HAL
open science

Un algorithme de programmation par contraintes pour la recherche d'allocations leximin-optimales

Sylvain Bouveret, Michel Lemaître

To cite this version:

Sylvain Bouveret, Michel Lemaître. Un algorithme de programmation par contraintes pour la recherche d'allocations leximin-optimales. Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC06), 2006, Nîmes - Ecole des Mines d'Alès / France. inria-00085776

HAL Id: inria-00085776

https://inria.hal.science/inria-00085776

Submitted on 14 Jul 2006

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.

Un algorithme de programmation par contraintes pour la recherche d'allocations leximin-optimales

Sylvain Bouveret **Michel Lemaître**

Office National d'Études et de Recherche Aérospatiales, Centre de Toulouse.
2, avenue Edouard Belin, B.P 4025, 31055 TOULOUSE CEDEX 4
sylvain.bouveret@onera.fr michel.lemaitre@onera.fr

Résumé

Dans le cadre de la programmation par contraintes, nous proposons un algorithme résolvant le problème suivant : allouer d'une manière équitable et efficace un ensemble fini d'objets à des agents ayant chacun leurs utilités propres, sous des contraintes d'admissibilité. L'algorithme calcule une allocation maximisant l'ordre leximin sur les profils d'utilités des agents.

Nous décrivons de plus le domaine d'application qui a motivé ces travaux : le partage de ressources satellitaires. Nous en extrayons un problème simple et précis d'allocation équitable, qui nous sert de base, grâce à un générateur de jeux de tests, pour l'évaluation de l'algorithme proposé. Deux implantations de l'algorithme sont comparées, l'une en programmation par contrainte «pure», avec Choco [14], l'autre en programmation linéaire mixte avec Cplex [12].

Abstract

Using the constraint programming framework, we propose an algorithm for solving the following problem : fairly and efficiently allocating a finite set of objects to a set of agents, each one having their own utilities, under admissibility constraints. Our algorithm computes an allocation maximizing the leximin order on the utility profiles of the agents.

Moreover, we describe the application domain that motivated this work : sharing of satellite resources. We extract from this real-world application a simple and precise fair allocation problem that allows for testing and evaluating our algorithms, using a benchmark generator. Two implementations of the algorithm are compared, the first one using the constraint programming tool Choco [14], and the second one using the integer linear programming tool Cplex [12].

1 Introduction

Allouer d'une manière équitable et efficace un ensemble limité de ressources à des agents ayant chacun leurs préférences propres est un problème général d'une portée considérable. De nombreux exemples de ce problème se retrouvent couramment, parmi lesquels on peut citer la construction d'emplois du temps, le partage de réseaux de communication, la gestion de ressources aéroportuaires impliquant plusieurs compagnies, le partage de l'espace aérien entre différents usagers, le partage de ressources satellitaires.

Dans cet article, nous abordons ce problème avec quatre hypothèses restrictives, mais qui laissent encore un champ d'application très large :
1) les ressources sont *discrètes, finies*, et se ramènent à des objets distincts, indivisibles et en nombre fini ;
2) les préférences des agents sur les allocations admissibles sont exprimées numériquement ;
3) on recherche des allocations *équitables* et *efficaces* – le sens de ces mots sera précisé et discuté plus loin ;
4) la recherche d'une allocation satisfaisante est réalisée de manière *centralisée* par un «arbitre» supposé juste et impartial et obéissant à des principes admis par tous les agents. Autrement dit, on ne s'intéresse pas ici à des procédures d'allocation ou de négociation distribuées entre agents.

Ce problème à caractère économique marqué touche à plusieurs champs de recherche actifs : la Recherche Opérationnelle (RO), l'Intelligence Artificielle (IA), la Microéconomie, la théorie du Choix Social. Notre contribution puise dans ces différents domaines. Des deux derniers nous empruntons l'idée d'*utilité* pour traduire des préférences numériques, et la comparaison

par l'*ordre leximin* pour traduire l'exigence d'équité et d'efficacité. La RO et l'IA nous fournissent le cadre de la programmation par contraintes, cadre dans lequel nous proposons un algorithme simple, centralisé, pour la recherche d'allocations leximin-optimales.

Préférences numériques et utilités Soit un ensemble d'alternatives admissibles $\mathcal{S}$ fini, dans lequel un arbitre doit choisir une alternative engageant n agents, chacun ayant ses préférences propres. Le modèle le plus classique de cette situation est celui du *welfarism* (voir par exemple [13, 18]). Selon ce modèle, que nous adoptons ici, les éléments de décision de l'arbitre sont entièrement contenus dans la donnée, pour chaque agent et pour chaque alternative, de son niveau de «bien-être». Ce niveau est mesuré, dans la version cardinale du modèle, par un index numérique mesurant l'*utilité individuelle* $u_i(s)$ de l'agent i pour l'alternative s. On suppose que les utilités individuelles sont comparables entre agents (elles sont données sur une échelle commune des utilités). À chaque alternative s correspond donc un *profil d'utilité* $\langle u_1(s), \ldots, u_n(s)\rangle$ et la comparaison entre deux alternatives s'effectue sur la seule base des deux profils associés.

Une façon commode de comparer les profils d'utilités individuelles est d'agréger chacun en un index d'*utilité collective* représentant le bien-être collectif de la société d'agents. Ainsi, à chaque alternative $s \in \mathcal{S}$ va correspondre une utilité collective $uc(s) = g(u_1(s), \ldots, u_n(s))$, où g est une fonction d'agrégation bien choisie. Une décision optimale est l'une de celles qui maximise cette utilité collective.

Équité et efficacité avec l'ordre leximin La difficulté de notre problème d'allocation équitable réside dans le fait qu'il faut concilier les intérêts contradictoires des agents. Il n'existe pas en général d'allocation qui satisfasse pleinement tous les agents à la fois. On recherche donc, à travers la fonction d'agrégation g, des compromis équitables (la répartition doit être «juste») et efficaces (les ressources doivent être pleinement utilisées), cette dernière notion étant classiquement traduite par la notion de Pareto-optimalité [1].

Le problème du choix de la fonction d'agrégation g dépasse largement le cadre de cet article. On se contentera de citer les deux fonctions les plus couramment proposées, et qui correspondent à deux visions extrêmes du bien-être collectif [2] : la fonction somme et la fonction minimum, correspondant respectivement aux notions d'*utilitarisme classique* et d'*égalitarisme*. La première n'est pas très pertinente dans notre contexte car l'utilité collective résultante ne dépend pas de la répartition des utilités individuelles. Par contre, la fonction minimum, quoiqu'un peu extrémiste, est particulièrement adaptée aux problèmes qui nous intéressent ici pour lesquels l'équité joue un grand rôle, car les décisions optimales associées sont celles qui maximisent la satisfaction du moins heureux des agents. Cependant, le problème de cette fonction, bien connu dans la communauté des CSP flous et couramment appelé «effet de noyade» [4], est qu'elle laisse un très grand nombre d'alternatives, pourtant très différentes, indistinguables les unes des autres. Ainsi par exemple, les profils d'utilité $\langle 0, \ldots, 0\rangle$ et $\langle 1000, \ldots, 1000, 0\rangle$ produiront la même utilité collective 0. Autrement dit, la maximisation de la fonction min peut produire des décisions non efficaces (non Pareto-optimales).

Deux raffinements de l'ordre induit par la fonction min et n'ayant pas cet inconvénient sont classiquement proposés dans le domaine des CSP flous pour pallier cet effet. Il s'agit des ordres discrimin et leximin [7]. Le discrimin a plusieurs inconvénients : tout d'abord il ne s'agit pas d'un préordre total, et il laisse de nombreuses incomparabilités entre profils ; en outre, en vertu du principe d'*anonymat* unanimement admis en théorie du choix social, la qualité d'une décision collective est insensible à la permutation des utilités des agents, ce qui n'est pas tout-à-fait le cas pour le discrimin (les deux qualités sont incomparables). le discrimin n'est donc pas pertinent pour le classement d'alternatives. Le leximin, que nous proposons d'utiliser ici, est employé classiquement en choix social [17]. Nous l'introduisons informellement, avant de le définir précisément dans la section 2. La comparaison de deux profils d'utilités selon l'ordre leximin ne s'opère pas à travers une fonction d'agrégation g, mais directement sur les profils. On recherche d'abord les deux valeurs minimales des deux profils. Si elles sont différentes, la plus grande des deux l'emporte. Sinon, on «élimine» ces minimaux de chacun des deux profils et on poursuit itérativement. Par exemple, soit à comparer les profils $\langle 4, 2, 3, 2\rangle$ et $\langle 2, 7, 2, 2\rangle$. Les minimaux sont les mêmes (2) donc on les élimine ($\rightarrow \langle 4, 3, 2\rangle$ et $\langle 7, 2, 2\rangle$). Les minimaux sont encore égaux (2) ; on les élimine ($\rightarrow \langle 4, 3\rangle$ et $\langle 7, 2\rangle$). Les nouveaux minimaux sont différents (3 et 2) : le premier profil est donc leximin-supérieur au second. Une façon équivalente d'exprimer l'ordre leximin est celle-ci : chaque profil est trié en ordre non décroissant puis on compare les profils ainsi triés selon l'ordre lexicographique (d'où le nom leximin).

Cet article est organisé ainsi : la section 2 définit formellement notre problème dans un cadre CSP. La section 3 décrit la principale contribution de cet

1. Une décision est Pareto-optimale si et seulement si on ne peut augmenter strictement la satisfaction d'un agent qu'en diminuant strictement la satisfaction d'au moins un autre agent.

2. Des compromis sont possibles entre ces deux extrêmes. Voir par exemple [18, page 68] (sommes de puissances) ou [22] (*Ordered Weighted Averaging aggregators*).

article : un algorithme de calcul d'une alternative leximin-optimale dans un cadre de programmation par contraintes, avec sa preuve. La section 4 est consacrée à l'application qui a motivé ces travaux : le partage de ressources satellitaires. Nous en extrayons un problème simplifié qui nous permet de tester deux implantations de l'algorithme en programmation par contraintes (avec Choco [14]), et en programmation linéaire (avec Cplex [12]). La section 5 présente des travaux voisins, avant les conclusions et perspectives, section 6.

2 Cadre formel

Le cadre de la programmation par contraintes est très utilisé dans la résolution de problèmes combinatoires aussi divers que les problèmes d'emploi du temps, de planification, d'allocation de fréquences . . . . Ce paradigme est fondé sur la notion de réseau de contraintes. Un réseau de contraintes est formé d'un ensemble de variables $\mathcal{X} = \{x_1, \ldots, x_p\}$, d'un ensemble de domaines $\mathcal{D} = \{d_{x_1}, \ldots, d_{x_p}\}$, où d_{x_i} est un ensemble fini de valeurs possibles pour x_i (nous supposons que $d_{x_i} \subset \mathbb{N}$, et notons $\underline{x_i} = \min(d_{x_i})$ et $\overline{x_i} = \max(d_{x_i})$), et d'un ensemble de contraintes $\mathcal{C}$. Chaque contrainte $C \in \mathcal{C}$ spécifie un ensemble de tuples autorisés $R(C)$ sur un ensemble de variables $X(C)$.

Une instanciation v d'un ensemble S de variables est une application qui à toute variable $x \in S$ associe une valeur $v(x)$ de son domaine d_x. Si $S = \mathcal{X}$, cette instanciation est complète, sinon, elle est partielle. Si $S' \subsetneq S$, la projection d'une instanciation de S sur S' est la restriction de cette instanciation à S' et est notée $v_{\downarrow S'}$. Une instanciation est cohérente si et seulement si elle ne viole aucune contrainte. Étant donné un réseau de contraintes, le problème d'existence d'une instanciation complète cohérente à ce réseau de contraintes est appelé Problème de Satisfaction de Contraintes (CSP) [16] et est NP-complet. Une telle instanciation, si elle existe, est une solution du CSP.

Il existe une déclinaison du CSP en problème d'optimisation (issue de l'extension max-CSP des problèmes de satisfaction de contraintes), dans laquelle une variable o joue le rôle de *variable objectif*. Une solution d'une instance de ce problème d'optimisation est une instanciation complète cohérente $\widehat{v}$ du réseau de contraintes telle que $\widehat{v}(o) = \max\{v'(o) | v' \text{ instanciation complète cohérente}\}$.

Soit $\overrightarrow{x} = \langle x_1, \ldots, x_n \rangle$ un vecteur d'entiers ; nous notons $\overrightarrow{x}^{\uparrow} = \langle x_1^{\uparrow}, \ldots, x_n^{\uparrow} \rangle$ la version ordonnée dans l'ordre non-décroissant de ce vecteur. Nous définissons l'ordre leximin sur les vecteurs d'entiers :

Définition 1 (Ordre leximin) *Soient $\overrightarrow{x}$ et $\overrightarrow{y}$ deux vecteurs de $\mathbb{N}^n$. $\overrightarrow{x}$ et $\overrightarrow{y}$ seront dits* leximin-*indifférents (noté $\overrightarrow{x} \sim_{leximin} \overrightarrow{y}$) si et seulement si $\overrightarrow{x}^{\uparrow} = \overrightarrow{y}^{\uparrow}$. Le vecteur $\overrightarrow{y}$ est* leximin-*préféré à $\overrightarrow{x}$ (noté $\overrightarrow{x} \prec_{leximin} \overrightarrow{y}$) si et seulement si $\exists i \in [\![0, n-1]\!]$ tel que $\forall j \in [\![1, i]\!]$, $x_j^{\uparrow} = y_j^{\uparrow}$ et $x_{i+1}^{\uparrow} < y_{i+1}^{\uparrow}$. On notera $\overrightarrow{x} \preceq_{leximin} \overrightarrow{y}$ pour $\overrightarrow{x} \prec_{leximin} \overrightarrow{y}$ ou $\overrightarrow{x} \sim_{leximin} \overrightarrow{y}$.*

La relation binaire $\preceq_{leximin}$ est un préordre total.

Dans un problème de décision collective, une solution leximin-optimale est une alternative dont le profil d'utilités associé est maximal pour l'ordre $\preceq_{leximin}$. Une telle solution a l'avantage, outre d'être min-optimale, d'être aussi Pareto-efficace.

La déclinaison des CSP présentée ci-avant permet d'encoder un certain nombre de problèmes d'optimisation combinatoire issus du choix social. Par exemple, si l'on veut modéliser le problème de maximisation de l'utilité collective égalitariste (fonction d'agrégation $g = \min$) dans un problème à n agents, on peut introduire n variables $\langle u_1, \ldots, u_n \rangle$ correspondant aux utilités de chaque agent, et une variable objectif uc liée aux autres variables par les contraintes $\{C_1, \ldots, C_n\}$, avec $C_i = (uc \leq u_i)$.

En revanche, la modélisation du problème de calcul d'une décision leximin-optimale dans le formalisme CSP n'est pas si évidente, et elle nécessite une légère transcription de la variante optimisation du CSP. Nous nous intéresserons au problème modélisé comme suit.

[Leximin-Optimal]

Entrées : un réseau de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; un vecteur de variables $\overrightarrow{u} = \langle u_1, \ldots, u_n \rangle$ ($\forall i,\ u_i \in \mathcal{X}$), appelé *vecteur objectif*.

Sortie : «Incohérent» s'il n'existe pas d'instanciation complète cohérente. Sinon, une instanciation $\widehat{v_s}$ complète cohérente telle que $\forall v$ instanciation complète cohérente, $v(\overrightarrow{u}) \preceq_{leximin} \widehat{v_s}(\overrightarrow{u})$.

Nous proposons un algorithme de résolution de ce problème, fondé sur une méta-contrainte de cardinalité.

3 Algorithme proposé

Le principe de l'algorithme 1 est de calculer itérativement chaque composante du vecteur correspondant aux valeurs ordonnées de l'instanciation leximin-optimale de $\overrightarrow{u}$. Pour cela, on introduit aux lignes 3 et 4 un vecteur de variables d'optimisation, le rôle de chaque variable y_i étant de calculer la valeur de l'indice i du leximin-optimal (on notera $m = \min\{\underline{u_i} | 1 \leq i \leq n\}$ et $M = \max\{\overline{u_i} | 1 \leq i \leq n\}$). À chaque itération i de la boucle 6..10, on ajoute une contrainte de cardinalité correspondant à la composante en cours de calcul (ligne 7), et on calcule (ligne 8) la valeur maximale de

la variable y_i telle que le réseau de contraintes courant (qui correspond au réseau initial additionné des variables y_k et des contraintes de cardinalité des itérations précédentes) ait une solution. La variable y_i est fixée à cette valeur optimale (ligne 9) pour toutes les itérations suivantes. La ligne 10 restreint le domaine de la prochaine variable y_{i+1} de manière sûre ; cependant, les tests montrent qu'elle n'influe pas de manière significative sur les temps de calcul, certainement car la propagation de contraintes est capable de filtrer très rapidement cette partie du domaine de y_{i+1}.

Algorithme 1 : Algorithme de recherche d'une solution leximin-optimale d'un réseau de contraintes.

entrées : un réseau de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; un vecteur $\langle u_1, \ldots, u_n \rangle$ de variables de $\mathcal{X}$
sortie : Une solution du problème [LEXIMIN-OPTIMAL] ou «Incohérent»

1 **si solve**$(\mathcal{X}, \mathcal{D}, \mathcal{C})$ = *«Incohérent»* **alors**
2 ⌊ **retourner** *«Incohérent»*
3 $\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$;
4 $\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;
5 $\mathcal{C}' \leftarrow \mathcal{C}$;
6 **pour** $i \leftarrow 1$ **à** n **faire**
7 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{$**AtLeast**$(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;
8 $\quad \widehat{v} \leftarrow$ **maximize**$(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;
9 $\quad d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;
10 $\quad d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$
11 **retourner** $\widehat{v}_{\downarrow \mathcal{X}}$

L'algorithme utilise la méta-contrainte de cardinalité **AtLeast** :

Définition 2 (Méta-contrainte AtLeast) *Soit* Γ *un ensemble de* p *contraintes, et* $k \in [\![1, p]\!]$ *un entier. Alors la méta-contrainte* **AtLeast**(Γ, k) *est la contrainte portant sur l'ensemble des variables sur lesquelles portent les contraintes de* Γ*, et autorisant uniquement les tuples de valeurs pour lesquels au moins* k *contraintes de* Γ *sont satisfaites.*

Cette méta-contrainte[3] est introduite sous le nom de *cardinality combinator* par exemple dans [11]. Son rôle dans l'algorithme 1 est de simuler une contrainte d'ordre leximin (l'ensemble des contraintes de cardinalité impose aux solutions suivantes d'être leximin-supérieures au vecteur leximin-optimal partiel).

Les fonctions **solve** et **maximize** (dont le détail est du ressort de la résolution de problèmes de satisfaction de contraintes) des lignes 1 et 8 renvoient respectivement une solution du réseau de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ (ou «Incohérent» si une telle solution n'existe pas), et une solution optimale du réseau de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec variable objectif y (ou «Incohérent» si une telle solution n'existe pas). Nous considérons – contrairement aux solveurs de contraintes usuels – que ces deux fonctions ne modifient pas les réseaux de contraintes.

Proposition 1 *Si les fonctions* ***maximize*** *et* ***solve*** *sont correctes et terminent, l'algorithme 1 termine et renvoie une solution au problème de calcul du* leximin-*optimal d'un CSP.*

Preuve : La preuve de la terminaison est immédiate si les fonctions **solve** et **leximin** terminent.

Si le réseau de contraintes initial n'a pas de solution, et si la fonction **solve** est correcte, alors l'algorithme renvoie «Incohérent». Nous supposons dans la suite de la preuve que nous ne sommes pas dans ce cas-là.

Dans la preuve, on notera $\widehat{v_i}$ l'instanciation retournée à l'itération i par la fonction **maximize**, et $\widehat{v_s}$ la solution au problème [LEXIMIN-OPTIMAL].

Squelette de la preuve : Pour montrer que l'algorithme est correct, il suffit de montrer que l'appel à **maximize** ne renvoie jamais «Incohérent», et que $\widehat{v_n}(\overrightarrow{u})^{\uparrow} = \widehat{v_s}(\overrightarrow{u})^{\uparrow}$. Pour ce faire, on se sert de l'hypothèse de récurrence suivante : $(H_i) = \left(\widehat{v_i} \text{ existe et } \forall j \leq i,\ \widehat{v_i}(\overrightarrow{u})^{\uparrow}_j = \widehat{v_i}(y_j) = \widehat{v_s}(\overrightarrow{u})^{\uparrow}_j\right)$.

À l'itération 1, la contrainte **AtLeast**$(\{u_1 \geq y_1, \ldots, u_n \geq y_1\}, n)$ est équivalent à la contrainte $y_1 \leq \min u_i$. La fonction **maximize** renvoie donc une extension $\widehat{v_1}$ d'une solution du réseau de contraintes initial telle que $\widehat{v_1}(y_1) = \max\{\min_i(v_1(u_i)) | v_1$ instanciation complète cohérente$\}$. Donc $\widehat{v_1}(y_1) = \widehat{v_1}(\overrightarrow{u})^{\uparrow}_1$. De plus, $\widehat{v_1}(y_1) \geq \widehat{v_s}(\overrightarrow{u})^{\uparrow}_1$ si la fonction **maximize** est correcte (il existe une extension de $\widehat{v_s}$ sur $(\mathcal{X}', \mathcal{D}', \mathcal{C}')$ cohérente). On ne peut avoir $\widehat{v_1}(y_1) > \widehat{v_s}(\overrightarrow{u})^{\uparrow}_1$, car cela voudrait dire que $\widehat{v_1}(\overrightarrow{u})^{\uparrow}_1 > \widehat{v_s}(\overrightarrow{u})^{\uparrow}_1$ et donc que $\widehat{v_1}$ serait une solution leximin-supérieure à $\widehat{v_s}$, ce qui n'est pas possible Ceci prouve (H_1).

Montrons que $(H_i) \Rightarrow (H_{i+1})$ $(i \in [\![1, n-1]\!])$.

Montrons que $\widehat{v_{i+1}}$ existe (c'est-à-dire qu'il existe au moins une solution au réseau de contraintes de l'itération $(i+1)$). L'instanciation $\widehat{v_s}$ est la projection sur $\mathcal{X}$ d'une solution du réseau de contraintes de l'itération $i+1$. En effet :

— par définition, $\widehat{v_s}$ satisfait toutes les contraintes du réseau initial ;

— $\forall j \leq i$, $\widehat{v_i}(y_j) = \widehat{v_s}(\overrightarrow{u})^{\uparrow}_j$ (d'après (H_i)), donc $\widehat{v_s}(\overrightarrow{u})^{\uparrow}_j$ et toutes ses composantes suivantes dans le vecteur ordonnées (soit $n - j + 1$ composantes de $\widehat{v_s}(\overrightarrow{u})$) sont supérieures ou égales à $\widehat{v_j}(y_j)$, ce qui satisfait la contrainte **AtLeast** de l'itération j ;

— $\widehat{v_s}(\overrightarrow{u})^{\uparrow}_{i+1} \geq \widehat{v_s}(\overrightarrow{u})^{\uparrow}_i$ par définition, donc il existe au moins une valeur cohérente pour y_{i+1} : $\widehat{v_i}(y_i)$.

Donc il existe au moins une solution au réseau de contraintes de l'itération $i+1$ (donc $\widehat{v_{i+1}}$ existe).

3. Le préfixe «méta» indique que cette contrainte prend en paramètre d'autres contraintes.

En outre, pour tout $j \leq i+1$, $\widehat{v_{i+1}}(\overrightarrow{u})^{\uparrow}_j \geq \widehat{v_{i+1}}(y_j)$ (sinon au moins une des contraintes **AtLeast** est violée). En remarquant qu'une allocation admissible pour $(\mathcal{X}', \mathcal{D}', \mathcal{C}')$ à l'itération $i+1$ l'est aussi à l'itération i (car entre deux itérations successives on ne fait qu'ajouter une contrainte et réduire le domaine d'une variable), on en déduit qu'on ne peut avoir $\widehat{v_{i+1}}(\overrightarrow{u})^{\uparrow}_j > \widehat{v_{i+1}}(y_j)$. En effet, dans ce cas, puisque $\widehat{v_{i+1}}(y_j) = \widehat{v_j}(y_j)$ (pour $j < i+1$, le domaine de y_j est un singleton), $\widehat{v_{i+1}}$ aurait été strictement meilleure que $\widehat{v_j}$ pour y_j à l'itération j, et si **maximize** est correcte, ce n'est pas possible. Donc $\forall j \leq i+1$, $\widehat{v_{i+1}}(\overrightarrow{u})^{\uparrow}_j = \widehat{v_{i+1}}(y_j)$, ce qui prouve la première égalité.

L'extension de $\widehat{v_s}$ qui affecte la valeur $\widehat{v_s}(\overrightarrow{u})^{\uparrow}_{i+1}$ à y_{i+1} est faisable à l'itération $i+1$ (elle satisfait, en plus des autres contraintes, la contrainte **AtLeast** de l'itération $i+1$). Donc $\widehat{v_{i+1}}(y_{i+1}) \geq \widehat{v_s}(\overrightarrow{u})^{\uparrow}_{i+1}$. Si l'on avait $\widehat{v_{i+1}}(y_{i+1}) > \widehat{v_s}(\overrightarrow{u})^{\uparrow}_{i+1}$, alors la projection de $\widehat{v_{i+1}}$ sur $\mathcal{X}$ serait une solution de ce réseau de contraintes tel que $\forall j < i+1$, $\widehat{v_{i+1}}(\overrightarrow{u})^{\uparrow}_j = \widehat{v_s}(\overrightarrow{u})^{\uparrow}_j$ et $\widehat{v_{i+1}}(\overrightarrow{u})^{\uparrow}_{i+1} > \widehat{v_s}(\overrightarrow{u})^{\uparrow}_{i+1}$, donc une solution leximin-supérieure à $\widehat{v_s}$, ce qui n'est pas possible. On a donc bien $\widehat{v_{i+1}}(y_{i+1}) = \widehat{v_s}(\overrightarrow{u})^{\uparrow}_{i+1}$, ce qui achève de prouver (H_{i+1}).

Par récurrence, on a donc : (1) $\widehat{v_n}$ est une solution du réseau de contraintes à l'itération n, donc *a fortiori*, sa projection sur $\mathcal{X}$ est une solution et (2) pour tout i, $\widehat{v_n}(\overrightarrow{u})^{\uparrow}_i = \widehat{v_s}(\overrightarrow{u})^{\uparrow}_i$, donc $\widehat{v_n}(\overrightarrow{u})$ et $\widehat{v_s}(\overrightarrow{u})$ sont leximin-indifférents. Donc l'instanciation renvoyée par l'algorithme est bien une solution du problème [LEXIMIN-OPTIMAL]. ■

Si la programmation par contraintes se prête particulièrement à l'implantation de cet algorithme, la méta-contrainte de cardinalité utilisée dans l'algorithme s'exprime aussi dans le domaine de la programmation linéaire [10, p.11] grâce à l'introduction de n variables 0–1 $\{\delta_1, \ldots, \delta_n\}$. La méta-contrainte **AtLeast**$(\{x_1 \geq y, \ldots, x_n \geq y\}, k)$ est équivalente à l'ensemble de contraintes linéaires $\{x_1 + \delta_1 \overline{y} \geq y, \ldots, x_n + \delta_n \overline{y} \geq y, \sum_{i=1}^{n} \delta_i \leq n-k\}$.

4 Application à un problème de partage de ressources satellitaires

Nous décrivons maintenant l'application qui a motivé ces travaux, et qui nous a servi pour expérimenter et évaluer l'algorithme proposé en situation réaliste.

4.1 Description de l'application

L'application concerne l'exploitation commune, par plusieurs agents (pays, organismes internationaux...), d'une constellation de satellites d'observation de la Terre. La mission de ce type de satellites consiste, comme l'illustre la figure 1, à acquérir des photographies de la Terre, en réponse à des demandes de photographies déposées par les agents. Ces agents déposent,

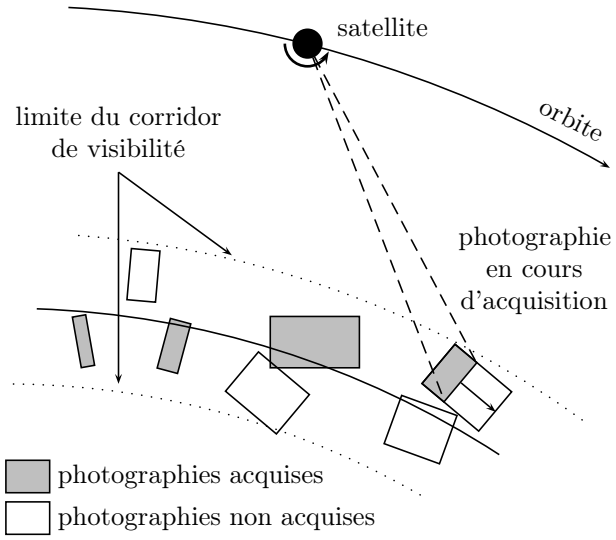


FIGURE 1 – Acquisition d'une photographie par un satellite d'observation de la Terre.

auprès d'un centre de planification commun, des demandes de photographie valables pour un jour donné. La planification globale des prises de vue de tous les satellites de la constellation est organisée par intervalles de temps successifs, généralement 1 jour. Le centre de planification détermine donc, parmi les demandes concernant un jour donné, l'ensemble des demandes qui seront satisfaites, c'est-à-dire l'ensemble des photographies qui seront acquises ce jour-là par la constellation. Cet ensemble de demandes satisfaites constitue une *allocation* journalière des demandes aux agents.

Les contraintes physiques d'exploitation et le nombre important de demandes concernant certaines zones génèrent des conflits entre demandes. Il est donc en général impossible de satisfaire simultanément toutes les demandes déposées pour un jour donné. Autrement dit, seul un sous-ensemble des demandes pourront être satisfaites. Toutes ces contraintes définissent l'ensemble des *allocations admissibles*.

Voici quelques ordres de grandeur concernant le problème réel. Les agents sont entre 3 et 6. Plusieurs centaines de demandes sont candidates chaque jour, parmi lesquelles 100 à 200 seront satisfaites.

Les demandes d'un agent sont d'importances inégales. Chaque agent traduit l'importance relative de ses demandes en associant à chacune un *poids*, qui est un nombre positif ou nul[4], et correspond implicitement à des préférences additives : étant donnés deux ensembles de demandes d'un agent dont la somme des poids est identique, l'agent concerné est indifférent devant l'obtention de l'un ou l'autre ensemble de demandes. L'*utilité individuelle* d'une allocation pour un

4. Un poids nul marque simplement le fait qu'un agent n'est pas intéressé par la demande.

agent est la somme des poids de ses demandes satisfaites par l'allocation. Un dispositif de *normalisation des utilités* – dont le détail ne concerne pas cet article – est utilisé afin de rendre comparables les utilités individuelles. Nous considérons ici implicitement des utilités (et des poids) normalisés.

Tous les agents n'ont pas contribué de manière égale au financement de la constellation ; le «droit de retour sur investissement» prévu pour chacun est donc différent. Il existe différentes manières de prendre en compte ces droits différents. Nous traduisons ici cette inégalité par des contraintes de consommation (s'ajoutant aux contraintes d'admissibilité) : chaque agent a droit à une consommation maximum de ressources par jour, ce maximum étant différent pour chaque agent.

Au-delà de la prise en compte de ces droits inégaux, l'allocation des demandes aux agents doit être équitable. Des solutions assez différentes à ce problème ont été proposées dans [15], puis dans [6]. Pour satisfaire l'exigence d'équité, l'un des protocoles de partage proposés consiste à choisir une allocation qui maximise l'ordre leximin sur les profils d'utilités individuelles.

4.2 Un problème d'allocation équitable

Nous avons tiré de cette application un problème simplifié d'allocation équitable d'objets à des agents. Il se présente sous la forme d'une extension du problème LEXIMIN-OPTIMAL décrit section 2. Les objets correspondent aux demandes de notre application. Les conflits entre demandes sont modélisés de manière approximée mais convenable sous la forme de «contraintes de volume généralisé», linéaires. Noter que dans ce problème (1) tous les objets ne seront pas nécessairement alloués, et (2) un même objet peut être alloué à plusieurs agents [5].

Voici la description formelle de ce problème d'allocation équitable. Tout d'abord les données :

— $\mathcal{A}$ est un ensemble d'*agents* ;

— $\mathcal{O}$ est un ensemble d'*objets* à allouer aux agents ;

— w_{io}, nombre positif ou nul, est le *poids* attribué à l'objet o par l'agent i ;

— r_o est la *ressource consommée* par l'objet o ;

— $rmax_i$ est la *consommation maximum de ressources* autorisée pour l'agent i ;

— $\mathcal{C}$ est l'ensemble de *contraintes de volume généralisé* ;

— v_{co} est le *volume* de l'objet o dans la contrainte de volume généralisé c ;

— $vmax_c$ est le *volume maximum* dans la contrainte de volume généralisé c.

On définit maintenant les variables suivantes :

— $x_{io} = 1$ si l'objet o est alloué à l'agent i, et 0 sinon, $o \in \mathcal{O}, i \in \mathcal{A}$. Les affectations possibles des variables x_{io} représentent l'ensemble des allocations possibles (parmi lesquelles se trouvent les admissibles) ;

— $u_i = \sum_{o \in \mathcal{O}} x_{io} \cdot w_{io}$ est l'utilité individuelle de l'agent i, $i \in \mathcal{A}$;

— $s_o = \max_{i \in \mathcal{A}} x_{io} = 1$ si l'objet o est alloué à un agent au moins, et 0 sinon.

Le problème est de trouver une allocation x maximisant l'ordre leximin sur les profils d'utilités $\langle u_i \rangle_{i \in \mathcal{A}}$, sous les contraintes d'admissibilité suivantes :

— $w_{io} = 0 \Rightarrow x_{io} = 0$ (les objets de poids nul ne sont pas alloués à un agent ayant attribué ce poids) [6] ;

— $\sum_{o \in \mathcal{O}} x_{io} \cdot r_o \leq rmax_i$, pour tout $i \in \mathcal{A}$ (contraintes de consommations de ressources) ;

— $\sum_{o \in \mathcal{O}} s_o \cdot v_{co} \leq vmax_c$, pour tout $c \in \mathcal{C}$ (contraintes de volume généralisé).

Complexité du problème Pour une instance du problème d'allocation équitable, nous notons $\overrightarrow{u(x)}$ le profil d'utilités résultant de l'allocation complète x (dans laquelle toutes les variables de x sont affectées). Considérons le problème de décision ainsi défini : *Étant donnée une instance du problème d'allocation équitable et un profil d'utilités $\overrightarrow{UC}$, existe-t-il une allocation x admissible (satisfaisant toutes les contraintes) telle que $\overrightarrow{UC} \preceq_{leximin} \overrightarrow{u(x)}$?*

Lorsqu'il n'y a qu'un seul agent et une seule contrainte (de consommation ou de volume), on reconnaît aisément le problème du sac à dos ([KNAPSACK], [9, page 65]), problème NP-complet. Notre problème de décision est donc bien sûr NP-complet.

On remarque qu'il s'agit d'une généralisation du problème de sac à dos multidimensionnel en variables bivalentes [21], mais avec un critère d'optimisation très particulier.

4.3 Un jeu de tests

Nous avons construit un générateur d'instances aléatoires du problème d'allocation équitable décrit en 4.2.

Ce générateur [7] est très facilement paramétrable, grâce à quatre groupes de paramètres :

— les paramètres généraux : nombre d'agents, nombres d'objets, graine du générateur aléatoire ;

— les paramètres de poids des objets ;

— les paramètres de contraintes de consommation ;

— les paramètres de contraintes de volume généralisé.

5. Ce qui est possible dans notre application.

6. Cette contrainte permet de cerner les allocations réellement significatives.

7. Écrit en Java, il est disponible en `http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`.

Les poids des objets sont générés aléatoirement. Deux types de distributions sont possibles : une distribution uniforme entre 0 et w_{max} et une répartition en différentes classes. La répartition en différentes classes approche les conditions de l'application réelle. Une telle répartition est paramétrée par f_c (par exemple $f_c = 10$), le facteur multiplicatif entre classes, et n_c (par exemple $n_c = 4$), le nombre de classes. Les poids appartenant à la classe $i \in [\![1, n_c]\!]$ sont tirés aléatoirement entre $\frac{1}{2}(f_c)^i$ et $\frac{3}{2}(f_c)^i$. De plus, on s'assure qu'il y a plus de demandes de classes faibles (moins importantes), que de demandes de classes fortes.

Comme indiqué précédemment, les contraintes de consommation permettent de simuler des droits d'accès à la ressource inégaux selon les agents. Dans notre générateur, les droits inégaux dépendent de deux paramètres : le droit de l'agent de plus faible droit r_{min}, et le facteur multiplicatif entre les droits f_d. L'agent i a un droit de $r_{min} \times (f_d)^{i-1}$.

Les paramètres concernant les contraintes de volume généralisé sont les suivants :

— l'arité des contraintes n_S ;

— le volume maximum autorisé pour chaque contrainte (fixe ou aléatoire) ;

— le volume de chaque objet (fixe ou aléatoire).

Le générateur permet aussi d'instancier ces paramètres en spécifiant la dureté des contraintes (ici le rapport du nombre d'objets interdits sur l'arité de la contrainte), celles-ci portant sur n_S objets consécutifs.

4.4 Résultats obtenus

L'algorithme proposé en section 3 a été testé sur les instances de notre générateur aléatoire grâce à deux implantations différentes : l'une avec la librairie de programmation par contraintes Choco [14] sous Java, et l'autre avec Cplex 10.0 [12] interfacé avec Java. Les tests sont fondés sur une instance moyenne ressemblant aux caractéristiques du problème réel, comportant 4 agents possédant des droits inégaux et 150 objets. Les contraintes de volume sont de dureté moyenne (autorisant 10 objets sur 20 consécutifs). Nous avons cherché à mettre en valeur l'influence de trois paramètres sur le temps de calcul : le nombre d'objets, la répartition des poids, et la dureté des contraintes de volume. Pour chaque valeur, l'algorithme est exécuté sur 20 instances choisies aléatoirement.

On constate d'abord que Cplex est bien meilleur que Choco sur toutes les instances (voir la figure 2). Pour expliquer cette différence de performance, on peut noter que Cplex est un outil spécialisé en PL et PLNE, utilisant des coupes spécifiques. La PPC est un outil beaucoup plus général. De plus, nous n'avons pas encore exploité, avec Choco, les possibilités de guidage dynamique de la recherche (choix de variables et valeurs de branchement). Toutefois, si les poids sont trop dispersés (répartition non-uniforme, facteurs 10 ou 100), Cplex commet de deux types d'erreur :

— la solution trouvée est légèrement différente de la solution optimale sur une composante du vecteur leximin, ce qui rend les composantes suivantes potentiellement très différentes ;

— la fonction **maximize** de l'algorithme 1 renvoie «Incohérent», alors qu'elle ne devrait pas.

Le nombre d'erreurs du second type sur les 20 instances résolues est reporté sur la figure 2(c) (le nombre d'erreurs du premier type est à peu près équivalent). Ces erreurs sont certainement dues à des erreurs d'arrondis dans l'algorithme utilisé par Cplex. Elles pourraient peut-être être corrigées par un réglage plus fin des paramètres de Cplex.

On remarque de plus, concernant la répartition des poids des objets, que les instances à répartition uniforme sont plus difficiles à résoudre que les autres. Ceci est particulièrement flagrant avec Choco, comme on peut le voir sur la figure 2(b). En outre, le facteur multiplicatif de classes f_c n'a quasiment pas d'influence sur la résolution. Pour expliquer ce fait, on peut avancer ceci : bien que les classes avec $f_c = 10$ ne soient pas complètement «étanches» [8], elles le sont suffisamment pour que les instances générées et leurs solutions soient proches de celles avec f_c =100.

Concernant l'influence de la dureté des contraintes, on constate classiquement l'existence de «pics» de complexité, d'une part avec Cplex, seulement pour des répartitions uniformes des poids, et avec Choco pour les deux répartitions étudiées. Les pics pour les deux outils ne sont pas exactement au même endroit.

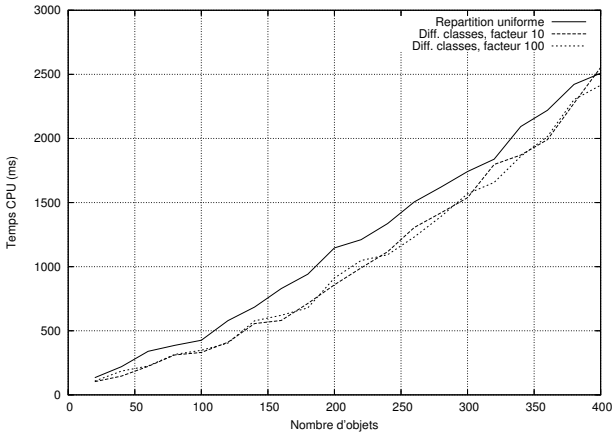
5 Approches alternatives et travaux voisins

5.1 Approches alternatives

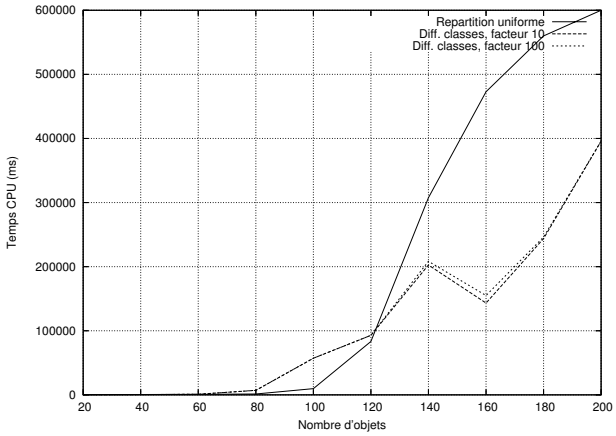
D'autres approches du calcul d'une solution leximin-optimale peuvent être proposées. La solution la plus immédiate est une comparaison directe de toutes les alternatives admissibles, comme on peut le voir dans [5, p.162]. Cependant, cette solution ne peut pas être utilisée en pratique dans notre cas, car le nombre d'alternatives admissibles est très grand.

Une autre solution, proposée dans [6] est d'implanter un *branch and bound* qui compare à chaque nœud

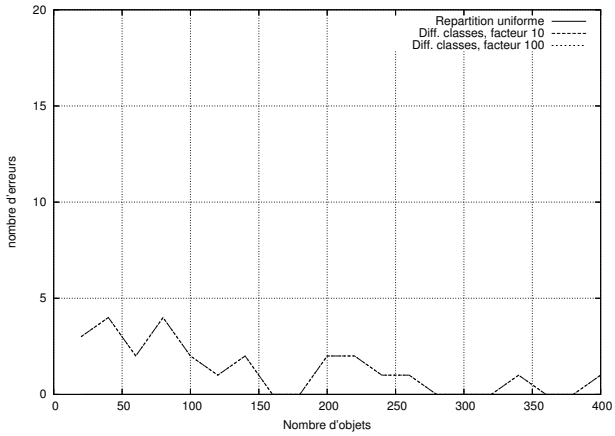
8. Voici informellement ce que l'on entend par «étanchéité» des classes : pour $f_c = 10$, 4 objets de poids 15 sont potentiellement meilleurs qu'un objet de plus faible poids (50) de la classe supérieure, alors qu'il en faut au moins 34 pour f_c =100. L'étanchéité est plus grande avec $f_c = 100$ qu'avec f_c =10.



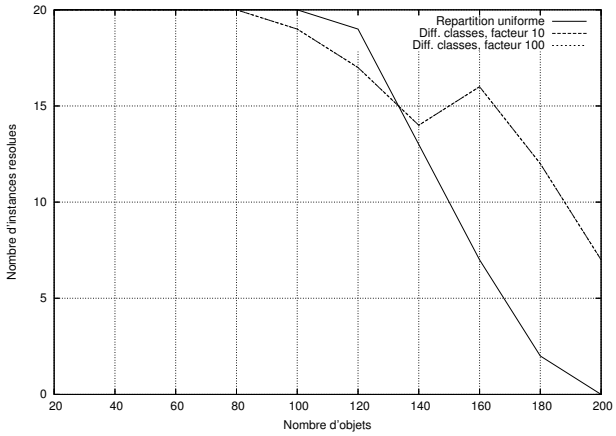
(a) Influence du nombre d'objets : temps de résolution avec Cplex (moyenne sur 20 instances).



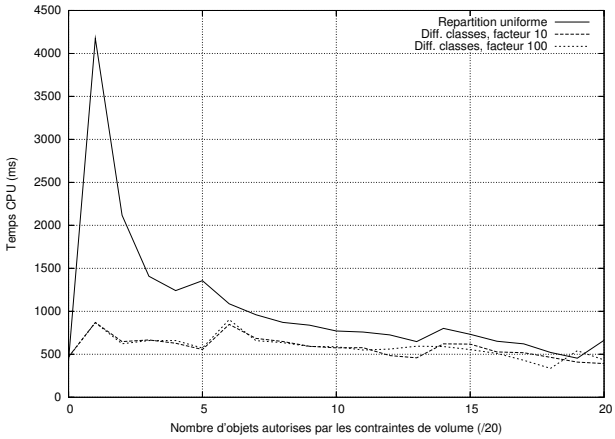
(b) Influence du nombre d'objets : temps de résolution avec Choco (moyenne sur 20 instances).



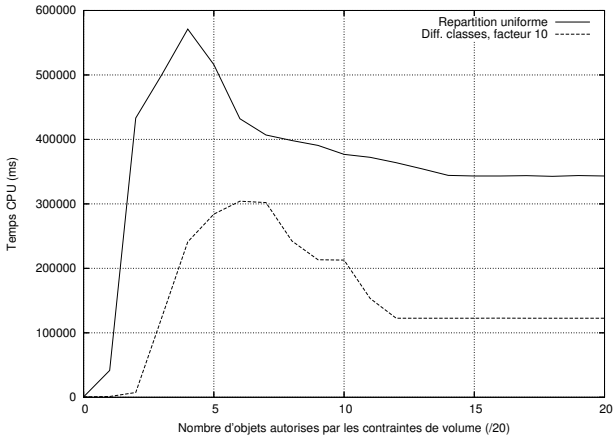
(c) Influence du nombre d'objets : nombre d'erreurs avec Cplex (sur 20 instances pour un nombre d'objets fixé).



(d) Influence du nombre d'objets : nombre d'instances résolues en moins de 10 minutes avec Choco (sur 20 instances pour un nombre d'objets fixé).



(e) Influence de la dureté des contraintes avec Cplex (temps de résolution moyen pour 20 instances).



(f) Influence de la dureté des contraintes avec Choco (temps de résolution moyen pour 20 instances).

Figure 2 – Influence du nombre d'objets et de la dureté des contraintes sur le temps de résolution.

de l'arbre de recherche le vecteur solution courant et le meilleur vecteur solution trouvé jusqu'ici.

Dans le même ordre idée, nous pouvons utiliser l'algorithme de PPC suivant : d'abord on calcule une solution au réseau de contraintes initial ; puis, à chaque nouvelle solution trouvée, on ajoute une contrainte obligeant les vecteurs solutions à être leximin-supérieurs au dernier vecteur trouvé ; on s'arrête lorsque le réseau est incohérent. Cet algorithme a été testé avec une contrainte d'ordre leximin fondée sur la contrainte de cardinalité **AtLeast**. Les résultats sont assez décevants, probablement en partie à cause du fait que notre contrainte d'ordre leximin n'a pas une puissance de filtrage assez conséquente pour détecter très rapidement les incohérences.

Ce dernier algorithme (ainsi que l'algorithme présenté en section 3) pourrait bénéficier de plusieurs travaux sur les contraintes. Tout d'abord, [8] introduit une contrainte *Multiset Ordering*, correspondant, dans le contexte des multi-ensembles, à une contrainte d'ordre leximax sur les vecteurs. Un algorithme efficace pour assurer l'arc-cohérence généralisée sur cette contrainte y est présenté, algorithme aisément adaptable à une contrainte d'ordre leximin. Nous pouvons aussi citer les travaux de [2] qui introduisent un algorithme de filtrage de bornes pour une contrainte $\mathbf{sort}(\overrightarrow{x}, \overrightarrow{y})$, dont le rôle est d'assurer que $\overrightarrow{y}$ est la version triée dans l'ordre non-décroissant de $\overrightarrow{x}$.

5.2 Travaux liés

L'article [1] décrit une méthode incomplète, de type recherche tabou, pour la résolution d'un problème multiagents d'allocation équitable de ressources satellitaires, qui est en fait une variante très proche de l'application réelle évoquée section 4, prenant en compte la plupart des nombreuses contraintes opérationnelles. Dans cette variante, les auteurs s'intéressent aussi à la recherche d'une allocation leximin-optimale, mais l'ordre leximin n'est pas traité directement comme nous l'avons fait ici : il est représenté de manière très proche par une fonction d'utilité collective à base d'OWA [22]. En outre, la méthode de résolution utilisée est dédiée spécifiquement à ce problème très précis, et à ce type de contraintes opérationnelles.

Le calcul de solutions leximin-optimales a d'autres domaines d'application que le choix social. On le trouve comme indiqué dans la section 1 en tant que raffinement de l'opérateur min dans le domaine des contraintes floues [7]. Il existe dans ce domaine des algorithmes de recherche de solutions discrimin- et leximin-optimales fondés sur un calcul récursif de degrés de cohérence de CSP flous [3], mais ces algorithmes sont inapplicables dans notre cas, en raison de la nature des contraintes et du nombre d'allocations admissibles.

Même si la prise en compte de l'équité dans les problèmes d'optimisation combinatoire ne semble pas très étudiée, on peut citer, entre autres, les travaux de Pesant et Régin [19] concernant une contrainte globale d'équilibrage de critères. Les procédures de filtrage qui lui sont associées sont issues de lois statistiques. Sur le plan du principe de prise en compte de l'équité, cette approche constitue une alternative intéressante à celle présentée ici. Il faudrait alors la coupler avec une optimisation garantissant une certaine forme d'efficacité.

6 Conclusions et perspectives

Nous avons étudié le problème général d'*allocation équitable* d'un ensemble de biens indivisibles à des agents, en présence de contraintes d'admissibilité quelconques, chaque agent possédant sa propre fonction d'utilité sur les allocations admissibles. Nous avons traduit l'équité par une particularisation Pareto-efficace du maximin sur les utilités des agents : la notion d'allocation *leximin-optimale*, qui s'applique potentiellement à tous les problèmes d'allocation multiagents pour lesquels la notion d'équité a un sens fort.

La contribution principale de cet article est la proposition d'un algorithme de calcul d'une allocation leximin-optimale, dans un cadre de programmation par contraintes (PPC), fondé sur l'utilisation de la méta-contrainte de cardinalité **AtLeast**. Le cadre général de la PPC est ici particulièrement intéressant, car il permet de décrire de façon séparée l'algorithme dédié au leximin, et d'autre part les fonctions d'utilité des agents et les contraintes d'admissibilité propres à chaque application potentielle. Dans un contexte où une bonne part des contraintes d'admissibilité est rarement fixée dans le marbre et évolue avec la vie de l'application, des outils comme la PPC permettent de s'adapter rapidement sans tout modifier.

Cet algorithme est justement proposé dans le cadre d'une application réelle : le partage de ressources satellitaires. De cette application nous avons extrait un problème simplifié d'allocation multiagents pour lequel nous avons construit un générateur aléatoire d'instances paramétré. Celui-ci nous a permis de tester deux implantations de l'algorithme proposé, en PPC (avec Choco [14]), et en PLNE (avec Cplex [12]). Les résultats obtenus montrent un net avantage en faveur de la PLNE, ce qui peut s'expliquer par la spécificité de ce cadre par rapport à la PPC dans notre cas. Plusieurs pistes restent à explorer pour rendre l'implantation en PPC plus efficace : heuristiques de choix des variables à instancier et de choix des valeurs des domaines, et procédures de filtrage des domaines des

variables (détection plus rapide des solutions incohérentes ou sous-optimales) entre autres.

L'article constitue une approche algorithmique du problème parmi d'autres, peut-être plus efficaces. Même si nous n'avons envisagé ici que des méthodes exactes, on pourra utiliser, pour des instances plus ardues, des méthodes incomplètes comme celles de [20], ou [21, 1] mixant programmation linéaire et recherche tabou. Nous espérons que notre générateur d'instances (en ligne) permettra à des équipes intéressées par le problème de proposer et valider d'autres approches.

Parmi les suites possibles de ce travail, citons :

— la comparaison expérimentale de l'algorithme proposé avec les autres algorithmes cités en section 5 ;

— l'étude d'approches plus souples et générales de l'équité, remplaçant l'ordre leximin par une fonction d'utilité collective paramétrée réalisant des compromis entre égalitarisme et utilitarisme classique ;

— l'application de l'algorithme à d'autres champs pratiques, comme la construction d'emplois du temps ou le partage de ressources aéroportuaires.

Les auteurs remercient les relecteurs pour leurs remarques pertinentes, Jérôme Lang et Jean-Michel Lachiver pour leurs discussions stimulantes sur le sujet, et Simon de Givry pour ses conseils sur CPLEX.

Références

[1] N. Bianchessi, J.-F. Cordeau, J. Desrosiers, G. Laporte, and V. Raymond. A heuristic for the multi-satellite, multi-orbit and multi-user management of earth observation satellites. *European Journal of Operational Research*, available online February 2006. doi :10.1016/j.ejor.2005.12.026.

[2] N. Bleuzen-Guernalec and A. Colmerauer. Narrowing a block of sortings in quadratic time. In *Proc. of CP'97*, pages 2–16, Linz, Austria, 1997.

[3] D. Dubois, H. Fargier, and H. Prade. Refinements of the maximin approach to decision-making in fuzzy environment. *Fuzzy Sets and Syst.*, 81 :103–122, 1996.

[4] D. Dubois and P. Fortemps. Computing improved optimal solutions to max-min flexible constraint satisfaction problems. *European Journal of Operational Research*, 1999.

[5] M. Ehrgott. *Multicriteria Optimization.* Number 491 in Lecture Notes in Economics and Mathematical Systems. Springer, 2000.

[6] H. Fargier, J. Lang, M. Lemaître, and G. Verfaillie. Partage équitable de ressources communes. (1) Un modèle général et son application au partage de ressources satellitaires. (2) Éléments de complexité et d'algorithmique. *Technique et Science Informatiques*, 23(9) :1187–1238, 2004.

[7] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proc. of EUFIT'93*, Aachen, 1993.

[8] A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Multiset ordering constraints. In *Proc. of IJCAI'03*, February 2003.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability, a guide to the theory of NP-completeness.* Freeman, 1979.

[10] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming.* Wiley-Interscience, 1972.

[11] P. Van Hentenryck, H. Simonis, and M. Dincbas. Constraint satisfaction using constraint logic programming. *A.I.*, 58(1-3) :113–159, 1992.

[12] ILOG. Cplex 10.0. `http://www.ilog.com/products/cplex/`.

[13] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives : Preferences and Value Tradeoffs.* John Wiley and Sons, 1976.

[14] F. Laburthe. CHOCO : Implémentation du noyau d'un système de contraintes. In *Actes des JNPC-00*, Marseille, France, 2000. `http://sourceforge.net/projects/choco`.

[15] M. Lemaître, G. Verfaillie, and N. Bataille. Exploiting a Common Property Resource under a Fairness Constraint : a Case Study. In *Proc. of IJCAI-99*, pages 206–211, Stockholm, 1999.

[16] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, 7 :95–132, 1974.

[17] H. Moulin. *Axioms of Cooperative Decision Making.* Cambridge University Press, 1988.

[18] H. Moulin. *Fair division and collective welfare.* MIT Press, 2003.

[19] G. Pesant and J-C. Régin. SPREAD : A balancing constraint based on statistics. In *Proc. of CP'05*, Sitges, Spain, 2005.

[20] M. Vasquez and J.-K. Hao. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Journal of Computational Optimization and Applications*, 20(2) :137–157, 2001.

[21] M. Vasquez and J.K. Hao. A Hybrid Approach for the 0–1 Multidimensional Knapsack Problem. In *Proc. of IJCAI-01*, volume 1, pages 328–333, August 2001.

[22] R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man, and Cybernetics*, 18 :183–190, 1988.