HAL
open science

Formulations linéaires pour la programmation par contraintes

Mohand Ou Idir Khemmoudj, Hachémi Bennaceur

To cite this version:

Mohand Ou Idir Khemmoudj, Hachémi Bennaceur. Formulations linéaires pour la programmation par contraintes. JFPC 2008- Quatrièmes Journées Francophones de Programmation par Contraintes, LINA - Université de Nantes - Ecole des Mines de Nantes, Jun 2008, Nantes, France. pp.143-151. inria-00291553

HAL Id: inria-00291553

https://inria.hal.science/inria-00291553

Submitted on 27 Jun 2008

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.

Formulations Linéaires Pour La Programmation Par Contraintes

Mohand Ou Idir Khemmoudj[†] **Hachemi Bennaceur**[‡]

[†]Laboratoire Modélisation Information et Systèmes (MIS), Université de Picardie Jules Verne
33 rue Saint Leu 80039 Amiens, France
[‡]Centre de Recherche en Informatique de Lens (CRIL) UMR 8188, Université d'Artois
rue Jean Souvraz SP 18 F 62307 Lens, France
ikhemmoud@insset.u-picardie.fr bennaceur@cril.univ-artois.fr

Résumé

Dans ce papier, nous présentons dans un premier temps des techniques génériques permettant de formuler toute contrainte d'un CSP comme un Programme Linéaire en Nombres Entiers (PLNE). Cela conduit à exploiter en Programmation Par Contraintes (PPC) de nombreux outils algorithmiques proposés par la communauté de la Recherche Opérationnelle (RO) dans le but, par exemple, de développer des contraintes globales. Ensuite, nous proposons un modèle linéaire générique pour améliorer la technique de filtrage basée sur les coûts réduits [5]. La résolution de ce modèle linéaire permet de calculer des coûts réduits plus intéressants que ceux calculés en résolvant la relaxation continue classique d'un PLNE.

Abstract

In this paper we firstly present generic techniques to formulate any constraint of a CSP as an Integer Linear Programming problem. This leads to exploit in CP many algorithm tools proposed by the Operational Research community in order, for example, to develop global constraints. Then, we propose a generic linear model to improve the reduced cost-based filtering technique [5]. The solving of this linear model allows to compute more interesting reduced costs than those computed by solving the classical continuous relaxation of an Integer Linear Programming problem.

1 Introduction

Les problèmes de satisfaction (CSP) et d'optimisation de systèmes de contraintes (CSOP) sont deux formalismes très utilisés pour la représentation de connaissances dans les domaines de l'Intelligence Artificielle (IA) et de la Recherche Opérationnelle (RO). Un CSP consiste à déterminer une affectation de variables vérifiant un ensemble de contraintes. Un CSOP est un CSP muni d'une fonction objectif exprimant des préférences entre les différentes solutions du CSP.

La Programmation Par Contraintes (PPC) est une approche énumérative permettant de résoudre les CSP et CSOP. Elle effectue une recherche arborescente de type séparation et évaluation. La séparation consiste à décomposer le problème en sous-problèmes plus faciles à résoudre. L'évaluation consiste à former une relaxation plus facile que le problème initial, sa résolution est exploitée pour la réduction de l'espace des solutions. Chacune des contraintes du problème peut être considérée comme une relaxation. Si une contrainte du problème est insatisfiable alors le problème est insatisfiable. Ainsi, toute combinaison de valeurs non autorisée par une contrainte ne peut figurer dans aucune solution du problème. Cette idée simple est très exploitée par la propagation de contraintes en PPC pour réduire l'espace de recherche de solutions.

La PPC offre la possibilité de construire des solveurs flexibles. En effet, dans ce paradigme, il est possible d'associer à chaque contrainte (ou à un sous-ensemble de contraintes) un algorithme approprié de résolution. Les algorithmes associés aux contraintes sont indépendants, ils communiquent à travers la réduction de domaines des variables communes aux contraintes.

Une méthode de filtrage de domaines à base de coûts réduits a été proposée dans [5] . Son but est de couper les branches ne conduisant pas à des solutions meilleures que la solution courante. Une valeur sera éliminée de son domaine, lorsque l'estimation de la solution partielle incluant cette valeur est supé-

rieure au coût de la meilleure solution. Les contraintes globales implémentant cette technique sont nommées Contraintes globales d'optimisation [6].

L'élimination de toutes les valeurs des variables qui ne conduisent pas à des solutions réalisables (et optimales) d'une contrainte globale (d'optimisation) revient à l'application de l'arc-consistance généralisée sur cette contrainte globale (d'optimisation). Malheureusement, en général ce problème est d'une complexité exponentielle. Une alternative pour contourner ce problème difficile est d'appliquer l'arc-consistance généralisée sur une relaxation de la contrainte.

En Recherche Opérationnelle (RO), deux types de relaxations sont très utilisées : la relaxation Lagrangienne et la relaxation continue. La relaxation Lagrangienne consiste à injecter un sous ensemble de contraintes du problème dans la fonction objectif alors que la relaxation continue consiste à relâcher la condition d'intégrité des variables. Dans le cas des contraintes linéaires, la relaxation continue donne lieu à un programme linéaire (PL) qu'on peut résoudre avec un algorithme de complexité polynomiale. Nous présentons dans ce papier des techniques génériques qui permettent de formuler toute contrainte sous forme d'un Programme Linéaire en Nombres Entiers (PLNE) [14]. Cette formulation permet d'exploiter plusieurs outils algorithmiques de la RO en PPC, dans le but par exemple, de développer des contraintes globales d'optimisation.

Pour effectuer le filtrage à base de coûts d'une relaxation continue d'un PLNE exprimant une contrainte globale d'optimisation, plusieurs PL doivent être résolus (un PL par valeur de chaque variable de la contrainte). De plus, si le processus de propagation de contraintes est exécuté jusqu'au un point fixe, un même PL peut être réoptimisé plusieurs fois. Ainsi, le temps du filtrage peut-être élevé. Cependant, il est possible d'effectuer uniquement le filtrage à base de coûts réduits [5]. Cette méthode exploite la valeur de la relaxation continue et les coûts réduits obtenus lors de son calcul. La valeur de cette relaxation est une borne inférieure pour le PLNE. Le coût réduit correspondant à une valeur est une sous-estimation d'un coût à rajouter à la borne inférieure si cette valeur participe à une solution. Lorsque cette somme (borne inférieure plus coût réduit associé à une valeur) est supérieure au coût de la meilleure solution déterminée au préalable (borne supérieure du PLNE) alors cette valeur peut être éliminée. Les coûts réduits déterminés lors de la résolution d'un PL ne permettent pas nécessairement de réaliser le filtrage le plus efficace [18]. Dans ce papier, nous illustrons cet inconvénient puis nous proposons une solution qui consiste en un PL particulier dont la résolution permet d'effectuer un filtrage à base de coûts réduits plus performant.

Ce papier est organisé de la façon suivante. Dans la section 2, nous rappelons les formalismes CSP et CSOP et certains préliminaires. La section 3 présente notre première contribution. Elle décrit deux formulations différentes d'un CSOP sous forme d'un PLNE. La première formulation est basée sur la binarisation des contraintes alors que la seconde formulation est basée sur la notion d'inégalité valide qui généralise la notion d'inégalité valide binaire proposée dans nos travaux antérieures [8, 9]. La section 4 présente la technique de filtrage à base de coûts réduits et son inconvénient ainsi qu'une solution pour y remédier. La section 5 présente des résultats expérimentaux et la section 6 conclue.

2 Formalismes CSP et CSOP

Formellement, un CSP est défini par un triplet (X, D, C) où :

1. $X = \{X_1, X_2, .., X_n\}$ est un ensemble de n variables ;
2. $D = \{D_1, D_2, ..., D_n\}$ est un ensemble de n domaines où chaque D_i représente l'ensemble fini de $d_i \leq d$ valeurs possibles pour X_i (d est la taille du plus grand domaine). Dans la suite, nous désignerons par (i, k) la valeur $k \in D_i$ de X_i ;
3. C est un ensemble de e contraintes. Chaque contrainte $\zeta \in C$ porte sur un sous-ensemble $SCOPE(\zeta) \subseteq X$ de variables et associe les valeurs 1 ou 0 à chaque tuple $I \in \prod_{X_i \in SCOPE(\zeta)} D_i$:

$$\zeta(I) = \begin{cases} 1 \text{ si } I \text{ satisfie } \zeta \\ 0 \text{ sinon} \end{cases}$$

Dans la suite, si I est un tuple dans $\prod_{X_i \in X} D_i$ alors on notera $\zeta(I) = \zeta(I_{\downarrow SCOPE(\zeta)})$ où $I_{\downarrow SCOPE(\zeta)}$ est la projection de I sur $SCOPE(\zeta)$. Par exemple, la projection de $I = (X_1 = v_1, X_2 = v_2, X_3 = v_3)$ sur (X_1, X_3) est $(X_1 = v_1, X_3 = v_3)$. Nous utiliserons la notation $I_{\downarrow X_i}$ au lieu de $I_{\downarrow \{X_i\}}$. Quand $\zeta(I) = 1$, nous dirons que I est un support ou une solution pour ζ et si de plus $I_{\downarrow X_i} = k$ alors nous dirons que I est un support pour (i, k) sur ζ.

Un tuple $I \in \prod_{X_i \in X} D_i$ est une solution du CSP si et seulement si $\sum_{\zeta \in C} \zeta(I) = |C| = e$ et le CSP est inconsistant si et seulement si $\sum_{\zeta \in C} \zeta(I) < e, \forall I \in \prod_{X_i \in X} D_i$.

Un CSOP est un CSP auquel est associée une fonction objectif $c : \prod_{X_i \in X} D_i \rightarrow \mathbb{R}$ pour exprimer des préférences entre solutions.

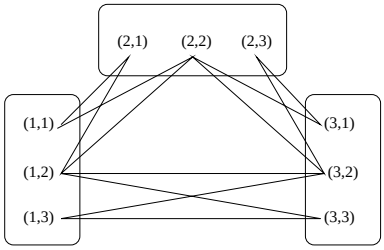

FIG. 1 – Exemple de CSP avec 3 variables X_1, X_2, X_3 possédant les valeurs 1, 2 et 3. Il contient trois contraintes binaires ζ_{12}, ζ_{13} et ζ_{23}. Les arcs expriment les incompatibilities entre valeurs : il existe un arc entre deux valeurs (i,k) et (j,l) si et seulement si $\zeta_{ij}(X_i = k, X_j = l) = 0$.

Un tuple $I \in \prod_{X_i \in X} D_i$ est une solution du CSOP si $\sum_{\zeta \in C} \zeta(I) = e$. Une telle solution est optimale si $c(I) \leq c(I'), \forall I' \in \prod_{X_i \in X} D_i$.

Remarque 1 *En définissant une variable (de coût) Obj dont le domaine est* $D_{Obj} = \prod_{X_i \in X} D_i$ *et en considérant c comme étant une contrainte unaire pondérée qui porte sur Obj et qui associe à chaque valeur* $I \in D_{Obj}$ *le poids* $c(I)$*, le CSOP peut être transformé en un autre CSOP équivalent défini par le triplet* $(X \cup \{Obj\}, D \cup \{D_{Obj}\}, C \cup \{c\})$ *et par l'objectif qui consiste à chercher une solution I au plus petit poids* $c(I)$.

Dans la suite, pour simplifier la présentation, nous considérerons que c est une combinaison de n fonctions entières $c_i : D_i \to \mathbb{N}, i = \overline{1,n}$ telles que $c(I) = \sum_{i=1}^{n} c_i(I_{\downarrow X_i})$.

3 Formulations en ProgramImation Linéaire en Nombre Entiers pour les CSOP

Dans cette section, nous présentons deux formulations sous forme de Programmes Linéaires en Nombre Entiers (PLNE) pour les CSOP. Elles introduisent pour chaque valeur (i,k) une variable booléenne $x_i(k)$:

$$x_i(k) = \begin{cases} 1 \text{ si } X_i = k \\ 0 \text{ sinon} \end{cases}$$

Nous désignerons par x le vecteur des variables booléennes $x_i(k), i = \overline{1,n}, k \in D_i$ introduites. Si $I = (X_1 = v_1, X_2 = v_2, ..., X_n = v_n)$ alors nous désignerons par x^I le vecteur à composantes $x_i^I(v_i) = 1, \forall X_i \in X$ et $x_i^I(k) = 0, \forall X_i \in X, \forall k \in D_i \setminus \{v_i\}$.

Exemple 1 *Pour le CSP de la figure 1 nous avons* $x = (x_1(1), x_1(2), x_1(3), x_2(1), x_2(2), x_2(3), x_3(1), x_3(2), x_3(3))$. *Si on considère la solution* $I = (X_1 = 1, X_2 = 3, X_3 = 3)$ *alors on a* $x^I = (1,0,0,0,0,1,0,0,1)$.

Pour exprimer le fait que chaque variable $X_i \in X$ doit prendre exactement une valeur dans D_i nous écrivons :

$$\sum_{k \in D_i} x_i(k) = 1 \tag{1}$$

Le coût d'une solution x est donné par $c(x)$:

$$c(x) = \sum_{i=1}^{n} \sum_{k \in D_i} c_i(k) x_i(k). \tag{2}$$

Un CSP peut être considéré comme étant un CSOP avec $c_i(k) = 0, \forall (i,k)$.

Les deux sous-sections suivantes décrivent deux formulations différentes pour les contraintes.

3.1 Formulation basée sur la binarisation des contraintes

La formulation basée sur la binarisation d'une contrainte $\zeta \in C$ consiste à :

1. considérer ζ comme étant une variable à domaine fini D_ζ :

$$D_\zeta = \{I \in \prod_{X_i \in SCOPE(\zeta)} D_i : \zeta(I) = 1\}$$

   D_ζ est l'ensemble des supports (solutions) de ζ ;

2. associer pour chaque support $I \in D_\zeta$ de ζ une variable booléenne $x_\zeta(I)$ telle que $\sum_{I \in D_\zeta} x_\zeta(I) = 1$ et écrire pour chaque valeur $(i,k) : X_i \in SCOPE(\zeta)$ l'égalité suivante :

$$x_i(k) = \sum_{I \in D_\zeta : I_{\downarrow X_i} = k} x_\zeta(I) \tag{3}$$

   Cette égalité exprime le fait que si $x_i(k)$ prend la valeur 1 alors un support de (i,k) sur ζ (un tuple $I \in D_\zeta : I_{\downarrow X_i} = k$) doit être sélectionné (une variable $x_\zeta(I) : \zeta(I) = 1$ et $I_{\downarrow X_i} = k$ doit prendre la valeur 1) et inversement. Elle exprime aussi le fait que si $x_i(k)$ prend la valeur 0 alors toutes les variables $x_\zeta(I)\ \forall I \in D_\zeta : I_{\downarrow X_i} = k$ doivent prendre la valeur 0 (les supports de (i,k) sur ζ doivent être éliminés) et inversement.

Le programme complet s'écrit :

$$BILP \begin{cases} \min c(x) = \sum_{i=1}^{n} \sum_{k \in D_i} c_i(k) x_i(k) \\ s.c \quad i. \quad x_i(k) = \sum_{I \in D_\zeta : I_{|X_i} = k} x_\zeta(I) \\ \qquad \forall (i,k), \forall \zeta : X_i \in SCOPE(\zeta) \\ \qquad ii. \quad \sum_{k \in D_i} x_i(k) = 1 \quad \forall X_i \in X \\ \qquad iii. \quad x_i(k) \in \{0,1\} \ \forall (i,k) \\ \qquad iv. \quad x_\zeta(I) \in \{0,1\} \ \forall \zeta \in C, \ \forall I \in D_\zeta \end{cases}$$

Remarque 2 *Le nombre de variables booléennes du programme BILP (BILP pour Binarization based Integer Linear Program) est en* $O(nd + \sum_{\zeta \in C} d^{|SCOPE(\zeta)|})$ *(*$O(nd + ed^2)$ *pour le cas de CSOP binaire) : une variable* $x_i(k)$ *pour chaque valeur* (i,k) *et une variable* $x_\zeta(I)$ *pour chaque support* I *de chaque contrainte* $\zeta \in C$ *où le nombre de supports d'une contrainte* ζ *est majoré par* $d^{|SCOPE(\zeta)|}$*. Le nombre d'égalités du BILP est majoré par* $n + d \sum_{\zeta \in C} |SCOPE(\zeta)|$ *(*$n + 2ed$ *pour les cas de CSOP binaire). Dans [10] un programme de la forme BILP est proposé pour les problèmes de satisfaction de contraintes pondérées (WCSP pour Weighted Constraint Satisfaction Problems [2]). La résolution de la relaxation continue de ce programme permet le calcul de la borne à base d'arc-consistance optimale (OSAC pour Optimal Soft Arc Consistency [4]).*

Exemple 2 *La formulation BILP du CSP de la figure 1 introduit 24 variables booléennes : une variable booléenne* $x_i(k)$ *pour chacune des 9 valeurs* $(i,k), i = \overline{1,3}, k = \overline{1,3}$ *et une variable booléenne* $x_{ij}(k,l)$ *pour chaque support des 3 contraintes. Chacune de ces 3 contraintes possède 5 supports différents. Le nombre d'égalités nécessaires de type (3) est égal à 21.*

Le principal inconvénient de $BILP$ réside dans le nombre de variables nécessaires qui est exponentiel dans le pire des cas. Cependant, cette formulation reste intéressante pour sa généricité. L'exploitation des techniques, telles que la génération de colonnes [12], pour la résolution de programmes linéaires de grande taille peut atténuer cet inconvénient.

3.2 Formulation à base d'inégalités valides

Dans nos travaux antérieurs [8, 9] nous avons introduit la notion d'inégalité valide binaire et montré comment elle peut être exploitée pour formuler les WCSP binaires et calculer des bornes inférieures pour ces problèmes. Nous montrons ici, comment cette notion peut être généralisée. Nous commençons par donner quelques définitions.

Définition 1 (Inégalité valide) *Soit* $b \in \mathbb{R}$ *et* $a = \left(a_i(k) \in \mathbb{R}, i = \overline{1,n}, k \in D_i\right)$ *un vecteur de* $\sum_{i=1}^{n} |D_i| = \sum_{i=1}^{n} d_i$ *composantes, nous dirons que l'inégalité* $ax \leq b$ *où* $ax = \sum_{i=1}^{n} \sum_{k \in D_i} a_i(k) x_i(k)$ *est valide si et seulement si pour toute solution* $I = (X_1 = v_1, X_2 = v_2, ..., X_n = v_n)$ *du CSOP* $(\sum_{\zeta \in C} \zeta(I) = e)$ *la condition* $ax^I \leq b$ *est vérifiée. Elle est une inégalité qui n'exclue aucune solution du CSOP.*

Définition 2 (Clique) *Une clique est une inégalité valide de la forme* $ax \leq 1$ *où les* $a_i(k), i = \overline{1,n}, k \in D_i$ *sont des coefficients dans* $\{0,1\}$*. Elle signifie qu'au plus une des variables* $x_i(k) : a_i(k) = 1$ *peut prendre la valeur 1. Elle est maximale si l'inégalité qu'on obtient par le remplacement d'un coefficient nul par un 1 n'est pas valide. Dans un graphe, une clique est un ensemble de sommets deux-à-deux adjacents [3].*

Exemple 3 *Pour le CSP donné par la figure 1,* $x_1(1) + x_1(2) + x_2(1) + x_2(2) \leq 1$ *est une clique binaire maximale* [1] *[8, 9] qui exprime la contrainte* ζ_{12}.

La définition du domaine d'une inégalité valide est comme suit :

Définition 3 (Domaine d'une inégalité valide) *Nous dirons qu'une instanciation complète* $I \in \prod_{X_i \in X} D_i$ *appartient au domaine d'une inégalide valide* $ax \leq b$ *si et seulement si* $ax^I \leq b$*. Nous désignerons par* $D_{ax \leq b}$ *ce domaine.*

Les instanciations partielles qui ne peuvent être étendues en instanciations complètes éléments de $D_{ax \leq b}$ sont des nogoods. Cela signifie qu'une inégalité valide est une expression compacte de tout un ensemble de nogoods.

Définition 4 (Système d'inégalités valides) *Un système* $Ax \leq b$ *de* $m \geq 1$ *inégalités* $a_j x \leq b_j, j = \overline{1,m}$ *est valide si et seulement si* $\{I \in \prod_{X_i \in X} D_i : \sum_{\zeta \in C} \zeta(I) = e\} \subseteq D_{Ax \leq b} = \bigcap_{j=1}^{m} D_{a_j x \leq b_j}$*. Un tel système augmenté par la fonction objectif (2) est une relaxation pour le CSOP. Il est équivalent au CSOP si et seulement si* $\{I \in \prod_{X_i \in X} D_i : \sum_{\zeta \in C} \zeta(I) = e\} = D_{Ax \leq b}$.

[1] L'inégalité qu'on obtient par le remplacement du coefficient nul de $x_1(3)$ ou $x_2(3)$ (ou les deux) par un 1 n'est pas valide.

Nous pouvons à présent annoncer le théorème suivant :

Théorème 1 *Toute contrainte ζ peut être exprimée par un système d'inégalités valides.*

Démonstration 1 *Soit ζ une contrainte. Elle peut être exprimée par un ensemble d'inégalités valides comme suit :*

$$\textit{Pour tout } I \in \prod_{X_i \in SCOPE(\zeta)} D_i : \zeta(I) = 0 \textit{ on écrit}$$

$$\sum_{i:X_i \in SCOPE(\zeta)} x_i(I_{\downarrow X_i}) \leq |SCOPE(\zeta)| - 1.$$

Chacune de ces inégalités valides exprime l'inconsistance d'un tuple $I \in \prod_{X_i \in SCOPE(\zeta)} D_i$ interdit par ζ.

Ainsi, l'ensemble des contraintes d'un CSOP peuvent être exprimées par un système d'inégalités valides. Si chaque contrainte $\zeta \in C$ est exprimée par un système linéaire équivalent $A_\zeta x \leq b_\zeta$ alors la formulation complète à base d'inégalités valides est comme suit :

$$VILP \begin{cases} \min c(x) = \sum_{i=1}^{n} \sum_{k \in D_i} c_i(k) x_i(k) \\ s.c \quad A_\zeta x \leq b_\zeta \quad \forall \zeta \in C \\ \qquad \sum_{k \in D_i} x_i(k) = 1 \quad \forall X_i \in X \\ \qquad x_i(k) \in \{0,1\} \; \forall (i,k) \end{cases}$$

Le nombre d'inégalités nécessaires pour exprimer une contrainte ζ sous forme d'un $VILP$ ($VILP$ pour Valid inequality based Integer Linear Program) est en $O(d^{|SCOPE(\zeta)|})$. Cependant, plusieurs contraintes usuelles peuvent être exprimées par un nombre réduit d'inégalités valides [15].

Exemple 4 *La fameuse contrainte globale* Alldif *[16] portant sur p variables $X_1, X_2, ..., X_p$ peut nécessiter un très grand nombre d'inégalités de type $\sum_{i=1}^{p} x_i(I_{\downarrow X_i}) \leq p - 1$. Cependant, en exploitant sa structure, elle peut être exprimée par seulement $m = |D_1 \cup D_2 \cup ... \cup D_p|$ cliques comme suit :*

$$\sum_{1 \leq i \leq p : k \in D_i} x_i(k) \leq 1, \forall k \in D_1 \cup D_2 \cup ... \cup D_p.$$

Pour le cas des contraintes binaires, différentes formulations à base d'inégalités valides ont été proposées [7, 9, 17].

Exemple 5 *Les contraintes binaires du CSP de la figure 1 peuvent être exprimées par le système de cliques binaires maximales [8, 9] suivant :*

$$\begin{cases} x_1(1) + x_1(2) + x_2(1) + x_2(2) \leq 1 \\ x_1(2) + x_1(3) + x_3(1) + x_3(2) \leq 1 \\ x_2(2) + x_2(3) + x_3(2) + x_3(3) \leq 1 \end{cases}$$

4 Amélioration du filtrage à base de coûts réduits

Dans cette section nous présentons la technique de filtrage à base de coûts réduits [5] et un inconvénient de cette technique. Puis nous proposons une solution pour y remédier. Sans perte de généralité, nous considérons dans cette section uniquement la formulation à base d'inégalités valides.

4.1 Filtrage à base de coûts réduits

Soit ub une borne supérieure pour un CSOP exprimé sous forme d'un programme VILP de $m \geq 1$ inégalités valides :

$$VILP \begin{cases} \min c(x) = \sum_{i=1}^{n} \sum_{k \in D_i} c_i(k) x_i(k) \\ s.c \sum_{i=1}^{n} \sum_{k \in D_i} a_{ij}(k) x_i(k) \leq b_j \quad j = \overline{1,m} \\ \quad \sum_{k \in D_i} x_i(k) = 1 \quad i = \overline{1,n} \\ \quad x_i(k) \in \{0,1\} \; \forall (i,k) \end{cases}$$

où $a_{ij}(k)$ est le coefficient de $x_i(k)$ dans l'inégalité numéro j.

Nous associons pour les m inégalités un vecteur λ de m multiplicateurs de Lagrange $\lambda_j \in \mathbb{R}^+ : j = \overline{1,m}$ et pour les n égalités un vecteur u de n multiplicateurs de Lagrange $u_i \in \mathbb{R} : i = \overline{1,n}$. En utilisant le vecteur λ pour injecter dans l'objectif les inégalités valides on obtient le problème suivant :

$$Lr(\lambda) \begin{cases} \min c(x) + \sum_{j=1}^{m} \lambda_j (\sum_{i=1}^{n} \sum_{k \in D_i} a_{ij}(k) x_i(k) - b_j) \\ s.c \sum_{k \in D_i} x_i(k) = 1 \quad i = \overline{1,n} \\ \quad x_i(k) \in \{0,1\} \; \forall (i,k) \end{cases}$$

L'objectif de $Lr(\lambda)$ peut être réécrit comme suit :

$$\begin{aligned} & -\sum_{j=1}^{m} \lambda_j b_j + \\ & \sum_{i=1}^{n} \sum_{k \in D_i} \left(c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) \right) x_i(k) \end{aligned} \tag{4}$$

Si on ajoute et soustrait la même quantité $\sum_{i=1}^{n} u_i$ à (4) alors on obtiendra l'expression équivalente suivante :

$$\sum_{i=1}^{n} u_i - \sum_{j=1}^{m} \lambda_j b_j + \sum_{i=1}^{n} \sum_{k \in D_i} \left(c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \right) x_i(k) \quad (5)$$

Les expressions (4) et (5) sont équivalentes car les contraintes de type (1) doivent être satisfaites.

On peut vérifier que $lb = \sum_{i=1}^{n} u_i - \sum_{j=1}^{m} \lambda_j b_j$ est une borne inférieure si $\sum_{i=1}^{n} \sum_{k \in D_i} \left(c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \right) x_i(k) >= 0$. En particulier, si la condition suivante est vérifiée pour chaque valeur (i, k) :

$$rc_i(k) = c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \geq 0 \quad (6)$$

$rc_i(k)$ est le coût réduit associé à (i, k) et correspondant au couple $(\lambda, u) = (\lambda_1, \lambda_2, ..., \lambda_m, u_1, u_2, ..., u_n)$. C'est une sous-estimation du coût qu'on peut ajouter à la borne inférieure lb si la valeur k est affectée à X_i. Si la somme $lb + rc_i(k)$ est supérieure ou égale à la borne supérieure ub ($lb + rc_i(k) \geq ub$) alors la valeur (i, k) peut être éliminée.

Remarque 3 *Puisque la fonction objectif est entière, la condition suffisante $lb + rc_i(k) \geq ub$ pour filtrer la valeur (i, k) peut être réécrite comme suit :*

$$lb + rc_i(k) > ub - 1. \quad (7)$$

Une question importante est : comment choisir le couple (λ, u) ? Souvent, la relaxation continue de $VILP$ est résolue pour répondre à cette question. Nous désignerons par VLP cette relaxation. Elle consiste à remplacer les contraintes d'intégrité des variables ($x_i(k) \in \{0, 1\}$ $\forall (i, k)$) par les contraintes $x_i(k) \geq 0$ $\forall (i, k)$. Les contraintes $x_i(k) \leq 1$ $\forall (i, k)$ ne sont pas nécessaires puisqu'elles sont impliquées par les contraintes de type (1).

Après résolution de VLP, une solution duale (λ, u) peut être obtenue et utilisée pour calculer des coûts réduits. Cela peut être effectué par la résolution directe du programme dual de VLP :

$$Dual \begin{cases} \max lb = \sum_{i=1}^{n} u_i - \sum_{j=1}^{m} \lambda_j b_j \\ s.c \quad rc_i(k) = c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \geq 0 \\ \qquad \forall (i, k) \\ \lambda_j \geq 0 \quad j = \overline{1, m} \end{cases}$$

Une solution optimale de *Dual* n'est pas toujours intéressante : sa valeur peut être inférieure à la borne supérieure ub et les coûts réduits associés peuvent être insuffisants pour filtrer des valeurs. De plus, *Dual* peut bien posséder une solution pas forcément optimale mais qui peut être utilisée pour filter mieux les domaines des variables [18].

Exemple 6 *Considérons encore le CSP de la figure 1. Si nous utilisons la formulation donnée par l'exemple 5 pour ses contraintes binaires alors un programme $VILP$ équivalent peut être obtenu. Le programme dual (Dual) de sa relaxation continue (VLP) consiste à maximiser $u_1 +_2 +u_3 - \lambda_1 - \lambda_2 - \lambda_3$ sous l'ensemble $\{\lambda_1 - u_1 \geq 0; \lambda_1 + \lambda_2 - u_1 \geq 0; \lambda_2 - u_1 \geq 0; \lambda_1 - u_2 \geq 0; \lambda_1 + \lambda_3 - u_2 \geq 0; \lambda_3 - u_2 \geq 0; \lambda_2 - u_3 \geq 0; \lambda_2 + \lambda_3 - u_3 \geq 0; \lambda_3 - u_3 \geq 0; \lambda_1 \geq 0; \lambda_2 \geq 0; \lambda_3 \geq 0\}$ de contraintes linéaires. On peut vérifier que $(0, 0, 0, 0, 0, 0)$ et $(1, 1, 1, 1, 1, 1)$ sont deux solutions optimales pour ce Dual. Les coûts réduits relatifs à la solution $(0, 0, 0, 0, 0, 0)$ sont tous nuls : aucune valeur ne peut être éliminée grâce à cette solution. Les coûts réduits des valeurs $(1, 2), (2, 2)$ et $(3, 2)$ relatifs à la solution $(1, 1, 1, 1, 1, 1)$ sont égaux à 1 : ces valeurs peuvent donc être éliminées grâce à cette deuxième solution puisque dans le cas de CSP une valeur peut être éliminée si seulement la borne inférieure augmentée par son coût réduit dépasse 0. L'arc-consistance ne permet de filtrer aucune valeur dans cet exemple.*

Exemple 7 *Considérons un CSP où $X = \{X_1, X_2, X_3\}, D_1 = D_2 = \{1, 2\}, D_3 = \{1, 2, 3\}, C = \{Alldif(X_1, X_2, X_3)\}$ que nous formulons comme suit :*

$$\begin{cases} \min 0 \\ s.c \quad x_1(1) + x_2(1) + x_3(1) \leq 1 \\ \qquad x_1(2) + x_2(2) + x_3(2) \leq 1 \\ \qquad x_1(1) + x_1(2) = 1 \\ \qquad x_2(1) + x_2(2) = 1 \\ \qquad x_3(1) + x_3(2) + x_3(3) = 1 \\ \qquad x_1(1), x_1(2), x_2(1), x_2(2), \\ \qquad x_3(1), x_3(2), x_3(3) \in \{0, 1\} \end{cases}$$

Le Dual associé à la relaxation continue VLP de ce $VILP$ consiste à maximiser $u_1 + u_2 + u_3 - \lambda_1 - \lambda_2$ sous l'ensemble $\{\lambda_1 - u_1 \geq 0; \lambda_2 - u_1 \geq 0; \lambda_1 - u_2 \geq$

$0; \lambda_2 - u_2 \geq 0; \lambda_1 - u_3 \geq 0; \lambda_2 - u_3 \geq 0; -u_3 \geq 0; \lambda_1 \geq 0, \lambda_2 \geq 0\}$ *de contraintes linéaires. On peut vérifier que* $(0,0,0,0,0)$ *et* $(1,1,1,1,0)$ *sont deux solutions différentes pour ce Dual. Les coûts réduits relatifs à la première solution sont nuls. C'est une solution qui ne permet de filtrer aucune valeur. Les coûts réduits relatifs à la deuxième solution sont égaux à 1 pour le cas des valeurs* $(3,1)$ *et* $(3,2)$*. Ces deux valeurs peuvent donc être filtrées grâce à cette deuxième solution. Ce sont les mêmes valeurs que l'arc-consistance généralisée [13] peut filtrer.*

La technique proposée dans [18] exploite la relaxation Lagrangienne. Le problème dual Lagrangien est résolu par une méthode itérative. Elle vérifie en chaque itération si des valeurs peuvent être éliminées. Cependant, l'objectif premier de cette technique est de maximiser la valeur de la borne inférieure et non pas de détecter des valeurs à éliminer.

Nous proposons une nouvelle fonction objectif à optimiser dans le but de guider la recherche vers une solution (λ, u) de *Dual* qui permet de filtrer mieux le problème.

4.2 Filtrage par coûts réduits amélioré

Considérons le programme linéaire suivant :

$$\overline{VLP}\begin{cases} \min 0 \\ s.c \sum_{i=1}^{n} \sum_{k \in D_i} c_i(k) x_i(k) \leq ub - 1 \\ \quad \sum_{i=1}^{n} \sum_{k \in D_i} a_{ij}(k) x_i(k) \leq b_j \quad j = \overline{1,m} \\ \quad \sum_{k \in D_i} x_i(k) = 1 \quad i = \overline{1,n} \\ \quad x_i(k) \geq 0 \; \forall (i,k) \end{cases}$$

Ce programme est une relaxation du CSP composé des contraintes du VLP et de la contrainte $c(x) \leq ub - 1$. Son dual est le suivant :

$$\overline{Dual}\begin{cases} \max \overline{lb} = \sum_{i=1}^{n} u_i - \lambda_0(ub-1) - \sum_{j=1}^{m} \lambda_j b_j \\ s.c \quad \overline{rc}_i(k) = \lambda_0 c_i(k) + \\ \qquad \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \geq 0 \;\; \forall (i,k) \\ \\ \quad \lambda_j \geq 0 \quad j = \overline{1,m} \end{cases}$$

Théorème 2 *Si* $(\lambda_0, \lambda_1, ..., \lambda_m, u_1, u_2, ..., u_n)$ *est une solution de* $\overline{Dual}$ *et si*

$$\overline{rc}_i(k) + \overline{lb} > 0 \tag{8}$$

alors la valeur (i,k) *est inconsistante.*

Démonstration 2 *Le programme* $\overline{Dual}$ *est le dual d'une relaxation d'un CSP. Ainsi, étant que 1 est forcément une borne supérieure alors si la borne inférieure augmentée par le coût réduit d'une valeur dépasse 0 alors cette valeur est inconsistante et peut être filtrée.*

Théorème 3 *Si* $(\lambda, u) = (\lambda_1, ..., \lambda_m, u_1, u_2, ..., u_n)$ *:* $\lambda \geq 0$ *est une solution de Dual permettant d'éliminer une valeur donnée* (j,l) *alors* $(1, \lambda, u) = (1, \lambda_1, ..., \lambda_m, u_1, u_2, ..., u_n)$ *est une solution de* $\overline{Dual}$ *permettant d'éliminer* (j,l) *également.*

Démonstration 3 *Soit* (λ, u) *:* $\lambda \geq 0$ *une solution de Dual permttant d'éliminer* (j,l)*. Cela signifie que :*

1. $rc_i(k) = c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \geq 0, \; \forall (i,k)$ *(contraintes (6)) ;*
2. $lb + rc_j(l) > ub - 1$ *(la condition (7) est vérifiée pour* (j,l)*) avec* $lb = \sum_{i=1}^{n} u_i - \sum_{j=1}^{m} \lambda_j b_j$.

On peut vérifier que :

1. $(1, \lambda, u)$ *est une solution de* $\overline{Dual}$*. En effet, puisque* $\lambda_0 = 1$ *et* $rc_i(k) \geq 0, \; \forall (i,k)$ *nous avons :* $\overline{rc}_i(k) = c_i(k) + \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i = rc_i(k) \geq 0 \; \forall (i,k)$ *;*
2. $\overline{lb} + \overline{rc}_j(l) > 0$ *(la condition (8) est vérifiée pour* (j,l)*). En effet, puisque* $\lambda_0 = 1, \overline{rc}_j(l) = rc_j(l)$ *et* $lb + rc_j(l) > ub - 1$ *nous avons* $\overline{lb} + \overline{rc}_j(l) = \sum_{i=1}^{n} u_i - (ub-1) - \left(\sum_{j=1}^{m} \lambda_j b_j \right) + rc_j(l) = lb + rc_j(l) - (ub-1) > 0$.

Grâce aux théorèmes 2 et 3, il est maintenant clair que, dans le but de filtrer des valeurs, vérifier la condition (8) est au moins aussi intéressant que vérifier la condition (7).

La condition (8) est issue du programme $\overline{Dual}$ dont la fonction objectif est nulle (le problème initial est transformé en un problème de satisfaction). Nous proposons maintenant une fonction objectif à exploiter pour mieux filtrer le problème. Pour cela, nous associons pour chaque valeur (i,k) une variable $\delta_i(k) \geq 0$ et nous considérons la nouvelle condition suivante :

$$\overline{lb} + \overline{rc}_i(k) + \delta_i(k) \geq 1 \tag{9}$$

Remarque 4 *Si* $\delta_i(k) < 1$ *et si la condition (9) est vérifiée alors la condition (8) est aussi vérifiée. De même, si la condition (8) est vérifiée alors on peut choisir* $\delta_i(k) < 1$ *pour que la condition (9) soit vérifiée également.*

À présent, nous considérons le nouveau programme linéaire suivant :

$$\overline{\overline{Dual}} \begin{cases} \min \sum_{i=1}^{n} \sum_{k \in D_i} \delta_i(k) \\ s.c \quad \overline{lb} = \sum_{i=1}^{n} u_i - \lambda_0(ub-1) - \sum_{j=1}^{m} \lambda_j b_j \\ \quad \overline{lb} + \overline{rc}_i(k) + \delta_i(k) \geq 1 \quad \forall (i,k) \\ \quad \overline{rc}_i(k) = \lambda_0 c_i(k) + \\ \qquad \sum_{j=1}^{m} \lambda_j a_{ij}(k) - u_i \geq 0 \quad \forall (i,k) \\ \\ \quad \lambda_j \geq 0 \quad j = \overline{1,m} \\ \quad \delta_i(k) \geq 0 \quad \forall (i,k) \end{cases}$$

Théorème 4 *La valeur optimale $V(\overline{\overline{Dual}})$ de $\overline{\overline{Dual}}$ et au plus égale à $\sum_{i=1}^{n} |D_i|$ et au moins $\omega = \left\lceil \sum_{i=1}^{n} |D_i| - V(\overline{\overline{Dual}}) \right\rceil$ valeurs peuvent être filtrées après la résolution de $\overline{\overline{Dual}}$ à l'optimalité.*

Démonstration 4 *En choisissant $\lambda_j = 0, \forall j = \overline{0,m}, u_i = 0, \forall i = \overline{1,n}$ et $\delta_i(k) = 1 \forall (i,k)$, on obtient une solution de coût $\sum_{i=1}^{n} |D_i|$ pour $\overline{\overline{Dual}}$. Ainsi, $V(\overline{\overline{Dual}})$ est au plus égale à $\sum_{i=1}^{n} |D_i|$ et si $V(\overline{\overline{Dual}}) < \sum_{i=1}^{n} |D_i|$ alors il existe forcément au moins une valeur $(i,k) : \delta_i(k) < 1$.*

Supposons que nous avons une solution optimale de coût $V(\overline{\overline{Dual}}) = \sum_{i=1}^{n} \sum_{k \in D_i} \delta_i(k)$ et considérons l'algorithme suivant :

1. $\alpha = 0$, $\beta = 0$;
2. ***For*** $i = \overline{1,n}$
3. $\quad E_i = D_i$;
4. $\quad$ ***Tant que***$(E_i \neq \emptyset)$
5. $\quad\quad$ *choisir une value k dans E_i, $E_i \leftarrow E_i \backslash \{k\}$;*
6. $\quad\quad$ ***Si*** $(\delta_i(k) < 1)$ $\alpha = \alpha + 1$; `% La valeur` (i,k) `peut être éliminée`
7. $\quad\quad$ ***Sinon*** $\beta = \beta + \delta_i(k)$

*Après l'exécution de cet algorithme, α est égal au nombre de valeurs qu'on peut éliminer et nous avons forcément $\alpha + \beta \geq \sum_{i=1}^{n} |D_i|$. En effet, la somme $\alpha + \beta$ augmente par au moins 1 à chaque itération de la boucle **Tant que**. Ainsi, $\alpha \geq \left\lceil \sum_{i=1}^{n} |D_i| - \beta \right\rceil \geq \left\lceil \sum_{i=1}^{n} |D_i| - \sum_{i=1}^{n} \sum_{k \in D_i} \delta_i(k) \right\rceil = \omega$, puisque α est entier et $\beta \leq \sum_{i=1}^{n} \sum_{k \in D_i} \delta_i(k)$ (β augmente au moins par $\delta_i(k)$ à chaque itération de la boucle **Tant que**).*

Exemple 8 *Pour le CSP donné par la figure 1 où le nombre de valeurs (i,k) différentes est égal à 9, on peut vérifier que $V(\overline{\overline{Dual}}) = 6$ et $3 = 9 - 6$ est le nombre de valeurs qui peuvent être éliminées. Ces valeurs sont $(1,2), (2,2)$ et $(3,2)$. On peut aussi vérifier que, pour le CSP de l'exemple 7 où le nombre de différentes valeurs (i,k) est 7 et les variables doivent prendre des valeurs différentes, que $V(\overline{\overline{Dual}}) = 5$ et $2 = 7-5$ est le nombre de valeurs qui peuvent être éliminées. Ces valeurs sont $(3,1), (3,2)$.*

Remarque 5 *Si la valeur optimale de VLP est supérieure ou égale à la borne supérieure ub alors le programme $\overline{\overline{Dual}}$ possède une solution de valeur nulle : toutes les valeurs peuvent être éliminées.*

Le programme $\overline{\overline{Dual}}$ est très intéressant : les nombres de ses variables et contraintes sont du même ordre que ceux du programme $\overline{Dual}$. Il peut être exploité pour développer des contraintes globales.

5 Expérimentations

Nous avons comparé le filtrage classique à base de coûts réduits (**C-RCBF** pour **C**lassical **R**educed **C**ost-**b**ased **F**iltering) avec l'amélioration que nous venons de proposer pour ce filtrage (**I-RCBF** pour **I**mproved **R**educed **C**ost-**B**ased **F**iltering). Les problèmes testés sont des CSP binaires générés de manière aléatoire selon le protocole défini dans [11]. Trois classes ont été considérées. Elles sont caractérisées par la densité des graphes de contraintes : peu denses pour la première classe, denses pour la deuxième et complets pour la troisième. Le nombre de variables n est choisi égal à 40 pour les CSP de la première classe, à 30 pour ceux de la deuxième et à 25 pour ceux de la troisième. La taille des domaines est toujours choisie égale à 10 (pour les trois classes). La dureté des contraintes est choisie égale à la dureté minimale à partir de laquelle les problèmes générés sont difficiles à résoudre. Nous noterons ces classes SL (pour Sparse Loose), DL (pour Dense Loose) et CL (pour Complet Loose) respectivement (voir http ://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/softCSP pour plus de détailles). Pour chaque classe, 50 CSP différents sont générés. Nous avons exploité la formulation basée sur la binarisation des contraintes. Pour la résolution des programmes linéaires obtenus, nous avons utilisé la méthode barrier de CPLEX.

# valeurs filtrées	SL	DL	CL
C-RCBF	0	0	0
I-RCBF	$10,4$	$0,74$	0

temps CPU (s)	SL	DL	CL
C-RCBF	$0,497$	$0,86$	$10,65$
I-RCBF	$1,4094$	$2,01$	$18,97$

FIG. 2 – Résultats obtenus sur les CSP testés.

Pour chaque classe, la première table sur la figure 2 présente le nombre moyen de valeurs filtrées par chacune des deux techniques **C-RCBF** et **I-RCBF**. La deuxième table présente les temps en secondes que nécessitent ces techniques. Pour les problèmes de la classe CL, aucune des deux méthodes n'a pu filtrer des valeurs. Pour les autres classes, surtout la classe SL, notre méthode arrive à filtrer des valeurs alors que la technique classique est impuissante. Cependant, notre technique nécessite plus de temps. Cela est dû au fait qu'elle résout des programmes linéaires de taille plus grande.

6 Conclusion

Dans ce papier, nous avons proposé des formulations génériques pour les CSOP. Ce sont des formulations qui peuvent être exploitées pour développer des contraintes globales. Nous avons également proposé une solution à un inconvénient de la technique de filtrage grâce aux coûts réduits qui est très exploitée pour développer des contraintes globales d'optimisation. Cet inconvénient réside dans le fait que lorsqu'un un programme linéaire est résolu, les coûts réduits obtenus ne sont pas forcément les meilleurs qu'ont peut utiliser pour effectuer du filtrage.

La solution que nous proposons est un programme linéaire avec une fonction objectif dont l'optimisation permet de calculer de bons coûts réduits pour réduire les domaines. Le développement de contraintes globales en exploitant la technique que nous avons proposée reste une perspective pour ce travail. L'exploitation de cette technique dans le cadre d'une méthode combinant la relaxation Lagrangienne et la Programmation Par Contraintes (CP-based Lagrangian relaxation [18]) est une autre perspective intéressante.

Références

[1] Beldiceanu, N., Carlsson, M., Rampon, J.X. : Global Constraint Catalog, Technical report T2005 :08, SICS (2006).

[2] Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., Fargier, H. : Semiring-Based CSPs and Valued CSPs : Frameworks, Properties, and Compar- ison, Constraints, 4-3 (1999) 199-240.

[3] Bomze, I., Budinich, M., Pardalos, P., Pelillo, M. : The maximum clique problem, Handbook of Combinatorial Optimization, 4 (1999).

[4] Cooper, M., de Givry, S., Schiex, T. : Optimal soft arc consistency. IJCAI (2007) 68-73.

[5] FOCACCI, F., LODI A., MILANO M. : Cost-based domain filtering CP (1999) 189-203.

[6] FOCACCI, F., LODI A., MILANO M. : Optimization-Oriented Global Constraints, Constraints, 7 :3-4 (2002) 351-365.

[7] Khemmoudj, M.O.I., Bennaceur, H., Nagih, A. : Combining Arc-Consistency and Dual Lagrangean Relaxation for Filtering CSPs. CPAIOR (2005) 258-272.

[8] Khemmoudj, M.O.I., Bennaceur, H. : Clique Inference Process for Solving Max- CSP. CP (2006) 746-750.

[9] Khemmoudj, M.O.I., Bennaceur, H. : Valid Inequality Based Lower Bounds for WCSP. CP (2007) 394-408.

[10] Koster, A. : Frequency Assignment : Models and Algorithms, Phd Thesis(1999).

[11] Larrosa, J., Schiex, T. : In the quest of the best form of local consistency for Weighted CSP. *IJCAI* (2003) 239-244.

[12] Maculan, N., de Mendona Passini, M., Andr de Moura Brito, J., Loiseau, I. : Column Generation in Integer Linear Programming, RAIRO-Operations Research 37(2003) 67-83.

[13] Mohr, L., Masini, G. : Good old discrete relaxation. ECAI (1988) 651-656.

[14] Nemhauser, G.L., Wolsey, L.A. : Integer and Combinatorial Optimization, (1999).

[15] Refalo, P. : Linear Formulation of Constraint Programming Models and Hybrid Solvers. CP (2000) 369-383.

[16] 16. Régin, J.C. : A Filtering Algorithm for Constraints of Diference in CSPs, AAAI (1994) 362-367.

[17] Sellmann, M. : A Totally Unimodular Description of the Consistent Value Polytope for Binary Constraint Programming, CPAIOR (2006) 16-28.

[18] Sellmann, M. : Theoretical Foundations of CP-Based Lagrangian Relaxation. CP (2004) 634-647.