Une contrainte globale pour le lot sizing

Grigori German[1] **Hadrien Cambazard**[1] **Jean-Philippe Gayon**[1] **Bernard Penz**[1]

[1] Univ. Grenoble Alpes, G-SCOP, F-38000 Grenoble, France
CNRS, G-SCOP, F-38000 Grenoble, France
{prénom.nom}@grenoble-inp.fr

Résumé

Cet article présente les premiers éléments d'une approche par contraintes pour la résolution de problèmes de planification de type *lot sizing*. Un problème de *lot sizing* avec bornes inférieures et supérieures de production et de stockage est considéré, et abordé comme une contrainte globale. Cette dernière permet de calculer des bornes inférieures sur le coût global des solutions, mais aussi sur les coûts de setup, les coûts unitaires de production et les coûts de stockage et assure un filtrage efficace. Les tests numériques valident l'approche par rapport aux approches classiques en PLNE, ce qui permet d'envisager son utilisation dans la résolution de problèmes multi-produits difficiles. Mots-clés : *lot sizing*, contrainte globale, programmation par contraintes.

Abstract

This paper presents a constraint programming approach aiming at solving productions planning problems such as *lot sizing* problems. A *lot sizing* problem with lower- and upper bounds of production and storage is studied and considered as a global constraint. This constraint is able to compute cost-related lower bounds - specifically on the global cost, the setup cost, the variable production cost and the storage cost. Numerical tests show the competitiveness of the approach in comparison to classical linear programming approaches and allows us to consider its exploitation in the resolution of hard multi-item problems. Keywords : *lot sizing*, global constraint, constraints programming.

1 Introduction

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 *lot sizing*, introduit par [18], ont été largement étudiés. Beaucoup de ces problèmes sont polynômiaux et abordés par programmation dynamique. En présence de contraintes additionnelles, comme des capacités de production variables au cours du temps, ces problèmes deviennent NP-complets. Ces derniers sont souvent traités par programmation linéaire en nombres entiers. De nombreuses formulations linéaires ont été proposées [13, 3]. Plus récemment, des algorithmes d'approximation ont été proposés pour des variantes NP-difficiles du problème [15]. Très récemment, des travaux se sont intéressés à la résolution de problèmes de *lot sizing* par la programmation par contraintes [8]. 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 [8] relève cependant davantage de l'ordonnancement (détermination des dates de début des activités) alors que le présent papier considère un horizon temporel plus tactique (détermination des quantités à produire sur un horizon discrétisé), ce qui différencie les problèmes considérés.

L'objectif de cet article est d'introduire une contrainte globale pour un problème de *lot sizing* mono-produit relativement général, prenant en compte des capacités et des coûts variables au cours de l'horizon de planification. On considère à la fois des coûts de setup et les 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 nombreux problèmes multi-produits difficiles. Nous posons les bases algorithmiques pour la réalisation du filtrage. Des résultats expérimentaux préliminaires valident les bornes inférieures proposées en comparant l'approche avec des modèles de Programmation Linéaire en Nombres Entiers (PLNE). Le filtrage apporte

des informations intéressantes sur les domaines des variables de décision (production et stockage) ainsi que des bornes inférieures des coûts pris individuellement (coûts de setup de production, coûts variables de production, coûts de stockage).
L'article est structuré de la façon suivante : Une définition de la contrainte globale est proposée dans la section 3. Des algorithmes de filtrage sont ensuite présentés dans les sections 4 et 5. La description des expérimentations ainsi que l'analyse des premiers résultats obtenus font l'objet de la section 6. La section 7 illustre l'utilisation de la contrainte globale proposée dans cet article au sein de différents problèmes déjà abordés dans la littérature. Pour finir des pistes de recherche sont proposées dans la section 8.

2 Rappel sur la programmation par contraintes et notations générales

Un problème de satisfaction de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, est constitué d'un ensemble de variables $\mathcal{X}$, un ensemble de domaines $\mathcal{D}$ spécifiant un domaine (ensemble fini de valeurs) pour chaque variable et un ensemble $\mathcal{C}$ de contraintes. Une contrainte détermine les combinaisons autorisées de valeurs pour un sous-ensemble de variables. Les variables, *e.g* X_i, sont notées par des lettres majuscules. $\overline{X_i}$ et $\underline{X_i}$ dénotent respectivement les bornes supérieure et inférieure du domaine de X_i tandis que $D(X_i)$ désigne son domaine. Étant donnée une contrainte C sur un ensemble de variables $\langle X_1, \ldots, X_n \rangle$, un *support de bornes* pour C est un tuple $\langle v_1, \ldots, v_n \rangle$ de valeurs qui soit solution de / qui vérifie C et tel que $\underline{X_i} \leq v_i \leq \overline{X_i}$ pour toute variable X_i. Une variable X_i de C est dite cohérente aux bornes (BC) pour C si $\underline{X_i}$ et $\overline{X_i}$ appartiennent à un support de bornes de C. La contrainte C est dite BC si toutes ses variables sont BC.

3 Une contrainte globale pour le *lot sizing*

3.1 Présentation du problème et des notations

Le problème de *lot sizing* 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 de setup 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 de setup 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 à la période t.
- $I_0 \in \mathbb{N}$: 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 de setup de production.

La figure 1 schématise le problème, et un modèle mathématique pour ce problème (L) peut s'écrire :

$$\text{Minimize } C = Cp + Ch + Cs$$

$$
\begin{array}{lll}
\text{(L.1)} & I_{t-1} + X_t - I_t = d_t & \forall\, t = 1 \ldots T \\
\text{(L.2)} & \underline{\alpha_t} \leq X_t \leq \overline{\alpha_t} & \forall\, t = 1 \ldots T \\
\text{(L.3)} & \underline{\beta_t} \leq I_t \leq \overline{\beta_t} & \forall\, t = 1 \ldots T \\
\text{(L.4)} & X_t \leq \overline{\alpha_t} Y_t & \forall\, t = 1 \ldots T \\
\text{(L.5)} & Cp = \sum_{t=1}^{T} p_t X_t & \\
\text{(L.6)} & Ch = \sum_{t=1}^{T} h_t I_t & \\
\text{(L.7)} & Cs = \sum_{t=1}^{T} s_t Y_t & \\
\text{(L.8)} & X_t, I_t \in \mathbb{N}, Y_t \in \{0,1\} & \forall\, t = 1 \ldots T
\end{array}
\quad (L)
$$

Les contraintes $(L.1)$ sont les contraintes de conservation du flot pour chaque période, les contraintes $(L.2)$ et $(L.3)$ sont les contraintes de capacités, les contraintes $(L.4)$ sont les contraintes de setup ; notons également que le paiement d'un coût de setup à t (*i.e* $Y_t = 1$) n'implique pas forcément une production à t (*i.e* $X_t > 0$), ce qui sera utile dans la modélisation

de problèmes multi-produits pour lesquels le coût de setup est partagé. Finalement, les contraintes ($L.5$), ($L.6$) et ($L.7$) sont les expressions des différents coûts.

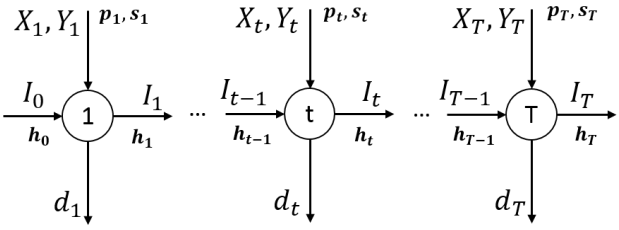

FIGURE 1 – Lot sizing

Notons que cette formulation est plus générale que la présentation classique du *lot sizing* car elle inclut des quantités minimales à produire et à stocker.

Des cas particuliers du problème (L) on été largement étudiés dans la littérature suivant les différents paramètres pris en compte. En 1958, Wagner et Whitin introduisent le problème fondateur dans [18], sans capacités, et le résolvent à l'aide d'un programme dynamique en $O(T^2)$. Cette complexité a depuis été améliorée en $O(T \log T)$ dans [5, 17, 1]. Le problème avec capacités de production constantes et coûts concaves a été résolu en $O(T^4)$ dans [6]. Avec des hypothèses de coûts linéaires pour le stockage, la complexité a pu être abaissée à $O(T^3)$ dans [16]. Le *lot sizing* avec capacités de stockage mais sans coûts de setup est résolu dans [12] en $O(T^3)$ puis résolu avec capacités de stockage et coûts de setup en $O(T^2)$ dans [2].

Les problèmes de *lot sizing* mono-produit avec des coûts non concaves ou capacités de production variables au cours du temps sont eux NP-difficiles [4].

3.2 Sémantique de la contrainte globale

Dans cette section, nous définissons formellement la contrainte globale. La contrainte porte sur les vecteurs de variables $X = \langle X_1, \ldots, X_T \rangle$, $I = \langle I_1, \ldots, I_T \rangle$ et $Y = \langle Y_1, \ldots, Y_T \rangle$ et les quatre variables de coûts Cs, Cp, Ch et C, ces variables correspondant aux variables de (L). Enfin l'ensemble des paramètres du problème est noté $data = \{I_0\} \cup \{(p_t, h_t, s_t, d_t) | t \in \{1, \ldots, T\}\}$.

Définition 1 LOTSIZING$(X, I, Y, Cp, Ch, Cs, C, data)$ *possède une solution si et seulement si il existe un plan de production solution de (L) ayant des coûts de production et stockage inférieurs ou égaux à* $\overline{Cp}$, $\overline{Cs}$, $\overline{Ch}$ *et un coût global inférieur ou égal à* $\overline{C}$.

Propriété 1 *Réaliser la consistance de bornes (BC) pour la contrainte de* LOTSIZING *est NP-Difficile.*

Preuve : La contrainte LOTSIZING possède un *support de bornes* si et seulement si il existe un plan de production respectant les capacités de production ($\underline{X_t}$, $\overline{X_t}$), les capacités de stockage ($\underline{I_t}$, $\overline{I_t}$) et les bornes supérieures des coûts. Obtenir un plan de production de coût inférieur à $\overline{C}$ seul est un problème de *lot sizing mono-produit avec capacités* qui est NP-difficile [4]. Vérifier si une borne appartient à un support de bornes est donc NP-difficile et la réalisation de la consistance de bornes également. □

4 Réalisabilité et élimination des bornes inférieures de production/stockage

Les relaxations proposées dans la suite de l'article n'ont à notre connaissance pas été traitées auparavant sur ce type de problème, en particulier l'utilisation du Weighted Interval Scheduling Problem présentée à la section 5.2.
On ignore dans un premier temps le respect des bornes supérieures des coûts : l'algorithme 1 effectue la consistance de bornes des équations (L.1) $I_{t-1} + X_t - I_t = d_t$ et détecte si le problème est réalisable du point de vue des capacités et de la demande.

Algorithm 1 testerRealisabilite

```
1: boolean modif ← true
2: while modif do
3:     modif ← false
4:     for t = 1...T do
5:         modif ← modif ∨ majBornes(t)
6: end algorithm
```

L'algorithme $majBornes(t)$ met à jour les bornes des variables X_t et I_t pour une période donnée et renvoie un booléen qui indique si une des bornes a été modifiée. Les mises à jour à t sont les suivantes :

- $\underline{X_t} \geq d_t - \overline{I_{t-1}} + \underline{I_t}$
- $\overline{X_t} \leq d_t - \underline{I_{t-1}} + \overline{I_t}$
- $\underline{I_t} \geq -d_t + \underline{I_{t-1}} + \underline{X_t}$ et $\underline{I_t} \geq d_{t+1} + \underline{I_{t+1}} - \overline{X_{t+1}}$
- $\overline{I_t} \leq -d_t + \overline{I_{t-1}} + \overline{X_t}$ et $\overline{I_t} \leq d_{t+1} + \overline{I_{t+1}} - \underline{X_{t+1}}$

Propriété 2 *Il y a équivalence entre :*

1. LOTSIZING *possède un plan de production respectant les capacités de stockage et de production.*
2. $\forall t \in \{1..T\}, D(X_t) \neq \emptyset$ ***et*** $D(I_t) \neq \emptyset$ *après l'exécution de l'algorithme 1.*

Preuve :
Cas 1 ⇒ 2 : direct.
Cas 2 ⇒ 1 : Supposons que les bornes et variables X_t et I_t ont été mises à jour par l'algorithme 1. Construisons par récurrence une solution réalisable. On pose

Algorithm 2 ecouler

Entrée : Q, t

1: $i \leftarrow t$
2: **while** $Q > 0$ **do**
3: $\quad d'_i \leftarrow \max\{0, d_i - Q\}$
4: $\quad Q \leftarrow Q - d_i$
5: $\quad$ **if** $Q > 0$ **then**
6: $\quad\quad \overline{I'_i} \leftarrow \overline{I_i} - Q$
7: $\quad\quad \underline{I'_i} \leftarrow \max\{0, \underline{I_i} - Q\}$
8: $\quad i \leftarrow i + 1$
9: **end algorithm**

Algorithm 3 transformerProbleme

1: **for** $t = 1 \dots T$ **do**
2: $\quad$ **if** $\underline{Y_t} > 0$ **then**
3: $\quad\quad s'_t \leftarrow 0$
4: $\quad$ **if** $\underline{X_t} > 0$ **then**
5: $\quad\quad \overline{X'_t} \leftarrow \overline{X_t} - \underline{X_t}$
6: $\quad\quad \underline{X'_t} \leftarrow 0$
7: $\quad\quad ecouler(\underline{X_t}, t)$
8: $\quad$ **if** $\underline{I_t} > 0$ **then**
9: $\quad\quad d'_t \leftarrow d_t + \underline{I_t}$
10: $\quad\quad \overline{I'_t} \leftarrow \overline{I_t} - \underline{I_t}$
11: $\quad\quad \underline{I'_t} \leftarrow 0$
12: $\quad\quad ecouler(\underline{I_t}, t+1)$
13: **end algorithm**

$X_1 = \underline{X_1}$ et $I_1 = \underline{I_1}$ qui vérifient la conservation du flot (d_1 est satisfaite) et respecte les capacités à $t = 1$. Supposons qu'on dispose d'une solution réalisable pour les périodes $1, \dots, t-1$ telle que $I_{t-1} = \underline{I_{t-1}}$. On obtient une solution réalisable pour t en posant :

(a) $X_t = d_t + \underline{I_t} - \underline{I_{t-1}}$ et (b) $I_t = \underline{I_t}$

On vérifie la conservation du flot à t (satisfaction de la demande assurée par (a)) et si X_t ne viole pas la capacité de production. Or d'après l'algorithme 1 :

$\underline{I_t} \geq -d_t + \underline{I_{t-1}} + \underline{X_t}$ et $\underline{I_{t-1}} \geq d_t + \underline{I_t} - \overline{X_t}$

En remplaçant $\underline{I_t}$ dans (a) on vérifie que $X_t \geq \underline{X_t}$ et en remplaçant $\underline{I_{t-1}}$ on vérifie $X_t \leq \overline{X_t}$. □

Dans la suite, on suppose que les contraintes de conservation du flot (L.1) ont été propagées jusqu'au point fixe (consistance de bornes) ainsi que les implications $X_t > 0 \implies Y_t = 1$ (L.4). On note P le problème correspondant.

Afin de simplifier les algorithmes de filtrage, P peut être transformé davantage en supprimant les bornes inférieures sur la production et les stocks. L'idée est que s'il existe t tel que $\underline{X_t} > 0$, on peut écouler $\underline{X_t}$ sur les demandes suivantes - c'est-à-dire que $\underline{X_t}$ satisfait un certain nombre de demandes à partir de t. De même si $\underline{I_t} > 0$, on peut reporter $\underline{I_t}$ sur les périodes suivantes. Cette quantité doit alors être produite à une période précédente. On ignore laquelle mais cela revient à augmenter d_t de $\underline{I_t}$.

Le problème obtenu après transformation est noté P' et ses variables et paramètres sont notés $X', I', Y', Cp', Ch', Cs', C', p', h', s', d'$. L'algorithme 2 permet d'écouler une quantité donnée Q sur les demandes à partir de la période t donnée. L'algorithme 3 permet de transformer P en éliminant les bornes inférieures des variables X_t et I_t. Ainsi on s'assure que P' vérifie : $\underline{X'_t} = 0$, $\underline{I'_t} = 0$ et $Y_t = Y'_t$. La transformation affecte également les coûts de setup et les demandes.

La figure 2 donne un exemple de transformation de P sur un horizon $T = 3$. Les nombres entre crochets représentent le domaine de X_t et I_t. La première étape consiste à éliminer la quantité minimale à produire à la période 1 en l'écoulant sur les demandes suivantes. La deuxième étape permet d'éliminer la quantité minimale à stocker entre les périodes 2 et 3 en l'écoulant sur la demande suivante. On observe bien qu'alors la demande d_2 est augmentée (flèche plus large). La dernière étape permet d'éliminer la borne inférieure de X_3 en diminuant la demande d_3.

Propriété 3 *A toute solution optimale de P correspond une solution optimale de P' de coût égal au coût optimal de P plus une constante.*

Preuve : Écouler $\underline{X_t}$ sur les demandes suivantes permet de prendre en compte un coût de production qui est obligatoire dans P sans affecter les variables de décision du problème. De même, on peut écouler $\underline{I_t}$ sur les demandes suivantes et ainsi comptabiliser un coût de stockage obligatoire. Afin d'éviter de prendre une décision quant au coût de production induit par cette quantité, on la reporte sur la demande d_t, ce qui assure la conservation du flot à t. □

En supprimant les bornes inférieures de production et de stockage, il faut comptabiliser les coûts qu'on a été obligé de payer. Définissons $Cp_{min}, Ch_{min}, Cs_{min}$ et C_{min}, les coûts à comptabiliser après le changement de variable. On rappelle qu'on suppose que la consistance de bornes sur les équations du flot a été atteinte.

$Cp_{min} = \sum_{t=1}^{T} p_t \underline{X_t}$

$Ch_{min} = \sum_{t=1}^{T} h_t \underline{I_t}$

$Cs_{min} = \sum_{t|\underline{Y_t}=1} s_t$

$C_{min} = Cp_{min} + Ch_{min} + Cs_{min}$

Les relations entre les variables de P et P' permettent de propager les bornes obtenues par la suite avec P' dans P :

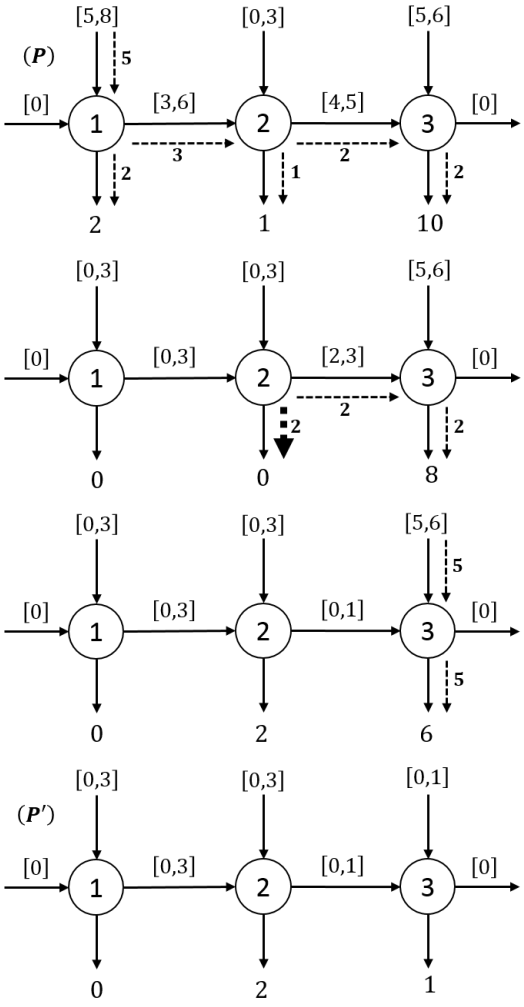

FIGURE 2 – Exemple de transformation avec $T = 3$

$Cp = Cp' + Cp_{min}$ et $Ch = Ch' + Ch_{min}$

$Cs = Cs' + Cs_{min}$ et $C = C' + C_{min}$

$X_t = X'_t + \underline{X_t}$ et $I_t = I'_t + \underline{I_t}$

5 Calcul de bornes inférieures des coûts et filtrage

Le problème P' reste NP-difficile et on étudie différentes relaxations pour calculer des bornes sur les coûts et les variables afin de les propager dans P.

5.1 Bornes sur C'

Une première relaxation porte sur les coûts de setup en les répartissant sur toute la capacité de production à chaque période. On pose $\widetilde{p_t} = p_t + \frac{s'_t}{\overline{X'_t}}$ si $\overline{X'_t} > 0$ et $\widetilde{p_t} = p_t$ sinon. Dans le cas où l'on produit à pleine capacité à t, on va payer $p_t\overline{X'_t} + s'_t$, sinon on ne va payer qu'une fraction du coût de setup. Il s'agit donc bien d'une borne inférieure du coût de production à la période t.

On définit le graphe $G = (V, E)$ (cf. figure 3) tel que :

- V est composé de $T + 2$ nœuds, un nœud t pour chaque période, une source s et un puits p.
- E est composé de $3T - 1$ arêtes. Chaque nœud t possède deux arcs sortants : (t, p) avec un coût nul et une capacité d'_t ainsi que $(t, t + 1)$ (sauf pour T) avec un coût h_t et une capacité $\overline{I'_t}$. Enfin T arcs (s, t) partent de la source vers chaque nœud t avec un coût $\widetilde{p_t}$ et une capacité $\overline{X'_t}$.

Notons $C_{flotPHS}$ le coût minimal d'un flot de quantité $\sum_{t=1}^{T} d'_t$ entre s et p.

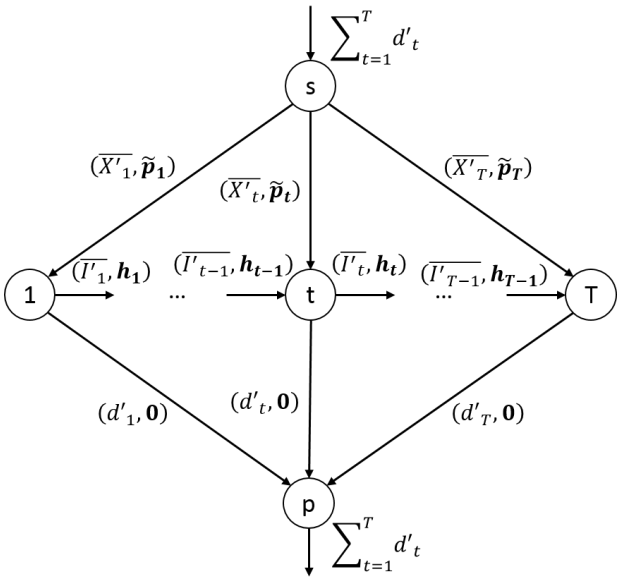

FIGURE 3 – Graphe G pour une borne inférieure de C'

Propriété 4

$$\underline{C'} \geq C_{flotPHS}$$

Preuve : Toute solution de P' est un flot réalisable dans G. □

On peut propager une deuxième borne sur C' en s'appuyant sur $\underline{Cs'}$ et sur une borne prenant en compte uniquement les coûts unitaires de production et de stockage. Cette borne est calculée en faisant circuler $\sum_{t=1}^{T} d'_t$ unités de flot dans un graphe G_2 défini comme G mais avec $\widetilde{p_t} = p_t$. Ce nouveau réseau permet de prendre en compte tous les coûts à l'exception du setup. Soit C_{flotPH} le coût optimal du problème de flot sur G_2. On a alors

$$\underline{C'} \geq C_{flotPH} + \underline{Cs'}$$

5.2 Borne sur Cs'

Le problème est difficile dès que des capacités de production et des coûts de setup sont présents.

Propriété 5 *Le problème* (L) *avec* $h_t = p_t = \underline{\beta_t} = \underline{\alpha_t} = 0$ *et* $\overline{\beta_t} = +\infty$ *est NP-difficile.*

Preuve :

La réduction se fait à partir du problème Sac-à-dos : *Entrée* : n objets de poids w_i, de valeurs v_i. Deux entiers V et W.
Question : Existe-t-il un ensemble S d'objets tel que $\sum_{i\in S} v_i \geq V$ et $\sum_{i\in S} w_i \leq W$?

On pose $s_t = w_t$, $\overline{X_t} = v_t$, $\forall t = 1 \dots T$ et $d = \{0, .., 0, V\}$. On cherche un plan de production tel que $\sum_{t\in S} X_t \geq V$ et $\sum_{t\in S|X_t>0} s_t \leq W$. N'importe quelle solution du *lot sizing* est équivalente à une solution où l'on produit à pleine capacité. On peut donc se restreindre à cet ensemble de solutions. La transformation est polynômiale. Une solution au problème de sac-à-dos est aussi solution du *lot sizing*. Et inversement une solution du problème de *lot sizing* est une solution du sac-à-dos. □

Nous proposons une approche par programmation dynamique pour obtenir une borne inférieure de Cs'. L'idée est d'associer une borne inférieure du coût de setup à chaque sous-plan de production possible et de chercher un ensemble de sous-plans disjoints maximisant la somme de ces valeurs.

On considère un sous-plan de production, c'est-à-dire un intervalle de périodes $[t_1, t_2]$ $(t_1 < t_2)$. On cherche à borner le nombre minimum de périodes de production $n_{t_1t_2}$ dans un tel sous-plan pour obtenir une borne inférieure $w_{t_1t_2}$ du coût de setup nécessaire à l'intervalle $[t_1, t_2]$.

Les périodes entre t_1 et t_2 sont simplement ordonnées par capacités de production décroissantes d'une part et par coûts de setup croissants d'autre part. On considère donc un ordre $\gamma_1, \dots, \gamma_{t_2-t_1+1}$ des périodes de sorte que : $\overline{X'_{\gamma_1}} \geq \dots \geq \overline{X'_{\gamma_{t_2-t_1+1}}}$

Et un ordre $\delta_1, \dots, \delta_{t_2-t_1+1}$ des périodes de sorte que : $s_{\delta_1} \leq \dots \leq s_{\delta_{t_2-t_1+1}}$

Pour estimer la quantité minimale à produire sur $[t_1, t_2]$, on suppose que le stock maximum possible arrive en t_1 et le stock minimum possible est laissé en t_2 *i.e.* que $I'_{t_1-1} = \overline{I'_{t_1-1}}$ et $I'_{t_2} = 0$. De plus, on suppose que la production est maximum aux périodes déjà actives et ainsi couvrir $Q_{t_1,t_2} = \sum_{t\in[t_1,t_2]|Y'_t=1} \overline{X'_t}$ unités. Pour couvrir la demande restante dans $[t_1, t_2]$ il faut donc activer au moins $n_{t_1t_2}$ périodes :

$$n_{t_1t_2} = \min\{k \mid \sum_{j=1}^{k} \overline{X'_{\gamma_j}} \geq \sum_{t=t_1}^{t_2} d_t - \overline{I'_{t_1-1}} - Q_{t_1,t_2}\}$$

Le coût de setup minimum nécessaire au sous-plan $[t_1, t_2]$ est simplement la somme des $n_{t_1t_2}$ plus petits coûts de setup des Y' non instanciés ($|D(Y'_t)| = 2$) soit :

$$w_{t_1t_2} = \sum_{j=1,\dots,n_{t_1t_2}| \, |D(Y'_{\delta_j})|=2} s_{\delta_j}$$

Tous les sous-plans possibles (il y en a donc $\frac{T(T-1)}{2}$) sont ordonnés par dates de fin croissantes d'abord puis dates de début croissantes : $[1,2], [1,3], [2,3], [1,4], [2,4], [3,4], \dots, [T-1, T]$. Pour le $i^{ème}$ sous-plan correspondant à l'intervalle $[t_1, t_2]$ on note w_i la borne inférieure de son coût de setup c'est à dire $w_i = w_{t_1t_2}$. On remarque que n'importe quel ensemble de sous-plans disjoints permet d'obtenir une borne inférieure du coût de setup global en considérant la somme des w_i associés à ces sous-plans disjoints.
Pour obtenir la meilleure borne inférieure selon ce principe, on cherche donc **un ensemble S de sous-plans disjoints qui maximise** $\sum_{k\in S} w_k$. Ce problème est précisément le Weighted Interval Scheduling Problem [9]. Il s'agit d'un problème polynômial résolu dans le cas général en $O(n \log n)$ (avec n le nombre d'intervalles), en s'appuyant sur un tri des intervalles en $O(n \log n)$ et un programme dynamique de complexité linéaire $O(n)$. Dans notre cas, les intervalles sont déjà ordonnés et l'algorithme de programmation dynamique peut donc être appliqué en $O(T^2)$. On considère l'ordre des intervalles indiqué plus haut et on note $f^*(i)$ le poids maximum possible avec les i premiers intervalles. En considérant $f^*(0) = 0$, on peut écrire pour tout $i = 1, \dots, \frac{T(T-1)}{2}$ que :

$$f^*(i) = \max(f^*(i-1), f^*(p_i) + w_i)$$

où p_i est le plus grand entier, plus petit que i (donc $p_i < i$), tel que les intervalles p_i et i sont disjoints. Ainsi p_i est le premier intervalle avant le $i^{ème}$ et compatible temporellement avec le $i^{ème}$. Par exemple $[3, 4]$ est le $6^{ème}$ intervalle et $p_6 = 1$ car $[1, 2]$ est compatible avec $[3, 4]$ mais $[1, 3]$ ne l'est pas. On cherche donc la valeur $C_{dynS} = f^*(\frac{T(T-1)}{2})$.

Propriété 6

$$\underline{Cs'} \geq C_{dynS}$$

Preuve : La validité s'appuie sur deux éléments. w_{t_1,t_2} représente une borne inférieure sur le coût de setup du sous-plan $[t_1, t_2]$. De plus, n'importe quel ensemble de sous-plans disjoints permet d'obtenir une borne inférieure du coût de setup global en considérant la somme des w_i associés à ces sous-plans. □

5.3 Borne sur Ch'

Pour obtenir une borne sur le coût de stockage, on relâche les coûts de setup et les coûts unitaires de pro-

Algorithm 4 produireAuPlusTard

```
1: C_glouH ← 0
2: for t = 1...T do
3:     UBX_t ← X̄_t
4: for t = T...1 do
5:     Q ← d_t
6:     i ← t
7:     while Q > 0 do
8:         if i ≠ t then
9:             C_glouH ← C_glouH + h_t Q
10:        prod ← min{Q, UBX_i}
11:        Q ← Q − prod
12:        UBX_i ← UBX_i − prod
13:        i ← i − 1
14: end algorithm
```

duction (*i.e.* $s'_t = p'_t = 0$). Ce problème relaxé est polynômial et il suffit d'un algorithme qui produit chaque demande au plus tard en respectant les capacités (voir algorithme 4). Cet algorithme est présenté en $O(T^2)$ par souci de clarté mais peut aussi s'écrire en $O(T)$.

Soit C_{glouH} le coût optimal de ce problème relaxé :

$$\underline{Ch'} \geq C_{glouH}$$

5.4 Borne sur Cp'

On définit le graphe G_3 comme G (cf. 5.1) avec $\widetilde{p}_t = p_t$ et $h_t = 0$. Le coût minimum d'un flot de $\sum_{t=1}^{T} d'_t$ unités sur ce graphe donne une borne sur le coût unitaire de production.

Soit C_{flotP} le coût optimal de ce problème de flot. On a alors

$$\underline{Cp'} \geq C_{flotP}$$

5.5 Filtrage par coûts réduits

Rappelons que nous faisons l'hypothèse ici qu'une solution réalisable existe (il existe ainsi un flot réalisable pour le réseau G) et qu'il n'y a pas de bornes inférieures sur les quantités à produire et à stocker.

La résolution du flot de coût minimum fournit à l'optimum des coûts réduits pour les arcs de production *i.e* les arcs (s,t) de G et les arcs de stockage entre deux périodes consécutives *i.e* les arcs $(t, t+1)$. Ces coûts réduits peuvent être utilisés pour filtrer les bornes des variables X'_t et I'_t comme dans [7].
Considérons un arc de stockage $(t, t+1) \in E$. Son coût réduit est défini par :

$$c^\pi(t, t+1) = h_t + \pi(t+1) - \pi(t)$$

avec $\pi(t+1)$ (resp. $\pi(t)$) la valeur du plus court chemin de s à $t+1$ (resp. t) dans le graphe résiduel G_f du flot optimal. On rappelle que $C_{flotPHS}$ est le coût du flot optimal et on pose I_t^f la valeur du flot optimal sur l'arc $(t, t+1)$. Le coût réduit $c^\pi(t, t+1)$ représente la variation de z pour une variation de I'_t d'une unité.

Ainsi dans le cas où $c^\pi(t, t+1) > 0$, on a $I_t^f = 0$ (conditions d'optimalité du flot) et on peut mettre à jour la borne supérieure de I'_t en appliquant :

$$C_{flotPHS} + kc^\pi(t, t+1) > \overline{C'} \Rightarrow \overline{I'_t} < k$$

Si au contraire $c^\pi(t, t+1) < 0$ alors $I_t^f = \overline{I'_t}$ et on peut mettre à jour la borne inférieure de I'_t :

$$C_{flotPHS} - kc^\pi(t, t+1) > \overline{C'} \Rightarrow \underline{I'_t} > \overline{I'_t} - k$$

Dans le cas où $0 < I_t^f < \overline{I'_t}$, le coût réduit est nul et aucun filtrage ne se produit. Les variables de production X'_t sont filtrées de la même manière en utilisant les coûts réduits des arcs de production (s,t).

5.6 Récapitulatif

Le tableau 1 recense les différentes bornes concernées, les valeurs propagées ainsi que les paramètres pris en compte.

Bornes	Valeur	$\overline{X_t}$	$\overline{I_t}$	p_t	h_t	s_t
$\underline{C'}$	$C_{flotPHS}$	√	√	√	√	√
	C_{flotPH} + $\underline{Cs'}$	√	√	√	√	
$\underline{Cp'}$	C_{flotP}	√	√	√		
$\underline{Ch'}$	C_{glouH}	√	√		√	
$\underline{Cs'}$	C_{dynS}	√				√

TABLE 1 – Récapitulatif des bornes

Les algorithmes de filtrage décrits ci-dessus sont utilisés dans l'algorithme de propagation (algorithme 5) de la contrainte globale.

Algorithm 5 Propager

```
1: testerRealisabilite() - Section 4
2: transformerProbleme() - Section 4
3: filtrerCs() - Section 5.2
4: filtrerCh() - Section 5.3
5: filtrerCp() - Section 5.4
6: filtrerC() - Section 5.1
7: filtrerCoutsReduits() - Section 5.5
```

6 Expérimentations

6.1 Résolution du problème par programmation par contraintes et PLNE

Une fois les instants de production décidés (*i.e.* une fois que les Y_t sont fixés), le problème devient facile et est résolu en $O(T^2)$ grâce à l'algorithme de flot sur le réseau G_2 défini à la section 5.1. Il

suffit donc de brancher uniquement sur les variables Y_t. Pour le moment, le branchement est chronologique.

L'implémentation est réalisée en JAVA sur Choco 3.0 [14] sur des instances générées aléatoirement. Le problème traité étant résolu facilement par la PLNE, les benchmarks existants sur le lot-sizing visent plutôt les problèmes avec plusieurs types de produits ou plusieurs niveaux (un produit et ses constituant définissant une nomenclature du produit). Les modèles PLNE sont résolus par ILOG CPLEX 12.5. Deux modèles PLNE sont utilisés à titre de référence : le modèle agrégé (AGG) du *lot sizing* (cf. modèle présenté à la section 3.1) et le modèle *facility location* (FL) [10]. De plus nous renforçons ces modèles en utilisant les bornes obtenues après le point fixe sur les contraintes de flot ce qui renforce leur relaxation linéaire (RL). Bien que la relaxation linéaire du modèle FL soit meilleure que celle du modèle AGG, le grand nombre de variables l'empêche de fournir des solutions en un temps raisonnable. Nous comparons donc seulement les modèles PPC et AGG. Les résultats du modèle simple de PPC consistant à poser la décomposition de la contrainte globale ne sont pas présentés car ce modèle n'est absolument pas compétitif.

6.2 Benchmarking et tests

La génération d'instances se fait de la façon suivante : Les coûts de stockage sont constants et égaux à 1. Les coûts de setup et les coûts unitaires de production sont générés à l'aide de 2 paramètres : e et θ. e représente le coût total unitaire de production (*i.e.* le coût unitaire si la capacité de production est saturée). On le fixe à $e = 10$. $\theta \in [0, 1]$ représente la part du coût de setup par rapport au coût unitaire de production. Le coût de production total à t (*i.e.* e fois la capacité de production) sera donc dû pour θ à son coût de setup et pour $1 - \theta$ à son coût unitaire de production. Pour chaque période, θ est pris aléatoirement de façon uniforme dans un intervalle $[\underline{\theta}, \overline{\theta}]$. La demande est générée selon une loi uniforme dans l'intervalle $[d_{\text{avg}} - \delta, d_{\text{avg}} + \delta]$. Les capacités de production et de stockage sont constantes et peuvent satisfaire $2.5d_{\text{avg}}$. Les cinq classes évaluées sont les suivantes :

- C1 : $d_{\text{avg}} = 1000$, $\delta = 100$, $\theta \in [0.8, 1]$, $T = 40$
- C2 : $d_{\text{avg}} = 1000$, $\delta = 500$, $\theta \in [0.4, 0.6]$,$T = 40$
- C3 : $d_{\text{avg}} = 1000$, $\delta = 100$, $\theta \in [0.8, 1]$, $T = 80$
- C4 : $d_{\text{avg}} = 1000$, $\delta = 500$, $\theta \in [0.4, 0.6]$, $T = 80$
- C5 : $d_{\text{avg}} = 1000$, $\delta = 10$, $\theta = 0.5$, $T = 80$

La limite de temps est fixée à 1 minute d'exécution.

Le tableau 2 présente les résultats par classe et algorithme. Il indique l'écart moyen à la valeur optimale en % (AvgG) de la meilleure solution trouvée, le nombre de preuves d'optimalité effectuées (#R), le temps moyen de résolution en secondes (CPU), le nombre moyen de nœuds explorés (Search) et le nombre de solutions optimales trouvées (#Opt).

	PPC					
CL	T	AvgG	#R	CPU	Search	#Opt
C1	40	0.15 %	0	60.00	72449	5
C2	40	0.30 %	0	60.01	69117	4
C3	80	0.83 %	0	60.01	31015	1
C4	80	0.68 %	0	60.02	27911	0
C5	80	0.13 %	0	60.01	36411	1
	AGG					
CL	T	AvgG	#R	CPU	Search	#Opt
C1	40	0.00 %	10	0.45	1955	10
C2	40	0.00 %	10	0.26	1143	10
C3	80	0.00 %	10	2.07	3663	10
C4	80	0.00 %	10	2.30	3919	10
C5	80	0.00 %	10	3.54	6736	10

TABLE 2 – Comparaison des modèles PPC et AGG

Le tableau 3 montre pour chaque classe d'instances considérée l'écart moyen des différentes bornes sur C par rapport à l'optimal (en %). Les deux dernières colonnes montrent les bornes au nœud racine (BNR) du modèle AGG et sa relaxation linéaire (RL).

CL	$C_{flotPHS}$	$C_{flotPH} + C_{dynS}$	$C_{flotP} + C_{glouH} + C_{dynS}$	AGG BNR	AGG RL
C1	10.67%	13.03%	17.07%	3.13%	10.67%
C2	8.83%	12.89%	16.97%	1.43%	8.83%
C3	9.79%	14.32%	18.55%	2.32%	9.79%
C4	9.03%	14.16%	18.45%	1.77%	9.03%
C5	9.37%	8.57%	8.57%	1.43%	9.37%

TABLE 3 – Comparaison des bornes au nœud racine

On observe bien que $C_{flotP} + C_{glouH} + C_{dynS}$ est dominée par $C_{flotPHS}$. On calcule tout de même cette borne car on a besoin de C_{flotP} et de C_{glouH} pour filtrer *Cp* et *Ch*. Par ailleurs, comme on informe AGG avec les capacités obtenues après la consistance de bornes au nœud racine, la relaxation linéaire de AGG est bien identique à $C_{flotPHS}$. Enfin, on remarque qu'avec des coûts s_t et p_t variables (classes C1 à C4), la borne $C_{flotPHS}$ domine la borne $C_{flotPH} + C_{dynS}$. Cela vient du fait que le calcul des bornes inférieures sur les sous-plans ($w_{t_1 t_2}$) est encore trop faible et prend mal en compte la variabilité des coûts de setup. Ce n'est pas le cas sur C5 où ces coûts sont constants.

Le modèle AGG reste plus efficace à ce stade sur ce problème pur (en particulier pour réaliser la preuve

d'optimalité), les coupes extraites au nœud racine lui donnent une borne initiale très forte.

7 Utilisation de la contrainte globale dans des problèmes multi-items

Cette section illustre l'utilisation de la contrainte globale proposée au travers de trois problèmes multi-produits NP-difficiles. Ces problèmes font tous apparaître des sous-problèmes de *lot sizing* mono-produit avec des contraintes liantes provenant soit des capacités, soit des coûts, soit d'un entrepôt intermédiaire.

Dans chacun, on cherche à planifier la production de N types de produits indicés par $n = 1 \dots N$. Les variables introduites dans le *lot sizing* mono-produit sont maintenant indicées par n. Chaque produit de type n voit une demande $d_{n,t}$ pour $t \in 1 \dots T$. La production d'un produit n induit un coût de setup $s_{n,t}$ et un coût unitaire $p_{n,t}$ en période t. Enfin, le stockage d'un produit n d'une période t à la suivante induit un coût de stockage $h_{n,t}$. Les autres variables sont indicées suivant le même principe.

7.1 Lot sizing multi-produits avec capacité de production commune

Ce premier problème [3] consiste à planifier la production de N types de produits sur une même ligne de production ayant la capacité de produire une quantité maximale $\overline{\alpha_t}$ en période t, tous produits confondus. Le modèle s'écrit :

$C_{global} = \sum_{n=1}^{N} C_n$

(1) $\sum_{n=1}^{N} X_{n,t} \leq \overline{\alpha_t} \qquad \forall\, t = 1 \dots T$

(2) $\textsc{LotSizing}(X_n, I_n, Y_n, Cp_n, Ch_n, Cs_n, C_n, data_n), \quad \forall\, n = 1 \dots N$

(3) $X_n, I_n \in \mathbb{N}^T \qquad \forall\, n = 1 \dots N$

(4) $Y_n \in \{0,1\}^T \qquad \forall\, n = 1 \dots N$

(5) $Cp, Ch, Cs, C \in \mathbb{R}_+^N$

7.2 Joint Replenishment Problem

Dans le Joint Replenishment Problem (JRP), des économies d'échelle sont réalisées si l'on commande (ou fabrique) plusieurs produits à la fois [11]. La production d'au moins un produit en période t engendre un coût de setup *majeur* s_t. On appellera coût de setup *mineur* le coût de setup $s_{n,t}$ associé à la production d'un produit de type n.

Soit Y_t la variable binaire de setup valant 1 si au moins un produit n a été fabriqué à t, 0 sinon. Le modèle s'écrit :

$C_{global} = \sum_{n=1}^{N} C_n + \sum_{t=1}^{T} s_t Y_t$

(1) $\textsc{LotSizing}(X_n, I_n, Y_n, Cp_n, Ch_n, Cs_n, C_n, data_n), \quad \forall\, n = 1 \dots N$

(2) $Y_{n,t} = 1 \Rightarrow Y_t = 1 \qquad \forall\, n = 1 \dots N, \forall\, t = 1 \dots T$

(3) $X_n, I_n \in \mathbb{N}^T \qquad \forall\, n = 1 \dots N$

(4) $Cp, Ch, C, Cs \in \mathbb{R}_+^N$

(5) $Y \in \{0,1\}^T$

7.3 One-Warehouse Multiretailer Problem

Le One-Warehouse Multiretailer Problem (OWMR) est une généralisation du JRP [15]. Un entrepôt central (warehouse) noté 0 approvisionne N détaillants (retailers) indicés par $n = 1 \dots N$ (voir figure 4). Chaque détaillant a les mêmes caractéristiques que le problème mono-produit présenté en section 3. Les quantités commandées par les détaillants sont bornées par les stocks disponibles à l'entrepôt central. Les paramètres et variables caractérisant l'entrepôt sont indicés par 0 : $s_{0,t}$, $p_{0,t}$, $h_{0,t}$, etc. La quantité commandée $X_{0,t}$ par l'entrepôt n'est pas bornée.

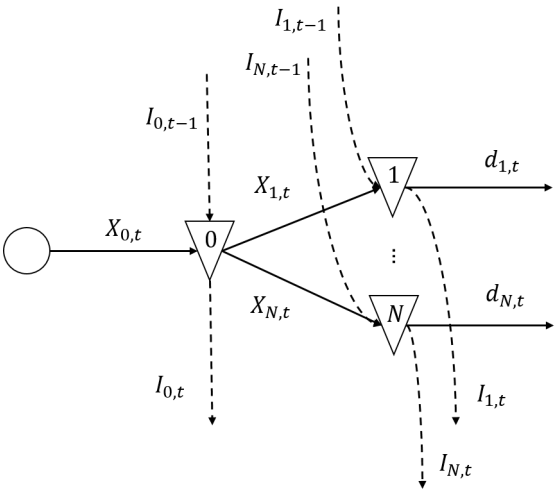

FIGURE 4 – Représentation schématique du problème OWMR à une période t.

En s'appuyant sur la contrainte de LotSizing le modèle s'écrit :

$C_{global} = \sum_{n=1}^{N} C_n + \sum_{t=1}^{T} (p_{0,t}X_{0,t} + s_{0,t}Y_{0,t} + h_{0,t}I_{0,t})$

(2) $X_{0,t} > 0 \Rightarrow Y_{0,t} = 1 \qquad \forall\, t = 1 \dots T$

(3) $I_{0,t-1} + X_{0,t} = \sum_{n=1}^{N} X_{n,t} + I_{0,t} \qquad \forall\, t = 1 \dots T$

(4) $\textsc{LotSizing}(X_n, I_n, Y_n, Cp_n, Ch_n, Cs_n, C_n, data_n) \qquad \forall\, n = 1 \dots N$

(5) $X_n, I_n \in \mathbb{N}^T, Y_n \in \{0,1\}^T \quad \forall\, n = 1 \dots N$

(6) $C, Cp, Ch, Cs \in \mathbb{R}_+^N$

La contrainte (3) est la contrainte de conservation du flot à l'entrepôt.

8 Conclusion et perspectives

Dans un premier temps, nous proposons une contrainte globale pour la résolution d'un problème de *lot sizing* mono-produit très général. Nous étudions la complexité de la contrainte et proposons un cadre (réalisabilité et simplification du problème) pour réaliser le filtrage. Plusieurs bornes inférieures sont données dont une borne équivalente à la relaxation linéaire d'un modèle classique PLNE et une borne originale focalisée sur les coûts de setup. Nos résultats expérimentaux préliminaires valident cette première étape. La suite de l'élaboration de cette contrainte portera sur la généralisation des algorithmes de calcul de bornes et le filtrage des variables de décision qui est encore incomplet. Par ailleurs, nous n'avons pas encore exploré les relaxations des capacités de production et de stockage.

Dans un deuxième temps, nous montrons l'intérêt de cette contrainte pour modéliser de nombreux problèmes multi-produits pour lesquels la PLNE est mise en échec. Nous pensons que la programmation par contraintes peut être plus adaptée pour intégrer les contraintes couplantes. Nous comptons également étudier cette approche sur des réseaux logistiques plus complexes grâce à des reformulations en stocks échelon [19].

Références

[1] Alok Aggarwal and James K. Park. Improved algorithms for economic lot size problems. *Operations Research*, 41(3) :549–571, 1993.

[2] Alper Atamtürk and Simge Küçükyavuz. An $O(n^2)$ algorithm for lot sizing with inventory bounds and fixed costs. *Operations Research Letters*, 36(3) :297–299, 2008.

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

[4] Gabriel R. Bitran and Horacio H. Yanasse. Computational complexity of the capacitated lot size problem. *Management Science*, 28(10) :1174–1186, 1982.

[5] 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(8) :909–925, 1991.

[6] Michael Florian and Morton Klein. Deterministic production planning with concave costs and capacity constraints. *Management Science*, 18(1) :12–20, 1971.

[7] Filippo Focacci, Andrea Lodi, and Michela Milano. Cost-based domain filtering. In *Principles and Practice of Constraint Programming–CP99*, pages 189–203. Springer, 1999.

[8] Vinasétan Ratheil Houndji, Pierre Schaus, Laurence Wolsey, and Yves Deville. The stockingcost constraint. In *Principles and Practice of Constraint Programming*, pages 382–397. Springer, 2014.

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

[10] Jakob Krarup and Ole Bilde. *Plant Location, Set Covering and Economic Lot Size : An O(mn)-Algorithm for Structured Problems*. Springer, 1977.

[11] Retsef Levi, Robin Roundy, David Shmoys, and Maxim Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4) :763–776, 2008.

[12] Stephen F. Love. Bounded production and inventory models with piecewise concave costs. *Management Science*, 20(3) :313–318, 1973.

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

[14] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.

[15] Gautier Stauffer, Guillaume Massonnet, Christophe Rapine, and Jean-Philippe Gayon. A simple and fast 2-approximation algorithm for the one-warehouse multi-retailers problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 67–79. SIAM, 2011.

[16] C. P. M. Van Hoesel and Albert Peter Marie Wagelmans. An $O(t^3)$ algorithm for the economic lot-sizing problem with constant capacities. *Management Science*, 42(1) :142–150, 1996.

[17] 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 :S145–S156, 1992.

[18] Harvey M. Wagner and Thomson M. Whitin. Dynamic version of the economic lot size model. *Management science*, 5(1) :89–96, 1958.

[19] Paul Herbert Zipkin. Foundations of inventory management. 2000.