Raisonnement capacitaire et élimination de variables dans les problèmes de rangement

Diego Olivier Fernandez Pons

Raisonnement capacitaire et élimination de variables dans les problèmes de rangement

Diego Olivier Fernandez Pons [1,2]

[1] Université Pierre et Marie Curie - Paris VI, 4 place Jussieu 75005 Paris
[2] ILOG S.A, 9 rue de Verdun 94253 Gentilly Cedex
diego-olivier.fernandez-pons@cicrp.jussieu.fr dofpons@ilog.fr

Résumé

Les problèmes d'ordonnancement et de dimensionnement de réseaux ont en commun de contenir un sous-problème de rangement (*packing*) qui surgit de la nécessité d'affecter des ressources suffisantes à des entités qui les consomment. Nous rapprocherons les inégalités valides obtenues par agrégation et projection des contraintes de sac-à-dos utilisées en programmation linéaire pour le dimensionnement de réseaux et le raisonnement énergétique utilisé en ordonnancement par les méthodes de programmation par contraintes. Nous regroupons ces approches sous la dénomination d'équations capacitaires obtenues par élimination de variables et les appliquons à un problème d'emploi du temps.

Abstract

Scheduling problems and network design problems share the fact that they contain packing problems as a sub-problem. Packing problems appear every time some resource has to be assigned to some consumers in this case machine disponibility and network capacity. Linear programming methods for network design problems use valid inequalities obtained by aggregation and projection of knapsack constraints. On the other hand, scheduling has integrated what is called energetic reasonning. We will show that these two techniques can be seen as specializations of a more general capacity reasonning and are obtained by variable elimination. Our experiments are conduced on a timetabling problem.

1 Contraintes capacitaires

De nombreux problèmes d'optimisation comportent une composante de rangement, d'affectation de ressources suffisantes à des entités qui les consomment, plus généralement, une dimension capacitaire. De ce fait, leur formulation mathématique contient des inégalités capacitaires (*knapsack constraints*). Tel est le cas par exemple de problèmes aussi divers que ceux de dimensionnement de réseaux de télécommunications, d'ordonnancement ou d'emploi du temps.

En raison de leur importance, les contraintes capacitaires ont reçu très tôt une attention particulière en programmation linéaire : leur étude polyédrique a été entreprise indépendamment par Balas [2], Hammer et al. [6] et Wolsey [20] aboutissant au renforcement de chaque contrainte capacitaire par l'ajout d'inégalités de couverture (*cover-cuts*) [3]. Ont également été étudiées les conjonctions avec diverses contraintes courantes ; ainsi en présence d'incompatibilités (partition du graphe des variables par des cliques) on peut générer des inégalités de couverture renforcées dites *general upper bounding* (GUB) [5]. Enfin pour certains problèmes on est en mesure d'exploiter des structures spécifiques comme les inégalités métriques qui généralisent au multiflot le théorème flot-max/coupe-min. Ces inégalités et les inégalités de bipartition, cas particulier des premières, sont désormais communément utilisées en dimensionnement de réseaux de communications [13].

En programmation par contraintes cependant l'étude des inégalités capacitaires semble avoir été plutôt dirigée par les applications. Certes Trick [18], Sellmann [16] et Shaw [17] ont proposé des propagations renforcées pour les inégalités capacitaires. Aussi Puget [14] a-t-il étudié le cas où le nombre d'éléments dans le "contenant" est fixé. Mais les travaux les plus nombreux concernent les applications spécifiques. Ainsi Shaw était intéressé par le problème de rangement (*bin packing*) et il existe un abondante littérature sur les problèmes d'ordonnancement dont on sait que nombres de techniques peuvent être vues comme des cas particuliers de "raisonnement énergétique" [11].

Notre propos est de déterminer les difficultés que devrait avoir à résoudre une méthode automatique de renforcement de modèle par sommation des contraintes capacitaires.

2 Renforcement individuel d'inégalités capacitaires

Le renforcement des inégalités capacitaires en programmation linéaire est basé sur la détection d'exclusions mutuelles entre éléments. Ainsi, étant donnée une inégalité capacitaire (1), pour tout ensemble C de coefficients tels que leur somme dépasse la capacité totale S, il est possible de générer une inégalité dite de couverture leur interdisant d'être tous simultanément présents (2).

$$\sum_k a_k x_k \leq S \tag{1}$$

$$\forall C,\ \left[\sum_{k\in C} a_k > S\right] \Rightarrow \left[\sum_{k\in C} x_k < |C|\right] \tag{2}$$

En programmation par contraintes, Trick [18] a mis au point une méthode de filtrage par programmation dynamique garantissant l'arc consistance mais fort coûteux. Fahle et Sellmann [4] ont incorporé les bornes de Martello et Toth [12] dans une contrainte. Sellmann a également proposé des méthodes de filtrage basées sur des algorithmes d'approximation [16].

Jean-François Puget [14] lors de l'étude d'un problème de partition d'un graphe par des cliques de taille fixe (BIBD), introduit le renforcement des inégalités capacitaires par une contrainte de cardinalité :

$$\left\{\begin{array}{l} \underline{S} \leq \sum_k a_k x_k \leq \overline{S} \\ \underline{C} \leq \sum_k x_k \leq \overline{C} \end{array}\right. \tag{3}$$

$$a_k \in \mathbb{N} \quad x_k \in \{0,1\}$$

Le cas étudié par Puget où la cardinalité est une constante $\underline{C} = \overline{C}$ peut être étendu au cas d'une variable à domaine fini. Avec les données numériques

$$\forall k \in \{0\ldots 24\},\ a_k = k$$
$$\underline{S} = 12 \quad \overline{S} = 25$$
$$\underline{C} = 5 \quad \overline{C} = 10$$

les 543 solutions du système sont trouvées avec 22 retours en arrière contre 802 lors de l'utilisation indépendant de la borne consistance du produit scalaire et de la somme.

En effet, le calcul de la borne inférieure du produit scalaire se fait par remplacement de chaque variable x_k par sa borne inférieure $\underline{x_k}$:

$$\forall k, x_k \in \{0,1\} \ \Rightarrow\ \sum_k a_k \underline{x_k} = 0$$

Quand au moins C variables booléennes doivent valoir un, on peut borner plus finement par la somme des C plus petits coefficients. La borne consistance du produit scalaire peut être modifiée en conséquence pour utiliser les deux bornes.

De même une extension aux contraintes d'exclusion semblable aux *general upper bounds* (GUB) est possible :

$$\left\{\begin{array}{l} \underline{S} \leq \sum_k a_k x_k \leq \overline{S} \\ \underline{C} \leq \sum_k x_k \leq \overline{C} \\ \forall i,\ \sum_{k\in Q_i} x_k \leq 1 \end{array}\right. \tag{4}$$

$$a_k \in \mathbb{N} \quad x_k \in \{0,1\} \quad \forall i,j\ Q_i \cap Q_j = \emptyset$$

Il suffit d'incorporer les nouvelles informations dans la borne inférieure en imposant que les C plus petits coefficients considérés appartiennent à des classes différentes.

L'algorithme de filtrage proposé par Paul Shaw [17] est un intermédiaire entre la borne consistance du produit scalaire et l'arc-consistance de l'algorithme de programmation dynamique.

Shaw note qu'il ne peut y avoir de sous-ensemble dont la capacité est comprise entre la somme des k plus grands éléments et la somme des $k+1$ plus petits (c'est une conséquence de la borne issue de la cardinalité). Il définit une notion plus générale d'ensembles voisins (*neighboring subsets*) et propose un algorithme de filtrage correspondant, encapsulé dans une contrainte spécialisée au problème de rangement :

$$\left\{\begin{array}{l} \forall j,\ \sum_i a_{ij} x_{ij} = S_j \\ \forall i, \sum_j x_{ij} = 1 \end{array}\right. \tag{5}$$

$$a_{ij} \in \mathbb{N} \quad x_{ij} \in \{0,1\}$$

La contrainte de rangement **IloPack** a été étendue pour incorporer des variables de cardinalité :

$$\left\{\begin{array}{l} \forall j,\ \sum_i a_{ij} x_{ij} = S_j \\ \forall j, \sum_i x_{ij} = C_j \\ \forall i, \sum_j x_{ij} = 1 \end{array}\right. \tag{6}$$

$$a_{ij} \in \mathbb{N} \quad x_{ij} \in \{0,1\}$$

C'est cette contrainte que nous utilisons dans nos expérimentations.

3 Analyse des méthodes de sommation et élimination de variables en dimensionnement de réseaux

Le problème de dimensionnement de réseaux de télécommunications consiste à construire un réseau qui permette d'acheminer toutes les communications voulues en choisissant pour chaque arc d'un graphe support un type de lien. Chaque type de lien est pourvu d'un coût et d'une capacité. Le problème peut être représenté par le programme mathématique schématique :

$$\min \sum_{ij \in G} \mathsf{Cout}\,[y_{ij}]$$

$$\forall ij \in G,\ \sum_{d} \mathsf{Volume}_d \times x_{ij}^d \ \leq\ \mathsf{Capacite}\,[y_{ij}] \quad (7)$$

$$\forall d,\ (x_{ij}^d)_{ij \in G} \text{ flot entre } s^d \text{et } t^d \quad (8)$$

$$x_{ij}^d \in [0,1] \quad y_{ij} \in \{0 \dots k\}$$

où y_{ij} désigne le type de lien choisi pour l'arc (i,j) et x_{ij}^d la fraction de la demande d d'origine s^d et de destination t^d traversant l'arc (i,j).

Les grandeurs $\mathsf{Cout}_{ij}[]$ et $\mathsf{Capacite}_{ij}[]$ sont des fonctions de y_{ij}, les profils les plus courant étant les fonctions en escalier et les fonctions à charge fixe. Elles présentent souvent une tendance concave qui représente les économies d'échelle réalisées lorsque de nombreuses communications sont regroupées.

La difficulté du problème de dimensionnement de réseaux réside dans les bornes inférieures médiocres que fournit la relaxation linéaire, en général en raison d'une sous-estimation considérable de la capacité requise pour router toutes les communications. Les approches usuellement adoptées [13] consistent à ajouter des inégalités valides portant sur la capacité du réseau : inégalités de bipartition, inégalités de k-partition et leur généralisation, les inégalités métriques.

3.1 Inégalités de bipartition

Considérer pour tout ensemble de noeuds S les demandes ayant leur origine dans S et leur destination dans son complément $\overline{S}$. Toutes ces demandes doivent passer au moins une fois par un arc joignant un noeud de S à un noeud de $\overline{S}$. En conséquence, la capacité totale des arcs reliant S à $\overline{S}$ doit être supérieure au volume total des demandes devant les traverser :

$$\forall S \subseteq N,\ \sum_{(s^d,t^d) \in (S,\overline{S})} \mathsf{Volume}^d \leq \sum_{(i,j) \in (S,\overline{S})} \mathsf{Capacite}\,[y_{ij}] \quad (9)$$

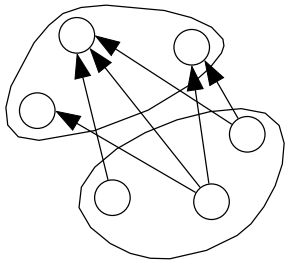

FIG. 1 – Demandes allant des noeuds de S vers les noeuds de $\overline{S}$. La capacité transversale doit être supérieure au flot transversal afin que toutes les demandes puissent être routées.

3.2 Obtention par sommation

Les inégalités de bipartition (9) peuvent aussi être vues comme la somme des inégalités capacitaires (7) le long de la coupe $(S,\overline{S})$

$$\sum_{(s^d,t^d) \in (S,\overline{S})} \mathsf{Volume}^d \times x_{ij}^d \leq \sum_{(i,j) \in (S,\overline{S})} \mathsf{Capacite}\,[y_{ij}] \quad (10)$$

combinée à l'inégalité de connexité

$$\forall d,\ \sum_{(i,j) \in (S,\overline{S})} x_{ij}^d \geq 1 \quad (11)$$

3.3 Obtention par projection

Iri et indépendamment Onaga et Kakusho [13] ont étendu le théorème coupe-min/flot-max aux multiflots : en supposant que la capacité des arcs a pour valeur fixe $\mathsf{Capacite}_{ij}$, il existe un multiflot si et seulement si toutes les inégalités métriques sont satisfaites.

Dans la mesure où les inégalités de bipartition sont un cas particulier d'inégalités métriques, une condition nécessaire pour l'existence d'un multiflot est :

$$\forall S, \mathsf{Volume}_{(S,\overline{S})} \ \leq \sum_{(S,\overline{S})} \mathsf{Capacite}_{ij} \quad (12)$$

Il convient de noter la disparition des variables x_{ij}^k dans les équations (9) et (12) : ces inégalités ne prennent pas en compte la "position" des demandes mais seulement la capacité des arcs.

Ainsi l'approche de la programmation linéaire pour les problèmes de dimensionnement de réseau consiste à obtenir des inégalités valides par transformation du problème en variables (x_{ij}^d, y_{ij}) en un problème en variables y_{ij} exclusivement. Les inégalités capacitaires redondantes on été obtenues par élimination des variables de position x_{ij}^d au moyen d'une sommation.

4 Exemple d'inégalité capacitaire redondante : la contrainte agrégée de sac-à-dos

Karen Aardal souligne [1] la possibilité de reformuler un problème linéaire en y ajoutant de la redondance (variables et contraintes) afin de déclencher certaines heuristiques de génération de coupes disponibles dans les solveurs linéaires. L'ajout de certaines sommes de contraintes de sac-à-dos (*surrogate knapsack*) lui permet ainsi d'améliorer la résolution d'un problème de localisation (*facility location problem*). Michael Trick parvient au même résultat sur un problème de transport [19].

La programmation linéaire désigne par "contrainte redondante" une inégalité qui contrairement aux plans coupants (*cutting planes*) n'élimine aucune région du polyèdre défini par la relaxation linéaire du problème (figure 2). Dans la conception traditionnelle de la programmation linéaire, l'efficacité de la résolution d'un problème est déterminée par l'acuité de sa relaxation linéaire. Les contraintes redondantes n'améliorant pas la relaxation linéaire, l'interprétation de leur efficacité reste difficile. Trick attribue l'accroissement observé des performances à la part heuristique de la génération de coupes dans les solveurs linéaires, dans ce cas précis les *cover-cuts*.

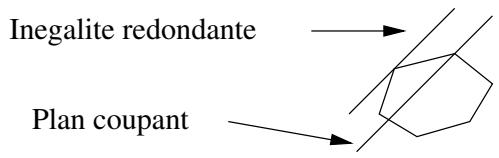

FIG. 2 – Contrairement à la contrainte redondante, le plan coupant élimine une région du polyèdre défini par la relaxation linéaire du problème. De ce fait, il participe au renforcement de la relaxation linéaire.

Nous avons étudié l'ajout de contraintes agrégées de sac-à-dos sur un problème de rangement avec capacités variables (*variable-sized bin packing problem*) : il s'agit de choisir M boîtes parmi différents modèles possibles afin d'y ranger un nombre prédéfini d'éléments. Chaque modèle de boîte a un coût, une limitation en volume et nombre d'éléments qu'on peut y placer. Le coût total est la somme des coûts des boîtes.

Si on considère le problème de dimensionnement de réseaux (7 - 8) dans lequel le routage se fait par des chemins au lieu de flots $x_{ij}^d \in \{0,1\}$, le problème de rangement avec capacités variables en est la restriction aux arcs d'une coupe $\{ij \mid i \in S,\ j \notin S\}$

$$\min \sum_b \mathsf{Cout}[y_b]$$

$$\forall e,\ \sum_b x_b^e = \mathsf{N}_e \tag{13}$$

$$\forall b,\ \sum_e \mathsf{Taille}_e x_b^e \ \leq\ \mathsf{Capacite}\,[y_b] \tag{14}$$

$$\forall b,\ \sum_e x_b^e \ \leq\ \mathsf{Cardinalite}\,[y_b] \tag{15}$$

$$x_b^e \in \mathbb{N} \quad y_b \in \{1 \ldots \mathsf{NbModeles}\}$$

La variable y_b désigne le type de la boîte b et la variable x_b^e le nombre d'éléments de type e placés dans la boîte b. La contrainte (13) garantit que le nombre total d'éléments de type e est bien N_e. La contrainte (14) s'assure que la boîte b a une capacité suffisante par rapport aux éléments qu'elle contient en terme de volume et la contrainte (15) en terme de nombre d'éléments.

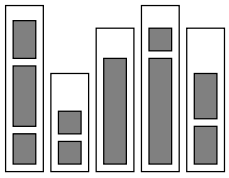

FIG. 3 – Exemple de solution du *variable sized bin packing problem*. Les boîtes grises correspondent aux élément qu'il faut placer, les boîtes blanches aux "conteneurs" dont il faut déterminer la taille. Le problème est mono-dimensionnel.

Les contraintes (16) et (17), agrégations respectives des contraintes capacitaires (14) et (15) sont obtenues par sommation sur toutes les boîtes. Ces contraintes imposent que le volume total des boîtes soit supérieur au volume total des éléments à y placer - idem pour la cardinalité. En ce sens, elles sont le pendant des inégalités de bipartition du problème de dimensionnement de réseaux. Sur la figure 3 cela correspond à noter que la surface totale blanche doit être supérieure à la surface grise.

$$\sum_e \mathsf{Taille}_e \times \mathsf{N_e} \ \leq\ \sum_b \mathsf{Capacite}\,[y_b] \tag{16}$$

$$\sum_e \mathsf{N}_e \ \leq\ \sum_b \mathsf{Cardinalite}\,[y_b] \tag{17}$$

Ils convient de noter que ces contraintes redondantes portent uniquement sur l'ensemble des M boîtes choisies et non sur la position x_b^i des éléments.

Les instances du problème 031 de la CSPLib (*the rack configuration problem*) sont aisément résolues par le solveur linéaire ILOG Cplex [8] avec le modèle simple (13 - 15). L'ajout des contraintes (16) et (17) permet de faire de même avec le solveur de contraintes ILOG Solver ce qui n'était pas le cas avec d'autres approches telle la rupture de symétries [21]. Cependant, sur des instances considérablement plus difficiles générées à partir d'un problème de dimensionnement de réseaux, le gain apporté à la résolution avec Cplex ne s'élève qu'à 30%.

En revanche, la combinaison des inégalités agrégées de sac-à-dos avec les variables de gaspillage (variables d'écart en programmation linéaire) conduit à un gain d'environ un facteur $\times 1000$ à la fois en programmation linéaire et en programmation par contraintes.

Ainsi convient-il d'ajouter les variables d'écart $\mathcal{V}_b$ et $\mathcal{C}_b$ dans les contraintes (14) et (15) :

$$\forall b,\ \sum_e \mathsf{Taille}_e x_b^e \ +\ \mathcal{V}_b \ =\ \mathsf{Volume}\,[y_b] \qquad (14\text{ bis})$$

$$\forall b,\ \sum_e x_b^e \ +\ \mathcal{C}_b \ =\ \mathsf{Cardinalite}\,[y_b] \qquad (15\text{ bis})$$

$$\mathcal{V}_b \in \mathbb{N} \quad \mathcal{C}_b \in \mathbb{N}$$

puis dans leurs agrégations respectives

$$\sum_e \mathsf{Taille}_e \times \mathsf{N}_e \ +\ \sum_b \mathcal{V}_b \ =\ \sum_b \mathsf{Volume}\,[y_b] \qquad (16\text{ bis})$$

$$\sum_e \mathsf{N}_e \ +\ \sum_b \mathcal{C}_b \ =\ \sum_b \mathsf{Cardinalite}\,[y_b] \qquad (17\text{ bis})$$

L'efficacité de ces contraintes redondantes à la fois en programmation linéaire et en programmation par contraintes suggère une origine autre que purement heuristique comme avancé par Aardal et Trick.

Afin de guider l'interprétation du rôle de ces variables et contraintes redondantes, nous considérons un exemple (figure 4) : supposons donné un problème dont la somme totale des éléments est 1000, ayant un élément de taille 80, dont la plus petite boîte de taille ≥ 80 soit de taille 100 et n'ayant aucun élément de taille ≤ 20.

D'après (16 bis) on a

$$\sum_b \mathsf{Volume}\,[y_b] \ =\ 1000 + \sum_b \mathcal{V}_b \ \geq\ 1000$$

De même suivant (16)

$$\sum_b \mathsf{Volume}\,[y_b] \ \geq\ 1000$$

Si on place l'élément de taille 80 dans la boîte 1 alors le solveur calcule que la variable de gaspillage $\mathcal{V}_1$ est minorée par 20.

Maintenant, l'équation (16 bis) donne

$$\sum_b \mathsf{Volume}\,[y_b] \ =\ 1000 + \sum_b \mathcal{V}_b \ \geq\ 1020$$

Tandis que (16) demeure inchangée.

Ainsi les variables de gaspillage ont "propagé" une information découverte lors du branchement et liée à l'indivisibilité des éléments et de ce fait indépendante de la relaxation linéaire. Cette propagation peut être déclenchée par plusieurs facteurs dont l'affectation d'éléments (par exemple celle de tous les éléments de taille ≤ 20 à des boîtes autres que 1) ou la découverte d'une solution réalisable qui interdit l'utilisation de boîtes de taille supérieure à 100.

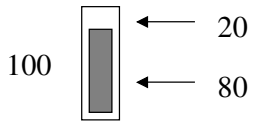

FIG. 4 – Borne inférieure sur la variable gaspillage quand on place un élément de taille 80 dans une boîte de taille 100, en l'absence de tout élément de taille inférieure à 20

Il nous semble que l'efficacité de la contrainte agrégée de sac-à-dos n'est pas associée au déclenchement d'heuristiques de renforcement de modèle mais bien à la structure du problème de rangement. Ainsi les améliorations sont observables même avec ILOG Solver qui ne génère pourtant aucun type de renforcement heuristique du modèle contrairement aux *cover cuts* des solveurs linéaires.

La mise en oeuvre d'une procédure de reformulation automatique exploitant les propriétés du problème de rangement semble cependant délicate. Certes l'élimination par sommation des variables de position x_b^e permet d'obtenir des contraintes redondantes prenant en considération les inégalités capacitaires dans leur globalité, mais ce faisant on perd les contraintes d'intégrité sur les variables x_b^e. Ce n'est que la réintroduction indirecte de l'intégralité de x_b^e via les variables de gaspillage qui permet des gains de performances significatifs. Enfin, le filtrage produit par les variables de gaspillage serait inefficace si les boîtes étaient disponibles dans toutes les tailles entières possibles, il est donc lié à la "linéarité" des fonctions de coût, capacité et cardinalité des boîtes.

5 Un problème d'emploi du temps

Nous nous intéressons désormais à un problème de rangement dans lequel toutes les tailles disponibles de boîtes sont possibles, donc rendant inutile les variables de gaspillage. Notre propos est d'évaluer les contraintes capacitaires agrégées dans ce cas de figure et d'étudier des possibles spécialisations qui exploiteraient plus spécifiquement certaines propriété du problème. A cette fin nous avons choisi un problème d'emploi du temps intermédiaire entre le rangement et l'ordonnancement.

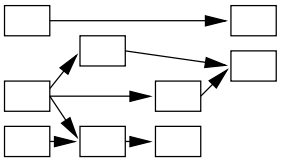

FIG. 5 – Exemple de solution pour le BACP : les cours devant avoir lieu pendant le même semestre sont représentés par des rectangles alignés verticalement. Les flèches entre cours indiquent des relations de précédence.

Dans le problème d'organisation d'un cursus (*balanced academic curriculum problem* - BACP), problème numéro 30 de la CSPLib (`www.csplib.org`), les cours d'un cursus doivent être organisés de façon à respecter la durée de la formation, les contraintes de précédence entre cours et le nombre minimal et maximal de cours et crédits par semestre. L'objectif est de lisser la charge pour les étudiants en terme de nombre de crédits par semestre. La figure 5 montre un exemple de solution.

Les précédences rendent le problème proche des problèmes de gestion de projet (*project scheduling problem* - PSP) mais contrairement à ces derniers d'une part les tâches ont des durées unitaires, d'autre part le makespan est fixe et les ressources variables au lieu du contraire.

D'autre part, borner indépendamment la capacité et la cardinalité des périodes temporelles revient — dans l'optique des problèmes de rangement à capacité variable — à permettre toutes les boîtes dans le produit cartésien des intervalles entiers de capacité et cardinalité.

5.1 Modèles naïfs

Le BACP peut être modélisé de la façon suivante :

$$\min \left\{ \max_j \text{Somme}_j \right\}$$

$$\forall i,\ \sum_j x_j^i = 1 \tag{18}$$

$$\forall j,\ \text{Somme}_j = \sum_i \mathsf{W}^i x_j^i \tag{19}$$

$$\forall j,\ \text{Card}_j = \sum_i x_j^i \tag{20}$$

$$\forall i,\ [\text{position}_i = k] \Leftrightarrow \left[x_k^i = 1\right] \tag{21}$$

$$\forall ab,\ a \ \mathsf{precede}\ b \Rightarrow \text{position}_a < \text{position}_b \tag{22}$$

$$\forall j,\ \mathsf{MinCard} \leq \text{Card}_j \leq \mathsf{MaxCard} \tag{23}$$

$$\forall j,\ \mathsf{MinSomme} \leq \text{Somme}_j \leq \mathsf{MaxSomme} \tag{24}$$

$$x_j^i \in \{0,1\} \quad \text{position}_i \in \{0 \ldots \mathsf{nbColonnes} - 1\}$$

$$\text{Somme}_j \in \mathbb{N}, \quad \text{Card}_j \in \mathbb{N}$$

L'indice i variant entre 0 et $\mathsf{NbLignes} - 1$ représente les cours. L'indice j variant entre 0 et $\mathsf{NbColonnes} - 1$ représente les périodes. La variable booléenne x_j^i indique si le cours i est planifié pour la période j tandis que la contrainte (18) s'assure que chaque cours n'est placé qu'une seule fois dans l'emploi du temps. La variable position_i est la variable domaine correspondant aux booléens x_j^i pour j variant sur toutes les périodes, comme indiqué par (21) et l'équation (22) garantit que les précédences sont bien vérifiées. Les contraintes (19) et (20) définissent le nombre de cours et de crédits par période tandis que (24) et (23) en fixent les bornes.

Les équations (24) et (23) sont des artifices pour réduire la combinatoire du problème. Une autre option serait d'affiner l'équilibrage des charges directement dans la fonction objectif, par exemple

$$\min \left[(\max_j \text{Somme}_j) - (\min_j \text{Somme}_j) \right]$$

Ce modèle naïf peut être immédiatement fourni à ILOG Solver [9]. Cependant, les contraintes (21) et (22) n'étant pas linéaires, afin de tester notre modèle avec ILOG Cplex [8] nous les remplaçons par les contraintes :

$$\forall i,\ \text{position}_i = \sum_j j \times x_j^i \tag{21 LP}$$

$$\mathsf{precede}\ b \Rightarrow \text{position}_a + 1 \leq \text{position}_b \tag{22 LP}$$

La contrainte (22 LP) est juste une réécriture pour éviter des imprécisions qui pourraient survenir dans la relaxation linéaire en raison d'une inégalité stricte entre flottants.

5.2 Premières expérimentations

Hnich et al. [7] ont étudié le BACP à la fois en programmation linéaire en nombres entiers, en programmation par contraintes et par des méthodes hybrides.

	8	10	12
Hnich Solver	> 150 s	> 150 s	> 150 s
Hnich Cplex	1.27 s	3.17 s	13.77 s
Solver naïf	> 600 s	> 600 s	> 600 s
Cplex naïf	2 s	3 s	25 s
Solver + (25-26)	0 s	0 s	1 s
Cplex + (25-26)	2 s	1 s	9 s

TAB. 1 – Résultats comparés du modèle naïf et du modèle comportant les contraintes agrégées de sac-à-dos. Notre plate-forme de test est un Pentium III 500 Mhz avec 256 Mo de RAM sous Linux Les résultats de Hnich et al. sont issus de [7].

La différence entre leur modèle CP et notre modèle naïf est qu'ils ajoutent une contrainte globale de cardinalité (GCC) [15]. Quant à leur modèle PLNE, il utilise une linéarisation différente des contraintes de précédence (21) et (22) :

$$\forall ab,\ a \text{ precede } b \Rightarrow \forall k,\ x_k^a \leq \sum_{j>k} x_j^b \qquad \text{(21 Hnich)}$$

$$\forall ab,\ a \text{ precede } b \Rightarrow x^a_{\mathsf{NbPeriodes}} = 0 \qquad \text{(22 Hnich)}$$

et les variables de position sont éliminées.

Nos résultats avec le modèle naïf sont du même ordre de grandeur que ceux annoncés par Hnich et al. (table 1). L'ajout d'une contrainte de cardinalité GCC au modèle naïf ou l'utilisation des équations de Hnich au lieu de (21) et (22) ne change pas grandement le résultat.

Les contraintes de sac-à-dos agrégé somme des inégalités capacitaires (19) et (20) s'écrivent :

$$\begin{aligned}\sum_j \text{Somme}_j &= \sum_j \sum_i \mathsf{W}^i x_j^i \\ &= \sum_i \mathsf{W}^i \\ &= \mathsf{SommeTotale} \qquad (25)\\ \sum_j \text{Card}_j &= \sum_j \sum_i x_j^i \\ &= \mathsf{NbLignes} \qquad (26)\end{aligned}$$

Avec ces contraintes, toutes les instances de la CSPLib sont résolues par Solver et Cplex instantanément.

5.3 Raisonnement capacitaire général

Une contrainte capacitaire pour les problèmes de rangement, plus générale que la contrainte agrégée de sac-à-dos et obtenue par le même principe d'élimination des variables de position x_b^i pourrait s'écrire :

$$\forall S \subseteq B, \sum_i \lambda(i,S) \times \mathsf{Taille}_i \ \leq \ \sum_{b\in S} \text{Capacite}_b \qquad (27)$$

avec

$$\lambda(i,S) = \begin{cases} 1 \text{ si } \sum_{b\in S} x_b^i = 1 \\ 0 \text{ sinon} \end{cases}$$

5.4 Difficultés de l'approche générale

La difficulté de mise en oeuvre de l'équation (27) provient du fait que la valeur de λ dépend du domaine courant des variables de position des éléments à ranger. Dans le cas du dimensionnement de réseaux un argument de connexité permettait de déduire des bornes inférieures sur le membre gauche de l'équation. Dans le problème de rangement il n'y a rien de semblable.

Il est certes possible d'extraire de l'équation (27) une procédure permettant de générer pendant la recherche arborescente des contraintes capacitaires, conduisant à une approche de type génération de contraintes semblable au *branch-and-cut* ou dans une certaine mesure aux méthodes dites de décomposition de Benders : en tout noeud de l'arbre de recherche, après exclusion des éléments ayant déjà été affectés à une boîte, il suffit de considérer un sous-ensemble I des éléments et de prendre pour ensemble S dans l'équation (27) l'union de leurs domaines.

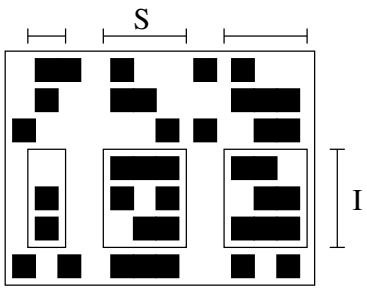

FIG. 6 – Domaines possibles des éléments en un noeud de l'arbre de recherche. Les éléments (cours) sont représentés en ordonnée, leur domaine (positions possibles) en abscisse. I désigne un sous-ensemble des éléments et S l'union de leur domaines. Ainsi, les cours du sous ensemble I devant nécessairement être disposés dans les positions S, on peut en déduire une contrainte sur la capacité minimale de la somme des positions de S.

La contrainte générée est

$$\sum_{i\in I} \mathsf{Taille}^i \leq \sum_{b\in S} \mathsf{Capacite}_b$$

On peut par exemple résoudre heuristiquement le problème suivant :

$$\min \frac{\sum_i \sum_b \mathsf{pos}_b^i \lambda_i y_b}{\left(\sum_i \lambda_i\right) \times \left(\sum_j y_j\right)}$$

$$\forall i, b,\ y_b \geq \mathsf{pos}_b^i \times \lambda_i \tag{28}$$

$$\forall b,\ \sum_i \mathsf{pos}_b^i \times \lambda_i \geq y_b \tag{29}$$

avec

$$I = \{\ i\ |\lambda_i = 1\}$$
$$S = \{\ b\ |y_b = 1\}$$
$$\forall i,\ \lambda(i,S) = 1\ \Leftrightarrow\ i \in I$$

La contrainte 28 garantit que S est un sur-ensemble de l'union des domaines des éléments de I tandis que 29 garantit qu'elle en est un sous-ensemble. Le critère d'optimisation minimise le rapport entre la surface $I \times S$ et son contenu effectif (figure 6).

Bien entendu, d'autres critères d'optimisation sont possibles - par exemple utilisant la taille des éléments. Néanmoins, déterminer une bonne heuristique de génération de contraintes s'avère souvent un problème très complexe en raison des multiples compromis qu'il faut établir. De surcroît utiliser une telle méthode dans un environnement de programmation par contraintes la rendrait dépendante de la qualité de la propagation, problème qui n'apparaît pas dans en programmation linéaire en nombres entiers car les procédures de séparation se basent sur la solution de la relaxation linéaire.

Il nous a donc semblé plus avisé de fixer dans le modèle et en fonction des propriétés du problème considéré les sous-ensembles de boîtes sur lesquelles va s'appliquer l'inégalité capacitaire (27) mais en contrepartie de garder la grandeur λ sous forme de variable.

5.5 Raisonnement capacitaire spécialisé pour les relations de précédences

Étant données les relations de précédence dans le BACP, le choix des ensembles S_k s'est porté sur la famille d'ensembles de périodes $S_k = \{k \ldots \mathsf{NbColonnes} - 1\}$ (figure 7).

Nous introduisons pour tout éléments i et tout ensemble S_k une variable $\lambda_k^i \in \{0,1\}$ qui indique si le domaine de l'éléments i est inclus dans S_k. En réutilisant la structure de S_k cela se réécrit

$$\forall i,\ \forall k,\ [\lambda_k^i = 1]\ \Leftrightarrow\ [\text{position}_i \geq k] \tag{30}$$

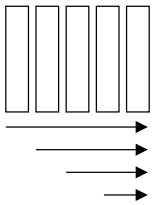

FIG. 7 – Sous-ensembles S choisis pour le BACP

Nous introduisons également par commodité les variables suivantes

$$\begin{aligned} \forall k,\ \text{SommeCumul}_k &= \sum_{j \in S_k} \text{Somme}_j \\ &= \sum_{j \geq k} \sum_i \mathsf{W}^i x_j^i \end{aligned} \tag{31}$$

$$\begin{aligned} \forall k,\ \text{CardCumul}_k &= \sum_{j \in S_k} \text{Card}_j \\ &= \sum_{j \geq k} \sum_i x_j^i \end{aligned} \tag{32}$$

et les contraintes permettant d'en resserrer les bornes

$$\forall k, \sum_{j<k} \text{Somme}_j + \text{SommeCumul}_k\ =\ \sum_i \mathsf{W}^i \tag{33}$$

$$\forall k, \sum_{j<k} \text{Card}_j + \text{CardCumul}_k\ =\ \mathsf{NbLignes} \tag{34}$$

Les contraintes de capacité s'écrivent alors

$$\forall k,\ \sum_i \lambda_k^i \mathsf{W}^i\ =\ \text{SommeCumul}_k \tag{35}$$

$$\forall k,\ \sum_i \lambda_k^i\ =\ \text{CardCumul}_k \tag{36}$$

Elles généralisent bien les contraintes agrégées de sac-à-dos qui en sont le cas particulier dans lequel $k = 0$.

Cependant, la transitivité des contraintes de précédence n'est pas amplement exploitée. On peut, pour tenter d'améliorer cette situation, introduire des variables booléennes s_{ij} capturant la notion de succession entre les éléments :

$$\forall ij,\ [s_{ij} = 1]\ \Leftrightarrow\ [\text{position}_i < \text{position}_j] \tag{37}$$

Il convient ensuite de remarquer que le positionnement d'un élément i en une colonne j en raison des contraintes de précédence va imposer une capacité minimale à la somme cumulée des capacités des colonnes postérieures à j. Cette contrainte se transcrit à l'aide des variables précédentes par :

$$\forall i,\ \text{SommeCumul}[\text{position}_i + 1]\ =\ \sum_k s_{ik} \mathsf{W}^k \tag{38}$$

$$\forall i,\ \text{CardCumul}[\text{position}_i + 1]\ =\ \sum_k s_{ik} \tag{39}$$

Enfin peut-on dériver des bornes inférieures sur l'objectif en utilisant le fait que le maximum d'un ensemble de nombres est toujours supérieur à leur moyenne. Cette inégalité appliquée aux sous-ensembles S_k donne :

$$\forall k,\ \text{SommeCumul}_k \leq z \times (\mathsf{nbColonnes} - k - 1) \quad (40)$$

5.6 Expérimentations

Les premières expérimentations avec modèle (18) - (40) ont été menées sur l'instance 12 de la CSPLib. Le BACP semble particulièrement sensible à la stratégie de recherche adoptée. Dans tous les résultats de la table 1 les cours sont "poussés vers le fond" en branchant sur les variables x_j^i colonne après colonne, vers 0 en premier et en considérant les variables ayant le plus grand coefficient W^i en dernier (c'est la stratégie utilisée par Hnich et ses prédécesseurs). Renverser l'ordre de sélection des variables dégrade quelque peu les performances en augmentant le temps de résolution de 1 à 36 secondes. Par contre, le branchement ligne par ligne ne permet pas de fermer le problème en 10 minutes.

Nous comparerons donc :
- (A) Le modèle naïf + IloPack étendu + borne inférieure statique
- (B) Le modèle (25)-(42) + IloPack étendu
- (C) Le modèle (25)-(42) symétrisé + IloPack étendu

méthode	branchement	fails	temps
A	colonne / croissant	638	0 s
B	colonne / croissant	638	2 s
C	colonne / croissant	638	3 s
A	colonne / décroissant	44818	35 s
B	colonne / décroissant	44423	226 s
C	colonne / décroissant	44423	504 s
A	ligne / décroissant	-	$>$ 1h
B	ligne / décroissant	6083	45 s
C	ligne / décroissant	6076	140 s
A	ligne / croissant	-	$>$ 1h
B	ligne / croissant	-	$>$ 1h
C	ligne / croissant	-	$>$ 1h

Nous avons utilisé des stratégies de branchement qui pourraient sembler surprenantes pour lisser la charge. En effet comme nous nous intéressons aux problèmes de rangement en tant que sous-problèmes de problèmes plus importants tels le dimensionnement de réseaux, nous ne pouvons garantir que le branchement "naturel" pour le problème initial se traduise par un branchement accommodant pour le sous-problème.

Nous avons également voulu déterminer la difficulté de résolution du problème à capacité fixée. La borne inférieure (17) étant réalisable, c'est la valeur que nous avons utilisée.

méthode	branchement	fails	temps
A	colonne / croissant	0	0 s
B	colonne / croissant	0	0 s
C	colonne / croissant	0	1 s
A	colonne / décroissant	44575	35 s
B	colonne / décroissant	44180	226 s
C	colonne / décroissant	44180	552 s
A	ligne / décroissant	-	$>$ 1h
B	ligne / décroissant	2	0 s
C	ligne / décroissant	2	1 s
A	ligne / croissant	-	$>$ 1h
B	ligne / croissant	-	$>$ 1h
C	ligne / croissant	-	$>$ 1h

Alors que l'ajout des contraintes redondantes de sac-à-dos a permis un gain de performances aisé, l'ajout de contraintes capacitaires plus générales même spécialisées à la structure du problème n'a permis que des gains modestes en regard des efforts déployés. Cet échec est-il imputable à la faiblesse des contraintes capacitaires redondantes obtenues ? A des défauts d'implémentation ?

Les contraintes redondantes obtenues sont aussi fortes que des méthodes classiques en ordonnancement, en particulier nous réduisons autant les bornes des variables que si nous avions utilisé une contrainte de *edge-finding* combinée à une *précédence énergie*, toutes deux spécialisées à notre problème. En effet, le raisonnement capacitaire est aussi général que le raisonnement énergétique, lequel moyennant un choix idoine des sous-ensembles de tâches et d'intervalles à considérer, peut recouvrir de nombreuses règles de propagation classiques en ordonnancement dont les disjonctions par paires, les tâches non-premières/non-dernières, l'edge-finding et la précédence énergie [11].

La seule contrainte d'ordonnancement dont nous ne retrouvons pas la propagation est la version étendue de la *balance constraint* [10], laquelle détecte des précédences plus fines entre évènements.

En ce qui concerne l'implantation, des essais menées avec des problèmes de gestion de projet issus de la PSPLib et transformés en BACP semblent montrer que les contraintes de capacité souffrent de la lourdeur des très nombreuses variables et contraintes supplémentaires.

Les résultats des expérimentations montrent une fois de plus la difficulté de la mise en place d'une approche automatique.

6 Conclusion

Les problèmes de dimensionnement de réseaux et d'ordonnancement contiennent tous deux en tant que sous-problème des problèmes de rangement.

La programmation linéaire utilise une méthode d'élimination de variables — dite aussi de projection – pour déduire des inégalités valides en dimensionnement de réseaux. En appliquant cette même méthode, nous avons obtenu des contraintes redondantes de type capacitaire applicables à des problèmes de programmation par contraintes. Ces contraintes capacitaires se sont avérées très proches de contraintes connues en ordonnancement.

Ainsi, les méthodes polyédriques utilisées en programmation linéaire et les raisonnements utilisés en ordonnancement se trouvent rapprochés par le biais de la méthode d'élimination de variables par sommation. La programmation linéaire semblant avoir eu tendance à procéder du général au particulier et l'ordonnancement en sens contraire.

Notre étude a cependant souligné certaines difficultés qu'il faut être en mesure de résoudre si l'on veut pouvoir mettre en oeuvre une méthode de renforcement automatique des contraintes capacitaires.

Références

[1] Karen Aardal. Reformulation of capacitated facility location problems : how redundant information can help. *Annals of Operations Research*, 82 :289–308, 1998.

[2] Egon Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8 :146–164, 1975.

[3] Harlan Crowder, Ellis. L. Johnson, and Manfred W. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5) :803–834, September-October 1983.

[4] Torsten Fahle and Meinolf Sellmann. Cost-based filtering for the constrained knapsack problem. *Annals of Operations Research*, 115 :73–93, 2002.

[5] Zonghao Gu, George L. Nemhauser, and Martin W. P. Savelsbergh. Cover inequalities for 0-1 linear programs : Computation. *INFORMS Journal on Computing*, 10 :427–437, 1998.

[6] Peter L. Hammer, Ellis L. Johnson, and Uri N. Peled. Facets of regular 0-1 polytopes. *Mathematical Programming*, 8 :179–206, 1975.

[7] Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Hybrid modelling for robust solving. *Annals of Operations Research*, 130(1-4) :19–39, 2004.

[8] ILOG. *ILOG Cplex 10.0 User Manual*, 2006.

[9] ILOG. *ILOG Solver 6.2 User Manual*, 2006.

[10] Philippe Laborie. Algorithms for propagating resource constraints in ai planning and scheduling : Existing approaches and new results. *Artificial Intelligence*, 143(2) :151–188, 2003.

[11] Pierre Lopez. Approche par contraintes des problèmes d'ordonnancement et d'affectation : structures temporelles et mécanismes de propagation, Décembre 2003. Mémoire d'Habilitation à Diriger des Recherches, LAAS.

[12] Silvano Martello and Paolo Toth. *Knapsack problems : algorithms and computer implementations*. Wiley Interscience, 1990.

[13] Michel Minoux. Discrete cost multicommodity network optimization problems and exact solution methods. *Annals of Operations Research*, 106 :19–46, 2001.

[14] Jean-François Puget. Improved bound computation in presence of several clique constraints. In *10th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 527–541, 2004.

[15] Jean-Charles Régin. Generalized arc consistency for global cardinality constraints. In *13th National Conference on Artificial Intelligence (AAAI)*, 1996.

[16] Meinolf Sellmann. Approximated consistency for knapsack constraints. In *9th International Conference on the Principles and Practice of Constraint Programming (CP)*, pages 679–693, 2003.

[17] Paul Shaw. A constraint for bin packing. In *10th International Conference on Principles and Practice of Constraint Programming (CP)*, pages 649–662, 2004.

[18] Michael A. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints. In *3rd International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 113–124, 2001.

[19] Michael A. Trick. Formulations and reformulations for integer programming. In *2nd International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 366–379, 2005.

[20] Laurence A. Wolsey. Faces of linear inequalities in 0-1 variables. *Mathematical Programming*, 8 :165–178, 1975.

[21] Zeynep Zikiltan. *Symmetry breaking ordering constraints*. PhD thesis, Uppsala University, 2004.