# Vérification de consistance pour la contrainte de bin packing revisitée

Julien Dupuis, Pierre Schaus, Yves Deville

▶ **To cite this version:**

Julien Dupuis, Pierre Schaus, Yves Deville. Vérification de consistance pour la contrainte de bin packing revisitée. JFPC 2010 - Sixièmes Journées Francophones de Programmation par Contraintes, Jun 2010, Caen, France. pp.133-138. inria-00520301

**HAL Id: inria-00520301**
**https://inria.hal.science/inria-00520301**

Submitted on 22 Sep 2010

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Vérification de consistance pour la contrainte de bin packing revisitée

**Julien Dupuis**[1] **Pierre Schaus**[2] **Yves Deville**[1]

[1] Departement d'ingénierie informatique, UCLouvain, Belgique
[2] Dynadec Europe, Belgique
{julien.dupuis,yves.deville}@uclouvain.be pschaus@dynadec.com

## Résumé

En plus d'un algorithme de filtrage, la contrainte `Pack` pour le bin packing à une dimension introduite par Shaw [Shaw04] utilise un algorithme de détection d'inconsistance. Ce test se base sur une réduction de la solution partielle à un problème de bin packing et sur le calcul d'une borne inférieure du nombre de boîtes sur le problème réduit. Ce papier propose deux nouveaux algorithmes de rédution et prouve que l'un d'eux domine théoriquement les autres. Les résultats expérimentaux montrent qu'une combinaison de ces réductions améliore la qualité du filtrage.

## 1 Introduction

Le problème de bin packing (BP) à une dimension consiste en la recherche du nombre minimal de boîtes nécessaires pour placer un ensemble d'objets de sorte que la taille totale des objets dans chaque boîte ne dépasse pas la capacité $C$ des boîtes. La capacité est commune à toutes les boîtes.

Ce problème peut être résolu en programmation par contraintes (CP) en introduisant une variable de placement $x_i$ pour chaque objet et une variable de charge $l_j$ pour chaque boîte.

La contrainte `Pack` introduite par Shaw [9] lie les variables de placement $x_1, \dots, x_n$ de $n$ objets ayant les poids $w_1, \dots, w_n$ avec les variables de charge de $m$ boîtes $l_1, \dots, l_m$ ayant pour domaines $\{0, \dots, C\}$. Plus précisément, la contrainte s'assure que $\forall j \in \{1, \dots, m\} : l_j = \sum_{i=1}^{n} (x_i = j) \cdot w_i$ où $x_i = j$ est réifiée à 1 en cas d'égalité et à 0 autrement. La contrainte `Pack` a été utilisée dans diverses applications, comme les problèmes d'équilibrage de lignes d'assemblage [5], les problèmes de Steel Mill Slab [2] et les problèmes de Nurse Rostering [6].

En plus des contraintes de décomposition $\forall j \in \{1, \dots, m\} : l_j = \sum_{i=1}^{n} (x_i = j) \cdot w_i$ et de la contrainte redondante $\sum_{i=1}^{n} w_i = \sum_{j=1}^{n} l_j$, Shaw a introduit :

1. un algorithme de filtrage basé sur des raisonnements de sac à dos à l'intérieur de chaque boîte, et
2. un algorithme de détection d'inconsistances basé sur une réduction de la solution partielle à un problème de bin packing.

Le présent travail se concentre sur l'amélioration de l'algorithme de détection d'inconsistances.

## 2 Reductions à des problèmes de bin packing

Shaw [9] décrit un algorithme rapide de détection d'inconsistances pour la contrainte `Pack` utilisant une borne inférieure sur nombre de boîtes (bin packing lower bound : BPLB). L'idée est de réduire l'assignement partiel courant des variables (c'est-à-dire que certains objets sont déjà assignés à une boîte) de la contrainte `Pack` à un problème de bin packing. Ensuite, une inconsistance est détectée si la borne inférieure est plus grande que le nombre $m$ de boîtes disponibles.

Nous proposons deux nouvelles réductions de la solution partielle à un problème de bin packing. La première peut dans certains cas dominer la réduction de Shaw, tandis que la seconde domine en théorie les deux autres.

**Réduction de Paul Shaw : R0** La réduction de Shaw consiste à créer un problème de bin packing avec les propriétés suivantes : la capacité des boîtes est la plus grande borne supérieure des variables de charge,

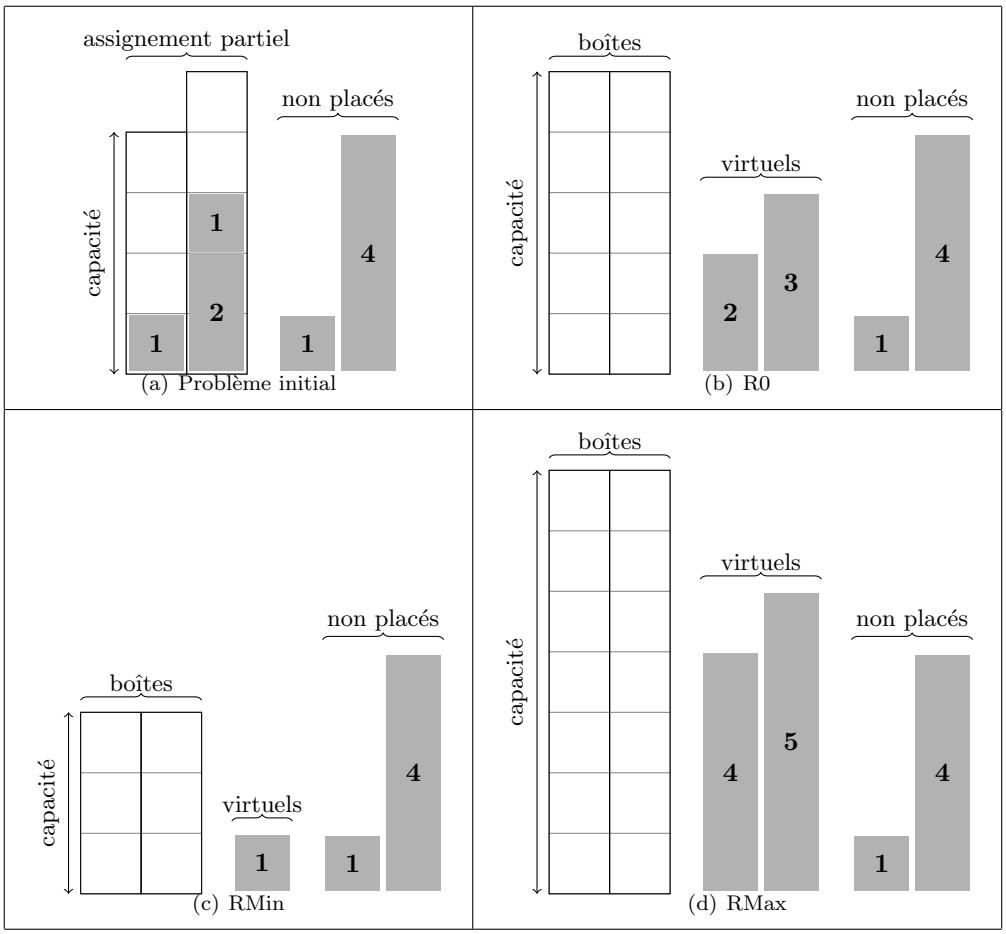



FIG. 1 – Exemple des trois réductions pour le problème de bin packing

c'est-à-dire $c = \max_{j\in\{1,\dots,m\}}(l_j^{\max})$. Tous les objets qui ne sont pas assignés à une boîte font partie des objets du problème réduit. De plus, pour chaque boîte, un objet virtuel est ajouté au problème réduit pour représenter (1) la dissimilarité de borne supérieure des variables de charge et (2) les objets déjà placés. Plus précisément, la taille de l'objet virtuel d'une boîte $j$ est $(c - l_j^{\max} + \sum_{\{i|x_i=j\}} w_i)$, c'est-à-dire la capacité $c$ réduite de la capacité réelle de la boîte, plus la taille totale des objets qui sont déjà placés dans cette boîte. Un exemple est montré à la figure 1(b).

**RMin** Nous introduisons RMin qui est obtenu à partir de R0 en réduisant la capacité des boîtes et la taille de tous les objets virtuels de la taille du plus petit objet virtuel. Les objets virtuels ont pour taille $(c - l_j^{\max} + \sum_{\{i|x_i=j\}} w_i - min_k(c - l_k^{\max} + \sum_{\{i|x_i=k\}} w_i))$. Cette réduction est illustrée à la figure 1(c).

**RMax** Nous proposons RMax qui consiste à augmenter la capacité et la taille des objets virtuels par une unique quantité, de sorte que, lorsque les objets sont placés par un algorithme de bin packing, il soit garanti que les objets virtuels occupent chacun une boîte différente. Afin que ceci soit vérifié, la taille de chaque objet virtuel doit être plus grande que la moitié de la capacité des boîtes.

Dans R0, appelons $p$ la taille du plus petit objet virtuel, et $c$ la capacité des boîtes. La taille des objets virtuels et la capacité doivent être augmentées de $(c - 2p + 1)$. Le plus petit objet virtuel aura ainsi une taille de $s = (c - p - 1)$ et la capacité des boîtes sera $(2c - 2p + 1) = 2s - 1$. On peut facilement remarquer que le plus petit objet virtuel a une taille plus grande que la moitié de la capacité. Si $c = 2p - 1$, cette rédution est équivalente à celle de Shaw. Notons que si $c < 2p-1$, la capacité et la taille des objets virtuels seront réduites.

Les objects virtuels ont une taille de $(2c - 2p + 1 - l_j^{\max} + \sum_{\{i|x_i=j\}} w_i)$. Cette réduction est illustrée à la figure 1(d).

**Réduction générique : R$\delta$** Toutes ces réductions sont un cas particulier d'une réduction générique (R$\delta$) qui, à partir de R0, consiste à ajouter un delta ($\delta$) positif ou négatif à la capacité et à la taille des objets

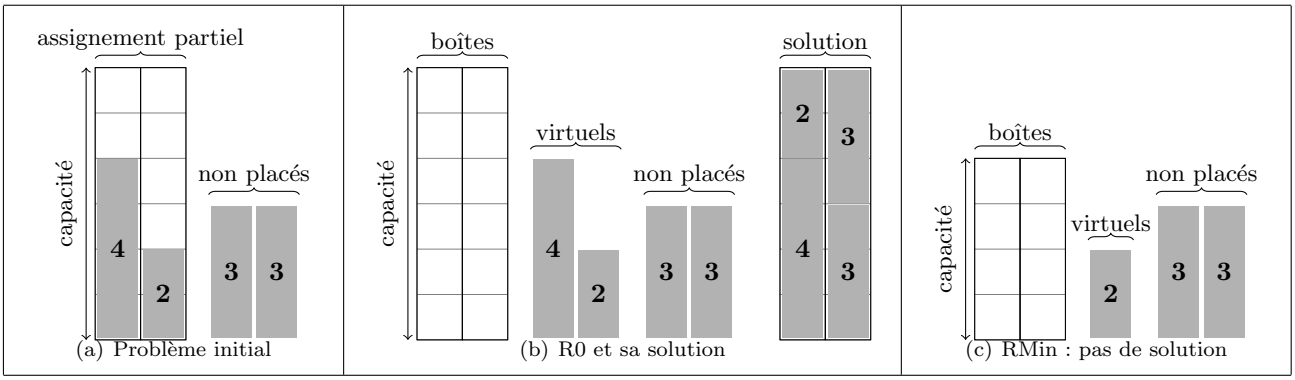



FIG. 2 – Instance de bin packing où R0 ne peut détecter l'inconsistance

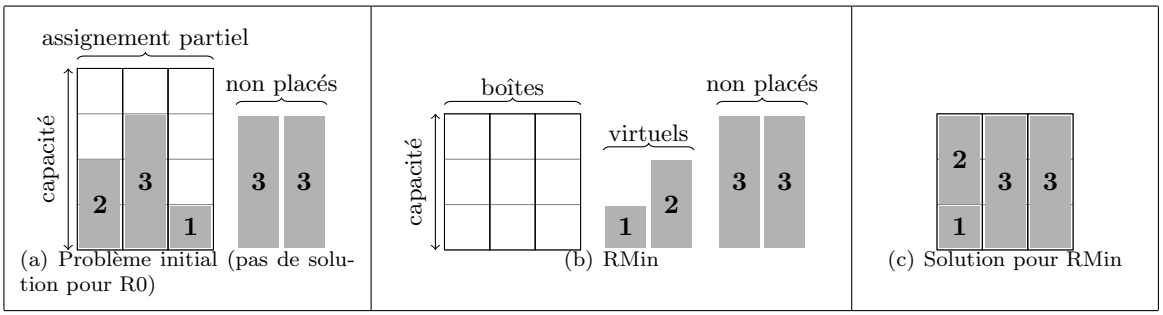



FIG. 3 – Instance de bin packing où RMin ne peut détecter l'inconsistance

virtuels.

Pour R0, $\delta = 0$. Pour RMin, $\delta$ est la plus petite valeur gardant toutes les tailles positives. Un plus petit $\delta$ créerait une inconsistance, puisque le plus petit objet virtuel aurait une taille négative. $\delta_{RMin}$ est toujours négatif ou nul. Pour RMax, $\delta$ est la plus petite valeur garantissant que les objets virtuels ne s'empilent pas. Notons que dans certains cas, $\delta_{RMin}$ ou $\delta_{RMax}$ peuvent être nuls. Notons aussi que $\delta_{R0}$ peut être plus grand que les deux autres.

## 3 Comparaison théorique des trois réductions

**Définition** Soit $A$ et $B$ deux réductions de la contrainte `Pack` à un problème de bin packing. On dit que $A$ *domine* $B$ si, pour toute instance de la contrainte `Pack`, le nombre de boîtes requises dans $A$ est plus grand que le nombre de boîtes requises dans $B$.

**Théorème** R$\delta$ est une relaxation du problème de tester la consistance de la contrainte `Pack`.

*Démonstration.* Si une solution partielle de la contrainte `Pack` peut être étendue à une solution où tous les objets sont placés, alors R$\delta$ a également une solution : si chaque objet virtuel est placé dans sa boîte initiale, l'espace libre de chaque boîte est égal à l'espace libre dans la solution partielle, et les objets non placés peuvent donc être placés dans la même boîte que dans la solution étendue de l'assignement partiel. □

**Théorème** Ni R0 ni RMin ne domine l'autre.

*Démonstration.* La figure 2 montre une instance où R0 a une solution et pas RMin. La figure 3 montre une instance où RMin a une solution et pas R0. □

**Théorème** RMax est équivalent au problème de tester la consistence de la contrainte `Pack`.

*Démonstration.* Par le théorème 3, RMax est une relaxation de la solution partielle du problème de bin packing. Il reste à montrer que s'il existe une solution pour RMax, alors la solution partielle peut être étendue à une solution complète de la contrainte `Pack`. Appelons $v$ la boîte d'où vient l'objet virtuel $v$. Il est garanti, par la taille des objets virtuels, que ceux-ci seront chacun placés dans une boîte $b_v$ différente. L'espace restant dans chaque boîte $b_v$ correspond à l'espace libre de la boîte $v$ dans le problème original. Une solution étendue de la contrainte `Pack` est obtenue en plaçant dans $v$ tous les objets se trouvant dans $b_v$. □

**Corollaire** RMax domine R0 et RMin.

TAB. 1 – Comparaison du nombre d'inconsistances détectées avec différents réductions

| Instances | Nombre d'inconsistances détectées(%) | | | | | |
|---|---|---|---|---|---|---|
| | RMin | R25 | R50 | R75 | RMax | R0 |
| Inst1 | 74.16 | 78.87 | 86.40 | 89.53 | **99.58** | 74.79 |
| Inst2 | **99.93** | 86.75 | 87.03 | 87.8 | 87.15 | **99.93** |
| Inst3 | 80.64 | 86.55 | 93.37 | 97.75 | **99.39** | 98.52 |

D'un point de vue théorique, la réduction RMax est toujours meilleure ou équivalente à R0, RMin et toute autre instance de $R\delta$. En pratique, cependant, ce n'est pas toujours le cas, comme nous le montrons dans la section suivante.

## 4 Experimental comparison

Le test d'inconsistance de Shaw [9] utilise l'algorithme de borne inférieure de bin packing $\mathcal{L}_2$ de Martello et Toth [4], qui peut être calculé en temps linéaire. Récemment, il a été prouvé [1] que l'algorithme de borne inférieure $\mathcal{L}_3$ de Labbé [3] donne toujours une borne plus grande ou égale à $\mathcal{L}_2$, et bénéficie d'une meilleure performance asymptotique (3/4 pour $\mathcal{L}_3$ [1] et 2/3 pour $\mathcal{L}_2$ [4]), tout en ayant une complexité temporelle linéaire. Les expériences montrent que $\mathcal{L}_3$ permet de détecter environ 20% d'inconsistances en plus que $\mathcal{L}_2$.

Dans les expériences ci-dessous, l'algorithme $\mathcal{L}_3$ est utilisé. Tous les programmes pour les expériences ont été implémentés dans le langage Comet.

Bien qu'en théorie, RMax est toujours meilleur que R0 et RMin, les résultats pratiques sont moins systématiques. Cela est dû au fait que $\mathcal{L}_3$ (tout autant que $\mathcal{L}_2$) n'est pas monotone, ce qui veut dire qu'une instance de bin packing nécessitant un plus grand nombre de boîtes qu'une autre instance peut produire une borne inférieure plus petite que la seconde. En fait, $\mathcal{L}_3$ est mieux adapté aux instances où la plupart des objets ont une taille plus grande que le tiers de la capacité des boîtes. RMax augmente la capacité, rendant ainsi les objets proportionnellement plus petits. Pour chacune des réductions R0, RMin et RMax, il y a des instances pour lesquelles ils contribuent à détecter une inconsistance, alors que les deux autres ne le permettent pas.

La table 1 présente les performances de la détection d'inconsistances en utilisant chacune des réductions. Elle montre le rapport du nombre d'inconsistances détectées en utilisant chacune des réductions sur le nombre total d'inconsistances détectées par au moins un des filtres. Des réductions supplémentaires ont été expérimentées, avec $\delta$ étant placé entre $\delta_{RMin}$ et $\delta_{RMax}$ à 25%, 50% et 75%. Ces résultats ont été obtenus en générant plus de 1000 instances aléatoires et en calculant $\mathcal{L}_3$ sur chacune des réductions. Voici commment ces instances ont été produites :

**Inst1** Le nombre de boîtes, le nombre d'objets et la capacité $C$ sont choisis aléatoirement entre 30 et 50. Les boîtes sont déjà remplies jusque $1..C$. La taille des objets est choisie aléatoirement dans $\{1, \ldots, C\}$.

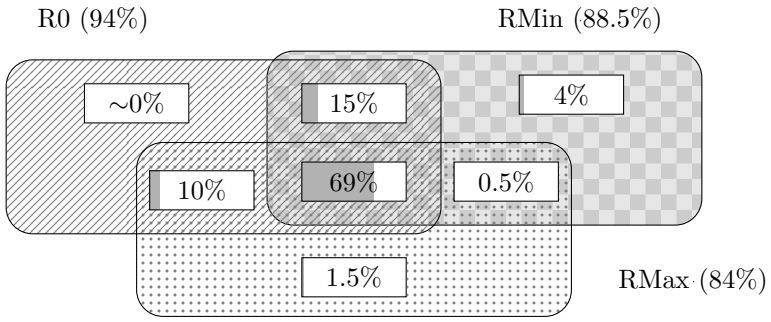
**Inst2** Il y a 50 boîtes. La capacité est de 100. Le nombre d'objets est entre 100 et 200. La taille des objets suit une distribution normale ($\mu = 5000/n$, $\sigma \in \{3n, 2n, n, n/2, n/3\}$ où $n$ est le nombre d'objets). Parmi ceux-ci, le pourcentages d'objets déjà placés $\in \{10\%, 20\%, 30\%, 40\%, 50\%\}$.

**Inst3** Mêmes paramètres que pour la deuxième instance, mais le pourcentage d'objets déjà placés est de 90% ou 95%.

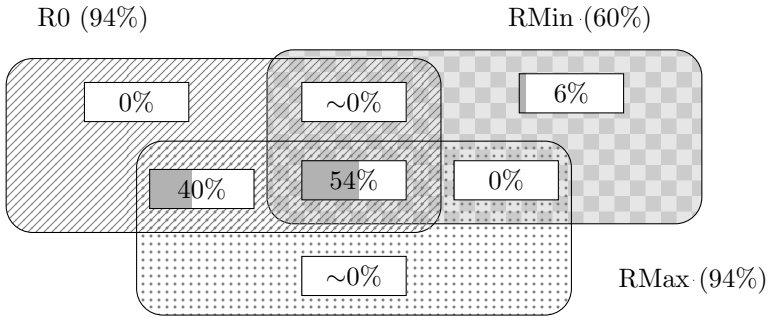
Ceci révèle que certains types d'instances sont mieux adaptées à R0 et RMin, tandis que d'autres sont mieux adaptées à RMax. Les réductions R25, R50 et R75 ne sont jamais meilleures, en moyenne, que RMin et RMax. C'est pourquoi ces réductions intermédiaires ne sont plus utilisées dans les expériences suivantes.

**Comparaison sur des données de la litérature.** Pour que l'analyse soit plus précise, nous comparons le comportement des trois réductions proposées sur des instances réelles. Des algorithmes CP ont été exécutés sur les instances SALBP-1 de Scholl [7] et sur les instances de bin packing de Scholl [8] (premier jeu de données avec n=50 et n=100), et à chaque changement du domaine des variables, la solution partielle courante a été extraite. 30 000 instances ont été sélectionnées aléatoirement parmi celles-ci, pour chaque jeu de données. Dans le second cas, seules des instances pour lesquelles au moins un des filtres détectait une inconsistance ont été sélectionnées. Les trois réductions ont été appliquées aux instances sélectionnées, avec $\mathcal{L}_3$. La figure 4 donne un schéma des résultats.

Ces résultats montrent que R0 détecte un plus grand nombre d'inconsistances. Mais (presque) toutes ces inconsistances sont également détectés par RMin ou RMax. On peut en conclure que combiner RMin et RMax est meilleur que R0 seul. Il est aussi inutile de combiner R0 avec RMin et RMax.




(a) Instance de SALBP-1




(b) Instances de bin packing

FIG. 4 – Proportions de détetion d'inconsistances en utilisant chaque réduction sur les instances de SALBP-1 (en haut) et les instances de bin packing (en bas)

TAB. 2 – Comparaison des réductions sur la résulution du problème de bin packing

| | Pas de filtre | R0 | RMin | RMax | RMin & RMax |
|---|---|---|---|---|---|
| Nombre de solutions optimales | 281 | 317 | 315 | 309 | **319** |
| Temps moyen (s) | 5.39 | 1.88 | 1.60 | 3.50 | **1.25** |

**Impact sur une recherche CP.** Nous avons comparé l'effet d'appliquer l'algorithme de détection d'inconsistances dans une recherche CP sur les instances de bin packing de Scholl N1 et N2 (360 instances au total), en utilisant R0, RMin, RMax et la combinaison de RMin et RMax, avec une limite de temps de cinq minutes pour chaque instance. Le temps moyen d'exécution a été calculé pour les instances pour lesquelles toutes les réductions menaient à la même solution. Tous ces résultats sont montrés dans la table 2. On peut observer que RMin et RMax combinés trouvent plus de solutions optimales (bien que la différence ne soit pas significative), et mène plus rapidement vers une solution que les autres (accélération de 33% par rapport à R0).

## 5 Conclusion

Ce papier présentait deux nouvelles réductions d'une solution partielle de la contrainte `Pack` à un problème de bin packing. Lors d'une recherche CP, ces réductions sont soumises à un algorithme de borne inférieure du nombre de bin afin de détecter les inconsistances de la contrainte `Pack`, comme le suggère Shaw [9].

Nous prouvons que notre deuxième réduction (RMax) donne en théorie un meilleur filtrage que les autres, en supposant que l'algorithme de borne inférieure est parfait. Nous concluons que la meilleure stratégie est de considérer conjointement les filtres RMin et RMax lors d'une recherche CP.

**Remerciements** Le premier auteur est supporté par le FNRS belge (Fonds National de la Recherche Scientifique). Cette recherche est également partiellement supportée par le Programme d'Attraction Interuniversitaire et le projet FRFC 2.4504.10 du FNRS belge.

## Références


[1] Jean-Marie Bourjolly and Vianney Rebetez. An analysis of lower bound procedures for the bin packing problem. *Comput. Oper. Res.*, 32(3) :395–405, 2005.

[2] P. Van Hentenryck and L. Michel. The steel mill slab design problem revisited. *CP'AI'OR-08, Paris, France*, 5015 :377–381, May 2008.

[3] Martine Labbé, Gilbert Laporte, and Hélène Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39(4) :616–622, 1991.

[4] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Appl. Math.*, 28(1) :59–70, 1990.

[5] P. Schaus and Y. Deville. A global constraint for bin-packing with precedences : Application to the assembly line balancing problem. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 369–374, Chicago, Illinois, USA, July 2008. AAAI Press.

[6] P. Schaus, P. Van Hentenryck, and J-C. Régin. Scalable load balancing in nurse to patient assignment problems. In *6th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, Lecture Notes in Computer Science, Pittsburgh, Pennsylvania, USA, 2009. Springer.

[7] Armin Scholl. Data of assembly line balancing problems. *Technische Universität Darmstadt*, 93.

[8] Armin Scholl, Robert Klein, and Christian Jürgens. Bison : A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7) :627 – 645, 1997.

[9] Paul Shaw. A constraint for bin packing. *Principles and Practice of Constraint Programming - CP 2004*, 3258 :648–662, 2004.