Chapitre 6
Expérimentations

Nous avons introduit au chapitre précédent plusieurs algorithmes dédiés à l'optimisation du préordre leximin dans le cadre de la programmation par contraintes. Nous allons, dans ce chapitre, présenter les résultats expérimentaux obtenus sur la comparaison de ces algorithmes sur des instances de problèmes de partage combinatoires générées artificiellement ou réelles.

L'intérêt de ce travail expérimental est double. D'une part il permet de mettre en valeur les caractéristiques opérationnelles des algorithmes introduits dans le chapitre 5, en comparant leurs efficacités respectives sur des instances très différentes. D'autre part, il nous a permis de travailler sur la génération d'instances réalistes de problèmes de partage. Cette problématique est cruciale dans le domaine du partage et plus généralement du choix social, car, la discipline n'ayant été que récemment étudiée par les chercheurs en informatique et intelligence artificielle, peu de travaux abordent le problème de la génération d'instances réalistes aléatoires dans ce contexte. On trouve toutefois une exception dans le domaine des enchères combinatoires [Leyton-Brown *et al.*, 2000], dont nous allons parler à la section 6.2.

Dans ce chapitre, nous allons introduire les quatre problèmes de partage particuliers sur lesquels nous avons travaillé, sous l'angle de la modélisation, de la génération d'instances, et de la comparaison des algorithmes dédiés au calcul de solutions leximin-optimales. Ces problèmes sont les suivants :

- ▷ **Le problème Pléiades simplifié :** il s'agit d'une version simplifiée du problème de partage de la constellation de satellites Pléiades, dans laquelle les contraintes opérationnelles sont approchées par des contraintes linéaires, et les préférences des agents représentées dans un langage additif.
- ▷ **Les enchères combinatoires équitables :** il s'agit d'une adaptation du *Winner Determination Problem* dans les enchères combinatoires, adaptation dans laquelle le critère est le préordre leximin et non la somme comme dans le problème classique.
- ▷ **Le problème de partage de biens indivisibles général :** il s'agit du problème de partage de biens indivisibles tel qu'il a été introduit au chapitre 3 dans la définition 3.38 (préférences et contraintes spécifiées dans un langage à base de logique).
- ▷ **Problème d'affectation de sujets de travaux expérimentaux :** il s'agit d'une instance réelle d'un problème d'affectation de sujets à des élèves.

Pour tous ces problèmes, nous avons travaillé sur des instances créées artificiellement, sauf pour le dernier d'entre eux, pour lequel nous disposons d'une instance réelle. Nous présenterons, pour chacune de ces applications, une description formelle du problème dans un premier temps, puis suivra un aperçu de la manière dont sont générées les instances, et enfin des résultats obtenus sur la comparaison expérimentale des algorithmes du chapitre 5 sur ces instances.

Spécifications techniques : Toutes les implantations logicielles décrites dans ce chapitre ont été effectuées dans le langage Java version 1.5.0. La bibliothèque de programmation par contraintes utilisée pour la résolution des problèmes est la bibliothèque Choco [Laburthe, 2000], et nous avons de même utilisé la bibliothèque ILOG CPLEX [ILOG, 2006] pour la résolution de certains problèmes linéaires. Les expérimentations ont été exécutées sur une station Sun équipée d'un processeur `SUNW,UltraSPARC-IIIi` tournant à 1,6GHz, d'une mémoire vive de 1Go et du système d'exploitation Solaris 10. Bien entendu, les temps de calcul donnés dans ce chapitre ne sont pas vraiment significatifs de l'efficacité absolue des algorithmes : ils dépendent de la configuration matérielle et logicielle de la plate-forme utilisée, ainsi que du système de programmation par contraintes lui-même. L'intérêt de ces résultats porte donc sur l'efficacité *relative* des algorithmes, étant donné qu'ils ont été exécutés sur les mêmes instances et dans les mêmes conditions.

6.1 Le problème Pléiades simplifié

6.1.1 Description et modélisation du problème

Afin de tester nos algorithmes de résolution dédiés au problème [MAXLEXIMINCSP] sur un problème simple à modéliser et relativement réaliste, nous avons extrait un problème simplifié d'allocation équitable d'objets à des agents de l'application 1 concernant le partage de la constellation de satellites Pléiades. Cette version simplifiée peut être vue comme un sous-problème du problème 3.38 (avec contraintes de volume), défini de la manière suivante.

- ▷ Les demandes sont atomiques, c'est-à-dire correspondant à des préférences additives, le poids d'un objet o dans les préférences de l'agent i étant noté $w_{i,o}$.
- ▷ Les contraintes sont de deux types :
  - Les contraintes de volume particulières à chaque agent : r_o est la *ressource consommée* par l'objet o, et $rmax_i$ est la *consommation de ressource maximale* autorisée pour l'agent i. Ces contraintes de volume permettent de simuler des droits exogènes inégaux sur les agents, en jouant sur l'introduction d'un quota sur la ressource consommée par les demandes plutôt que sur la fonction d'utilité collective elle-même.
  - Les contraintes de volume généralisées $\mathscr{C}_{vol}$: $v_{C,o}$ est le *volume* de l'objet o dans la contrainte de volume généralisé C, et $vmax_C$ est le *volume maximum* dans la contrainte de volume généralisé C. L'objectif de cet ensemble de contraintes est de simuler les contraintes physiques du problème réel qui restreignent l'ensemble des allocations admissibles : sur un intervalle de temps donné, le volume d'images acquises est limité par les capacités d'acquisition et d'agilité du satellite.

Notons que dans ce problème, la contrainte de préemption n'est pas présente, et l'on ne requiert pas la complétude de l'allocation. Ce problème peut être modélisé à l'aide de contraintes linéaires uniquement. Cette modélisation est fondée sur les variables suivantes :

- ▷ Un ensemble de variables 0–1 $\mathbf{x_{i,o}}$: $\mathbf{x_{i,o}} = 1$ si l'objet o est alloué à l'agent i, et 0 sinon, $o \in \mathscr{O}$, $i \in \mathscr{N}$. Les affectations possibles des variables $\mathbf{x_{i,o}}$ représentent l'ensemble des allocations possibles (parmi lesquelles se trouvent les admissibles).
- ▷ Un ensemble de variables numériques $\mathbf{u_i}$ représentant les utilités individuelles des agents : $\mathbf{u_i} = \sum_{o \in \mathscr{O}} \mathbf{x_{i,o}} \cdot w_{i,o}$ est l'utilité individuelle de l'agent i, $i \in \mathscr{N}$;
- ▷ Un ensemble de variables 0–1 $\mathbf{s_o}$: $\mathbf{s_o} = \max_{i \in \mathscr{N}} \mathbf{x_{i,o}} = 1$ si l'objet o est alloué à un agent au moins, et 0 sinon.

Les contraintes d'admissibilité du problème, c'est-à-dire les contraintes de volume particulières et généralisées sont les suivantes :

▷ $\sum_{o \in \mathscr{O}} \mathbf{x_{i,o}} \cdot r_o \leq rmax_i$, pour tout $i \in \mathscr{N}$ (contraintes de consommation de ressources) ;
▷ $\sum_{o \in \mathscr{O}} \mathbf{s_o} \cdot v_{c,o} \leq vmax_c$, pour tout $c \in \mathscr{C}$ (contraintes de volume généralisé).

Nous avons ajouté dans la modélisation du problème la contrainte suivante, dont le but est de cerner les allocations réellement significatives et d'augmenter l'efficacité de la modélisation : $w_{i,o} = 0 \Rightarrow \mathbf{x_{i,o}} = 0$ (un objet ne peut être alloué à un agent qui lui attribue un poids nul).

Nous allons donc nous intéresser au problème d'optimisation suivant :

Problème 17: [PLÉIADES SIMPLIFIÉ]

INSTANCE : Un ensemble d'agents $\mathscr{N}$, un ensemble d'objets $\mathscr{O}$, un ensemble de contraintes de consommation définies par $\{r_o \mid o \in \mathscr{O}\}$ et $\{rmax_i \mid i \in \mathscr{N}\}$, et un ensemble de contraintes de volume généralisées $\mathscr{C}_{vol}$.
SOLUTION : Une solution v de $maxleximin((\mathscr{X}, \mathcal{D}, \mathscr{C}), \overrightarrow{\mathbf{u}})$, avec $(\mathscr{X}, \mathcal{D}, \mathscr{C})$ le réseau de contraintes défini par les variables $\mathbf{x_{i,o}}$, $\mathbf{u_i}$ et $\mathbf{s_o}$ et les contraintes de volume généralisées et de consommation.

On peut remarquer que lorsqu'il n'y a qu'un seul agent et une seule contrainte (de consommation ou de volume), ce problème se réduit au problème suivant :

Problème 12 (rappel): [KNAPSACK] [Garey et Johnson, 1979, page 65]

INSTANCE : Un ensemble fini $\mathscr{S}$, une fonction de taille $s : \mathscr{S} \to \mathbb{N}$, une taille maximale $S_{max} \in \mathbb{N}$, une fonction de valeur (d'utilité) $u : \mathscr{S} \to \mathbb{N}$, et un entier K.
QUESTION : Existe-t-il un sous-ensemble $\mathscr{S}' \subseteq \mathscr{S}$ tel que $\sum_{a \in \mathscr{S}'} s(a) \leq S_{max}$ et $\sum_{a \in \mathscr{S}'} u(a) \geq K_{max}$?

S'agissant d'un problème NP-complet, notre problème de décision est donc bien sûr aussi NP-complet.

Plus précisément, on peut remarquer qu'il s'agit d'une généralisation du problème de sac à dos multidimensionnel en variables bivalentes [Vasquez et Hao, 2001], mais avec un critère d'optimisation leximin, au lieu du critère somme.

6.1.2 Génération des instances

Nous avons construit un générateur d'instances aléatoires de ce problème d'allocation équitable.

Ce générateur, écrit en Java, est disponible en ligne à l'URL `http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`. L'algorithme de génération des instances est réglable 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.

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.

On pourra trouver une description plus détaillée sur le générateur d'instances sur la page pointée par l'URL citée ci-avant.

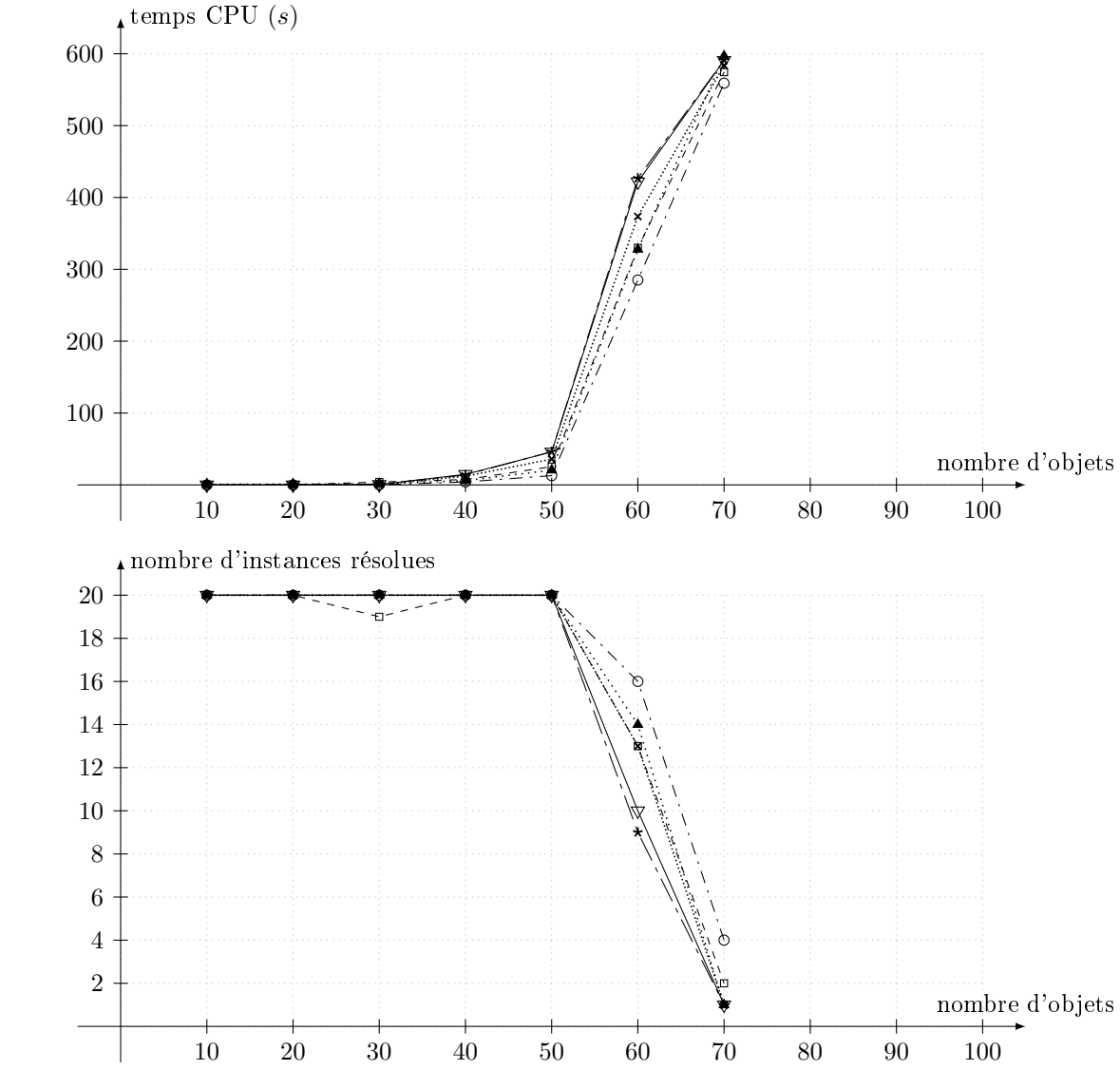
6.1.3 Résultats

Nous avons cherché pour les expérimentations à mettre en valeur l'influence du nombre d'agents et l'influence du nombre d'objets sur l'efficacité des algorithmes. D'un côté le nombre d'agents détermine la taille du vecteur objectif, et de l'autre l'influence du nombre d'objets permet de mettre en valeur le comportement des algorithmes face à la taille des instances, mais surtout de faire varier le type de problème auquel on a affaire.

Nous avons conduit trois séries d'expérimentations sur le problème Pléiades linéaire pour un nombre d'agents respectivement fixé à 4, 10 et 20 et pour un nombre d'objets variable. Dans ces trois séries, les instances ont été générées avec les paramètres par défaut du programme, sauf pour les poids des demandes, dont la répartition est uniforme entre 0 et 100, et pour les droits des agents, qui sont considérés comme égaux (ce qui rend le problème plus difficile en pratique). Les expérimentations ont été conduites sur 20 instances de chaque type donné, le temps de résolution étant limité à 10 minutes par instance. Les courbes présentées dans ce chapitre représentent d'une part le temps moyen d'exécution sur les 20 instances de chaque type, et d'autre part le nombre d'instances résolues dans la limite des 10 minutes. Notons que le temps moyen calculé n'est plus significatif si le nombre d'instances résolues en moins de 10 minutes est trop faible, car le temps de ces instances non résolues est pris en compte dans la moyenne comme étant égal à 10 minutes, ce qui biaise évidemment la moyenne. En conséquence, l'interprétation des courbes de moyenne ne peut se faire séparément de celle des courbes de nombre d'instances résolues.

On peut voir sur la figure 6.1 l'évolution du temps de calcul en fonction du nombre d'objets sur des instances de type Pléiades simplifié à 4 agents. Cette figure ne nous apporte que peu d'information sur l'efficacité relative des algorithmes, car les temps d'exécution de ceux-ci sont quasiment identiques. Cette similarité n'est en fait pas très étonnante. Le problème ne comportant que peu d'agents, il est donc relativement proche d'un problème monoagent sur lequel toutes les approches sont équivalentes (car fondé sur la maximisation de l'utilité individuelle de l'agent en question).

Lorsque l'on augmente le nombre d'agents, on commence à voir apparaître quelques différences entre les algorithmes, comme on peut le remarquer sur la figure 6.2 (10 agents). Sans surprise, l'algorithme fondé sur un calcul et une comparaison explicite de toutes les solutions du problème se révèle largement moins efficace que les autre approches sur ce type d'instances. Notons que l'algorithme 8 fondé sur les transformations max-min ne s'avère pas très efficace non plus ici. Dans le haut du tableau, on pourra remarquer les algorithmes 2 et 1, respectivement fondés sur le calcul des sous-ensembles saturés et sur la contrainte **Leximin**. Nous pouvons remarquer le début d'un



Algorithme 6 fondé sur la contrainte **AtLeast**.

Algorithme 2, inspiré de [Dubois et Fortemps, 1999].

Algorithme fondé sur le calcul et la comparaison de toutes les solutions du problème, inspiré par [Ehrgott, 2000].

Algorithme 8 fondé sur les transformations max-min.

Algorithme 1 fondé sur la contrainte **Leximin**.

Algorithme 5 fondé sur la contrainte **Sort**.

Figure 6.1 — Comparaison des temps de calcul des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances du type Pléiades simplifié à 4 agents.

phénomène concernant l'algorithme 2, qui va s'amplifier sur l'exemple suivant : lorsque le nombre d'objets est relativement faible, on note une perte d'efficacité de cet algorithme. L'explication en est relativement simple : lorsque le nombre d'objets est faible et de l'ordre du nombre d'agents, le jeu des contraintes implique l'impossibilité de satisfaire tous les agents, créant ainsi des *ex-aequo* dans les solutions leximin-optimales, rendant d'autant plus difficile le calcul des sous-ensembles saturés de cardinalité minimale (rappelons que la complexité de ce calcul dépend de manière mécanique du nombre d'utilités égales dans le vecteur objectif de la solution leximin-optimale).

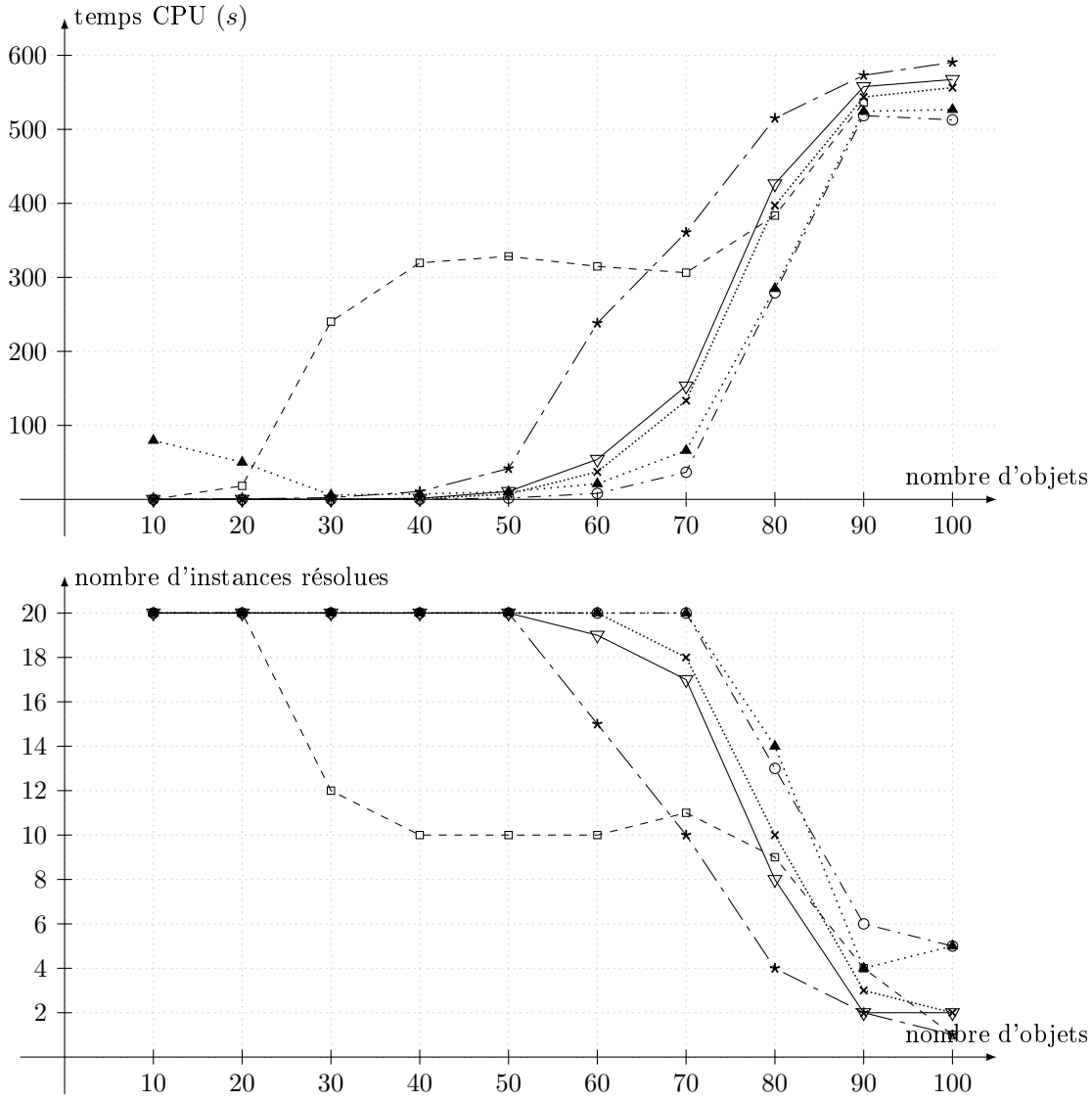
La figure 6.3 (20 agents) est particulièrement intéressante à plusieurs points de vue. D'une part, les écarts entre les algorithmes se creusent, permettant ainsi de les séparer en deux groupes : les algorithmes 1, 5 et 6 semblent relativement efficaces sur ce type d'instances. Le cas de l'algorithme 2 est très particulier : il se révèle complètement inefficace sur les instances à faible nombre d'objets, pour les raisons évoquées ci-avant concernant le calcul des sous-ensembles saturés, mais il semble relativement efficace sur les instances comportant un nombre d'objets plus élevé.

Nous pouvons remarquer sur toutes les courbes précédentes que deux algorithmes donnent des résultats extrêmement similaires, illustrés par la proximité de leurs courbes sur tous les graphes présentés : l'algorithme fondé sur la contrainte **AtLeast** et celui fondé sur la contrainte **Sort**. Cette similarité n'est pas surprenante à la lumière de la remarque que nous avons introduit lors de la description de l'algorithme 6 : ces deux contraintes sont fondées sur la même approche.

Heuristique de choix des variables Nous avons implanté pour les tests une heuristique de choix de variables fondée sur les principes énoncés dans la section 5.4.4 du chapitre précédent (les résultats présentés dans cette section tiennent compte de cette heuristique). Cette heuristique choisit la prochaine variable à instancier de la manière suivante : on choisit, parmi les objets de l'agent actuellement le moins satisfait, l'objet qui a un plus fort poids. L'efficacité de cette heuristique est remarquable. Selon les instances, les temps de calcul peuvent baisser de plus de 50%.

Programmation linéaire Concluons enfin ces commentaires sur les résultats des algorithmes testés sur les instances du modèle Pléiades simplifié en mentionnant le fait que nous avons adapté l'implantation de l'algorithme fondé sur la contrainte **AtLeast** à l'interface permettant de faire la liaison avec le solveur linéaire CPLEX. Nous avons pour cela utilisé la linéarisation de cette dernière contrainte, mentionnée lorsque nous l'avons introduite dans le chapitre 5. Les différences d'efficacité avec l'implantation fondée sur Choco sont flagrantes : dans les instances précédentes, CPLEX peut calculer une solution leximin-optimale en quelques secondes jusqu'à plus de 200 objets. Nous avons cependant rencontré quelques problèmes liés au passage en flottant par CPLEX, provoquant dans certains cas des erreurs de calcul qui, comme nous l'avons mentionné en fin du chapitre 5, peuvent provoquer un arrêt de l'algorithme (dû à une incohérence à une certaine itération), ou peuvent avoir de grandes répercussions sur les valeurs du vecteur leximin-optimal trouvé. La source de ces problèmes est peut-être due à un mauvais paramétrage de CPLEX.

Notons que la comparaison du temps de calcul entre l'implantation fondée sur CPLEX et les implantations fondées sur Choco n'a pas vraiment de sens : on cherche dans ce chapitre à comparer des différences d'approche du problème d'optimisation leximin, et non à comparer les outils de résolution eux-mêmes, comme nous l'avons fait remarquer en début de chapitre. Il serait en revanche intéressant de comparer dans le cadre de la programmation linéaire avec un outil comme CPLEX, l'algorithme fondé sur la contrainte **AtLeast** avec l'algorithme de [Dubois et Fortemps, 1999], l'algorithme inspiré de [Ehrgott, 2000], et l'algorithme de [Maschler *et al.*, 1992] fondé sur les transformations max-min.



Algorithme 6 fondé sur la contrainte **AtLeast**.

Algorithme 2, inspiré de [Dubois et Fortemps, 1999].

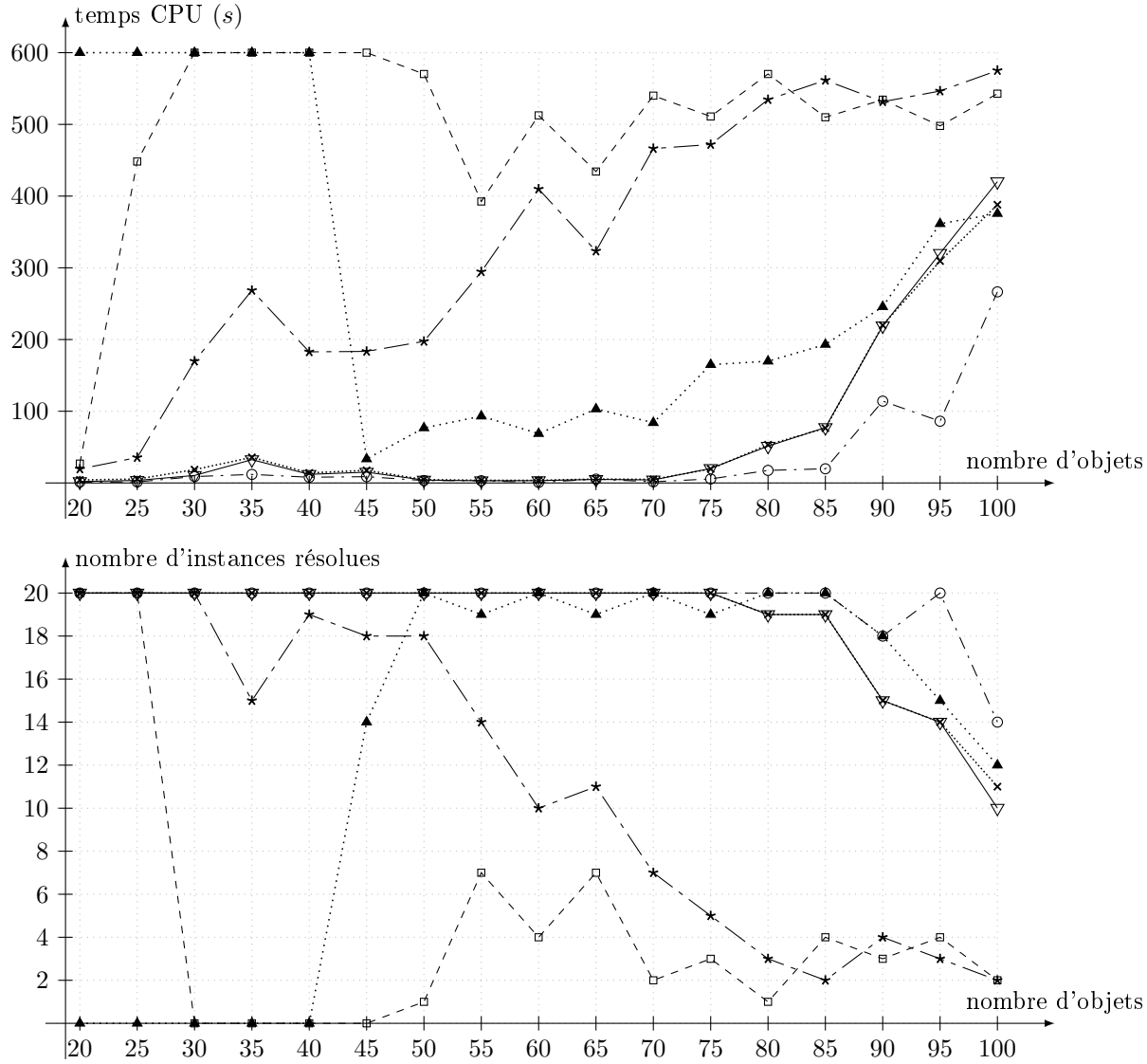
Algorithme fondé sur le calcul et la comparaison de toutes les solutions du problème, inspiré par [Ehrgott, 2000].

Algorithme 8 fondé sur les transformations max-min.

Algorithme 1 fondé sur la contrainte **Leximin**.

Algorithme 5 fondé sur la contrainte **Sort**.

Figure 6.2 — Comparaison des temps de calcul des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances du type Pléiades simplifié à 10 agents.



- Algorithme 6 fondé sur la contrainte **AtLeast**.
- Algorithme 2, inspiré de [Dubois et Fortemps, 1999].
- Algorithme fondé sur le calcul et la comparaison de toutes les solutions du problème, inspiré par [Ehrgott, 2000].
- Algorithme 8 fondé sur les transformations max-min.
- Algorithme 1 fondé sur la contrainte **Leximin**.
- Algorithme 5 fondé sur la contrainte **Sort**.

Figure 6.3 — Comparaison des temps de calcul des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances du type Pléiades simplifié à 20 agents.

6.2 Enchères combinatoires équitables

6.2.1 Description et modélisation du problème

Comme nous avons pu le voir au cours des chapitres précédents, le domaine des enchères combinatoires a suscité un intérêt croissant de la communauté du choix social et de l'intelligence artificielle ces dernières années, comme en témoignent les nombreuses publications qui lui sont consacrées [Cramton *et al.*, 2006; Sandholm, 1999, 2002; Rothkopf *et al.*, 1998; Lehmann *et al.*, 1999] (cette liste n'étant bien entendu pas exhaustive). Rappelons que le problème central des enchères combinatoires est le *Winner Determination Problem*, dédié à l'attribution des lots d'objets aux agents de manière à maximiser le gain du commissaire-priseur, ce qui correspond exactement à la maximisation de l'utilité collective utilitariste classique (la somme) si l'on considère que l'utilité d'un agent correspond au prix total auquel il estime les lots qu'il reçoit.

Si la sémantique des enchères combinatoires semble exclure l'égalitarisme comme critère d'agrégation des préférences individuelles, en revanche il est possible de s'appuyer sur ce problème et sur les travaux qui le concernent (notamment sur les langages d'expression des préférences) pour définir un problème de partage équitable d'objets. Ce n'est donc plus un problème d'enchères à proprement parler (car on ne s'intéresse plus aux gains du commissaire-priseur), mais un problème de partage défini comme suit :

Problème 18: Enchères combinatoires équitables

INSTANCE : Un ensemble de n agents $\mathscr{N}$, un ensemble d'objets $\mathscr{O}$ et un ensemble de mises $\mathscr{M}_i$ par agent i, exprimées dans l'un des langages de lots pour les enchères combinatoires introduits aux chapitre 3.

SOLUTION : Un partage $\overrightarrow{\pi}$ respectant la contrainte de préemption et tel qu'il n'existe aucun partage $\overrightarrow{\pi}'$ (respectant aussi la contrainte de préemption) tel que $(u(\pi_1), \ldots, u(\pi_n)) \prec (u(\pi'_1), \ldots, u(\pi'_n))$.

Les fonctions d'utilité des agents ont été définies lors de l'introduction des langages de lots dans la section 3.2.5.

6.2.2 Génération des instances

Comme nous l'avons fait remarquer en introduction de ce chapitre, s'il existe dans le domaine du choix social en général assez peu de travaux sur la génération d'instances de problèmes, le domaine particulier des enchères combinatoires fait exception à cette règle. Il existe en effet un générateur d'instances «réalistes» pour le problème d'enchères combinatoires, dont l'utilisation commence à se répandre et qui est devenu une référence dans le domaine : CATS [Leyton-Brown *et al.*, 2000] (disponible en ligne à l'URL `http://www.cs.ubc.ca/~kevinlb/CATS/`). Ce logiciel est dédié à la génération de mises pour les enchères combinatoires. La sémantique de ces mises est celle du *Winner Determination Problem* avec le langage OR* (voir la section 3.2.5 du chapitre 3). Les lots correspondant à ces mises ne sont pas mutuellement exclusifs sauf s'ils ont des objets en commun, et l'introduction d'objets factices permet de simuler des mises XOR, et donc d'enrichir le langage en permettant d'exprimer des substituabilités (rappelons que le langage OR seul ne le peut pas).

Décrivons rapidement la manière dont CATS génère l'ensemble de mises. Afin d'être les plus réalistes possibles, les algorithmes de génération des instances s'inspirent de problèmes réels. Cinq types de problèmes sont proposés (pour des informations plus détaillées, on pourra consulter l'article [Leyton-Brown *et al.*, 2000]) :

- ▷ **Chemins dans l'espace :** De nombreux problèmes réels d'enchères impliquent une recherche de chemins dans un graphe (tournées de véhicules, allocation de bande passante dans des réseaux, droit à utiliser un réseau ferré,...). CATS commence par générer un graphe approximativement planaire dont les nœuds (représentant des villes par exemple) sont placés de manière aléatoire dans le plan. Puis plusieurs ensembles de mises XOR sont générés : pour chaque ensemble on tire aléatoirement deux villes, puis on place un ensemble de mises XOR correspondant à tous les chemins possibles entre ces deux villes (la valeur des mises est aléatoire et dépendante de la distance euclidienne entre les deux villes).
- ▷ **Proximité dans l'espace :** Dans un autre grand type de problèmes, la complémentarité entre des objets est engendrée par leur proximité spatiale (exemple de vente aux enchères de terrains agricoles, droits de forages dans certaines zones, etc.). CATS génère un graphe qui est en fait un graphe d'adjacence (entre des terrains par exemple) : partant d'une grille parfaite (chaque nœud est connecté à 4 de ses voisins), il autorise avec une certaine probabilité, que certains nœuds soient connectés en diagonale, et que certaines connexions soient «oubliées» (représentant ainsi des terrains non rectangulaires). Les mises sont ensuite générées de manière incrémentale et aléatoire, en ajoutant les objets un par un dans les lots, avec une plus forte probabilité d'ajouter un objet s'il est connecté à l'un des objets du lot.
- ▷ **Relations arbitraires :** De manière générale (pour les ventes d'objets par exemple), il n'y a pas de relation spécifique entre les objets comme précédemment, mais on observe une certaine régularité dans les lots (régularité qui fait par exemple qu'un agent sera plus enclin à demander un lot {télévision, lecteur de DVD} qu'un lot {télévision, paire de chaussettes}). La génération des mises utilise une clique (un objet par nœud), avec une probabilité associée à chaque paire d'objets, qui indique la tendance qu'auront ces objets à s'associer. Comme précédemment, les lots sont générés de manière incrémentale.
- ▷ **Association temporelle :** Dans certains problèmes réels d'enchères, l'association entre les objets est temporelle. Par exemple, dans le problème de l'allocation de créneaux de décollage et d'atterrissage dans les aéroports (application 3), les créneaux sont liés par le fait qu'un avion qui effectue une liaison entre deux aéroports doit avoir les créneaux de décollage et d'atterrissage correspondant au temps de vol entre ces deux aéroports. Il y a aussi des lots substituables, correspondant aux différents couples décollage / atterrissage pour un même avion. La génération des lots prend en paramètre une carte des aéroports, et pour chaque vol, considère une utilité max u_{max} si le vol a ses créneaux décollage / atterrissage optimaux ; et une utilité inversement proportionnelle au retard engendré si ce n'est pas le cas.
- ▷ **Planification temporelle :** Il s'agit d'une formulation enchères combinatoires du problème de *job-shop scheduling*. Une usine lance une enchère pour les tranches d'utilisation d'une certaine ressource. Chaque enchérisseur a une tâche qui requiert un certain temps d'utilisation de la machine, et une *deadline*. Certaines tâches ont des *deadlines* additionnelles qui sont moins désirables pour l'enchérisseur. Dans la formulation orientée enchères combinatoires, un bien est un créneau d'utilisation de la machine. Les biens substituables correspondent aux différentes planifications possibles pour la même tâche.

En plus de ces cinq types de distribution, CATS peut générer les distributions classiques utilisées dans les papiers dédiés aux enchères combinatoires (distribution appelées *legacy*).

CATS considère que chaque agent exprime ses préférences sous forme de mises mutuellement exclusives (mises XOR) ; en d'autres termes, une seule mise peut être sélectionnée pour chaque agent. Puisque les mises XOR sont simulées par l'introduction d'objets factices (dans le langage OR*), on peut accéder à l'identité de l'agent concerné par une mise donnée grâce à l'objet factice qui accompagne le lot de la mise[1]. Puisque dans notre problème de recherche d'une solution lexi-

[1] Contrairement au *Winner Determination Problem* pour lequel l'information d'identité des agents n'est pas utile,

min optimale le nombre d'agents en jeu est un paramètre crucial, nous avons modifié légèrement l'exploitation du générateur afin de permettre à l'utilisateur de choisir un nombre d'agents inférieur au nombre d'agents généré par CATS, en fusionnant les mises de plusieurs agents différents de la manière suivante : nous groupons les agents deux par deux de manière aléatoire jusqu'à obtenir le nombre d'agents requis. Deux agents sont fusionnés simplement en groupant leurs préférences respectives en une mise OR* unique. Notons que cette manière de fusionner les mises n'est pas réellement satisfaisante, car le rapprochement des préférences des agents est complètement artificiel et crée des préférences non réalistes.

6.2.3 Résultats

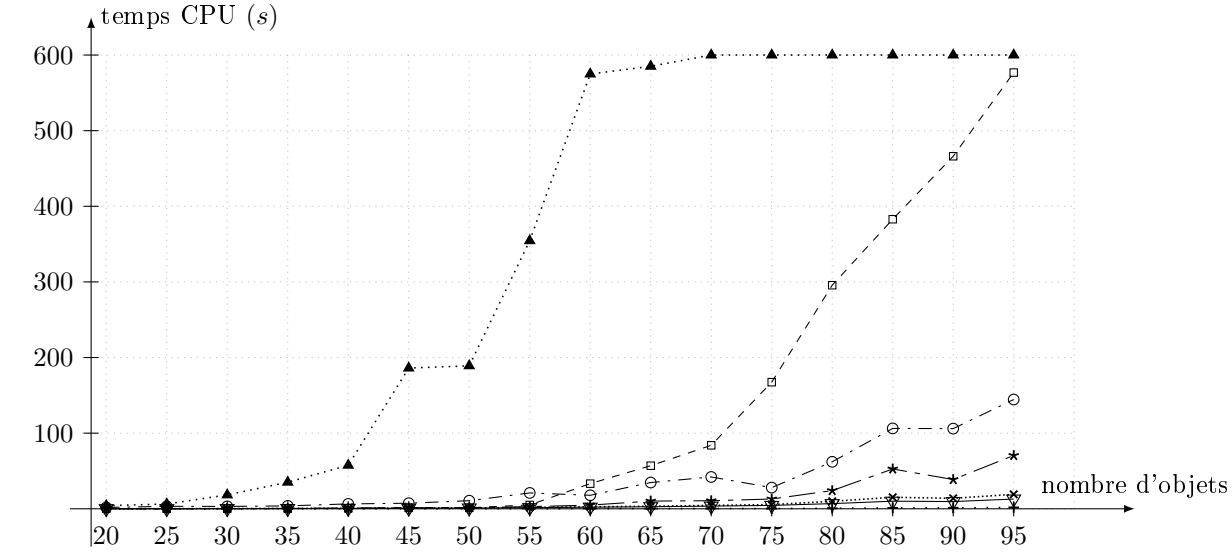
Nous avons mené plusieurs séries d'expérimentations sur les enchères combinatoires, afin toujours de mettre en valeur l'influence du nombre d'objets et du nombre d'agents sur les algorithmes. Les résultats de l'une de ces séries sont présentés en figure 6.4 (dans cette figure, nous avons omis le nombre d'instances résolues, car celui-ci est presque toujours égal à 20, sauf pour les deux algorithmes les moins performants). Pour cette série, les instances ont été générées à l'aide de CATS paramétré avec des lots de type «régions» (proximité dans l'espace), un nombre de mises égal au nombre d'objets et 20 agents. La figure met clairement en avant une nette infériorité des approches fondées sur les sous-ensembles saturés et sur la comparaison exhaustive de toutes les solutions. La taille des lots associée aux contraintes d'exclusion mutuelle pesant sur les lots peut expliquer l'échec du premier de ces deux algorithmes : dans de telles instances, il est très difficile de satisfaire tout le monde, donc il y a beaucoup d'agents non satisfaits dans les solutions leximin-optimales : on retombe dans le problème de calcul des sous-ensembles saturés.

L'algorithme fondé sur la contrainte **Leximin** semble beaucoup moins efficace que les algorithmes itératifs dans ce cas particulier (alors qu'il l'était beaucoup plus dans les instances de type Pléiades simplifié). L'explication que nous proposons est que le calcul des premières composantes du vecteur leximin-optimal dans les algorithmes itératifs résulte en un filtrage très important dès le début de la résolution (à cause de la morphologie du problème), ce qui peut expliquer leur efficacité dans ce cas bien précis.

La figure 6.5 présente des résultats relativement similaires à la figure 6.4. Pour cette série, les instances ont été générées avec des lots de type «arbitraires», un nombre de mises égal à 10 fois le nombre d'agents et 100 objets. L'échec de l'algorithme fondé sur les sous-ensembles saturés s'explique une nouvelle fois par l'augmentation du nombre d'agents par rapport au nombre d'objets qui reste fixe. On note un recul de l'efficacité de l'algorithme fondé sur les transformations max-min lorsque le nombre d'agents augmente : ces transformations sont certainement relativement coûteuses, car elles nécessitent l'introduction de variables supplémentaires dont le nombre croît de manière quadratique avec le nombre d'agents. Les algorithmes les plus efficaces sont ceux qui sont fondés sur la contrainte **Sort** et sur la contrainte **AtLeast**.

Enfin, nous avons comparé les temps de calcul d'une solution leximin-optimale au temps de calcul d'une solution somme-optimale, pour le même modèle, en programmation par contraintes. Bien entendu, la programmation par contraintes n'est pas l'outil le plus adapté pour traiter le *Winner Determination Problem*, et en particulier les solveurs *ad-hoc* proposés dans la littérature sur les enchères combinatoires sont bien plus performants. Nous pouvons toutefois noter que le calcul d'une solution leximin-optimale n'est pas énormément plus coûteuse que le calcul d'une solution somme-optimale dans le même modèle.

nous avons besoin de connaître cette identité pour le problème de recherche d'une solution leximin-optimale.



Algorithme 6 fondé sur la contrainte **AtLeast**.

Algorithme 2, inspiré de [Dubois et Fortemps, 1999].

Algorithme fondé sur le calcul et la comparaison de toutes les solutions du problème, inspiré par [Ehrgott, 2000].

Algorithme 8 fondé sur les transformations max-min.

Algorithme 1 fondé sur la contrainte **Leximin**.

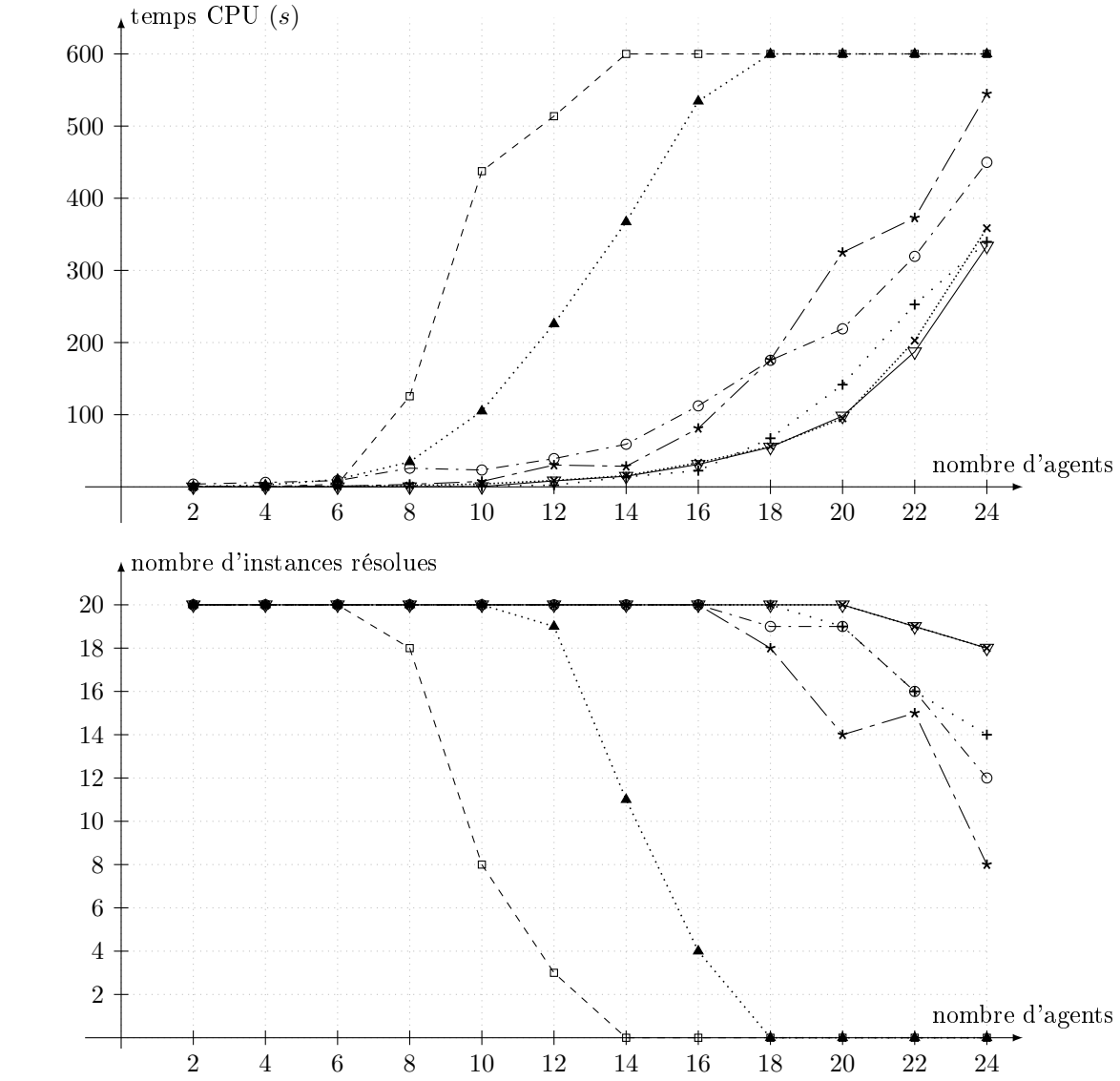
Algorithme 5 fondé sur la contrainte **Sort**.

Calcul de la solution optimale au sens de la somme (correspond au *Winner Determination Problem*).

Figure 6.4 — Comparaison des temps de calcul des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances de type enchères combinatoires à 20 agents.

Heuristique de choix des variables Tout comme pour les instances de type Pléiades simplifié, nous avons décliné l'heuristique de choix des variables suggérée dans la section 5.4.4 aux problèmes d'enchères combinatoires. Plus précisément, l'heuristique est la suivante : on choisit comme prochain lot à allouer le premier d'entre les lots non encore alloués, selon l'ordre lexicographique sur les critères suivants : (1) l'utilité courante de l'agent qui a placé cette mise, (2) le prix du lot. En d'autres termes, le prochain lot à allouer est celui qui a le prix le plus fort parmi ceux de l'agent le moins satisfait. Encore une fois, on relève une amélioration notable des performances grâce à l'utilisation de cette heuristique.

Équité de la solution Intéressons-nous brièvement à la caractérisation de l'équité des solutions obtenues par maximisation de l'ordre social leximin par rapport à l'équité des solutions obtenues par résolution du *Winner Determination Problem* avec critère utilitariste classique. Nous avons tracé sur la figure 6.6 les courbes de Lorenz des deux solutions utilitariste classique et égalitariste leximin pour une instance du problème d'enchères combinatoires avec 20 agents. La représentation graphique de ces deux courbes est relativement démonstrative : la solution égalitariste est beaucoup plus proche de la droite d'égalité parfaite entre les agents. En terme d'indices d'inégalité, le calcul des indices de Gini pour les deux profils donne une valeur d'environ 0,83 pour l'utilitarisme classique et 0,64 pour la solution leximin. Rappelons que plus l'indice d'inégalité est proche de 0, plus la distribution



Algorithme 6 fondé sur la contrainte **AtLeast**.

Algorithme 2, inspiré de [Dubois et Fortemps, 1999].

Algorithme fondé sur le calcul et la comparaison de toutes les solutions du problème, inspiré par [Ehrgott, 2000].

Algorithme 8 fondé sur les transformations max-min.

Algorithme 1 fondé sur la contrainte **Leximin**.

Algorithme 5 fondé sur la contrainte **Sort**.

Calcul de la solution optimale au sens de la somme (correspond au *Winner Determination Problem*).

Figure 6.5 — Comparaison des temps de calcul des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances de type enchères combinatoires à 100 objets.

est égalitaire.

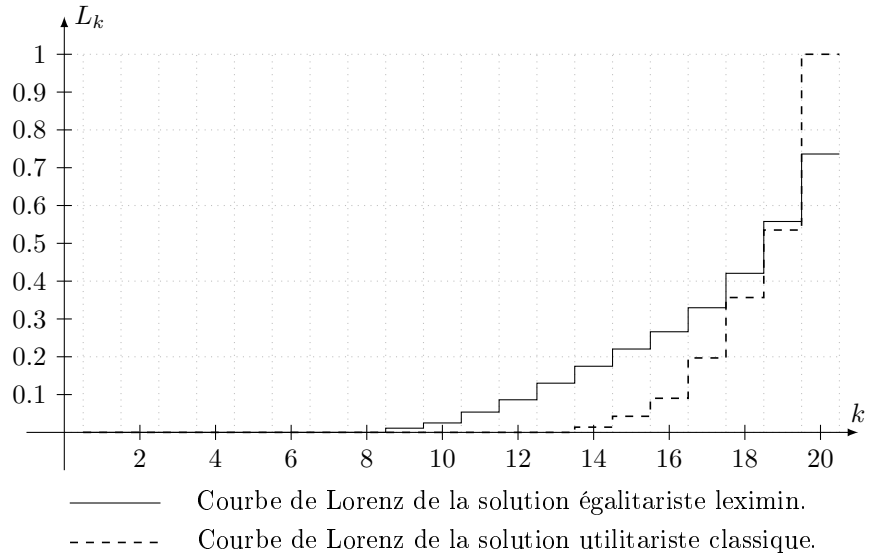


Figure 6.6 — Courbes de Lorenz des solutions utilitariste classique et égalitariste leximin pour une instance du problème d'enchères combinatoires.

6.3 Problème de partage de biens indivisibles générique

6.3.1 Description du modèle

Nous avons implanté le modèle du problème de partage de biens indivisibles générique fondé sur la représentation des préférences sous forme de logique pondérée et la représentation des contraintes sous forme de logique, problème que nous avons introduit dans la définition 3.38 dans le chapitre 3. Rappelons que ce problème est fondé sur :

- ▷ un ensemble d'agents $\mathscr{N}$;
- ▷ un ensemble d'objets $\mathscr{O}$;
- ▷ un ensemble de contraintes $\mathscr{C}$ exprimées sous forme logique sur le langage $L_{\mathscr{O}}^{alloc}$;
- ▷ les préférences des agents exprimées par un ensemble de formules logiques pondérées sur $\mathcal{L}_{\mathscr{O}}$;

- ▷ deux opérateurs d'agrégation.

Dans notre implantation de ce modèle, nous avons ajouté la possibilité d'exprimer deux types de contraintes spécifiques en plus des contraintes logiques :

- ▷ des contraintes de préemption, portant sur des ensembles d'objets donnés ;
- ▷ des contraintes de volume, définie par un volume maximum et portant sur des ensembles d'objets donnés auxquels on attribue un volume.

Ces contraintes étant omniprésentes dans les problèmes de partage réels, leur introduction spécifique permet de simplifier leur expression et leur prise en compte par les algorithmes de résolution, même si, comme nous l'avons précisé lors de l'introduction de ce modèle au chapitre 3, ces contraintes peuvent être encodées par des formules logiques de notre langage.

Dans notre implantation, les contraintes logiques peuvent être spécifiées de différentes manières :

- ▷ comme une formule logique générique (spécifiée par application récursive d'un opérateur logique sur des sous-formules) ;

- ▷ comme une clause du langage propositionnel ;
- ▷ comme une formule en forme normale disjonctive (par spécification des cubes sous forme d'un tableau).

De même, les formules logiques associées aux préférences peuvent être définies de différentes manières :

- ▷ comme une formule logique générique ;
- ▷ comme une formule en forme normale conjonctive ;
- ▷ comme une formule en forme normale disjonctive.

La possibilité de définir les contraintes et les préférences de différentes manières permet de prendre en compte de manière spécifique ces types de formules spécifiques (par exemple en introduisant des mécanismes de filtrage dédiés aux clauses ou aux cubes), afin d'une part de faciliter la spécification des problèmes, et d'autre part d'améliorer l'efficacité des algorithmes de résolution qui leur sont dédiés.

6.3.2 Génération des instances

Nous avons développé un générateur d'instances de ce problème, dont l'objectif est, tout comme le logiciel CATS pour les enchères combinatoires, de créer des instances réalistes de ce problème. Ce générateur, disponible en ligne à l'adresse `http://www.cert.fr/dcsd/THESES/sbouveret/benchmark2007/`, permet de créer trois types d'instances : générique, Pléiades et Spot.

6.3.2.1 Instances de type générique

Ce type d'instances est destiné à représenter des problèmes de partage génériques relativement simples. Les caractéristiques de ces instances sont les suivantes :

- ▷ les préférences sont constituées de formules en forme normale disjonctive (partant du principe qu'un agent sera plus enclin à exprimer naturellement ce genre de préférences) de taille et de poids aléatoires ;
- ▷ les contraintes logiques sont de deux types :
  - interdiction d'un objet particulier à un agent particulier,
  - interdiction d'attribuer un ensemble donné d'objets de taille aléatoire au même agent.
- ▷ la contrainte de préemption portant sur l'ensemble des objets est présente ou non.

Le générateur est bien entendu entièrement paramétrable. La manière de le paramétrer et les valeurs par défaut sont spécifiés sur la page web dédiée au générateur d'instances.

6.3.2.2 Instances de type Pléiades

Ce deuxième type d'instances du problème de partage de biens indivisibles est destiné à introduire un modèle du problème de partage de la constellation de satellite Pléiades qui soit légèrement plus réaliste que le problème linéaire introduit dans la section 6.1 de ce chapitre. La génération des instances se fait en deux temps.

Dans un premier temps, le programme génère une instance du modèle fondé sur les éléments suivants :

- ▷ La constellation est constituée d'un seul satellite, et le problème considère n_r révolutions du satellite autour de la Terre.
- ▷ Les agents fournissent un ensemble de demandes d'observation.
- ▷ Une demande d'observation est constituée d'une (requête simple) ou de plusieurs (requêtes *stereo* ou de zones larges) acquisitions qui doivent toutes être satisfaites pour satisfaire la

demande. Un poids est aussi associé à la demande, poids reflétant l'importance de cette demande pour l'agent.

- ▷ Une acquisition est constituée de plusieurs opportunités : une par révolution. Une acquisition est satisfaite si et seulement si au moins l'une des opportunités est satisfaite. Chaque acquisition est caractérisée par un numéro d'identification et le numéro de la révolution qu'elle concerne.
- ▷ Un ensemble de contraintes de volume simule les contraintes physiques, tout comme dans le premier modèle simplifié du problème.
- ▷ Certaines demandes d'observation sont de type particulier : les «demandes communes». Ces demandes d'observation concernent tous les agents en même temps, et si elles sont satisfaites, tous les agents sont satisfaits.
- ▷ Certaines demandes d'observation complexes comme les demandes *stereo* ou les images concernant une zone large doivent être acquises depuis la même révolution.

L'instance de ce modèle créée par le générateur est ensuite transformée en une instance du problème de partage de biens indivisibles. Ce générateur (entièrement paramétrable) permet de créer des instances légèrement plus réalistes que celles du problème Pléiades simplifié introduit en début de chapitre.

6.3.2.3 Instances de type Spot

Le troisième type d'instances particulières considéré pour le problème de partage de biens indivisibles est assez similaire aux instances de type Pléiades. Il est fondé sur le modèle constitué des éléments suivants :

- ▷ La constellation est formée de plusieurs satellites, et le problème considère n_r révolutions des satellites autour de la Terre. Les satellites possèdent chacun 3 instruments (avant, milieu et arrière), permettant de multiplier les opportunités d'acquisition. Les satellites ne sont pas agiles, mais un système de miroirs permet de changer dans une certaine mesure l'angle de visée latéral des instruments.
- ▷ Les agents fournissent un ensemble de demandes d'observation.
- ▷ Une demande observation est constituée d'une (requête simple) ou de plusieurs (requêtes *stereo*) acquisitions qui doivent toutes être satisfaites pour satisfaire la demande. Un poids est aussi associé à la demande, poids reflétant l'importance de cette demande pour l'agent.
- ▷ Une acquisition est constituée de plusieurs opportunités : une par révolution, par instrument et par satellite (sauf pour les images *stereo*, qui doivent être impérativement acquises par les instruments avant et arrière, lors de la même orbite). Chaque opportunité est caractérisée par l'instrument et la révolution concernés, un angle de déportation, et des temps de début et de fin de prise de vue. Les temps sont générés de manière aléatoire, soit selon une loi uniforme, soit concentrés autour de certains «points chauds», représentant des points d'intérêt particulier (comme des zones de conflits par exemple).
- ▷ Les contraintes sont de deux types :
  - des contraintes de volume, simulant les contraintes de consommation à bord : à chaque acquisition est associée une consommation de ressources qui dépend de sa durée, et chaque satellite ne peut acquérir plus d'images que sa consommation limite ne le permet ;
  - des contraintes d'exclusion mutuelle, fondées sur les temps de début et de fin des opportunités et sur leurs angles de déportation : le générateur effectue un calcul de faisabilité d'enchaînement de deux opportunités concernant le même satellite, et ajoute une contrainte d'exclusion mutuelle si cet enchaînement est impossible.

Après avoir créé une instance de ce problème particulier, le générateur la transforme, tout comme précédemment, en une instance du problème de partage de biens indivisibles.

6.3.3 Résultats

Les expérimentations des algorithmes sur ce type d'instances fournissent des résultats relativement décevants : les algorithmes sont rapidement mis en échec sur un très petit nombre d'objets et un nombre d'agents raisonnable, comme nous pouvons le voir sur la figure 6.7 et sur le tableau 6.1, concernant respectivement des instances de type logique générique et des instances de type Pléiades logique. Cette relative inefficacité peut s'expliquer par la difficulté des instances, qui impliquent un grand nombre de contraintes logiques[2]. On pourra néanmoins remarquer la supériorité relative des approches itératives (à l'exception de celle qui est fondée sur les transformations max-min) sur les approches fondées sur la contrainte **Leximin** et sur la comparaison exhaustive de toutes les solutions. On pourra remarquer aussi comme toujours la proximité entre les approches fondées sur la contrainte **AtLeast** et sur la contrainte **Sort**.

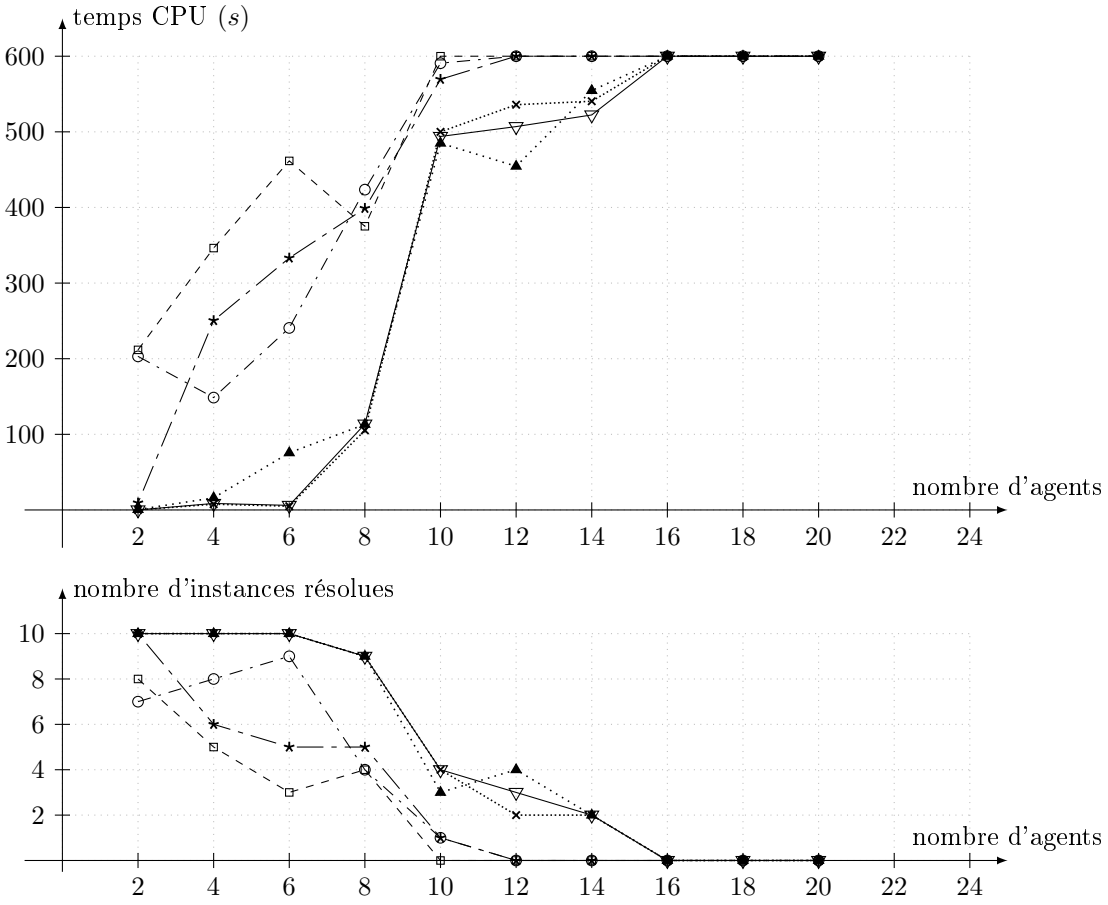
Instance		**AtLeast**				sous-ensembles saturés				comparaison exhaustive			
n	p	min	max	moy	nb	min	max	moy	nb	min	max	moy	nb
2	20	4	309	**45**	10	281	639	447	10	6	1675	295	10
2	25	6	942	202	10	369	1831	705	10	4	97940	12924	10
2	30	10	1448	275	10	407	2060	867	10	23	15770	4284	10
6	20	15	(—)	240070	6	2564	(—)	241853	6	38024	(—)	494419	2
6	25	98	(—)	364452	4	2887	(—)	371697	4	4216	(—)	540421	1
6	30	75	(—)	**423021**	3	3040	(—)	426813	3	(—)	(—)	(—)	0
10	20	231	(—)	**540023**	1	63931	(—)	546393	1	(—)	(—)	(—)	0

Instance		Max-min				**Leximin**				**Sort**			
n	p	min	max	moy	nb	min	max	moy	nb	min	max	moy	nb
2	20	3	552	87	10	4	56844	5968	10	3	76	21	10
2	25	4	3863	942	10	2	18310	5330	10	4	959	**192**	10
2	30	9	13299	1517	10	21	8223	2777	10	10	1371	**266**	10
6	20	21	(—)	360056	4	94769	(—)	513778	2	16	(—)	**240042**	6
6	25	236	(—)	540023	1	5037	(—)	540503	1	97	(—)	**364449**	4
6	30	(—)	(—)	(—)	0	(—)	(—)	(—)	0	85	(—)	424616	3
10	20	267	(—)	540026	1	(—)	(—)	(—)	0	240	(—)	540024	1

Tableau 6.1 — Temps de calcul (en ms) des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances de type problème logique Pléiades, à n agents et p objets.

Heuristique de choix des variables L'heuristique de choix des variables est plus difficile à mettre en œuvre dans le contexte des instances du problème logique générique, car l'effet de l'attribution d'un objet sur la satisfaction d'un agent n'est pas direct, du fait de la complexité sémantique des préférences. En outre, des considérations telles que le nombre de demandes dans lesquels apparaît l'objet et leur longueur doivent être prises en compte pour le choix des variables. Nous ne nous sommes pas encore penchés sur ce problème ; l'heuristique utilisée actuellement est la suivante : on choisit comme prochain objet à attribuer un objet apparaissant dans la demande de plus fort poids de l'agent le moins satisfait.

[2]Les paramètres par défaut pour le générateur aléatoire générique sont les suivants : entre $n/2$ et $3n/2$ contraintes particulières, entre 0 et $p/2$ contraintes d'exclusion concernant entre 2 et $p/2$ objets, entre 1 et 10 formules par agent concernant entre 1 et 15 objets chacune.



Algorithme 6 fondé sur la contrainte **AtLeast**.

Algorithme 2, inspiré de [Dubois et Fortemps, 1999].

Algorithme fondé sur le calcul et la comparaison de toutes les solutions du problème, inspiré par [Ehrgott, 2000].

Algorithme 8 fondé sur les transformations max-min.

Algorithme 1 fondé sur la contrainte **Leximin**.

Algorithme 5 fondé sur la contrainte **Sort**.

Figure 6.7 — Comparaison des temps de calcul des algorithmes de résolution du problème [MAXLEXIMINCSP] sur des instances de type problème logique générique, à 20 objets.

6.4 Problème d'affectation de sujets de travaux expérimentaux

Le problème d'affectation de TRavaux EXpérimentaux (« TREX ») est un exemple typique de problèmes de distribution de ressources indivisibles préemptives (le même objet ne peut être donné à plusieurs agents) sous contraintes, et avec un ensemble de préférences, tels qu'on peut en trouver dans les institutions comme des écoles, des universités, ou encore des entreprises. Le but de ce genre de problèmes est d'affecter un ou plusieurs sujets, tâches, ou objets, à chacun des agents présents, en tenant compte de leurs préférences exprimées comme un ordre sur les objets. Ce problème est présenté succinctement dans le chapitre d'introduction (application 4).

Nous avons à disposition une instance réelle, provenant de l'expression des choix des élèves de deuxième année à SUPAÉRO et des contraintes spécifiques aux sujets en 2005–2006. Nous allons dans un premier temps décrire l'instance particulière à laquelle nous avons affaire, puis nous nous intéresserons au calcul d'une solution leximin-optimale, et nous montrerons que toutes les approches testées ont échoué sur ce problème.

6.4.1 Description de l'instance réelle

On doit affecter un ensemble de 38 sujets de TREX à un ensemble d'élèves, regroupés en 63 binômes, formant eux-mêmes deux groupes D et H de 31 et 32 binômes respectivement. Les sujets sont regroupés en 9 différents pôles de compétence (Mathématiques, Informatique, Aérodynamique, ...) de la manière suivante : $\langle [\![1,5]\!], [\![6,10]\!], [\![11,13]\!], [\![14,15]\!], [\![16,18]\!], [\![19,22]\!], [\![23,30]\!], [\![31,37]\!], [\![38]\!] \rangle$.

On doit allouer les TREX aux binômes, sous les contraintes suivantes.

- ▷ **Préemption :** Un même sujet ne peut être affecté à deux binômes différents du même groupe dans la même série.
- ▷ **Fourniture :** Chaque binôme doit être pourvu d'un TREX exactement pour chaque série (1 et 2).
- ▷ **Variété de sujets :** Un même binôme ne peut se voir affecter deux sujets du même pôle de compétences.
- ▷ **Sujets spécifiques :** Certains sujets ont des contraintes spécifiques :
  - les 6 sujets suivants ne peuvent pas être choisis par un binôme du groupe D en série 2 : 5, 9, 32, 34, 35, 36 ;
  - les 6 sujets suivants ne peuvent pas être choisis par un binôme du groupe D en série 1 : 8, 18, 29, 31, 33, 37 ;
  - le sujet 38 ne peut être choisi par aucun binôme du groupe D, que ce soit en série 1 ou en série 2 ;
  - les 6 sujets suivants ne peuvent pas être choisis par un binôme du groupe H en série 2 : 8, 29, 31, 33, 37, 38 ;
  - les 6 sujets suivants ne peuvent pas être choisis par un binôme du groupe H en série 1 : 5, 18, 32, 34, 35, 36 ;

Chaque binôme a exprimé une préférence sur l'ensemble des sujets, sous la forme d'un ordre total (pas d'*ex-aequo* possible), en attribuant au sujet préféré la valeur 1, et au sujet le plus bas dans l'ordre de préférences la valeur 38. Le manière de calculer l'utilité d'un binôme en fonction des deux sujets qu'il reçoit n'est pas spécifiée dans les données du problème. Il y a toutefois trois manières intuitives de définir cette utilité :

- ▷ $u_i = 76 - k_1 - k_2$, si k_1 (resp. k_2) est la place du sujet reçu par le binôme i en série 1 (resp. série 2), en d'autres termes, la désutilité d'un agent à la somme (ou à la moyenne) des rangs des deux sujets qu'il reçoit ;
- ▷ $u_i = 38 - \max(k_1, k_2)$ (seul compte le pire des sujets), avec variante leximax possible ;

▷ $u_i = 38 - \min(k_1, k_2)$ (seul compte le meilleur des sujets), avec variante leximin possible.
Ces trois formulations de l'utilité individuelle sont valables d'un point de vue sémantique. Nous penchons plutôt pour la formulation ne prenant en compte que le pire des sujets, pour une raison d'efficacité de résolution : ce choix permet de modéliser le problème d'optimisation max-min «presque» comme un problème de flot dans un graphe, comme nous allons le voir plus loin.

L'objectif du problème est de trouver une affectation des sujets aux binômes qui satisfait toutes les contraintes, et soit équitable et efficace. Bien entendu, ce critère est assez flou et laisse toute latitude à l'interprétation des notions d'équité et d'efficacité. Toutefois, la modélisation de ces notions par le biais du préordre leximin nous semble relativement pertinent ici. Notons que le critère qui actuellement utilisé est le critère max-min / max-sum, dont nous avons parlé au tout début du chapitre 5.

Nous pouvons remarquer qu'étant donnée la spécification du problème, celui-ci peut être séparé en deux sous-problèmes disjoints : celui du groupe D et celui du groupe H. Dans les sous-sections suivantes consacrées à la résolution du problème, nous ne nous intéresserons qu'à la première moitié du problème, c'est-à-dire celle qui concerne le groupe D (sauf dans la formulation mathématique qui implique de manière explicite les deux groupes).

6.4.2 Modélisation et résolution du problème

6.4.2.1 Formulation mathématique du problème

Modélisation Ce problème peut être modélisé de manière naturelle sous la forme d'un réseau de contraintes constitué d'un ensemble de variables 0–1, correspondant à l'affectation des sujets aux binômes de chaque série. Ces variables sont regroupées en quatre matrices correspondant aux quatre couples possibles (groupe, série). Le réseau de contraintes est donc le suivant :

▷ **variables :** 4 matrices $\mathbf{D^{(1)}}$, $\mathbf{D^{(2)}}$ (de tailles 31×38), $\mathbf{H^{(1)}}$, $\mathbf{H^{(2)}}$ (de tailles 32×38) de variables ; pour une matrice X, on notera $x_{i,j}$ ses éléments ;

		Série 1 : $1 \cdots 38$	Série 2 : $1 \cdots 38$
Groupe D	$1 \vdots 31$	$\mathbf{D^{(1)}}$	$\mathbf{D^{(2)}}$
Groupe H	$1 \vdots 32$	$\mathbf{H^{(1)}}$	$\mathbf{H^{(2)}}$

▷ **domaines :** tous les domaines sont $\{0, 1\}$;

▷ **contraintes :**

- $\forall j \in [\![1, 38]\!]$, $\sum_{i=1}^{31} \mathbf{d_{ij}^{(1)}} \leq 1$ et $\sum_{i=1}^{31} \mathbf{d_{ij}^{(2)}} \leq 1$, } contraintes de préemption
- $\forall j \in [\![1, 38]\!]$, $\sum_{i=1}^{32} \mathbf{h_{ij}^{(1)}} \leq 1$ et $\sum_{i=1}^{32} \mathbf{h_{ij}^{(2)}} \leq 1$, } contraintes de préemption
- $\forall i \in [\![1, 31]\!]$, $\sum_{j=1}^{38} \mathbf{d_{ij}^{(1)}} = 1$ et $\sum_{j=1}^{38} \mathbf{d_{ij}^{(2)}} = 1$, } contraintes de fourniture
- $\forall i \in [\![1, 32]\!]$, $\sum_{j=1}^{38} \mathbf{h_{ij}^{(1)}} = 1$ et $\sum_{j=1}^{38} \mathbf{h_{ij}^{(2)}} = 1$, } contraintes de fourniture

- $\forall i \in [\![1,31]\!],\ \sum_{j=1}^{5} \mathbf{d_{i,j}^{(1)}} + \mathbf{d_{i,j}^{(2)}} \leq 1,$
- $\forall i \in [\![1,32]\!],\ \sum_{j=1}^{5} \mathbf{h_{i,j}^{(1)}} + \mathbf{h_{i,j}^{(2)}} \leq 1,$
- ...
- $\forall i \in [\![1,31]\!],\ \sum_{j=31}^{37} \mathbf{d_{i,j}^{(1)}} + \mathbf{d_{i,j}^{(2)}} \leq 1,$
- $\forall i \in [\![1,32]\!],\ \sum_{j=31}^{37} \mathbf{h_{i,j}^{(1)}} + \mathbf{h_{i,j}^{(2)}} \leq 1,$

} contraintes de variété

- $\forall j \in \{5,9,32,34,35,36\},\ \sum_{i=1}^{31} \mathbf{d_{i,j}^{(1)}} = 1$ et $\sum_{i=1}^{31} \mathbf{d_{i,j}^{(2)}} = 0,$
- $\forall j \in \{8,18,29,31,33,37\},\ \sum_{i=1}^{32} \mathbf{d_{i,j}^{(1)}} = 0$ et $\sum_{i=1}^{32} \mathbf{d_{i,j}^{(2)}} = 1,$
- $\sum_{i=1}^{31} \mathbf{d_{i,38}^{(1)}} + \sum_{i=1}^{31} \mathbf{d_{i,38}^{(2)}} = 0,$
- $\forall j \in \{8,29,31,33,37,38\},\ \sum_{i=1}^{31} \mathbf{h_{i,j}^{(1)}} = 1$ et $\sum_{i=1}^{31} \mathbf{h_{i,j}^{(2)}} = 0,$
- $\forall j \in \{5,18,32,34,35,36\},\ \sum_{i=1}^{32} \mathbf{h_{i,j}^{(1)}} = 0$ et $\sum_{i=1}^{32} \mathbf{h_{i,j}^{(2)}} = 1$

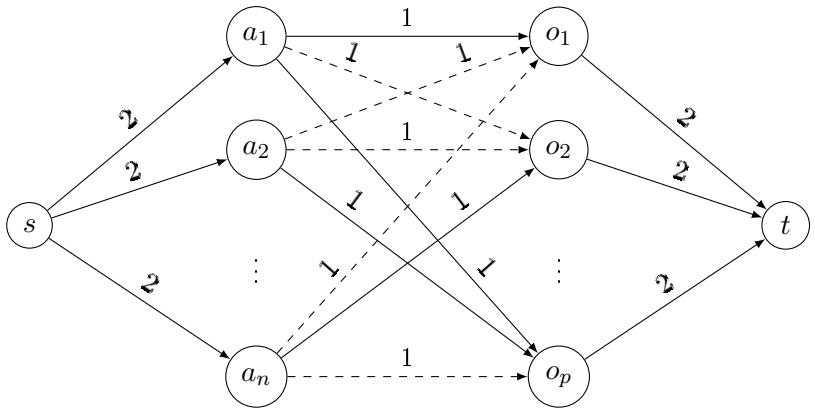
} contraintes spécifiques

Résolution Nous avons implanté en Java deux versions du modèle précédent : une version programmation par contraintes, destinée à être traitée avec la bibliothèque Choco, et une version en programmation linéaire, dédiée à l'interface avec le programme ILOG CPLEX. Contre toute attente, les approches tentées s'avèrent complètement inefficace sur le problème d'optimisation max-min (préalable à toute optimisation leximin) pour cette instance réelle : Choco comme CPLEX échouent à trouver une solution en un temps raisonnable (plusieurs jours), et ce, avec n'importe laquelle des trois fonctions d'utilité individuelle citées ci-avant. Ce résultat est d'autant plus étonnant que le calcul du max-somme est très rapide (quelques secondes). Remarquons que la méthode actuelle utilisée à SUPAÉRO est fondée sur des optimisations max-somme successives avec contrainte de borne sur l'utilité minimum : à chaque pas on essaie d'augmenter la borne de l'utilité minimum, jusqu'à ce que le problème mette trop de temps à être résolu. Il s'agit d'une approximation du critère max-min / max-somme (dont nous avons contesté la sémantique au début du chapitre 5).

6.4.2.2 Problème de flot sous contraintes

La formulation mathématique du problème dissimule quelque peu sa structure réelle, héritée de tous les problèmes d'affectation «de type couplage». Ce genre de problèmes se modélisent de manière assez naturelle sous la forme de problèmes de couplages dans un graphe bipartite, ou encore de problèmes de flots (ce qui est équivalent), exactement de la même manière que dans la proposition 4.23 page 162, détaillant la complexité du problème de partage de biens indivisibles pour des demandes atomiques et une fonction d'utilité collective de type min. Si l'on s'intéresse au cas où la fonction d'utilité individuelle est telle que seul compte le pire des sujets, on peut «presque» modéliser le problème d'affectation sous la forme du problème de flot présenté dans la figure 6.8. Le problème d'optimisation max-min revient à chercher la valeur K minimale telle qu'il existe un flot de valeur $2n$ dans le graphe pour lequel on a enlevé toutes les arêtes (a_i, o_j) tels que le sujet o_j est classé en dessous du rang K dans les préférences du binôme a_i. Cette valeur de K peut être calculée par dichotomie sur tout l'ensemble de valeurs possibles.

Le seul obstacle à cette modélisation est l'introduction des contraintes spécifiques et des contraintes de variété, qui rendent donc impossible la réduction de ce problème en un problème de flot. Cependant, on peut se fonder sur l'expression du problème de flot sans les contraintes spécifiques et de variété pour dériver un nouveau modèle linéaire : les variables correspondent aux arêtes sélectionnées ou non dans le graphe, et les contraintes expriment la conservation du flot à chaque nœud et les contraintes de capacité des arcs. On ajoute ensuite dans le modèle linéaire les contraintes spécifiques et de variété du problème d'affectation de sujets, et on détermine la valeur max-min par calculs de «flots» successifs comme ci-dessus.



Les arcs en traits pleins (resp. pointillés) correspondent aux couples (a_i, o_j) tels que le sujet o_j est classé en dessous (resp. au-dessus) du rang K dans les préférences du binôme a_i.

Figure 6.8 — Problème de flot correspondant au problème d'affectation de sujets de TREX.

Aussi étonnant que cela puisse paraître, cette seconde modélisation s'avère beaucoup plus efficace que la première pour le calcul de la valeur max-min optimale. À titre d'exemple, CPLEX met environ 10 millisecondes pour calculer cette valeur, ce qui est une avancée remarquable dans la résolution du problème.

Cette approche pose cependant problème pour le calcul d'une solution leximin-optimale. En effet, le fait que le calcul de la valeur max-min optimale soit effectué par améliorations successives d'une borne qui n'est pas une variable du solveur linéaire nous empêche d'utiliser un algorithme de résolution tel que celui qui est fondé sur la contrainte **AtLeast** (algorithme 6, page 191). Parmi les approches du problème [MAXLEXIMINCSP] présentées au chapitre 5, seul l'algorithme 2 inspiré de [Dubois et Fortemps, 1999] peut s'adapter à cette modélisation. Cependant, l'utilisation de cet algorithme dans ce cas précis se heurte à l'écueil du calcul des sous-ensembles saturés, que nous avons décrit lors de l'introduction de cet algorithme, et que nous avons mis en évidence sur les expérimentations précédentes. Dans le problème d'affectation de sujets, on peut estimer que les sous-ensembles saturés de cardinalité minimale contiennent en moyenne 10 variables. Si chaque résolution du problème de flot dure en moyenne 10 millisecondes, le parcours de tous les sous-ensembles de variables objectif (il y en a 62 dans notre cas) de taille inférieure ou égale à 10 durera $3,65 \times 10^6$ heures, soit 478 années : nous tombons dans le piège combinatoire.

6.5 Conclusion

Nous avons présenté dans ce chapitre un comparatif de l'efficacité des algorithmes introduits au chapitre 5 sur quatre types de jeux de tests : Pléiades simplifié, enchères combinatoires, partage de biens indivisibles à base de logique, et affectation de sujets de travaux expérimentaux. Ces tests ont permis de mettre en évidence la relative efficacité des approches fondées sur la contrainte **AtLeast** et la contrainte **Sort**, qui fournissent des temps de calcul corrects sur la plupart des instances des deux premiers jeux de tests. L'algorithme fondé sur la contrainte **Leximin** se montre plus efficace que les autres sur les instances Pléiades linéaires, mais moins efficace sur les autres types d'instances. L'algorithme inspiré de [Dubois et Fortemps, 1999] se révèle sans surprise à éviter à tout prix sur des instances produisant de nombreux *ex-aequo*, mais fournit des temps de calcul tout-à-fait raisonnables dès que l'on sort de ce type d'instances. L'algorithme fondé sur les transformations max-min a une

efficacité limitée, et enfin l'algorithme fondé sur une comparaison exhaustive de toutes les solutions est à éviter.

Cependant, comme le montrent les résultats obtenus sur les instances des deux derniers jeux de test (et en particulier sur l'instance réelle du problème d'affectation de travaux expérimentaux), les algorithmes se révèlent complètement inefficaces sur des instances plus complexes ou sur un problème réel d'affectation. Cette constatation plaide donc en faveur d'autres approches, telles que les algorithmes de type k-meilleurs [Gonzales *et al.*, 2006] par exemple. Le développement d'algorithmes de recherche incomplets, du type recherche locale, peut constituer une autre alternative crédible et en tout cas prometteuse.