as the ratio increases, the optimality gap decreases and the solution quality results better.

1.3.1.2 Rolling-horizon framework

In practice the values of future demands are not always well-estimated for the whole length of the long-term activity. In a rolling planning horizon scheme for lot-sizing problems, the uncertain demands are firstly forecasted. Moving closer in time, the forecast is replaced with booked customer orders as each time period enters the replenishment planning horizon. The resulting problem is a short-term, or static lot-sizing problem with *deterministic,* dynamic demand. Therefore, the future is represented as a set of these short-term, independent problems, for which demand values are deterministic. When solving each static problem, replenishment decisions within the frozen order interval are implemented rolling through time. After a pre-specified re-planning periodicity, a new short-term plan is constructed using the updated demand received since the last planning cycle. When constructing the new replenishment plan, orders from the prior planning cycle that lay outside the frozen order interval may be rescheduled if doing so results in an improved solution for the new planning cycle. In this manner, solutions to a series of linked static planning problems are implemented rolling through time. (Blackburn and Millen 1982).

In fact, rolling schedule procedures provide a mechanism for dealing with incomplete demand data. The difference between optimal solution for the static problem, which assumes demand is

known for the entire problem duration, and the rolling schedule solution provides a measure for the cost of limited demand information (Narayanan2010). Indeed, the rolling schedule procedures provide a heuristic solution to the overall planning problem. Therefore, instead of examining the effectiveness of an algorithm over the expected future lifetime of a firm (or a sufficiently long period of time), the method under examination is applied to a series of problems generally between ten and twenty periods in length (Blackburn *et al.* 1980). The quality of the heuristic solution, is influenced by the specific lot-sizing solving method and decision variables governing implementation of the lot-sizing procedures.

The comparative researches for single and multi-item lot-sizing problems implies that the behavior of an optimal method changes when applied to the rolling schedules. For example in the case of single-item uncapacitated lot-sizing problem with deterministic demand, the optimal solution can be easily found by the well-known Wagner and Whitin dynamic programming algorithm in a static planning horizon; while within a rolling horizon framework, it can be outperformed by the heuristic strategies (Brandimarte 2006). So the best performing lot-size procedure in a static environment, may not be the best performer in a rolling horizon framework (when new time buckets are added to the current planning horizon). This is due to the end-of-horizon effect, where not knowing the demand beyond the planning horizon may lead to relatively poor quality short-term schedules (Narayanan *et al*. 2010). A common end-effect in production scheduling is to set the level of inventory at the end of the planning horizon to zero, which is a policy that can result in excessive setup costs or stockouts in the long run. Therefore, even with optimal solutions to the static problems, their implementation horizon environment provides at best a heuristic solution (Simpson, 1999). Blackburn and Millen (1980) found that due to end-of-horizon effects simple heuristics may perform better than optimization procedures in rolling schedule environment. Heuristic approaches do not exhibit the same degree of nervousness as the optimal procedures, since the heuristic methods do not consider the all information about future in making the current ordering decisions. As an example, if only one additional time period is added to the problem horizon, with optimal solutions like WW in the case of single-item lot-sizing problem, the timing and size of all previously planned orders may be changed while only the final order may be affected by the additional period, in the use of a heuristic procedure. So while the heuristics may provide less effective decisions than optimal solution methods in a static experimental framework, their myopia actually reduces the amount of schedule instability in a rolling-schedule environment (Blackburn *et al.* 1980). It is experienced that myopic lot-sizing heuristics, like the Silver-Meal heuristic in the case of single-item lot-sizing problem may outperform optimal solutions obtained by exact algorithms.

In the present section, the Forward-Pass as a myopic heuristic for the CLSP, is evaluated in a rolling-schedule environment against the optimal solution obtained by the *Gurobi* optimizer applied in each planning cycle to the deterministic CLSP. The optimal solution for the whole long-run horizon of 300-periods is also obtained by *Gurobi* for each subset of test problem instances.

To this aim, a set of simulation experiments was designed to assess the impact of different factors on the performance of the Forward-Pass in the rolling horizon framework. The forecast window

(planning horizon length), the frozen interval length (number of periods for which the planned schedule will be implemented), and the joint cost value are considered as the factors in our computational experiments. The forecast window or planning horizon T is defined as the number of future time periods (including the current period) for which demands are known for all items in the family (or equivalently, the length of the multi-period model). In order to study the effect of Master Production Schedule (MPS) design policy parameters such as the planning horizon and frozen interval, on the total schedule cost, the planning horizon of 2 through 15 are considered in our computational experiments and the replenishment schedules are frozen for k periods where k ranges from 2 to T. In all simulations re-planning occurs at the end of the frozen interval.

A long-run horizon of 300 time periods is considered in which normal demands are randomly generated in the same way of previous section, but the standard deviation of item demands was set to $\sigma = 10$. In addition, we assumed that the family consists of 5 item types. The inventory holding cost was set to 1 and the other cost parameters are designed as the following, assuming all constant during the problem horizon.

Major setup cost:	$S_t \sim \mathcal{N}(\mu_S, 36)$,	$\mu_S \in \{500, 1500, 2000\}$,
Minor setup costs:	$s_{t,i} \sim \mathcal{N}(60,18)$,	
Demand values for even-numbered items:	$d_{t,i} \sim \mathcal{N}(100,10)$,	
Demand values for odd-numbered items:	$d_{t,i} \sim \mathcal{N}(50,10)$.	

The 300-period problems were solved in the following way: at the first planning cycle, a deterministic CLSP consisting of demands in periods 1,…,T is solved, but only a set of decisions which are in the frozen horizon (i.e., the periods 1,…,k) are implemented. Then the planning process rolls forward to cycle 2, where the new deterministic CLSP schedule is computed for demands in periods k+1,…, k+T (the demands in periods T+1, …,T+k are newly received) and similarly only the ordered placed in the frozen interval are implemented and so on. In this manner, replenishment schedules are updated rolling through time. Since our objective in this section, is to find out if a heuristic method such as Forward-Pass might outperform an optimal procedure in the rolling horizon system or not, we conducted both heuristic and optimal procedures rolling through time for a 300-period horizon problem. The solution of two methods rolled through time were compare to the optimal solution of the whole 300-period CLSP. The quantitative measure of the quality of the joint replenishment procedures in the rolling schedule, was the percentage of gap optimality over whole 300-period horizon.

Table (2) provides the experimental results of gap optimality for both optimal procedure and forward-Pass heuristic, in the function of policy variables governing the rolling horizon and the joint cost value.

		Optimality Gap Percentage (%)					
		Mean Joint Cost: 500		Mean Joint Cost: 1500		Mean Joint Cost: 2000	
Planning Horizon	Frozen Interval	Gurobi	FP	Gurobi	FP	Gurobi	FP
2	1	0.00	0.00	12.57	12.57	19.87	19.87
	2	0.00	0.00	12.57	12.57	19.87	19.87
3	1	7.46	0.00	0.19	0.19	1.50	1.50
	2	0.69	0.00	12.97	12.97	20.22	20.22
	3	9.28	12.16	0.19	0.19	1.50	1.50
4	1	0.00	0.00	3.65	0.19	0.59	0.84
	2	0.00	0.00	3.65	0.19	0.59	0.88
	3	5.82	0.00	0.47	0.19	1.74	1.70
	4	0.00	0.00	3.65	22.40	0.59	22.85
6	1	0.00	0.00	0.19	0.19	1.50	0.84
	2	0.00	0.00	2.08	0.19	4.39	0.84
	3	2.86	0.00	0.19	0.19	1.50	0.84
	5	3.24	0.00	4.74	5.53	6.29	6.60
	6	0.00	0.00	0.19	0.19	1.50	2.41
7	1	2.45	0.00	1.57	0.19	0.35	0.84
	2	0.58	0.00	1.46	0.19	0.73	0.84
	3	2.28	0.00	0.45	0.19	1.71	0.84
	5	3.94	0.00	2.36	0.19	5.96	1.40
	7	3.29	5.33	1.66	12.26	0.59	11.19
10	1	0.00	0.00	0.84	0.19	0.20	0.84
	2	0.00	0.00	0.66	0.19	0.72	0.84
	3	0.73	0.00	0.41	0.19	1.40	0.84
	5	0.83	0.00	0.90	0.19	1.31	0.84
	10	0.00	0.00	1.06	9.06	0.69	6.03
15	1	0.71	0.00	0.19	0.19	0.05	0.84
	2	0.56	0.00	0.81	0.19	0.16	0.84
	3	0.00	0.00	0.19	0.19	0.21	0.84
	5	0.87	0.00	0.88	0.19	0.22	0.84
	10	0.56	0.00	0.66	0.19	1.31	0.84
	15	1.38	2.43	0.19	0.19	0.46	3.13

Table 2: Percentage of Gap optimality of Forward-Pass heuristic and optimal procedure in rolling horizon framework.

Our computational results within a rolling horizon environment, have demonstrated that in general and under certain conditions, the Forward-Pass heuristic can provide cost performance supplier to the solution found by the optimization solver in particular *Gurobi* that solves each planning cycle to optimality, in our rolling horizon framework.

As mentioned above, in our experimental design we can distinct two types of factors; the environmental factors and the Master Production Schedule (MPS) design parameters. In fact the effectiveness of the rolling horizon strategy, apart from the short-term procedure solver, depends

significantly on the rolling horizon policy design variables such as the planning horizon length, the frozen interval length, re-planning periodicity etc.
The major setup cost was considered as the only environmental factor while fixed the other costs in the problems. Having fixed the number of items in the family, the joint setup cost value reflects the joint or major time-between orders $TBO_{family.}$ The time between joint orders at period t is defined as the number of periods for which all items demands are satisfied by an order placed in the period t. As reported in Narayanan *et al*. (2010) and originally calculated in Federgruen *et al*. (2007), the time between joint orders in the joint replenishment problems, can be calculated as $TBO_{family} = \left(\sqrt{2K/hD}\right)$, where K is the joint setup cost, h the per unit inventory carrying cost, and D is the average demands for the family (averaged among all items in the family). Since we considered three values of joint cost 500, 1500 and 2000, having fixed the number of items and the average demand for the family and also the inventory holding, there are three levels for TBO_{family}. In this way, as the joint cost value increases, the TBO_{family} also increases.

Regarding the MPS parameters such as planning horizon length and the frozen interval, we can observe that the policy design variables in rolling framework have a significant impact on schedule cost. In fact, if these parameters particularly the planning horizon length are not adjusted properly, the entire schedule cost will have a significantly gap from optimality. An obvious observation in our experiments is that if the length of the planning horizon is lower than the TBO_{family}, some of the demands in future time periods which could be satisfied by an order placed in the current period may be neglected and it creates an increasing schedule cost. So first intuitive point is that the planning horizon length should be greater than the time-between joint orders.
Calculating the TBO_{family} values at three levels of joint cost value by using the equation mentioned above, and comparing them to the TBO_{family} in the related optimal solution of 300-period problem obtained by *Gurobi,* the TBO_{family} is always 2 periods when the joint cost is 500, but it alters between 3 and 4 periods at both levels of joint cost 1500 and 2000. So the problem sets at the levels of joint cost=1500 and 2000 have a significantly high gap optimality when the planning window is considered 2 periods, while in the case of joint cost=500, the solution with the rolling planning window of 2 periods, is optimal. Indeed, at two levels of joint cost 1500 and 2000, in the test problems with planning window less than 4, both optimal and heuristic procedures have a significant gap optimality. Therefore, the joint replenishment problem when implemented in rolling framework, the performance is highly sensitive to the length of the planning window. As denoted in Narayanan (2010) and originally shown by Baker (1977), the best planning window is an integer multiple of economic order cycle length. The economic natural order cycle length for the joint replenishment problem is calculated in Ballou (1988) as: $NC = \sqrt{2(K + \sum_{i=1}^{N} s_i)/hD}$,
where K is the joint setup cost value, s_i the setup cost for item type i, h the inventory carrying cost and D the average demand for the family.

To conclude, we can say at the level of joint cost=500, the Forward-Pass heuristic outperforms the solution obtained by *Gurobi* optimizer when implemented in rolling horizon framework. The only exception is in the cases where the frozen interval is equal to the planning horizon (assuming that

re-planning occurs at the end of the frozen horizon), In other words when all decisions obtained by the short-term problems are implemented without considering of any exploitation of information about future demands; so obviously in this case the Forward-Pass has a gap optimality with respect to the *Gurobi*. At the other levels of joint cost value, for the planning window greater than 3, i.e. greater than TBO_{family}, for some values of frozen interval, in particular when it is a multiplier of TBO_{family}, the rolling horizon scheme performs better and in overall, the Heuristic procedures outperforms the optimal procedure. Since our objective in this experiment, is not to study the design of MPS parameters but is to compare the performance of Forward-Pass heuristic with respect to the optimal procedure in a rolling-schedule environment, we are satisfied with the overall results and do not deepen into design of MPS parameters. Therefore, we can conclude that in addition to the side benefits of computational simplicity, certain lot-sizing heuristics can at times, provides cost performance supplier to optimal solutions in the rolling schedule scheme.

Chapter 2

Stochastic Programming models

The main assumption in deterministic programming models is that the parameters are always known and certain in the model. But in the real-world and in the practical applications of such models, this assumption is so restrictive, as the problem data are always exposed to some degree of uncertainty.

In manufacturing context, different types of uncertainty can arise in the system and the production plan can highly be affected by them. Uncertainty in the external supply process, in the internal supply process and uncertainty in demand values and in demands forecast are the major types of uncertainty in manufacturing systems. The uncertainty in replenishment lead times, and in the replenishment quantities can be accounted as examples of external uncertainties. The delivery date of raw materials may not be known with certainty due to capacity constraints in supplier distribution processes or disruptions due to a work stopping or because of weather disorders. The supplier's delivery amount also can be differ from the expectations (the buyer might reject some amount of orders due to quality problems). Furthermore, the uncertainty in the production flow amount and in the process duration which are depending on the equipment availability, dispatch rules and the other process conditions, are two examples of uncertainty in internal system processes. The uncertainty in future demand forecast, is actually the largest source of uncertainty, as all production planning systems rely on a forecast of future demands or demand plan as their system's input. Since none of forecast strategies is perfect, the actual demand which will be realized may be substantially different from the forecasted value, resulting an increase in the system costs.

Formulation and then development of solution strategies (both optimal and heuristic) for deterministic mathematical models have been the dominant thrust for years in the production planning literature. The main intent of deterministic models has been to determine the requirements to find a feasible production plan and to capture the key cost tradeoffs that depend on the production plan. Typically, a feasible production plan is one that satisfies the *given* demand over the planning horizon with no backorders or lost sales, abides by specified production recipes for each final product, and does not violate any capacity constraints (Graves 2011). Hence the deterministic model literature is largely oblivious to any uncertainty. Much like research on the

economic-order-quantity (EOQ) model, the contention is that the value of these models is in optimizing critical cost tradeoffs, often in the context of tight constraints. In fact, dealing with uncertainty in this literature, is of a secondary importance and also there is an assumption that the uncertainty can be handled independently of the production plan determination and by the other measures. This assumption is a shortcoming of this literature but it makes the deterministic models tractable.

In production planning, uncertainty may give rise to infeasibilities and production planning disturbances. Considering uncertainty systematically in the model, is as important as having the model itself. Producing feasible, robust and optimal schedules, is the objective of methodologies for process scheduling under uncertainty. In other words, uncertainty consideration plays the role of validating the use of mathematical models and preserving plan feasibility and viability during operations (Li and Ierapetritou 2007). Therefore, planning under uncertainty has been under attention in recent years in the open literature in operations research field. Based on the different uncertainty descriptions, alternative planning approaches and relevant optimization models will be applied.

According to the uncertainty treatment, there are two modes of scheduling: reactive and preventive scheduling. Reactive scheduling (dynamic or online scheduling) means to modify the original scheduling policy or generate new scheduling policy on the time when uncertainty occurs. In contrast, preventive scheduling, tries to generate robust planning policy *before* occurring the uncertainty. When we have not enough data prior to realization of the uncertain parameters, protective actions are impossible, for instance in the case of disruptive events, order cancellation, rush order arrivals, or machine breakdowns etc. However, by preventive scheduling, we can deal with uncertainty, such as processing times, products demands or prices. In this cases, historical data and forecasting techniques may be useful to derive information about uncertain parameters behavior in the future, by kind of parameters range or stochastic distributions. Preventive scheduling acts as a basis in planning support activities and commitments are based on the preventive schedules. Preventive scheduling may be handled in the following approaches: stochastic based methods, robust optimization approaches, fuzzy programming, sensitivity analysis and parametric programming methods (Li and Ierapetritou 2007).

Sensitivity analysis (SA) tends to ascertain how the output of a given model, depends to the input parameters. This is a traditional method for checking the quality of a model, besides it is a powerful tool for checking the reliability and robustness of any solution. This typical approach, is based on studying the way in which changes in data elements would impact on optimal solution, considering that the structure of the model remains the same. The solution may be sensitive to the individual data, what means that it could change (more or less significantly) when this data is modified. In fact, sensitivity analysis is a useful tool to study the impact of uncertainty in post-optimality analysis but it might be insufficient to get solutions in which uncertainty is treated in a really proper way. However in SA the dynamic nature of decision making under uncertainty and the information flow is neglected and so the solution structure obtained from the deterministic model formulations while varying the parameters, may be totally different from the model structure in which

uncertainty is modeled. Therefore, in practical applications sensitivity analysis is almost useless when dealing with uncertainty.

In contrast, in order to improve the flexibility of a schedule prior to execution, one can try to measure the performance of a deterministic plan under changing conditions that are due to uncertain parameters. Robust Optimization methods focuses on developing the preventive planning to minimize the disruptions effect on the plan performance measure and they try to ensure that the realized schedules will not differ drastically, while holding a high level of schedule performance. In other words, while introducing simple perturbations in the system, it may find a solution which is robust to the data uncertainty, by reformulating the initial problem, or solving a sequence of problems. In this framework, an optimization solution is referred as solution robust, if it remains around the optimal solution for all scenarios, and is regarded as model robust, if for all scenarios it remains feasible. The main advantage of robust optimization (in comparison to stochastic programming) is that no assumption is needed related to the underlying probability distribution of the uncertain parameters.

Stochastic Programming (SP) is the most commonly used of preventive scheduling approaches, in which the original deterministic programming model is transformed into stochastic version, by treating the uncertain parameters as stochastic variables. The main intent in SP methodology is to capture the uncertain parameters in some way in the system. When making decisions is affected by uncertainty, one can be interested in *explicitly* adding and modeling the uncertainty. Recent works have shown the importance of explicitly incorporate demand uncertainty in the economic setting, along with the evidence about the computational burden of solving the relative stochastic programming models for a sufficiently large number of scenarios (Manerba *et al*. 2017). Yet, with the increasing computational power and improvements in the optimization algorithms, the feasibility of using SPs for production planning has been always explored and a good portion of literature examines the applicability of stochastic optimization, in particular the stochastic programming methods, to production planning models.
The main assumption in SP is that the probability distributions of the random events are known or can be approximated. When there are some information about the behavior of uncertainty, a common approach to handle the uncertain parameters, is through the *Probability description*. In other words, using the probabilistic models to describe the uncertain parameters. In fact, uncertain parameters are characterized by the probabilities associated with different events. In this way, one can model uncertainty using either discrete probability distributions or with the help of a discretization of continuous probability distribution function. In other words, this approach is based on the either discrete probability distributions or a discretization of continuous probability distribution function. The objective of SP is to identify a feasible solution that optimizes the expected value of a function over all possible realizations of the random events. In other words, in SP an expectation of a certain performance criterion, such as the expected system cost, is optimized with respect to planning decision variables. In general Stochastic Programming models are divided into the following categories: Two-stage or Multi-stage stochastic programming and Chance-

constrained programming based approach. In the Chance-constrained or probabilistically constrained model, the idea is to settle for a solution with a probabilistic guarantee, so it has a sensible interpretation and may be useful in some engineering problems where robustness in more relevant than flexibility. However this approach has definite limitations since it also neglect modeling the flow of information and the sequence of decision stages. As well, the chance-constrained models are not convex, and they can lead to quite risky solutions. Our models and solution methods are based on the stochastic approach known as *recourse* problems.

In the present study, our objective is to apply the stochastic programming approaches to formulate models for the coordinated lot-sizing problem under demand uncertainty ($CLSP_{ud}$) in such way the uncertainty can be explicitly represented in the model. So as the first step to build our stochastic model, we are faced to the problem of representing demand uncertainty in a suitable form. To this aim, as it is usual for demand uncertainty, we try to create a model, by means of a simple discrete approximation of the assumed demand distribution that serves as input to our model. To this aim, we may take advantage of taking into account more information about future demands, than only the expected value of them. As we assume demand values follow the normal distribution, the standard deviation of demand values may be considered as an information to model uncertainty. So in brief, our SP models are based on the present demand (that is accurately forecasted for the first time period), the expected value and standard deviation of demand values.
The contents of this chapter are structures as follows:
In section §2.1 we make a brief introduction to the stochastic programming models with recourse and the main concepts in SP models which will be useful later.
Then in §2.2 we present our stochastic coordinated lot-sizing model by stating it as a multi-stage stochastic problem with recourse. And finally in §2.3 and §2.4 we describe our scenario generation method and the demand history generating approach which are the base of simulation architecture in our numerical experiments in the next chapter.

2.1 Stochastic Programming with Recourse

Stochastic programming with recourse was firstly introduced by Dantzig (1955) for the problems subject to uncertainties. Since then, it has become one of the most important methods to optimize systems that include uncertain parameters in some or all aspects of the system. The idea in recourse problems is to distinguish the decision stages and the decision variables. In two-stage recourse problems which are the most extensively studied SP models, decision variables are partitioned into two sets. The first stage decisions are decided before the realization of the uncertain parameters. They must satisfy immediate constraints having immediate first-stage costs. Once the random event occurred, the second-stage (recourse) actions are taken given the fixed first-stage decisions, resulting in second-stage costs. The objective is to make first-stage decisions in a way that

minimizes the sum of first-stage costs and the expected value of objective function of the random second-stage costs.

Multi-stage stochastic programming formulations arise naturally as a generalization of two-stage stochastic models, in order to model the complex systems where decisions are made under uncertainty in multiple time stages. The decision and uncertainty realization sequence of two-stage SPs are generalized in multi-stage SPs. In particular, decisions are first made at the beginning of each stage before any of the future realizations are observed. These decisions depend only on the previously observed realizations and decisions made. Following the observation of random data, the next stage decisions are made. Indeed the only information we may use when making each decision, consist on the history so far. The objective is making decisions at each stage, sequentially, that minimizes the expected value of objective function over all possible realizations. One main difference from two-stage SPs is that the expected recourse functions are recursive.

Let's consider a T-stage problem where a series of decisions $\{x_t\}_{t=1}^{T}$ is made depending on the Random Events $\{\xi_t\}_{t=1}^{T}$. The decision $x_t \in \mathbb{R}^{n_t}$ at time stage t is made based on the information of the previous stages decisions $x_1, \dots, x_{t-1}$ and the observed random events $\xi_1, \dots, \xi_t$ while minimizing the objective function $f_t(x_1, \dots, x_{t-1}, \xi_1, \dots, \xi_{t+1})$. And the decision variables x_t which depend on $x_1, \dots, x_{t-1}$, and $\xi_1, \dots, \xi_t$ must satisfy the constraints. Then the multi-stage stochastic problem with recourse, can be represented as follows (Solak 2007):

$$\begin{aligned}
&\min f_1(x_1) + E_{\xi_1}\Big[\min f_2(x_1, x_2, \xi_1) \\
&\qquad + E_{\xi_2|\xi_1}\big[\min f_3(x_1, x_2, x_3, \xi_1, \xi_1) + \dots \\
&\qquad + E_{\xi_{T-1}|\xi_1,\dots,\xi_{T-2}}[\min f_T(x_1, \dots, x_T, \xi_1, \dots, \xi_{T-1})] \dots\big]\Big]
\end{aligned}$$

s.t.

$$\begin{gathered}
g_1(x_1) \le 0 \\
g_2(x_1, x_2, \xi_1) \le 0 \\
\vdots \\
g_T(x_1, \dots, x_T, \xi_1, \dots, \xi_{T-1}) \le 0 \\
x_t \in X_T, \quad t = 1,2, \dots, T
\end{gathered}$$

where $x_t \in \mathbb{R}^{n_t}$, $\xi_t \in \mathbb{R}^{m_t}$, $t = 1, 2, \dots, T$;

$f_t: \mathcal{R}^{n_1+\cdots+n_t} \times \mathcal{R}^{m_1+\cdots+m_{t-1}} \to \mathcal{R}$ and $g_t: \mathcal{R}^{n_1+\cdots+n_t} \times \mathcal{R}^{m_1+\cdots+m_{t-1}} \to \mathcal{R}^{d_t}$.

In general, when the random variables are continuous, the two-stage and multi-stage stochastic problem formulations with recourse cannot be solved easily because of the multi-dimensional integrals in the recourse objective function that we do not really know. In fact, despite of some interesting properties of recourse function (such as convexity in some cases) they are not practically used to solve the stochastic problems. In practical applications of multi-stage SP

problems, it is common to assume that the random vector ξ follows a known discrete probability distribution resulting in a finite number of possible realizations of the random events (i.e. scenarios). In such way and by considering all those scenarios the so-called *Deterministic Equivalent Problem* (DEP) of the multi-stage SP, can be easily formulated and more tractable.

Therefore, we explicitly consider a set of potential scenarios denoting by $\mathcal{S}$ and the probability of a particular realization of a scenario $s \in \mathcal{S}$ by P_s, such that the standard axiom $\sum_{s\in\mathcal{S}} P_s = 1$ is satisfied. An example of a Four-stage scenario tree is depicted in the figure (1) below.

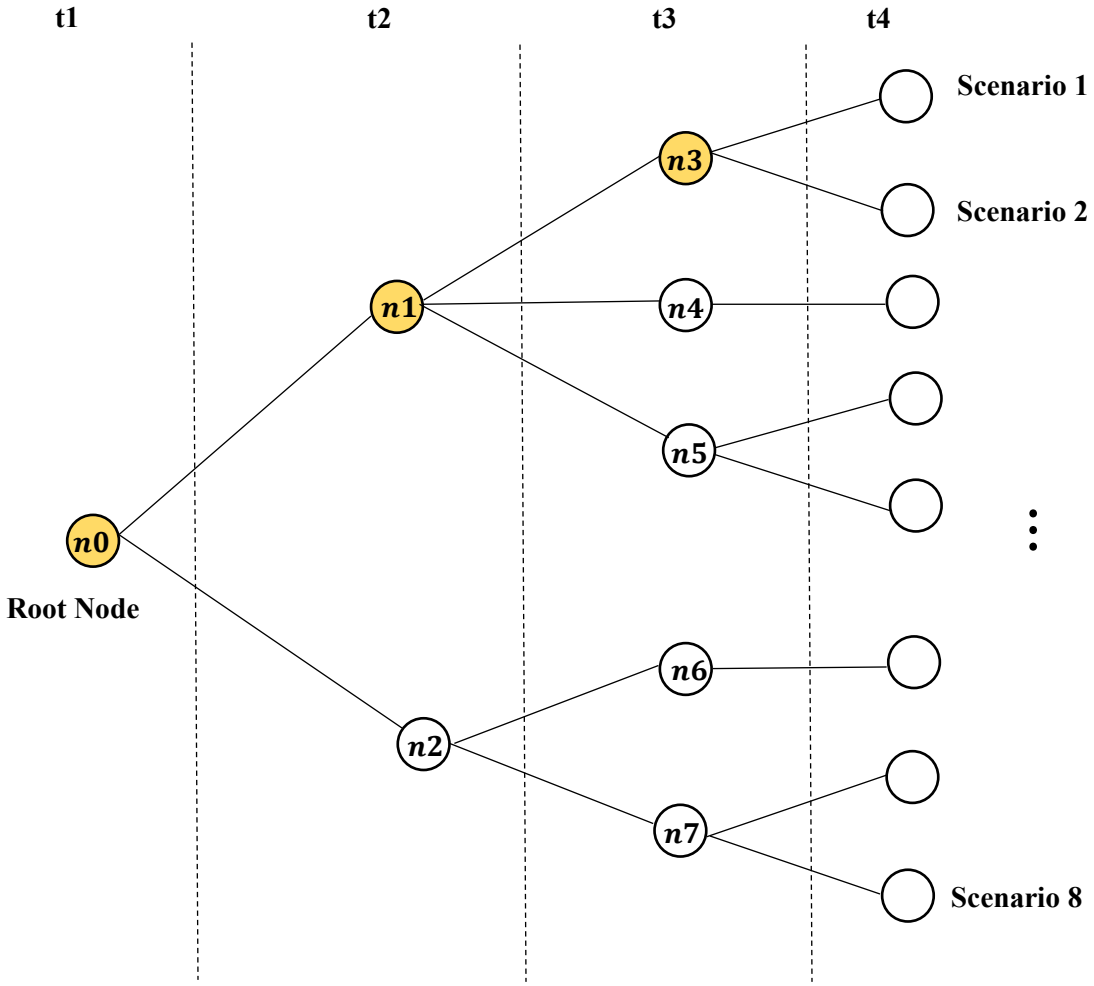


Figure (1): a four-stage scenario tree

Each node in the tree corresponds to an event, where we should make some decision.

Considering a set of all possible scenarios $\mathcal{S}$, the Deterministic Equivalent (DE) of the multi-stage SP can be formulated as follows:

$$\text{Min} \sum_{s \in \mathcal{S}} P_s [c_1^s x_1^s + c_2^s x_2^s + \cdots + c_T^s x_T^s]$$

s.t.

$$A_{11}^s x_1^s \leq b_1^s \quad \forall s \in \mathcal{S}$$

$$A_{21}^s x_1^s + A_{22}^s x_2^s \leq b_2^s \quad \forall s \in \mathcal{S}$$

$$\vdots$$

$$A_{TT-1}^s x_{T-1}^s + A_{TT}^s x_T^s \leq b_T^s \quad \forall s \in \mathcal{S}$$

$$x_t^s - x_t^{s'} = 0 \quad \forall s, s' \in \mathcal{S} : (\xi_1^s, \dots, \xi_t^s) = (\xi_1^{s'}, \dots, \xi_t^{s'}), t = 1, 2, \dots, T \qquad (*)$$

$$x_t^s \geq 0 \quad \forall s \in \mathcal{S}, t = 1, 2, \dots, T$$

In the formulation given above the objective function and the constraints are supposed to be linear. Constraints $(*)$, are the *Non-Anticipativity (NA)* constraints, which ensure the equivalence of the decisions that are made at time stage t for all scenarios that have the same past history.
These constraints are added because although we must use a solution method that takes into account the fact that there will be future decision points, we cannot require the decision maker to know the future in order to know what to do at any point in time. The Non-Anticipativity constraints ensure that the decisions are implementable (Solak 2007). For example in the figure (1) the implementability policy or NA implies that the decision variables at the root node of the tree have to be same for all scenarios $s = 1, 2, \dots, 8$, decision made at node $n1$ have to be same for scenarios $1, 2, 3, 4$ and 5 and at node $n3$ have to be same for 1 and 2. The similar logic applies to other nodes and scenarios.
Therefore, in order to practically tackle with multi-stage stochastic problems, we should resort to statistical sampling (scenario generation). We will discuss our scenario generation method in the section §2.3.

2.2 Stochastic Coordinated Lot-sizing Problem ($CLSP_{ud}$)

Since our problem under consideration is an uncapacitated variant of joint replenishment problem, and we focus on the special case in which only the demands are stochastic, *Demand Uncertainty*

is the main complicating issue in our problem. Using the notation in Manerba *et al*. (2017) we refer to our Stochastic Coordinated Lot-Sizing Problem as $CLSP_{ud}$.

As the first step to build our stochastic model, we must clarify the way in which the information about item demands flows in our model and accordingly to it, decisions are made. As it is classified and also depicted in Brandimarte (2006), there are different uncertainty profiles. In production planning models when a decision maker has no idea about the product demands even in the current time bucket, the planning model is subject to total (100%) uncertainty; a less uncertain model can be existed when the decision maker knows exactly the demands in the first period, but has some degree of uncertainty in the future time periods. We consider the case in which the items demand in current time bucket are discovered before planning production for the first time bucket, and we have a constant level of uncertainty for the future time buckets. So the order decisions must be taken only knowing the demand of the first period. In other words, we have no uncertainty currently, but "full" uncertainty for the future.

Concerning the stochastic lot-sizing problems, an explicit representation of stochasticity has become more interesting and also important in the years. As described before, scenario tree is a typical choice to explicitly model the system uncertainty. Scenario tree can be regarded as a discretization of a possibly complex probability distribution where a demand value for each item is associated with each node. Our stochastic approach is based on a tree of possible demand scenarios involving their associated probabilities of occurrence. Considering that demand follows normal distribution, assuming to know the mean value and the standard deviation of each item demand.

In order to start with formulation of our stochastic programming model, we remind that there are two basic ways to build a multistage stochastic programming model: the *split-variable* and the *compact* model formulations. The choice between two frameworks depends on the solution algorithm we want to apply. We first introduce the compact model formulation.

General notation

We are now in the position to describe the stochastic programming formulation of the $CLSP_{ud}$ and introduce some general notation in the model. Our mathematical formulation of $CLSP_{ud}$ is derived from the classical formulation of the deterministic version of the model discussed in the previous chapter, and tree-based representation of uncertainty.

In order to extend the deterministic formulation to a stochastic model, we assume that the uncertain problem parameters evolve as discrete time stochastic process with a finite probability space and generate a filtration. This information structure can be interpreted as a scenario tree where the nodes n in stage t of the tree constitute the states of the world that can be distinguished by information available up to time stage t (Ahmed *et al*. 2003).

For each item $i;\ i \in \{1, 2, \dots, N\}$ (N is the number of items in the family) and each time period $t;\ t \in \{1, 2, \dots, T\}$, and $\mathcal{N}$ a set of nodes, the scenario tree based model is made of the following elements:

- $n \in \mathcal{N}$ A generic node in the tree; 1 and $\mathcal{T}$ refer to the root node and the set of terminal node respectively;
- $T(n)$ The time period for node n;
- $a(n)$ The immediate predecessor for node n, ($n \neq 1$);
- $\Omega(n,t)$ The unique ancestor of node n at time period t ($n \neq 1, t < T(n)$);
- $\Sigma(n,t)$ The set of successor nodes of n at time period t ($n \notin \mathcal{T}, t > T(n)$).
- $d_i^{[n]}$ The demand for item i in node n.
- $p^{[n]}$ The unconditional probability of node n.

$T(n)$, denotes the time stage corresponding to the node n. Each node n of the tree, except the root node, has a unique parent $a(n)$, and each non-terminal node n is the root of a sub-tree $\tau(n)$. Thus $\tau(0)$ denotes the entire tree.
At each time, the branching of the tree has conditional probability in such way the unconditional occurrence probability for all nodes of the tree can be easily calculated. We denote by $\mathcal{P}^{[n]}$, the probability associated with the state of the world in node n.
Similarly to the deterministic model, the problem parameters are as follows:

— S Joint (Major) setup;
— s_i Individual (Minor) setup cost for item type i;
— h_i Per-unit inventory holding cost for item type i;
— g_i Per-unit penalty cost for item type i;
— T Time horizon;
— H Number of periods in tree horizon;
— N Number of item products in the family;
— I_i^0 Initial inventory level of each item i (given as model data).

And the decision variables are:

$x_i^{[n]}$ Replenishment units quantity of item type i in node n;
$I_i^{[n]}$ Inventory level of item type i at the end of the time of node n, (ending inventory);
$k_i^{[n]}$ Lost sale (the amount of demand we fail to meet) of item type i in node n;
$y_i^{[n]}$ Binary variables = 1 if and only if item type i is replenished in node n, i.e. $y_i^{[n]} = 1$ if $x_i^{[n]} > 0$;
$z^{[n]}$ Binary variables taking the value 1 if an order is placed in node n.

Having defined our scenario tree and the associated probabilities of each state node, and by considering a risk-neutral objective of minimizing expected total cost, the Stochastic Coordinated Lot-Sizing Problem (CLSPud) can be stated as:

$$(\mathrm{CLSP_{ud}}) \quad \min \sum_{n\epsilon N} \mathcal{P}^{[n]} \left(Sz^{[n]} + \sum_{i=1}^{N} \left(s_i y_i^{[n]} + h_i I_i^{[n]} + g_i k_i^{[n]} \right) \right), \tag{2.2.1}$$

s.t.

$$I_i^1 = I_i^0 + x_i^1 - (d_i^1 - k_i^1), \qquad i \in \{1, \dots, N\}, \tag{2.2.2}$$

$$I_i^{[n]} = I_i^{[a(n)]} + x_i^{[n]} - \left(d_i^{[n]} - k_i^{[n]} \right), \qquad i \in \{1, \dots, N\}, \ n \in \mathcal{N} \backslash \{1\}, \tag{2.2.3}$$

$$x_i^{[n]} \leq M y_i^{[n]}, \qquad i \in \{1, \dots, N\}, \ n \in \mathcal{N}, \tag{2.2.4}$$

$$\sum_{i=1}^{N} y_i^{[n]} \leq N z^{[n]}, \qquad n \in \mathcal{N}, \tag{2.2.5}$$

$$I_i^{[n]}, x_i^{[n]}, k_i^{[n]} \geq 0; \quad z^{[n]}, y_i^{[n]} \in \{0, 1\}, \qquad i \in \{1, \dots, N\}. \tag{2.2.6}$$

Constraints (2.2.3) ensure demand satisfaction for each item type at demand node n: the ending inventory level of each item at the end of the time period of each node n, should be equal to the ending inventory level at the time period of the immediate predecessor node $a(n)$, plus the item order lot-size in node n, minus the consumed unit amount of this item (difference between demand and lost sales of each item) in node n. In (2.2.2) the flow balance constraint (2.2.3) is stated particularly for the root node of the tree; as its predecessor node is not defined, therefore the starting inventories for all items in the root node $I_i^{[a(n)]}$ are given as the model data. This constraint is explicitly written in order to be in harmony with our computational code.

Constraints (2.2.4) that link lot-sizing variables to the setup binary variables in each node of the tree, state that the replenishment quantity of an item type in each node can be positive only if that item type is replenished at the same node, while constraints (2.2.5) mean that individual item types can only be included in a joint replenishment in each node if the family replenishment is made in this node. Finally, the objective function (2.2.1) accounts for the expected value that computes the sum of orders setup costs, inventory holding costs and lost sales penalty costs during the model time horizon. Note that constraints (2.2.4) can be tightened if big-M is properly replaced for each item in each node, by the maximum quantity amount that can be ordered in this node, which is the sum of the item demands in node n, plus the sum of maximum demand values of this item type in all time bucket succeeding the time of node n. We replaced it with the following constraints in our computational code, while the second one considers the special case for terminal nodes i.e., the nodes in the last time period of the tree horizon ($n \in \mathcal{T}$):

$$x_i^{[n]} \leq \left(d_i^{[n]} + \sum_{t=T(n)+1}^{H} \max_{j \in \Sigma(n,t)} d_i^j \right) y_i^{[n]}, \qquad i \in \{1, \dots, N\}, \ n \in \mathcal{N} \backslash \mathcal{T} \tag{2.2.7}$$

$$x_i^{[n]} \leq d_i^{[n]} y_i^{[n]}, \qquad i \in \{1, \dots, N\}, \quad n \in \mathcal{T}. \tag{2.2.8}$$

As noted above, this stochastic formulation of CLSP is based on the natural formulation of the deterministic problem, Surely there are other mathematical formulation which are more efficient than the classical one, from computational point of view, having the tighter lower bound on the optimal value of CLSP, but we construct our stochastic model based on the natural formulation because of its simplicity and clarity to deal with our problem.

2.3 Scenario Generation

The multi-stage stochastic mixed-integer problems are computationally intensive, so as the first step to build our SP model, we need a parsimonious and effective way for representing demand uncertainty. So we are faced to the scenario generation and sampling issues. In fact, scenario generation is the most important part in stochastic programming models, as a bad scenario generation approach may spoil the whole optimization result. There are several choice for scenario generation approach, but the most standard strategy in particular for demand uncertainty, is based on the scenario tree. A tree consists of possible scenarios of demands and the associated occurrence probabilities, obtaining from a known statistical distribution (our case) or an expert opinion. Indeed, scenario tree is a simple discrete approximation of the assumed statistical distribution of demands that serves as input to our model. However, the computational effort of solving a SP grows exponentially with the branching factor at each node and number of time stages and consequently the multi-stage problem will not be tractable anymore. So, the first point to be considered is that the size of the tree must be quite limited, therefore we should apply a good way to describe the demand distribution by only reduced number of samples
In this study, we assume the normally distributed demands for each item type. In some cases of our computational experiments we consider dynamic demands which allow the demands to vary during the time, while in other cases we assume static demands.

In scenario generation process, after deciding the size of the tree, i.e. the branching factors at each node and the length of the tree time horizon, prior to sample generation, we have to decide if we replicate scenarios across the tree or not. Replication means that at each stage, the tree fans are generated by the same demand samples.
Indeed, two methods (replication or not) are statistically equivalent since at each stage, demands are generated independently of the predecessor nodes, but using different samples, may result in spurious time-dependence. However, one may prefer using different samples, and argue that, in the case of taking ‘bad’ sample, it will not be repeated. Furthermore, when branching factor is one and consequently the node is the unique successor of its predecessor, it make sense to use the expected value of demand rather than a random sample (Brandimarte 2006). In our model, we apply the replication of sample scenarios, although a priori it is not clear which is the best strategy.

In Brandimarte (2006), several sampling generation methods are listed, including the pure random sampling, variance reduction strategies, Latin hypercube sampling and optimized sampling methods. The pure random sampling method may be the simplest approach in which at each stage, given the branching factor per node, we can generate random samples from the multivariate normal distribution, using the standard methods to generate the desired multivariate normal distribution (see Law 2013). An alternative to random sampling, is optimized scenario sampling in which the objective is to generate a set of 'optimized' scenarios by trying to match the known moments of demand (expected value, variance, skewness and kurtosis) as much as possible. It requires the solution of a nonlinear programming model to generate data paths that fit the required statistics (Brandimarte 2001). In this way, a limited number of discrete outcomes will be generated, satisfying the specified statistical properties.

In the case of *normally* distributed demands, assuming to know for each item i, the expected value μ_i, and the variance σ_i^2, and for each pair of items (i, j), the set of covariances σ_{ij}, and knowing the fact that in normal distribution the kurtosis is equal to 3 and the skewness should be zero. Denoting by $d_i^{[s]}$ the demand for item i in node s belonging to a certain branching of size S (note that the parameters are known a priori and they not estimated from data). Then the natural requirements are:

$$\frac{1}{S}\sum_s d_i^{[s]} \approx \mu_i, \quad \forall i,$$

$$\frac{1}{S}\sum_s \left(d_i^{[s]} - \mu_i\right)\left(d_j^{[s]} - \mu_j\right) \approx \sigma_{ij}, \quad \forall i, j,$$

$$\frac{1}{S}\sum_s \frac{\left(d_i^{[s]} - \mu_i\right)^3}{\sigma_i^3} \approx 0, \quad \forall i,$$

$$\frac{1}{S}\sum_s \frac{\left(d_i^{[s]} - \mu_i\right)^4}{\sigma_i^4} \approx 3, \quad \forall i.$$

In this approach, the objective function minimizes the squared error over non-negative demand values $d_i^{[s]}$, by assigning four weights to each term which may be used to fine-tune performance (Brandimarte 2006).

The strategy we apply in the present study, is antithetic sampling that is one possible variance reduction approach and is easy to apply. In some cases antithetic sampling leads to reduction in sample mean variance by inducing some correlation between samples. It produces negatively correlated pairs of samples in such way that pair-averaged samples are independent and in this way it tries to reduce the mean variance of these pair-averaged samples (Brandimarte 2001). In our model it is obtained by taking pairs of symmetric samples with respect to the expected value.

When the tree branching factor is an odd number, one samples is always set to the expected value of demand. In this way we will be able to match the first and third conditions above i.e., the expected value and the skewness of the normal distribution (because of symmetricity in sampling the skewness is zero).

To this aim, given the branching factor *N*, we divide the interval [0, 1] in *N* equally probable intervals, then the center points of these parts are used as the input probabilities to the inverse function of the related cdf (cumulative distribution function) that gives us the demand sample as its output. In this way, the assigned probability to each sample is 1/*N*. actually, this is the "bracket mean" method in which the even moments always and odd moments sometimes are underestimated (Miller and Rice 1983).

We used the MATLAB function "norminv" with the related expected value and standard deviation, to calculate these values. Having obtained the samples, we replicate them across the tree at each time stage.

As an example, assuming a scenario tree in Figure (2) with branching structure [1, 7, 3] we calculate the demand samples for a 5-item family with expected values 80, 100, 120, 70, 90 respectively and the same standard deviation equal to $\sigma = 10$.

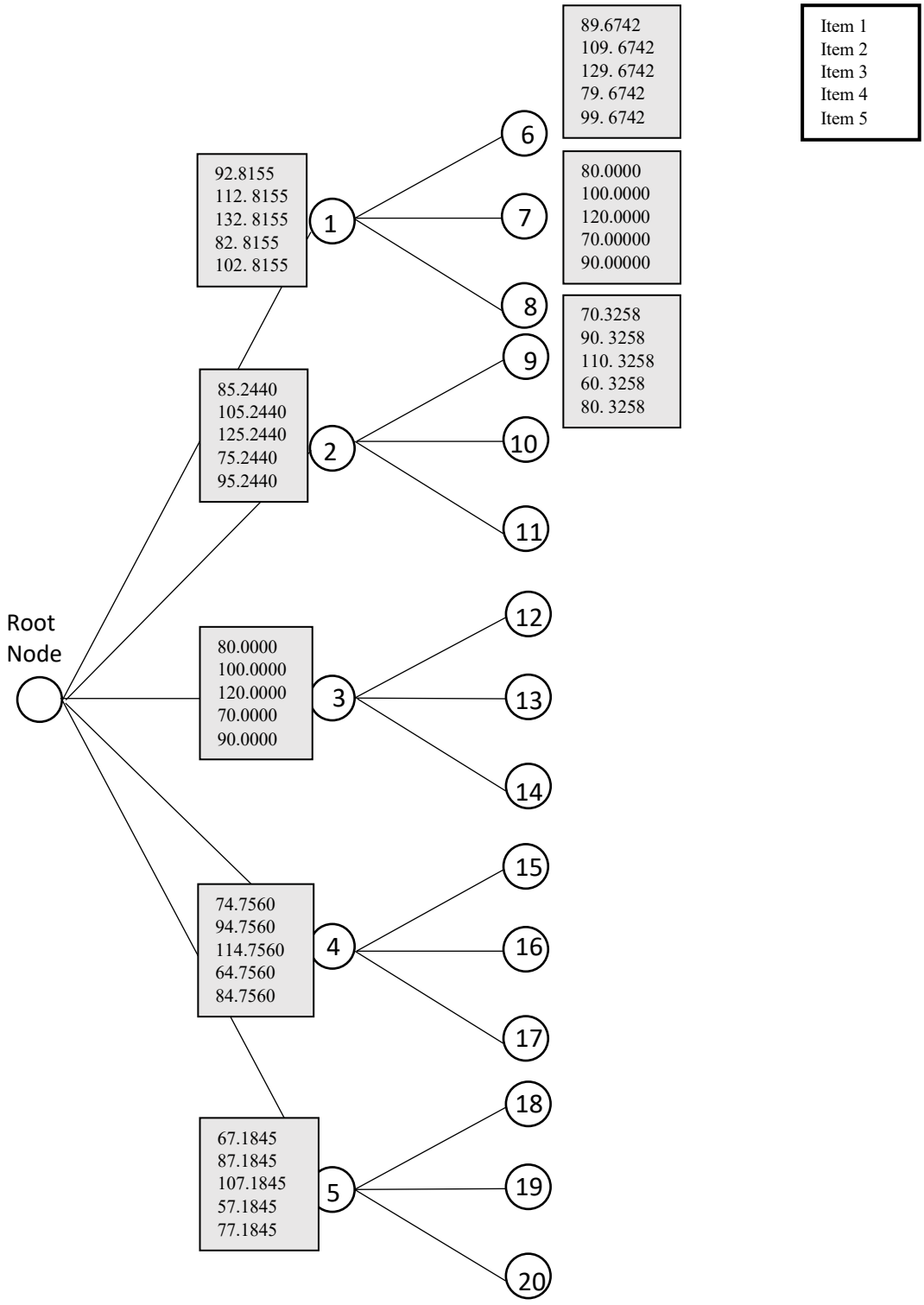


Figure (2): Sample Generation for 3-period scenario tree

2.4 Demand History Generation

After building a multi-stage stochastic planning model, it should be assessed in a real context. In order to test the performance of our multi-stage $CLSP_{ud}$, we use a history of demand values by generating out-of-sample scenarios. This history accounts for the "true" values of item demands. Then we simulate an empirical process within a rolling horizon environment.
In order to generate the out-of-sample scenarios, we apply the pure random sampling method that is the simple approach, by using the distribution parameters same as in our scenario tree. In this way, we can generate a history of demand values at each time period, by a multivariate normal distribution.
Assuming the normally distributed demands, with mean vector μ (vector of expected values of item demands) and covariance matrix Σ (assuming item demands are mutually uncorrelated, the covariance matrix is diagonal, having the variance of item demands as the diagonal elements) the vector of item demand values at each time period is obtained by $D = \mu + ZL$ where Z is the vector of i.i.d standard normal variates and L is the Cholesky upper triangular matrix of Σ such that $\Sigma = L^T L$. In our case assuming all items have the same standard deviation σ and mutually uncorrelated, $D = \mu + Z\sigma$ gives us the vector of item demands at each stage where Z is drawn by the MATLAB function **randn** which generates pseudorandom numbers from the standard normal distribution. Repeating it at each time period we generate the whole demand history for the long run.
In alternative we can use the function **mvnrnd** that generates random variates from the multivariate normal distribution (see Brandimarte 2001 and Law 2013)
These two functions are provided by the Statistics Toolbox in MATLAB.

Chapter 3

Solution methods

A wide range of solution methods are proposed for solving stochastic problems. Typically they can be classified into exact and approximation methods. Obviously, the approximated solution methods, solve the stochastic problem by approximating the objective function while exact methods provides the optimal value of the objective function.
In SP models, when the random variable associated with the uncertain parameter, is assumed to follow a known statistical distribution having finite and moderately small number of realizations (i.e., possible scenarios), the stochastic model can be tackled with building the so-called Deterministic Equivalent (DE) problem, and solving optimally by an optimization algorithm. In contrast, in the SP models with large number of scenarios, the approximation methods such as "Monte Carlo Sampling-based method" may be useful to estimate the objective function value. In the present study, we consider the exact solution approaches. The DE problem of a linear stochastic problems may be easily solved by simplex algorithm while in the case of mixed-integer problems, branch-and-bound algorithm can be applied by casting it into a state-of-the art optimization solver like Cplex or Gurobi.

The MILPs are inherently rigid as the branch-and-bound search tree may be difficult to prune in an efficient way. Combining this issue by the large-scale scenario tree, the resulting stochastic mixed-integer model will not be easily tackled with, even if in the form of deterministic equivalent, and therefore they are encountered to the numerical difficulties when solving with standard solvers. In such situations, to fix the problem, either we may try to keep the branch-and-bound search tree to a reasonable size and losing the optimality guarantee, or we can apply the decomposition methods. In the case of really large-scale SPs, we may resort to the *Decomposition* strategies based on cutting planes or dualization of non-anticipativity constraints. In general, decomposition methods are classified into stage-based decomposition and scenario-based decomposition approaches. L-shaped method and its variants are the known stage-based decomposition method in which each stage t has a number of sub-problems that is related to each node at time stage t. In this work, we focus on the scenario-based decomposition strategies known also as the Lagrangian decomposition in which the objective is to decompose the complicated large-scale problem into scenario sub-problems. In the Lagrangian decomposition method, the complicated Non-Anticipativity (NA) constraints are relaxed by the Lagrangian penalty terms. In this way, the model becomes a set of deterministic scenario sub-problems.

In order to determine the size of a SP model, the mathematical model dimensions and the number of realizations of random vectors need to be considered. One of our interests in this work, is to find the largest $CLSP_{ud}$ model size, in both mathematical model dimension particularly the number of items in the family, and scenario tree size, that can be solved optimally by the solver in a reasonable amount of CPU time. We compare different scenario trees, not only from computational time point of view, but also the performance of them, by running simulations following a rolling horizon strategy, by means of out-of-sample scenarios ("true" demand values). So we start our computational experiments by testing different sizes of stochastic models and solving their DE problems using the optimization solver *Gurobi*. As it is expected, our computational results indicate that even the modest sized mixed-integer stochastic models need many hours to be solved optimally by this standard solver. This encourage us to seek good heuristic approaches for our stochastic mixed-integer models.
Progressive Hedging Algorithm (PHA) is one scenario-based decomposition solution method, proposed in order to tackle with large-scale SPs. Indeed, the PHA converts the stochastic problem into its deterministic equivalent. We applied a PH-base algorithm to our stochastic coordinated lot-sizing problem. Our computational results indicate that our solution strategy is competitive with direct solution of the deterministic equivalent of the problem.
This chapter is organized as the following:
In §3.1 we solve the DE of our stochastic model, optimally by the solver and perform several numerical experiments to evaluate the impact of model size on the time needed to optimally solve the model. Then we evaluate the performance of our SP models using different scenario tree structures, by running simulations
In §3.2 we apply a PH-based algorithm as a heuristic method to our $CLSP_{ud}$ and provide the related computational results.

3.1 Optimal Solution to CLSPud

Even if the best and most mature optimization solvers will struggle with a particular problem that seems straightforward, while another solver may have no difficulty with that one, but fail to find an accurate solution on another, in particular when we are dealing with a mixed-integer problem. A MIP is not convex, so finding the global optimum requires the combination of a traditional convex algorithm with an exhaustive search such as branch-and-bound algorithm. Solvers must perform what is effectively an exhaustive search among the integer variables to determine the correct solution. Even the best solvers cannot guarantee that every moderately-sized mixed integer problems can be solved in a reasonable amount of time. There are some intelligent and innovative ways to speed up that search, and to improve the performance of mixed-integer problem. But there will always be models for which the exhaustive search will simply take too long even if the

problem is formulated properly and it is of reasonable size. One of the particular tips when encountering the mixed-integer problems is to connect and use different solvers that are compatible with our solver.
As mentioned above, to determine the size of a SP model, the mathematical model dimensions and the number of realizations of random vectors need to be considered. In this section we would like to find the largest size of our Stochastic Coordinated Lot-Sizing Problem ($CLSP_{ud}$) which can be solved in a reasonable amount of CPU time. To this aim we consider the *tree branching structure* and the *number of items in the family* as two factors in our model. We also assess different structures of scenario tree, not only from the computational time point of view, but also the performance of the models.
Our computational experiments are carried out using two of the state-of-the-art optimization solvers Gurobi and CVX. Indeed, we model our problem in CVX using MATLAB and choose Gurobi as an active solver for it. So our objective in this section is to compare the computational burden loosely, as the amount of time a good solver, needs to find an optimal solution rather than strictly mathematically, for different sizes of our MIP model.

3.1.1 SP model Running Time

3.1.1.1 The effect of number of items in the family

Our joint replenishment problem is composed of a single-family including different item types. So, as the first phase of our computational experiments, we would like to investigate the effect of increasing the number of item types on the time needed to optimally solve our scenario tree based stochastic model using the *Gurobi* optimizer.*;* In other words, we would like to find out if our stochastic model would be more complex as the size of the model increases in terms of the number of items in the family or not.
Since in the joint replenishment problems, the Joint (Major) cost value relative to other costs (inventory holding, penalty cost and items setup costs), plays an important role in the problem solution, would be advantageous to take into account this ratio as a measure, in our computational experiments as it is considered in (Erenguc 1988) in the deterministic version of the problem. Indeed, we calculate this ratio in the optimal solution, not using the problem data. Therefore the ratio *(total Joint fixed cost) / (total cost)* at optimality is reported in our numerical experiments. The higher the value of this ratio, the more joint cost dominant will be the problem.

To start the computational experiments, we considered a specific medium-sized scenario tree with branching structure [1, 5, 3, 3, 3] whose time horizon is equal to T=5. Then we generated the test problems by considering different values for number of items belonging to {2, 5, 10, 15, 20}, and two category of standard deviation σ=10, 22 of normally distributed demands. In all test problems the inventory holding cost was set to one (h=1) for all item types in all periods, and the penalty cost for each lost sale was considered g=100 for all items constant over all periods. The minor

setup cost was fixed to s=60 in all instances, and the value of joint setup cost belongs to $\{120, 480, 960\}$. We assumed normally distributed demands are dynamic over time, with the following expected values in the test problems consisting of 2 to 5-item family. We repeat this scheme for each 5 additional item types in the family across the test problems.

$d_{t,1} \sim \mathcal{N}(\mu_{t,1}, \sigma)$	$\mu_{t,1} \in \{80, 150, 180, 80, 150\}$	t=1, 2, .., 5	$\sigma \in \{10, 22\}$
$d_{t,2} \sim \mathcal{N}(\mu_{t,2}, \sigma)$	$\mu_{t,2} \in \{100, 180, 120, 100, 200\}$	t=1, 2, .., 5	$\sigma \in \{10, 22\}$
$d_{t,3} \sim \mathcal{N}(\mu_{t,3}, \sigma)$	$\mu_{t,3} \in \{120, 250, 80, 120, 200\}$	t=1, 2, .., 5	$\sigma \in \{10, 22\}$
$d_{t,4} \sim \mathcal{N}(\mu_{t,4}, \sigma)$	$\mu_{t,4} \in \{100, 100, 70, 100, 100\}$	t=1, 2, .., 5	$\sigma \in \{10, 22\}$
$d_{t,5} \sim \mathcal{N}(\mu_{t,5}, \sigma)$	$\mu_{t,5} \in \{80, 40, 90, 80, 80\}$	t=1, 2, .., 5	$\sigma \in \{10, 22\}$

where $d_{t,i}$ is the demand of item type i at time period t.
The optimal solution of each test problem is obtained using *Gurobi* version 7.5.2 solving the (2.2.1)-(2.2.6). The main computational results are summarized in the Table (3) below.

Branching structure: [1, 5, 3, 3, 3]

		σ=10		**σ=22**	
Joint Cost	**Size (N × T)**	**Joint costs Ratio at optimality**	**CPU Time (seconds)**	**Joint costs Ratio at optimality**	**CPU Time (seconds)**
120	2 × 5	0.40	1.04	0.50	0.52
	5 × 5	0.28	0.59	0.28	0.40
	10 × 5	0.16	0.88	0.16	0.64
	15 × 5	0.11	0.84	0.11	0.89
	20 × 5	0.09	1.09	0.09	1.07
480	2 × 5	0.59	4.46	0.57	8.21
	5 × 5	0.51	13.68	0.50	103.37
	10 × 5	0.44	1.31	0.44	1.10
	15 × 5	0.34	1.08	0.34	0.94
	20 × 5	0.28	1.08	0.28	1.18
960	2 × 5	0.54	8.24	0.54	225.27
	5 × 5	0.55	37.33	0.52	1711.18
	10 × 5	0.51	13.75	0.50	125.38
	15 × 5	0.41	12.45	0.51	6.85
	20 × 5	0.44	1.83	0.44	1.90

Table 3: The effect of number of items in the family on the CPU time needed for solving the stochastic model to optimality by Gurobi.

The numerical results show that, in each problem set and at each level of standard deviation of demands, since the test problems are generated with the same probability distribution, in general as the number of item types increased in the family, the problems became less joint cost dominant and it spent less amount of CPU time. So we can say, as the model becomes less joint cost

dominant, it will be relatively easier to solve (needs less CPU time).And generally when the ratio is around 50 percent the problem needs much time to be solved optimally by the solver.
Overally, the problems with 20 items were relatively faster than the models which consist of 2 or 5 item types. For example in the third problem set (joint cost=960) and at the level σ=22, the problem with 5 products was much more difficult to solve than the problem with 20 products in the same set. To interpret such a difference, we may consider their resulted optimal order schedules. The orders in the first optimal schedule take place in the first, second and forth periods while in the problem with 20 items the demands for each period should be optimally satisfied from a replenishment on the same period itself. Therefore, as a preliminary conclusion, we can say, in the joint replenishment problem, increasing the number of item types in the family when fixing the other data in the model, does not certainly imply a more difficult problem in terms of running time; however the number of items has a direct effect on the value of the inventory holding cost and the sum of the item setup costs and therefore on the Ratio considered above. Therefore, what should be considered as a key to specify the degree of computational burden in the joint replenishment problem, is the value of joint cost relative to the other costs in the problem.

3.1.1.2 The effect of joint cost value

During the previous phase of computational experiments, we noticed that the joint setup cost plays an important role in the model running time. In fact the relative value of major (joint) setup cost with respect to the other costs may have a direct effect on the problem running time by an optimization solver (Gurobi in our case). So as the second phase of computational experiments, we would like to evaluate the role of joint setup cost value in the time needed to optimally solve the deterministic equivalent of our multi-stage stochastic mixed-integer problem which is modeled through a scenario tree.
To this aim, we consider one specific medium-sized scenario tree, same as the previous phase, with branching structure [1, 5, 3, 3, 3] and we fix the tree time horizon to T=5. In order to avoid a too large number of experiments, we fix some of the input parameters during this experiment, such as the inventory holding cost which was set to one unit as a reference (h=1); the items setup costs were considered the same for all items in each time period (s=60); and the unit penalty cost was (g=100) for the lost sales. Then we generate the test problem by assigning different values to the joint setup cost and two values for the number of items {5, 10}. Item demands are generated in the same way of the previous experiment. The CPU times resulted for each test problem are reported in the Table (4).

Branching structure: [1, 5, 3, 3, 3]

σ	Joint Cost	5 Items		10 Items	
		Joint costs Ratio at optimality	**CPU Time (seconds)**	**Joint costs Ratio at optimality**	**CPU Time (seconds)**
10	60	0.16	0.67	0.09	0.70
	480	0.51	**13.68**	0.44	1.31
	960	0.55	**37.33**	0.51	**13.75**
	2000	0.52	**8.99**	0.56	**76.70**
	4000	0.36	**239.55**	0.52	**92.93**
	7000	0.49	**12.17**	0.66	**605.37**
	14000	0.65	2.39	0.49	**11.40**
	18000	0.71	1.26	0.55	8.11
	20000	0.73	0.73	0.57	6.28
20	60	0.16	0.44	0.09	0.71
	480	0.50	**103.37**	0.44	1.10
	960	0.53	**1711.18**	0.50	**125.38**
	2000	0.51	**199.99**	0.54	**697.39**
	4000	0.33	**111161.34**	0.51	**305.34**
	7000	0.48	**97.25**	0.64	**2958.97**
	14000	0.63	3.65	0.48	**95.28**
	18000	0.69	3.12	0.53	30.4
	20000	0.71	2.34	0.55	18.22

Table 4: The effect of joint cost value on the CPU time needed for solving the stochastic model to optimality by Gurobi.

It is clear from the reported running times, that for a fixed-size scenario tree and problem dimension (in terms of number of items in the family), the time spent to solve the model to optimality is too sensitive to the joint cost value when fixing the other costs and model data. It means that in our multi-stage mixed-integer problem, even within a given size of problem (i.e., same branching structure of scenario tree, same number of items in the family, same number of constraints), the time spent by the solver *Gurobi,* can vary significantly from one instance to the next. Therefore, even for our medium-size scenario tree there are instances for which solving the model to optimality is drastically time consuming.

In general the test problems with joint cost ratio around 50 percent are much more difficult than other problems. Since in our experimental design the pattern of expected value of demands repeats for each additional 5 items in the family across the test problems, the result of a test problem with 5-itam family at each level of joint cost value S, corresponds to the result of the test problem with 10-item family at the level of 2S joint cost value.

We can easily observe that for each level of problem dimension (number of items in the family) and standard deviation of demands, there is a critical interval for the joint cost values. Indeed, when the joint cost value is outside this interval the problem was drastically easier for the solver comparing to the values belonging to this interval that require much greater computational efforts. For each problem dimension, the results associated with this critical interval, are colored in red. For instance, when σ=10, finding an optimal solution of the model including 5 item types when

the joint setup cost is 4000 takes almost 240 seconds of time, while it spends 13 seconds of CPU time when the joint cost equals to 7000 and only 2 seconds when the joint cost is 14000.

We may be interested in finding the trend of joint cost value impact on the required running time of the model; in other words, to estimate the values of joint cost for which the multi-stage mixed-integer stochastic problem becomes difficult to solve by the solver. Indeed, in each joint replenishment problem, there are two limit cases in which in the first, the joint setup cost value is so relatively small that the demands of each period will be met by a replenishment in the same period, i.e. in all periods we pay the joint replenishment cost; and in the second case the joint setup cost is relatively large enough that all demands for all periods in the time horizon, are satisfied by only one replenishment order in the first period. We call them lower and upper limits. The numerical results show that for the joint setup cost values between these two limits, the time needed to run the model is substantially much more than the required time in problem instances in which the joint cost values are outside this interval when fixing other parameters in the model.
In order to estimate these limits, in the case of dynamic demands (our case) we can heuristically exploit the modified Eisenhut (1975)'s decision criterion applied in the Forward-Pass heuristic approach discussed in the chapter 1, or alternatively, in the case of static demands (the same average family demand in all periods), having the number of items in the family, the average demand for the family, the items setup and inventory holding costs value, and also the length of the tree time horizon, by using the time between joint orders (TBO_{family}) formula that was discussed in the first chapter, we can estimate these two limits of joint cost values. Indeed, in order to estimate the upper limit, TBO_{family} (which is calculated as $TBO_{family} = (\sqrt{2K/hD})$, where K is the joint setup cost, h the per unit inventory carrying cost, and D is the average demands for the product family), should be equal to the length of tree horizon (all demands are satisfied by an order in first period) and it should be one (demands at each period are satisfied by an order at the same period), in order to estimate the lower limit. Having known this fact would be helpful when designing the scenario tree in the stochastic model. We will discuss it in the (3.1.1.4) when we test the performance of our SP model using different scenario tree structures.

3.1.1.3 Tree structure

As noted above, we consider the scenario tree branching structure as a factor in the computational experiments as it can be a severe limiting factor when solving the problem in the computer implementation and so the time needed to solve the model to optimality by the solver. As our numerical experiments will indicate, this role is more substantial on the size of the model, than the number of item types in the family. It is obvious that in the scenario tree-based stochastic problem, the form of the scenario tree determines the total number of scenarios, the length of the planning horizon, the accuracy in the representation of uncertainty, the number of nodes to be explored by the solver and consequently the CPU time needed for solving the problem by the solver, etc. In fact, the branching structure is arbitrary and one may argue that the more branches at each stage, the more our ability to model demand uncertainty; but the number of tree nodes grows easily with

the number of stages, and consequently the more computational effort are needed for solving the model to optimality.

In this phase of our numerical experiments, our objective is to compare the running time needed for solving the stochastic model, in the function of branching factor and length of the tree time horizon. To this aim, since it is quite common in scenario tree generation, to have high initial branching factors and a short time horizon, we fix the tree time horizon to T=5 and let the second period branching factor to increase from 3 to 40. The number of branching in other periods are the same and equal to 3. So that we obtain scenario trees such as [1, 3, 3, 3, 3], [1, 5, 3, 3, 3], [1, 7, 3, 3, 3], [1, 10, 3, 3, 3], etc. we tried to solve each scenario tree model to optimality by Gurobi. The results are provided at two level of number of item types in the family in the Tables (5) and (6). We denoted by NR (not reasonable), the test problems that were not solved in less than 10 hours of running time.

	CPU Time (second)				
	Number of items: 5	**σ=10**			
Joint cost:	**120**	**960**	**4000**	**14000**	**20000**
Branching structure					
[1, 3, 3, 3, 3]	0.39	9.81	46.44	0.73	0.37
[1, 5, 3, 3, 3]	0.49	37.33	239.55	2.39	0.87
[1, 7, 3, 3, 3]	0.60	327.21	10963.84	6.99	2.01
[1, 10, 3, 3, 3]	0.70	758.25	NR	10.17	5.28
[1, 12, 3, 3, 3]	0.95	18927.61	NR	24.44	6.47
[1, 20, 3, 3, 3]	1.87	NR	NR	211.51	24.78
[1, 40, 3, 3, 3]	5.53	NR	NR	6285.55	515.21

Table 5: The effect of increase in the second branching factor in the tree on the CPU time needed for solving the model to optimality; with 5 item types in the family. NR refers to not reasonable amount of CPU time (more than 10 hours).

	CPU Time (second)				
	Number of items: 10	**σ=10**			
Joint cost:	**120**	**960**	**4000**	**14000**	**20000**
Branching structure					
[1, 3, 3, 3, 3]	0.55	8.00	12.66	3.66	2.33
[1, 5, 3, 3, 3]	0.99	13.75	92.93	11..40	6.28
[1, 7, 3, 3, 3]	1.13	90.62	610.44	148.30	15.82
[1, 10, 3, 3, 3]	1.73	320.95	4613.69	478.70	58.99
[1, 12, 3, 3, 3]	1.90	478.14	10847.11	1970.29	183.35
[1, 20, 3, 3, 3]	3.37		NR	6545.10	4749.31
[1, 40, 3, 3, 3]	15.23		NR	9087.33	7323.09

Table 6: The effect of increase in the second branching factor in the tree on the CPU time needed for solving the model to optimality; with 10 item types in the family. NR refers to not reasonable amount of CPU time (more than 10 hours).

We can observe that although the model running time increases when we increase the second branching factor, it depends significantly on the joint cost value. For the joint cost values that lead a TBO_{family} greater than 1 and less than tree time horizon, the CPU time spent, is substantially high. For instance, when the number of item types in the family is 5, the CPU time needed for solving the tree [1, 7, 3, 3, 3] with joint cost 4000 is about 51 times the CPU time spent for solving the tree [1, 20, 3, 3, 3] with joint cost 14000, when fixing other data in the model. However, obviously for a fixed joint cost value, the more branching factor, the more time we spend for solving the model. According to the results, the trees with second branching factor greater than 10 and small branching factors in the next periods, cannot be solved by Gurobi in a reasonable amount of time.

Then, in order to investigate the effect of length of the tree horizon on the spent running time, we consider a medium-sized tree [1, 7, 5, 3, 1] and increase the tree horizon by adding the time stages with branching factor equal to one. The results are reported in the Table (7).

	CPU Time (second)				
	Number of items: 5		**σ=10**		
Joint cost:	**120**	**960**	**4000**	**14000**	**20000**
Branching structure					
[1, 7, 5, 3, 1]	0.40	1336.81	NR	2.83	3.93
[1, 7, 5, 3, 1, 1]	0.51	252.28	NR	44.88	13.78
[1, 7, 5, 3, 1, 1, 1]	0.85	NR	NR	NR	124.68
[1, 7, 5, 3, 1, 1, 1, 1]	1.12	133.04	NR	NR	741.04
[1, 7, 5, 3, 1, 1, 1, 1, 1]	1.30	453.88	NR	NR	NR
[1, 7, 5, 3, 1, 1, 1, 1, 1, 1]	1.34	604.41	NR	NR	NR

Table 7: The effect of increase in tree horizon length on the running time needed for solving the model to optimality; with 5 item types in the family.

Also in this case, the results show that for the joint cost value which lead a planning schedule with TBO_{family} different from 1 and length of the a scenario tree, the computational efforts are drastically high and in the scenario trees with longer time horizon, the solution is not obtained in an reasonable amount of time (less than 10 hours in our experiments).

3.1.2 SP model Performance

The performance of the model is expected to be contingent more on the tree branching structure than other factors. In other words, scenario generation is an important part of the modelling and solving process for stochastic programming models. Of course, scenario generation is (at best) an approximation of the true uncertainty, and a proper way to evaluate the performance must simulate the application of the model to out-of-sample scenarios, according to a rolling horizon strategy (Brandimarte 2006).
Here our objective is to test the performance of SP model using different structures of scenario tree in terms of branching factor and length of the time horizon. To this aim, we ran several computational experiments simulating the out-of-sample scenarios, using different scenario tree instances in terms of branching factor at each stage and the tree time horizon. Having fixed some problem input parameters, like the per unit inventory holding cost that was set to 1 for all items (h=1), the per unit penalty cost which was considered to 100 for all items (g=100) and the starting level of inventory that was set to zero at the beginning of all simulations, (I_{0i} for each i) then the problem versions are generated considering four different value for joint setup cost and three level of standard deviation for item demands. We assumed that the expected value for each item demand is constant during the time.

Although our objective here is not to introduce a scenario tree generation method, but there are some points that should be considered when generating the tree instances. As indicated in the section (2.3), when we are dealing with scenario tree- based stochastic lot-sizing problems, if the length of the tree time horizon is less than TBO, in particular TBO_{family} in joint replenishment problem, it is obvious that it leads to a large gap optimality in the application of such SP model. Therefore, as the first step to generate scenario tree instances, we should have an estimation for the TBO_{family} or more precisely an estimation for order cycle length in our stochastic coordinated lot-sizing problem ($CLSP_{ud}$). To this aim, we used the Expected Value (EV) problem i.e. the model where each stochastic variable is substituted by its deterministic expected value. Since here we assumed item demands are stationary during the long term horizon, the length of the order cycle will be constant during the time. In alternative, we can use the formulation of natural order cycle, *N,* provided in Ballou (1998) for the joint replenishment problem.
Results are expressed in bar charts representing percentage deviations from the optimal solution for the whole horizon with 40-time periods (the solution is meant as the complete-simulation cost). The optimal solution was obtained solving the deterministic 40-period coordinated lot-sizing problem by *Gurobi* optimizer, using demand history.

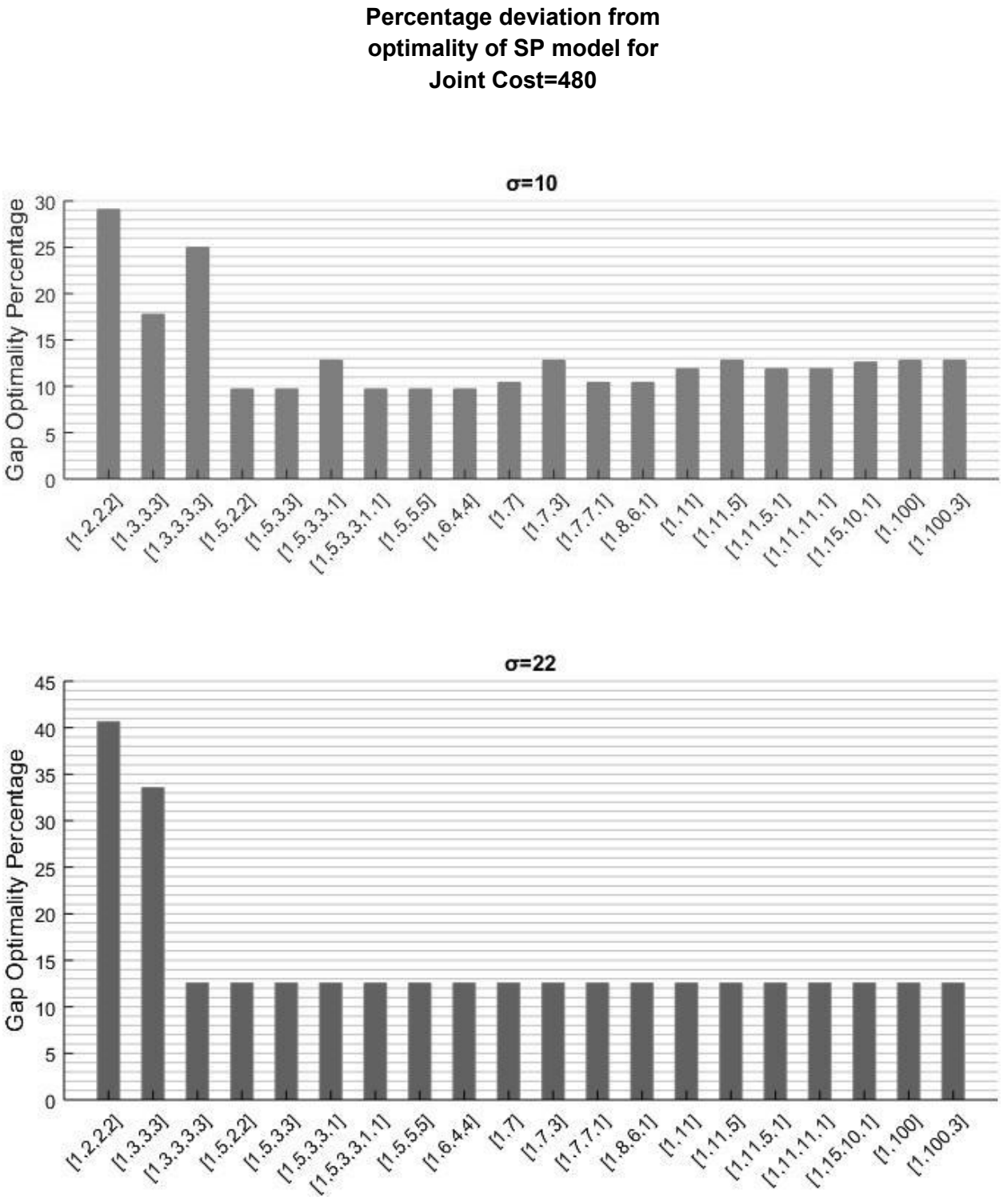


Figure 3: Percentage deviation from optimality of complete-simulation costs providing by SP model, using diverse tree structures with Joint cost=480.

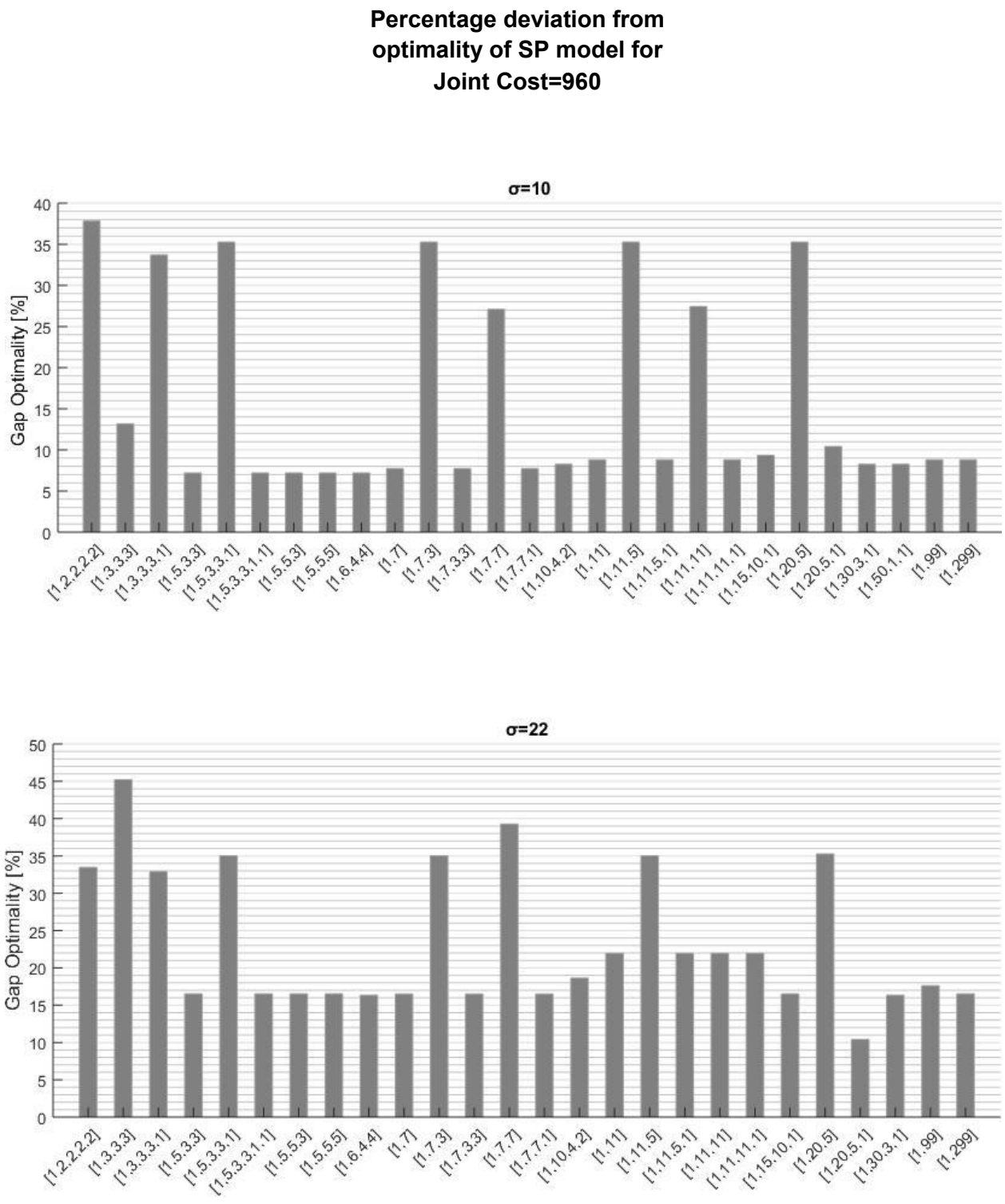


Figure 4: Percentage deviation from optimality of complete-simulation costs providing by SP model, using diverse tree structures with Joint cost=960.

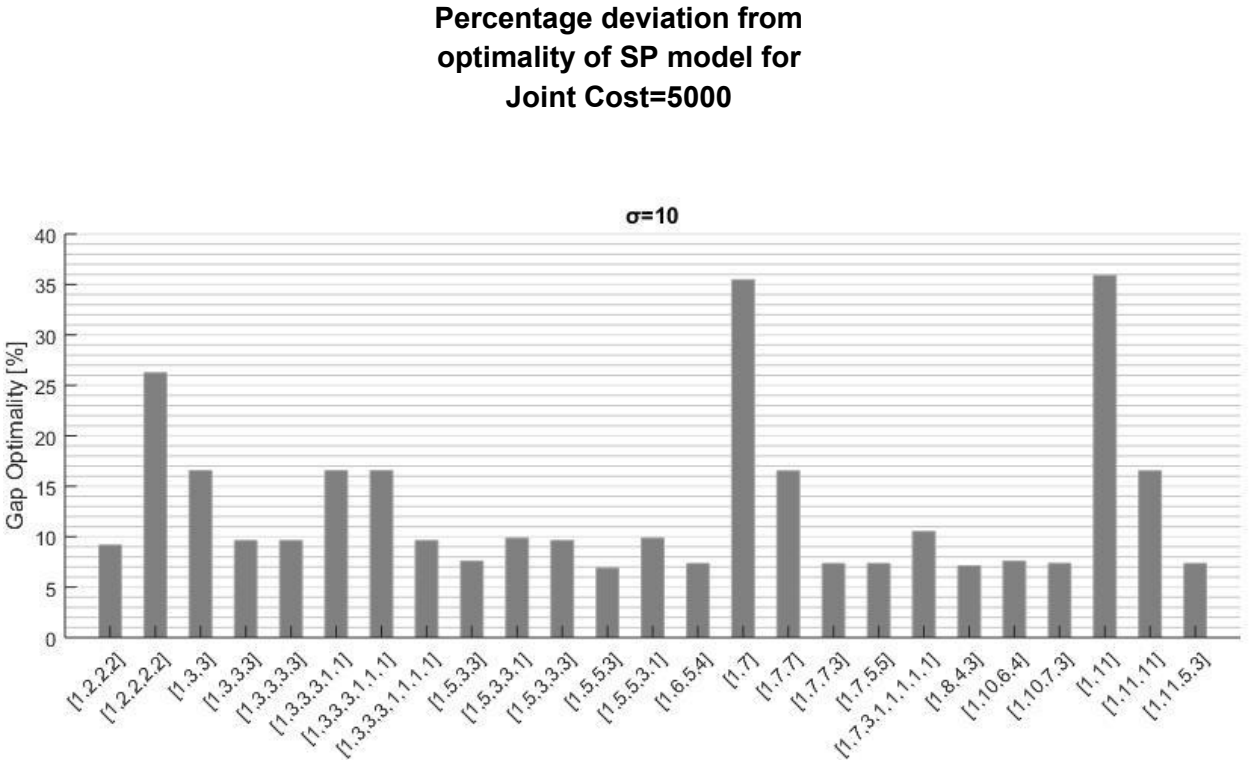


Figure 5: Percentage deviation from optimality of complete-simulation costs providing by SP model, using diverse tree structures with Joint cost=5000.

We can easily observe that in general, at each level of joint cost value and standard deviation of demands, for the scenario trees whose length of horizon is less than TBO_{family}, the performance of SP model is not acceptable. As an obvious example, when joint cost is 5000 and σ=10, the scenario trees with time horizon less than 3 such as [1, 7] or [1, 7, 7] we have a large gap from optimality with respect to the trees with length equal or greater than 4; because in this case the TBO_{family} (resulting from the EV model) is 4. In contrast in the cases with joint cost=480 and 960, these scenario tree performs well since time- between joint orders in these cases in 2.
Therefore, in general scenario trees whose time horizon is equal or is an integer multiple of joint TBO, performs substantially better than other tree structures.

3.2 Decomposition Strategy

As indicated by our computational experiments in the previous section, multi-stage stochastic programming models are computationally intensive, even in the case of continuous underlying problems. The MILPs such as our coordinated lot-sizing problem are inherently rigid as the branch-and-bound search tree may be difficult to prune in an efficient way. Combining this issue by the large-scale scenario tree, the resulting stochastic mixed-integer model will not be easily tackled with, even if in the form of deterministic equivalent, and therefore they are encountered to the numerical difficulties when solving with standard solvers. In such situations, to fix the problem, either we may try to keep the branch-and-bound search tree to a reasonable size and losing the optimality guarantee, or we can apply the decomposition methods.
In the case of really large-scale SPs, we may resort to the *Decomposition* strategies based on cutting planes or dualization of non-anticipativity constraints. In general, decomposition methods are classified into stage-based decomposition and scenario-based decomposition approaches. L-shaped method and its variants are the known stage-based decomposition method in which each stage t has a number of sub-problems that is related to each node at time stage t.
In this work, we focus on the scenario-based decomposition strategies known also as the Lagrangian decomposition in which the objective is to decompose the complicated large-scale problem into scenario sub-problems. In the Lagrangian decomposition method, the complicated Non-Anticipativity (NA) constraints are relaxed by the Lagrangian penalty terms. In this way, the model becomes a set of deterministic scenario sub-problems which are then solved iteratively.

Progressive Hedging (PH) is a decomposition based method that has been used to devise heuristics for hard mixed-integer multi-stage stochastic programs. It basically is a Lagrangian decomposition scheme based on solving and coordinating single scenario sub-problems.
In fact, many of stochastic planning models that are proposed to multi-period problems are actually two-stage and they assumed that the decisions for the overall planning horizon will not be adapted progressively as more and more information is collected (Brandimarte 2006). The progressive hedging unlike the other stochastic methods, takes advantage of the problem structure and it is a strategy that can be applied to multi-stage possibly nonlinear stochastic programming problems.
In Haugen *et al*. (2001), a heuristic approach based on progressive hedging is applied to the uncapacitated single-item lot-sizing problem. We tried to apply a PH-based algorithm to our stochastic coordinated lot-sizing problem.
Before proceeding in the PH approach we present the *split-variable* formulation of Deterministic Equivalent Problem (DEP) of the multi-stage stochastic coordinated lot-sizing problem ($CLSP_{ud}$), which is compatible with our solution strategy we want to apply. In the split-variable formulation we define the decision variables for each time period in each scenario. This formulation is also based on the natural formulation of coordinated lot-sizing problem. As indicated in the Compact formulation of DEP of $CLSP_{ud}$ presented in the section (2.2), we assumed that the random vector ξ follows a known discrete distribution involving a finite number of possible scenarios. More precisely, we explicitly consider a set $\mathcal{S}$ of potential scenarios. Each scenario $s \in \mathcal{S}$ is associated

with a realization of demand occurring with probability $\mathcal{P}_s$, such that the standard axiom $\sum_{s\in\mathcal{S}} \mathcal{P}_s = 1$ is satisfied. Then the DEP formulation of the multi-stage stochastic coordinated lot-sizing problem can be stated as the following while the objective function minimizes the expected value of the CLSP objective function across all possible scenarios, requiring the solutions to be feasible for all scenarios when taken independently (regardless of the scenario which is ultimately realized) and under non-anticipativity constraints.

$$(CLSP_{ud}) \quad \min \sum_{s\in\mathcal{S}} \mathcal{P}^s \sum_{t=1}^{T} \left[S_t z_t^s + \sum_{i=1}^{N} \left(s_{t,i} y_{t,i}^s + h_{t,i} I_{t,i}^s + g_{t,i} k_{t,i}^s \right) \right] \tag{3.2.1}$$

s.t.

$$I_{(t-1),i}^s + x_{t,i}^s - (d_{t,i}^s - k_{t,i}^s) = I_{t,i}^s, \quad \forall\, s \in \mathcal{S}, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \tag{3.2.2}$$

$$x_{t,i}^s \le M y_{t,i}^s, \quad \forall\, s \in \mathcal{S}, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \tag{3.2.3}$$

$$\sum_{i=1}^{N} y_{t,i}^s \le N z_t^s, \quad \forall\, s \in \mathcal{S}, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \tag{3.2.4}$$

$$x_{t,i}^s, I_{t,i}^s, k_{t,i}^s \ge 0, \quad \forall\, s \in \mathcal{S}, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \tag{3.2.5}$$

$$y_{t,i}^s, z_t^s, \in \{0, 1\}, \quad \forall\, s \in \mathcal{S}, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \tag{3.2.6}$$

$$x, I, k, y, z \in \boldsymbol{\mathcal{N}_S}. \tag{3.2.7}$$

where $\boldsymbol{\mathcal{N}_S}$ is the set of all implementable (non-anticipative) solution vectors. The last constraint is added to ensure that feasible solutions are scenario-invariant at each tree node.

The important point in this formulation is that, as we define the decision variables in this way, we must enforce the Non-Anticipativity (or implementability) constraints explicitly; i.e., the decision variables corresponding to different scenarios at the same time t must be equal if the two scenarios are indistinguishable at time t. In fact the way in which the NA requirements are expressed, is not unique and depends on the chosen solution approach. The last non-anticipativity constraint is equivalent to the following ones:

$$x_{t,i}^s = x_{t,i}^{s'}, \quad \forall\, s, s' \in \{s\}_t, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.8}$$

$$I_{t,i}^s = I_{t,i}^{s'}, \quad \forall\, s, s' \in \{s\}_t, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.9}$$

$$k_{t,i}^s = k_{t,i}^{s'}, \quad \forall\, s, s' \in \{s\}_t, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.9}$$

$$y_{t,i}^s = y_{t,i}^{s'}, \quad \forall\, s, s' \in \{s\}_t, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.10}$$

$$z_t^s = z_t^{s'}, \quad \forall\, s, s' \in \{s\}_t, \quad t \in \{1, \dots, T\}; \tag{3.2.11}$$

where for each time period t, $\{s\}_t$ is the set of all scenarios having $\xi_0^s, \dots, \xi_{t-1}^s$ in common with scenario s, in other words the set of scenarios which are not distinguishable from s up to time t.
The following figure (6) depicts the disaggregated form of the standard scenario tree in the Figure (1) in the previous chapter, where the NA constraints on decision variables are specified in red dotted lines.

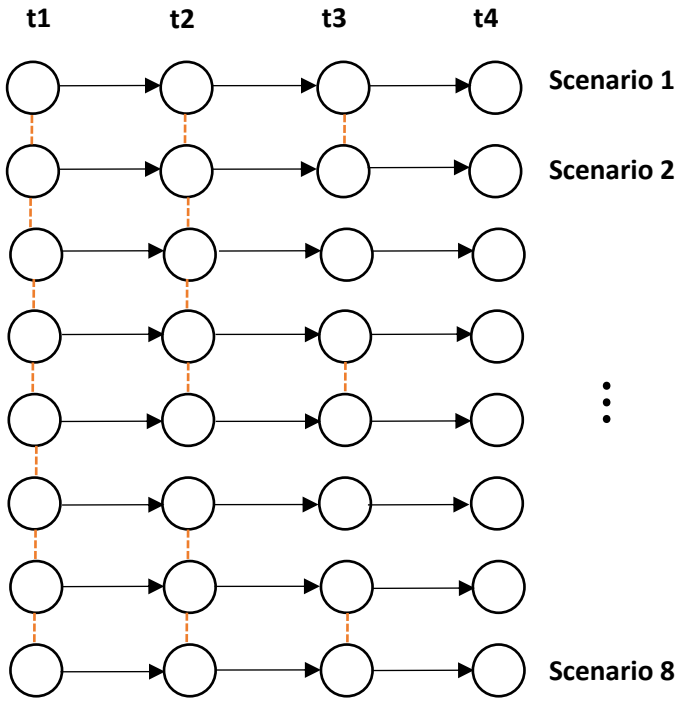


Figure (6): Disaggregated form of scenario tree in the figure (1).

Decision variables in the first period are identical for all 8 scenarios. In time 2, as there are two state nodes, first node share the decisions in scenarios 1 ,2, 3, 4 and 5, while the second node share the decision variables in scenarios 6, 7 and 8, and in the same way in other time periods..

3.2.1 Progressive Hedging Algorithm

The Progressive Hedging Algorithm (PHA) of Rockafellar and Wets (1991) is a scenario-based decomposition method for SP models.
Since we modeled the uncertainty explicitly in our stochastic problem by a set of potential scenarios, the complicating constraints in the DEP of such model, are those related to the non-anticipativity constraints. In PHA the non-Anticipativity constraints are relaxed by an Augmented

Lagrangian Relaxation method and in this way the problem becomes separable by each scenario. Indeed the progressive hedging converts the original stochastic model into a deterministic model.

At each iteration, the mono-scenario sub-problems ($mCLSP_{ud}$) are solved independently as a deterministic problem, with an augmented objective function which includes linear and quadratic terms corresponding to the relaxed non-anticipativity constraints; then the algorithm evaluates if the solutions are implementable or not, in other words if they satisfy the non-anticipativity or not. Naturally, the solutions obtained under each scenario do not initially satisfy NA, and the decision variables that should be equal by NA, take different values; PHA enforces the implementability requirement algorithmically, i.e., it attempts to equalize the decision variables by penalizing deviations from the average value in all scenarios, through augmentation of the objective function. At each iteration a non-necessarily feasible Temporary Global Solution (TGS) is calculated, by collecting and averaging the mono-scenario optimal solutions according to the NA constraints and scenario probabilities. The algorithm stops when complete consensus over all scenarios is met, i.e., when the solutions become implementable, otherwise it updates the Lagrangean Multipliers for each decision variable in each scenario, according to the deviation of each scenario solution from TGS, and iterates again until the NA constraints are "practically" satisfied.

Due to space limitations, we cannot describe the algorithm in complete detail here. For more details, the reader is referred to Rockafellar and wets (1991).
The possibility of solving stochastic problems with very large number of scenarios, makes the PHA interesting, as the extensive form of the stochastic optimization problem is either too difficult to be solved directly, or expensive (e.g., in the course of a branch-and-cut procedure). However, unfortunately, the PH has been proved to be convergent to the optimal solution only in the case of continuous linear program, hence using the same conceptual framework to tackle MILP problems (as first proposed in Løkketangen and Woodruff, 1996) may result in a heuristic approach (Manerba et al. 2017).
In Haugen *et al.* (2001), a heuristic approach based on progressive hedging is applied on the uncapacitated single-item lot-sizing problem. In the same spirit, we applied a PH-based heuristic solution approach to the stochastic coordinated lot-sizing problem ($CLSP_{ud}$). We applied a progressive hedging-based algorithm similar to the one proposed in Manerba *et al.* (2017) for the two-stage stochastic capacitated supplier selection problem. They proposed and applied two PH-based methods to efficiently cope with their stochastic model. Their algorithm had several improvements with respect to the original progressive hedging algorithm, using several acceleration strategies. We applied one of this strategies when relaxing the NA constraints. Indeed, we used the Augmented Lagrangean method only for the binary variables while for the continuous decision variables, the Classical Lagrangean method.
The following pseudo-code describes the overall structure of PH-based algorithm we have applied to $CLSP_{ud}$:

Progressive Hedging-based Algorithm

1: Decompose the $CLSP_{ud}$ by scenarios by applying *Augmented* Lagrangean Relaxation to binary variables constraints and *Classical* Lagrangean Relaxation to continuous decision variables constraints;
2: Set the Lagrangean multipliers and penalty parameters to *zero;*
3: $\nu \leftarrow 0$
4: **for** each scenario $s \in \mathcal{S}$ **do**
5: Solve *optimally* the related $mCLSP_{ud}$ sub-problem
6: **end for**
7: Calculate $\left(\bar{x}^{(0)}, \bar{I}^{(0)}, \bar{k}^{(0)}, \bar{y}^{(0)}, \bar{z}^{(0)}\right)$ at each node of the scenario tree, taking the *conditional* expectation of the solution values of all scenarios at that node;
8: Initialize the penalty parameters with a positive value;
9: Update the Lagrangean multipliers
10: **while** any termination criterion is not satisfied **do**
11: $\nu \leftarrow \nu + 1$
12: **for** each scenario$s \in \mathcal{S}$ **do**
13: Solve the corresponding $mCLSP_{ud}$ sub-problems;
14: **end for**
15: Calculate $\left(\bar{x}^{(\nu)}, \bar{I}^{(\nu)}, \bar{k}^{(\nu)}, \bar{y}^{(\nu)}, \bar{z}^{(\nu)}\right)$ at each node of the scenario tree, taking the *conditional expectation* of the solution values of all scenarios at that node.
16: **if** consensus is met **then**
17: **break**
18: **else**
19: Update the Lagrangean multipliers (and the penalty parameters if it is needed);
20: **end if**
21: **end while**
22: Solve the original model (3.2.1)−(3.2.11) to optimality while fixing the decision variables that met the consensus.

To start the progressive hedging, we need an initial solution as a reference for the Lagrangean relaxation method. To this aim, when having decomposed the $CLSP_{ud}$ using the classical and augmented Lagrangean relaxation method (step 1), we set the price system values and the penalty parameters to zero, then the optimal solutions to the given mono-scenario deterministic sub-problems are independently found. After that, at each node of the scenario tree, a temporary global solution $\left(\bar{x}^{(0)}, \bar{I}^{(0)}, \bar{k}^{(0)}, \bar{y}^{(0)}, \bar{z}^{(0)}\right)$ is easily calculated, taking the *conditional* expectation of the solution values of all scenarios at that node (Steps 2−7). Having this reference solution, now we initialize the penalty parameters to a positive value (step 8) and update the Lagrangean multipliers (step 9). Then we proceed in the core steps of the standard PHA (steps 10−21), by solving

iteratively the $mCLSP_{ud}$ sub-problems, each one as an independent deterministic problem (Steps 12−14). At each iteration, the global solution $(\bar{x}^{(\nu)}, \bar{I}^{(\nu)}, \bar{k}^{(\nu)}, \bar{y}^{(\nu)}, \bar{z}^{(\nu)})$ is obtained at each node of the scenario tree (step 15), by taking the *conditional* expectation of the solution values of all scenarios at that node (see 3.2.1.2) and it stops if the complete consensus is achieved over all scenarios, i.e., when the global solution becomes implementable for all decision variables; otherwise the PHA updates the Lagrangean multipliers for each scenario at each time period and the penalty parameters if it is needed. Then the PHA iterates in the same way until satisfying any termination criteria.

At the end of the algorithm, either the complete convergence is obtained or any termination criterion stops it before achieving the convergence for all decision variables. In the second case we fix the value of those decision variables that satisfy the consensus, to their value obtained by the last global solution and solve the original stochastic model (3.2.1)−(3.2.11) which is now a pure linear stochastic problem or still a mixed-integer stochastic problem, but much easier than the original mixed-integer problem to be solved by the optimal procedure.

Now we discuss in details, the main steps of our PH-based algorithm in the following sections.

3.2.1.1 Scenario Decomposition

Considering the original stochastic model (3.2.1)−(3.2.11), in order to relax the complicated non-anticipativity constraints (3.2.8)−(3.2.11), we firstly substitute them by the following constraints:

$$x_{t,i}^{s'} = \bar{x}(\{s\}_t)_i \quad \forall s' \in \{s\}_t, \quad i \in \{1, 2, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.12}$$

$$I_{t,i}^{s'} = \bar{I}(\{s\}_t)_i \quad \forall s' \in \{s\}_t, \quad i \in \{1, 2, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.13}$$

$$k_{t,i}^{s'} = \bar{k}(\{s\}_t)_i \quad \forall s' \in \{s\}_t, \quad i \in \{1, 2, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.14}$$

$$y_{t,i}^{s'} = \bar{y}(\{s\}_t)_i \quad \forall s' \in \{s\}_t, \quad i \in \{1, 2, \dots, N\}, \quad t \in \{1, \dots, T\} \tag{3.2.15}$$

$$z_t^{s'} = \bar{z}(\{s\}_t) \quad \forall s' \in \{s\}_t, \quad t \in \{1, \dots, T\} \tag{3.2.16}$$

where for each time period t, $\{s\}_t$ is the set of all scenarios having $\xi_0^s, \dots, \xi_{t-1}^s$ in common with scenario s, in other words the set of scenarios which are not distinguishable from s up to time t. And the $\bar{x}(\{s\}_t)_i$, $\bar{I}(\{s\}_t)_i$, $\bar{k}(\{s\}_t)_i \geq 0$ and $\bar{y}(\{s\}_t)_i$, $\bar{z}(\{s\}_t) \in \{0,1\}$ are the conditional expectations at each node of the tree, relative to the relevant scenario bundle, i.e., the conditional expectation for the solution values of all scenarios that have this node in common at time t. So they are relative to the available information. In other words they are the implementable solutions that can be interpreted as the projection on the set of non-anticipativity policies. Then as used in Manerba *et al.* (2017), in order to relax these non-anticipativity constraints, we apply the *Classical* Lagrangean relaxation technique to relax the constraints (3.2.12), (3.2.13) and (3.2.14) which are related to the continuous variables $x, I, k,$ and the *Augmented* Lagrangean method for the NA

constraints (3.2.15) and (3.2.16) related to the binary variables y, z. This yields the following objective function:

$$\min \sum_{s\in\mathcal{S}} \mathcal{P}^s \Bigg\{ \sum_{t=1}^{T} S_t z_t^s + \sum_{t=1}^{T}\sum_{i=1}^{N} \left(s_{t,i} y_{t,i}^s + h_{t,i} I_{t,i}^s + g_{t,i} k_{t,i}^s\right) + \sum_{t=1}^{T} \lambda_t^s \left(z_t^s - \bar{z}(\{s\}_t)\right) + \sum_{t=1}^{T}\sum_{i=1}^{N} \pi_{t,i}^s \left(y_{t,i}^s - \bar{y}(\{s\}_t)_i\right) + \sum_{t=1}^{T}\sum_{i=1}^{N} \gamma_{t,i}^s \left(I_{t,i}^s - \bar{I}(\{s\}_t)_i\right) + \sum_{t=1}^{T}\sum_{i=1}^{N} \eta_{t,i}^s \left(k_{t,i}^s - \bar{k}(\{s\}_t)_i\right) + \sum_{t=1}^{T}\sum_{i=1}^{N} \mu_{t,i}^s \left(x_{t,i}^s - \bar{x}(\{s\}_t)_i\right) + \frac{1}{2}\sum_{t=1}^{T} \rho_1 \left(z_t^s - \bar{z}(\{s\}_t)\right)^2 + \frac{1}{2}\sum_{t=1}^{T}\sum_{i=1}^{N} \rho_2 \left(y_{t,i}^s - \bar{y}(\{s\}_t)_i\right)^2 \Bigg\},$$

where $\mu_{t,i}^s$, $\gamma_{t,i}^s$, $\eta_{t,i}^s$, $\pi_{t,i}^s$ and λ_t^s are the Lagrangean multipliers for relaxed constraints (3.2.12)–(3.2.16) respectively. The parameters ρ_1 , ρ_2 , are the PH penalty parameters for the constraints related to binary variables (3.2.16) and (3.2.15) respectively.

The main characteristic of this PH-based algorithm with respect to the original one is that here we do not consider the quadratic expression to penalize the deviation of continuous variables from the temporary global solution and in this way, the linearity of the objective function is maintained. So, given the binary condition of z, y variables, it can be rewritten as:

$$\min \sum_{s\in\mathcal{S}} \mathcal{P}^s \Bigg\{ \sum_{t=1}^{T} z_t^s \left(S_t + \lambda_t^s + \frac{\rho_1}{2} - \rho_1 \bar{z}(\{s\}_t)\right) + \bar{z}(\{s\}_t)\left(\frac{\rho_1}{2} - \lambda_t^s\right) + \sum_{t=1}^{T}\sum_{i=1}^{N} y_{t,i}^s \left(s_{t,i} + \pi_{t,i}^s + \frac{\rho_2}{2} - \rho_2 \bar{y}(\{s\}_t)_i\right) + \bar{y}(\{s\}_t)_i \left(\frac{\rho_2}{2} - \pi_{t,i}^s\right) + \sum_{t=1}^{T}\sum_{i=1}^{N} I_{t,i}^s \left(h_{t,i} + \gamma_{t,i}^s\right) - \bar{I}(\{s\}_t)_i \gamma_{t,i}^s + \sum_{t=1}^{T}\sum_{i=1}^{N} k_{t,i}^s \left(g_{t,i} + \eta_{t,i}^s\right) - \bar{k}(\{s\}_t)_i \eta_{t,i}^s + \sum_{t=1}^{T}\sum_{i=1}^{N} x_{t,i}^s \mu_{t,i}^s - \bar{x}(\{s\}_t)_i \mu_{t,i}^s \Bigg\}.$$

Now our model is decomposed by scenarios, so the following sub-problem must be solved for each mono-scenario s, *at each iteration* of the PHA (but we omit the PHA iteration index in the interest of clarity):

$$(\text{mCLSP}_{\text{ud}}) \quad \min \sum_{t=1}^{T} z_t^s \left(S_t + \lambda_t^s + \frac{\rho_1}{2} - \rho_1 \bar{z}(\{s\}_t) \right) + \bar{z}(\{s\}_t) \left(\frac{\rho_1}{2} - \lambda_t^s \right)$$
$$+ \sum_{t=1}^{T} \sum_{i=1}^{N} y_{t,i}^s \left(s_{t,i} + \pi_{t,i}^s + \frac{\rho_2}{2} - \rho_2 \bar{y}(\{s\}_t)_i \right) + \bar{y}(\{s\}_t)_i \left(\frac{\rho_2}{2} - \pi_{t,i}^s \right)$$
$$+ \sum_{t=1}^{T} \sum_{i=1}^{N} I_{t,i}^s (h_{t,i} + \gamma_{t,i}^s) - \bar{I}(\{s\}_t)_i \gamma_{t,i}^s + \sum_{t=1}^{T} \sum_{i=1}^{N} k_{t,i}^s (g_{t,i} + \eta_{t,i}^s) - \bar{k}(\{s\}_t)_i \eta_{t,i}^s$$
$$+ \sum_{t=1}^{T} \sum_{i=1}^{N} x_{t,i}^s \mu_{t,i}^s - \bar{x}(\{s\}_t)_i \mu_{t,i}^s \qquad (3.2.17)$$

s.t.

$$I_{(t-1),i}^s + x_{t,i}^s - I_{t,i}^s + k_{t,i}^s = d_{t,i}^s, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \qquad (3.2.18)$$

$$x_{t,i}^s \leq M y_{t,i}^s, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \qquad (3.2.19)$$

$$\sum_{i=1}^{N} y_{t,i}^s \leq N z_t^s, \quad t \in \{1, \dots, T\}, \qquad (3.2.20)$$

$$x_{t,i}^s, I_{t,i}^s, k_{t,i}^s \geq 0, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}; \qquad (3.2.21)$$

$$y_{t,i}^s, z_t^s \in \{0,1\}, \quad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}. \qquad (3.2.22)$$

Therefore in this way, we maintain the linearity of the objective function in the sub-problems. Each sub-problem is a deterministic version of CLSP, devoted to individual scenario, in which the stochastic item demands are substituted by the values to be realized under that scenario.
Although in progressive hedging algorithm it is not necessary to optimally solve the scenario problems (since in any case convergence is problematic) and so these scenario sub-problems can be solved by any solution method valid to the deterministic CLSP, we found the sub-problems solution using the *Gurobi* optimizer.

3.2.1.2 Temporary Global solution calculation

At *each iteration* υ of the algorithm, given the solutions of all mono-scenario sub-problems, at each node of the scenario tree, the temporary global solution $(\bar{x}, \bar{I}, \bar{k}, \bar{z}, \bar{y})$ can be easily obtained, by taking the conditional expectation for all solution values related to all scenarios that have this node in common (we omit the PHA iteration index for the sake of clarity):

$$\bar{z}(\{s\}_t) = \sum_{s' \in \{s\}_t} \frac{\pi^{s'} z_t^{s'}}{\mathcal{P}(\{s\}_t)} \qquad t \in \{1, \dots, T\},$$

$$\bar{y}(\{s\}_t)_i = \sum_{s' \in \{s\}_t} \frac{\pi^{s'} y_{t,i}^{s'}}{\mathcal{P}(\{s\}_t)} \qquad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\};$$

$$\bar{x}(\{s\}_t)_i = \sum_{s' \in \{s\}_t} \frac{\pi^{s'} x_{t,i}^{s'}}{\mathcal{P}(\{s\}_t)} \qquad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\};$$

$$\bar{I}(\{s\}_t)_i = \sum_{s' \in \{s\}_t} \frac{\pi^{s'} I_{t,i}^{s'}}{\mathcal{P}(\{s\}_t)} \qquad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\};$$

$$\bar{k}(\{s\}_t)_i = \sum_{s' \in \{s\}_t} \frac{\pi^{s'} k_{t,i}^{s'}}{\mathcal{P}(\{s\}_t)} \qquad i \in \{1, \dots, N\}, \quad t \in \{1, \dots, T\}.$$

As indicated above, for each time period t, $\{s\}_t$ is the set of scenarios which are not distinguishable from s up to time t.

3.2.1.3 Lagrangean multipliers and Penalty parameters adjustment

Initially the Lagrangean multipliers or price system values (dual variables) are set to zero. Then at each iteration of PHA, having the aggregated solutions at each node of the scenario tree, they are updated using the subgradient scheme. Indeed, they are recalculated for each scenario at each time period as the following:

$$\lambda_t^{s(\nu)} \leftarrow \lambda_t^{s(\nu-1)} + \rho_1 \left(z_t^{s(\nu)} - \bar{z}_t^{(\nu)} \right);$$

$$\pi_{t,i}^{s}{}^{(\nu)} \leftarrow \pi_{t,i}^{s}{}^{(\nu-1)} + \rho_2 \left(y_{t,i}^{s}{}^{(\nu)} - \bar{y}_{t,i}{}^{(\nu)} \right);$$

$$\mu_{t,i}^{s}{}^{(\nu)} \leftarrow \mu_{t,i}^{s}{}^{(\nu-1)} + \rho_3 \left(x_{t,i}^{s}{}^{(\nu)} - \bar{x}_{t,i}{}^{(\nu)} \right);$$

$$\gamma_{t,i}^{s}{}^{(\nu)} \leftarrow \gamma_{t,i}^{s}{}^{(\nu-1)} + \rho_4 \left(I_{t,i}^{s}{}^{(\nu)} - \bar{I}_{t,i}{}^{(\nu)} \right);$$

$$\eta_{t,i}^{s}{}^{(\nu)} \leftarrow \eta_{t,i}^{s}{}^{(\nu-1)} + \rho_5 \left(k_{t,i}^{s}{}^{(\nu)} - \bar{k}_{t,i}{}^{(\nu)} \right).$$

The penalty parameters for each iteration $\nu > 0$, are set to a positive value. In fact, we tested several methods for adjusting the value of penalty factors such as the cost-proportional method (see Watson *et al*. 2010) in which the value of ρ is in proportion to the unit-cost of the related decision variable. Although, they conclude that the per-iteration penalty parameter ρ^{ν} may be more appropriate for some problem applications, in this work we fixed the penalties to 0.8 for all decision variables in our computational experiments as it is considered in Haugen *et al*. (2001); because we realized that our solution method (binary convergence, see 3.2.1.4) is not sensitive to the value of penalty factors.

3.2.1.4 Solution strategies

The Progressive hedging algorithm provably converges to an optimal solution (Rockafellar and wets 1991) in the case of continuous decision variables. Since our stochastic problem is a mixed-integer and therefore a non-convex problem, such convergence guarantee does not exist and so, applying the progressive hedging framework to MILP problems may result in a heuristic approach. As mentioned in Manerba *et al.* (2017), the basic PHA of Rockafellar and Wets has several flaws. First, it is not able to find any feasible solution until the complete consensus is obtained, i.e. until near end of the procedure. Second, the convergence may be very slow, and consequently deteriorating the effectiveness of the algorithm while the number of iterations increases. However, several enhancement strategies to the basic PHA have been introduced to ensure and accelerate convergence in the case of mixed-integer problems (see Watson and Woodruff 2010).
In this work, we have applied some acceleration strategies to the PH-based algorithm to tackle our MILP, taking into account the model properties.
Firstly, in the decomposition step of the algorithm, since the quadratic mixed-integer programs are much more difficult to solve than the linear mixed-integer programs, in order to maintain the linearity of the objective function in sub-problems, we have applied the pure Lagrangean relaxation method to relax the non-anticipativity constraints related to continuous decision variables (x, I, k) , instead the augmented Lagrangean relaxation method which is usual approach in progressive hedging. Because the augmented Lagrangean method includes a quadratic penalty (proximal) term that complicates the problem, but in the case of binary decision variables (y, z), the quadratic terms simply reduce to linear penalty terms, as they only take 0-1 values.
Secondly, we just look for the Binary Convergence, in particular the z-variables, because:
In general, the greater the value of the variables, the more the iterations required to reach consensus among these variables (Manerba et al. 2017). In our mixed-integer problem, the continuous variables in particular the x-variables which are lot-size and the I-variables the holding inventories may take large values, hence they need much more computational efforts to be converged. Moreover it is more difficult to evaluate the convergence for continuous variables than for the binaries, because an integrality check is not enough. Hence in order to speed up the algorithm, we terminate it when all the binary decisions in particular the z-variables which are the joint setups, satisfy the NA constraints for a number of iteration, by fixing them to the converged value. Then we set the y-variables which are the minor setup variables, according to the fixed z-variables, in such a way: in each period, if the z-variable is fixed to 1, i.e., if we must pay the joint setup cost in this period, we also fix the y-variables for all items in this period, in contrast if the z is fixed to 0 in each period, we also set the y- variables to 0 for all items. Then we switch the solution algorithm to a regular branch and cut scheme, because by fixing all binary variable to their converged values, the remaining problem will be a pure LP that can be easily solved by the solver. As is usual in the PHA when applied to the mixed-integer stochastic problems, when a discrete variable satisfy the NA constraints for a number of iteration, it will be fixed to the converged value. But fixing the variables in such a manner may lead to sub-optimal solutions. But we do such fixing when the consensus is met for all z -binary variables (binary convergence).

The combination of these factors has enabled implementation of the PH algorithm to be effective, with relatively good quality of solutions. The primary potential advantage of PH over direct solution of either the explicit or implicit extensive form lies in scalability, when very large numbers of scenarios are considered. In this cases, the extensive form of stochastic model is often computationally intractable for optimization solvers, in terms of memory, runtime or both.

3.2.1.5 Computational results

In our computational experiments, the stochasticity structure is similar to in Haugen *et al.* (2001), where a heuristic approach based on progressive hedging is applied on the uncapacitated single-item lot-sizing problem.

In the scenario tree, we fixed the branching factor to 3 for all stages after the first, so there are three realizations of demands for each item at each stage. We generate the normally distributed demands for items, symmetrically at each stage. The expected value of demands varies among items and during the tree time horizon but we fix the standard deviation of demands to $\sigma = 10$ for all items. We generate the stochastic test problems considering two values for the length of the tree time horizon $T = \{5, 6\}$.

For each stochastic test case the deterministic equivalent problem is built by solving (2.2.1)-(2.2.6) directly, resulting in a large block-diagonal mixed-integer linear programming model. They are used to test the performance of our progressive hedging solution method. Some information about the scenario trees and the dimension of the resulting DE problems are reported in the table below:

Scenario tree size			**Deterministic Equivalent (DE) dimension**		
Stages	Items	Scenarios	Variables	0/1 Integers	Constraints
5	2	81	1210	363	726
5	5	81	3025	726	1815
5	8	81	4840	1089	2904
6	2	243	3640	1092	2184
6	5	243	9100	2184	5460
6	8	243	14560	3276	8736

Table 8: Testcase and DE dimensions.

As described by the first step of the PH-based Algorithm, to obtain the first reference for the aggregated solutions $\left(\bar{x}^{(0)}, \bar{I}^{(0)}, \bar{k}^{(0)}, \bar{y}^{(0)}, \bar{z}^{(0)}\right)$ which is used in the first iteration of the algorithm, we solve each scenario sub-problem to optimality and calculate simply the average of them. We solved each sub-problem optimally using the *Gurobi* solver. This temporary policy global solution is implementable but not necessarily admissible. In the case the solution of all scenarios are equal, this aggregated solution (temporary global solution) is admissible and one can stop the progressive

hedging algorithm here and report this solution. There are other PH-based algorithm applied to stochastic mixed integer problems, where this first reference solution is given by solving the Expected Value (EV) problem (see Manerba et al. 2017) or other methods.
As our results indicate, there are many cases in which the *binary* variables (i.e. z-variables and y-variables, the joint setup and items setup variables respectively) solution of scenarios are all initially convergent (it depends obviously to the joint cost value), and consequently the aggregated solution is initially admissible and can say the PHA is binary converged. Indeed as seen in the previous chapter, in the coordinated lot-sizing problems the relative value of the joint setup cost, plays an important role in the problem solution, for example if the joint cost value is large or small enough, the different scenarios may result in the same binary solutions (the setups take place in the same periods in all scenarios) and consequently the algorithm is binary converged.
Having the binary converged problem, we can build easily the deterministic equivalent of the problem which is now a Linear Problem (we call it LDE) and solve it using an optimization solver. All mono-scenario sub-problems ($mCLSP_{ud}$) and DE problems are coded in Matlab as CVX optimization models and solved by selecting the *Gurobi* as the solver. The computer used to run the both DE and LDE models is a Lenovo Yoga notebook PC, with an Intel(R) core(TM) i3-606U @ 2.00 GHz processor. The main results are reported in the next two tables where PHA/DE refers to the solution of LDE to the solution of the DE of the stochastic programming model.

		$[\mathbf{1},\mathbf{3},\mathbf{3},\mathbf{3},\mathbf{3}]$		
		PHA /DE Ratio		
Joint Cost	*Size (N x T)*	Solution Values	Solution Times	Number of Iterations To Binary Convergence
120	2 x 5	1.0000	0.8283	0
	5 x 5	1.0000	0.8194	0
	8 x 5	1.0000	0.7839	0
480	2 x 5	1.0000	0.6748	1
	5 x 5	1.0135	0.1316	1
	8 x 5	1.0000	0.0978	1
960	2 x 5	1.0000	0.7151	1
	5 x 5	1.0308	0.1139	1
	8 x 5	1.0671	0.0346	1
2000	2 x 5	1.0000	0.2465	0
	5 x 5	1.0000	0.3711	0
	8 x 5	1.0011	0.1379	0
4000	2 x 5	1.0000	0.8636	0
	5 x 5	1.0038	0.0301	0
	8 x 5	1.0000	0.1784	0

Table 9: Solution quality and running time ratios.

		$[1, 3, 3, 3, 3, 3]$		
		PHA/DE Ratio		
Joint Cost	**Size (N x T)**	**Solution Values**	**Solution Times**	**Number of Iterations To Binary Convergence**
120	2 x 5	1.0000	0.5851	0
	5 x 5	1.0000	0.7973	0
	8 x 5	1.0000	0.7093	0
480	2 x 5	1.0000	0.7257	0
	5 x 5	1.0186	0.1114	1
	8 x 5	1.0000	0.0056	1
960	2 x 5	1.0040	0.1325	1
	5 x 5	1.0000	0.0661	0
	8 x 5	1.0065	0.0165	1
2000	2 x 5	1.0000	0.0161	1
	5 x 5	1.0000	0.0278	1
	8 x 5	1.0000	0.0260	1
4000	2 x 5	1.0000	0.1327	0
	5 x 5	1.0000	0.0066	1
	8 x 5	1.0000	0.0430	1

Table 10: Solution quality and running time ratios.

According to the obtained results, in many cases the implementable solution is initially obtained. In fact this situation happens for the values of joint cost that leads a same planning schedule (in terms of binary decisions and in particular z-variables that refer to joint orders) in all scenarios, for example when the joint cost value is small enough relatively to other costs, the optimal schedule for all scenarios is the same (in all mono-scenarios the z-variables are equal to 1); Or in contrast, when the joint cost is large enough and also in this case the solution of all scenarios is the same (z-variables in the first period of all mono-scenarios are 1 and in other periods are 0) and consequently the algorithm is initially binary converged. For the other value of joint cost which imply the schedules different from these two limit cases, the binary convergence is obtained in fist iteration of the PH algorithm. As an instance when the joint cost value is 120 for all problem dimension, we have initially binary convergence; but for joint cost 4000, in the tree [1, 3, 3, 3, 3], since the length of the tree in less than TBO_{family} and so we obtained the binary convergence initially, while in the tree [1, 3, 3, 3, 3, 3], the value 4000 does not lead a same schedule in all mono-scenarios.

As can be seen, the PH-based algorithm we applied, apart from the fast performance, has an acceptable quality of solution. We believe that this PH-based algorithm we applied to $CLSP_{ud}$

would be more efficient in the case of larger scenario trees which cannot be solved directly using solvers.

3.3 The value of the stochastic model solution

The stochastic programs and in particular the multi-stage stochastic programming models are computationally problematic even in the case of continuous decision variables. Adding the complexity of mixed-integer models, the resulted stochastic model will be intensive to be tackled with in the realistic sized applications. Therefore, before solving a stochastic model, it can be useful to consider and evaluate the solution of some simpler methods based on the available level of information related to the uncertain parameters. In fact, it may be useful to know how a deterministic approach based on the expected value of the uncertain parameter, performs in a real stochastic framework.

To this aim and in order to analyze the importance of modeling the demand uncertainty explicitly through a stochastic $CLSP_{ud}$ model, in comparison to the simpler methods that are based on the expected vale of demands, we employ and calculate one of the well-known measures in the stochastic programming literature. The *Value of the Stochastic Solution* (VSS), is expressed as VSS=EEV-RP, where EEV denotes the expected result of using the expected value problem (EV). The EV is a deterministic model where all random variables are replaced by their expected values; and RP (recourse problem solution) is the value of objective function in the deterministic equivalent of the stochastic model. In fact, the VSS compares the *here* and *now* solution with the expected values approaches (Escudero et al. 2007). It is the expected gain from solving a SP model rather than its deterministic counterpart. A large VSS means that uncertainty is important for the optimal solution, and the deterministic solution is "bad" (Maggioni et al. 2011).

Indeed, these concepts had been firstly defined and studied in literature for the two-stage stochastic models; Escudero *et al.* generalized them to the multi-stage models, by defining a chain of values VSS_t which is based on the information available until stage t of the related deterministic model. They calculates the expected value of EV (average scenario solution) in each time periods, denoting by EEV_t. In other words, it expressed the expected result in t, of using the expected value solution. So EEV_1 is equal to the RP solution; EEV_2 is the solution of recourse problem in which the first-stage decisions are fixed by the optimal values of EV model; and in general EEVt, for $t = 2, \dots, T$ is the optimal value of RP where decision variables until stage $t-1$, are fixed at optimal values of EV model. Then the VSS_t, the value of stochastic solution in time period t, is defined as:

$$VSS_t = EEV_t - RP \quad t = 1, \dots, T.$$

We consider a $CLSP_{ud}$ consisting of a family of 3 item types, in which uncertainty in items demands is modeled through one special tree with branching structure [1, 5, 3, 3, 3]. The major and minor costs are 800 and 60 respectively. The inventory holding cost is set to 1 and the per-unit

penalty cost is considered 100 for each lost sale. Item demands are considered dynamic over time, following the normal distribution. Figure (8) shows the demand samples for each item in each time period.

First, we solved the compact version of deterministic equivalent (DE) of the stochastic model using *Gurobi* optimizer. The optimal solution of RP is depicted in the figure (9). We can observe that, the order replenishments are placed in the nodes belonging to the first, second and fourth time periods with the expected cost of EEV_1 = RP = 3930. Indeed, this is the policy obtained by our stochastic model. Then by substituting the random demands for each item at each period, by their expected values, we obtain the average scenario model (EV); which is a deterministic 5-period CLSP. The optimal solution of this 5-period deterministic CLSP is shown in the figure (7) in which the family is replenished in the first and fourth time periods with schedule cost = 3720. Having the optimal solution of EV, by fixing the first period policy for all scenarios in RP, at the optimal solution of EV in this period, we obtain the EEV_2 = 4060, fixing the first and second decisions to EV solution, EEV_3 = 4060, fixing the three first period decisions, EEV_4 = 4765 and finally when fixing the four first time stages decisions EEV_5 = 4990, which is the general value of expected result using the EV in our multi-stage model and corresponds to the EEV introduced for two-stage stochastic problems. These solutions are depicted in figures (9)−(12).

Having these values we can easily calculate the percentage value of our stochastic model at each time t with respect to RP objective value, as the following:

$$VSS_t\% = \frac{100(EEV_t - RP)}{RP}$$

$$VSS_1 = 0\%$$

$$VSS_2 = 3.30\%$$

$$VSS_3 = 3.30\%$$

$$VSS_4 = 21.24\%$$

$$VSS_5 = 26.97\%$$

We can observe that $0 \leq VSS_t \leq VSS_{t+1}$ for each $t = 1, \dots, T-1$. In fact, these sequence of non-negative values, can be interpreted as the "cost of ignoring uncertainty until stage t when making decisions in multi-stage models" (Escudero et al. 2007).

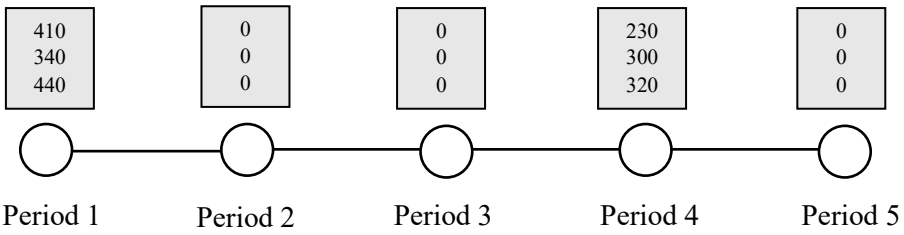


Figure (7): Average scenario solution (EV)

Figure (8): Demand samples for 3-item family

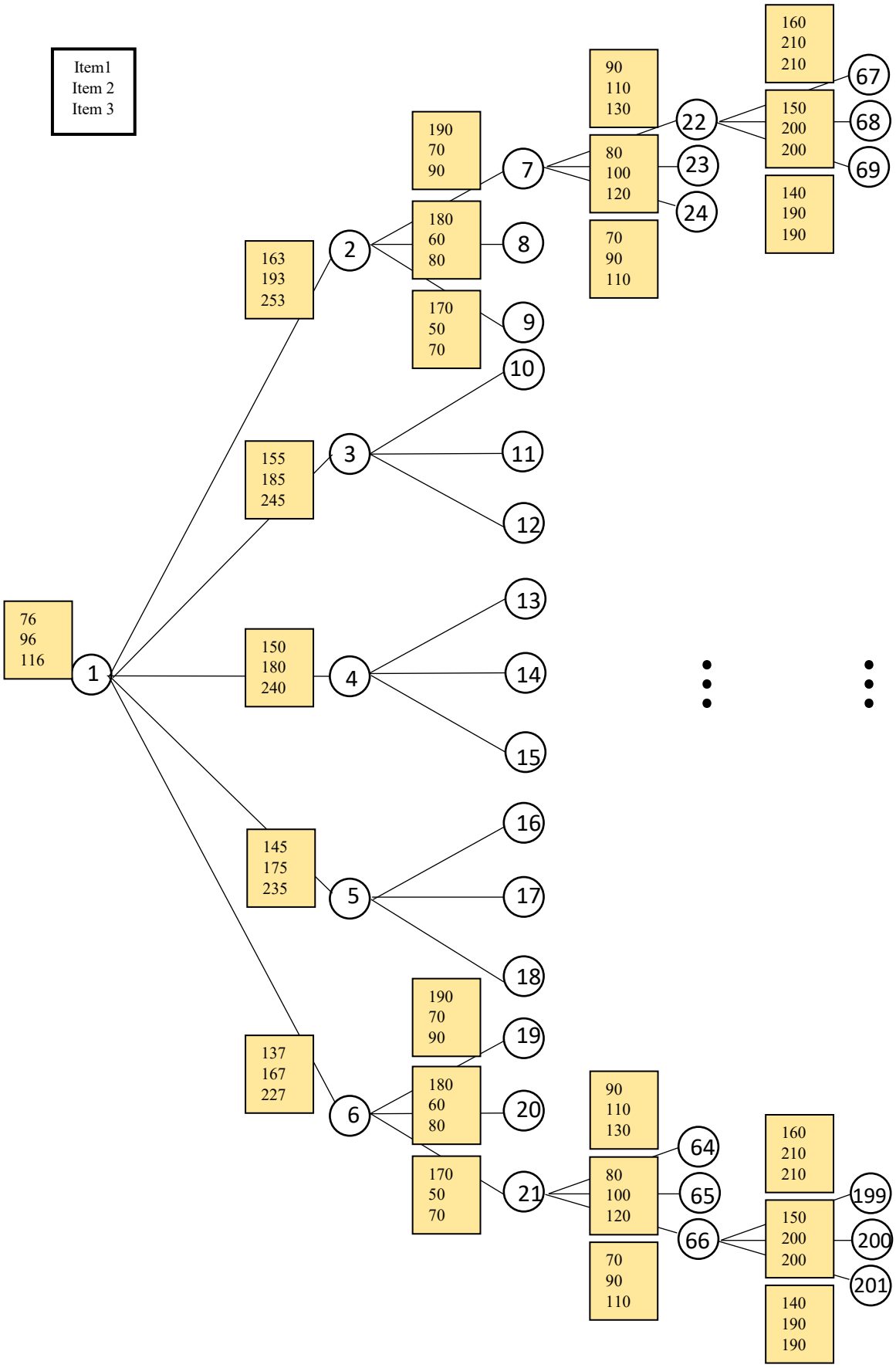


Figure (9): RP solution.

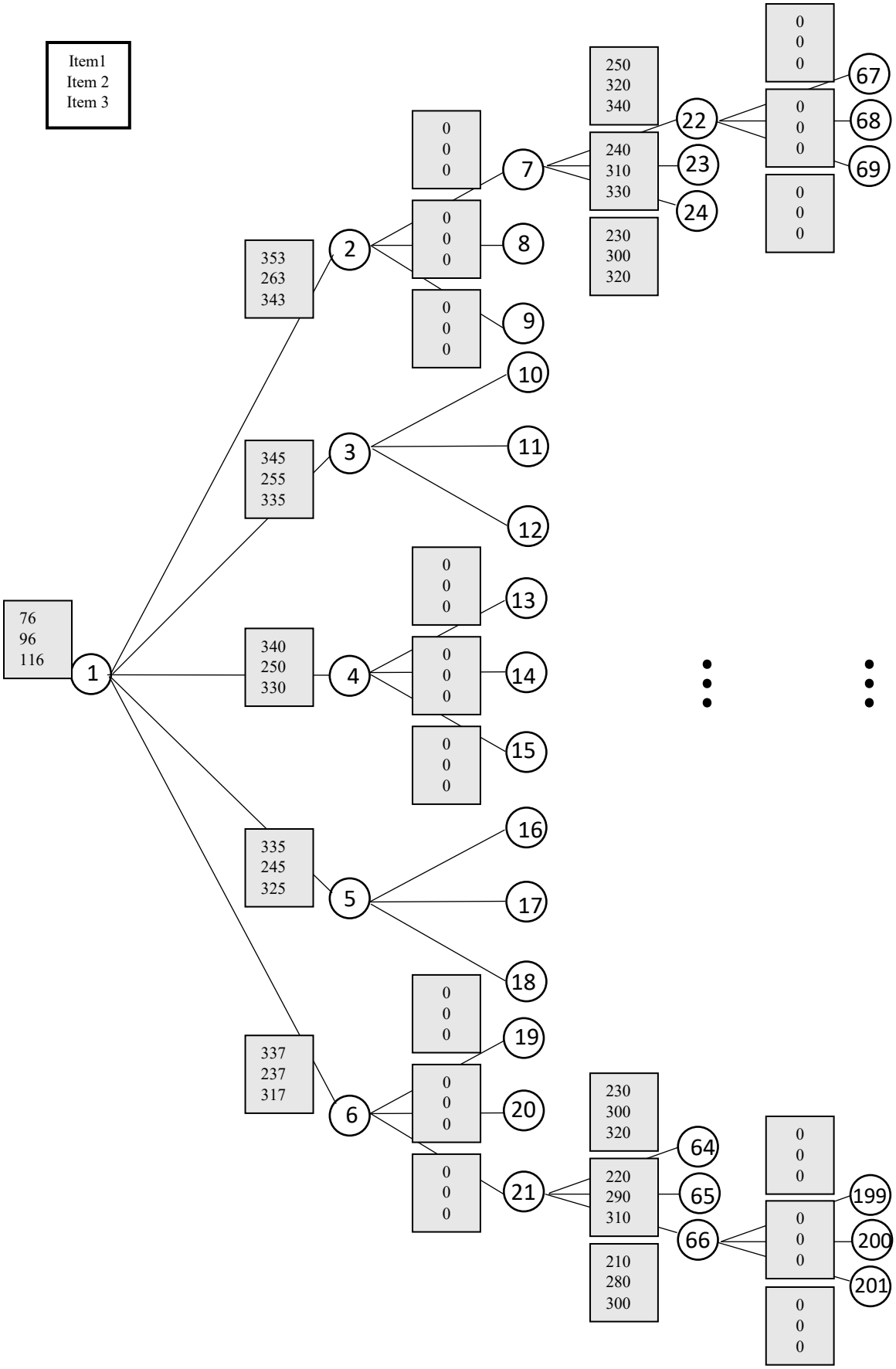


Figure (10): EEV_2 and EEV_3.

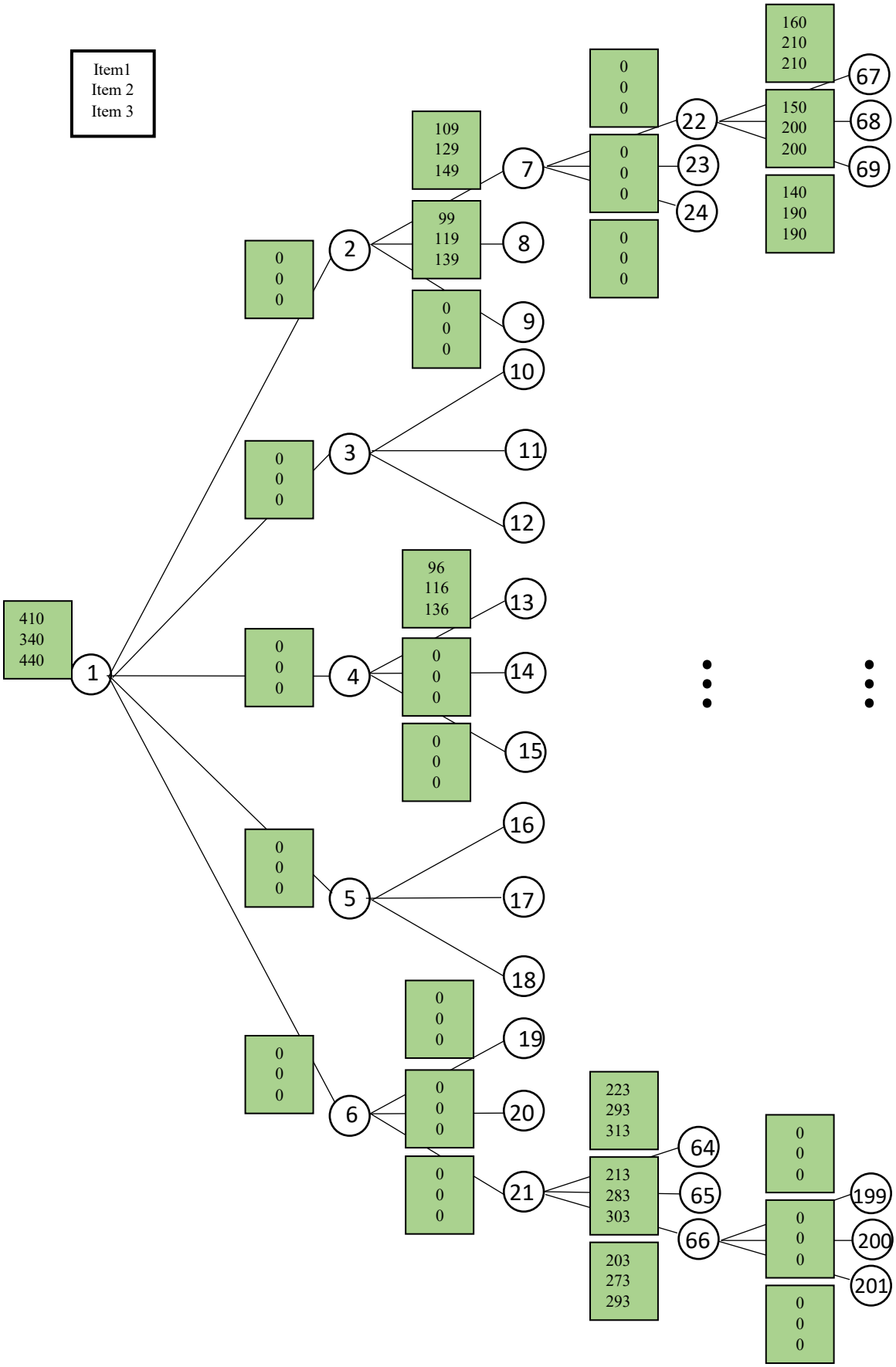


Figure (11): EEV_4.

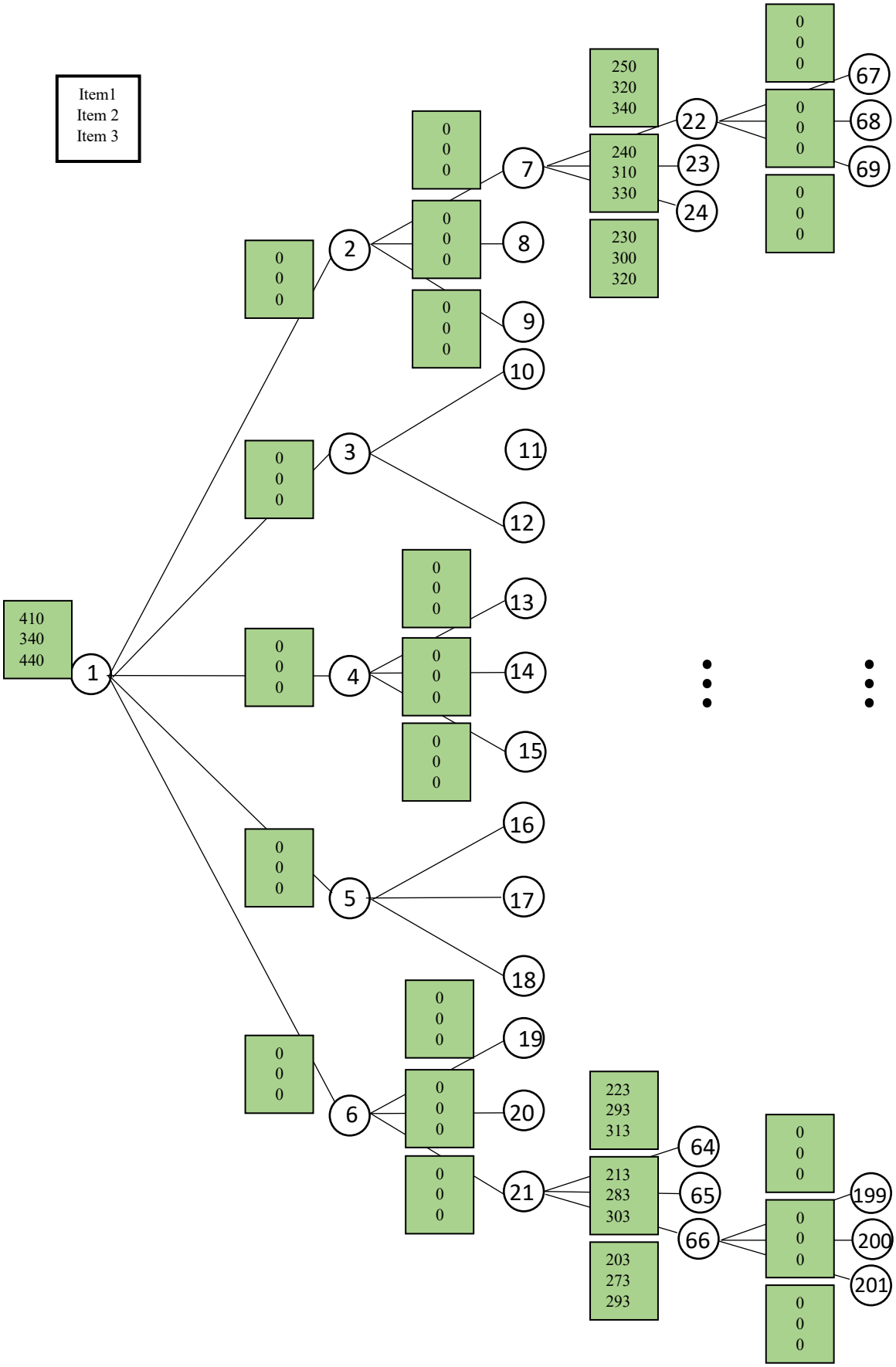
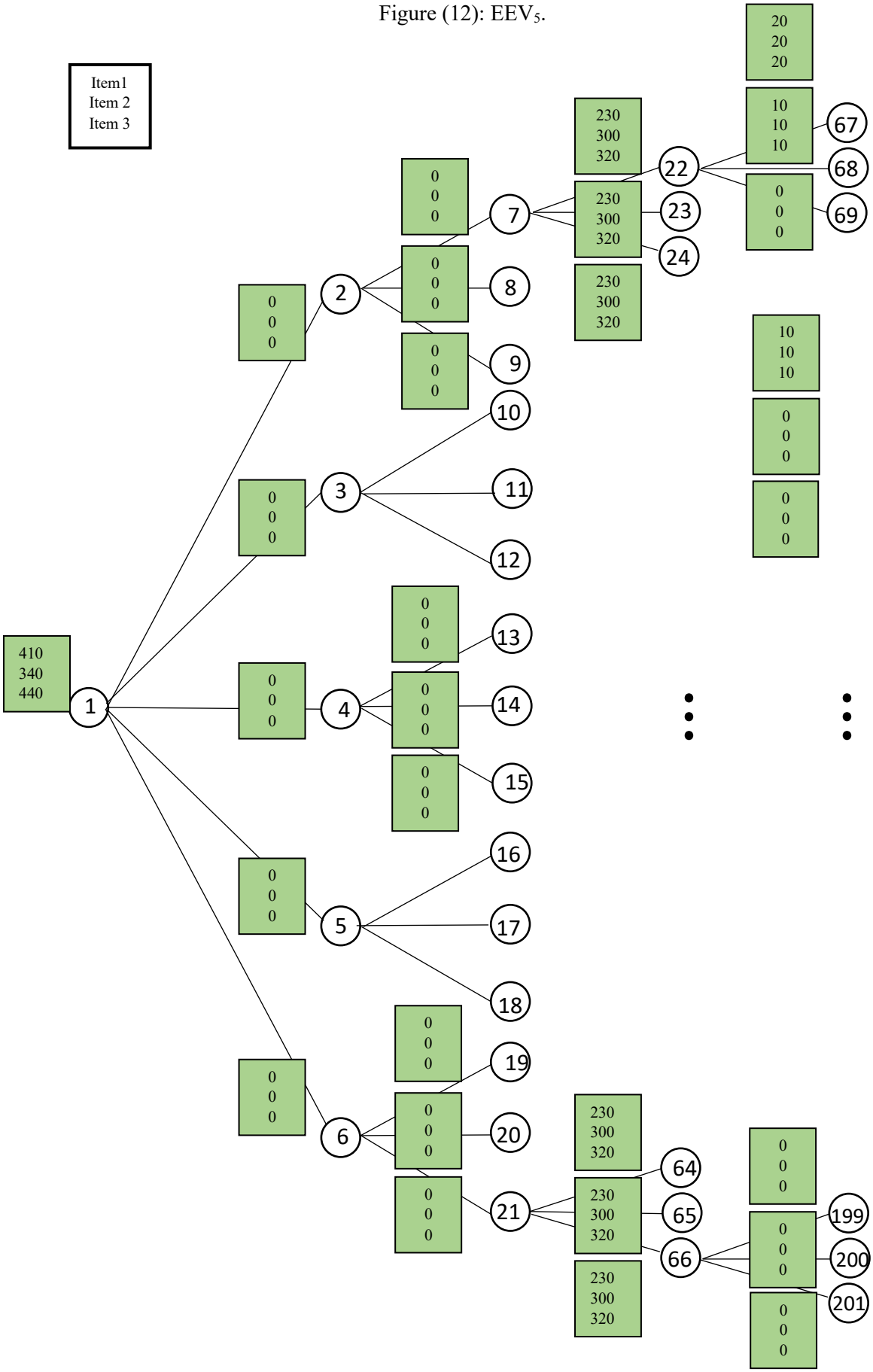


Figure (12): EEV_5.



Conclusions

In the presented work, we have considered the Coordinated Lot-sizing problem (CLSP) or Joint Replenishment Problem (JRP) which consists of a single-family of different item types without any capacity constraint. The problem aims at finding a time-phased replenishment schedule that minimizes the sum of major and minor setup costs, inventory holding and penalty costs.

We firstly considered the deterministic version of the problem and presented the natural formulation for it which resulted in mixed-integer linear programming model. Then we evaluated the performance of Forward-Pass heuristic method in both static and rolling horizon framework. We obtained the optimal solution as a benchmark using a state-of-the-art optimization software *Gurobi*. The results showed that under certain condition the FP outperforms the optimal solution in the rolling horizon environment.

Then by adding the uncertainty to item demands we dealt with the stochastic version of the problem ($CLSP_{ud}$). We modeled the demand uncertainty through a typical scenario tree which resulted in a multi-stage stochastic mixed-integer problem. We firstly tried to solve the deterministic equivalent (DE) of the SP model to optimality by Gurobi and performed several numerical experiments in order to find the largest sized model (in both problem dimension and scenario tree size) which is solvable in a reasonable amount of time by the optimization solver. We also compared the performance of the SP model using different scenario. However, the SP model have been shown to be difficultly solved by Gurobi for large number of scenarios. Therefore we resorted to the decomposition strategy in particular the Progressive Hedging Algorithm which is a scenario-based decomposition strategy and it has been proposed to tackle with large-scale stochastic models. The PHA has been shown to be convergent only in the convex cases. In the presence of discrete decision variables, because of non-convexity of the problem, the PHA can be effectively used as a heuristic method. We applied a PH-based algorithm to our multi-stage $CLSP_{ud}$. Computational experiments showed that algorithm outperforms the Gurobi solution in efficiency.

Finally we analyzed the convenience of our stochastic model with respect to the simpler methods which are based on the expected demand values, by computing a well-known stochastic programming measure VSS, which resulted in a positive value showing the value of our stochastic model.

Acknowledgments

I would like to express my sincere gratitude to Dr. Paolo Brandimarte, not only for his guidance in all steps of doing my thesis but also for his instructive and interesting classes.
My appreciation extends to department of mathematical science in Politecnico di Torino that was the quite right and best place for me to go forward in my mathematical interests.

Bibliography

- Ahmed, S., King, A.J., Parija, G., (2003). A Multi-stage Stochastic Integer Programming Approach for Capacity Expansion under uncertainty. *Journal of Global Optimization*, 26, 3-24.
- Arkin, E., Joneja, D., Roundy, R., (1989). Computational complexity of uncapacitated multi-echelon production planning problems. Operations Research letters, 8, 61-66.
- Baker, K.R., (1977). An experimental study of the effectiveness of rolling schedules in production planning. Journal of Decision Sciences Institute, 1, 19-27.
- Ballou, R.H., (1988). Business Logistics management 4th ed Prentice Hall, Upper Saddle River, NJ.
- Blackburn, J.D., Millen, R.A., (1980). Heuristic lot-sizing performance in a rolling schedule environment. Decision Science 11, 691-701.
- Blackburn, J.D., Millen, R.A., (1982). The impact of a rolling schedule in a multilevel MPR system. Journal of Operations Management 2, 125-135.
- Boctor, F.F., Laporte, G., Renaud, J., (2004). Models and Algorithms for the dynamic-demand joint replenishment problem. International Journal of Production Research, No. 13, 2667-2678.
- Brandimarte, P., Multi-stage capacitated lot-sizing with demand uncertainty. International Journal of Production Research, 2006, 24, No. 15, 2997-3022.
- Brandimarte, P., *Numerical Methods in Finance: A MATLAB-based Introduction*, 2001 (Wiley: New York).
- Dantzig, G.B., (1955). Linear programming under uncertainty. Management Science, 1, 197-206.
- Eisenhut, P.S., (1975). A dynamic lot sizing algorithm with capacity constraints. AIIE Trans 7, 170-176.
- Erenguc, S.S., (1988). Multi-product dynamic lot-sizing model with coordinated replenishments. Naval Research Logistics, 35, 1-22.
- Escudero, L.F., Garín, A., Merino, M., Pérez, G., (2007). The value of the stochastic solution in multistage problems. Top 15, 48-64.
- Federgruen, A., Meissner, J., Tzur, M., (2007). Progressive interval heuristics for multi-item lot-sizing problems. Operations Research 55. 490-502.

- Graves, S.C., (2011). Uncertainty and Production Planning. In Book: Planning production and Inventories in the Extended Enterprise. *International Series in Operations & Management Science* 151, 83-101.
- Haugen, K.K., Løkketangen, A. and Woodruff, D.L., Progressive Hedging as a meta-heuristic applied to stochastic lot-sizing, *European Journal of operational Research*, 2001, 132, 116-122.
- Hoyland, K., Wallace, S.W., (2001). Generating scenario trees for multistage decision problems. Management Science, 47, 295-307.
- Joneja, D, (1990). The joint replenishment problem: New heuristics and worst case performance bounds. *Operations Research*, 38(4), 711-723.
- Kall, P., Wallace, S.W., (1994). Stochastic Programming, John Wiley, Chichester.
- Karimi, B., Fatemi Ghomi, S.M.T, and Wilson, J.M, The Capacitated Lot-sizing Problem: A review of models and algorithms. Omega, 31, 365-378.
- Kao, E.P.C., (1979). A multi-product dynamic lot-size model with individual and joint setup costs. Operations Research 27, 279-289.
- Lambrecht, M.R., Vanderveken, H., (1979). Heuristic procedures for the single operations, multi-item loading problem. AIIE Trans 11, 319-326.
- Law, A.M., (2013). Simulation Modelling and Analysis. *Mc Graw hill Education*.
- Li, Z., Ierapetritou, M., (2007). Process scheduling under uncertainty: Review and challenges. Computers and Chemical Engineering 32, 715-727.
- Maggioni, F., Allevi, E., Bertocchi, M., (2011). Measures of information in multi-stage stochastic programming.
- Manerba, D., Perboli, G., 2017. New Solution Approaches for the Capacitated Supplier Selection problem with total quantity discount policy and activation costs under uncertainty. CIRRELT, 68.
- Manerba, D., Mansini, R., Perboli, G., 2018. The Capacitated Supplier Selection problem with total quantity discount policy and activation costs under uncertainty. International Journal of Production Economics, 198, 119-132.
- Narayanan, A., Robinson, P., (2010). Evaluation of joint replenishment lot-sizing procedures in rolling horizon planning systems. Int. J. Production Economics, 127, 85-94.
- Nezir, A., (2012). Sampling based progressive hedging algorithm for stochastic programming problems. Doctor of Philosophy, Wayne State University.
- Suwondo, E., and Yuliando, H., DYNAMIC LOT-SIZING PROBLEMS: A Review on Model and Efficient Algorithm, Agroindustrial journal, 2012, 1, 34-96.
- Robinson, E.P., Lawrence, F.B., (2004). Coordinated capacitated lot-sizing problem with dynamic demand: a Lagrangian heuristic. Decision Science 35, 25-54.
- Robinson, E.P., Narayanan A., Gao, L.L., (2006). Effective heuristics for the dynamic demand joint replenishment problem. Journal of Operational Research Society 58, 808-815.

- Rockafellar, R.T., Wets, R.J.B., (1991). Scenario and policy aggregation in optimization under uncertainty. Mathematics and Operations research, 16, 119-147.
- Silver, E.A., Meal, H.C., (1973). A heuristic for selecting lot size quantities for the case of a deterministic time varying demand rate and discrete opportunities for replenishment. Production Journal Management J 14, 64-74.
- Silver, E.A., (1979). Coordinated replenishments of items under time varying demand: dynamic programming formulation. Naval research logistics Quarterly 26, 141-151.
- Simpson, N.C., (1999). Multiple level production planning in rolling horizon assembly environments. European journal of Operational Research 114, 15-28.
- Solak, S., (2007). Efficient solution procedures for multistage stochastic formulations of two problem classes. Doctor of Philosophy, Georgia Institute of Technology.
- Wagner, H.M., Whitin, T.M., (1958). Dynamic version of the economic lot size model. Management Science, 5, 89-96.
- Watson, J.P., Woodruff, D.L., (2010). Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. Comput Manag Sci 8, 355-370.