Constraint programming for lot-sizing problems

Grigori German

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Grigori GERMAN

Thèse dirigée par
Jean-Philippe GAYON, Professeur à l'Université Clermont Auvergne
et codirigée par
Hadrien CAMBAZARD, Maître de conférences à l'Institut National Polytechnique de Grenoble

préparée au sein du laboratoire **G-SCOP**
et de l'école doctorale **MSTII**

Programmation par contraintes pour le dimensionnement de lots de production

Constraint programming for lot-sizing problems

Thèse soutenue publiquement le **5/03/2018**,
devant le jury composé de :

M. Jean-Philippe GAYON
Professeur à l'Université Clermont Auvergne, Directeur de thèse
M. Hadrien CAMBAZARD
Maître de conférences à l'Institut National Polytechnique de Grenoble, Co-Encadrant de thèse
M. Christophe LECOUTRE
Professeur à l'Université d'Artois, Président
M. Stéphane DAUZERE-PERES
Professeur à l'Ecole des Mines de Saint-Etienne, Examinateur
Mme Safia KEDAD-SIDHOUM
Maître de conférences à l'Université Paris 6, Rapporteur
M. Christian ARTIGUES
Directeur de recherche CNRS au LAAS, Rapporteur

Merci à Hadrien, Jean-Philippe, Safia, Christian, Christophe, Stéphane, Bernard, Olivier, Vincent, Pierre, Lucie, Matthieu, Lisa, Lucas, Sylvain, Alexandre, Tom, Clément, Nicolas, Hugo, Julien, Florence, mon papa et ma maman.

Contents

List of Figures

List of Tables

Résumé en français

La planification de production est un domaine riche en problèmes complexes de la recherche opérationnelle et de l'optimisation combinatoire. En particulier, les problèmes de dimensionnement de lots (lot-sizing), introduits par [99], ont été largement étudiés. Beaucoup de ces problèmes sont polynômiaux et abordés par programmation dynamique. En présence de certaines extensions, comme des capacités de production variables au cours du temps, ces problèmes peuvent devenir NP-complets. Ces derniers sont souvent traités par programmation linéaire en nombres entiers et de nombreuses formulations linéaires ont été proposées [13, 71]. Plus récemment, des travaux se sont intéressés à la résolution de problèmes de lot-sizing par la programmation par contraintes [52]. Les auteurs introduisent une contrainte globale en planification de production qui considère un ensemble de produits à réaliser avant leurs dates limites de production sur une machine de capacité donnée avec un coût de stockage limité. L'approche traitée dans [52] relève cependant davantage de l'ordonnancement (détermination des dates de début des activités) alors que cette thèse considère un horizon temporel plus tactique (détermination des quantités à produire sur un horizon discrétisé).

L'objectif de cette thèse est d'étudier la pertinence d'un solveur par contraintes pour les problèmes de lot-sizing. Nous introduisons une contrainte globale pour un problème de lot-sizing mono-produit relativement général et le filtrage est réalisé à partir de techniques de filtrage de ressources sur les graphes d'états d'algorithmes de programmation dynamique. Nous définissons aussi une nouvelle technique de filtrage par programmation linéaire qui pourrait permettre d'améliorer le filtrage de la contrainte globale. Il s'agit d'une technique générique très utile lors de la conception de nouvelles contraintes globales en programmation par contraintes.

La thèse est structurée en quatre chapitres. Le chapitre 1 définit les notions nécessaires pour suivre la construction de la contrainte globale et présente un algorithme de filtrage qui s'appuie sur la programmation dynamique.

Le chapitre 2 introduit la contrainte globale LOTSIZING prenant en compte des

capacités et des coûts variables au cours de l'horizon de planification. Nous considérons à la fois des coûts fixes et des coûts unitaires (par unité de produit) de production, des coûts unitaires de stockage, ainsi que des capacités de production et de stockage. Nous pensons que cette contrainte globale est une brique importante pour la modélisation et la résolution de nombreuses extensions difficiles (multi-produits, multi-échelons, etc). Nous posons les bases algorithmiques pour la réalisation du filtrage qui apporte des informations intéressantes sur les domaines des variables de décision (production et stockage) ainsi que des bornes inférieures des coûts pris individuellement (coûts fixes de production, coûts variables de production, coûts de stockage). Des résultats expérimentaux valident l'approche proposée en comparant la résolution de différents modèles avec des modèles de programmation linéaire en nombres entiers.

Le chapitre 3 s'intéresse aux problèmes multi-produits et plus particulièrement au problème de lot-sizing multi-produits avec capacités et coûts fixes partagés. Nous analysons comment des problèmes complexes peuvent être abordés grâce à LOTSIZING. Dans ce même chapitre la contrainte globale est étendue afin de prendre en compte des coûts unitaires de production et de stockage linéaires par morceaux. Le filtrage est lui aussi renforcé par l'utilisation de variables de cardinalités redondantes.

Finalement, le chapitre 4 présente une contribution plus générique de la thèse avec la définition d'un procédé de filtrage par programmation linéaire. Il s'agit d'une méthode générique qui peut se révéler précieuse lors de la conception de contraintes globales. Nous montrons qu'il est possible d'obtenir l'arc cohérence pour n'importe quel ensemble de contraintes qui a une formulation idéale par une unique résolution d'un programme linéaire. Nous adaptons ce résultat pour faire du filtrage partiel dans le cas ou l'ensemble de contraintes considéré n'a pas de restriction. Appliquée à la contrainte LOTSIZING, elle peut être utile afin de propager plus de raisonnements sur les problèmes de flot ou de multi-flot durant la recherche et de fournir l'arc cohérence pour les contraintes de SEQUENCE définies par les variables de cardinalité.

Le problème de lot-sizing mono-produit

Le problème de lot-sizing (appelé (L)) consiste à planifier la production d'un unique type de produit sur un horizon fini de T périodes afin de satisfaire une demande d_t à chaque période t. Le coût de production à une période t est constitué d'un coût unitaire p_t (coût payé par produit) et d'un coût fixe s_t payé si au moins une unité est produite. Le coût de stockage est un coût unitaire h_t payé par produit en stock à la fin de la période t. De plus, la production (respectivement le stockage) est limitée par des capacités minimales et maximales $\underline{\alpha_t}$ et $\overline{\alpha_t}$ (respectivement $\underline{\beta_t}$ et

$\overline{\beta_t}$) à chaque période. L'objectif est de définir un plan de production satisfaisant les demandes à chaque période, respectant les capacités et minimisant le coût global du plan de production. Les notations utilisées pour la représentation des données et pour les variables sont résumées ci-dessous:

Paramètres :

- T : Nombre de périodes.
- $p_t \in \mathbb{R}_+$: Coût unitaire de production à t.
- $h_t \in \mathbb{R}_+$: Coût unitaire de stockage entre les périodes t et $t+1$.
- $s_t \in \mathbb{R}_+$: Coût fixe à t (payé si au moins une unité a été produite à t).
- $d_t \in \mathbb{N}$: Demande à la période t.
- $\underline{\alpha_t}, \overline{\alpha_t} \in \mathbb{N}$: Quantités minimales et maximales de production à la période t.
- $\underline{\beta_t}, \overline{\beta_t} \in \mathbb{N}$: Quantités minimales et maximales de stockage entre les périodes t et $t+1$.
- $I_0 \in \mathbb{N}$: Niveau de stock initial.

Variables :

- $X_t \in \mathbb{N}$: Quantité produite à t.
- $I_t \in \mathbb{N}$: Quantité stockée entre t et $t+1$.
- $Y_t \in \{0,1\}$: Vaut 1 si au moins une unité est produite à t, 0 sinon.
- $C \in \mathbb{R}_+$: Coût global du problème.
- $Cp \in \mathbb{R}_+$: Somme des coûts unitaires de production.
- $Ch \in \mathbb{R}_+$: Somme des coûts unitaires de stockage.
- $Cs \in \mathbb{R}_+$: Somme des coûts fixes de production.

La figure 1 présente le problème comme un graphe avec les variables et les paramètres sur chaque arc. Pour chaque période, les arcs entrants correspondent à la production (arcs verticaux) et au stockage depuis la période précédente (arcs horizontaux) potentiels. Les arcs sortants correspondent à la demande (arcs verticaux) et au stockage potentiel en fin de période (arcs horizontaux).

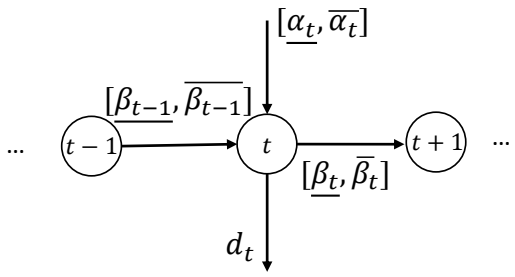

Figure 1: Graphe de flot du lot-sizing mono-produit

Un modèle mathématique pour le problème (L) peut s'écrire :

$$\textit{minimize } C = Cp + Ch + Cs \tag{1}$$
$$I_{t-1} + X_t = d_t + I_t \qquad \forall\, t = 1 \dots T \tag{2}$$
$$X_t \leq \overline{\alpha_t} Y_t \qquad \forall\, t = 1 \dots T \tag{3}$$
$$Cp = \sum_{t=1}^{T} p_t X_t \tag{4}$$
$$Ch = \sum_{t=1}^{T} h_t I_t \tag{5}$$
$$Cs = \sum_{t=1}^{T} s_t Y_t \tag{6}$$
$$X_t \in \{\underline{\alpha_t}, \dots, \overline{\alpha_t}\} \qquad \forall\, t = 1 \dots T \tag{7}$$
$$I_t \in \{\underline{\beta_t}, \dots, \overline{\beta_t}\} \qquad \forall\, t = 1 \dots T \tag{8}$$
$$Y_t \in \{0, 1\} \qquad \forall\, t = 1 \dots T \tag{9}$$

où (2) sont les contraintes d'équilibre du flot pour chaque période et (3) sont les contraintes de coût fixe fixant Y_t à 1 si une production est effectivement faite à la période t ; notons également que le paiement d'un coût fixe à t (*i.e* $Y_t = 1$) n'implique pas forcément une production à t (*i.e* $X_t > 0$). Finalement, (4), (5) et (6) sont les expressions des différents coûts.

Une contrainte globale pour le lot-sizing

Nous introduisons formellement la contrainte globale LOTSIZING. Elle porte sur le vecteur de variables $X = \langle X_1, \dots, X_T \rangle$, $I = \langle I_1, \dots, I_T \rangle$, $Y = \langle Y_1, \dots, Y_T \rangle$ et les quatre variables de coût suivantes Cp, Ch, Cs, C de (L). Les données sont définies par $data = \{(p_t, h_t, s_t, d_t, \underline{\alpha_t}, \overline{\alpha_t}, \underline{\beta_t}, \overline{\beta_t}) \mid t \in [\![1, T]\!]\}$.

Définition LOTSIZING$(X, I, Y, Cp, Ch, Cs, C, data)$ *a une solution si et seulement si il existe un plan de production, solution de* (L) *qui satisfait:*

$$Cp \leq \overline{Cp}$$
$$Ch \leq \overline{Ch}$$
$$Cs \leq \overline{Cs}$$
$$C \leq \overline{C}$$

Cette contrainte globale est NP-difficile car le problème de lot-sizing mono-produit considéré est NP-difficile. Nous avons analysé la complexité d'obtenir différents niveaux de cohérences pour LOTSIZING. La borne cohérence de LotSizing peut être atteinte en temps pseudo-polynomial. Si l'on considère seulement les contraintes de réalisabilité, c'est-à-dire sans les bornes supérieures des différents coûts, la borne cohérence se fait en temps polynomial. Finalement, sur ce sous-problème de réalisabilité, la borne cohérence est équivalente à la range cohérence.

En définitive, trouver une solution réalisable est facile, ce sont les coûts qui rendent ce problème difficile. C'est pour cela que nous avons orienté les algorithmes de filtrages développés dans la thèse vers la programmation dynamique. Cette dernière peut se révéler coûteuse mais fournit des raisonnements forts. La contrainte globale LOTSIZING comprend plusieurs éléments.

Règle de dominance. Une fois que les variables de coût fixe sont instanciées, le problème devient polynomial car il s'agit d'un simple problème de flot. Nous avons utilisé cette propriété pour éviter de brancher sur les variables de quantités X et I. Ainsi le branchement ne se fait que sur les variables binaires Y et le problème de flot est résolu une fois que tous les Y sont instanciés afin de définir les quantités optimales. Cette propriété de dominance ne tient pas toujours suivant les contraintes additionnelles posées par l'utilisateur.

Transformation du problème. Afin de simplifier les algorithmes de filtrage et d'accélérer les algorithmes de programmation dynamique, une transformation simple est appliquée pour obtenir un problème équivalent sans bornes inférieures.

Calcul de bornes inférieures. Des bornes inférieures des différents coûts sont calculées en résolvant des problèmes de flot pour Cp, Ch et Cs et par programmation dynamique pour Cs et C.

Filtrage des variables. Finalement, le filtrage des variables s'opère grâce à un algorithme de filtrage qui s'appuie sur l'utilisation de ressources dans le graphe d'états du programme dynamique. Le programme dynamique peut être représenté

par un graphe où la solution optimale est trouvée en résolvant un problème de plus court chemin.

La résolution du programme dynamique est un processus lourd et peut ne pas passer à l'échelle. Dans ce cas, nous utilisons un procédé classique en lot-sizing, la décomposition en sous-problèmes. Les sous-problèmes de taille "raisonnable" peuvent alors être résolus par programmation dynamique, les autres sont résolus sous-optimalement par flot. Une borne inférieure et le filtrage sont alors dérivés de la résolution d'un problème d'ordonnancement d'intervalles pondérés (Weigthed Interval Scheduling).

Les résultats numériques montrent d'une part que le modèle décomposé de programmation par contraintes ne peut pas résoudre le problème mono-produit alors que le modèle s'appuyant sur LOTSIZING donne des solutions en temps raisonnable. Les expérimentations avec des contraintes additionnelles combinatoires avec lesquelles le modèle PLNE a plus de difficulté, montrent que la programmation par contraintes peut se révéler plus efficace pour résoudre certains problèmes de lot-sizing. Nous avons aussi validé numériquement l'adaptation des algorithmes de filtrage sur des grosses instances pour lesquelles le programme dynamique ne passe pas à l'échelle.

Vers des problèmes plus complexes

Nous étudions une extension naturelle du problème de lot-sizing mono-produit: le multi-produits. Il s'agit alors d'optimiser globalement la production de plusieurs types de produits qui utilisent les mêmes ressources : les coûts fixes et les capacités peuvent être partagés par tous les produits. Dans ce cas, la propriété de dominance qui nous permet de brancher seulement sur les variables Y n'est plus valide. Nous montrons qu'une autre forme de dominance existe et qu'il est tout de même possible d'éviter de brancher sur les variables de quantités. Il faut considérer le problème global une fois que toutes les variables binaires sont fixées et en général, soit le problème est polynomial soit il est rapidement résolu en pratique par PLNE car les structures des problèmes de lot-sizing sont très proches de problèmes de flot.

Nous étendons LOTSIZING en permettant la modélisation des coûts unitaires de production et de stockage par des fonctions linéaires par morceaux convexes. Il s'agit d'une caractéristique qui simplifie énormément la modélisation de certains problèmes de lot-sizing et ne dégrade pas la complexité du problème.

Nous améliorons la puissance du filtrage en ajoutant des variables de cardinalités redondantes. Le programme dynamique prenant en compte ces variables est cependant trop coûteux. Le gain pourrait être significatif en développant un algorithme de filtrage plus simple.

Nous explorons les possibilités de modélisation grâce à des contraintes

LOTSIZING redondantes. Celles ci permettent de capturer la globalité du problème, de raisonner plus efficacement et de calculer de meilleures bornes inférieures. Les expérimentations montrent que les modèles de programmation par contraintes ne sont pas compétitives par rapport à la PLNE pour ces problèmes classiques purs sans contraintes additionnelles plus combinatoires.

Un nouvel algorithme de filtrage par programmation linéaire

La programmation par contraintes utilise parfois la programmation linéaire pour filtrer des variables. Par exemple, le filtrage par coûts réduits vient de la recherche opérationnelle et des méthodes d'optimisation linéaires et permet de réduire les domaines des variables grâce aux coûts réduits obtenus après résolution d'un programme linéaire. Une autre technique de filtrage consiste à fixer une variable à une valeur puis de résoudre un programme linéaire afin de vérifier si cette valeur est cohérente. Cette dernière est assez coûteuse car elle nécessite autant d'appels à un solveur de programmation linéaire que de couples variable/valeur. Nous avons développé un nouvel algorithme de filtrage générique qui nécessite une unique résolution de programme linéaire. Dans le cas où la formulation est idéale, c'est-à-dire que l'ensemble des contraintes a la propriété d'intégralité, le filtrage fait l'arc cohérence. C'est particulièrement le cas dans le domaine de la planification de production qui comporte souvent des sous-problèmes de flots ou multi-flots.

Le principe du théorème général est simple. Prenons un ensemble de contraintes qui a une formulation idéale avec N variables et M contraintes. L'objectif est de rechercher un point intérieur x du polytope défini par ces contraintes. Ce point intérieur permet de décoller au maximum les variables de leurs bornes. S'il arrive qu'une variable x_t de x soit toujours collée à sa borne, cela signifie que toutes les solutions entières de l'ensemble de contraintes vérifient que x_t est collée à cette même borne. Si on prend une formulation idéale avec une variable $0/1$ pour chaque couple variable/valeur, on obtient l'arc cohérence du réseau de contraintes.

Nous montrons qu'il est possible de trouver un point intérieur grâce à un unique programme linéaire avec $N + M$ contraintes et $2N$ variables. La complexité traditionnellement observée de la résolution d'un programme linéaire étant $O(M \log N)$, notre technique de filtrage a une complexité $O((N + M) \log N)$. La complexité de la technique de filtrage mentionnée plus haut, avec autant d'appel au solveur linéaire que de couples variable/valeur, a une complexité $O(NM \log N)$.

Les résultats numériques valident la procédure sur les contraintes globales ALLDIFFERENT et la famille des contraintes de SEQUENCE avec une application au problème d'ordonnancement de voitures (car sequencing).

Ce résultat est adapté au cas où la formulation n'est pas idéale et nous montrons

que le filtrage est toujours valide mais incomplet, c'est-à-dire que les valeurs qui sont filtrées sont bien incohérentes mais que toutes les valeurs incohérentes ne sont pas filtrées.

Chapter 1

Introduction

Lot-sizing problems are crucial in the production planning field and consist in determining quantities to produce each time period to optimize several costs – among which one can find production, inventory or setup costs. Classical approaches for academic pure lot-sizing problems, such as mixed integer linear programming or dynamic programming, are generic and efficient. However, real world problems with constraints inherent to the industry are often dealt with non generic heuristic or complex approaches. Constraint programming is a generic optimization framework that allows the modeling of a wide range of problems. It performs very well on problems with strong combinatorial structures, where a feasible solution is usually hard to find and where the cost function can give strong inferences, *i.e.* the constraints modeling the cost can provide strong reasoning mechanisms. Scheduling problems are one of the most successful applications of constraint programming with problems such as nurse rostering, task sequencing and job-shop scheduling to mention a few.

This thesis is a first step to tackle lot-sizing problems using constraint programming. We use and combine some of the huge amount of knowledge on lot-sizing problems since 1958 to develop a constraint programming framework. We have the intention to help solve intricate, full of shady side constraints lot-sizing problems that mixed integer linear programming might have difficulties to handle. In this thesis, we build a global constraint that embeds a most generic single-item single-level lot-sizing problem. The basic constraint programming decomposed model is inefficient as there are no global reasoning on the variables. We claim that this new global constraint is a useful modeling brick that is a first step to model and solve complex lot-sizing problems. The propagation and filtering algorithms are widely based on dynamic programming, linear programming and network flow well-known techniques.

In order to evaluate the feasibility of building a CP solver for production plan-

ning problems, the thesis is divided into four chapters that gradually investigate the construction of a global constraint for lot-sizing.

The goal of this introduction is to provide the reader with some context around lot-sizing problems and insights on how to solve them using generic and classical optimization methods. We present an overview of how lot-sizing fits into the production planning field, then the main optimization method used in the thesis, namely constraint programming. We also investigate the power of global constraints in constraint programming and the advantages of building NP-hard global constraints. Finally, we detail a resource-based filtering algorithm derived from the use of dynamic programming algorithms and show its application on an example. It is a key filtering algorithm that is applied several times in this work.

Chapter 2 defines the LOTSIZING global constraint as the conjunction of the constraints of the single-item single-level lot-sizing problem. We study the complexity of achieving several consistency levels and describe the propagation and filtering mechanisms that are mainly based on dynamic programming. We show a way to tackle scalability problems using a natural and classical decomposition into subproblems. Several problems with side constraints are then solved and compared to other classical models to assess the power of the global constraint.

Chapter 3 is dedicated to the multi-item version. This important extension of the single-item lot-sizing problem helps us improve the filtering of the global constraint and we show how to use it on more complex lot-sizing problems. The improvements use redundant cardinality constraints and extend LOTSIZING as we model the unitary costs as step functions.

Finally, in a more general constraint programming framework, chapter 4 presents a new generic filtering algorithm based on linear programming. This is an improvement of a naive filtering algorithm that is commonly used in early stages of global constraints design. We show that arc-consistency can be achieved with a single call to a linear programming solver if the set of constraints has the integrality property. We show that the filtering remains correct although incomplete when there is no restriction on the set of constraints.

1.1 Production planning and lot-sizing

Production planning consists in planning the use of resources to achieve defined production goals over a time horizon [88]. These goals can be to determine either the resources that are needed, the production rate, the capacities, the schedule of the workers, etc. Most of the problems require a lot of data such as costs and demands that are usually estimated or forecast. We do not consider here how they are obtained. Production planning is divided in three levels depending on the time hori-

zon considered. Long-term planning is the **strategic level** of production planning and is generally defined over a few years depending on the product and market. Its goal is to define the infrastructures (warehouses, factories, products, etc) and activities needed to achieve certain goals (manufacture a product, etc). The time unit of the strategic level is usually the month up to the year. Mid-term planning is the **tactical level** and is defined over a few months. The infrastructures being already fixed by the strategic level, what remains is to define the machines, the quantities to produce at each period, the production rate and build different production plans. These production plans allow the decision makers to evaluate alternative solutions, to define capacity and material requirement and to take informed decisions. The time unit can be the day or the week up to the month. Short-term planning is the **operational level** – scheduling constitutes a great part of it – where infrastructures, machines, activities/tasks and products are already fixed and the goal is to exploit them at best. The operational level produces schedules for the machines day by day: for each day, it finds the best feasible sequence of activities/tasks to optimize the inventory, the idle times, the setup times, the production costs, etc. The time unit is the minute up to the hour.

Many pure academic problems can be considered to model each level of production planning. The thesis focuses on problems from the tactical level and more specifically lot-sizing problems. Note that all the problems studied in the thesis are deterministic problems as opposed to stochastic problems where some data or decisions can depend on probabilistic laws.

Lot-sizing problems are typically problems from the tactical level and can be declined in many different forms. A **lot** is a specified quantity of certain products that is destined to be produced or sold. The main question asked is how to decide lot sizes? Given a discrete time horizon – represented by a series of T periods – , the goal is to decide the quantities to produce at each period while minimizing a combination of different costs. The optimization is usually a trade-off between inventory, production and setup costs. Lot-sizing models come from stock optimization with the classical economic order quantity (EOQ) model [47, 100]. The introduction of [64] gives a comprehensive review of litterature on this problem and on the evolution of inventory management problems in general. The core problem was defined in 1958 by Wagner and Within: the uncapacitated single-item lot-sizing problem [99]. Since, many extensions of the original problem have been studied and a non exhaustive list is presented in table 1.1 [71]. Lot-sizing problems and the state of the art algorithms have been discussed in several surveys over the years [26, 28, 34, 44, 54].

The time decomposition, or the definition of the periods, in lot-sizing problems depends on the product and industry at hand. The size of the buckets – the time discretization – allows lot-sizing problems to be integrated to a certain extent with the operational level of production planning. The main goal of lot-sizing is to ex-

Extension	Description
Backlog	Demands can be satisfied later but you must pay for the late units
Start ups	A start up cost is paid at the start of a sequence of setup costs
Varying capacities	The capacities of the resources can be different at each period
Sales	More units than the demand can be sold every period
Multi-level	The problem is divided in layers (levels) where the demands of a layer is usually the production of the next layer. The resources form a network
Multi-item	Different types of products flow in the network with constraints that link them: they are produced on the same machine for instance
Piecewise/concave/convex costs	Different cost functions can be used
Minimum length setup sequences	Each time a production is activated, at least a certain number of production periods must follow
Lost sales	Not satisfying the whole demand is possible but to a certain cost
Production time windows	The demands must be satisfied from a production in a certain time window (It can be used to model the perishability of products)
Lower bounds	If there is a production, a minimum amount must be produced
Online	Demands are not totally known beforehand and are revealed incrementally

Table 1.1: Common extensions to the single-item lot-sizing problem

ploit batch sizing flexibility before the activities are fixed as in scheduling. Discrete or continuous-time horizons can be considered when planning. Long horizons are more suited for aggregate planning problems with large buckets whereas small buckets or continuous-time models are fit for very short-term planning problems. Lot-sizing usually considers a long, discrete time horizon and does not examine the details of the production but rather deals with the products in lots, and thus can be seen as an abstraction of scheduling. They have sometimes been integrated with problems from the operational level as scheduling problems. In scheduling, the tasks are often derived from the quantities decided by lot-sizing problems. Small bucket lot-sizing models integrate scheduling as the time decomposition brings the problems closer to the operational models. Two main examples are the discrete lot-sizing and scheduling problem [40] and the job-shop lot-sizing and scheduling problem [59].

In conclusion, lot-sizing is an adaptable and scalable modeling tool for many production planning problems. Besides very simple lot-sizing models, most of them

are NP-hard. The main problem studied in this thesis is a most generic version of the single-item single-level lot-sizing problem and will be presented in the next chapter. It is an NP-Hard problem (knapsack 0/1 is a special case [71]). Some works have been done with constraint programming [52, 90], but to our knowledge, the results on lot-sizing for the last 60 years were not exploited to perform filtering.

1.2 Constraint Programming

> "Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."
>
> Eugene C. Freuder, Constraints, April 1997

Constraint Programming (CP) is a generic exact optimization technique that appeared in the early 1970s and comes mostly from the Artificial Intelligence field. It is a declarative and model-based method that is widely used to solve mathematical and combinatorial problems such as scheduling problems. The theoretical basis of CP solvers are the use of logical inferences to search for feasible solutions: the constraints reason and make deductions on the domains of the variables to reduce the search space. The goal is to find feasible assignments of values to variables in a problem where the relations between variables are modeled as arbitrary constraints. As in linear programming, the word "programming" does not refer here to programming in any computer language but rather to an earlier definition of the word, a plan or a notice on the set of actions to take to achieve a goal.

1.2.1 Constraint Satisfaction Problem and definitions

A Constraint Satisfaction Problems (CSP) is the problem to find an assignment of values to the decision variables that satisfies all given constraints [50]. A CSP consists of a set of **variables**, with a finite **domain** of values for each variable, and a set of **constraints** on these variables. We denote by $D(V_i) \in \mathbb{Z}$ the domain of variable V_i and by $\overline{V_i}$ (resp. $\underline{V_i}$) the maximum (resp. minimum) value in $D(V_i)$. CP relies on the propagation of the constraints and the filtering of the variables during the search to efficiently prune the search tree and find feasible solutions. These notions are specified just after, the main idea being that in a node of the search tree, the constraints reason in turn on their variables to remove inconsistent values and find contradictions. Objective functions are naturally tackled by CP when only improving feasible solutions are allowed. CP, just as mixed integer linear programming,

uses an intuitive and expressive modeling language. One advantage of CP is that any type of constraints can be modeled, they are not necessarily linear.

In order to reduce the search space and remove only unfeasible solutions, CP relies on the notion of **consistency**. Consistency is a property of the domains of the variables that implies they do not contain a contradiction to some extent. Enforcing consistency is a means to reduce the search space by removing inconsistent values from the domains of the variables, *i.e.* values that do not belong to a feasible solution. Different levels of consistency can be achieved when removing all or just a part of inconsistent values. Let us formally define some notions and three levels of consistency. Let c be a constraint on variables $\langle V_1, \ldots, V_n \rangle$.

Definition 1 *A **support** for c is a tuple of values $\langle v_1, \ldots, v_n \rangle$ which satisfies c and such that $v_i \in D(V_i)$ for each variable V_i.*

Definition 2 *A **bound support** is a tuple of values $\langle v_1, \ldots, v_n \rangle$ which satisfies c and such that $\underline{V_i} \leq v_i \leq \overline{V_i}$ and $v_i \in \mathbb{Z}$ for each V_i.*

Note that a value v_i that is in a bound support does not necessarily belong to the domain of the corresponding variable V_i since domains can have holes.

Definition 3 *A variable V_i is **arc consistent** (AC) for constraint c if each value of $D(V_i)$ belongs to a support for c.*

Definition 4 *A variable V_i is **bound consistent** (BC) for constraint c if $\underline{V_i}$ and $\overline{V_i}$ belong to a bound support for c.*

Definition 5 *A variable V_i is **range consistent** (RC) for constraint c if each value of $D(V_i)$ belongs to a bound support for c.*

Definition 6 *A constraint c is AC (resp. BC, RC) if all its variables are AC (resp. BC, RC).*

Definition 7 *A CSP is AC (resp. BC, RC) if each constraint is AC (resp. BC, RC).*

The following example illustrates the three notions of AC, BC and RC. Consider the following linear constraint over two integer variables x and y:

$$2x = y, \qquad x \in \{1, 2, 4\} \text{ and } y \in \{4, 5, 6, 7, 8\}$$

The three levels of consistency are applied to the example and showed in table 1.2. The **bound consistent** domains are $\{2, 4\}$ and $\{4, 5, 6, 7, 8\}$ since we just check if the bounds belong to a bound support: only 1 can be removed from the domain of x as $\langle 1, 2 \rangle$ is not a bound support. For instance, the bound support for $y = 8$ is $\langle 4, 8 \rangle$. The **range consistent** domains are $\{2, 4\}$ and $\{4, 6, 8\}$ since values 5 and 7 do not belong to a bound support. Indeed $\langle 2.5, 5 \rangle$ and $\langle 3.5, 7 \rangle$ are not bound supports as the values in a bound support must be in $\mathbb{Z}$. The value 6 for y is range consistent since $\langle 3, 6 \rangle$ is a bound support. Finally the **arc consistent** domains are $\{2, 4\}$ and $\{4, 8\}$ as we remove all inconsistent values.

Consistency level	$D(x)$	$D(y)$
Initial domains	$\{1,2,4\}$	$\{4,5,6,7,8\}$
BC	$\{2,4\}$	$\{4,5,6,7,8\}$
RC	$\{2,4\}$	$\{4,6,8\}$
AC	$\{2,4\}$	$\{4,8\}$

Table 1.2: Three consistency levels

1.2.2 The resolution

The search in CP is a tree search where at each node a certain level of consistency is enforced before the solver takes a decision which is an assignment of a variable to one of the values of its domain. Each constraints c is associated to a filtering algorithm that may remove some values that are inconsistent with c. At each node of the search tree, the filtering algorithms of the constraints are called in turn until a fixed point is reached, i.e. the domains of the variables are no more modified. In short, all the constraints are **propagated** one by one (the reasoning mechanisms of each constraint is activated) and **filter** their variables (they remove values that are not consistent with the current states of the variables). If a contradiction is found (a domain is emptied, a constraint cannot be satisfied, etc) there is a backtrack. CP solvers are not entirely black box solvers since branching is left to the user although a lot of effort has been made in recent years to design generic heuristics [25, 46, 61, 66]. A great variety of branching strategies can be implemented to suit the user's problem and they usually tremendously impact the resolution.

1.2.3 Global constraints and complexity

Let us now define the main focus of this thesis and one of the main strengths of Constraint Programming, namely global constraints [17, 77, 79, 81]. A global constraint encapsulates a conjunction of constraints that propagate together in order to filter more efficiently. *"Global constraints specify patterns that reoccur in many problems. There are, however, only a limited number of common constraints which repeatedly occur in problems. One strategy for developing new global constraints is to identify conjunctions of constraints that often occur together, and developing constraint propagation algorithm for their combination"* [21]. The – rather intuitive – idea is to not let each constraint reason on its own but to consider certain groups of constraint as a whole. Moreover global constraints try to regroup constraints for which there is a propagation algorithm that filters the variables efficiently and that form a logical, specialized and useful pattern of constraints to model many problems. In addition, global constraints are a very interesting modeling tool as well since they allow the user to easier describe its problems and in an aggregated way. Indeed, it is more practical

to set one global constraint than to set each constraint independently.

The global constraint ALLDIFFERENT$(X_1, \ldots, X_n)$ for instance, states that all the variables X_1 to X_n must take different values and uses matching theory ([77]) to filter more than the decomposition of all the combination of *difference* constraints – the set of constraints where the variables are pairwise different. ALLDIFFERENT(X, Y, Z) defines the same feasible region as $\{X \neq Y, X \neq Z, Z \neq Y\}$, yet the global constraint propagates stronger reasoning. Take the following example:

$$X \in \{1, 2, 3\}, Y \in \{1, 2\}, \text{ and } Z \in \{1, 2\}$$

When propagating the difference constraints one at a time, no value can be removed: they are locally consistent. However ALLDIFFERENT(X, Y, Z) can see the three constraints together and remove the values 1 and 2 from the domain of X.

A lot of work has been done on global constraints to represent a great variety of conjunction of constraints. There exists a global constraint catalog that references some 400 global constraints at that date [16]. One of the main characterizations of global constraints is the difficulty of the underlying problem defined by the set of constraints. Some global constraints are polynomial (ALLDIFFERENT [60], ELEMENT [94], COUNT [30], etc) while others are NP-hard (N-VALUE [20, 69], CUMULATIVE [6], etc). More details on the general complexity of global constraints are given in [21]. The main advantage of building polynomial global constraints is that AC is fast to achieve. In practice, the user tries to solve a problem, let us call it P. Global constraints allow to find structural properties of P and cluster the constraints. The user finds a sub-problem Q of P that can be modeled with a global constraint and thus improves the filtering of Q instead of having the decomposed set of constraints of Q propagating one by one. If Q is NP-Hard, modeling it with polynomial constraints may weaken a lot its propagation and filtering. When creating NP-hard global constraints, the modeling bricks are often more general and help better capture the nature of the combinatorial difficulty of the problem. The drawback being that it is usually very costly to achieve AC, even in practice and that the user may have to settle for a lesser level of filtering.

1.3 Filtering via dynamic programming

We present now Dynamic Programming (DP) and how we use it in this thesis in combination with CP to filter the decision variables of a problem. DP is a well-known and widely used exact optimization method [19,33,35]. In the thesis, we are going to use and present only deterministic DP; insights and details about stochastic DP can be found in [83]. This technique is based on the decomposition of the

problem into stages. The problem is decomposed into sub-problems and the optimal solution of a sub-problem is used as input for the next sub-problem. The sub-problems differ depending on the problem at hand. Their optimization is therefore not detailed by the generic definition of dynamic programming and depends on the problem. For the DP algorithms used in this thesis, it usually boils down to the computation of a minimum or a maximum between several values. Three elements are needed to define a DP algorithm over a given problem: an initial condition, a recursion formula and an objective function. The space complexity is usually given by all the combinations of the state variables.

1.3.1 The example of the Knapsack problem

A classical example is the well-known DP algorithm for the Knapsack problem. Given a bag of fixed capacity C_{max} and N items with sizes (w_i) and values (v_i), the Knapsack problem [32, 65] consists in finding the set of items to fit in the bag that maximizes the sum of their values. The classical integer programming model writes:

$$maximize \sum_{i=1}^{N} v_i x_i \tag{1.1}$$

$$\sum_{n=1}^{N} w_i x_i \leq C_{max} \tag{1.2}$$

$$x_i \in \{0,1\} \qquad \forall\, i = 1 \ldots N \tag{1.3}$$

where the binary variable x_i equals to 1 is item i is taken in the bag and 0 otherwise. The objective function (1.1) tends to maximize the sum of values of the items taken in the bag and constraint (1.2) models the capacity. Table 1.3 gives a small example with $N = 4$ items where the bag capacity is $C_{max} = 3$.

i	w_i	v_i
1	1	3
2	2	6
3	1	1
4	1	3

Table 1.3: A small example of the Knapsack problem with four items

We call DPKnap the dynamic programming algorithm that solves the Knapsack problem. $g(i, w)$ is the maximum value that can be reached using the i first items

and using a capacity w of the bag. The recursive formula is the following:

$$\forall\, i \in [\![1, N]\!], \forall\, w \in [\![0, C_{max}]\!]$$

$$g(i,w) = \begin{cases} \max\{g(i-1, w-w_i) + v_i, g(i-1,w)\} & \text{if } w - w_i \geq 0 \\ g(i-1,w) & \text{otherwise} \end{cases}$$

The initial conditions are $g(0,*) = 0$. The optimal solution is given by $g(N, C_{max})$. In the example, the optimal value is $g(4,3) = 9$. The algorithm runs in pseudo-polynomial time $O(NC_{max})$.

There exists another way of implementing the algorithm DPKnap. Forward and backward recursion are two ways of solving a DP algorithm. In the case of the Knapsack problem the reverse DP is computed by taking the items in the reverse order. $g_r(i,w)$ is the maximum value that can be reached using the $n-i$ last items and using a capacity w of the bag. The recursion formula is:

$$\forall\, i \in [\![1, N]\!], \forall\, w \in [\![0, C_{max}]\!]$$

$$g_r(i,w) = \begin{cases} \max\{g_r(i+1, w-w_i) + v_i, g_r(i+1,w)\} & \text{if } w - w_i \geq 0 \\ g_r(i+1,w) & \text{otherwise} \end{cases}$$

The initial conditions are $g_r(N,*) = 0$. The optimal solution is given by $g_r(0, C_{max})$. In the example, the optimal value is given by $g_r = (4,3) = g(0,3) = 9$.

1.3.2 Filtering via the interpretation of DP as a graph

The filtering algorithm that we present uses the interpretation of a DP algorithm as a graph assuming a finite number of discrete states. A node in this graph corresponds to a state of the execution of the DP algorithm. The graph with all the states of the example is shown in figure 1.1. The costs on the arcs are the increase of the objective function when going from one state to the next. The optimal solution is obtained by finding a longest path in this graph: the two optimal solutions of the example are shown in bold in the graph.

We use this graph and the graph of the reverse DP to design a filtering algorithm. This is based on the principle of resource-based filtering that exploits resource additivity. It originates from rules used to simplify big graph instances [9, 14] and later used in CP to compute shortest path filtering [45]. The two graphs and lower bound of the objective (upper bound in case of a minimization problem) allow for the filtering of the decision variables. When representing all the states in a graph, the optimal solution is found by solving a path problem (finding the longest – shortest when minimizing – path from the initial state to the final state usually). As the Knapsack problem is a maximization problem, take a lower bound LB on the objective – any feasible solution for instance. We filter the variables with the following rules:

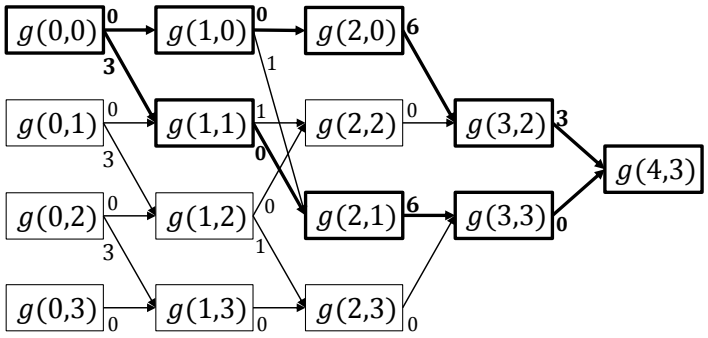

Figure 1.1: The graph of DPKnap

$$\forall\, i \in [\![1, N]\!]$$

$$\max_{w \in [\![0, C_{max}]\!]} g\,(i-1, w) + v_i + g_r\,(i+1, w - w_i) < LB \Rightarrow \text{item } i \text{ cannot be taken}$$

$$\max_{w \in [\![0, C_{max}]\!]} g\,(i-1, w) + g_r\,(i+1, w) < LB \Rightarrow \text{item } i \text{ has to be taken}$$

These rules are based on sums of partial upper bounds. For the first one, we add:

- $g\,(i-1, w)$: an upper bound of the value of considering the $i-1$ first items
- v_i: the cost of taking item i
- $g_r\,(i+1, w - w_i)$: an upper bound of the value of considering the $n - i + 1$ last items

The resulting sum is the maximum value that can be reached when taking item i in the bag. We then compare this sum to the global lower bound of the problem and if it is strictly smaller, this means that item i cannot be taken in an optimal solution. For the second one, we add:

- $g\,(i-1, w)$: an upper bound of the value of considering the $i-1$ first items
- $g_r\,(i+1, w)$: an upper bound of the value of considering the $n - i + 1$ last items

The result is the maximum value that can be reached when *not* taking item i in the bag. Again, if this sum is smaller than the lower bound, we can deduce that item i has to be taken into the bag. Here if we have a lower bound $LB = 8$, item 3 cannot be taken in the bag, item 2 has to be taken and items 1 and 4 are not fixed.

1.4 Other optimization tools

1.4.1 Mixed integer linear programming

Mixed Integer Linear Programming (MILP) is a very common optimization technique, widely used in operations research. It is the process of optimizing a linear function subject to a finite number of linear equality and inequality constraints on integer and real variables. Generic solvers such as CPLEX get more and more efficient and are typically used to fast model and solve a lot of production planning problems. The reader can refer to [31], [101] and [71] to get a more detailed introduction to MILP and its applications to production planning.

1.4.2 Integrated methods

The three optimization methods presented here (CP, DP and MILP) are often used together to make the most of the advantages of all of the techniques. The filtering algorithms of global constraints are often derived from OR methods (graph theory, network flows, dynamic programming, linear programming, etc). Some problems are also solved using hybrid methods [53, 84] in the case where CP or MILP can have difficulties on some parts of the problems. Hybridization sometimes happens by decomposing the problem into a MILP master problem and a CP slave problem. Having a CP master problem then a MILP slave problem is uncommon since relaxations and cuts are both better understood in MILP and feasibility sub-problems, where inference be very strong, are usually better solved with CP. Other work successfully integrate MILP and CP to solve problems that were intractable with only one of the two methods [3, 24, 91].

Chapter 2

Single-item lot-sizing

2.1 Introduction

The field of production planning addresses numerous complex problems covered by operations research and combinatorial optimization. In particular, lot-sizing problems have been broadly studied. The core problem [99] and several variants have been solved by Dynamic Programming (DP) in polynomial or pseudo-polynomial time. Other variants (e.g. time varying production capacity and setup costs, multi-echelon) are NP-hard and are most of the time dealt with Mixed Integer Linear Programming (MILP) formulations (see e.g. [13, 71]).

State-of-the-art approaches for complex lot-sizing problems are currently based on polyhedral techniques such as cutting plane algorithms and can handle a large class of problems with side constraints. Nonetheless theses techniques may eventually fail when facing combinatorial additional constraints. In this chapter, we investigate alternative generic approaches based on combinatorial techniques and designed within the Constraint Programming (CP) framework. The rationale is that a lot of algorithmic results have been obtained on the fundamental problems in this field over the last sixty years. We propose to reuse them as filtering mechanisms and building blocks of a generic solver for lot-sizing. This thesis is a first step in that direction: we introduce a new global constraint LOTSIZING embedding the single-item lot-sizing problem. LOTSIZING appears to be especially generic and suits well in the modeling of a great variety of lot-sizing problems. The problem being NP-hard, we prove several complexity results on achieving different consistency levels for the constraint. We use a time decomposition to propose a new lower bound for the single-item lot-sizing problem. This time decomposition combined with classical results, namely DP algorithms, enables us to derive interesting cost-based filtering algorithms for LOTSIZING.

The capacitated single-item lot-sizing problem In this chapter, we focus on the following single-item lot-sizing problem – denoted by (L) – which is used as a building block to tackle more complex lot-sizing problems. The objective is to plan the production of a single product over a finite horizon of T periods $[\![1, T]\!]$ in order to satisfy a demand d_t at each period t, and to minimize the total cost. The (per unit) production cost at t is p_t and a setup cost s_t is paid if at least one unit is produced at t. An holding cost h_t is paid for each unit stored at the end of period t. Furthermore the production (resp. the inventory) is bounded by minimal and maximal capacities $\underline{\alpha_t}$ and $\overline{\alpha_t}$ (resp. $\underline{\beta_t}$ and $\overline{\beta_t}$) at each period t.

Figure 2.1 shows the problem as a graph with the bounds of the variables on each arc. For each period, the incoming arcs corresponds to the possible production (vertical arcs) and inventory from the previous period (horizontal arcs). The outgoing arcs correspond to the demand (vertical arcs) and inventory at the end of the period (horizontal arcs). A mathematical model is given in 2.2.2.

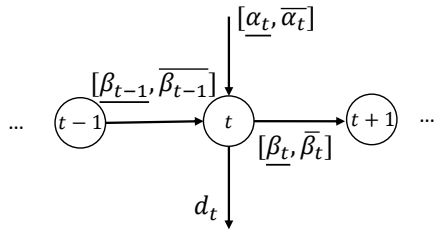

Figure 2.1: Flow representation of the single-item lot-sizing problem

In the literature, one can found several models with upper bounds on either production or inventory. It is however unusual to include lower bounds. We make this assumption to be consistent with the CP framework that states domains for the variables.

Literature review The CP literature is very limited in the field of lot-sizing problems. To the best of our knowledge, [52] is the only paper to study a global constraint. They consider a production planning problem in which a set of items has to be produced before their production deadline on a limited capacity machine, with the objective of minimizing stocking costs. This problem can be solved in polynomial time and is a special case of (L) where production costs are set to zero ($p_t = s_t = 0, h_t = h$). It can be seen as a scheduling problem with deadlines and the objective of minimizing the total earliness ($1|\tilde{d}_j, p_j = 1|\sum E_j$) with Graham notation). In their approach, a decision variable is associated to each item and specifies in which period the item has to be produced. This approach is suitable to deal with scheduling problems but seems less relevant to address lot-sizing problems for

which large quantities of the same item can be produced in the same period. Note also that CP solvers have been used in the past to solve lot-sizing problems (see e.g. [90] for a distribution multi-echelon system).

We now focus the literature review on some special cases of problem (L). There is no paper, to our knowledge, that considers lower bounds on production and inventory. [99] shows that the uncapacitated problem($\overline{\alpha_t} = \overline{\beta_t} = +\infty$) can be solved by DP in $O(T^2)$. This complexity has later been improved to $O(T \log T)$ [5, 38, 98]. When adding a constant production capacity and a constant setup cost, ($s_t = s$, $\overline{\alpha_t} = \alpha$), the problem can be solved in $O(T^4)$ with concave costs [41] and in $O(T^3)$ with linear costs [95]. When the production capacity varies with time, the problem is NP-hard [22]. Note that when $p_t = h_t = 0$, (L) is equivalent to a knapsack problem. With time-varying inventory capacities, the problem can be solved in $O(T^2)$ with production and inventory setup costs [11, 62].

The rest of the chapter is organized as follows. Section 2.2 presents algorithms from the literature that will be re-used latter. Section 2.3 presents a new lower bound for this problem based on a time decomposition. Section 2.4 presents the LOTSIZING global constraint and states complexity results for achieving bound and range consistency. Section 2.5 presents cost-based filtering mechanisms for LOTSIZING. Section 2.6 compares numerically the performances of LOTSIZING with two MILP formulations, DP and a basic CP model. Section 2.7 considers two extensions with side constraints.

2.2 Preliminaries

This section presents classical MILP formulations and DP approaches, that will be used latter in the chapter. We also show that problem (L) is equivalent to a problem without lower bounds on production and inventory.

2.2.1 Notations and example

We list below a summary of the main notations.

Parameters

- $T \in \mathbb{N}$: Number of periods.
- $p_t \in \mathbb{N}$: Unit production cost at t.
- $h_t \in \mathbb{N}$: Unit holding cost between t and $t+1$.
- $s_t \in \mathbb{N}$: Setup cost at t (paid if at least one item is produced at t).

- $d_t \in \mathbb{N}$: Demand at t.
- $\underline{\alpha_t}, \overline{\alpha_t} \in \mathbb{N}$: Minimal and maximal production quantities at t.
- $\underline{\beta_t}, \overline{\beta_t} \in \mathbb{N}$: Minimal and maximal inventory at the end of period t.
- $I_0 \in \mathbb{N}$: Initial inventory.

Variables

- $X_t \in \mathbb{N}$: Quantity produced at t.
- $Y_t \in \{0, 1\}$: Setup variable that equals 1 if at least one item is produced at t, 0 otherwise.
- $I_t \in \mathbb{N}$: Inventory at the end of period t.
- $C \in \mathbb{N}$: Total cost.
- $Cp \in \mathbb{N}$: Sum of unit production costs.
- $Cs \in \mathbb{N}$: Sum of setup production costs.
- $Ch \in \mathbb{N}$: Sum of holding costs.

We denote by X, I and Y the vectors $\langle X_1, \ldots, X_T \rangle$, $\langle I_1, \ldots, I_T \rangle$ and $\langle Y_1, \ldots, Y_T \rangle$. Without loss of generality we consider $I_0 = 0$. We also consider $I_T = 0$. Indeed, we can compute the minimum mandatory quantity to store at the end of period T from the production and inventory capacity constraints. If this quantity q is strictly positive, we add a dummy period $T + 1$ at the end of the time horizon with $p_{T+1} = h_{T+1} = 0$, $\overline{\alpha_{T+1}} = \overline{\beta_{T+1}} = 0$ and $d_{T+1} = q$.

To better understand the optimization challenges of the problem, take the example in figure 2.2. The parameters are directly available on the graph.

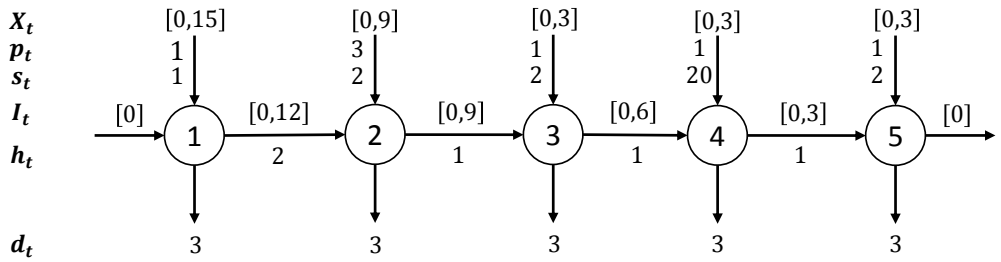

Figure 2.2: A small example

Let us look at three different solutions. Figure 2.3 presents a solution that intends to minimize the global setup cost: the whole demand is produced at the beginning of the horizon. The consequence is that the global inventory cost is very high. Figure 2.4 presents a solution that intends to minimize the global inventory cost: the demand is produced at each period with the consequence that the global setup cost is high. Finally, the optimal solution is presented in figure 2.5 the production, inventory and setup costs are optimized while taking into account the production capacities.

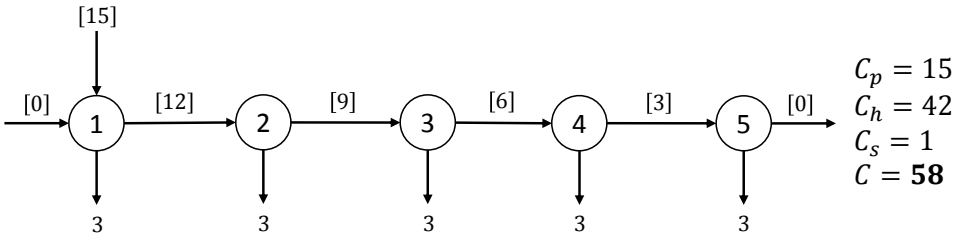

Figure 2.3: Minimizing the setup cost

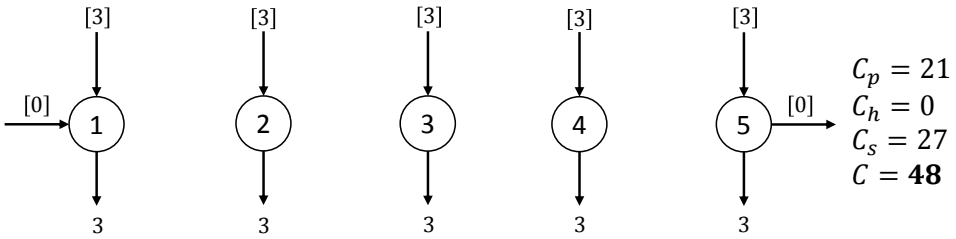

Figure 2.4: Minimizing the inventory cost

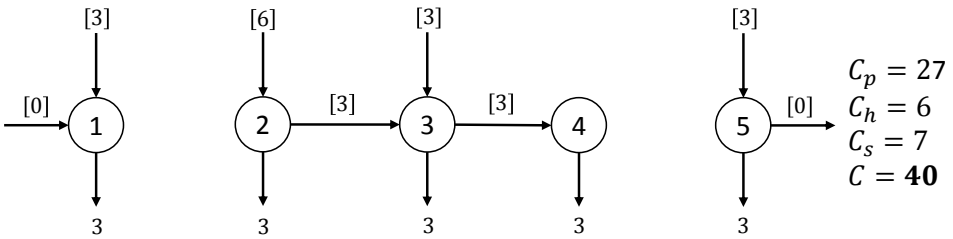

Figure 2.5: An optimal solution

2.2.2 Mixed integer linear formulations

Problem (L) can be fomulated with an aggregated MILP model (see e.g. [71]):

$$
\begin{aligned}
& \textit{minimize } C = Cp + Ch + Cs && && (2.1)\\
& I_{t-1} + X_t = d_t + I_t && \forall\, t = 1 \dots T && (2.2)\\
& X_t \leq \overline{\alpha_t} Y_t && \forall\, t = 1 \dots T && (2.3)\\
& Cp = \sum_{t=1}^{T} p_t X_t && && (2.4)\\
(MILP\_AGG) \qquad & Ch = \sum_{t=1}^{T} h_t I_t && && (2.5)\\
& Cs = \sum_{t=1}^{T} s_t Y_t && && (2.6)\\
& X_t \in \{\underline{\alpha_t}, \dots, \overline{\alpha_t}\} && \forall\, t = 1 \dots T && (2.7)\\
& I_t \in \{\underline{\beta_t}, \dots, \overline{\beta_t}\} && \forall\, t = 1 \dots T && (2.8)\\
& Y_t \in \{0, 1\} && \forall\, t = 1 \dots T && (2.9)
\end{aligned}
$$

where (2.2) are the flow balance constraints for each period and (2.3) are the setup constraints enforcing Y_t to 1 if a production is made at t. Note that paying a setup cost at t (i.e. $Y_t = 1$) does not necessarily imply a production at t (i.e. $X_t > 0$), which will be useful in multi-item problems with shared setup costs. Finally, (2.4), (2.5) and (2.6) express the various costs. When (L) is solved as a MILP, the variables X and I can be relaxed and considered real [71].

(L) can also be modeled as a facility location problem [57]: the variables I and X are channeled to the variables X_{tr}, where X_{tr} represents the proportion of demand

d_r produced in period t and stored from t to r. The model can be written as follows:

$$(MILP\_UFL) \qquad \begin{aligned} & (2.1), (2.4), (2.5), (2.6), (2.7), (2.8), (2.9) \\ & X_t = \sum_{r=t}^{T} d_r X_{tr} && \forall\, t = 1 \dots T && (2.10) \\ & I_t = \sum_{q=1}^{t} \sum_{r=t+1}^{T} d_r X_{qr} && \forall\, t = 1 \dots T && (2.11) \\ & X_{tr} \leq Y_t && \forall\, t = 1 \dots T,\ r = t \dots T && (2.12) \\ & \sum_{t=1}^{r} X_{tr} = 1 && \forall\, r = 1 \dots T && (2.13) \\ & X_{tr} \in [0, 1] && \forall\, t = 1 \dots T,\ r = t \dots T && (2.14) \end{aligned}$$

Though the number of variables is increased, this model has the advantage to tighten the *big M constraints* (2.3) of the first formulation by stating constraints (2.12) and is known to generally provide a better linear relaxation.

2.2.3 Linear relaxation

Solving the linear relaxation of MILP_AGG (i.e. $Y_t \in [0, 1], \forall\, t \in [\![1, T]\!]$) is equivalent to a minimum cost network flow problem [7]. The graph of this flow is presented in Figure 2.6. On each arc, (u, c) represents the capacity (u) and unitary cost (c) of the arc. The units flow from source node S to sink node W. On each production arc (S, t), the capacity is the production capacity and the cost is $p'_t = p_t + \frac{s_t}{\alpha_t}$. On each inventory arc (t, $t+1$), the capacity is the inventory capacity and the cost is h_t. Finally on each demand arc (t, W), there must be exactly d_t units and the unitary cost is 0.

The flow problem can be solved in $O(T^2)$ with the successive shortest path algorithm [7].

2.2.4 An equivalent problem without lower bounds

In this subsection, we show that problem (L) is equivalent to a problem without lower bounds. It will allow us to re-use several classical lot-sizing algorithms and will also simplify the presentation of some results.

Solving a maximum flow problem on a network with lower bounds on flows is equivalent to solving a maximum flow problem on a transformed network without

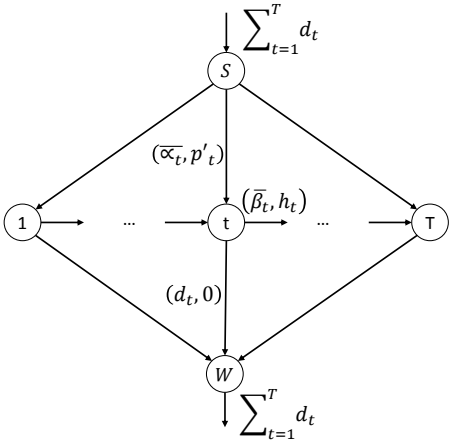

Figure 2.6: The linear relaxation of MILP_AGG is a minimum cost network flow problem

lower bounds as shown in [7]. We denote by (L') the resulting problem of that transformation applied to (L) slightly adapted to take into account setup costs. The parameters of (L') are:

$$
\begin{aligned}
&\underline{X'_t} = 0 \text{ and } \overline{X'_t} = \overline{X_t} - \underline{X_t} \\
&\underline{I'_t} = 0 \text{ and } \overline{I'_t} = \overline{I_t} - \underline{I_t} \\
&p'_t = p_t \\
&h'_t = h_t \\
&s'_t = \begin{cases} 0 & \text{if } \underline{X_t} > 0 \\ s_t & \text{otherwise} \end{cases} \\
&d'_t = d_t + \underline{I_t} - \underline{X_t} - \underline{I_{t-1}}
\end{aligned}
$$

(X, I) is a solution of (L) if and only if (X', I') is a solution of (L'). The intuition is that the production and inventory lower bounds are considered as mandatory quantities. As these quantities must be produced/stored at a precise period, no decisions have to be made about them and thus they can be removed from the problem. Note that the demands are also affected by the transformation.

The mandatory costs associated to the lower bounds are:

$$\begin{aligned} Cp_{min} &= \sum_{t=1}^{T} p_t \, \underline{X_t} \\ Ch_{min} &= \sum_{t=1}^{T} h_t \, \underline{I_t} \\ Cs_{min} &= \sum_{t=1}^{T} s_t \, \mathbb{1}_{\underline{X_t}>0} \\ C_{min} &= Cp_{min} + Ch_{min} + Cs_{min} \end{aligned}$$

and the variables of (L) and (L') are linked as follows:

$$\begin{aligned} X_t &= X'_t + \underline{X_t} \\ I_t &= I'_t + \underline{I_t} \\ Cp &= Cp' + Cp_{min} \\ Ch &= Ch' + Ch_{min} \\ Cs &= Cs' + Cs_{min} \\ C &= C' + C_{min} \end{aligned}$$

2.2.5 Dynamic programming

(L) can also be solved via DP [41]. We provide here the algorithm without lower bounds on production and inventory. The algorithm (called DPLS in the thesis) iterates over the inventory levels. We denote $f(t, I_t)$ as the minimum cost for producing the demands from d_1 to d_t knowing that the stock level at t is I_t:

$$\begin{aligned} &\forall\, t \in [\![1, T]\!] \text{ and } \forall\, I_t \in [\![0, \overline{\beta_t}]\!] \\ &f(t, I_t) = \min_{I_{t-1}=a\ldots b} \{f(t-1, I_{t-1}) + \mathbb{1}_{X_t>0} s_t + p_t X_t + h_t I_t\} \end{aligned} \tag{2.15}$$

where $a = \max\{0, d_t + I_t - \overline{\alpha_t}\}$, $b = \min\{\overline{\beta_{t-1}}, d_t + I_t\}$ and $X_t = I_t + d_t - I_{t-1}$. We define $I_{max} = \max\{\overline{\beta_t} \mid t \in [\![1, T]\!]\}$. The initial states are $f(0, 0) = 0$ and $\forall\, I_t \in [\![1, I_{max}]\!]$, $f(0, I_t) = +\infty$. The value $f(T, 0)$ gives the optimal cost of (L). This dynamic programming algorithm runs in pseudo-polynomial time $O(TI_{max}^2)$.

Note that DPLS consists in finding a shortest path in the graph for which there is a node for each inventory level at each period. The cost on an arc between two nodes (t, I_t) and $(t+1, I_{t+1})$ corresponds to the cost for satisfying demand d_t and having an inventory level I_{t+1} at the end of period $t+1$ knowing that there was an inventory level I_t at the end of period t.

DPLS is a DP algorithm referred to as "forward" since it considers the periods in chronological order. We can also write the reverse (or "backward") DPLS. Let $f_r(t, I_t)$ be the minimum cost for producing the demands from d_{t+1} to d_T knowing that the stock level at t is I_t:

$$\forall\, t \in [\![0, T-1]\!] \text{ and } \forall\, I_t \in [\![0, \overline{\beta_t}]\!]$$
$$f_r(t, I_t) = \min_{I_{t+1}=c\ldots d} \{f_r(t+1, I_{t+1}) + \mathbb{1}_{X_{t+1}>0} s_{t+1} + p_{t+1}X_{t+1} + h_{t+1}I_{t+1}\}$$

where $c = \max\{0, I_t - d_{t+1}\}$, $d = \min\{\overline{\beta_{t+1}}, I_t - d_{t+1} + \overline{\alpha_{t+1}}\}$ and $X_{t+1} = d_{t+1} + I_{t+1} - I_t$. The initial states are $f_r(T, 0) = 0$ and $\forall\, I_t \in [\![1, I_{max}]\!]$, $f_r(T, I_t) = +\infty$. As described above, $f_r(t, I_t)$ can be seen as the shortest path from the node (t, I_t) to the node $(T, 0)$.

2.3 A new lower bound for the single-item lot-sizing

In this section, we present a new lower bound for the total cost C and how it can be adapted for the setup cost C_s. The general idea is to decompose (L) into sub-problems, then to compute a lower bound on each of these sub-problems and finally combine them at best to find a global lower bound. We suppose here that $\underline{\alpha_t} = \underline{\beta_t} = 0$, $\forall\, t \in [\![1, T]\!]$. This assumption is not restrictive as production and inventory lower bounds can be easily removed in (L) as shown in 2.2.4.

2.3.1 Lot-sizing sub-problem

A sub-problem $(L_{u,v})$, with $u < v$, is defined exactly as (L) except that:

$$d_t = 0 \qquad \forall\, t \notin [\![u, v]\!]$$
$$s_t = 0 \qquad \forall\, t \notin [\![u, v]\!]$$

The cost variables of $(L_{u,v})$ are denoted C^{uv}, C_p^{uv}, C_h^{uv} and C_s^{uv}, corresponding to the total cost, sum of production costs, sum of holding costs and sum of setup costs of $(L_{u,v})$. Figure 2.7 illustrates the data used in sub-problem $(L_{u,v})$.

Sub-problem (L_{1T}) corresponds to the entire problem (L). As there is no demand after period v and no lower bounds of production, the solutions of $(L_{u,v})$ are dominated by solutions with null inventory at the end of period v ($I_v = 0$). This means that there exists an optimal solution of $(L_{u,v})$ with zero inventory at the end

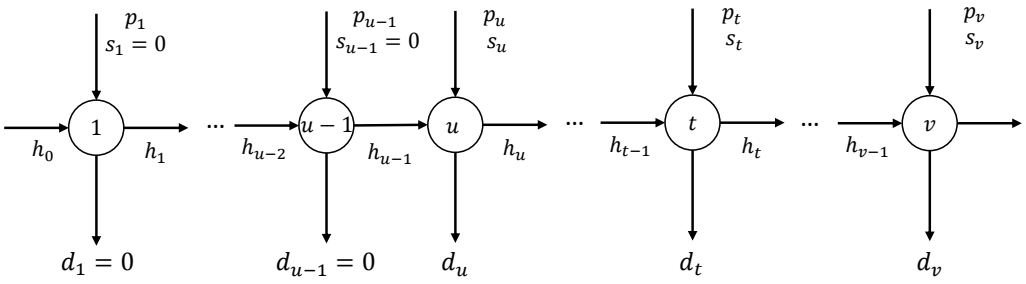

Figure 2.7: Sub-problem $(L_{u,v})$

of the last period. Note also that, in $(L_{u,v})$, some demands in $\{d_u, d_{u+1}, \ldots, d_v\}$ can be satisfied by a production made without setup costs before period u. Finally, an optimal solution of $(L_{u,v})$ provides a lower bound of the cost for satisfying the set of demands $\{d_u, d_{u+1}, \ldots, d_v\}$ in problem (L). Indeed $(L_{u,v})$ is a relaxation of problem (L).

There are $T(T-1)/2$ sub-problems and we order them by increasing end times first, then by increasing start times (see Table 2.1). Sub-problem (L_{u_i,v_i}) will be referred to as sub-problem i.

Index	1	2	3	4	5	6	...	$\frac{T(T-1)}{2}$
Sub-problem	$(L_{1,2})$	$(L_{1,3})$	$(L_{2,3})$	$(L_{1,4})$	$(L_{2,4})$	$(L_{3,4})$	...	$(L_{T-1,T})$

Table 2.1: Indexing sub-problems

Definition 8 *Sub-problems $(L_{u,v})$ and $(L_{u',v'})$ are **disjoint** if*

$$[\![u, v]\!] \cap [\![u', v']\!] = \emptyset$$

2.3.2 Combining disjoint sub-problems provides a lower bound

For sub-problem i, we denote by w_i a lower bound of its total cost $C^{u_i v_i}$. Disjoint sub-problems can be combined to obtain a lower bound for the total cost of (L).

Theorem 1 *For any set S of disjoint sub-problems, we have*

$$\sum_{i \in S} w_i \leq C.$$

Proof. Let E^* be an optimal production plan for (L) of cost C^* and S be a set of disjoint sub-problems. We will build from E^* a feasible solution to each sub-problem i of S and prove that their costs add up to less than C^*.

Consider a sub-problem i in S. For each demand d_t, $t \in [\![u_i, v_i]\!]$ we produce d_t at the same periods as it is produced in E^* (with a First Come First Served policy). We obtain by this process a feasible solution to sub-problem i and denote its cost by K_i.

As the sub-problems in S are disjoint and we keep the same production orders, the sum of setup costs payed in all of these sub-problems is less than or equal to the sum of setup costs payed in E^*. The production and inventory costs are identical to the costs payed in E^* for the demands included in $\cup_{i \in S} [\![u_i, v_i]\!]$. It follows that $\sum_{i \in S} K_i \leq C^*$. Finally, as for each sub-problem i, w_i is a lower bound of $C^{u_i v_i}$, we get: $\sum_{i \in S} w_i \leq \sum_{i \in S} K_i$.

□

2.3.3 Combining lower bounds at best

Given a lower bound for each sub-problem, we wish to find the best lower bound of C, i.e. to determine the set S of disjoint sub-problems that maximizes $\sum_{i \in S} w_i$.

This problem can be seen as a Weighted Interval Scheduling Problem (WISP) which can be solved in $O(n \log(n))$ where n is the number of intervals [56]. The algorithm sorts the intervals in $O(n \log(n))$ and then applies a DP that runs in $O(n)$. In our case, there are $n = T(T-1)/2$ intervals (sub-problems) which are already sorted and the DP algorithm (called DPWisp in the thesis) runs in $O(T^2)$.

We use the indexing of intervals given in Table 2.1 and we denote by $wisp(i)$ the maximal weight that can be achieved using the i first intervals. The forward DP writes as

$$\begin{aligned} &wisp(0) = 0 \\ &wisp(i) = \max\{wisp(i-1), wisp(prec_i) + w_i\}, \forall\, i = 1, \ldots, n \end{aligned}$$

where $prec_i$ is the biggest integer, smaller than i ($prec_i < i$), such as the intervals $prec_i$ and i are disjoint. For each sub-problem i such that $u = 1$, we define $prec_i = 0$. Hence $prec_i$ is the first interval before the i^{th} one that is disjoint with it. For instance, $[3,4]$ is the 6^{th} interval and $prec_6 = 1$ since $[1,2]$ and $[3,4]$ are disjoint while $[2,3]$ and $[3,4]$ are not. The value $wisp(n)$ is then a lower bound of the global cost C.

We can also write the reverse version DPWisp (with the sub-problems considered backwards). The sub-problems are now sorted by decreasing start times first, then by decreasing end times.

$$\begin{aligned} &wisp_r(n) = 0 \\ &wisp_r(i) = \max\{wisp_r(i+1), w_i + wisp_r(succ_i)\}, \forall\, i = 1, \ldots, n \end{aligned}$$

where $succ_i$ is the biggest integer, greater than i ($succ_i > i$), such as the intervals $succ_i$ and i are disjoint.

2.3.4 Computing lower bounds for sub-problems

In our algorithms, we will solve exactly the sub-problem by DP when the size is reasonable and solve the linear relaxation otherwise. The optimal cost of sub-problem $(L_{u,v})$ can be computed with DPLS (see Section 2.2.5) applied to the periods u to v. We need to pre-compute $f(u-1, q_{u-1})$ for $q_{u-1} \in [\![0, \min\{\sum_{t=u}^{v} d_t, \overline{\beta_{u-1}}\}]\!]$. This can be done with a greedy algorithm which determines the cheapest periods in order to produce the requested quantity and to store it until u. The DPLS applied to a sub-problem $(L_{u,v})$ runs then in $O((v-u+1)(I_{max}^{uv})^2)$ where $I_{max}^{uv} = \max\{\overline{\beta_t} \mid t \in [\![u, v]\!]\}$.

The linear relaxation can be seen as a minimum cost flow problem (see Section 2.2.3) and can be solved in $O(T^2)$.

2.3.5 Adaptation to a lower bound on setup costs

We can re-use this previous approach to obtain a lower bound on the setup cost variable *Cs*. Sub-problems are defined similarly except that we remove unitary production costs and inventory costs ($p_t = h_t = 0, \forall t$).

Note that we could use the same approach for *Cp* and *Ch*. However, it is not necessary as (L) is polynomial when removing setup costs.

2.4 The lot-sizing global constraint

In this section, we provide some CP background before presenting the LOTSIZING global constraint. We also study the complexity of achieving different consistency levels.

2.4.1 Constraint programming background

A Constraint Satisfaction Problem (CSP) [50] consists of a set of variables, with a finite domain of values for each variable, and a set of constraints on these variables. Upper cases are used for variables (e.g. V_i) and lower cases for values (e.g. v_i). We denote by $D(V_i)$ the domain of variable V_i and by $\overline{V_i}$ (resp. $\underline{V_i}$) the minimum (resp. maximum) value in $D(V_i)$.

Let c a constraint on variables $\langle V_1, \ldots, V_n \rangle$. A *support* for c is a tuple $\langle v_1, \ldots, v_n \rangle$ which satisfies c and such that $v_i \in D(V_i)$ for each variable V_i. A *bound support* is a tuple $\langle v_1, \ldots, v_n \rangle$ which satisfies c and such that $\underline{V_i} \leq v_i \leq \overline{V_i}$ for each V_i and $V_i \in \mathbb{Z}$.

A variable V_i is *arc consistent* (AC) for constraint c if each value of $D(V_i)$ belongs to a support for c. A variable V_i is *bounds consistent* (BC) for constraint c if $\underline{V_i}$ and $\overline{V_i}$ belong to a bound support for c. A variable V_i is *range consistent* (RC) for constraint c if each value of $D(V_i)$ belongs to a bound support for c. A constraint c is AC (resp. BC, RC) if all its variables are AC (resp. BC, RC). A CSP problem is AC (resp. BC, RC) if each constraint is AC (resp. BC, RC).

2.4.2 Definition

We formally define here the global constraint LOTSIZING. This constraint is stated on the variable vectors $X = \langle X_1, \ldots, X_T \rangle$, $I = \langle I_1, \ldots, I_T \rangle$, $Y = \langle Y_1, \ldots, Y_T \rangle$ and the four cost variables Cp, Ch, Cs, C of (L). The data of the problem is denoted by $data = \{(p_t, h_t, s_t, d_t, \underline{\alpha_t}, \overline{\alpha_t}, \underline{\beta_t}, \overline{\beta_t}) \mid t \in [\![1, T]\!]\}$.

Definition 9 LOTSIZING$(X, I, Y, Cp, Ch, Cs, C, data)$ *has a solution if and only if there exists a production plan, solution of* (L) *that satisfies:*

$$Cp \leq \overline{Cp} \tag{2.16}$$
$$Ch \leq \overline{Ch} \tag{2.17}$$
$$Cs \leq \overline{Cs} \tag{2.18}$$
$$C \leq \overline{C} \tag{2.19}$$

The LOTSIZING global constraint has a solution if and only if the set of constraints $\{(2.2), \ldots, (2.9), (2.16), \ldots, (2.19)\}$ has a solution.

2.4.3 Complexity

This subsection presents theorems on the complexity of achieving BC, RC or AC on LOTSIZING and one of its restrictions that does not take into account the costs and focuses on the flow equations. Let us first give a property on the complexity of (L).

Property. *Problem* (L) *with* $h_t = p_t = \underline{\beta_t} = \underline{\alpha_t} = 0$ *and* $\overline{\beta_t} = +\infty$ *is NP-hard.*

Proof. The reduction is made with the knapsack problem:
Input: n objects of weight w_i, of values v_i. Two integers V and W.
Question: Is there a subset S of the objects such that $\sum_{i \in S} v_i \geq V$ and $\sum_{i \in S} w_i \leq W$?
We define $s_t = w_t$, $\overline{X_t} = v_t$, $\forall\, t = 1 \ldots T$ and $d = \{0, .., 0, V\}$. We look for a production plan such that $\sum_{t \in S} X_t \geq V$ and $\sum_{t \in S | X_t > 0} s_t \leq W$. This transformation is polynomial. Any solution of the lot-sizing problem is equivalent to a solution where we produce at full capacity. We can therefore restrain ourselves to this set

of solutions. Any solution to the knapsack problem is also a solution of this lot-sizing problem. Conversely, a solution of the lot-sizing problem is a solution to the knapsack problem. □

Theorem 2 *Achieving BC for* LOTSIZING *can be done in pseudo-polynomial time.*

Proof. BC for LOTSIZING can be achieved by solving a Shortest Path Problem with Resource Constraints (SPPRC) in the graph of DPLS (see Section 2.2.5). where the resources are the three intermediate costs of respective capacities $\overline{Cp}, \overline{Ch}, \overline{Cs}$ and the global cost variable C with capacity $\overline{C}$. SPPRC is known to be weakly NP-hard [39]. □

We denote by (L_r) the feasibility problem associated to (L).

$$
(L_r) \quad \begin{aligned}
I_{t-1} + X_t &= d_t + I_t && \forall\, t = 1 \dots T \\
X_t &\leq \overline{\alpha_t} Y_t && \forall\, t = 1 \dots T \\
X_t &\in \{\underline{\alpha_t}, \dots, \overline{\alpha_t}\} && \forall\, t = 1 \dots T \\
I_t &\in \{\underline{\beta_t}, \dots, \overline{\beta_t}\} && \forall\, t = 1 \dots T \\
Y_t &\in \{0, 1\} && \forall\, t = 1 \dots T
\end{aligned}
$$

The constraints of (L_r) describes the dynamics of the problem, *i.e.* the feasibility of the problem without considering the upper bounds on the costs.

Theorem 3 *Achieving BC on* (L_r) *can be done in* $O(T)$.

Proof. Figure 2.8 represents the constraint network (L_r) as well as the corresponding intersection graph. The rectangular-shaped constraints are the flow balance constraints and the dashed oval ones are the setup constraints. The intersection graph is built as follows: we set a vertex for each constraint and two vertices are linked if and only if the corresponding constraints have at least one variable in common. As the intersection graph is acyclic and each pair of constraints has at most one variable in common, the constraint network is Berge-acyclic. It is known that if we filter each constraint of a Berge-acyclic constraint network in an appropriate order then each constraint needs only to be woken twice in order to reach the fix-point [58]. Each constraint of the network can be filtered in $O(1)$, hence we can achieve BC on (L_r) in $O(T)$.

□

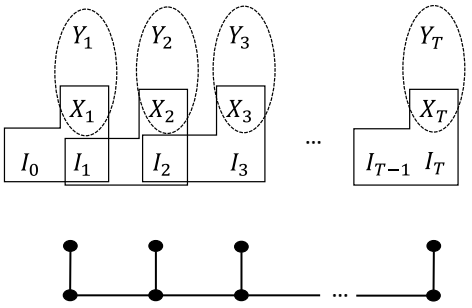

Figure 2.8: The constraint network (L_r) and the corresponding intersection graph

In order to investigate RC for (L_r), we consider (F) the more general problem of finding an integer flow in a directed graph $G = (V, E)$:

$$(F) \qquad \begin{aligned} l_{ij} \leq x_{ij} &\leq u_{ij} && \forall\, (i,j) \in E \\ \sum_{j \in \delta_i^+} x_{ij} - \sum_{j \in \delta_i^-} x_{ji} &= b_i && \forall\, i \in V \\ x_{ij} &\in \mathbb{N} && \forall\, (i,j) \in E \end{aligned}$$

where x_{ij} is the flow going from node i to node j, δ_i^+ (resp. δ_i^-) is the set of successor (resp. predecessor) nodes of node i and $b_i \in \{-1, 0, 1\}$.

Theorem 4 *BC and RC are equivalent for* (F).

Proof. By definition, RC implies BC for (F). Conversely, assume BC for (F).

In order to show the converse, we show that if $k \in [\![l_{ij}, u_{ij}]\!]$, then k belongs to a bound support for (F).

Let i_0 be a direct predecessor of j_0 in the graph. i_0 and j_0 are periods in the lot-sizing problem. BC implies that there exists x' (resp. x'') an integer solution of (F) such that $x'_{i_0 j_0} = l_{i_0 j_0}$ (resp. $x''_{i_0 j_0} = u_{i_0 j_0}$). Let $\gamma \in [0, 1]$ such as $k = \gamma\, l_{i_0 j_0} + (1 - \gamma) u_{i_0 j_0}$ and $k \in \mathbb{N}$. Let's show that there exists an integer solution of (F) where $x_{i_0 j_0} = k$. We modify the bounds $\forall\, (i, j) \in E$:

$$\widehat{l_{ij}} = \lfloor \gamma x'_{ij} + (1-\gamma) x''_{ij} \rfloor$$
$$\widehat{u_{ij}} = \lceil \gamma x'_{ij} + (1-\gamma) x''_{ij} \rceil$$

We can then show that $\forall\, i, j\ \widehat{l_{ij}} \geq l_{ij}$ and $\widehat{u_{ij}} \leq u_{ij}$. The constraint matrix of (F) is totally unimodular since it is a flow problem, hence the application of the simplex algorithm to (F) with the updated bounds gives an integer solution x''' where $x'''_{i_0 j_0} = k$.

□

Since the setup variables Y_t are binary, the following result follows.

Corollary 1 *Achieving BC on (L_r) is equivalent to achieving RC on (L_r).*

When there are no holes in the domains of X and I, RC is equivalent to AC for (L_r). Note that there may exist holes in lot-sizing problems when considering batching constraints for instance. Batching constraints force the products to be produced by batches of given sizes: the domain of X_t is therefore $D(X_t) = \{0, 10, 20, 30\}$ with batches of size 10.

2.5 Filtering the LOTSIZING constraint

This section describes the filtering of the LOTSIZING constraint. It also gives some implementation details to improve the incrementality of the global constraint.

Algorithm 1 gives an overview of the main filtering steps of LOTSIZING. Each step refers to the corresponding section for detailed explanations. When all setup variables are instantiated the problem amounts to a minimum cost flow problem (lines 1-2). If not, the general case is as follows. Firstly, the problem is transformed by removing all lower bounds (line 4) as LOTSIZING is defined with lower bounds and these can increase during the search. Secondly production and inventory costs lower bounds are computed (lines 5-6). Thirdly, when the overall problem is of reasonable size (lines 7-11) the remaining filtering is performed using dynamic programming. If not, the WISP relaxation is used and filtering is performed via the WISP support (lines 13-17).

2.5.1 Filtering when the setup variables are instantiated

When all the setup variables Y are instantiated, problem (L) becomes polynomial and amounts to a minimum cost flow problem. Solving a minimum cost flow problem on the flow graph presented in Figure 2.6 finds a solution to (L) that minimizes C. This property does not always hold depending on the side constraints. Therefore we let the user specify if all the production and inventory variables can be instantiated to the optimal solution of the flow problem. For sake of simplicity, Algorithm 1 implements this dominance rule. Note that a minimum cost flow dedicated filtering algorithm [89] can be used at this stage. Either of these two options allows the user to branch only on the Y variables.

Algorithm 1 filtering algorithm of LOTSIZING

```
1: if all the Y are instantiated and dominance is activated (§ 2.5.1) then
2:     solve the min flow problem and instantiate X and I
3: else
4:     check feasibility (Corollary 1)
5:     remove lower bounds (§ 2.2.4)
6:     update Cp with flow relaxation restricted to production costs (§ 2.5.2)
7:     update Ch with flow relaxation restricted to inventory costs (§ 2.5.2)
8:     if the DP is scalable then
9:         update C with DPLS (§ 2.5.2)
10:        filter variables via DP filtering (§ 2.5.3)
11:        update Cs with DP (§ 2.5.2)
12:        filter variables via DP filtering (§ 2.5.3)
13:    else
14:        compute all the C_uv with appropriate relaxations (§ 2.3.4)
15:        update C with DPWisp (§ 2.3.2)
16:        filter variables via WISP support filtering (§ 2.5.4)
17:        update Cs with DPWisp (§ 2.3.5)
18:        filter variables via WISP support filtering (§ 2.5.4)
19:    end if
20: end if
21: end algorithm
```

2.5.2 Filtering cost lower bounds

Lower bounds of the cost variables are computed as follows:

- A lower bound on the production cost Cp is computed by solving a minimum cost flow problem on the graph presented in § 2.2.3 considering only the variable production costs.
- A lower bound on the inventory cost Ch is computed by solving a minimum cost flow problem on the graph presented in § 2.2.3 considering only the variable inventory costs.
- A lower bound on the global cost C is computed using DPLS. The lower bound is given by the value $f(T, 0)$.
- A lower bound on the setup cost Cs is computed via dynamic programming as well. We consider here the problem (L) without production or inventory costs. As mentioned in the literature review, this problem can be solved using the traditional knapsack dynamic programming algorithm slightly adapted

to take into account the inventory upper bounds (we call this algorithm DP-Knap).

2.5.3 Filtering X and I via dynamic programming

DPLS gives a lower bound of the global cost C. In order to filter the variables we use the tables created by DPLS and reverse DPLS. Remember that in the graph described in 2.2.5, $f(t, I_t)$ can be seen as the shortest path from the node $(0, 0)$ to (t, I_t) and $f_r(t, I_t)$ is the shortest path from (t, I_t) to $(T, 0)$. We filter each value i_t in the domain of I_t in $O(TI_{max})$:

$$\begin{aligned} &\forall\, t \in [\![1, T]\!], i_t \in D(I_t) \\ &f(t, i_t) + f_r(t, i_t) > \overline{C} \Rightarrow I_t \neq i_t \end{aligned} \tag{2.20}$$

We filter each value in the domain of X_t in $O(TI_{max}^2)$:

$$\begin{aligned} &\forall\, t \in [\![1, T]\!], i_{t-1} \in D(I_{t-1}), i_t \in D(I_t) \\ &f(t-1, i_{t-1}) + cost(t, i_{t-1}, i_t) + f_r(t, i_t) > \overline{C} \Rightarrow X_t \neq x_t \end{aligned} \tag{2.21}$$

where $x_t = d_t + i_t - i_{t-1}$ and $cost(t, i_{t-1}, i_t) = \mathbb{1}_{X_t>0} s_t + p_t x_t + h_t i_t$.

2.5.4 Scaling the filtering based on dynamic programming

In the case where DPLS has memory issues on the overall problem (L), we solve a WISP (see Section 2.3.2) to find a lower bound on C. We can adapt the filtering rules (2.20) and (2.21) on the sub-problems of reasonable size. In order to compare the shortest paths to the global upper bound $\overline{C}$, we need to have a lower bound on the cost of the production outside the sub-problem. We use DPWisp and its reverse version to do so. We can then define:

- lbBefore$(t) = wisp\,(\frac{t(t-1)}{2})$ which is the best bound we can get by combining the sub-problems ending by at most t. It is a lower bound on the satisfaction of the demands d_1 to d_t. We set lbBefore$(0) = 0$.

- lbAfter$(t) = wisp_r(\frac{(T-t)(T-t+1)}{2})$ which is the best bound we can get by combining the sub-problems starting at t. It is a lower bound of the cost for satisfying the demands from d_t to d_T. We set lbAfter$(T+1) = 0$.

Figure 2.9 represents the different lower bounds computed while filtering the value i_t for the variable I_t. I_t belongs to the sub-problem $(L_{u,v})$, $lbBefore(u-1)$ and $lbAfter(v+1)$ are computed via DPWisp outside that sub-problem.

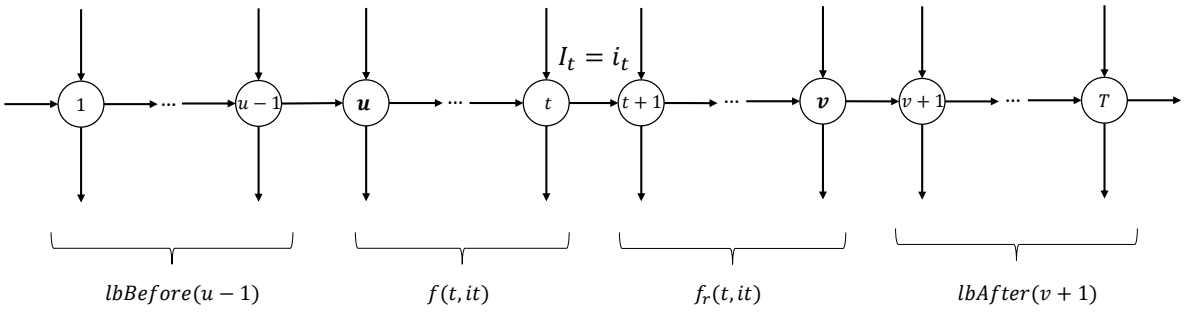

Figure 2.9: Bounds when filtering I_t with the WISP support filtering

We filter the variables via the two following rules:

$$\forall\, t \in [\![u, v]\!], i_t \in D(I_t)$$
$$\text{lbBefore}(u-1) + f(t, i_t) + f_r(t, i_t) + \text{lbAfter}(v+1) > \overline{C} \Rightarrow I_t \neq i_t$$

$$\forall\, t \in [\![u, v]\!], i_{t-1} \in D(I_{t-1}), i_t \in D(I_t)$$
$$\text{lbBefore}(u-1) + f(t-1, i_{t-1}) + cost(t, i_{t-1}, i_t) + f_r(t, i_t) + \text{lbAfter}(v+1) > \overline{C}$$
$$\Rightarrow X_t \neq x_t$$

For all the sub-problems of the support (or solution) of the WISP, we can compute DPLS and its reverse version. Note that here $f(t, i_t)$ and $f_r(t, i_t)$ come from DPLS applied to the sub-problems. Hence when considering sub-problem $(L_{u,v})$ and $u \leq t \leq v$, $f(t, i_t)$ is a lower bound of the cost for satisfying demands d_u to d_t with an inventory level $I_t = i_t$ at the end of period t. Similarly $f_r(t, i_t)$ is a lower bound on the cost for satisfying the demands d_{t+1} to d_v and having $I_t = i_t$. Moreover, since the sub-problems consider only the demands d_u to d_v, we cannot filter values greater than or equal to $\sum_{k=t+1}^{v} d_k$ for I_t and X_t. Indeed, greater values might be used to satisfy demands outside the sub-problem and are not considered when computing DPLS.

Note that although the scaling may affect the quality of the filtering, it is a pragmatic rule applied to avoid wakening costly propagation when little filtering is expected.

2.5.5 Adaptation to take into account the setup cost

In order to adapt the filtering to take into account $\overline{Cs}$, we do not take into account the production and inventory costs. The resulting problem can then be tackled by dynamic programming with DPKnap. The variables can be filtered via the WISP

support with DPKnap computed on the sub-problems. The filtering rules are applied with the upper bound of Cs.

2.6 Numerical results on the single-item lot-sizing problem

This section validates our global constraint and the filtering mechanisms described above. We compare the performance of LOTSIZING to four other methods on the single-item lot-sizing problem.

Five methods The five methods that solve the single-item lot-sizing are:

- A basic CP model (CP_Basic), which is a decomposition of the single-item lot-sizing problem basically equivalent to the MILP model with the implication constraints $X_t > 0 \Rightarrow Y_t = 1, \forall\, t \in [\![1, T]\!]$ instead of the setup constraints (2.3)
- A CP model with our LOTSIZING global constraint (CP_LS)
- The dynamic programming algorithm presented in 2.2.5 (DPLS)
- The classical aggregated MILP model (MILP_AGG)
- The facility location MILP model (MILP_UFL)

The MILP models were implemented with CPLEX version 12.6 and the CP models in Choco 3.3 [72].

Branching heuristics for the CP models A default branching heuristic is used to instantiate the variables to their lower bounds in a lexicographic order (chronological order here). The dominance rule described in 2.5.1 is valid for the single-item lot-sizing problem. For sake of comparison, the same improvement is done for CP_Basic. The search space is thus restricted to the setup variables (Y) for both CP models.

Cost upper bound Since we want to assess the quality of the filtering and LOTSIZING uses cost-based filtering, we choose to have the best possible upper bound on the global cost at the start of the resolution: the optimal cost. This means that our models still has to prove the optimal solution. In a more realistic setting, the model CP_LS can be used to find upper bounds.

Instance parameters The single-item instances are generated based on the parameters d_{avg}, e, δ, θ, λ, and T as follows:

- The inventory costs are constant and equal to 1 (i.e. $h_t = h = 1$).
- The setup and the unitary production costs are generated using two parameters: e and θ.
  - e represents the overall unitary production cost (i.e. the unitary production cost if the production capacity is saturated: $p_t\,\overline{\alpha_t} + s_t$ divided by $\overline{\alpha_t}$). We set $e = 10$.
  - $\theta \in [0, 1]$ represents the portion of the setup cost to the unitary production cost. The overall production cost at t (i.e. $e\,\overline{\alpha_t}$) will be imputable for θ to its setup cost and for $1 - \theta$ to the unitary production cost at t. For each period, θ is uniformly randomly set in the interval $[0, 1]$.
- The demands are uniformly generated in the interval $[d_{\text{avg}} - \delta, d_{\text{avg}} + \delta]$.
- The production and inventory capacities are constant and equal to $\lambda\, d_{\text{avg}}$.

For each problem, we give the set of parameters that were used to generate the instances. Each class of instances contains 10 instances.

Experimentation setup All the tests are run under Windows 8 on an Intel Core i5 @ 2.5 GHz with 12GB of RAM. We set a time limit of 200s and a memory limit of 4GB of RAM. The indicator NODE is the average number of nodes computed by each model on the class. CPU corresponds to the average CPU time used by the models. RNB is the average gap of the root node lower bound to the optimal value. LR is the average gap of the linear relaxation to optimal. Finally OPT is the number of solved instances in the class. The means are computed over all the instances of each class.

2.6.1 Single-item lot-sizing

The five instance classes are defined in table 2.2.

Class	d_{avg}	δ	θ	λ	T
C1LS	1000	100	$[0.8, 1]$	3	40
C2LS	1000	500	$[0.4, 0.6]$	3	40
C3LS	1000	100	$[0.8, 1]$	3	80
C4LS	1000	500	$[0.4, 0.6]$	3	80
C5LS	1000	50	0.5	3	40

Table 2.2: Single-item instance classes

The results of the five models on these instances are presented in tables 2.3 and 2.4.

	CP_Basic				CP_LS				DPLS	
Class	NODE	CPU	RNB	OPT	NODE	CPU	RNB	OPT	CPU	OPT
C1LS	8.3E+06	200	100%	0	**1**	1.1	**0**%	10	0.2	10
C2LS	1.7E+06	200	65%	0	**1**	1.0	**0**%	10	**0.2**	10
C3LS	6.5E+06	200	100%	0	**28**	2.1	**0**%	10	**0.4**	10
C4LS	3.4E+05	200	64%	0	**35**	2.2	**0**%	10	**0.4**	10
C5LS	8.5E+06	200	56%	0	**1**	1.1	**0**%	10	**0.2**	10

Table 2.3: Single-item lot-sizing - CP and DPLS

	MILP_AGG					MILP_UFL				
Class	NODE	CPU	RNB	LR	OPT	NODE	CPU	RNB	LR	OPT
C1LS	580	**0.1**	1%	10%	10	460	0.6	1%	**3**%	10
C2LS	1360	**0.2**	2%	10%	10	1643	1.4	2%	**3**%	10
C3LS	3213	1.7	2%	11%	10	13109	54.1	3%	**3**%	10
C4LS	2222	1.6	1%	10%	10	14366	61.6	2%	**2**%	10
C5LS	1691	0.3	2%	11%	10	9336	4.5	2%	**3**%	10

Table 2.4: Single-item lot-sizing - MILP

These tables show that:

- As expected, the basic CP model has a very large search space as it does not propagate any strong reasoning. We therefore did not use CP_Basic for the following results.
- As the upper bound provided is optimal and there is no upper bound on *Cp*,*Ch* and *Cs*, CP_LS achieves AC at the root node and branches backtrack-free towards an optimal solution.

- The linear relaxation of MILP_AGG is not as good as the MILP_UFL's as it was expected due to the setup constraints (2.3). CPLEX however provides a better root node lower bound for MILP_AGG. MILP_UFL is not as competitive as MILP_AGG because of the number of variables and constraints. We therefore did not use MILP_UFL for the following results.

2.6.2 Scaling the global constraint

We then test the WISP support filtering described in 2.5.4 when the DP has memory issues. In order to generate memory issues for DPLS, we add high consumption peaks in the instances. The peaks are added in periods 6 to 9, 12 to 15, 22 to 25 and 32 to 36 and correspond to demands of 50,000. The five instance classes have the parameters defined in table 2.5.

Class	d_{avg}	δ	θ	λ	T
C1Peaks	100	50	$[0.8, 1]$	4	40
C2Peaks	100	50	$[0.4, 0.6]$	4	40
C3Peaks	100	50	0.5	4	40
C4Peaks	100	20	$[0.8, 1]$	4	40
C5Peaks	100	20	$[0.4, 0.6]$	4	40

Table 2.5: Instance classes for the scaling of LOTSIZING

The branching heuristic is adapted to select first the setup variables of the high demand periods. Table 2.6 compares the two models CP_LS, MILP_AGG and the algorithm DPLS on these big instances.

	CP_LS				MILP_AGG					DPLS	
Class	NODE	CPU	RNB	OPT	NODE	CPU	RNB	LR	OPT	CPU	OPT
C1Peaks	125	1.1	2%	10	**0**	**0.0**	**0**%	13%	10	23.8	10
C2Peaks	468	5.7	2%	10	**0**	**0.0**	**0**%	6%	10	23.4	10
C3Peaks	2784	21.0	3%	10	**1**	**0.0**	**0**%	7%	10	22.6	10
C4Peaks	408	3.7	3%	10	**0**	**0.0**	**0**%	15%	10	23.4	10
C5Peaks	446	5.5	2%	10	**0**	**0.0**	**0**%	7%	10	24.0	10

Table 2.6: Scaling the global constraint

This table shows that the filtering of LOTSIZING is lighter, hence the root node lower bound gap increases as well as the number of nodes. The resolution using CP is however faster than DPLS even though some branching is now required. Although the linear relaxation degrades, CPLEX pre-processing behaves very well as shown by the root node lower bound.

2.7 Single-item lot-sizing with side constraints

We consider the single-item lot-sizing problem (L) with three side constraints (domain disjunction, limited production rate and a combination of the two). The instances created here are generated the same way as before and we added leveled production and/or constrained production rate. For the following tests, we compared only CP_LS to MILP_AGG and to DPLS when it is relevant.

2.7.1 Disjunctive production constraints

We consider here that the production is leveled, *i.e.* some production quantities are forbidden, which can happen with specific industrial constraints. The domains of each variable X_t is defined by a disjunction of n_t integer intervals $K_k = [\![\underline{K_k}, \overline{K_k}]\!]$, $\forall k \in [\![1, n_t]\!]$:

$$D(X_t) = \{0\} \cup K_1 \cup \ldots \cup K_{n_t}$$

DPLS can take into account the disjunctions without any loss of complexity. We add the following constraints to the MILP model MILP_AGG:

$$X_t = \sum_{k=1}^{n_t} X_t^k \qquad \forall\, t = 1 \ldots T \tag{2.22}$$

$$Y_t = \sum_{k=1}^{n_t} Y_t^k \qquad \forall\, t = 1 \ldots T \tag{2.23}$$

$$X_t^k \leq Y_t^k \overline{K_k} \qquad \forall\, t = 1 \ldots T, k = 1 \ldots n_t \tag{2.24}$$

$$Y_t^k \underline{K_k} \leq X_t^k \qquad \forall\, t = 1 \ldots T, k = 1 \ldots n_t \tag{2.25}$$

The five first classes for this problem are defined in table 2.7.

Class	d_{avg}	δ	θ	λ	T
C1Disj	100	50	$[0.8, 1]$	5	40
C2Disj	100	60	$[0.4, 0.6]$	5	40
C3Disj	100	70	$[0.3, 0.8]$	5	40
C4Disj	100	30	$[0.6, 1]$	5	40
C5Disj	100	50	$[0.9, 1]$	5	40

Table 2.7: Instance classes for lot-sizing with disjunctive constraints

We generated C6Disj, C7Disj, C8Disj, C9Disj and C10Disj that have the same parameters than the five instances above, but with $T = 80$. The disjunctions are added

as follows: $D(X_t) = [\![0, 30]\!] \cup [\![100, 150]\!] \cup [\![200, 240]\!]$. Table 2.8 gives the numerical results for the single-item lot-sizing with disjunctions.

	CP_LS				MILP_AGG					DPLS	
Class	NODE	CPU	RNB	OPT	NODE	CPU	RNB	LR	OPT	CPU	OPT
C1Disj	**1**	**0.0**	**0**%	10	6.4E+05	162.1	43%	52%	2	**0.0**	10
C2Disj	**2**	**0.0**	**0**%	10	1.4E+04	6.0	27%	38%	10	**0.0**	10
C3Disj	**2**	**0.0**	**0**%	10	1.6E+03	0.5	27%	38%	10	**0.0**	10
C4Disj	**2**	**0.0**	**0**%	10	2.2E+04	6.3	39%	48%	10	**0.0**	10
C5Disj	**1**	**0.0**	**0**%	10	9.6E+05	200.0	52%	61%	0	**0.0**	10
C6Disj	**2**	0.1	**0**%	10	3.2E+05	200.0	43%	52%	0	**0.0**	10
C7Disj	**2**	0.1	**0**%	10	5.0E+04	38.3	27%	39%	10	**0.0**	10
C8Disj	**1**	0.1	**0**%	10	5.7E+03	4.5	28%	39%	10	**0.0**	10
C9Disj	**2**	0.1	**0**%	10	3.6E+04	21.9	38%	48%	10	**0.0**	10
C10Disj	**2**	0.1	**0**%	10	3.3E+05	200.0	52%	60%	0	**0.0**	10

Table 2.8: Single-item lot-sizing with disjunctions

Table 2.8 shows that the CP model and DPLS are very fast to solve these instances. The dominance rule described in 2.5.1 of the setup variables is not valid for this problem: indeed the flow with disjunctions is not polynomial. However as LOTSIZING's filtering uses DPLS, the global constraint can handle disjunctions on the domains of the production variables. Therefore CP_LS achieves AC at the root node and branches backtrack free towards an optimal solution. Unsurprisingly we note that the MILP model does not handle these disjunction constraints well.

2.7.2 Q/R constraints

Q/R constraints are interesting side constraints for single-item lot-sizing problems [48, 49]. They relate to the production rate and state that, given two integers Q and R, there must be at least Q and at most R periods between two consecutive productions. Dynamic programming rapidly gets memory issues here, as the states should take into account what happened at least R periods before. The Q/R constraints can be modeled by two SEQUENCE constraints stated as follows:

$$\text{SEQUENCE}(0, 1, Q+1, [Y_1, \ldots, Y_T], \{1\})$$
$$\text{SEQUENCE}(1, R+1, R+1, [Y_1, \ldots, Y_T], \{1\})$$

The SEQUENCE constraint is defined as follows [15]: $\text{SEQUENCE}(l, u, k, [Z_1, \ldots, Z_n], v)$ holds if and only if:

$$\forall\, 1 \leq i \leq n-k+1 \quad l \leq |\{i \mid Z_i \in v\}| \leq u$$

We add the following constraints to the model MILP_AGG:

$$\sum_{t=u}^{v} Y_t \leq 1 \qquad \forall\, u, v \in [\![1, T]\!] \text{ s.t. } v - u + 1 = Q + 1 \tag{2.26}$$

$$\sum_{t=u}^{v} Y_t \geq 1 \qquad \forall\, u, v \in [\![1, T]\!] \text{ s.t. } v - u + 1 = R + 1 \tag{2.27}$$

We add the $\#_{uv}$ variables that count the number of effective production periods between period u and period v included:

$$\#_{uv} = \sum_{t=u}^{v} Y_t \qquad \forall\, u, v \in [\![1, T]\!] \tag{2.28}$$

These variables enable us to use the encoding of SEQUENCE presented in [29] to propagate the Q/R constraints. We also add the useful following redundant constraints:

$$\#_{1t} + \#_{t+1T} = \#_{1T} \qquad \forall\, t \in [\![2, T-1]\!] \tag{2.29}$$

$$\#_{1t} + Y_{t+1} = \#_{1t+1} \qquad \forall\, t \in [\![2, T-1]\!] \tag{2.30}$$

The ten classes (C1QR, ..., C10QR) for this problem have the same parameters than C1Disj, ..., C10Disj to which we add (Q=2, R=6) for classes 1, 2, 3, 6, 7, 8 and (Q=3, R=7) for classes 4, 5, 9, 10. Table 2.9 compares CP_LS to MILP_AGG on the instances with Q/R.

	CP_LS				MILP_AGG				
Class	NODE	CPU	RNB	OPT	NODE	CPU	RNB	LR	OPT
C1QR	**2**	0.3	**0**%	10	17	0.0	1%	14%	10
C2QR	**22**	0.2	**1**%	10	56	0.1	**1**%	13%	10
C3QR	84	0.4	**1**%	10	**27**	0.0	**1**%	12%	10
C4QR	**1**	0.3	**0**%	10	20	0.0	1%	12%	10
C5QR	**1**	0.4	**0**%	10	13	0.0	1%	16%	10
C6QR	**772**	7.2	**1**%	10	943	0.5	2%	16%	10
C7QR	6488	42.8	**0**%	10	**601**	0.4	1%	14%	10
C8QR	26716	134.1	**1**%	5	**392**	0.3	**1**%	13%	10
C9QR	**1**	1.1	**0**%	10	510	0.3	1%	15%	10
C10QR	**21**	1.9	**0**%	10	1175	0.5	3%	18%	10

Table 2.9: Single-item lot-sizing with Q/R

The linear relaxation and root node lower bound of MILP_AGG has slightly worsened without degrading the performance of the model. CP_LS stays competitive on most of the instances.

2.7.3 Disjunctive and Q/R constraints

We add both Q/R and disjunctive production constraints. The problem cannot be tackled via DP due to the Q/R constraints, hence we compared CP_LS to MILP_AGG. The instances have the same parameters as before with both the disjunctions and the Q/R parameters presented for the latter problems. The results are shown in table 2.10.

	CP_LS				MILP_AGG				
Class	NODE	CPU	RNB	OPT	NODE	CPU	RNB	LR	OPT
C1DijsQR	**1**	**0.3**	**0**%	10	1808	0.8	7%	19%	10
C2DijsQR	**6**	**0.2**	**0**%	10	637	0.4	2%	14%	10
C3DijsQR	**67**	**0.4**	**1**%	10	710	**0.4**	2%	13%	10
C4DijsQR	**3**	0.4	**0**%	10	964	**0.3**	5%	16%	10
C5DijsQR	**30**	0.5	**1**%	10	209	**0.2**	12%	25%	10
C6DijsQR	**669**	**4.7**	**0**%	10	40150	53.4	5%	19%	10
C7DijsQR	**3471**	17.8	**0**%	10	4839	**9.2**	2%	15%	10
C8DijsQR	22386	94.0	**1**%	7	**4618**	**7.4**	3%	14%	10
C9DijsQR	**53**	**1.5**	**0**%	10	5066	8.8	4%	17%	10
C10DijsQR	**7**	**2.6**	**1**%	10	1663	1.7	11%	26%	10

Table 2.10: Single-item lot-sizing with disjunctives and Q/R

On some classes, CP_LS does not solve all the instances yet is often competitive compared to MILP_AGG and has a near optimal root node lower bound.

2.8 Conclusion

In this chapter, we defined a global constraint LOTSIZING for a capacitated single-item lot-sizing problem. Firstly, we presented a new lower bound for this problem, based on a new decomposition of the problem into sub-problems. Secondly, we formally introduced our constraint and gave some complexity results. Thirdly, we developed filtering rules for the LOTSIZING global constraint based on dynamic programming. Finally, we presented a proof of concept for the filtering of the constraint via several numerical results. We can conclude that our approach based on constraint programming can yield interesting and competitive results for lot-sizing problems with side constraints.

The next step of this work will be to use the LOTSIZING global constraint as a building block to tackle multi-item and multi-echelon problems.

2.9 Practical use of LOTSIZING and tuning the consistency level

To consider a more practical perspective, we investigate different ways to parametrize the global constraint and achieve different consistency levels. The DP algorithms used to filter the LOTSIZING constraint can be very costly to solve depending on the size of the instance. Note that we have seen that DPLS does not have to be solved on the global problem but can be solved only on some of the smaller sub-problems when the instance is too big (see section 2.5.4). This is a way of weakening the filtering – we filter less than with DPLS applied on the whole problem but the algorithm scales better. The first mode – or setting – of LOTSIZING has been defined in section 2.5 and its speed is bound to the speed of the DP-based algorithms.

We define a new mode for LOTSIZING – a "light" mode – where we totally deactivate the use of DP algorithms. The level of filtering of LOTSIZING is lowered thus improving the propagation speed. In this mode, we compute global lower bounds and use two filtering algorithms without DP. We first compute the linear relaxation of the global problem with a flow algorithm to obtain a lower bound on the global cost. We also compute lower bounds on the intermediate costs Cp, Ch and Cs with a linear relaxation ignoring the other costs. In order to filter the decision variables, we use reduced-cost filtering. The principle of the method will be explained in the next chapter. Reduced-cost filtering is not complete (it does not remove all inconsistent values) as we will see that the filtered values depend on the dual solution. Different dual solutions can derive different filtering. The second way of filtering is a resource-based filtering similar to the filtering performed with DPLS in section 2.5.3. For each period $2 \leq t \leq T-1$ we compute the linear relaxation of sub-problems $L_{1,t-1}$ and $L_{t+1,T}$ and use them as partial lower bounds to filter the decision variables at t. Note that these two ways of filtering are not comparable.

We will see in the next chapter the application and test the performance of the "light" mode of LOTSIZING.

Chapter 3

Multi-item lot-sizing with shared setup costs

3.1 Introduction

In this chapter we investigate the behaviour of the LOTSIZING global constraint on single-level multi-item lot-sizing problems. The case study problem considered in this chapter is the capacitated multi-item lot-sizing problem with shared setup costs [93]. We chose a problem with only major setup costs – as opposed to minor setup costs where each item has its own setup cost – because its the simplest extension of single-item lot-sizing into a multi-item problem. It is a pure generic multi-item lot-sizing problem and is usually solved using MILP that is very effective for industrial size instances.

Multi-item lot-sizing problems are a very natural extension to single-item where the goal is to optimize the production and storage of a set of items. Figure 3.1 shows that at each period, production of several items can occur and when the corresponding demand is fully satisfied, the overload of each item is stored until the next period. The variables are double indexed and the first index corresponds to the item, the second to the period. $X_{n,t}$ is therefore the quantity of item n produced at period t.

The optimization usually consists in a trade-off between the different items: the decision of which item is more worthy to store or produce adds to the difficulty of deciding which periods to open. The items can be linked either by a shared production or storage capacity, by shared setup costs or by many other constraints. Multi-item problems can also use the lot-sizing extensions that are presented in the introduction of this thesis (backlog, lost sales, ...). These constraints model the fact that the items may require the same resources (budget, machines, workers, ...) or

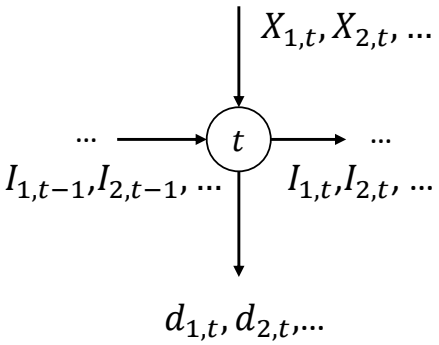

Figure 3.1: Multi-item lot-sizing problems

that decisions have to be made considering the items altogether. Multi-item problems are more representative and more realistic models of industrial real life applications since factories typically deal with the optimization of production plans of several kinds of products that need the same resources. LOTSIZING allows for an intuitive modeling of multi-item problems where each item is represented by only one LOTSIZING with additional constraints linking them.

Multi-item problems faster pose scalability issues and it can be even harder to tackle additional side constraints. Even though reformulations and valid inequalities derived from the single-item lot-sizing exist and help tackle multi-item problems [102], generic approaches are uncommon or complicated to implement [36]. Polyhedral results are limited but there are some exceptions [55, 67]. People tend nowadays to use decomposition or hybrid techniques for their particular multi-item problems with side constraints [1, 2, 27, 37]. Hence the motivation to build a generic CP approach for complex lot-sizing problems.

The goal of the chapter is to test and improve the reasoning mechanisms of LOTSIZING developed in the first chapter on this pure multi-item problem in order to show that the approach can be used on complex lot-sizing problems with side constraints. We first describe the problem and the classical MILP model then give the CP model based on our global constraint and the use of a multi-flow problem. We show that, as one might expect, the decomposed CP model is totally outperformed but also that LOTSIZING helps solving the problem in reasonable time. We then try to improve the filtering by introducing cardinality variables and extend the scope of LOTSIZING with piece-wise linear production and holding costs.

3.2 Description and models

The capacitated multi-item lot-sizing problem with shared setup costs aims at planning the production of N types of items indexed by $n \in [\![1, N]\!]$ on a single resource, with a global production capacity λ for each period t, regardless of the item. The variables introduced in the single-item lot-sizing are now indexed by n. Each item n is subject to a demand $d_{n,t}$ for $t \in [\![1, T]\!]$. A setup cost s_t is paid if at least one unit of any item is produced at period t. An unitary production cost $p_{n,t}$ is paid for the production of each unit of item n at period t. Finally, storing one unit of item n between periods t and $t+1$ induces a holding cost $h_{n,t}$. The set of data of each item is denoted by $data_n$. We give three models to solve the problem.

A MILP model. An aggregated MILP model can easily be derived from the classical MILP model for the single-item problem:

$$min\ z = \sum_{t=1}^{T} (s_t Y_t + \sum_{n=1}^{N} (p_{n,t} X_{n,t} + h_{n,t} I_{n,t})) \tag{3.1}$$

$$\sum_{n=1}^{N} X_{n,t} \leq \lambda \qquad \forall\, t = 1 \dots T \tag{3.2}$$

(MILP)

$$X_{n,t} + I_{n,t-1} = I_{n,t} + d_{n,t} \qquad \forall\, n = 1, \dots, N, \forall\, t = 1, \dots T \tag{3.3}$$

$$X_{n,t} \leq M_{n,t} Y_t \qquad \forall\, n = 1, \dots, N, \forall\, t = 1, \dots T \tag{3.4}$$

$$X_{n,t} \in \{\underline{\alpha_{n,t}}, \dots, \overline{\alpha_{n,t}}\} \qquad \forall\, n = 1, \dots, N, \forall\, t = 1 \dots T \tag{3.5}$$

$$I_{n,t} \in \{\underline{\beta_{n,t}}, \dots, \overline{\beta_{n,t}}\} \qquad \forall\, n = 1, \dots, N, \forall\, t = 1 \dots T \tag{3.6}$$

$$Y_t \in \{0, 1\} \qquad \forall\, t = 1 \dots T \tag{3.7}$$

where $M_{n,t}$ is an upper bound on $X_{n,t}$ such as $\overline{\alpha_{n,t}}$. Constraints (3.2) are the overall capacity constraints for all the periods. Constraints (3.3) are the flow balance constraints and constraints (3.4) are the setup activation constraints.

A decomposed CP model. We define a naive decomposed CP model – called DEC – which basically is the MILP model with the implications $X_{n,t} > 0 \Rightarrow Y_t = 1, \forall\, t \in [\![1, T]\!], \forall n \in [\![1, N]\!]$ instead of constraints (3.4).

A CP model using the LOTSIZING global constraint. As said earlier, the LOTSIZING constraint intuitively models each item. We have to add the capacity side constraints outside of these global constraints. The CP model based on LOTSIZING writes:

$$min\ z = Cs + \sum_{n=1}^{N} (Cp_n + Ch_n) \tag{3.8}$$

$$\sum_{n=1}^{N} X_{n,t} \leq \lambda \qquad \forall\, t = 1 \dots T \tag{3.9}$$

$$\text{LOTSIZING}(X_n, I_n, Y, Cp_n, Ch_n, Cs, C_n, data_n) \qquad \forall\, n = 1 \dots N \tag{3.10}$$

$$X_n, I_n \in \mathbb{N}^T \qquad \forall\, n = 1 \dots N \tag{3.11}$$

$$Y \in \mathbb{N}^T \tag{3.12}$$

$$Cs \in \mathbb{N} \tag{3.13}$$

$$Cp_n, Ch_n, C_n \in \mathbb{N}^N \tag{3.14}$$

Constraints (3.8) to (3.14) model the capacitated multi-item lot-sizing problem with shared setup costs. Constraints (3.9) are the overall capacity constraints for all the periods. Each item is represented by a LOTSIZING constraint (3.10). Note that the setup variables Y are common to each item hence Cs is the global setup cost.

3.3 Instances and experimental setup

The rest of the chapter aims at comparing different models based on LOTSIZING to solve the capacitated multi-item lot-sizing problem with shared setup costs. We first describe the instances that are used and the experimental setup.

Instances. To compare our models, we adapted the instances for the capacitated multi-item lot-sizing problem with setup times from [93]. The authors introduce a benchmark of 540 instances that are separated in three classes of 180 instances. Each class considers a time horizon of 20 periods and has a fixed number of items: 10, 20, or 30. We choose to keep the number of instances and increase the number of classes. We therefore generated 12 classes of 45 instances spread as explained in table 3.1.

The same way as it was done in [93], we generate demands uniformly between 75 and 125 for the first half of the class (23 instances out of 45 for each class). For the instances of the second half (from 24 to 45), demands are generated between 0

T	20			40			60			80		
N	10	20	30	10	20	30	10	20	30	10	20	30

Table 3.1: Classes of instances

and 200. Similarly as in the paper, to simulate an increasing trend, we randomly replaces 25% of the demands in the first four periods of each item by 0.

The capacity is the same as in the instances of [93].

In [93], the cost baseline is $p_{n,t} = 1$ and $h_{n,t} = h_n$ uniformly generated between 0.8 and 1.2. However we intend to have integer costs so our baseline is $p_{n,t} = 10$ and we generate $h_{n,t} = h_n$ uniformly between 8 and 12.

In [93], there are minor setup costs $s_{n,t} = s_n$. To generate our setup costs, we compute the minimum and maximum ratios $R_{N,min} = \min_{\text{instances with } N \text{ items}} \{\frac{s_n}{h_n}\}$ and $R_{N,max} = \max_{\text{instances with } N \text{ items}} \{\frac{s_n}{h_n}\}$. Then we uniformly generate a ratio $R_{N,n}$ between $R_{N,min}$ and $R_{N,max}$ for each item. For our instances we compute the setup cost: $s_{n,t} = s = \sum_{n=1}^{N} R_{N,n} h_n$.

Note that in these instances the production cost is constant among the items since $p_{n,t} = 10$. The optimization, as usually done in academic problems, is a trade-off between setup and inventory costs.

Setup. All the tests are run under Windows 8 on an Intel Core i5 @ 2.5 GHz with 12GB of RAM. We set a time limit of 300s and a memory limit of 8GB of RAM. The indicator NODE is the average number of nodes computed by each model on the given set of instances. CPU corresponds to the average CPU time used by the models. RNB is the average gap of the root node lower bound to the optimal value. LR is the average gap of the linear relaxation to optimal. Finally OPT is the number of solved instances in the set. The means are computed over all the instances of each set. As in the previous chapter, since we want to prove optimality, we give the optimal cost as an upper bound to the CP models.

The MILP models were implemented with CPLEX version 12.6 and the CP models in Choco 3.3 [72].

Numerical results are presented in the next four sections. The first section describes how to branch only on the setup variables. The second section compares different models of the problem. In the third section, we extend the LOTSIZING constraint by adding cardinality variables that count the number of open periods. The last section generalizes LOTSIZING further by allowing the production and inventory costs to be modeled with step functions.

3.4 Differences with the single-item

Solving the multi-item problem presents some changes with the resolution of the single-item. We will first see that we need to check if branching can still be performed on the setup variables alone. Then we will show that the basic model based on LOTSIZING has to be improved with a redundant constraint in order to increase the communication between the items. In this section we show and compare several other CP models to distinguish the main issues to focus on when solving multi-item problems.

3.4.1 Branching only on setup variables: the use of a multi-flow problem

The dominance rule described in section 2.5.1 does not apply in the case of the multi-item problem. Indeed once the setup variables are fixed, a single LOTSIZING constraint can not instantiate all its production and inventory variables (X_n and I_n) variables since they are not independent from the other items. However once the Y variables are fixed, the resulting problem is a multi-flow problem [92] in the flow graph presented in figure 2.1: each item can flow on each arc with a cost ($p_{n,t}$ if it is a production arc, $h_{n,t}$ if it is an inventory arc or 0 if it is an arc from a period node to the sink). The overall capacity of the production arcs – for a period t – is limited by λ. This problem is NP-hard in the general case. When solving multi-item lot-sizing problems, if the items are no longer linked when the setup variables Y are instantiated, the resulting multi-flow problem becomes polynomial as it boils down to several independent minimum cost maximum flow problems. Here the global production capacity λ still links the items. However as our graph is very particular, the constraint matrix of this multi-flow problem appears to be totally unimodular. The constraints are:

$$\sum_{n=1}^{N} X_{n,t} \leq \lambda \qquad \forall\, t = 1 \dots T \tag{3.15}$$

$$X_{n,t} + I_{n,t-1} = I_{n,t} + d_{n,t} \qquad \forall\, n = 1, \dots, N, \forall\, t = 1, \dots T \tag{3.16}$$

$$X_n, I_n \in \mathbb{N}^T \qquad \forall\, n = 1 \dots N \tag{3.17}$$

$$\tag{3.18}$$

Constraints (3.15) are the capacity constraints – let us call them *capa(t)* – and (3.16) are the flow balance constraints – let us call them *flow(n, t)*. Table 3.2 summarizes the coefficients of the production and inventory variables in the constraints at period t.

Variable	Coefficient	Constraint
$I_{n,t}$	$+1$	*flow*$(n, t+1)$
	-1	*flow*(n, t)
	0	Otherwise
$X_{n,t}$	$+1$	*flow*(n, t)
	$+1$	*capa*(t)
	0	Otherwise

Table 3.2: Coefficients of the variables in the multi-flow matrix

We define $L_1 = \{flow(n,t),\ \forall\, t \in [\![1,T]\!], n \in [\![1,N]\!]\}$ and $L_2 = \{capa(t),\ \forall\, t \in [\![1,T]\!]\}$. L_1 and L_2 partition the lines of the multi-flow matrix and when adding the coefficients on each column of L_1 and of L_2, we get the same vector which means the matrix is totally unimodular [101]. Our resulting multi-flow problem is therefore polynomial.

Once the setup variables (Y) are all fixed, a LP solver can be called to solve the resulting multi-flow problem and instantiate the production and inventory quantities. We create a global constraint that solves the multi-flow when the setup variables are fixed. More specifically, we embed the multi-flow problem resolution in a propagator – independent from LOTSIZING global constraint – that activates only when the setup variables are all fixed. This propagator is very generic and can be stated by the user.

3.4.2 A redundant LOTSIZING

The first CP model presented in section 3.2 considers the items separately since one item is modeled by one LOTSIZING constraint. These items are linked by the global capacity and the setup costs. In order to better link the items and reason on the problem as a whole, we add a redundant LOTSIZING that aggregates the production and inventory variables. Since LOTSIZING cannot make any difference between the items, the costs need to be revised. For this redundant constraint, we set the unitary production (respectively holding) cost to the minimum of the unitary production (respectively holding) costs of each item. The redundant constraint is therefore a relaxation of the overall problem and the variables Cp and Ch are lower bounds of the global production and holding costs, and C is a lower bound of the total cost z. The goal of this constraint is to help better enforce feasibility and estimate the different costs, in particular the global setup cost, by considering all the items together.

$$\text{LOTSIZING}(X_{agg}, I_{agg}, Y, Cp, Ch, Cs, C, data_{agg}) \tag{3.19}$$

$$X_{agg,t} = \sum_{n=1}^{N} X_{n,t} \qquad \forall\, t = 1 \dots T \tag{3.20}$$

$$I_{agg,t} = \sum_{n=1}^{N} I_{n,t} \qquad \forall\, t = 1 \dots T \tag{3.21}$$

$$X_{agg}, I_{agg} \in \mathbb{N}^T \tag{3.22}$$

$$Cp, Ch, C \in \mathbb{N} \tag{3.23}$$

Constraints (3.8) to (3.14) and (3.19) to (3.23) effectively constitute a model of the problem. Constraint (3.19) is the redundant LOTSIZING constraint and constraints (3.20) and (3.21) link the aggregate variables to the real production and inventory variables of the items. The aggregate data given to the redundant LOTSIZING is

$$\{\min_{n\in[\![1,N]\!]} p_{n,t}, \min_{n\in[\![1,N]\!]} h_{n,t}, s_t, \sum_{n=1}^{N} d_{n,t}, \sum_{n=1}^{N} \underline{\alpha_{n,t}}, \min\{\lambda, \sum_{n=1}^{N} \overline{\alpha_{n,t}}\}, \sum_{n=1}^{N} \underline{\beta_{n,t}}, \sum_{n=1}^{N} \overline{\beta_{n,t}}) \mid t \in [\![1,T]\!]\}$$

Figure 3.2 is a small example of the problem with 2 periods and 2 items to demonstrate some extra filtering performed by the redundant constraint. There are no unitary production or holding costs. Note that since the capacity is $\lambda = 2$ we have to open both periods to satisfy the demands. However no LOTSIZING on the items can force a period to open since they are locally consistent. Figure 3.3 shows the view of the redundant LOTSIZING and we can directly see that both periods have to be open when the two items are considered together.

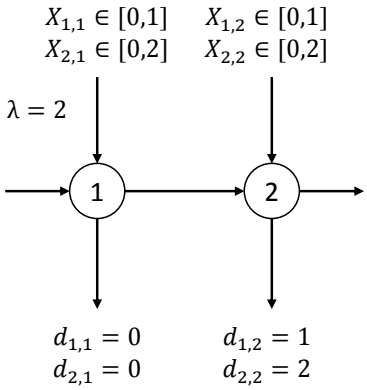

Figure 3.2: A small example of multi-item

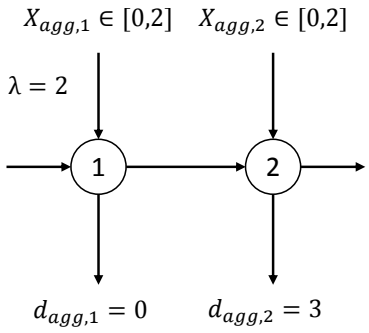

Figure 3.3: The view of the redundant LOTSIZING on the example

3.4.3 First numerical tests

In order to set the baseline for our numerical tests, we first compare three models:

- The model MILP.
- The decomposed model DEC.
- A LOTSIZING based model called CPLS. It is composed of constraints (3.8) to (3.14) and (3.19) to (3.23). The solver branches on the setup variables and the multi-flow is solved to set the quantity variables. In this model the redundant constraint uses LOTSIZING to filter the variables and the LOTSIZING constraints for each item are on "light" mode as explained at the end of chapter 2.

Tables 3.3 and 3.4 show the results for the three models. We choose the compare our models on the 40 first instances. More results are shown in the end of the section.

Models	NODE	CPU	RNB	LR	OPT
MILP	0.2	0.0	0.0 %	26.0%	20
CPLS	27.5	1.8	5.9 %	NA	20
DEC	1086747.9	300.0	95.8 %	NA	0

Table 3.3: Baseline (first 20 instances)

The first observation is that MILP performs very well on these instances for the problem. We can see that the naive decomposed model DEC cannot even find a solution, which was expected. CPLS is not very competitive with the MILP but can at least solve all the instances in a reasonable time. Unsurprisingly CPLS has a better root node lower bound than the linear relaxation. MILP has however a near optimal root node lower bound.

Models	NODE	CPU	RNB	LR	OPT
MILP	35.2	0.1	1.6 %	35.4%	20
CPLS	157.8	9.5	7.3 %	NA	20
DEC	1745997.0	300.0	95.4 %	NA	0

Table 3.4: Baseline (instances 21 to 40)

3.4.4 On the necessity to branch on setup variables

In order to assess the relevance of the use of the multi-flow problem as a way to avoid branching on the production variables we compare CPLS to CPLS without the multi-flow. $\text{CPLS}_{\setminus \text{MF}}$ has the same constraints than CPLS but branches first on the setup variables then on the production variables X. Note that when the production variables are fixed, the inventory variables are fixed as well. The results are presented in tables 3.5 and 3.6.

Models	NODE	CPU	RNB	LR	OPT
CPLS	27,5	1,8	5,9 %	NA	20
$\text{CPLS}_{\setminus \text{MF}}$	15132,2	16,6	5,9 %	NA	20

Table 3.5: With or without a multi-flow? (first 20 instances)

Models	NODE	CPU	RNB	LR	OPT
CPLS	157,8	10,5	7,3 %	NA	20
$\text{CPLS}_{\setminus \text{MF}}$	193351,9	211,1	7,3 %	NA	7

Table 3.6: With or without a multi-flow? (instances 21 to 40)

As expected, branching on the quantity variables (X or I) is too costly and is not a realistic solution. The multi-flow problem tackles this issue and is essential to reduce the search space. We also tried to solve the linear relaxation of the problem at each node to compute a global lower bound that could improve the resolution at small cost. We have however seen in section 3.4.3 that the MILP gives a poor linear relaxation. We noticed that the search time is the same as for CPLS yet the gain is negligible when computing the linear relaxation at each node: the lower bound computed by the redundant LOTSIZING is usually better.

3.4.5 Different levels of filtering

Since the LOTSIZING global constraint can perform more or less filtering, we want to investigate the trade-off between the level of filtering and the resolution time.

We define two variations of CPLS that have different strengths and the results are presented in tables 3.7 and 3.8.

- The model called BASIC is composed of constraints 3.8 to 3.14 and solves the multi-flow when all the setup variables are instantiated. Note that this is basically CPLS without the redundant constraint.
- The model called HEAVY is CPLS where all the LOTSIZING constraints on the items also perform filtering using DPLS. It considerably slows down the propagation at each node as each item performs a lot of reasoning. As CPLS, HEAVY solves the multi-flow when all the setup variables are instantiated.

Models	NODE	CPU	RNB	LR	OPT
CPLS	27.5	1.8	5.9 %	NA	20
HEAVY	27.5	9.0	5.9 %	NA	20
BASIC	301.4	1.6	6.5 %	NA	20

Table 3.7: Benefits of the redundant LOTSIZING (first 20 instances)

Models	NODE	CPU	RNB	LR	OPT
CPLS	157.8	9.5	7.3 %	NA	20
HEAVY	157.8	40.5	7.3 %	NA	20
BASIC	2057.6	10.4	8.4 %	NA	20

Table 3.8: Benefits of the redundant LOTSIZING (instances 21 to 40)

CPLS and BASIC are similar in time on those instances even though CPLS has a better root node lower bound. BASIC does many more nodes than CPLS and we think the model would not scale on bigger instances. HEAVY solves at least one DP for each item at each node, which explains the big resolution time and we can see no improvement in the search compared to CPLS. We believe that the best way to configure the model is CPLS: the redundant constraint that captures most of the problem performs the heavier computation whereas the LOTSIZING constraints dedicated to the item are on "light" mode. We also implemented an extension of DEC where we only add the redundant LOTSIZING constraint and noticed that the resolution time and branching space are very similar to CPLS.

3.4.6 Results on the benchmark

To better assess the quality of CPLS, we compare CPLS to MILP on the first three classes of the benchmark. These are the classes with $T = 20$ and $N \in \{10, 20, 30\}$

with 45 instances each. We did not solve the instances with bigger time-horizons. The results are presented in table 3.9.

Class	Models	NODE	CPU	RNB	LR	OPT
1	MILP	27,7	0,1	1,1 %	32,6%	45
	CPLS	127,5	9,5	7,0 %	NA	45
2	MILP	414,2	0,7	4,5 %	63,0%	45
	CPLS	620,1	180,6	6,6 %	NA	43
3	MILP	71,1	0,4	1,5 %	33,4%	45
	CPLS	122,7	68,5	7,2 %	NA	40

Table 3.9: Results on the first classes of the benchmark

We can see that the number of nodes is not very big compared to the resolution time as the filtering via DP can be very costly. A few instances are not proven optimal by CPLS in the given time limit.

3.5 Reasoning on the cardinalities

Although the redundant LotSizing constraint of CPLS filters more than all other constraints, it is a relaxation of the whole problem. It is known that adding redundant variables and constraints can significantly reduce the search space and increase the propagation of the other constraints by expressing certain properties of the solutions.

The role of redundancy in CP is that it can provide a more global view of the problem. To increase the communication between the constraints, one can reformulate them or link existing constraints and variables via redundancy. By definition, the use of redundant constraints and variables does not change the optimal solution(s) or the feasible region, it may nevertheless imply extra computational effort.

We extend the scope of LotSizing with redundant variables to better inform the other constraints and improve their filtering. Note that LotSizing is not extended to multi-item but the reasoning mechanisms on the single-item problem are reinforced. The structure of the model inspired us to use cardinality variables. As the branching occurs on the Y variables only, and that this is the combinatorial part of the problem, counting open production periods can provide reasonings that are not done yet. A production period is said to be open if the corresponding setup cost is paid – if $Y_t = 1$. We think that knowing how many periods are open can give information on which periods to actually open and thus improve the global setup cost lower bound and better filter the variables. Reasoning on cardinalities has proven to be useful in certain cases, for instance when reasoning on the bin-packing global

constraint [85]. We therefore introduce the following redundant cardinality variables:

$$card = \sum_{t=1}^{T} Y_t \tag{3.24}$$

$$card_{t,t'} = \sum_{i=t}^{t'} Y_i \qquad \forall\, t < t' \in [\![1, T]\!] \tag{3.25}$$

The variable *card* is the number of actual open periods of production. The additional variables $card_{t,t'}$ count the number of open periods in every sub-plan of size at least 2. Note that the set of constraints (3.24) and (3.25) has an ideal formulation. We will see in chapter 4 how to build a propagator for this sub-problem.

3.5.1 Extending the dynamic programming

In order to increase the reasoning power of LOTSIZING, we modify the dynamic programming algorithm DPLS to include the cardinalities: it boils down to consider *card* as a resource and solve a Shortest Path Problem with Resource Constraints (SPPRC) in the graph of DPLS. We call this DP algorithm DPCard and it runs in $O(T^2 I_{max}^2)$. We denote $f_\sharp(t, I_t, card)$ the minimum cost for producing the demands from d_1 to d_t knowing that the stock level at the end of period t is I_t and that there are exactly *card* open periods from 1 to t.

$$\begin{aligned}
&\forall\, t \in [\![1, T]\!] \text{ and } \forall\, I_t \in [\![0, \overline{\beta_t}]\!] \\
&f_\sharp(t, I_t, card) = \min_{I_{t-1}=a\ldots b} \{ f_\sharp(t-1, I_{t-1}, card) + h_t I_t, \\
&\qquad\qquad f_\sharp(t-1, I_{t-1}, card-1) + s_t + h_t I_t + p_t X_t \}
\end{aligned} \tag{3.26}$$

where $a = \max\{0, d_t + I_t - \overline{\alpha_t}\}$, $b = \min\{\overline{\beta_{t-1}}, d_t + I_t\}$ and $X_t = I_t + d_t - I_{t-1}$. The initial states are

$$\forall\, q \in D(card), f_\sharp(0, 0, q) = 0 \text{ and } \forall\, I_t \in [\![1, I_{max}]\!], q \in D(card)\,,\ f(0, I_t, q) = +\infty$$

The value $\min_{q \in D(card)} f_\sharp(T, 0, q)$ gives the optimal cost.

Note that DPCard can be adapted to take into account only the production, the inventory or the setup costs and thus give lower bounds on Cp, Ch and Cs. This can be very costly but may allow the user to configure LOTSIZING to tackle more precisely the costs that seem important.

3.5.2 Filtering based on cardinalities

Similarly as for the other DP algorithms presented in this thesis we denote by $f_{\sharp,r}$ the reverse dynamic programming and use it to filter the variables: $f_{\sharp,r}(t, I_t, card)$ is the minimum cost for producing the demands from d_{t+1} to d_T knowing that the stock level at the end of period t is I_t and that there are exactly *card* open periods from $t+1$ to T.

Filtering the variable *card* can be done with the following rule:

$$\begin{aligned} &\forall\, q \in D(card) \\ &f_{\sharp}(T, 0, q) > \overline{C} \Rightarrow card \neq q \end{aligned} \tag{3.27}$$

It is however too costly to filter all the $card_{ij}$ variables so we filter only some of them. We only filter the $card_{1,t}$, $card_{t,T}$ and $card_{t,t+1}$ variables and only when the global cardinality variable *card* is instantiated. Indeed DPCard has one more dimension than DPLS – the dimension added by the variable *card*. The filtering rules are similar to the one we used before:

$$\begin{aligned} &\forall\, t \in [\![1, T]\!], a \in D(card_{1,t}), \\ &\max_{i_t \in D(I_t)} (f_{\sharp}(t, i_t, a) + \max_{b|a+b=card} (f_{\sharp,r}(t+1, i_t, b))) > \overline{C} \Rightarrow card_{1,t} \neq a \end{aligned} \tag{3.28}$$

$$\begin{aligned} &\forall\, t \in [\![1, T]\!], b \in D(card_{t,T}), \\ &\max_{i_t \in D(I_t)} (\max_{a|a+b=card} (f_{\sharp}(t-1, i_t, a)) + f_{\sharp,r}(t, I_t, b)) > \overline{C} \Rightarrow card_{t,T} \neq b \end{aligned} \tag{3.29}$$

$$\begin{aligned} &\forall\, t \in [\![1, T]\!], i_{t-1} \in D(I_{t-1}), i_t \in D(I_t), i_{t+1} \in D(I_{t+1}) \\ &\max_{a,b,c|a+b+c=card} (f_{\sharp}(t-1, i_{t-1}, a) + cost\ (t, i_{t-1}, i_t) + cost\quad (t+1, i_t, i_{t+1}) + f_{\sharp,r}(t+1, i_{t+1}, b)) \\ &\qquad\qquad > \overline{C} \Rightarrow card_{t,t+1} \neq c \end{aligned} \tag{3.30}$$

where $x_t = d_t + i_t - i_{t-1}$

$c = \mathbb{1}_{x_t>0} + \mathbb{1}_{x_{t+1}>0}$

$cost(t, i_{t-1}, i_t) = \mathbb{1}_{x_t>0} s_t + p_t x_t + h_t i_t$

We filter the production and inventory variables with the following filtering rules (when *card* is instantiated):

$$\forall\, t \in [\![1,T]\!], i_t \in D(I_t)$$
$$\max_{a,b|a+b=card}(f_\sharp\,(t,i_t,a) + f_{\sharp,r}(t,i_t,b)) > \overline{C} \Rightarrow I_t \neq i_t \tag{3.31}$$

$$\forall\, t \in [\![1,T]\!], i_{t-1} \in D(I_{t-1}), i_t \in D(I_t)$$
$$\max_{a,b|a+b+\mathbb{1}_{x_t>0}=card}(f_\sharp\,(t-1,i_{t-1},a) + cost(t,i_{t-1},i_t) + f_{\sharp,r}(t,i_t,b)) > \overline{C} \Rightarrow X_t \neq x_t \tag{3.32}$$

where $x_t = d_t + i_t - i_{t-1}$ and $cost(t,i_{t-1},i_t) = \mathbb{1}_{x_t>0}s_t + p_t x_t + h_t i_t$.

The variable *card* is filtered in $O(T)$, each variable $card_{1,t}$ and $card_{t,T}$ in $O(T^2 I_{max})$, each variable $card_{t,t+1}$ in $O(TI_{max}^3)$. Each I_t is filtered in $O(TI_{max})$ and each X_t is filtered in $O(TI_{max}^2)$.

3.5.3 Numerical results

We implemented DPCard in LOTSIZING along with the *card* variables and solved our multi-item instances with different parameters. We implemented six models using the different options presented in table 3.10. Numerical results are presented in tables 3.11 and 3.12.

	Code	Options
Branching	1	Branching on the Y_t
	2	Branching on *card* then on the Y_t
Reasoning	a	Computing lower bounds on all costs with DPCard
	b	Filtering *card*, $card_{1,t}$, $card_{t,T}$ and $card_{t,t+1}$ from DPCard
	c	Filtering I_t and X_t from DPCard

Table 3.10: New CP models based on cardinality variables

In conclusion, redundant cardinality variables that count the number of open production periods reduce the search space and provide a better root node lower bound than CPLS. However, since dynamic programming algorithms and entailed filtering take most of the computational effort, the resolution time is bigger. CPLS is still the fastest model on our instances. It looks better to branch on the card variable first which is intuitive since when the number of open periods is fixed, finding which periods to open should be easier. We can see that models with *b* and *c* are better than just *a* that cannot even solve all the instances. Indeed, filtering the

Models	NODE	CPU	RNB	LR	OPT
CPLS	27.5	1.7	5.9 %	NA	20
CP1a	15.0	41.5	5.8 %	NA	20
CP1ab	1.6	11.1	2.4 %	NA	20
CP1abc	1.6	11.0	2.4 %	NA	20
CP2a	5.5	15.8	5.8 %	NA	20
CP2ab	1.6	10.6	2.4 %	NA	20
CP2abc	1.6	10.9	2.4 %	NA	20

Table 3.11: Redundancy through cardinality variables (first 20 instances)

Models	NODE	CPU	RNB	LR	OPT
CPLS	157.8	9.8	7.3 %	NA	20
CP1a	60.1	102.6	6.4 %	NA	17
CP1ab	32.1	52.6	4.6 %	NA	20
CP1abc	32.0	51.6	4.6 %	NA	20
CP2a	39.9	80.7	6.4 %	NA	19
CP2ab	37.8	38.7	4.6 %	NA	20
CP2abc	36.9	38.0	4.6 %	NA	20

Table 3.12: Redundancy through cardinality variables (instances 21 to 40)

cardinality, production and inventory variables provides contradictions faster. The number of nodes is similar but the resolution time is better since each node is faster to compute.

Note that solving a SPPRC can directly give the bound consistency for LOT-SIZING when considering the four costs as resources. The computation is however too costly as the upper bounds of the costs variables can be very important. With DPCard we consider *card* as a resource, hence the additional dimension is in $O(T)$ and much smaller in practice.

3.6 More general cost structures

The aggregated/redundant LOTSIZING underestimates the production and holding costs by construction of the constraint since it deals with the minimum unitary costs over all the items at each period. This constraint is therefore a relaxation of the problem. It can be a useful modeling tool to have the LOTSIZING constraint handle piece-wise linear unitary costs to better capture the individual costs of the items and extend the generality of the single-item problem modeled by the global constraint to be able to solve a larger set of problems [8].

3.6.1 Piece-wise linear production and inventory costs

We define PC_t and HC_t the paid production and holding costs at a period t. These costs follow a step function where the steps correspond to the production or storage capacities of each item. They therefore depend on the quantity produced/stored at each period. Given a period t, until a certain quantity of units produced – the first step –, the cost is as if the items were produced using the smallest unitary production cost. Further units until another greater quantity – the second step – are produced using the second smallest unitary production cost and so on. Figure 3.4 (respectively 3.5) shows the evolution of PC_t (respectively HC_t) for each level of production X_t (respectively stock I_t) at period t.

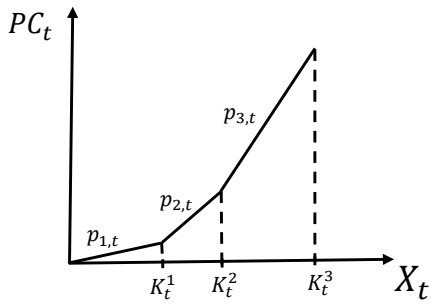

Figure 3.4: A step function to model unitary production costs

Consider the case of production costs. Take a period t and sort the unitary production costs in ascending order. In this order, for all steps $j = 1 \dots L$, we denote by K_t^j the production capacity of step j. We define $K_t^0 = 0$. The production unitary cost for units between two consecutive levels K_j and K_{j+1} is the j^{th} unitary production cost. Hence the production cost of X_t that is between two consecutive levels K_j and K_{j+1} is computed by:

$$PC_t = \sum_{l=1}^{j-1} p_{l,t}(K_t^l - K_t^{l-1}) + p_{j,t}(X_t - K_t^{j-1})$$

.

We can apply the same reasoning to the holding costs with M_t^j being the cumulative storage capacity. The holding cost of I_t that is between two consecutive levels M_j and M_{j+1} is computed by:

$$HC_t = \sum_{l=1}^{j-1} h_{l,t}(M_t^l - M_t^{l-1}) + h_{j,t}(I_t - M_t^{j-1})$$

.

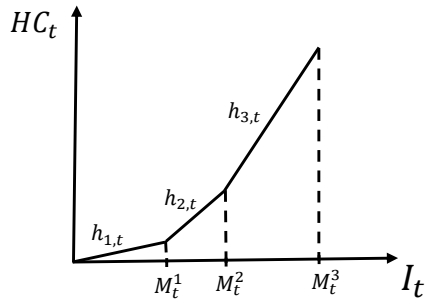

Figure 3.5: A step function to model unitary holding costs

The global production and inventory costs are now given by $Cp = \sum_{t=1}^{T} PC_t$ and $Ch = \sum_{t=1}^{T} HC_t$. The scope of LOTSIZING is modified to take into account the possibility to give a step function. In order to ensure the validity of the costs, we define PC_t and HC_t as variables in the model and we add a small propagator for each of them. In short, when the user adds the LOTSIZING constraint with step functions in the model, it also sets additional constraints that guaranty the correct computation of the production and holding costs PC_t and HC_t. In our multi-item problem, the step function of the redundant LOTSIZING is defined by the items. For each period, we sort the items by ascending unitary production/holding cost. In this order, for all item j, we define K_t^j and M_t^j as the cumulative production and inventory capacity (the upper bounds of $X_{j,t}$ and $I_{j,t}$).

In order to filter using the step functions, we define DPLSUC an extension of DPLS that takes into account the step functions. The computational complexity of DPLSUC is the same as for DPLS, indeed when the costs are needed, X_t and I_t are fixed hence the value of the corresponding production or holding cost is fixed as well. The filtering rules defined for DPLS also apply for DPLSUC.

Figure 3.6 shows a small example where the step functions improve the lower bound computed by DPLS. There are 2 items to produce over 2 periods. The production of item 1 is mandatory at period 1. Item 2 should be produced at period 1 since the setup cost at period 2 is quite high. The optimal cost is then $s_1 + h_{2,1} \text{ x } d_{2,2} = 1 + 2 \text{ x } 2 = \mathbf{5}$. The aggregated problem underestimates the holding cost between periods 1 and 2 to $h_{1,1} = 1$. DPLS then underestimates the overall cost to $s_1 + h_{1,1} \text{ x } d_{2,2} = 1 + 1 \text{ x } 2 = \mathbf{3}$. DPLSUC on the other hand takes into account the fact that only one item can be stored from periods 1 to 2 at cost $h_{1,1} = 1$ and returns a cost of $s_1 + h_{1,1} + h_{2,1} = 1 + 1 + 2 = \mathbf{4}$ and gives a better lower bound than DPLS.

Note that this small example displays a multi-item case. The improvement is made on LOTSIZING as we extend the global constraint to a more generic single-

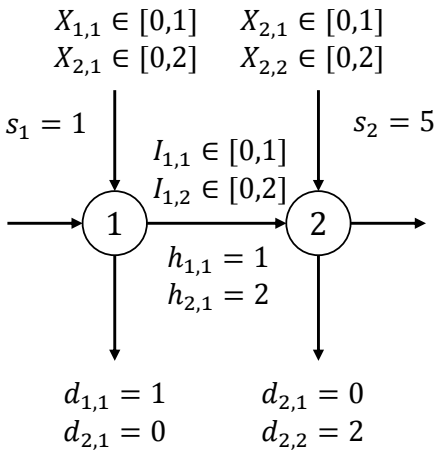

Figure 3.6: A better lower bound with a step function for unitary costs

item lot-sizing problem than in the previous chapter, where the unitary costs can follow a step function.

3.6.2 Numerical results

Tables 3.14 show the results of the resolution of the 40 instances and compares the model CPUC to CPLS.

Models	NODE	CPU	RNB	LR	OPT
CPLS	27,5	2,1	5,9 %	NA	20
CPUC	27,5	2,0	5,9 %	NA	20

Table 3.13: Variable unitary costs (first 20 instances)

Models	NODE	CPU	RNB	LR	OPT
CPLS	157,8	10.7	7,3%	NA	20
CPUC	157,9	10,5	7,3%	NA	20

Table 3.14: Variable unitary costs (instances 21 to 40)

The improved LOTSIZING is still a relaxation since no distinction is made between the products flowing though its network. The steps give only maximum quantities of products that can be produced/stored at the smaller costs. The drawback is that steps being computed at the beginning of the resolution, there are cases

where the steps are too high and provide poor or no filtering. Anyway, this improvement seems a very useful modeling tool and is free from the point of view of algorithmic complexity. To avoid having too big steps, it should be possible to maintain them during the search, thus tightening the reasoning made by DPLSUC.

3.7 Conclusion

In this chapter we have seen that the naive decomposed model cannot deal with the decisions that have to be made about the quantities to produce and store and thus cannot solve the instances let alone compete with MILP . Thanks to LOTSIZING, we have shown that there is a way to tackle multi-item problems using constraint programming when the branching is not done on the quantity variables but on the binary variables and dynamic programming techniques allow us to filter these quantities during the search. On the capacitated multi-item lot-sizing problem with shared setup costs, our model using LOTSIZING is not competitive against MILP but this was expected. We give reasoning mechanisms to help tackle problems with combinatorial side constraint, more complex than the pure academic lot-sizing problems. We also improve the strength and the modularity of LOTSIZING by adding redundant cardinality variables and a way to model piece-wise linear unitary costs. The latter could be improved using variables steps. Finally, to give some insights on how to solve lot-sizing problems, we believe that one should not branch on quantity variables. One should instead find where lies the combinatorial difficulty of the problem and tune the CP model using the different options of the LOTSIZING global constraint. The key is to find the resulting problem once the $0/1$ variables are fixed. Since the flow graph of lot-sizing problems has usually a very particular structure that makes it easy for MILP models to solve, the resulting problem is likely to be polynomial or easily solvable in practice. Finally LOTSIZING can easily be used to add powerful redundant constraints based on the structure of the problem.

Chapter 4

Filtering via linear programming

4.1 Introduction

Mixed Integer Linear Programming (MILP) and Constraint Programming (CP) have benefited from each other increasingly in recent years due to the complementary strengths of the two frameworks. Many approaches have been proposed to combine their modeling and solving capabilities [4, 10, 23, 76, 82]. On one side, CP tailored algorithms for specific constraints take advantage of local combinatorial structures to reduce the search space efficiently. On the other, MILP techniques usually encompass the whole problem and typically compute lower/upper bounds of the objective function that propagation through the domains fails to derive. A typical integration of the two approaches is to use the linear relaxation of the entire problem in addition to the local consistencies enforced by the CP solver. The relaxation is used to perform filtering, in particular by providing a global bound of the objective but also by filtering the domains using a technique referred to as reduced-cost-based filtering [50, 68]. Constraints can in turn provide specialized linear formulations and cutting planes.

As opposed to previous work, we investigate in this chapter how linear programming (LP) can be used to filter individual constraints and in particular to provide arc consistency algorithms. Let us suppose that an ideal linear formulation $\mathcal{F}$ over n variables is available for a global constraint. A formulation is referred to as ideal when it has the integrality property *i.e.* when the extreme points of the corresponding polytope are integer points. It is easy to come by such formulations for many global constraints [76] that include 0/1 variables typically encoding whether an original variable of the constraint is assigned to a given value of its domain. Since $\mathcal{F}$ is supposed ideal, a simple way to achieve arc consistency is to fix, in turn, each variable to each value and check the consistency by calling a linear solver. This is

very similar to the failed literal test mentioned in [12] for achieving arc consistency with unit propagation over a SAT encoding of the constraint.

In [73, 74], it was briefly remarked that the solution of the LP relaxation of a MILP obtained by a single call of the interior point algorithm allows domain filtering using the property that the algorithm converges toward an interior point of the polyhedron if it exists. Hence a variable set to one of its boundary in the relaxed solution can be fixed to its boundary in any optimal integer solution. It was also stated (without proof) that in case that the LP matrix of a global constraint is totally unimodular, i.e. the formulation is ideal, arc consistency is enforced by this procedure and that a potentially powerful level of consistency can be achieved on an LP of a global constraint whose matrix can be decomposed into totally unimodular submatrices.

We show, however, that arc consistency for a set of constraints stated over n variables can be achieved in this case by solving *a single linear program* with n additional variables and $2n$ additional constraints. The idea is to look for an interior point, of the convex hull of $\mathcal{F}$ maximizing the number of variables with a slack to their respective lower and upper bounds. Although this goes against the rationale explained above for integrating the two frameworks, we believe the advantages are twofold. First of all, since each solver only provides a handful of the existing constraints, it is particularly useful to quickly design arc consistency algorithms for many polynomial constraints. Secondly, it can provide a generic but competitive algorithm for constraints with a quadratic running time such as the GEN-SEQUENCE [63].

The linear relaxation has been used in the past for filtering and we now review several closely related works [4, 10, 76, 82] that propose frameworks for combining the linear relaxation, specialized cutting planes generation and filtering. An illustrative example is the work of [76] where each constraint is able to provide its linear relaxation so that a global relaxation of the entire problem is automatically derived from a CP model. Additionally, a constraint is able to add dedicated cutting planes during search, taking advantage of the corresponding combinatorial structure to build a stronger relaxation. The linear relaxations of common global constraints such as ELEMENT, ALLDIFFERENT, CIRCUIT and CUMULATIVE can be found in [51, 76] and relaxations of global constraints involving costs such as MINIMUMWEIGHTALLDIFFERENT or WEIGHTEDCIRCUIT are described in [43]. The linear relaxation is directly used for filtering by [4, 10, 43, 75, 76]. It can detect infeasibility, provide a bound for a cost variable and perform filtering using a technique referred to as reduced-cost based filtering [43, 50]. The latter is a specific case of cost-based filtering [42] that aims at filtering out values leading to non-improving solutions.

In the previous chapters, we use LP to deal with several sub-problems of the lot-sizing problems considered: for instance deciding the quantities when the dom-

inance rule of LOTSIZING applies or solving the polynomial multi-flow of the multi-item problem. This chapter is a first step to improve the reasoning mechanisms that LP can provide. Moreover, the cardinality constraints used to strengthen the filtering of LOTSIZING have an ideal formulation. The LOTSIZING constraint might benefit from having strong filtering algorithms reason on these sub-problems.

Section 4.2 summarizes the key notations. Section 4.3 reviews and explains reduced-cost based filtering in more details. The main result of this chapter is presented section 4.4 and its application to ALLDIFFERENT, GLOBALCARDINALITY and GEN-SEQUENCE constraints is described in section 4.5. Finally experimental results are reported on three benchmarks in section 4.6.

4.2 Notations

A constraint satisfaction problem is made of a set of variables, each with a given domain, *i.e.* a finite set of possible values, and a set of constraints specifying the allowed combinations of values for subset of variables. In the following, the variables, *e.g.* X_i, are denoted with upper case letters for the constraint programming models as opposed to the variables of linear programming models that are in lower case. $D(X_i) \subseteq \mathbb{Z}$ denotes the domain of X_i. A constraint c over a set of variables $\langle X_1, \ldots, X_n \rangle$ is defined by the allowed combinations of values (tuples) of its variables. Such tuples of values are also referred to as solutions of the constraint c. Given a constraint c with a scope $\langle X_1, \ldots, X_n \rangle$, a **support** for c is a tuple of values $\langle v_1, \ldots, v_n \rangle$ that is a solution of c and such that $v_i \in D(X_i)$ for all variable X_i in the scope of c. Consider a variable X_i in the scope of c, the domain $D(X_i)$ is said **arc consistent** for C if and only if all the values $v_j \in D(X_i)$ belong to a support for c. A constraint c is said arc consistent if and only if all its variables are arc consistent.

4.3 Traditional filtering using LP: reduced-cost filtering

Let us first review how linear programming is traditionally used to perform filtering. Suppose we are dealing with a minimization problem. Cost-based filtering relies on a known upper bound $\overline{z}$ of the objective function which is usually the cost of the best feasible solution found so far. Since there is no need to consider solutions with a greater cost than $\overline{z}$, values of the domains that would necessarily

lead to such non-improving solutions should be filtered. Linear reduced-costs provide valuable information to perform such reasonings. They are available from an optimal dual solution of the linear relaxation and give a minimum increase of the objective function. This increase can be used to detect if a bound of a variable leads to a non-improving solution. When applied to $0/1$ variables, *i.e.* variables with the domain $\{0,1\}$, any update of a bound leads to fixing a variable to 0 or 1. It has thus been known for a long time as *variable fixing*.

To our knowledge, the best account of this technique in the context of constraint programming is given in [50]. It is usually presented in textbooks on integer programming such as [68, 101] for $0/1$ variables. We give a summary of this technique in the more general case of integer variables. Consider a linear programming formulation (P) where the feasible region is defined by a polytope $\mathcal{Q} = \{x \in \mathbb{R}^T \mid Ax \geq b, l \leq x \leq u\}$. Note that each variable x_t for all $t \in \{1, \ldots, T\}$, has a given lower and upper bound *i.e.* $x_t \in [l_t, u_t]$.

$$(P) \qquad z^* \;=\; \min\{cx : x \in \mathcal{Q}\}$$

Program (P) is typically the linear relaxation of an integer programming formulation identified for the whole problem or for a single constraint. Let α be the m dual variables of the constraints $Ax \geq b$. Moreover, let x^* be an optimal solution of (P) and α^* a set of optimal values of the α variables. The reduced cost r_t of a variable x_t, with respect to α^*, is defined as $r_t = c_t - \alpha^* A_t$ where A_t is the t-th column of A. Note that the definition of r_t ignores the dual variables related to the lower and upper bounds since $x_t \leq u_t$ and $x_t \geq l_t$ are usually not added as constraints to the formulation of (P) but handled directly by the simplex algorithm (see [31] for more details). The reduced cost r_t is typically the quantity returned by linear programming solvers when the bounds are not explicitly stated as constraints in the model but directly as bounds of the variable's domains.

Reduced-cost-based filtering removes values necessarily leading to non-improving solutions *i.e.* solutions of cost greater than the known upper bound $\overline{z}$. The following rules can be used for filtering a variable x_t of (P) from an optimal dual solution.

Proposition 1 (Reduced cost filtering)

$$\textit{If } r_t > 0 \textit{ then } x_t \leq l_t + \frac{(\overline{z} - z^*)}{r_t} \textit{ in any solution of cost less than } \overline{z} \tag{4.1}$$

$$\textit{If } r_t < 0 \textit{ then } x_t \geq u_t - \frac{(\overline{z} - z^*)}{-r_t} \textit{ in any solution of cost less than } \overline{z} \tag{4.2}$$

Note that if x_t is originally an integer variable, the reasoning can be tightened as $x_t \leq l_t + \lfloor \frac{(\overline{z}-z^*)}{r_t} \rfloor$ and $x_t \geq u_t - \lfloor \frac{(\overline{z}-z^*)}{-r_t} \rfloor$.

The two rules are a direct consequence of traditional sensitivity analysis and the reader can refer to [50, 101] for more details.

The filtering obtained from a particular optimal dual solution is usually incomplete since r_t depends on the specific α^* found. In other words, considering several optimal dual solutions may provide more filtering. Let us go through a very simple example to illustrate this point. We consider a difference constraint $X_1 \neq X_2$ with $D(X_1) = \{1, 2\}$ and $D(X_2) = \{1\}$. Value 1 of $D(X_1)$ is thus expected to be filtered. A simple integer formulation of the feasible solutions can be written with 0/1 variables x_1, x_2, x_3 respectively encoding whether $X_1 = 1$, $X_1 = 2$ or $X_2 = 1$. The linear relaxation and its dual problem write as follows:

$$(P) \quad \begin{array}{lll} \min & 0 & \\ & x_1 + x_2 & \geq 1 \\ & x_3 & \geq 1 \\ & x_1 + x_3 & \leq 1 \\ & x_1 + x_2 & \leq 1 \\ & x_1, x_2, x_3 & \geq 0 \\ & x_1, x_2, x_3 & \leq 1 \end{array} \qquad (D) \quad \begin{array}{lll} \max & \sum_{i=1}^{4} \alpha_i + \sum_{i=1}^{3} \beta_i & \\ & \alpha_1 + \alpha_3 + \alpha_4 + \beta_1 & \leq 0 \\ & \alpha_1 + \alpha_4 + \beta_2 & \leq 0 \\ & \alpha_2 + \alpha_3 + \beta_3 & \leq 0 \\ & \alpha_1, \alpha_2 & \geq 0 \\ & \alpha_3, \alpha_4 & \leq 0 \\ & \beta_1, \beta_2, \beta_3 & \leq 0 \end{array}$$

$x^* = (0, 1, 1)$ is an optimal solution of (P). It is easy to see that $\alpha^* = (0, 0, 0, 0)$ or $\alpha^* = (0, 1, -1, 0)$ are two vectors of optimal values for α (with $\beta^* = (0, 0, 0)$). The reduced cost of x_1 is $r_1 = 0 - \alpha_1^* - \alpha_3^* - \alpha_4^*$. In the first case, $r_1 = 0$ and none of the rules given in Proposition 1 is triggered. In the second case, $r_1 = 1$ and the first rule applies with $\overline{z} = 0$ enforcing $x_1 \leq 0$ as expected. In both cases $r_2 = r_3 = 0$, therefore x_2 and x_3 are not filtered.

Note that this drawback occurs even if the polytope $\mathcal{Q}$ has integer extreme points which is the case in the example. Moreover, in any case, minimizing 0 in (P) implies that $\alpha^* = 0$ is an optimal vector for α and the linear solver can always return it. To reduce this phenomenon, it is possible to use another objective function cx as long as all feasible solutions have the same cost (see Section 4.6 for an example).

An alternative approach to reduced costs is proposed in the next section to perform a complete variable fixing.

4.4 A new generic filtering algorithm based on LP

Let us briefly explain the general idea and its application to filtering global constraints before stating the result in detail.

Consider a polytope $\mathcal{Q} = \{x \in \mathbb{R}^T \mid Ax \geq b, l \leq x \leq u\}$ with integer extreme points and $l, u \in \mathbb{Z}^T$. We show in this section that a variable that is fixed to one of its bound (either l_t or u_t) in all extreme points of $\mathcal{Q}$ can be detected by solving a linear program with T additional variables and $2T$ additional constraints. The idea is to look for an interior point of $\mathcal{Q}$ maximizing the number of variables with a slack to their respective lower and upper bounds. When no slack is possible, the variable is proved to be fixed to the same value in all extreme points.

For numerous polynomial global constraints, it is possible to give an *ideal integer programming formulation* $\mathcal{F}$ of the solutions where $0/1$ variables x_{ij} typically encode whether an integer variable X_i of the constraint's scope is assigned to value a v_j of its domain. The linear relaxation of $\mathcal{F}$ defines a polytope $\mathcal{Q}$ that represents the convex hull of the supports of the constraint. Each integer point in $\mathcal{Q}$ can be seen as a support of the constraint. The proposed technique identifies all $0/1$ variables that are set to 0 in all extreme points. Since all interior points of $\mathcal{Q}$ are a convex combination of the extreme points, the same variables are thus also set to 0 for all interior points of $\mathcal{Q}$, i.e. the corresponding values do not belong to any support. Since *complete variable fixing* can be performed (all inconsistent values are removed), the proposed technique gives the arc consistent domains for the constraint.

The main result is stated as Theorem 5. The polytope $\mathcal{Q} = \{x \in \mathbb{R}^T \mid Ax \geq b, l \leq x \leq u\}$ is assumed to have integer extreme points and $l, u \in \mathbb{Z}^T$. Let us also denote by $\mathcal{S}$ the set of extreme points of $\mathcal{Q}$.

Theorem 5 *Let us define* $\varepsilon = \dfrac{1}{(T+1)}$, *and* (P') *the following linear program:*

$$\begin{array}{lll} \min & z(x,e) = \displaystyle\sum_{t=1}^{T} e_t & \\ s.t. & x_t + e_t \geq l_t + \varepsilon & \forall t \in \{1,...,T\} \\ & x_t - e_t \leq u_t - \varepsilon & \forall t \in \{1,...,T\} \\ & e_t \geq 0 & \forall t \in \{1,...,T\} \\ & x \in \mathcal{Q} & \end{array}$$

For all $t \in \{1,...,T\}$, *all* $\delta \in \{l_t, u_t\}$, *and all optimal solution* (x^*, e^*) *of* (P') *we have:*

$$x_t^* = \delta \qquad \Leftrightarrow \qquad \hat{x}_t = \delta \quad \forall \hat{x} \in \mathcal{S}$$

Note that a feasible solution of (P') with $e_t = 0$ indicates that variable x_t has a slack of at least ε to its lower and upper bound. Keep also in mind that the objective of (P') is to minimize the sum of the e_t which tend to create slack. We will show that any optimal solution of (P') actually maximizes the number of variables that can be unstuck from their bounds revealing all the x_t that are always instantiated to either l_t or u_t. We first make a simple observation:

Remark. If (x^*, e^*) is an optimal solution of (P'), then $e^* = e(x^*)$ with $e : \mathbb{R}^T \to \mathbb{R}^T$ such that $e(x) = (e_1(x), ..., e_t(x), ..., e_T(x))$ and

$$e_t(x) = \max\{0, \varepsilon + l_t - x_t, \varepsilon - u_t + x_t\} \qquad \forall t \in \{1, ..., T\}$$

Proof. Each variable e_t only occurs in three constraints of (P'): $e_t \geq 0$, $e_t \geq \varepsilon + l_t - x_t$ and $e_t \geq \varepsilon - u_t + x_t$. Given a value of x^*, the minimum possible feasible value for e_t is thus $\max\{0, \varepsilon + l_t - x_t, \varepsilon - u_t + x_t\}$. □

The optimal objective value of (P') is at least ε times the number of variables that are necessarily fixed to either l_t or u_t. The proof given below builds a feasible solution of (P') reaching this bound by setting $e_t(x) = 0$ for all other variables. Thus, any optimal solution highlights the fixed variables. We now prove theorem 5.

Proof.

We denote by $\mathcal{T}^l = \{t \in \{1, ..., T\} \mid \hat{x}_t = l_t, \ \forall \hat{x} \in \mathcal{S}\}$ and $\mathcal{T}^u = \{t \in \{1, ..., T\} \mid \hat{x}_t = u_t, \ \forall \hat{x} \in \mathcal{S}\}$ the sets of indices referring to variables fixed respectively to their lower or upper bounds, in all extreme points of $\mathcal{Q}$. As mentioned above, a valid lower bound (P') is $\varepsilon(|\mathcal{T}^l| + |\mathcal{T}^u|)$.

Let $(\hat{x}^0, \hat{x}^1, \hat{x}^2, ..., \hat{x}^T)$ be a series of extreme points of $\mathcal{S}$ defined as follows. $\hat{x}^0$ is chosen arbitrarily in $\mathcal{S}$. Each $\hat{x}^t$ such that $t \notin \mathcal{T}^l \cup \mathcal{T}^u$ is chosen in $\mathcal{S}$ so that $\hat{x}^t_t \neq \hat{x}^0_t$. Finally, all remaining $\hat{x}^t$ are chosen arbitrarily in $\mathcal{S}$.

Based on this series of points, we can define a feasible solution $(\bar{x}, e(\bar{x}))$ of (P') by considering $\bar{x}$ as the following convex combination $\bar{x} = \dfrac{1}{T+1} \sum_{t=0}^{T} \hat{x}^t$.

Firstly, note that $\bar{x}_t \in \{l_t, u_t\}$ if and only if $t \in \mathcal{T}^l \cup \mathcal{T}^u$. Indeed, $l_t \leq \hat{x}_t \leq u_t$ for all $\hat{x} \in \mathcal{S}$, so we have $\bar{x}_t \in \{l_t, u_t\}$ if and only if $\hat{x}^{t'}_t = \bar{x}_t$ for all $t' \in \{0, ..., T\}$. Therefore, by construction, $\bar{x}_t \in \{l_t, u_t\}$ if and only if $\hat{x}_t = \hat{x}^0_t$ for all $\hat{x} \in \mathcal{S}$, *i.e.* $t \in \mathcal{T}^l \cup \mathcal{T}^u$.

Secondly, all other $\bar{x}_t$ have a slack of at least ε. For all $t \notin \mathcal{T}^l \cup \mathcal{T}^u$, we have $\hat{x}^t_t \neq \hat{x}^0_t$, therefore $\max\{\hat{x}^t_t, \hat{x}^0_t\} \geq l_t + 1$ and $\min\{\hat{x}^t_t, \hat{x}^0_t\} \leq u_t - 1$ since extreme points of $\mathcal{Q}$ are integers. As a result:

$$\begin{array}{lllllll} \bar{x}_t & \geq & \dfrac{1}{T+1}(\max\{\hat{x}^t_t, \hat{x}^0_t\} + T\, l_t) & \geq & \dfrac{1}{T+1}(l_t + 1 + T\, l_t) & = & l_t + \varepsilon \\ \bar{x}_t & \leq & \dfrac{1}{T+1}(\min\{\hat{x}^t_t, \hat{x}^0_t\} + T\, u_t) & \leq & \dfrac{1}{T+1}(u_t - 1 + T\, u_t) & = & u_t - \varepsilon \end{array}$$

Hence for all $t \in \{1, ..., T\}$, we have

$$e_t(\bar{x}) = \begin{cases} \varepsilon & \text{if } t \in \mathcal{T}^l \cup \mathcal{T}^u \\ 0 & \text{otherwise} \end{cases}$$

Thus $z(\bar{x}, e(\bar{x})) = \varepsilon(|\mathcal{T}^l| + |\mathcal{T}^u|)$ which proves that the solution $(\bar{x}, e(\bar{x}))$ is optimal. Any optimal solution x^* must therefore have a cost of $\varepsilon(|\mathcal{T}^l| + |\mathcal{T}^u|)$. Since $e_t(x^*) \geq 0$ for all t and $e_t(x^*) = \varepsilon$ for all $t \in \mathcal{T}^l \cup \mathcal{T}^u$, $e_t(x^*) = 0$ for all $t \notin \mathcal{T}^l \cup \mathcal{T}^u$.

Conclusion: for all optimal solutions (x^*, e^*) of (P'), all $t \in \{1, ..., T\}$ and all $\delta \in \{l_t, u_t\}$,

$$x_t^* = \delta \qquad \Leftrightarrow \qquad t \in \mathcal{T}^l \cup \mathcal{T}^u \qquad \Leftrightarrow \qquad \hat{x}_t = \delta \quad \forall \hat{x} \in \mathcal{S}$$

□

We now propose a simple application of this result to filtering global constraints. Consider a polynomial global constraint C over a scope $X = \langle X_1, \ldots, X_n \rangle$ of n integer variables with their respective domains $D(X_i) \in \mathbb{Z}$. The approach proposed to enforce arc consistency is summarized below:

LP-based filtering for constraint C:
Inputs: A constraint c over the variables $\mathcal{X} = \langle X_1, \ldots, X_n \rangle$. An ideal integer formulation $\mathcal{F}$ of the solutions of c where a $0/1$ variable x_{ij} is present for all $X_i \in X, v_j \in D(X_i)$ to encode whether variable X_i takes value v_j.
Output: arc consistent domains $D(X_1), \ldots, D(X_n)$ for constraint c.

1. Consider $\mathcal{Q}$ as the convex hull of $\mathcal{F}$ by simply relaxing the domain's constraint $x_{ij} \in \{0, 1\}$ into $x_{ij} \in [0, 1]$ for all x_{ij}
2. Find an optimal solution (x^*, e^*) of (P') as defined in theorem 5
3. For each $X_i \in \mathcal{X}$ and each $v_j \in D(X_i)$, if $x_{ij}^* = 0$, remove value v_j from $D(X_i)$

The procedure given above computes arc consistent domains as a direct consequence of theorem 5 : indeed $x_{ij}^* = 0$ means that $x_{ij} = 0$ for any solution of the LP, hence the corresponding value has to be removed. Furthermore, when $x_{ij}^* = 1$, $x_{ik}^* = 0$ for all $k \neq j$.

Corollary 1 *The procedure **LP-based filtering** is correct and establishes arc consistency for constraint c.*

Proof. Recall that the integer points of $\mathcal{Q}$ represent the supports of c. Since any interior point can be written as a convex combination of the extreme points, there exists at least one extreme point $\hat{x} \in \mathcal{S}$ such that $\hat{x}_{ij} = 1$ for any consistent value v_j of a $D(X_i)$. Similarly when all $\hat{x}_{ij} = 0$ for all $\hat{x} \in \mathcal{S}$, it is the case of all interior integer points. Keeping that in mind, we simply check that the procedure does not remove any consistent value (it is correct) and removes all inconsistent values (it is complete).

Correct: consider a consistent value v_j in $D(X_i)$. Since it belongs to a support, there exists at least one $\hat{x} \in \mathcal{S}$ such that $\hat{x}_{ij} = 1$. Therefore, $x^*_{ij} \neq 0$ according to theorem 5 and value v_j is not removed by the proposed procedure.

Complete: Let us check that all remaining values belong to a support. Consider a value v_j of a domain $D(X_i)$ after the procedure has been called. Since v_j has not been filtered, $x^*_{ij} \neq 0$ implying by theorem 5 that there exists at least one extreme point $\hat{x} \in \mathcal{S}$ such that $\hat{x}_{ij} = 1$. Therefore v_j belongs to the corresponding support. □

The complexity of LP-based filtering depends on the algorithm used to solve the LP. In practice, the simplex algorithm is known to have a number of iterations proportional to $m \log(n)$ [31] where n is the number of variables and m the number of constraints of the LP formulation.

4.5 Ideal formulations of polynomial global constraints

We now provide an ideal linear integer programming formulation $\mathcal{F}$ for a number of polynomial global constraints. The fact that a variable X_i takes one and one value only of its domain ($X_i \in D(X_i)$) is typically expressed in $\mathcal{F}$ following [76]:

$$\begin{cases} \sum\limits_{v_j \in D(x_i)} x_{ij} = 1 \\ x_{ij} \in \{0,1\} \qquad \forall v_j \in D(X_i) \end{cases} \tag{4.3}$$

4.5.1 AllDifferent and GlobalCardinality

The AllDifferent$(X_1, \ldots, X_n)$ constraint [78] is satisfied when the variables $X_1, \ldots, X_n$ take different values. The formulation given below is the classical formulation for the matching problem and is known to have the integrality property:

$$\mathcal{F} = \begin{cases} \sum\limits_{i | v_j \in D(X_i)} x_{ij} \leq 1 & \forall v_j \in \bigcup_{i=1}^{n} D(X_i) \\ \sum\limits_{v_j \in D(x_i)} x_{ij} = 1 & \forall i \in \{1, \ldots, n\} \\ x_{ij} \in \{0,1\} & \forall i \in \{1, \ldots, n\}, \forall v_j \in D(X_i) \end{cases} \tag{4.4}$$

A related global constraint is the GlobalCardinality constraint [80]. It enforces the number of occurrences of each value v_j to be at least l_j and at most u_j in the set

$X_1, \ldots, X_n$ of variables. The formulation $\mathcal{F}$ is the following:

$$\mathcal{F} = \begin{cases} \sum\limits_{i|v_j \in D(X_i)} x_{ij} & \leq & u_j & \forall v_j \in \bigcup_{i=1}^{n} D(X_i) \\ \sum\limits_{i|v_j \in D(X_i)} x_{ij} & \geq & l_j & \forall v_j \in \bigcup_{i=1}^{n} D(X_i) \\ \sum\limits_{v_j \in D(x_i)} x_{ij} & = & 1 & \forall i \in \{1, \ldots, n\} \\ x_{ij} & \in & \{0,1\} & \forall i \in \{1, \ldots, n\}, \forall v_j \in D(X_i) \end{cases} \tag{4.5}$$

$\mathcal{F}$ has the integrality property since the matrix can be seen as a specific case of a network flow matrix. Let us denote by $d = \max_{i=1}^{n} |D(X_i)|$, the maximum cardinality of the domains and $m = |\bigcup_{i=1}^{n} D(X_i)|$, the total number of distinct values. Both formulations given have $O(nd)$ variables and $O(n+m)$ constraints. Arc consistency can thus be established by solving the program (P') which has twice the number of variables and $O(nd+m)$ constraints. Recall that the dedicated algorithms for each constraint respectively runs in $O(n^{1.5}d)$ for the ALLDIFFERENT, $O(n^2m)$ for the GLOBALCARDINALITY and are incremental down a branch of the search tree.

4.5.2 The family of SEQUENCE constraints

The SEQUENCE constraint restricts the number of occurrences of some given values in any sequence of k variables. It can be expressed as a conjunction of AMONG constraints and has been used for car sequencing [18] and nurse rostering [29]. More precisely, AMONG$(l, u, \langle X_1, \ldots, X_k \rangle, V)$ holds if and only if $l \leq |\{i | X_i \in V\}| \leq u$. In other words, at least l and at most u of the variables take their values in the set V. The SEQUENCE constraint can be defined as a conjunction of *sliding* AMONG constraints over k consecutive variables. SEQUENCE$(l, u, k, \langle X_1, \ldots, X_n \rangle, V)$ holds if and only if $\forall i \in \{1, \ldots, n-k+1\}$ AMONG$(l, u, \langle X_i, \ldots, X_{i+k-1} \rangle, V)$ holds.

An incomplete filtering algorithm for SEQUENCE is proposed in [15]. Two arc consistency algorithms are later given in [96, 97] with respective running times of $O(n^3)$ and $O(2^k n)$. Additionally, an encoding achieving arc consistency is presented in [29] and runs in $O(n^2 log(n))$ down a branch of a search tree. It is latter improved in [63] by using the fact that a natural integer programming formulation of the constraint has the consecutive ones property on the columns. This is used to build a network flow graph and derive an arc consistency algorithm. The complexity of this flow-based propagator to enforce arc consistency is $O(n((n-k)(u-l)+u))$ when using the Ford-Fulkerson algorithm for finding a maximum flow. The incremental cost when fixing a single variable is only $O(n)$ so that the algorithm runs in $O(n^2)$ down a branch of a search tree.

A generalization of SEQUENCE is known as GEN-SEQUENCE and allows different occurrences (l and u) and sizes (k) for an arbitrary set of m AMONG constraints

over consecutive variables. GEN-SEQUENCE$(p_1, \ldots, p_m, \langle X_1, \ldots, X_n \rangle, V)$ holds if and only if $\forall\, 1 \leq i \leq m$, AMONG$(l_i, u_i, \langle X_{s_i}, \ldots, X_{s_i+k_i-1} \rangle, V)$ holds where $p_i = \{l_i, u_i, k_i, s_i\}$.

The GEN-SEQUENCE global constraint is defined in [96, 97] and a $O(n^4)$ algorithm is proposed to enforce arc consistency. The consecutive one property does not hold in general for a GEN-SEQUENCE constraint. Although it may sometimes be possible to re-order the lines of the matrix to have the consecutive one property on the columns or to find an equivalent network matrix, [63] outlines that not all GEN-SEQUENCE constraints can be expressed as network flows. The flow-based algorithm for SEQUENCE can therefore not be reused in general. Nonetheless, the encoding of SEQUENCE proposed in [29] extends to GEN-SEQUENCE and runs in $O(nm + n^2 \log n)$ [63]. Finally, in [12], a filtering method based on unit propagation over a conjunctive normal form encoding of the constraint is proposed and achieves arc consistency in $O(mn^3)$.

Note that the previous sequence constraints can be encoded with a simple boolean channeling, without hindering any filtering since the resulting constraint network is Berge-acyclic. Typically GEN-SEQUENCE$(p_1, \ldots, p_m, \langle X_1, \ldots, X_n \rangle, V)$ can be stated as:

$$\begin{cases} \text{GEN-SEQUENCE}(p_1, \ldots, p_m, \langle Y_1, \ldots, Y_n \rangle, 1) \\ Y_i = 1 \Leftrightarrow X_i \in V, \forall i \in \{1, \ldots, n\} \\ Y_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \end{cases} \tag{4.6}$$

All previous studies thus focused on the restricted case where $V = \{1\}$. The integer linear formulation for GEN-SEQUENCE$(p_1, \ldots, p_m, \langle Y_1, \ldots, Y_n \rangle, 1)$ is the following:

$$\mathcal{F} = \begin{cases} \sum_{j=s_i}^{s_i+k_i-1} y_j \leq u_i & \forall i \in \{1, \ldots, m\} \\ \sum_{j=s_i}^{s_i+k_i-1} y_j \geq l_i & \forall i \in \{1, \ldots, m\} \\ y_j \in \{0, 1\} & \forall j \in \{1, \ldots, n\} \end{cases} \tag{4.7}$$

This formulation has the integrality property, as already mentioned in [63]. The linear program (P') solved by the proposed LP-based procedure to enforce arc consistency has $O(n)$ variables and $O(m + n)$ constraints. Recall that the best arc consistency algorithm for GEN-SEQUENCE is the encoding of [63] and runs in $O(nm + n^2 \log n)$.

4.6 Numerical results

We carried out experiments on the ALLDIFFERENT and SEQUENCE global constraints. The first set of experiments compares the LP-based filtering to the dedicated filtering algorithm of ALLDIFFERENT (section 4.6.1). It also evaluates in practice the power of reduced-cost based filtering. The second one (section 4.6.2) compares two encodings of SEQUENCE to the LP-based filtering on random sequences alone following the experiments reported in [63]. Finally the last one evaluates the LP-based filtering on the Car-sequencing problem (section 4.6.3).

The experiments were performed with Windows 8 on an Intel Core i5 @ 2.5 GHz with 12GB of RAM. A memory limit of 4GB of RAM was used. The indicators shown in the result tables are the average resolution time in seconds (CPU), the average number of nodes (N) and the average speed of the resolution in node per second (N/s). The constraint solver used is Choco 3.3 [72].

4.6.1 LP and reduced-cost filtering for the ALLDIFFERENT constraint

The LP-based filtering is implemented for ALLDIFFERENT with the polytope given in Section 4.5.1. It is referred to as ALLDIFFERENTLPF (for LP Filtering) and compared with three other filtering algorithms: the Choco ALLDIFFERENT constraint, the decomposition into cliques of difference constraints (DEC) and the reduced-cost based filtering algorithm in addition to the decomposition (RCF). As mentioned in section 4.3, when filtering via reduced costs, the use of an objective function of the form cx can increase the chances of having non null reduced costs. To perturb the dual, the objective function used is $\sum_{i=1}^{n} c_i(\sum_{j \in D(X_i)} x_{ij})$, where the c_i are randomly chosen in $[-10, 10]$ at each node. Note that this function guarantees that all feasible solutions have the same cost.

We solve the QuasiGroup Completion problem [70]. The problem is to fill a n by n matrix previously filled at k% with numbers from 1 to n such that on each line and on each column, each number appears only once. We compare four models and look for the number of nodes needed to find all solutions for small instances with a lexicographic branching heuristic. Table 4.1 shows the average results on 10 randomly generated instances for each size $n \in \{5, 10, 15\}$. These three classes of instances are respectively filled at 10, 40 and 50% to have instances solvable under 3600s.

As expected, ALLDIFFERENTLPF is slower than Choco ALLDIFFERENT that uses a dedicated algorithm. It is also on average three times slower than RCF. We can however see that it achieves arc consistency as it explores the same number of nodes than Choco ALLDIFFERENT. When $n = 15$, RCF explores approximately twice the number of nodes of Choco ALLDIFFERENT or ALLDIFFERENTLPF. RCF does not

n	5			10			15		
Filtering	CPU	N	N/s	CPU	N	N/s	CPU	N	N/s
Choco AllDifferent	0.1	3190.1	51453.2	1.2	51591.0	41405.3	5.1	87680.6	17244.9
AllDifferentLPF	2.7	3190.1	1184.6	110.2	51591.0	468.3	478.2	87680.6	183.4
RCF	0.9	3223.0	3679.2	40.7	57679.4	1416.6	292.6	159399.0	544.7
DEC	0.0	3285.6	547600.0	0.8	126613.2	150015.6	161.5	14338395.8	88805.2

Table 4.1: QuasiGroup Completion: filtering the AllDifferent constraint

achieve arc consistency, yet filters more than the decomposition alone. DEC propagates small constraints and is faster than RCF but can explore up to a 100 times the number of nodes of RCF for these instances.

4.6.2 Filtering one Sequence constraint

The LP-based filtering is implemented for Sequence with the polytope given in section 4.5.2. It is referred to as SequenceLPF and compared with an encoding of Sequence: the PS encoding presented in [29] which achieves arc consistency using a decomposition based on partial sums with $O(nk^2)$ constraints. Following the experimentation of [63] we generated 20 instances of a single sequence for each combination of $n \in \{500, 1000, 2000, 3000, 4000, 5000\}$, $k \in \{5, 15, 50, 75\}$ and $\Delta = l - u \in \{1, 5\}$. We look for the first solution found with a heuristic that randomly chooses the variable and the value to branch on. Figure 4.1 shows the evolution of the resolution time (in s) with the size of the instances.

SequenceLPF is slower than the PS encoding on sequences with smaller k. However, when $k = 75$, PS runs out of memory due to the number of constraints, whereas SequenceLPF can solve the sequence. Most importantly, as we can see on Figure 4.1, LP filtering scales better with the problem: it does not seem very sensitive to the value of k.

4.6.3 The Car-sequencing problem

We evaluate the performance of SequenceLPF on the Car-sequencing problem [87]. The goal is to schedule n cars partitioned in k classes: the demand of a class c is denoted by d_c. Each class requires a subset of options in a set of m options. For each option j, there must be at most p_j cars that require that option in each sub-sequence of size q_j. We consider the two first sets of instances presented in [86]: Set 1 is composed of 70 feasible instances with 200 cars, Set 2 is composed of 4 feasible instances with 100 cars. All the instances are available in CSPLib. We use the model described by [86]:

Variables

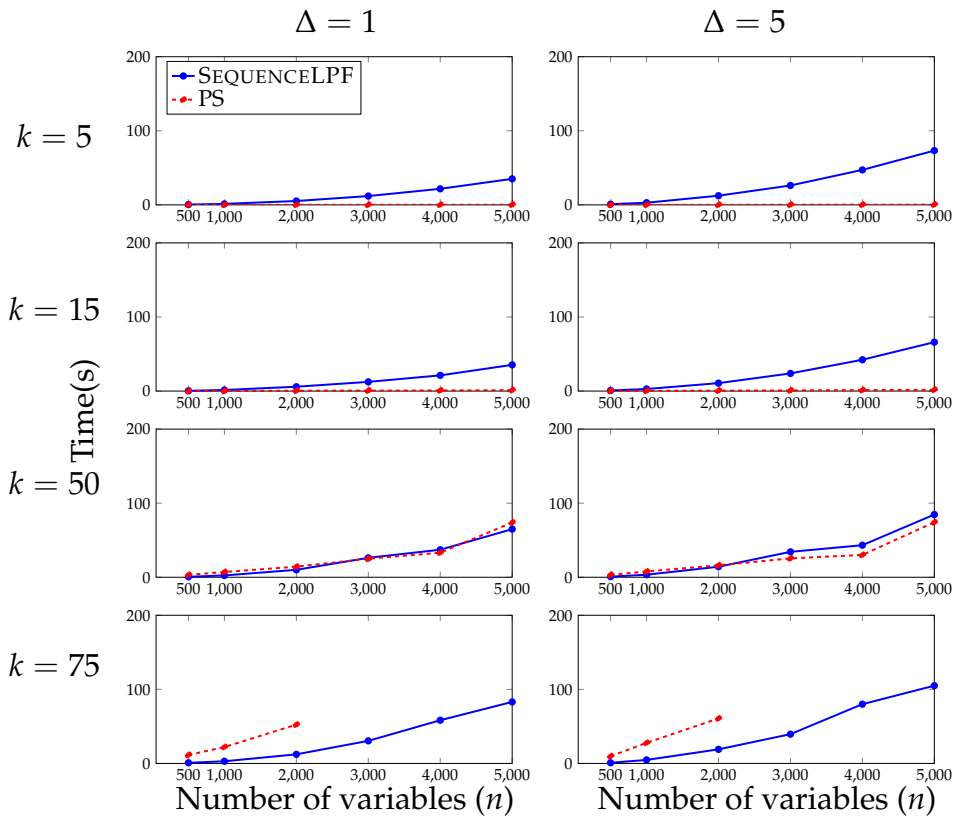

Figure 4.1: Sequence

- The class variables are n integer variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ taking their value in $\{1, \ldots, k\}$. $X_i = c$ means that a car of class c is scheduled in slot i.
- The option variables are nm boolean variables $\mathcal{Y} = \{Y_{1,1}, \ldots, Y_{n,m}\}$. $Y_{i,j} = 1$ means that the car in slot i has the option j.

Constraints

- The demand constraint states that the demand of each class must be satisfied and is enforced by GLOBALCARDINALITY($\mathcal{X}$, $\{d_1, \ldots, d_k\}$), meaning that for each class $c \in [1, n]$ the number of variables in $\mathcal{X}$ taking the value c should be exactly d_c.
- The capacity constraints: for each option j, in each sub-sequence of cars of size q_j, the number of cars requiring option j is limited by p_j. For each option j, we set: SEQUENCE($0, p_j, q_j, \{Y_{1,j}, \ldots, Y_{n,j}\}$)
- The option and class variables are linked using implication constraints. For

each class c, we define O_c the set of options required by the class. For each slot i, we set: $X_i = c \Rightarrow Y_{i,j} = 1 \ \forall j \in O_c$ and $X_i = c \Rightarrow Y_{i,j} = 0 \ \forall j \notin O_c$.

The problem is to find a single feasible solution and the search is done with a branching heuristic defined in [86] denoted by $\{class, lex, \delta, \leq_\Sigma\}$. It branches lexicographically on the class variables and chooses the class for which the sum of the loads of each option is the highest. We set a time limit of 1200s. Table 4.2 compares a model with the filtering of SEQUENCELPF to a model with the PS encoding. The columns CST and VAR show the average number of constraints and variables of the model.

	Set 1 (70 instances)						Set 2 (4 instances)					
MODEL	CST	VAR	SOLV	CPU	N	N/s	CST	VAR	SOLV	CPU	N	N/s
SEQUENCELPF	1006	1212	100 %	0.4	185.4	418.4	506	610	20.0 %	900.0	603588.8	670.7
PS	11872	4786	100 %	0.1	185.4	2948.9	5872	2384	20.0 %	900.0	10666137.5	11850.8

Table 4.2: Car-sequencing Sets 1 and 2

The branching heuristic is very efficient for the first set of instances whereas only 1 instance out of the 4 of Set 2 is solved. The SEQUENCE constraints are not very big ($n \leq 200$ and $k \leq 5$), hence PS has no more than 12000 constraints and is faster than SEQUENCELPF. For this more complex problem, LP filtering is only 20 times slower than the encoding that achieves arc consistency while for QuasiGroup Completion, it is 100 times slower than the Choco ALLDIFFERENT constraint.

4.7 Conclusion and future work

Given a formulation of a constraint with the integrality property, we have shown that arc consistency can be achieved by solving a single linear program. We believe it is very useful to provide arc consistency algorithms for numerous polynomial constraints that are not available in solvers. Although it is unlikely to be competitive with dedicated and incremental filtering algorithms, a number of improvements have yet to be investigated. Firstly, the algorithm boils down to the search for an interior point in a polytope and there might be more efficient techniques, although maybe more difficult to implement, than the simplex algorithm for that purpose. Secondly, the result itself is more general than the specific usage done here to enforce arc consistency since it can be used to detect integer variables necessarily grounded to a bound of their domain. This raises the question whether more sparse LP formulations that does not necessarily introduce a variable per value of the original domains can be used. Finally, polynomial constraints with high running

times often have a cost variable, for instance the MINIMUMWEIGHTALLDIFFERENT, so that a natural extension of this work is to handle an objective function.

4.8 What if the integrality property is not met?

In perspective of the work we did with LPF, arises the question of the validity of the LPF procedure when we no longer assume the integrality property. We will see that our LP filtering technique does not provide arc-consistency anymore but is still valid. This means that all the values that are filtered are indeed inconsistent values but all the inconsistent values are not filtered. We define theorem 6 and show that theorem 5 is a particular case then give two counter examples to show the LPF procedure is incomplete.

4.8.1 On the validity of the LPF procedure

Theorem 6 is an adaptation of theorem 5 with no restriction on the polytope. We also show that $\varepsilon = \frac{1}{T}$ is the biggest ε that can be taken.

Theorem 6 *Let $\mathcal{R} = \{x \in \mathbb{R}^T \mid Ax \geq b, l \leq x \leq u\}$ be a polytope of $\mathbb{R}^T$, with $l, u \in \mathbb{Z}^T$. Let $\mathcal{S} = \mathcal{R} \cap \mathbb{Z}^T$ be the set of integer points of $\mathcal{R}$. Let (P) be the following linear program with $\varepsilon = \frac{1}{T}$,*

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} e_t & & \\ s.t. \quad & e_t \geq \varepsilon + l_t - x_t & & \forall\, t \in \{1, ..., T\} \\ & e_t \geq \varepsilon - u_t + x_t & & \forall\, t \in \{1, ..., T\} \\ & e_t \geq 0 & & \forall\, t \in \{1, ..., T\} \\ & x \in \mathcal{R} & & \end{aligned}$$

For any (x^*, e^*) optimal solution of (P):

- *If $l_t \leq x_t^* < l_t + \varepsilon$ then $\hat{x}_t = l_t$ for all $\hat{x} \in \mathcal{S}$*
- *If $u_t - \varepsilon < x_t^* \leq u_t$ then $\hat{x}_t = u_t$ for all $\hat{x} \in \mathcal{S}$*

Remark 1: If (x^*, e^*) is an optimal solution of (P), then $e^* = e(x^*)$ with $e : \mathbb{R}^T \to \mathbb{R}^T$ defined by

$$\forall\, t \in \{1, \ldots, T\}$$

$$e(x) = (e_1(x), \ldots, e_t(x), \ldots, e_T(x)) \quad \text{and} \quad e_t(x) = \max\{0, \varepsilon + l_t - x_t, \varepsilon - u_t + x_t\}$$

Remark 2: For all $x \in \mathcal{R}$, $(x, e(x))$ is a feasible solution of (P).

Remark 3: For all $t \in \{1, \ldots, T\}$, e_t is a convex function. Figure 4.2 shows the variations of e_t.

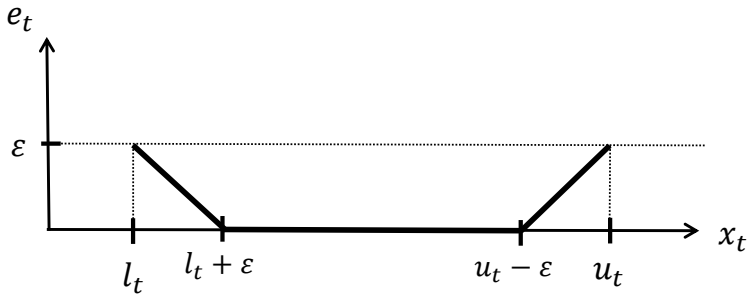

Figure 4.2: The function e_t is convex

We now prove theorem 6.

Proof. The idea of the proof is first to suppose there exists an optimal solution x^* of (P) with a variable $x^*_{t_0}$ that is in $[l_{t_0}, l_{t_0} + \varepsilon[$ and an integer point $\hat{x}$ of $\mathcal{R}$ where $\hat{x}_{t_0}$ is not set to l_{t_0}. We then find another solution $\bar{x}$ of (P) of cost strictly lower than x^* which contradicts its optimality and therefore the first assumption. We construct $\bar{x}$ as a convex combination of x^* and $\hat{x}$.

Let us suppose there exists an optimal solution (x^*, e^*) of (P), $t_0 \in \{1, \ldots, T\}$ and an integer solution $\hat{x} \in \mathcal{S}$ such that $l_{t_0} \leq x^*_{t_0} < l_{t_0} + \varepsilon$ and $\hat{x}_{t_0} \neq l_{t_0}$. We define $\alpha = (l_{t_0} + \varepsilon - x^*_{t_0})/(\hat{x}_{t_0} - x^*_{t_0})$. Since $\hat{x}_{t_0} \in \mathbb{Z}$ and $\hat{x}_{t_0} > l_{t_0}$, we deduce that $\hat{x}_{t_0} \geq l_{t_0} + 1 > l_{t_0} + \varepsilon > x^*_{t_0}$ therefore $\alpha \in]0, 1[$.

Take $\bar{x} = (1 - \alpha)x^* + \alpha\hat{x} = x^* + \alpha(\hat{x} - x^*)$. Note that $\bar{x} \in \mathcal{R}$ as it is a convex combination of $x^*, \hat{x} \in \mathcal{R}$. Let us now bound the cost of $\bar{x}$ by bounding the cost of each $e_t(\bar{x})$.

For all $t \in \{1, \ldots, T\} \setminus \{t_0\}$, by convexity of e_t:

$$e_t(\bar{x}) \;\leq\; (1 - \alpha)e_t(x^*) + \alpha e_t(\hat{x}) \;=\; e^*_t + \alpha(e_t(\hat{x}) - e^*_t) \;\leq\; e^*_t + \alpha\varepsilon$$

For $t = t_0$ we have

$$\bar{x}_{t_0} \;=\; x^*_{t_0} + \alpha(\hat{x}_{t_0} - x^*_{t_0}) \;=\; l_{t_0} + \varepsilon \;\text{ which implies }\; e_{t_0}(\bar{x}) \;=\; 0$$

Furthermore by definition of e^* and α: $e^*_{t_0} = l_{t_0} + \varepsilon - x^*_{t_0} = \alpha(\hat{x}_{t_0} - x^*_{t_0})$
and by definition of t_0: $\hat{x}_{t_0} - x^*_{t_0} > (l_{t_0} + 1) - (l_{t_0} + \varepsilon) = 1 - \varepsilon$

We deduce that

$$e_{t_0}(\bar{x}) \;=\; 0 \;=\; e^*_{t_0} - \alpha(\hat{x}_{t_0} - x^*_{t_0}) \;<\; e^*_{t_0} - \alpha(1-\varepsilon)$$

Hence

$$\sum_{t=1}^{T} e_t(\bar{x}) \;<\; \sum_{t\neq t_0} (e^*_t + \alpha\varepsilon) + e^*_{t_0} - \alpha(1-\varepsilon) \;=\; \sum_{t=1}^{T} e^*_t + \alpha(T\varepsilon - 1) \;=\; \sum_{t=1}^{T} e^*_t$$

Conclusion: (x^*, e^*) was not optimal since $(\bar{x}, e(\bar{x}))$ is a feasible solution of (P) and of better quality hence the contradiction.

To consider the case, $u_{t_0} - \varepsilon < x^*_{t_0} \leq u_{t_0}$ we apply a similar reasoning and define α by $\alpha = (\varepsilon - u_{t_0} + x^*_{t_0})/(x^*_{t_0} - \hat{x}_{t_0})$. □

Corollary 2 *Any $0 < \varepsilon \leq \frac{1}{T}$ can be taken.*

$0 < \varepsilon \leq \frac{1}{T}$ works since at the end of the proof of theorem 6 we just need that $\alpha(T\varepsilon - 1) \leq 0$. Note that we proved theorem 5 with $\varepsilon = \frac{1}{T+1}$ but thanks to corollary 2 any $0 < \varepsilon \leq \frac{1}{T}$ works. From the point of view of numerical analysis, we want to have the biggest ε to avoid numerical errors when computing the LP with a lot of variables.

Corollary 3 *Theorem 5 is a consequence of theorem 6.*

Take the polytope $\mathcal{Q} = \{x \in \mathbb{R}^T \mid Ax \geq b, l \leq x \leq u\}$ and assume it has integer extreme points and $l, u \in \mathbb{Z}^T$. Let us also denote by $\mathcal{S}$ the set of integer points of $\mathcal{Q}$. Corollary 3 is true as if $x^*_t = l_t$ then $l_t \leq x^*_t < l_t + \varepsilon$ therefore $\hat{x}_t = l_t$ for all $\hat{x} \in \mathcal{S}$, and if $\hat{x}_t = l_t$ for all $\hat{x} \in \mathcal{S}$ then $x^*_t = l_t$ as x^* is a convex combination of points of extreme points of $\mathcal{Q}$ that are in $\mathcal{S}$ by assumption. The same reasoning applies to u_t.

A small example. Let us consider a small example to show the application of theorem 6. Take the following set of constraints:

$$x_1 + x_2 = x_3 \tag{4.8}$$
$$x_1 + x_2 \leq 1.25 \tag{4.9}$$
$$x_1 + x_2 \geq 0.75 \tag{4.10}$$
$$x_1, x_2, x_3 \in \{0, 1\} \tag{4.11}$$

It is easy to see that x_3 must be equal to 1 and that x_1 or x_2 must be equal to 1, the other one to 0. We define $\epsilon = \frac{1}{3}$. An optimal solution is $\{\frac{5}{12}, \frac{1}{3}, \frac{3}{4}\}$ and $e^* = \{0, 0, 1/12\}$. We see that x_1 cannot be fixed since $l_1 + \varepsilon \leq x^*_1 \leq u_1 - \varepsilon$ ($\frac{1}{3} \leq \frac{5}{12} \leq \frac{2}{3}$) and that x_2 cannot be fixed since $l_2 + \varepsilon \leq x^*_2 \leq u_2 - \varepsilon$ ($\frac{1}{3} \leq \frac{1}{3} \leq \frac{2}{3}$). x_3 has to be fixed since $u_3 - \varepsilon < x^*_3$ ($\frac{2}{3} < \frac{3}{4}$).

4.8.2 Arc-consistency is not achieved

We show via two counter-examples that the LPF procedure does not necessarily achieve arc-consistency when the conjunction of constraints does not have the integrality property. We first look at a small feasibility example then consider the case of MINIMUMWEIGHTALLDIFFERENT, a global constraint with a cost function.

Example 1. Take the following set of constraints:

$$x_1 + x_2 = x_3 \tag{4.12}$$
$$x_1 + x_2 \leq 1.5 \tag{4.13}$$
$$x_1 + x_2 \geq 0.5 \tag{4.14}$$
$$x_1, x_2, x_3 \in \{0, 1\} \tag{4.15}$$

It is easy to see that x_3 must be equal to 1 and that x_1 or x_2 must be equal to 1, the other one to 0. We define $\epsilon = \frac{1}{3}$ and when the LPF procedure is applied, we see that each of the x_i variable can be offset from its bound by at least ϵ as shown in an optimal solution $\{\frac{1}{3}, \frac{1}{3}, \frac{2}{3}\}$. This means that one cannot deduce that x_3 must be equal to 1.

Example 2. Consider now an instance of MINIMUMWEIGHTALLDIFFERENT:

$$\text{MINIMUMWEIGHTALLDIFFERENT}(X, Y, Z, \{1, 3, 3, 2, 0, 0, 3\}, 7) \tag{4.16}$$
$$X \in \{1, 2\}, Y \in \{1, 2\}, Z \in \{1, 2, 3\} \tag{4.17}$$

This constraints forces X, Y and Z to take different values while restraining the sum of the affectations by an upper bound $UB = 7$. Affecting value 1 to X costs 1, value 2 for X costs 3, value 1 for Y costs 3 and so on. Figure 4.3 shows a graph representing the example. To apply the LPF procedure we need to channel X, Y, Z to the binary variables $\{x_1, x_2, y_1, y_2, z_1, z_2, z_3\}$ and build the LP with $\varepsilon = \frac{1}{7}$. For instance the variable $x_1 = 1$ if and only if X takes the value 1. The constraint on the objective is $1x_1 + 3x_2 + 3y_1 + 2y_2 + 0z_1 + 0z_2 + 3z_3 \leq 7$. An optimal solution of the LP is $\{\frac{6}{7}, \frac{1}{7}, \frac{1}{7}, \frac{6}{7}, 0, 0, 1\}$. One can see that variable Z is forced to take value 3 since z_3 is set to 1. The two other variables X and Y on the other hand are not fixed by the LP since the channeled variables $\{x_1, x_2, y_1, y_2\}$ are unstuck from the bounds of at least ε. Note that having $X = 2$ or $Y = 1$ costs 9 and should be forbidden as it exceeds UB whereas having $X = 1$ and $Y = 2$ costs only 6 and is feasible. The LPF procedure does not remove all inconsistent values in this case.

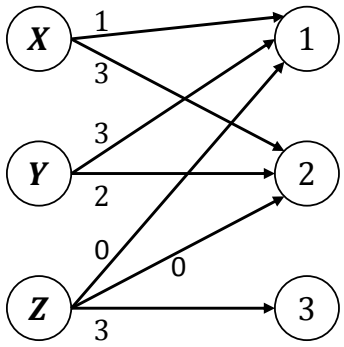

Figure 4.3: An example with MINIMUMWEIGHTALLDIFFERENT

Conclusion and future work

In this thesis we discuss the use of constraint programming to solve production planning problems and more precisely lot-sizing problems. We gradually build a CP framework and apply it on increasingly complex lot-sizing problems. We introduce a global constraint LOTSIZING defined by the conjunction of the constraints of the single-level single-item lot-sizing problem. The core problem is very generic and considers time-varying costs and production and inventory bounds. This choice of global constraint is motivated by the intuitive modeling it allows and by the amount of work that has been done on the problem for more than sixty years. For each problem considered in this work, we have endeavored to propose several models and provide a comparison with relevant existing methods. The contributions made in this thesis are the following.

A new global constraint LOTSIZING. We define the global constraint LOTSIZING for the single-level single-item lot-sizing problem. The constraint is equivalent to finding a feasible production plan that satisfies upper bounds on the overall production, inventory, setup and global costs. The global constraint is NP-hard and we show several consistency results: bound-consistent domains for the feasibility aspect are easily found but when considering the upper bounds on the costs, it becomes exponential. We also show that the filtering algorithms can be simplified as we transform the problem into an equivalent problem without lower bounds.

Strong filtering algorithms. We derive reasoning mechanisms from existing dynamic programming and flow-based algorithms. The idea is to exploit resource-based filtering in the graphs of the states of the DP algorithms to filter the variables using the different upper bounds on the costs. LOTSIZING is a NP-hard global constraint hence the use of pseudo-polynomial DP algorithms to filter.

The adaptation of the filtering when the DP does not scale. As DP algorithms can rapidly reach their computational and memory limits when the size of the instance increases, we show a way to adapt the filtering. We decompose the problem into sub-problems, compute the DP on small enough sub-problems and use an optimal solution of the weighted interval scheduling problem to filter the variables.

Filtering improvements with redundancy. We improve the strength of the global constraint with the use of redundant cardinality variables, at the expense of the resolution speed. Cardinality variables conveniently allow us to count the number of successive open periods. When branching on the variable $card = \sum_{t=1}^{T} Y_t$ first, then on the setup variables Y, the filtering algorithms benefit from the fact that the number of open periods is fixed.

An extension of LOTSIZING to model unitary costs with step functions. The structure of LOTSIZING turns out to be generic enough for us to naturally extend its scope and tackle piece-wise linear unitary production and inventory costs. The problem modeled remains a single-item lot-sizing but the modeling possibilities are improved as it allows the user to consider different costs functions. We also show that these costs are helpful to model a redundant constraint for multi-item problems.

Numerical tests on different lot-sizing problems. We implemented several CP models for each problem considered. On single-level single-item lot-sizing problems, we show that the naive decomposed CP model cannot be used but that the LOTSIZING-based CP model can solve the problem in reasonable time. We show that the adaptation of the propagation algorithms of LOTSIZING when the DP is not scalable is useful on big instances. Finally we study three problems with side constraints and show that either the classical DP algorithm or the MILP models can be less effective whereas the CP model can solve the problem.

On multi-item problems, we show that when the size of the problem is increased, the LOTSIZING-based models are slowed due to the important computation time needed for the DP algorithms but that they provide strong reasonings and in particular a sharp lower bound.

A new generic LP-based filtering algorithm Implementing and testing a lot of ideas and filtering mechanisms for LOTSIZING for three years, lead to a captivating improvement of a filtering algorithm using linear programming. We present a generic filtering algorithm that aims at improving the process of global constraint design and prototyping reasoning mechanisms. The technique filters the variables of any conjunction of constraints with a single call to a LP solver. In particular if

the set of constraints has an ideal formulation, it provides arc-consistent domains. When applied to the LOTSIZING constraint, it could be useful in order to propagate more reasoning on the flow or multi-flow problems during the search and provide arc consistency for the SEQUENCE constraints defined by the cardinality variables.

As a general conclusion, we saw that constraint programming needs a lot of work to be competitive. Basic decomposed models do not perform well but the LOTSIZING global constraint we developed is a relevant tool towards a constraint programming solver for production planning problems. This constraint simplifies the model description as it can represent each node of a lot-sizing network for which it computes strong filtering. The extension with variable unitary costs is very useful to model problems with piece-wise linear costs and to provide global redundant constraints and relaxations when the network is complex. The DP-based filtering algorithms perform well on single-level single-item problems with side constraints. The LOTSIZING-based models provide good lower bounds but are less effective on pure multi-item problems compared to MILP models.

Thanks to several experiments, we believe that a production planning CP solver should avoid branching on the quantity variables. The quantities can indeed be very important and to speed up the search the branching should occur only on binary variables. Fortunately, we have seen that once the binary variables are fixed, there are many cases where the resulting problem is quickly solvable. Furthermore, we think that LOTSIZING-based models should use global redundant constraints to capture the problem as a whole. The communication between the nodes of the network is sometimes not enough to provide strong reasoning.

The work on solving production planning problem with CP is still preliminary but given the results of this thesis, we strongly believe that complex lot-sizing problems with combinatorial side constraints can be solved using constraint programming.

Finally, the LPF procedure is a very good way to show that linear programming can successfully be combined with constraint programming to provide powerful global reasoning mechanisms. We believe that this is the key contribution of this thesis to CP. It is also very meaningful to the lot-sizing domain where a lot of subproblems related to flow and sequences are very well handled by LP.

Future work

As we tackled classical lot-sizing problems in the light of constraint programming, several perspectives of this work arise. We chose to highlight three important points.

Extension of the scope of LotSizing. Abstracting the scope of the global constraint can allow the user not only to easily model a complex problem but also to take into account more features in the propagation of LotSizing. Take the example of multi-level lot-sizing. The fact that multi-level usually represents a network poses some modeling challenges when trying to model each node with a LotSizing constraint. In this network, the output of one node is the input of the next. However only the input of a node is modeled with variables – the quantity variables –, its output is constant – the demands. We thing that extending the scope of LotSizing to take into account variable demands could improve the communication between the constraints in a complex network. Similarly, it would be interesting to add the possibility to model backlogs and lost sales. These constraints are usually modeled with negative inventory.

We started to study the one warehouse multi-retailers problem which is one of the simplest multi-level problem. We modeled it with LotSizing with constant demands. The first results are not compelling since the MILP models are very effective on these pure problems without side constraints.

Symmetries in lot-sizing. One of the main ideas that is not presented in the thesis is that symmetry could be a key factor to solve lot-sizing problems faster. Symmetries occur between the items in multi-item for instance: once a period is open, which item produce? Finding those symmetries and arbitrating them could efficiently reduce the search space. We tried introducing the $y_{t,t'}$ variables that equal 1 if the demand at t' is satisfied by a production at t but did not manage to solve the symmetry problem.

Improving the LP-based filtering. The last point revolves around the linear programming filtering technique presented in chapter 4. This is an interesting and useful improvement for the CP community and should be further analyzed.

First, there might be more efficient techniques, although maybe more difficult to implement, than the simplex algorithm to find an interior point.

Second, one can look for an encoding of the domains that does not represent each value for each variable to achieve bound consistency, therefore improving the speed of the algorithm.

The third step relates to polynomial constraints with an ideal formulation and an objective function. When building the linear program for LPF, the objective function is set as a constraint and the global formulation is therefore no longer ideal. Can the procedure be adapted to achieve arc-consistency in this case?

Finally, on the application to lot-sizing problems, it would be relevant to evaluate the strength of the filtering of the lot-sizing underlying flow problems at each node of the search.

Bibliography

[1] Nabil Absi, Boris Detienne, and Stéphane Dauzère-Pérès, *Heuristics for the multi-item capacitated lot-sizing problem with lost sales*, Computers & Operations Research **40** (2013), no. 1, 264–272.

[2] Nabil Absi and Safia Kedad-Sidhoum, *The multi-item capacitated lot-sizing problem with setup times and shortage costs*, European journal of operational research **185** (2008), no. 3, 1351–1374.

[3] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter, *Constraint integer programming: A new approach to integrate cp and mip*, Integration of AI and OR techniques in constraint programming for combinatorial optimization problems (2008), 6–20.

[4] ———, *Constraint integer programming: A new approach to integrate cp and mip* (2008), 6–20.

[5] Alok Aggarwal and James K. Park, *Improved algorithms for economic lot size problems*, Operations Research **41** (1993), no. 3, 549–571.

[6] Abderrahmane Aggoun and Nicolas Beldiceanu, *Extending chip in order to solve complex scheduling and placement problems*, Mathematical and computer modelling **17** (1993), no. 7, 57–73.

[7] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin, *Network flows: theory, algorithms, and applications* (1993).

[8] Ayse Akbalik and Christophe Rapine, *Polynomial time algorithms for the constant capacitated single-item lot sizing problem with stepwise production cost*, Operations Research Letters **40** (2012), no. 5, 390–397.

[9] Yash P Aneja, Vijay Aggarwal, and Kunhiraman PK Nair, *Shortest chain subject to side constraints*, Networks **13** (1983), no. 2, 295–302.

[10] Ionuţ Aron, John N. Hooker, and Tallys H. Yunes, *Simpl: A system for integrating optimization techniques* (2004), 21–36.

[11] Alper Atamtürk and Simge Küçükyavuz, *An O(n^2) algorithm for lot sizing with inventory bounds and fixed costs*, Operations Research Letters **36** (2008), no. 3, 297–299.

[12] Fahiem Bacchus, *Gac via unit propagation*, International conference on principles and practice of constraint programming, 2007, pp. 133–147.

[13] Imre Barany, Tony J. Van Roy, and Laurence A. Wolsey, *Strong formulations for multi-item capacitated lot sizing*, Management Science **30** (1984), no. 10, 1255–1261.

[14] John E Beasley and Nicos Christofides, *An algorithm for the resource constrained shortest path problem*, Networks **19** (1989), no. 4, 379–394.

[15] Nicolas Beldiceanu and Mats Carlsson, *Revisiting the cardinality operator and introducing the cardinality-pathconstraint family*, International conference on logic programming, 2001, pp. 59–73.

[16] Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon, *Global constraint catalog, (revision a)*, Swedish Institute of Computer Science, 2012.

[17] Nicolas Beldiceanu and Evelyne Contejean, *Introducing global constraints in chip*, Mathematical and computer Modelling **20** (1994), no. 12, 97–123.

[18] ———, *Introducing global constraints in chip*, Mathematical and computer Modelling **20** (1994), no. 12, 97–123.

[19] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas, *Dynamic programming and optimal control*, Vol. 1, Athena scientific Belmont, MA, 1995.

[20] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh, *Filtering algorithms for the nv alue constraint*, Constraints **11** (2006), no. 4, 271–293.

[21] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh, *The complexity of global constraints*, Aaai, 2004, pp. 112–117.

[22] Gabriel R. Bitran and Horacio H. Yanasse, *Computational complexity of the capacitated lot size problem*, Management Science **28** (1982), no. 10, 1174–1186.

[23] Alexander Bockmayr and Thomas Kasper, *Branch and infer: A unifying framework for integer and finite domain constraint programming*, INFORMS Journal on Computing **10** (1998), no. 3, 287–300.

[24] Alexander Bockmayr and Nicolai Pisaruk, *Solving assembly line balancing problems by combining ip and cp*, arXiv preprint cs (2001).

[25] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais, *Boosting systematic search by weighting constraints*, Proceedings of the 16th european conference on artificial intelligence, 2004, pp. 146–150.

[26] Nadjib Brahimi, Nabil Absi, Stéphane Dauzere-Peres, and Atle Nordli, *Single-item dynamic lot-sizing problems: An updated survey*, European Journal of Operational Research (2017).

[27] Nadjib Brahimi, Stéphane Dauzère-Pérès, and Najib M Najid, *Capacitated multi-item lot-sizing problems with time windows*, Operations Research **54** (2006), no. 5, 951–967.

[28] Nadjib Brahimi, Stéphane Dauzere-Peres, Najib M Najid, and Atle Nordli, *Single item lot sizing problems*, European Journal of Operational Research **168** (2006), no. 1, 1–16.

[29] Sebastian Brand, Nina Narodytska, Claude-Guy Quimper, Peter Stuckey, and Toby Walsh, *Encodings of the sequence constraint*, International conference on principles and practice of constraint programming, 2007, pp. 210–224.

[30] Mats Carlsson, Johan Widen, Johan Andersson, Stefan Andersson, Kent Boortz, Hans Nilsson, and Thomas Sjöland, *Sicstus prolog user's manual*, Vol. 3, Swedish Institute of Computer Science Kista, Sweden, 1988.

[31] Vašek Chvátal, *Linear programming*, Series of books in the mathematical sciences, W.H. Freeman, 1983.

[32] Tobias Dantzig, *Number: The language of science*, 1930.

[33] Eric V Denardo, *Dynamic programming: models and applications*, Courier Corporation, 2012.

[34] Andreas Drexl and Alf Kimms, *Lot sizing and scheduling survey and extensions*, European Journal of Operational Research **99** (1997), no. 2, 221–235.

[35] Stuart E. Dreyfus and Averill M. Law, *Art and theory of dynamic programming*, Academic Press, Inc., Orlando, FL, USA, 1977.

[36] Gary D Eppen and R Kipp Martin, *Solving multi-item capacitated lot-sizing problems using variable redefinition*, Operations Research **35** (1987), no. 6, 832–848.

[37] Awi Federgruen, Joern Meissner, and Michal Tzur, *Progressive interval heuristics for multi-item capacitated lot-sizing problems*, Operations Research **55** (2007), no. 3, 490–502.

[38] Awi Federgruen and Michal Tzur, *A simple forward algorithm to solve general dynamic lot sizing models with n periods in* O(*n* log *n) or* O(*n) time*, Management Science **37** (1991), no. 8, 909–925.

[39] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen, *An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems*, Networks **44** (2004), no. 3, 216–229.

[40] Bernhard Fleischmann, *The discrete lot-sizing and scheduling problem*, European Journal of Operational Research **44** (1990), no. 3, 337–348.

[41] Michael Florian and Morton Klein, *Deterministic production planning with concave costs and capacity constraints*, Management Science **18** (1971), no. 1, 12–20.

[42] Filippo Focacci, Andrea Lodi, and Michela Milano, *Cost-based domain filtering*, Principles and practice of constraint programming - cp'99, 5th international conference, alexandria, virginia, usa, october 11-14, 1999, proceedings, 1999, pp. 189–203.

[43] _____, *Embedding relaxations in global constraints for solving tsp and tsptw*, Ann. Math. Artif. Intell. **34** (2002), no. 4, 291–311.

[44] Céline Gicquel, Michel Minoux, and Yves Dallery, *Capacitated lot sizing models: a literature review* (2008).

[45] Stefano Gualandi and Federico Malucelli, *Resource constrained shortest paths with a super additive objective function.*, Cp, 2012, pp. 299–315.

[46] Youssef Hamadi, Eric Monfroy, and Frédéric Saubion, *What is autonomous search?*, Hybrid optimization, 2011, pp. 357–391.

[47] Ford W Harris, *How many parts to make at once*, Factory, the Magazine of Management **10** (1913), 135–136.

[48] Bertrand Hellion, Fabien Mangione, and Bernard Penz, *A polynomial time algorithm for the single-item lot sizing problem with capacities, minimum order quantities and dynamic time windows*, Operations Research Letters **42** (2014), no. 8, 500–504.

[49] _____, *Stability contracts between supplier and retailer: a new lot sizing model*, International Journal of Production Research **53** (2015), no. 1, 1–12.

[50] John. N. Hooker, *Operations research methods in constraint programming*, Handbook of constraint programming, 2006.

[51] John N. Hooker and Hong Yan, *A relaxation of the cumulative constraint*, Principles and practice of constraint programming - CP 2002, 8th international conference, CP 2002, ithaca, ny, usa, september 9-13, 2002, proceedings, 2002, pp. 686–690.

[52] Vinasétan Ratheil Houndji, Pierre Schaus, Laurence A. Wolsey, and Yves Deville, *The stocking-cost constraint*, Principles and practice of constraint programming, 2014, pp. 382–397.

[53] Vipul Jain and Ignacio E Grossmann, *Algorithms for hybrid milp/cp models for a class of optimization problems*, INFORMS Journal on computing **13** (2001), no. 4, 258–276.

[54] Behrooz Karimi, SMT Fatemi Ghomi, and JM Wilson, *The capacitated lot sizing problem: a review of models and algorithms*, Omega **31** (2003), no. 5, 365–378.

[55] Uday S Karmarkar and Linus Schrage, *The deterministic dynamic product cycling problem*, Operations Research **33** (1985), no. 2, 326–345.

[56] Jon Kleinberg and Éva Tardos, *Algorithm design*, Pearson Education India, 2006.

[57] Jakob Krarup and Ole Bilde, *Plant location, set covering and economic lot size: an o (mn)-algorithm for structured problems*, Numerische methoden bei optimierungsaufgaben **3** (1977), 155–180.

[58] Mikael Z Lagerkvist and Christian Schulte, *Propagator groups*, Principles and practice of constraint programming-cp 2009, 2009, pp. 524–538.

[59] Jean-Bernard Lasserre, *An integrated model for job-shop planning and scheduling*, Management Science **38** (1992), no. 8, 1201–1211.

[60] Jena-Lonis Lauriere, *A language and a program for stating and solving combinatorial problems*, Artificial intelligence **10** (1978), no. 1, 29–127.

[61] Christophe Lecoutre, Lakhdar Saïs, Sébastien Tabary, and Vincent Vidal, *Reasoning from last conflict (s) in constraint programming*, Artificial Intelligence **173** (2009), no. 18, 1592–1614.

[62] Stephen F. Love, *Bounded production and inventory models with piecewise concave costs*, Management Science **20** (1973), no. 3, 313–318.

[63] Michael Maher, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh, *Flow-based propagators for the sequence and related global constraints*, International conference on principles and practice of constraint programming, 2008, pp. 159–174.

[64] Guillaume Massonnet, *Algorithmes d'approximation pour la gestion de stock*, Ph.D. Thesis, 2013.

[65] George B Mathews, *On the partition of numbers*, Proceedings of the London Mathematical Society **1** (1896), no. 1, 486–490.

[66] Laurent Michel and Pascal Van Hentenryck, *Activity-based search for black-box constraint programming solvers*, Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (2012), 228–243.

[67] Andrew J Miller and Laurence A Wolsey, *Tight mip formulation for multi-item discrete lot-sizing problems*, Operations Research **51** (2003), no. 4, 557–565.

[68] George L. Nemhauser and Laurence A. Wolsey, *Integer and combinatorial optimization*, Wiley-Interscience, New York, NY, USA, 1988.

[69] François Pachet and Pierre Roy, *Automatic generation of music programs*, CP **99** (1999), 331–345.

[70] Gilles Pesant, *CSPLib problem 067: Quasigroup completion* (Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, and Ian P. Gent, eds.)

[71] Yves Pochet and Laurence A. Wolsey, *Production planning by mixed integer programming*, Springer Science & Business Media, 2006.

[72] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca, *Choco documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[73] Claude-Guy Quimper, *Private communication*, 2017.

[74] Claude-Guy Quimper and Alejandro López-Ortiz, *From linear relaxations to global constraint propagation*, International conference on principles and practice of constraint programming, 2005, pp. 867–867.

[75] Philippe Refalo, *Tight cooperation and its application in piecewise linear optimization* (1999), 375–389.

[76] ———, *Linear formulation of constraint programming models and hybrid solvers*, Principles and practice of constraint programming - CP 2000, 6th international conference, singapore, september 18-21, 2000, proceedings, 2000, pp. 369–383.

[77] Jean-Charles Régin, *A filtering algorithm for constraints of difference in csps*, Aaai, 1994, pp. 362–367.

[78] ———, *A filtering algorithm for constraints of difference in csps*, Aaai, 1994, pp. 362–367.

[79] ———, *Generalized arc consistency for global cardinality constraint*, Proceedings of the thirteenth national conference on artificial intelligence-volume 1, 1996, pp. 209–215.

[80] Jean-Charles Régin and Jean-François Puget, *A filtering algorithm for global sequencing constraints*, International conference on principles and practice of constraint programming, 1997, pp. 32–46.

[81] Jean-Charles Régin and Michel Rueher, *A global constraint combining a sum constraint and difference constraints*, Cp, 2000, pp. 384–395.

[82] Robert Rodosek, Mark G. Wallace, and Mozafar T. Hajian, *A new approach to integrating mixed integer programming and constraint logicprogramming*, Annals of Operations Research **86** (1999), no. 0, 63–87.

[83] Sheldon M Ross, *Introduction to stochastic dynamic programming*, Academic press, 2014.

[84] Domenico Salvagnin and Toby Walsh, *A hybrid mip/cp approach for multi-activity shift scheduling*, Principles and practice of constraint programming, 2012, pp. 633–646.

[85] Pierre Schaus, Jean-Charles Régin, Rowan Van Schaeren, Wout Dullaert, and Birger Raa, *Cardinality reasoning for bin-packing constraint: Application to a tank allocation problem.*, CP **7514** (2012), 815–822.

[86] Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet, *A study of constraint programming heuristics for the car-sequencing problem*, Engineering Applications of Artificial Intelligence **38** (2015), 34–44.

[87] Barbara Smith, *CSPLib problem 001: Car sequencing* (Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, and Ian P. Gent, eds.)

[88] Hartmut Stadtler and Christoph Kilger, *Supply chain management and advanced planning*, Concepts, Models, Software and Case Studies **4** (2008).

[89] Robin Steiger, Willem-Jan van Hoeve, and Radosław Szymanek, *An efficient generic network flow constraint*, Proceedings of the 2011 ACM Symposium on Applied Computing, 2011, pp. 893–900.

[90] Armagan Tarim and Ian Miguel, *Echelon stock formulation of arborescent distribution systems: An application to the wagner-whitin problem*, International conference on integration of artificial intelligence (ai) and operations research (or) techniques in constraint programming, 2004, pp. 302–318.

[91] Christian Timpe, *Solving planning and scheduling problems with combined integer and constraint programming*, OR spectrum **24** (2002), no. 4, 431–448.

[92] JA Tomlin, *Minimum-cost multicommodity network flows*, Operations Research **14** (1966), no. 1, 45–51.

[93] William W Trigeiro, L Joseph Thomas, and John O McClain, *Capacitated lot sizing with setup times*, Management science **35** (1989), no. 3, 353–366.

[94] Pascal Van Hentenryck and Jean-Philippe Carillon, *Generality versus specificity: An experience with ai and or techniques.*, Aaai, 1988, pp. 660–664.

[95] Stan Van Hoesel and Albert Peter Marie Wagelmans, *An* $O(T^3)$ *algorithm for the economic lot-sizing problem with constant capacities*, Management Science **42** (1996), no. 1, 142–150.

[96] Willem-Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal, *Revisiting the sequence constraint*, International conference on principles and practice of constraint programming, 2006, pp. 620–634.

[97] Willem-Jan Van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal, *New filtering algorithms for combinations of among constraints*, Constraints **14** (2009), no. 2, 273–292.

[98] Albert Wagelmans, Stan Van Hoesel, and Antoon Kolen, *Economic lot sizing: an* $O(n \log n)$ *algorithm that runs in linear time in the wagner-whitin case*, Operations Research **40** (1992), S145–S156.

[99] Harvey M. Wagner and Thomson M. Whitin, *Dynamic version of the economic lot size model*, Management science **5** (1958), no. 1, 89–96.

[100] RH Wilson, *A scientific routine for stock control*, Harvard business review **13** (1934), no. 1, 116–128.

[101] Laurence A. Wolsey, *Integer Programming*, Wiley-Interscience, 1998.

[102] Laurence A Wolsey, *Solving multi-item lot-sizing problems with an mip solver using classification and reformulation*, Management Science **48** (2002), no. 12, 1587–1602.

In this thesis we investigate the potential use of constraint programming to develop a production planning solver. We focus on lot-sizing problems that are crucial and challenging problems of the tactical level of production planning and use one of the main strengths of constraint programming, namely global constraints. The goal of this work is to set the grounds of a constraint programming framework for solving complex lot-sizing problems. We define a LOTSIZING global constraint based on a generic single-item, single-level lot-sizing problem that considers production and inventory capacities, unitary production and inventory costs and setup costs. This global constraint is an intuitive modeling tool for complex lot-sizing problems as it can model the nodes of lot-sizing networks. We use classical dynamic programming techniques of the lot-sizing field to develop powerful filtering algorithms for the global constraint. Furthermore we model multi-item problems that are natural extensions of the core problem.

Finally we introduce a new generic filtering algorithm based on linear programming. We show that arc consistency can be achieved with only one call to a linear programming solver when the global constraint has an ideal formulation and adapt the result to provide partial filtering when no restriction is made on the constraints. This technique can be useful to tackle polynomial lot-sizing underlying flow and sequence sub-problems.

Keywords: constraint programming, production planning, lot-sizing, global constraint, linear programming

Cette thèse a pour objectif d'étudier l'utilisation de la programmation par contraintes pour développer un solveur de planification de production. Nous nous concentrons sur des problèmes de dimensionnement de lots de production (lot-sizing) qui sont des problèmes majeurs et difficiles de la planification de la production et profitons d'une des principales forces de la programmation par contraintes, à savoir les contraintes globales. Nous définissons une contrainte globale LOTSIZING qui s'appuie sur un problème générique de lot-sizing mono-produit à un seul niveau, qui tient compte des capacités de production et de stockage, des coûts unitaires de production et de stockage et des coûts fixes. Cette contrainte globale est un outil de modélisation intuitif pour les problèmes complexes de lot-sizing car elle permet de modéliser chaque nœud des réseaux de distribution. Nous utilisons des techniques de programmation dynamique classiques du lot-sizing pour développer des algorithmes de filtrage pour la contrainte globale. Nous modélisons également des problèmes multi-produits.

Enfin, nous introduisons un nouvel algorithme de filtrage générique s'appuyant sur la programmation linéaire. Nous montrons que la cohérence d'arc pour les contraintes considérées peut être obtenue avec la résolution d'un seul programme linéaire lorsque la contrainte a une formulation idéale et nous généralisons le résultat pour faire du filtrage partiel lorsqu'aucune restriction n'est faite sur ces contraintes. Cette technique peut être pertinente lors de la résolution de sous-problèmes de flot ou de séquence sous-jacents au lot-sizing.

Mots clés : programmation par contraintes, planification de production, lot-sizing, contrainte globale, programmation linéaire