HAL
archives-ouvertes.fr

Contrainte globale pour le problème de recouvrement d'ensemble

Sébastien Mouthuy, Yves Deville, Grégoire Dooms

▶ **To cite this version:**

Sébastien Mouthuy, Yves Deville, Grégoire Dooms. Contrainte globale pour le problème de recouvrement d'ensemble. Troisièmes Journées Francophones de Programmationpar Contraintes (JFPC07), Jun 2007, INRIA, Domaine de Voluceau, Rocquencourt, Yvelines France, 2007, JFPC07. <inria-00151182>

HAL Id: inria-00151182
https://hal.inria.fr/inria-00151182

Submitted on 1 Jun 2007

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.

Contrainte globale pour le problème de recouvrement d'ensemble

Sébastien Mouthuy **Yves Deville** **Grégoire Dooms**

Département d'ingénierie informatique
Université catholique de Louvain
B-1348 Louvain-la-Neuve - Belgium

sebastien.mouthuy@student.uclouvain.be, {yves.deville,dooms}@info.ucl.ac.be

Abstract

Ce papier propose une approche par Programmation par Contrainte pour résoudre le problème de recouvrement d'ensemble. Ce problème d'optimisation combinatoire est fort utile pour formuler un grand nombre d'applications concrètes (assignation d'équipages, planification de tâches et de véhicules, construction de circuits imprimés) ainsi que des problèmes de graphes (Recouvrement par noeuds, ensemble de noeuds dominants et indépendants). Le problème de recouvrement d'ensemble est NP-difficile. Ce papier propose une contrainte globale SC pour le problème de recouvrement d'ensemble et un propagateur qui utilise une borne inférieure calculée par des relaxations empruntées à la Programmation Entière. Nous présentons aussi un algorithme incrémental pour calculer cette borne qui utilise une structure de données qui peut être mise à jour très vite pour des petits changements, la rendant très bien adaptée aux arbres de recherche. Notre approche est comparée avec deux autres propagateurs basés sur la relaxation linéaire et sur une approche gloutonne.

1 Introduction

Le problème de recouvrement d'ensemble (SC) peut être défini comme suit. Soit $\mathcal{U} = \{1, \ldots, m\}$ un ensemble de m éléments. Soit X une collection de sous-ensembles de $\mathcal{U}$: $X = \{S_1, \ldots, S_n\}$ où $S_i \subseteq \mathcal{U}, (1 \leq i \leq n)$ et soit $(c_j)_{1 \leq j \leq n}$ des poids associés aux sous-ensembles de la collection X. Une solution au problème SC est un sous-ensemble T d'indices de X tel que tous les éléments de $\mathcal{U}$ soient recouverts : $T \subseteq \{1, \ldots, n\} \mid \bigcup_{i \in T} S_i = \mathcal{U}$ et tel que $\sum_{j \in T} c_j$ est minimum. Un exemple d'une instance d'un tel problème est illustré à la Figure.1.

$$\mathcal{U} = \{1, 2, 3, 4, 5\}$$
$$S_1 = \{1, 3, 5\},\ S_2 = \{1, 2, 4\},\ S_3 = \{5, 2\},$$
$$S_4 = \{1, 3, 2\}$$
$$c_j = 1, \quad (1 \leq j \leq 4)$$

$$Sol = \{1, 2\}$$

FIG. 1 – Exemple d'un problème de recouvrement d'ensemble. Il est évident qu'il n'y a pas de sous-ensemble qui recouvre $\mathcal{U}$ tout seul. On observe que $S_1 \cup S_2 = \mathcal{U}$, donc $\{1, 2\}$ est un recouvrement minimum de $\mathcal{U}$.

Enormément d'applications peuvent être formulées à l'aide d'un problème de recouvrement : assignation de personnel et planification [25, 11, 6], construction optimale de circuits imprimés [22], positionnement d'équipement d'urgence [12, 26], problème de planification routière. De plus, un grand nombre de problème de la théorie des graphes peuvent être formulés aisément à l'aide de SC : recouvrement par noeuds, ensemble de noeuds dominants et indépendants.

Beaucoup de travail a été fait pour résoudre ce problème soit d'une manière complète soit par heuristique, généralement dans le cadre de la Programmation Entière. Deux excellentes revues sont [5, 7]. La formulation utilisée en Programmation Entière est

$$SC^* = \min_{x \in \{0,1\}^n} \sum_{1 \leq j \leq n} c_j x_j \quad tel\ que \qquad (SC)$$

$$\sum_{1 \leq j \leq n} a_{ij} x_j \geq 1 \quad \forall i \in \mathcal{U} \qquad (1)$$

$$x_j \in \{0,1\} \quad \forall j = 1 \ldots, n \tag{2}$$

où $a_{ij} = 1 \iff i \in S_j$.

Les problèmes de recouvrement d'ensemble sont très difficiles. Les résultats explicités dans [1] impliquent qu'il n'existe pas d'algorithme d'approximation en temps polynomial, à moins que $P = NP$; c'est à dire qu'il existe une constante $\epsilon > 0$ telle que le problème ne peut pas être approximé en temps polynomial avec une précision supérieure à $1 + \epsilon$. D'autres résultats négatifs à propos de cette complexité peuvent être trouvés dans [19].

Ce papier définit la contrainte SC, une contrainte globale pour le problème de recouvrement d'ensemble. Un propagateur pour cette contrainte est aussi présenté. Il utilise une technique de singleton consistante (appelé *shaving* en anglais) et une borne obtenue par une relaxation empruntée à la Programmation Entière. Le principal avantage de cet algorithme est que la structure de données interne qu'il utilise peut être mise à jour de manière incrémentale, ce qui fait que la borne peut être recalculée très rapidement tout le long de l'arbre de recherche.

Cet algorithme a été implémenté dans un cadre CP. Cela permet de spécifier des problèmes liés au problème de recouvrement d'ensemble très simplement sous la forme de CSP. L'avantage de ce paradigme de programmation est que dès qu'il existe une contrainte globale SC, on peut modéliser des problèmes de la vie réelle et ajouter autant de contraintes auxiliaires qu'on le veut sans pour autant diminuer l'efficacité.

A la Section 2, nous décrivons une approche CP générale pour SC et le travail qui a déjà été fait dans cette direction. La Section 3 décrit une relaxation de SC et présente un algorithme incrémental pour le résoudre efficacement. La Section 4 présente les résultats expérimentaux de notre approche CP.

2 Un approche CP pour SC

2.1 Approche Générale

L'objectif de cette approche CP est la conception d'une contrainte globale pour SC de la forme $SC(N, T, \mathcal{U}, X = \{S_1, \ldots, S_n\}, Cost)$ où

- $\mathcal{U} = \{1, \ldots, m\}$ est une ensemble d'éléments
- N est un entier
- X est une collection de sous-ensembles $S_i \subseteq \mathcal{U}, (1 \leq i \leq n)$
- T est un sous-ensemble d'indices de X : $T \subseteq \{1, \ldots, n\}$
- $Cost$ est une fonction qui spécifie un poids pour chaque sous-ensemble S_i ; $Cost : X \rightarrow \mathbb{R}$. Pour plus de simplicité nous dénoterons $Cost(S_i)$ par c_i.

Cette contrainte est respectée si et seulement si

$$\mathcal{U} = \bigcup_{i \in T} S_i \tag{3}$$

$$\sum_{i \in T} c_i \leq N \tag{4}$$

Dans la suite, N et T seront considérés comme des variables et $\mathcal{U}$, $S_i (i = 1, \ldots, n)$ et $Cost$ seront considérés comme donnés. N est une variable à domaine fini (FD) et T est une variable d'ensemble [10, 23]. $\overline{N}$ dénote la borne supérieure de N et $\underline{N}$ dénote sa borne inférieure : $\underline{N} \leq N \leq \overline{N}$. Nous utilisons les mêmes notations pour T : $\underline{T} \subseteq T \subseteq \overline{T}$.

Nous observons que (3) est très facile à satisfaire, alors que pour des petites valeurs de N, trouver un T satisfaisant (4) est très difficile. C'est pourquoi nous avons besoin d'une bonne borne inférieure sur N, afin de pouvoir élaguer le domaine de N et T.

L'objectif de l'algorithme de filtrage pour la contrainte globale SC est d'élaguer le domaine de N et de T en retirant les valeurs qui n'appartiennent à aucune solution de la contrainte.

2.1.1 Elagage dans le domaine de N

D'après la structure de SC, il est facile de voir que la meilleure borne supérieure sur N est $N \leq \sum_{i \in \overline{T}} c_i$. En effet, il existe une solution à la contrainte (cad une assignation des variable N et T, et telle que (3) et (4) soient repectées), alors une solution à la contrainte serait donnée par $N = \sum_{i \in \overline{T}} c_i, T = \overline{T}$. La règle de filtrage est donc $\overline{N} = \min(\overline{N}, \sum_{i \in \overline{T}} c_i)$.

D'un autre côté calculer la borne inférieure optimale pour N est très difficile, cela revient à résoudre le problème d'optimisation SC (cad trouver SC^*), ce qui est NP-difficile [13]. Dans ce papier, nous utiliserons différentes approximation de SC^* pour élaguer le domaine des variables. Pour trouver une borne inférieure de SC^*, nous résolvons une relaxation de SC. Notons cette relaxation par $SCRel$ et le coût de sa solution optimale par $SCRel^*$. Nous avons ainsi

$$lb_{SC} = SCRel^* \leq SC^*$$

La règle de filtrage est $\underline{N} = \max(\underline{N}, lb_{SC})$.

2.1.2 Elagage du domaine de T

T est une variable d'ensemble. L'algorithme de filtrage devrait élaguer le domaine en fonction des contraintes (3) et (4). A partir de (3) on peut déduire les deux règles de filtrages suivantes :

(P1) La contrainte SC n'est pas respectée si $\mathcal{U}$ ne peut pas être couvert par les sous-ensembles disponibles : $\exists e \in \mathcal{U}, \forall i \in T, e \notin S_i \rightarrow fail$

(P2) Comme les indices de T doivent recouvrir $\mathcal{U}$, quand il y a seulement un sous-ensemble qui contient un élément donné, il doit faire partie de la solution : $\exists e \in \mathcal{U}, \exists! i : \ e \in S_i \rightarrow i \in T$

Sans la contrainte (4), ces règles de filtrage permettrait d'atteindre l'arc-consistence (toute solution partielle pourrait être étendue en une solution admissible). Afin de filtrer la contrainte (4), deux autres règles doivent être considérées :

(PA) Retirer du domaine de T tous les indices des sous-ensembles qui n'appartiennent à aucune solution de coût dans $[\underline{N}; \overline{N}]$: $\overline{T} \leftarrow \overline{T} \setminus \{i \in \overline{T} \setminus \underline{T} \mid \neg SC(N, T \cup \{i\}, \mathcal{U}, X, Cost)\}$

(PB) Mettre dans T les indices i tels que S_i appartient à tous les recouvrements de coût dans $[\underline{N}; \overline{N}]$: $\underline{T} \leftarrow \underline{T} \cup \{i \in \overline{T} \setminus \underline{T} \mid \neg SC(N, T \setminus \{i\}, \mathcal{U}, X, Cost)\}$

Malheureusement, nous avons vu qu'atteindre l'arc-consistence pour la contrainte SC est NP-difficile, ainsi les deux dernières règles de filtrage peuvent n'être qu'approximées. Nous utilisons une borne inférieure sur N pour filtrer la contrainte :

(P'A) $\overline{T} \leftarrow \overline{T} \setminus \{i \in \overline{T} \setminus \underline{T} \mid lb_{SC}(N, T \cup \{i\}, \mathcal{U}, X, Cost) > \overline{N}\}$

(P'B) $\underline{T} \leftarrow \underline{T} \cup \{i \in \overline{T} \setminus \underline{T} \mid lb_{SC}(N, T \setminus \{i\}, \mathcal{U}, X, Cost) > \overline{N}\}$

Le filtrage (P'A) (resp. (P'B)) peut être fait au moyen d'une technique de singleton consistance : on retire de $\overline{T}$ (resp. on ajoute dans $\underline{T}$) chaque valeur $i \in \overline{T} \setminus \underline{T}$ une par une et on recalcule la nouvelle borne inférieure lb_i de N. Si $lb_i > \overline{N}$, alors on doit ajouter i dans $\underline{T}$ (resp. retirer i de $\overline{T}$).

2.2 Approches CP Existantes

A notre connaissance, la communauté scientifique ne s'est jamais penchée directement sur le problème du recouvrement d'ensemble par une approche de Programmation pas contraintes. Cependant quelques papiers ont été écrits sur une contrainte similaire $NValue$ qui compte le nombre de valeurs distinctes prises par une collection de variables. Cette contrainte fut introduite dans [21] et est une généralisation de la contrainte $AllDiff$ [24, 28].

Bessière et al. [4] ont décomposé $NValue$ en deux contraintes : $AtLeastNValue(N, X = \{X_1, \ldots, X_m\})$ et $AtMostNValue(N, X = \{X_1, \ldots, X_m\})$ où N est un entier et les X_i sont des variables à domaine fini ayant $D(X_i)$ comme domaine. $AtMostNValue$ tient si l'assignation des m variables X_i utilise au plus N différentes valeurs. Formellement, $AtMostNValue$ tient si et seulement si $|\{X_i : 1 \leq i \leq m\}| \leq N$ et $AtLeastNValue$ si et seulement si $|\{X_i : 1 \leq i \leq m\}| \geq N$. Nous montrons ici que $AtMostNValue$ est équivalent à SC avec des coûts unitaires ($c_j = 1, \forall j$).

Proposition 1 *Unicost SC est strictement équivalent à $AtMostNValue$.*

Preuve Soit $\mathcal{U} = \{1, \ldots, m\}$, $S_i = \{j \mid i \in D(X_j)\}, \forall i \in \bigcup_{1 \leq i \leq m} D(X_i)$ et $Cost : X \rightarrow \mathbb{R} : Cost(S_i) = 1, \forall S_i \in X$. $AtMostNValue$ est respectée si et seulement si $SC(N, T, \mathcal{U}, X, Cost)$ est respecté. ■

Comme résoudre le problème de recouvrement d'ensemble de manière optimale est NP-difficile, cela montre que la valeur minimum pour N est aussi NP-difficile à calculer.

Tout algorithme de filtrage développé pour une de ces contraintes peut facilement être utilisé pour filtrer l'autre contrainte.

2.2.1 Calculer une borne inférieure pour N dans $AtMostNValue$

Avant de décrire les approches existantes pour calculer une borne inférieure sur N, nous définissons le graphe d'intersection $G_X = (V, E)$ d'une collection de variables $X = \{X_1, \ldots, X_m\}$, où $V = \{1, \ldots, m\}$ et $E = \{(i, j) \mid D(X_i) \cap D(X_j) \neq \emptyset\}$. Le lien entre le graphe d'intersection et une instance donnée de SC - lorsque les coûts sont unitaires - est que trouver un ensemble de noeuds indépendants de cardinalité maximale dans G_X est équivalent à résoudre le problème dual de SC, le problème d'exclusion d'ensemble (Set Packing Problem) dont la formulation est

$$SP^* = \max_{y \in \{0,1\}^m} \sum_{1 \leq i \leq m} y_j \qquad tel\ que \qquad (SP)$$
$$y_i + y_j \leq 1 \quad \forall (i, j) \in E$$
$$y_i \in \{0, 1\} \quad \forall 1 \leq i \leq m$$

Le problème d'exclusion d'ensemble est NP-difficile. Soit SC^*(resp. SP^*) la solution optimale de SC (resp. SP), $SCLin^*$ (resp. $SPLin^*$) la solution optimale de la relaxation linéaire de SC (resp. de SP). La théorie de l'optimisation [29] nous donne que

$$SP^* \leq SPLin^* = SCLin^* \leq SC^* \qquad (5)$$

Donc une borne inférieure de SP^* est aussi une borne inférieure pour SC^*. Si nous définissons $\alpha(G)$ comme le nombre d'indépendance du graphe G (le nombre maximal de noeuds de G que nous pouvons prendre tels qu'ils ne sont pas voisins deux à deux), alors nous obtenons à partir de (5) que $\alpha(G_X) = SP^* \leq SC^*$.

2.3 Quatre bornes inférieures pour N

Quatre approches pour calculer une borne inférieure de SC^* ont été reprises dans [4]. Celles-ci seront comparées à notre approche présentée à la Section 3.

Relaxation linéaire de SC ($SCLin^*$) Cette relaxation s'obtient en relaxant la contrainte d'intégralité ; cad remplacer la contrainte (2) $x_j \in \{0,1\}, \forall i$ par $0 \leq x_j \leq 1, \forall i$. D'après (5), $SCLin^*$ est une borne inférieure de SC^* qui est meilleure que n'importe quelle borne de SP^*. Elle est néanmoins relativement onéreuse à calculer.

Intervalles ordonnés (OI) Dans [2], Beldiceanu se penche sur $NValue$ dans le cas précis où les domaines des variables X_i sont des intervalles. Dans ce cas particulier il atteint la borne-consistence en temps polynomial. Soit $I = \{I_1, \ldots, I_m\}$, m variables dont les domaines sont des intervalles, alors cet algorithme calcule $\alpha(G_I)$. Nous pouvons l'utiliser dans le cas général en approximant les domaines des variables $D(X_i)$ par le plus petit intervalle I_i contenant le domaine en entier ; nous avons en effet : $\alpha(G_I) \leq \alpha(G_X)$.

Algorithme gloutonne (MD) Soit le graphe d'intersection $G_X = (V, E)$. Soit $\Gamma(v), v \in V$ l'ensemble des voisins du noeud v et $\Gamma(S) = \bigcup_{v \in S} \Gamma(v)$. Posons $S = \emptyset$. Choisissons le noeud $v \in V \setminus (S \cup \Gamma(S))$ de degré minimum. Ajouter v à S et boucler ainsi de suite jusqu'à ce que $S \cup \Gamma(S) = V$. A la fin, S sera un ensemble de noeuds indépendants de G_X et $|S|$ sera une borne inférieure de $|S| \leq \alpha(G_X) = SP^* \leq SC^*$.

Théorème de Turán (TA) Nous pouvons aussi utiliser la borne que Turán a proposé [27] : $\left\lceil \frac{n^2}{2m+n} \right\rceil \leq \alpha(G_X) = SP^* \leq SC^*$.

D'après l'équation , nous déduisons que $SCLin^*$ est une meilleure borne que les trois autres. MD est au moins aussi forte que TA. OI est parfois aussi fort, parfois moins fort que MD et TA (voir [4] pour plus de détails).

Puisque AtMostNValue est équivalent à SC, nous pouvons utiliser TA, $SCLin^*$, MD ou OI pour calculer une borne valide de N et pour élaguer le domaine de T comme expliqué à la Section 2.1.

3 2SC, un autre propagateur pour la contrainte SC

Dans cette section nous présentons une autre relaxation de SC. Elle peut être utilisée par le propagateur général présenté à la Section 2.1. Le principal atout de cette relaxation est qu'elle peut être résolue de manière incrémentale, ce qui est très utile dans la Programmation par Contraintes où l'espace de recherche est parcouru le long d'un arbre de recherche, ce qui fait que le problème change très peu de structure entre deux calculs successifs.

3.1 Principes

Cette relaxation est basée sur le fait qu'un problème de recouvrement d'ensemble dont les sous-sensembles de la collection X ne contiennent que 2 éléments (que nous dénoterons 2SC) peut être résolu en temps polynomial. Cette relaxation a été travaillée dans un cadre de Programmation Entière dans [8] pour le problème de recouvrement et dans [17] pour le problème de partitionnement (cad avec des égalités au lieu des inégalités dans les contraintes (1)).

Pour résoudre $2SC$, nous pouvons construire le graphe $G = (V, E)$ où un noeud représente un élément à recouvrir et une arête représente un sous-ensemble S_i de la collection X. Un recouvrement par arête de poids minimum de G correspondrait à un recouvrement d'ensemble de poids minimum du problème original. Tout problème SC peut être décomposé en un problème 2SC de telle manière à ce que le graphe G sous-jacent soit biparti (voir Section 3.3). Obtenir un tel graphe est justifié par un critère de performance : trouver les meilleurs recouvrements par arête dans un graphe général est beaucoup plus onéreux en temps calcul (comme c'est le cas pour les problèmes d'appariemment).

Définissons $E(S_i)$, la décomposition de S_i en sous-ensemble de deux éléments : $E(S_i) = \{S_i^1, \ldots, S_i^k\}$. Nous avons que $\forall i : 1 \leq i \leq n$,

$$|S_i^j| = 2 \qquad \forall S_i^j \in E(S_i) \tag{6}$$

$$S_i = \bigcup_{S_i^j \in E(S_i)} S_i^j \tag{7}$$

Si nous définissons les poids c_i^j sur ces sous-ensembles S_i^j de cardinalité 2 tels que $c_i = \sum_{S_i^j \in E(S_i)} c_i^j, (1 \leq i \leq n)$ où c_i est le coût associé au sous-ensemble S_i, alors 2SC sera une relaxation de SC parce que tout recouvrement de SC sera un recouvrement de 2SC de coût exactement égal. Clairement, le coût d'une solution de 2SC donnera une borne inférieure du coût minimum du problème original.

Si nous dénotons la valeur optimale du problème 2SC par $2SC^*$, on peut montrer que

$$2SC^* \leq SCLin^* \leq SC^* \tag{8}$$

Cette borne peut donc être utilisée pour la contrainte SC. Nous montrons maintenant comment calculer cette borne de manière efficace.

3.2 Calcul Incrémental de $2SC^*$

Principes La relaxation présentée dans la section précédente ne serait pas très utile si nous ne pouvions pas la résoudre très efficacement. L'objectif de cette section est de présenter un nouvel algorithme incrémental pour le calcul de SC^*.

Le problème 2SC avec des poids introduits dans la section précédente est fort similaire au problème d'appariemment de poids maximum. Nous présentons dès lors une raisonnement similaire à celui de la méthode hongroise [16] pour résoudre 2SC. Comme dans la section précédente, nous faisons l'hypothèse que le graphe G contruit à partir du problème 2SC est biparti.

Premièrement nous donnons six conditions nécessaires et suffisantes que tout recouvrement R de poids minimum du graphe biparti G doit respecter. Ensuite nous décrivons un algorithme qui maintient les cinq premières conditions comme invariant et boucle jusqu'à ce que la sixième soit respectée. En troisième lieu, nous expliquons comment nous pouvons mettre la structure de donnée de l'algorithme à jour pour des petits changements (insertion, suppression d'arêtes, changements de poids). Pour finir, nous énumérons quelques petites améliorations qui sont apportées à l'algorithme de base.

Base théorique L'objectif du problème 2SC est de trouver un recouvrement de poids minimum R du graphe biparti $G = (V_1, V_2, E)$. Ce recouvrement par arête peut être contraint : une arête peut devoir faire partie de la solution ou peut ne pas pouvoir en faire partie. Une arête $(i, j) \in SURE$ si et seulement si (i, j) doit faire partie de la solution, $(i, j) \in REMOVED$ si et sulement si elle ne peut pas faire partie de la solution. Un noeud i est défini comme SURE s'il appartient à une arête qui est dans $SURE$. Un noeud de G est critique par rapport à R si exactement une arête de R est incidente à v. Autrement v est non-critique. Nous considérons les poids sur les arêtes constants.

Afin de pouvoir formaliser la suite, voici la formulation IP de 2SC :

$$2SC^* = \min \sum_{e \in F \cup SURE} c_i^j y_i^j \quad (9)$$

$$\sum_{(i,j) \in E} y_i^j \geq 1 \quad \forall vertex\ i : i \notin SURE \quad (10)$$

$$0 \leq y_i^j \leq 1 \quad \forall (i, j) \in F \quad (11)$$

où$F = E \setminus (SURE \cup REMOVED)$ est l'ensemble des arêtes dont on ne sait pas si elle font partie de la solution ou pas. La contrainte (11) est nécessaire puisque nous nous permettons des poids négatifs c_i^j. Il faut noter que dans la formulation précédente, nous n'avons pas contraint les variables y_i^j à être entière. En effet, nous pouvons montrer qu'il existe toujours une solution entière pour cette formulation continue (cela parce que la matrice qui définit les contraintes (10)-(11) est totalement unimodumlaire, voir [29]). Des résultats en théorie de l'optimisation (généralement appelés les conditions KKT, voir [14, 15]) donnent des conditions nécessaires et suffisantes pour qu'une solution d'un problème linéaire soit optimale. Soit les variables duales $\pi(i)$ des contraintes (10). Les conditions KKT énoncent qu'un ensemble d'arêtes (défini par les y_i^j) est un recouvrement optimal du graphe biparti $G = (V_1, V_2, E)$ si et seulement si

(i). $\pi(i) \geq 0 \quad \forall node\ i,\ \pi(i) = 0 \quad \forall\ i \in SURE$

(ii). $\bar{c}_i^j \leq 0$ si $(i, j) \in F$ et $y_i^j = 1$

(iii). $\bar{c}_i^j \geq 0$ si $(i, j) \in F$ et $y_i^j = 0$

(iv). $\sum_{(i,j) \in E} (y_i^j - 1) \geq 0 \quad \forall\ noeud\ i$

(v). $\pi(v) = 0$ si v est non-critique, $\forall v \in V_2$

(vi). $\pi(v) = 0$ si v est non-critique, $\forall v \in V_1$

où les coûts réduits $\bar{c}_i^j$ sont définis par $\bar{c}_i^j = c_i^j - \pi(i) - \pi(j)$. L'Algorithme 1 résoud le problème 2SC avec poids de manière optimale et est basé sur les conditions (i)-(vi). En commençant avec une solution qui respecte (i)-(v), l'algorithme boucle de telle manière à ce que strictement moins de noeuds i ne respectent pas (vi) à chaque itération. Il se termine lorsqu'il n'y a plus de tel noeud.

L'intuition derrière cet algorithme (comme pour les problèmes d'appariemment) est de selectionner un noeud v_0 qui ne respecte pas (vi) et de faire diminuer le nombre d'arêtes incidentes à ce noeud v_0 qui sont dans le recouvrement courant. Cela est fait en cherchant des chemins alternants. Un chemin alternant par rapport à un recouvrement R allant de v_0 à v_k est une séquence $p = [v_0, v_1, \ldots, v_k]$ de noeuds tels que les arêtes (v_i, v_{i+1}) sont prises en alternances entre R et $\setminus R$. Si $v_0 = v_k$ alors exactement une arête entre (v_0, v_1) et (v_{k-1}, v_k) appartient à R. Un chemin admissible L est défini comme un chemin alternant tel que

- Soit $(v_0, v_1) \in E \setminus R$ ou v_0 est non-critique
  ET
- Soit $(v_{k-1}, v_k) \in E \setminus R$ ou v_k est non-critique

Le résultat suivant, similaire à [3], énonce qu'un recouvrement augmenté par un chemin admissible reste un recouvrement.

Proposition 2 *Si L est un chemin de G admissible par rapport à R, alors l'ensemble d'arêtes $R' = R \oplus L = (R \setminus L) \cup (L \setminus R)$ est un recouvrement de G.* ∎

L'algorithme Le principe de cet algorithme est de trouver un noeud i qui ne respecte pas (vi) et augmenter le recouvrement courant avec un chemin admissible

qui part de i (avec la première arête qui appartient au receouvrement courant). Cela va faire strictement décroître le nombre d'arêtes incidentes à i qui sont dans le recouvrement. Après quelques itérations, le noeud i deviendra critique et repectera donc (vi). L'algorithme boucle jusqu'a ce qu'il n'existe plus de tel noeud i. L'algorithme explore les chemin admissibles de poids croissants (les plus courts chemins sont préférés).

La boucle intérieure (7)-(15) de Algorithme 1 modifie les valeurs duales π afin de trouver de nouveaux chemins admissibles du noeuds i. Nous observons que si $\delta = \beta$ ou $\delta = -\gamma$ alors l'arbre T croît strictement.Si $\delta = \alpha$ alors un chemin admissible est trouvé (i is non-critique et $e_k \in E \setminus R$). Puisque la boucle se termine si $\delta = \alpha$ (parce que $\exists w \in V_1^T \mid \pi(w) = 0$) et T ne peut pas croître indéfiniment, la boucle intérieure (7)-(15) n'iterera qu'un nombre fini de fois.

La boucle extérieure (5)-(18) iterera jusqu'à ce que le noeud i respectera la condition (vi). Après avoir modifié le recouvrement courant par $R \leftarrow R \oplus p$, il y'aura une arête incidente au noeud i en moins dans R car la première arête $(v_0, v_1) \in R$ pour tous les chemins admissibles $[v_0, v_1, \ldots, v_k]$ dans T (R est toujours un recouvrement d'après la proposition 2). Dès lors après un nombre fini d'itérations, le noeud i deviendra critique et la boucle extérieure (5)-(18) terminera.

L'avantage principal de cet algorithme est son caractère incrémental. Si nous retirons ou ajoutons quelques arêtes, nous avons juste besoin de modifier localement le recouvrement courant pour que (i)-(v) soient respectées. Ensuite, si nous réappliquons l'algorithme, (vi) sera respecté lui aussi. L'algorithme devrait seulement chercher pour un petit nombre de chemins admissibles de poids négatifs. Ceci est vraiment attrayant puisque cela nous permet de faire tourner le propagateur très vite à chaque appel lors du parcours de l'arbre de recherche. Cet avantage est démontré dans la Section 4 où nous observons le faible temps par appel du propagateur.

Nous pouvons encore améliorer l'algorithme d'après [20, p.423-424]. Nous pouvons remplacer la boucle intérieure (7)-(15), par un parcours d'arbre des plus courts chemins et calculer δ après celui-ci. L'algorithme utilisé lors des expérimentations contenait cette amélioration.

Dans la section suivante, nous montrons comment nous décomposons le problème de recouvrement original SC en problème 2SC, de telle manière à ce que le graphe sous-jacent soit biparti.

3.3 Construction du graphe biparti

Afin de calculer la borne présentée ci-dessus, construire le graphe biparti introduit à la section précédente. Plusieurs graphes G sont possibles pour une instance du problème SC donnée. Une heuristique est donnée dans [8]. Soit $H_j = \{e \in \mathcal{U} \mid e \in S_j \text{ et } q_j = \lfloor |H_j| \rfloor\}$. Pour relaxer SC en 2SC, ils créent une bipartition (R', R'') de l'ensemble $\mathcal{U} : \mathcal{U} = R' \cup R'',\ R' \cap R'' = \emptyset$. Pour un élément donné i, si $|\{j \mid i \in S_j\}| < q_j$, alors $i \in R'$. Autrement $i \in R''$. Ils construisent la partition en bouclant ainsi sur tous les éléments de $\mathcal{U}$ et en plaçant les éléments dans R' ou dans R'' de manière adéquate. Ils ajoutent deux éléments factices à R' et R''. Ensuite ils décomposent les sous-ensembles S_i en différents sous-ensembles S_i^j qui ne contiennent que deux éléments, un dans R' et un autre dans R'' et tel que (6)-(7) soient respectés $\forall i : 1 \leq i \leq n$. Dans le cas où G est biparti, 2SC peut être résolu en cherchant un recouvrement de poids minimum sur ce graphe biparti G.

4 Résultats expérimentaux

A partir des résultats expérimentaux repris dans [4], nous observons que TA et OI élaguent beaucoup moins les domaines des variables que les autres méthodes présentées (MD et $SCLin^*$), cela pour le même coût en calcul que MD. Nous avons donc décidé de comparer $2SC$ avec $SCLin^*$ et MD. Les résultats sont repris dans Table-1.

Nous avons généré des problèmes de recouvrement aléatoires en spécifiant quatre paramètres :

M **:** Le nombre d'éléments à couvrir

N **:** Le nombre de sous-ensembles de la collection X

S^- **:** La cardinalité minimale des sous-sensembles de la collection X

S^+ **:** La cardinalité maximale es sous-ensembles de la collection X

Nous avons alors créé de problèmes de recouvrement d'ensemble avec

- $\mathcal{U} = \{1, \ldots, M\}$
- $X = \{S_1, \ldots, S_N\}, S_i \subseteq \mathcal{U}, S^- \leq |S_i| \leq S^+ (1 \leq i \leq N)$ est une collection de sous-ensembles de $\mathcal{U}$ générés aléatoirement.
- $Cost(S_i) = 1 (1 \leq i \leq N)$

Des coûts unitaires ont été utilisés pour pouvoir appliquer MD. Nous avons créé des CSP avec seulement une contrainte $SC(N, T, \mathcal{U}, X, Cost)$ et essayé de trouver la valeur optimale de N (cad essayer de trouver SC^*) par branch and bound. Les trois relaxations sont utilisées : $2SC$, $SCLin^*$ et MD. Une heuristique de recherche très naïve est utilisée : elle branche sur $S_1, S_2, \ldots$, toujours dans le même ordre. Une branche est créée avec la contrainte supplémentaire $i \in T$. L'autre contient la contrainte $i \notin T$. Cette heuristique n'est pas efficace du tout pour résoudre des problèmes de recouvrement, mais cela permet d'être sûr que les

Algorithme 1 Algorithme pour 2SC avec poids

$computeSCRel^* \equiv computeMin2SC(2SC = (G, C, M))$

PRE: $G = (V_1, V_2, E)$ un graphe biparti, $C = \{c_i^j\}$: poids pour toutes les arêtes$(i, j) \in E$, $M \subseteq E$

POST: $R \cup M$ est un recouvrement de poids minimum EC^* de G tel que $M \subseteq EC^*$

Soit $R \leftarrow E$, définissons les valeurs initiales pour $\pi(v) \geq 0$ telles que (i)-(v) tiennent
Verifier que R couvre tous les noeuds de G. Sinon retourner IMPOSSIBLE
Soit $[v_1, \ldots, v_n]$ une énumération de tous les noeuds de V_1
for $i \mid \nexists j, (i, j) \in M$ **do**
  while v_i est non-critique et $\pi(v_i) > 0$ **do**
    $T = (V_1^T, V_2^T, E^T) \leftarrow computeT(G = (V, E), R, v_i)$
    while $\nexists w \in V_1^T \mid \pi(w) = 0$ et $\nexists w \in V_2^T, w$ non-critique **do**
      $\alpha \leftarrow \min\left\{\{+\infty\} \cup \{\pi(v) \mid v \in V_1^T\}\right\}$
      $\beta \leftarrow \min\left\{\{+\infty\} \cup \{\overline{c}_i^j \mid (i, j) \in E \setminus R, i \in V_2^T, j \notin V_1^T\}\right\}$
      $\gamma \leftarrow \max\left\{\{-\infty\} \cup \{\overline{c}_i^j \mid (i, j) \in R, i \in V_1^T, j \notin V_2^T\}\right\}$
      $\delta \leftarrow \min(\alpha, \beta, -\gamma)$
      $\pi(n_1) \leftarrow \pi(n_1) - \delta, \forall n_1 \in V_1^T$
      $\pi(n_2) \leftarrow \pi(n_2) + \delta, \forall n_2 \in V_2^T$
      $T = (V_1^T, V_2^T, E^T) \leftarrow computeT(G = (V, E), R, v_i)$
    end while
    Soit p un chemin admissible de v_i jusque w dans T
    $R \leftarrow R \oplus p$
  end while
end for
return $R \cup M$

$computeT(G = (V_1, V_2, E), R, v_0)$

PRE: $v_0 \in V_1 \cup V_2, R \subseteq E$

POST: Retourne un arbre construit avec des chemins alternés qui commencent à v_0 : $T = (V_A^T, V_B^T, E^T)$ tel que

- $V_A^T = \{v_k \in V_1 \mid \exists$ est un chemin alterné par rapport à R $[v_0, v_1, v_2, \ldots, v_{k-1}, v_k]$ avec $(v_0, v_1) \in R, \overline{c}_{v_{i-1}, v_i} = 0, \forall i = 1, \ldots, k$. $V_A^T = \{v_0\}$ si de tel chemin n'existe pas.
- $V_B^T = \{v_k \in V_2 \mid \exists$ est un chemin alterné par rapport à R, $[v_0, v_1, v_2, \ldots, v_{k-1}, v_k]$ avec $(v_0, v_1) \in R, \overline{c}_{v_{i-1}, v_i} = 0, \forall i = 1, \ldots, k$. $V_B^T = \emptyset$ si de tel chemin n'existe pas.
- $E^T = \{(v_i, v_{i+1}) \mid \exists$ est un chemin alterné par rapport à R, $[v_0, v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_{k-1}, v_k]$ avec $(v_0, v_1) \in R, \overline{c}_{v_{i-1}, v_i} = 0, \forall i = 1, \ldots, k$

trois propagateurs vont traverser l'arbre de recherche de la même manière. Aucune interaction entre le propagateur et l'heuristique de recherche n'est possible, ce qui pourrait mener à des résultats ambigus. Les instances sans solutions ne sont pas reprises dans Table-1. Les trois relaxations ont été implémentées comme propagateur dans Gecode [9]. $SCLin^*$ utilise la librairie Open-Source lp_solve 5.5 [18] pour résoudre les problèmes d'optimisation linéaire. Cette librairie implémente l'algorithme du simplexe qui permet de recalculer les solutions optimales de manière incrémentalle le long de l'arbre de recherche. Puisque ces problèmes sont tous des problèmes de recouvrement purs (sans contraintes auxiliaires), toutes les trois méthodes obtenaient de moins bonnes performances lorsque la singleton consistance était activée. Nous l'avons donc desactivée pour pouvoir comparer les méthodes de manière plus juste (le temps passé à faire de la singleton consistance inutile n'influe pas). Cependant s'il y avait des contraintes auxiliaires, les propagateurs avec de petits temps par appel devraient obtenir de meilleurs resultats (MD and $2SC$).

Aucune des méthodes ne dépassent uniformément les deux autres. Quelques observations sont toutefois très intéressantes. MD est généralement le meilleur quand la valeur $\frac{M}{N}$ est petite. Cela peut être expliqué par le fait que de tels problèmes ont beaucoup de solutions. Dès lors, il y a moins de filtrage des domaines possible et puisque MD est plus rapide pour calculer sa borne, il permet d'explorer plus de solutions par rapport aux deux autres méthodes qui passent plus de temps à calculer une borne qui ne permet pas de faire du filtrage supplémentaire.

D'un autre côté, $SCLin^*$ est meilleur quand la valeur est plus grande ($\frac{M}{N} \geq 1$). De tels problèmes ont très peu de solutions et l'excellente qualité de la borne $SCLin^*$ permet de contrebalancer l'effet négatif du temps nécessaire à son calcul.

$2SC$ semble être situé entre MD et $SCLin^*$. Quand MD est meilleur, $2SC$ atteint de meilleures performances que $SCLin^*$. De l'autre côté quand $SCLin^*$ est meilleur, $2SC$ est souvent meilleur que MD. Cela est du au fait que pour $2SC$, la relaxation $2SC$ est plus forte que $SCLin^*$. Mais MD est encore plus relaxé. Son coût de calcul est aussi situé entre les deux autres méthodes. $2SC$ semble être plus stable en fonction du type de problème à résoudre. Comme $2SC$ décompose les sous-ensembles en sous-ensembles de 2 éléments, il est meilleur quand les sous-ensembles S_i du problème original sont petits.

L'avantage de l'incrémentalité de l'algorithme pour $2SC$ est démontré par la Table-1. Nous observons que le temps par appel de ce propagateur est souvent endeça de $SCLin^*$, qui est aussi incrémental.

5 Recherche future

Deux lignes de recherche apparaissent. Premièrement comment utiliser des informations contenues dans les propagateurs pour orienter la recherche ? En utilisant les valeurs des variables duales avec SC ou $SCLin^*$, nous pourrions calculer quel sous-ensemble semble le plus utile d'ajouter au recouvrement que nous sommes en train de construire.

Deuxièmement, quelques tests de réduction existent dans la littérature pour filtrer, par avance, les sous-ensembles qui ne peuvent pas faire partie du recouvrement de coût optimal. De tels tests ne peuvent pas être utilisés tel quels puisque l'utilisateur n'est pas interessé par la solution optimale de la contrainte SC, mais la solution optimale qui respecte SC ainsi que beaucoup d'autres contraintes auxiliaires. Néanmoins lorsque nous calculons dans les propagateurs, de tels tests pourraient réduire la complexité de ce calcul. Ces tests seraient éxécutés lors de la propagation dans le premier noeud de l'arbre de recherche pour supprimer certains sous-ensembles de la représentation interne du problème. Ensuite, en fonction du chemin suivi le long de l'arbre de recherche, il faudrait rajouter des sous-ensembles à la représentation interne afin que la solution trouvée corresponde bien à l'optimum du problème complet.

6 Conclusion

Le problème de recouvrement d'ensemble est très important en optimisation combinatoire. Il est très utile pour modéliser des problèmes de la vie réelle, ainsi que des problèmes de la théorie des graphes. La conception d'une contrainte globale pour ce problème permet d'exprimer très facilement de tels problèmes dans un langage de Programmation par Contrainte, tout en fournissant des bornes de qualité pour résoudre ces problèmes efficacement.

D'après la structure du problème de recouvrement d'ensemble, le meilleur moyen pour filtrer des valeurs est de calculer des bornes numériques de qualité sur le paramètre N. Malheureusement, obtenir la meilleure borne possible est NP-difficile.

Dans ce papier, nous avons proposé un borne inférieure basée sur le calcul d'un problème de recouvrement par arête dans un graphe biparti. L'algorithme proposé est hautement incrémental, ce qui permet un calcul rapide de la borne après de petits changements dans le problème, ce qui le rend particulièrement bien adapté aux heuristiques de recherche utilisées dans la Programmation par Contrainte. La borne fournie est précise lorsque la taille des sous-ensembles est petite.

Quatre bornes inférieures ont été reprises dans [4],

M	**N**	S^-	S^+	$2SC$				SCL^*				MD			
				Sol	Time	failures	TC	Sol	Time	failures	TC	Sol	Time	failures	TC
10	500	2	6	223	> 30	38342	0,38	222	> 30	38760	0,37	**2**	**12,4**	263731	0,02
10	500	2	10	303	> 30	19418	0,73	235	> 30	35103	0,41	**1**	**8,46**	127222	0,03
10	200	2	6	**2**	4,09	19507	0,1	**2**	7,46	19507	0,18	**2**	**1,66**	45025	0,02
10	200	2	10	**1**	6,42	19703	0,16	**1**	8,11	19703	0,2	**1**	**1,18**	20875	0,03
20	200	2	4	**5**	**6,83**	55259	0,06	**5**	10,16	18946	0,26	9	> 30	771069	0,02
20	200	2	6	**4**	**4,5**	21706	0,1	**4**	10,4	19132	0,26	5	> 30	619822	0,02
20	200	2	10	**3**	**8,64**	22412	0,19	**3**	9,64	19317	0,24	**3**	> 30	398993	0,04
20	200	4	14	**2**	12,78	19507	0,32	**2**	12,28	19507	0,3	**2**	**8,97**	49310	0,09
20	200	8	10	**3**	15,27	25454	0,29	**3**	**9,57**	19407	0,24	**3**	> 30	266804	0,05
20	200	8	14	**2**	16,11	20092	0,39	**2**	**10,33**	19507	0,25	**2**	13,13	102288	0,06
50	500	2	4	141	> 30	64416	0,23	228	> 30	37053	0,39	**39**	> 30	551144	0,03
50	500	2	6	233	> 30	35513	0,41	236	> 30	34842	0,42	**28**	> 30	446423	0,03
50	500	2	10	314	> 30	17205	0,82	242	> 30	33228	0,44	**19**	> 30	318013	0,05
50	500	4	14	370	> 30	8405	1,62	260	> 30	28770	0,5	**12**	> 30	226824	0,07
50	500	8	10	365	> 30	9062	1,51	266	> 30	27381	0,53	**15**	> 30	212295	0,07
50	500	8	14	388	> 30	6259	2,12	266	> 30	27442	0,52	**11**	> 30	174868	0,08
10	50	2	6	**2**	**0,1**	1132	0,04	**2**	0,25	1132	0,1	**2**	**0,1**	3153	0,01
10	50	2	10	**1**	0,14	1178	0,05	**1**	0,26	1178	0,1	**1**	**0,09**	1446	0,03
100	500	2	4	**138**	> 30	67100	0,22	236	> 30	34938	0,41	199	> 30	511838	0,03
100	500	2	6	231	> 30	36210	0,4	244	> 30	32772	0,44	**104**	> 30	382419	0,04
100	500	2	10	312	> 30	17646	0,8	254	> 30	30312	0,48	**94**	> 30	269540	0,05
100	500	4	14	372	> 30	8227	1,65	269	> 30	26604	0,54	**45**	> 30	167095	0,09
100	500	8	10	364	> 30	9201	1,49	268	> 30	26897	0,53	**62**	> 30	176621	0,08
100	500	8	14	388	> 30	6257	2,12	277	> 30	24780	0,58	**42**	> 30	153117	0,1
50	200	2	4	24	> 30	291570	0,05	**15**	**13,29**	18405	0,35	36	> 30	749486	0,02
50	200	2	6	22	> 30	215045	0,07	**11**	**22,75**	20429	0,54	24	> 30	500720	0,03
50	200	2	10	11	> 30	126201	0,12	**7**	**12,53**	19119	0,32	15	> 30	393054	0,04
50	200	4	14	7	> 30	58297	0,25	**6**	**25,2**	21042	0,58	10	> 30	203992	0,07
50	200	8	10	10	> 30	62392	0,23	**7**	> 30	20723	0,7	12	> 30	243080	0,06
50	200	8	14	7	> 30	49796	0,29	**6**	> 30	20909	0,69	8	> 30	159690	0,09
400	1000	2	4	**791**	> 30	21935	0,64	862	> 30	9544	1,38	980	> 30	117592	0,1
400	1000	2	6	**836**	> 30	13515	1	876	> 30	7743	1,66	964	> 30	128167	0,09
400	1000	2	10	**885**	> 30	6651	1,88	**885**	> 30	6640	1,89	958	> 30	88089	0,14
400	1000	4	14	920	> 30	3179	3,41	**897**	> 30	5277	2,29	930	> 30	61146	0,22
400	1000	8	10	918	> 30	3330	3,3	**913**	> 30	3769	3	940	> 30	64338	0,2
400	1000	8	14	936	> 30	2029	4,69	**906**	> 30	4393	2,66	934	> 30	49467	0,27
400	500	2	10	385	> 30	15628	0,82	**318**	> 30	16981	0,84	462	> 30	121025	0,1
400	500	4	14	368	> 30	8755	1,56	**325**	> 30	15231	0,93	462	> 30	89955	0,14
400	500	8	10	364	> 30	9265	1,48	**336**	> 30	13385	1,05	463	> 30	73104	0,17
400	500	8	14	389	> 30	6164	2,15	**340**	> 30	12844	1,09	446	> 30	55121	0,23
50	50	2	6	15	> 30	321407	0,04	**14**	**0,28**	732	0,17	21	> 30	473335	0,02
50	50	2	10	10	> 30	218047	0,06	**9**	**0,37**	891	0,18	12	> 30	272193	0,04
50	50	4	14	**6**	> 30	110772	0,12	**6**	**0,41**	997	0,18	8	> 30	183474	0,06
50	50	8	10	**8**	> 30	154642	0,09	**8**	**0,59**	965	0,26	10	> 30	201884	0,06
50	50	8	14	7	> 30	129238	0,11	**6**	**0,61**	1044	0,25	7	> 30	143366	0,08
200	200	4	14	90	> 30	33609	0,42	**36**	> 30	15550	0,93	138	> 30	91178	0,15
200	200	8	14	73	> 30	30358	0,47	**33**	> 30	15642	0,93	84	> 30	89203	0,16
100	50	4	14	25	> 30	94787	0,11	**17**	**0,41**	686	0,27	23	> 30	130224	0,09
100	50	8	14	25	> 30	88691	0,12	**16**	**0,53**	757	0,31	29	> 30	95007	0,11
400	200	8	14	162	> 30	13554	0,75	**74**	> 30	10405	1,39	171	> 30	65742	0,18
50	20	4	14	**7**	0,08	471	0,06	**7**	**0,04**	98	0,15	**7**	0,14	775	0,06
50	20	8	10	**9**	0,42	2956	0,05	**9**	**0,03**	86	0,13	**9**	1,42	9487	0,06
50	20	8	14	**7**	0,47	2567	0,07	**7**	**0,05**	106	0,18	**7**	2,25	9991	0,08

TAB. 1 – Résultats expérimentaux. M est le nombre d'éléments à couvrir (cad la cardinalité de $\mathcal{U}$). N est la taille de la collection X. S^+ (resp. S^-) est le maximum de la cardinalité des sous-ensembles S_i (resp. la cardinalité minimum de S_i). Sol est la meilleure solution trouvée pour N. $Time$ est le temps total de recherche (recherche + propagation). Une limite de 30 secondes est utilisée (La plupart des problèmes sont trop compliqués que pouvoir être résolus avec l'heuristique de recherche naïve que nous avons utilisé). $failures$ est le nombre de fois que l'algorithm à échoué (retour en arrière dans l'arbre de recherche). TC est le temps par appel du propagateur en millisecondes.

et deux d'entre elles ($SCLin^*$ and MD) ont été identifiées comme meilleures que les deux autres. Nous avons comparé notre approche avec ces dernières. L'approche gloutonne atteint de bonnes performances lorsque le nombre de sous-ensembles disponibles est très grand comparé au nombre d'éléments à couvrir. La relaxation linéaire est la meilleure quand ce nombre est petit. $2SC$ est situé entre les deux autres, quand le nombre de sous-ensembles dans la collection X est le double du nombre d'éléments à couvrir. Plus important, $2SC$ atteint de meilleurs résultats que MD lorsque $SCLin^*$ est meilleur, et $2SC$ est meilleur que $SCLin^*$ lorsque MD est meilleur. $2SC$ est un bon choix lorsqu'on doit résoudre des problèmes avec des sous-ensembles de faible cardinalité. Avec des sous-ensembles plus grands, on peut choisir dynamiquement entre MD et $SCLin^*$ en fonction de la valeur de $\frac{M}{N}$.

Remerciements

Les auteurs remercient Jean-Charles Régin pour leur avoir suggéré cette contrainte globale. Merci aux reviewers pour leurs diverses suggestions. Une partie de cette recherche est supportée par la Région Wallonne, projet TransMaze (516207).

Références

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3) :501–555, 1998.

[2] Nicolas Beldiceanu, Mats Carlsson, and Sven Thiel. Cost-filtering algorithms for the two sides of the sum of weights of distinct values constraint. Technical Report 14, Swedish Institute of Computer Science, 2002. http ://www.sics.se/libindex.html#Technical.

[3] Claude Berge. Two theorems in graph theory. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 43-9, pages 842–844, 1957.

[4] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Filtering Algorithms for the NVALUE Constraint. In Roman Barták and Michela Milano, editors, *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-05)*, volume 3524 of *Lecture Notes in Computer Science*, pages 79–93, Prague, Czech Republic, May 2005. Springer-Verlag.

[5] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem, 1998.

[6] Alberto Caprara, Matteo Fischetti, and Paolo Toth. A heuristic method for the set covering problem. *Oper. Res.*, 47(5) :730–743, 1999.

[7] S. Ceria, P. Nobili, and A. Sassano. Set covering problem. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, chapter 23. John Wiley, 1997.

[8] Elia El-Darzi and Gautam Mitra. Solution of set-covering and set-partitioning problems using assignment relaxations. *The Journal of the Operational Research Society*, 43(5) :483–493, may 1992.

[9] Generic constraint development environment. http ://www.gecode.org/.

[10] Carmen Gervet. Interval propagation to reason about sets : Definition and implementation of a practical language. *Constraints*, 1(3) :191–244, march 1997.

[11] Karla L. Hoffman and Manfred Padberg. Solving airline crew scheduling problems by branch-and-cut. *Manage. Sci.*, 39(6) :657–682, 1993.

[12] Current J and O'Kelly M. Locating emergency warning sirens. *Decision Sciences*, 23(1) :221–234, 1992.

[13] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[14] W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois, 1939.

[15] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of 2nd Berkeley Symposium*, pages 481–492. Berkeley : University of California Press, 1950.

[16] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2 :83–97, 1955.

[17] Nemhauser George L. and Glenn M. Weber. Optimal set partitioning, matchings and lagrangian duality. *Naval Research Logistics Quarterly*, 26 :553–563, 1979.

[18] lp_solve library.

[19] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5) :960–981, 1994.

[20] Kurt Mehlhorn and Stefan Näher. *LEDA : a platform for combinatorial and geometric computing*. Cambridge University Press, New York, USA, 1999.

[21] François Pachet and Pierre Roy. Automatic generation of music programs. In *CP '99 : Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 331–345, London, UK, 1999. Springer-Verlag.

[22] John F. Pierce. Application of combinatorial programming to a class of all-zero-one integer programming problems. *Management Science*, 15(3) :191–209, nov 1968.

[23] J F Puget. Finite set intervals. In *Proceedings Workshop on Set Constraints, held in Conjunction with CP'96*, 1996.

[24] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI '94 : Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[25] Meinolf Sellmann, Kyriakos Zervoudakis, Panagiotis Stamatopoulos, and Torsten Fahle. Crew assignment via constraint programming : Integrating column generation and heuristic tree search. *Annals of Operations Research*, 115 :207–225, September 2002.

[26] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19(19) :1363–1373, 1971.

[27] P. Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48 :436–452, 1941.

[28] W. J. van Hoeve. The alldifferent constraint : A survey, 2001.

[29] L.A. Wolsey. *Integer Programming*. Wiley, New York, 1998.