La contrainte globale StockingCost pour les problèmes de planification de production

Vinasetan Ratheil Houndji[1] **Pierre Schaus**[1] **Laurence Wolsey** [1] **Yves Deville** [1]

[1] Université catholique de Louvain, Louvain-la-Neuve, Belgique

{vinasetan.houndji, pierre.schaus, laurence.wolsey, yves.deville}@uclouvain.be

Résumé

Dans beaucoup de problèmes de planification de production, il y a la minimisation des coûts de stockage des articles produits. Le papier [2] introduit la contrainte globale `StockingCost`$([X_1, \ldots, X_n], [d_1, \ldots, d_n], H, c)$ qui est satisfaite quand chacune des demandes X_i est produite avant sa date de livraison d_i, la capacité c de la machine est respectée et H est une borne supérieure sur le coût total de stockage. Ce papier est un résumé de [2] dans lequel est proposé un algorithme linéaire pour faire de la cohérence aux bornes sur la contrainte `StockingCost`. L'efficacité du propagateur `StockingCost` a été démontrée sur un problème de dimensionnement de lots.

Abstract

Many production planning problems call for the minimization of stocking/storage costs. The paper [2] introduces a new global constraint `StockingCost`$([X_1, \ldots, X_n], [d_1, \ldots, d_n], H, c)$ that holds when each item X_i is produced on or before its due date d_i, the capacity c of the machine is respected, and H is an upper bound on the stocking cost. This paper is a summary of [2] that proposes linear time algorithm to achieve bound consistency on the `StockingCost` constraint. On a version of the Discrete Lot Sizing Problem, we demonstrate experimentally the pruning and time efficiency of our algorithm compared to other state-of-the-art approaches.

1 Introduction

Les problèmes de planification de production, comme les problèmes de dimensionnement de lots, consistent à trouver un plan de production d'un ou de plusieurs articles qui satisfait les demandes en respectant la capacité de production des machines et qui minimise les coûts de production.

Pour gérer ce genre de problèmes de dimensionnement de lots, nous proposons la contrainte globale `StockingCost` et un algorithme linéaire pour faire de la cohérence aux bornes.

2 La contrainte StockingCost

La contrainte `StockingCost` a la forme suivante : `StockingCost`$([X_1, \ldots, X_n], [d_1, \ldots, d_n], H, c)$ où :

- la variable X_i est la date de production de la demande i,
- l'entier d_i est la date de livraison pour la demande i,
- l'entier c est la capacité maximum de production,
- si une demande est produite avant sa date de livraison, elle doit être stockée. La variable H est une borne supérieure sur le coût total de stockage.

La contrainte est satisfaite quand toutes les demandes sont produites avant leurs dates de livraison ($X_i \leq d_i$), la capacité totale de production est respectée (on ne peut pas produire plus de c articles à une même période) et H est une borne supérieure sur le coût total de stockage ($\sum_i (d_i - X_i) \leq H$). Example : `StockingCost`$([X_1 \in [1..2], X_2 \in [1..2]], [d_1 = 2, d_2 = 2], H \in [0..2], c = 1)$.

Une première décomposition de la contrainte est :

$X_i \leq d_i, \forall i$ (1)

$\sum_i (X_i = t) \leq c, \forall t$ (2)

$\sum_i (d_i - X_i) \leq H$ (3)

Les inégalités (2) peuvent être remplacées par une contrainte globale `global cardinality constraint (gcc)` (`allDifferent`, dans le cas $c = 1$) [4] ou encore les inégalités (2) et (3) par un `cost-gcc` [5] (`minimum assignment` pour $c = 1$ [1]).

3 Principe de l'algorithme de filtrage

Pour faire de la cohérence aux bornes $BC(H^{min})$ sur la variable de coût H (c'est à dire calculer $H^{opt} = \min \sum_i (d_i - X_i)$), l'idée est de produire les demandes le plus tard possible. Dans l'algorithme décrit dans [2], les demandes sont ordonnées dans l'ordre décroissant de X_i^{max}. Si le problème est faisable (c'est à dire que la contrainte *gcc* est satisfaite) et puisque le coût de stockage est uniforme, il suffit alors de prendre les demandes une à une et de les produire virtuellement à partir de $\max X_i^{max}$ en tenant compte à chaque fois du coût de stockage induit et de la capacité de production.

Si $H^{opt}_{X_i \leftarrow v}$ est le nouveau coût optimal quand X_i prend la valeur v, v doit être supprimée du domaine de X_i si $H^{opt}_{X_i \leftarrow v} > H^{max}$. L'intuition derrière l'agorithme pour faire de la cohérence aux bornes $BC(X_i^{min})$ est de calculer dans un premier temps pour chacune des variables X_i, la valeur ${v_i}^{opt}$ à partir de laquelle H^{min} augmente. Une borne inférieure (mais pas toujours exacte) peut être alors : $v_i^{opt} - (H^{max} - H^{min})$. Dans [2], une procédure a été décrite pour avoir la borne inférieure exacte de X_i en $O(n)$.

4 Résultats expérimentaux sur un problème de dimensionnement de lots

4.1 Description du problème

Le problème de dimensionnement de lots considéré ici, décrit dans [3], est un problème avec plusieurs articles à produire sur une machine de capacité $c = 1$. Il y a des coûts de stockage des articles et des coûts de transition (coût encouru lors du passage de la production d'un article à un autre). Toutes les demandes doivent être satisfaites avant leurs dates de livraison.

4.2 Un modèle en Programmation par Contraintes

Nous identifions de manière unique chaque demande. L'objectif est d'associer à chacune des demandes, une période qui respecte sa date de livraison. Notons $date(p) \in [1..nbPeriods]$, $\forall p \in [1..nbDemands]$ la période dans laquelle la demande p est satisfaite. Si $dueDate(p)$ est la date de livraison de la demande p, on a : $date(p) \leq dueDate(p)$. On peut casser certaines symétries concernant des demandes d'un même article en imposant : $date(p_1) < date(p_2), \forall (p_1, p_2)$ *tel que* $dueDate(p_1) < dueDate(p_2) \wedge item(p_1) = item(p_2)$. En notant $objStorage$ la borne supérieure sur le coût total de stockage, la partie de stockage peut être modélisée par la contrainte : `StockingCost`$(date, dueDate, objStorage, 1)$.

La seconde partie du problème concernant les coûts de transition peut être vue comme un modèle classique du problème du voyageur de commerce où les villes représentent les demandes et la distance entre deux villes est le coût de transition des demandes correspondantes. Si $successor(p)$, $\forall p \in [1..nbDemands]$ est la demande produite sur la machine juste après avoir produit la demande p, on a : $\forall p \in [1..nbDemands]$: $date(p) < date(successor(p))$. Une contrainte `minimum assignement` [1] est ajoutée sur les variables *successor* et les coûts de transition.

4.3 Résultats

Les expériences ont été effectuées avec le solveur OscaR [6]. La comparaison a été faite avec les trois autres décompositions décrites dans la section 2. Sur 15 instances générées aléatoirement, la contrainte `StockingCost` offre plus de filtrage et est plus rapide que les autres décompositions.

5 Conclusion

Pour modéliser certains problèmes de planification de production en Programmation par Contraintes, la contrainte globale `StockingCost` a été proposée ainsi qu'un propagateur efficace.

Références

[1] Filippo Focacci, Andrea Lodi, Michela Milano, and Daniele Vigo. Solving tsp through the integration of or and cp techniques. *Electronic notes in discrete mathematics*, 1 :13–25, 1999.

[2] Vinasetan Ratheil Houndji, Pierre Schaus, Laurence Wolsey, and Yves Deville. The stocking-cost constraint. In *Principles and Practice of Constraint Programming–CP 2014*, pages 382–397. Springer, 2014.

[3] Yves Pochet and Laurence Wolsey. *Production Planning by Mixed Integer Programming*. Springer, 2005.

[4] Claude-Guy Quimper, Peter Van Beek, Alejandro López-Ortiz, Alexander Golynski, and Sayyed Bashir Sadjad. An efficient bounds consistency algorithm for the global cardinality constraint. In *Principles and Practice of Constraint Programming–CP 2003*, pages 600–614. Springer, 2003.

[5] Jean-Charles Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3-4) :387–405, 2002.

[6] OscaR Team. Oscar : Scala in or. https ://bitbucket.org/oscarlib/oscar, 2012.