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

Sylvain Bouveret et Michel Lemaître

Office National d'Études et de Recherches Aérospatiales
Centre National d'Études Spatiales
Institut de Recherche en Informatique de Toulouse

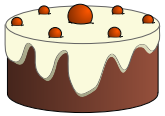
Journées Francophones de Programmation par Contraintes,
Nîmes, 9 juin 2006

Le problème de partage

Une ressource. . .

Le problème de partage

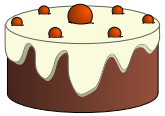
Une ressource. . .



Le problème de partage

Des agents. . .

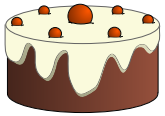
Une ressource. . .

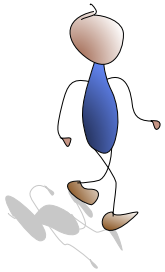


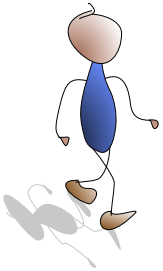
Le problème de partage

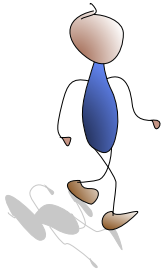
Des agents. . .

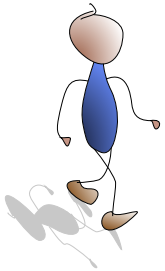
Une ressource. . .







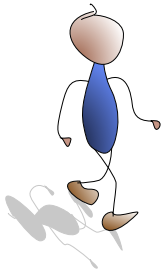


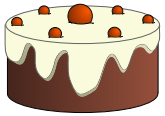


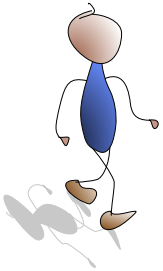
Le problème de partage

Des agents. . .

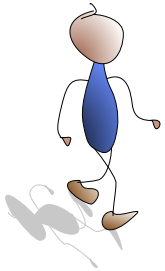
Une ressource. . .

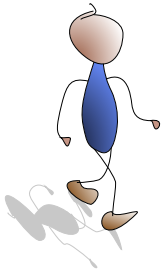




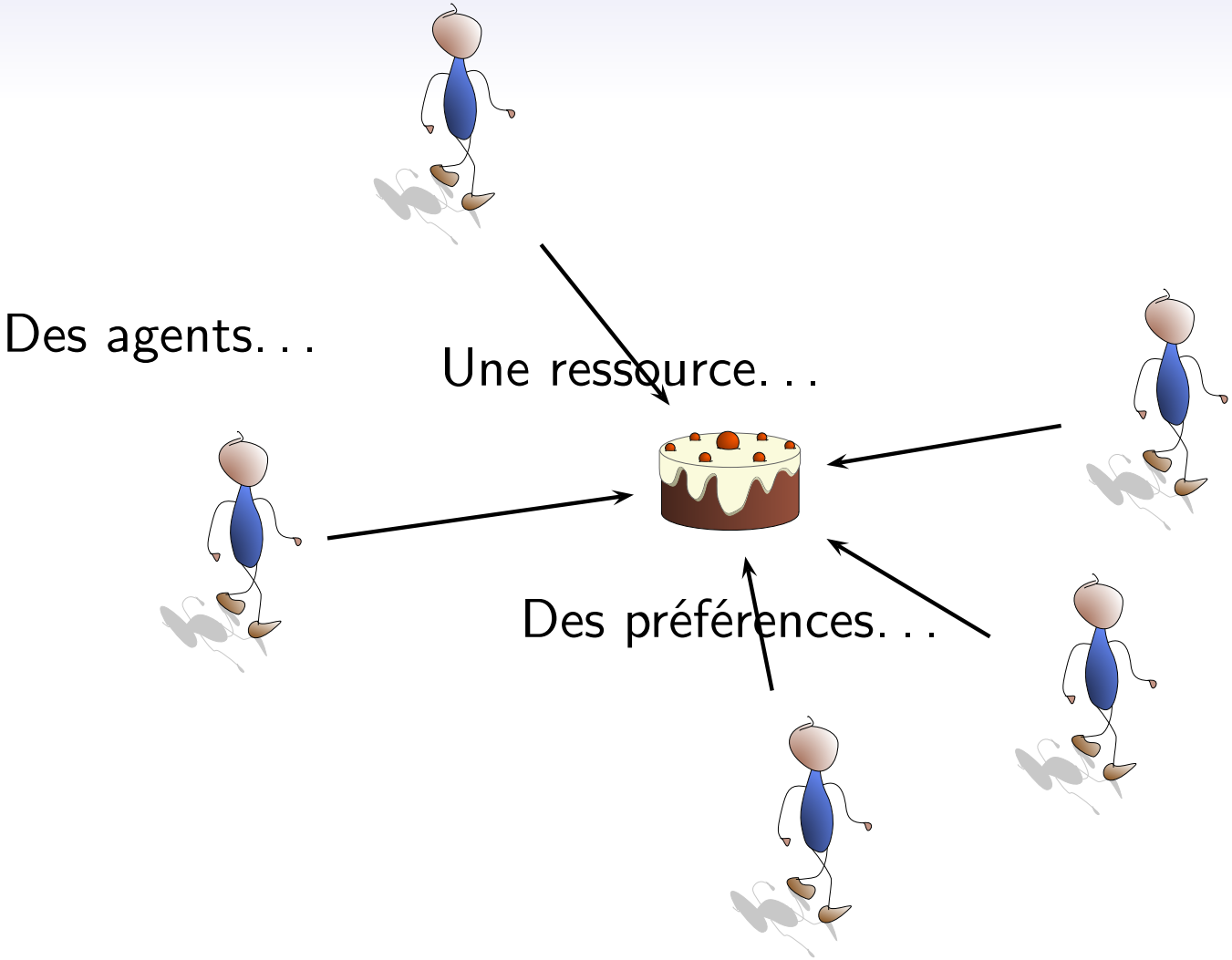


Des préférences. . .

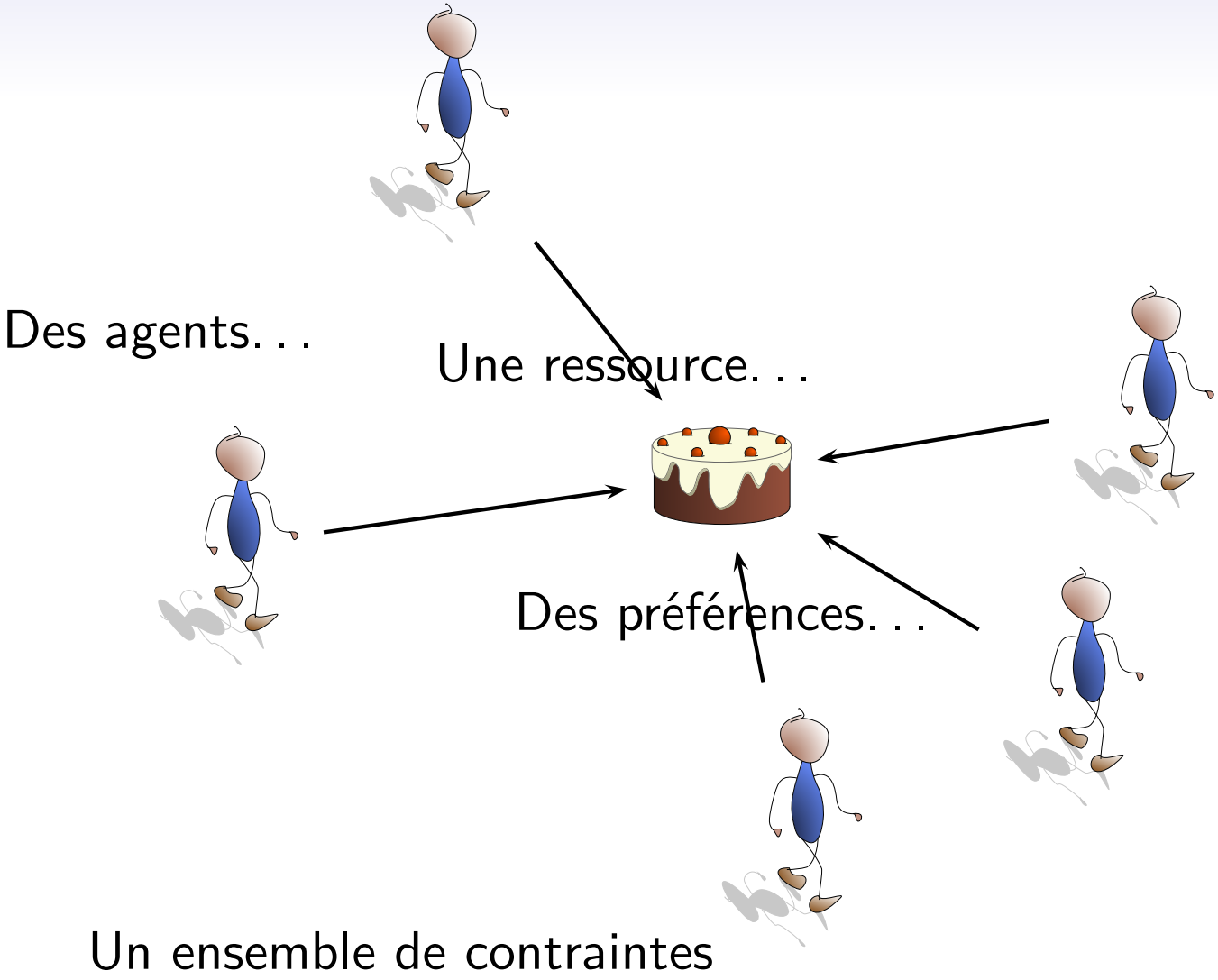




Le problème de partage



Le problème de partage



Le problème posé

Étant donnés :

- un ensemble d'agents ;
- un ensemble d'objets indivisibles (ressource) ;
- un ensemble de préférences des agents sur les objets ;
- un ensemble de contraintes d'admissibilité ;

Comment partager **équitablement** et **efficacement** de manière centralisée l'ensemble d'objets entre les agents ?

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

Problème de partage et *welfarism*

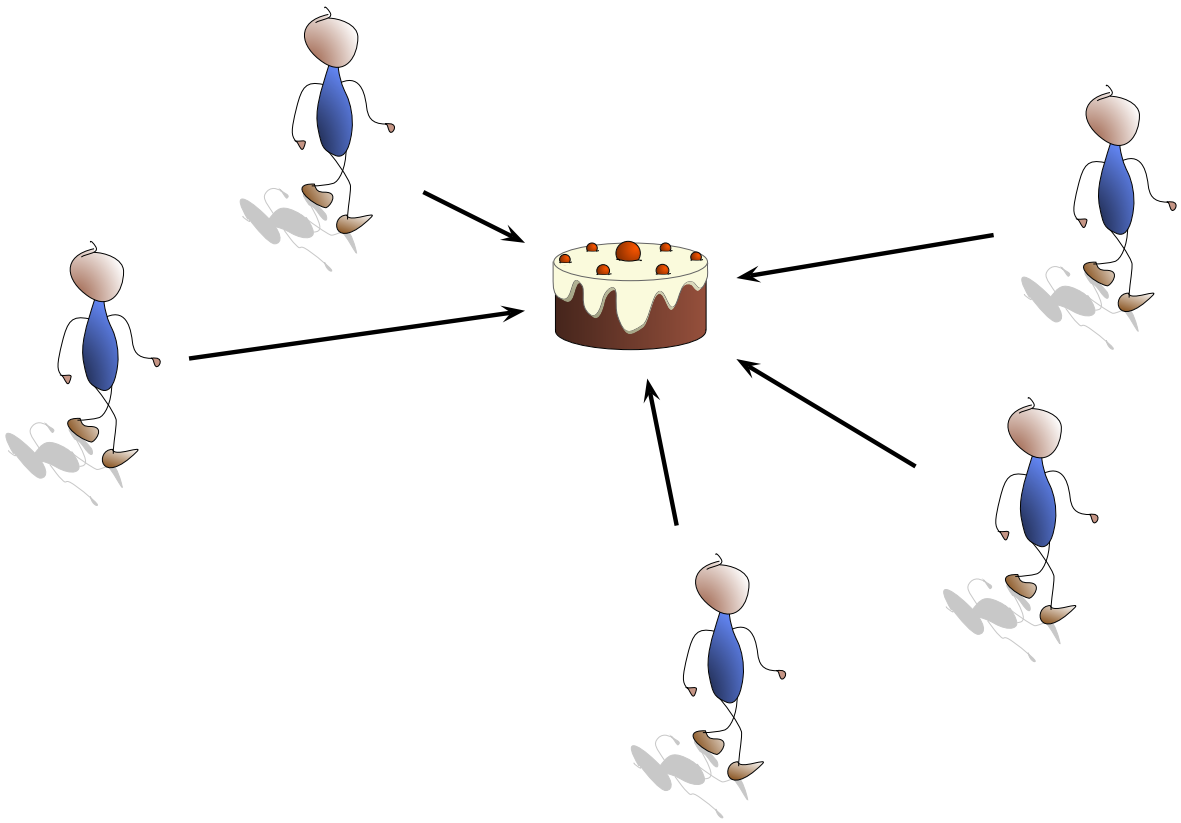
Hypothèse : La décision collective ne dépend que de la donnée, pour chaque agent et pour chaque alternative (partage), d'un niveau de satisfaction (indice numérique) : l'utilité individuelle.

Fonction d'utilité individuelle

Étant donné un agent a_i et un ensemble d'alternative $\mathcal{S}$ (ensemble des partages possibles), la fonction d'utilité individuelle de a_i est une fonction $u_i : \mathcal{S} \rightarrow \mathbb{N}$.

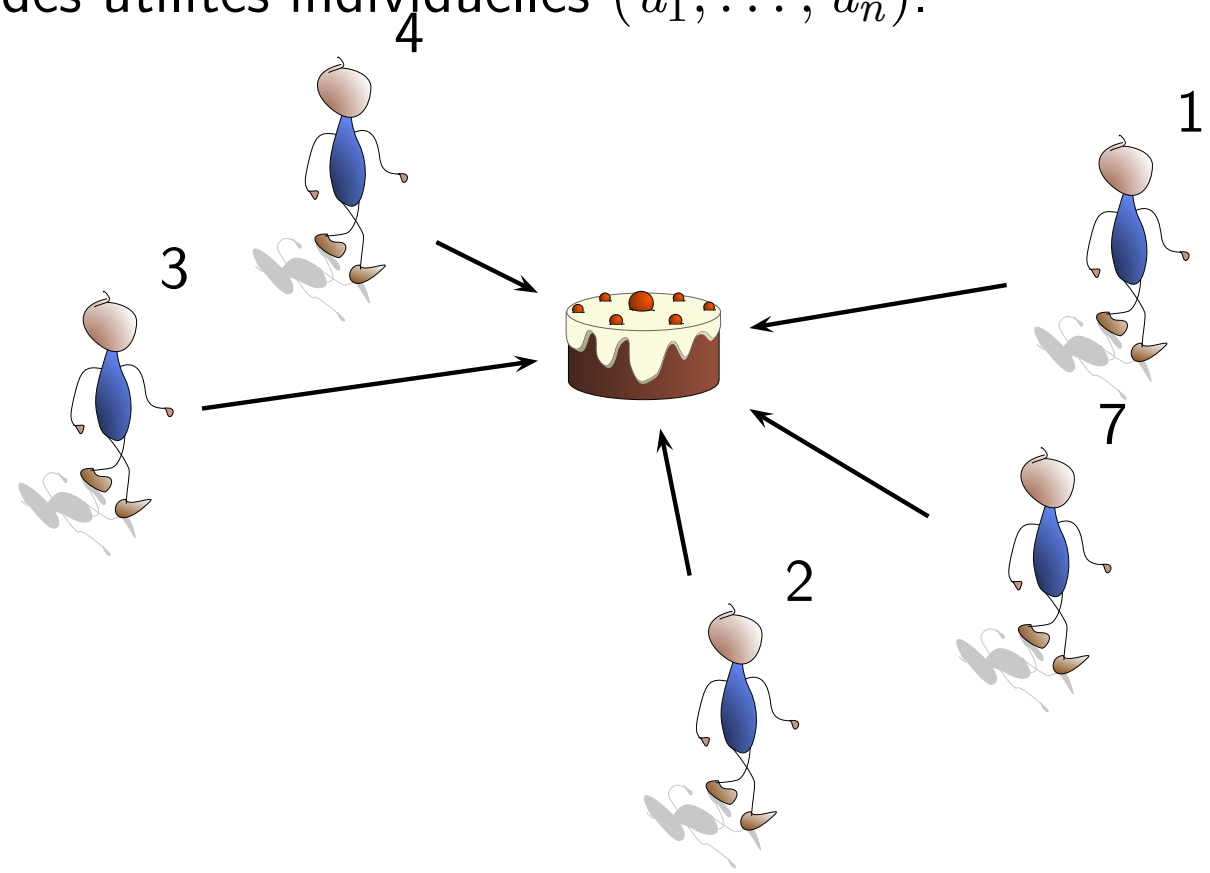
Problème de partage et *welfarism*

La théorie du **bien-être social** [Mou88] traite le problème de décision collective en attachant à chaque alternative faisable le vecteur des utilités individuelles $(u_1, \ldots, u_n)$.



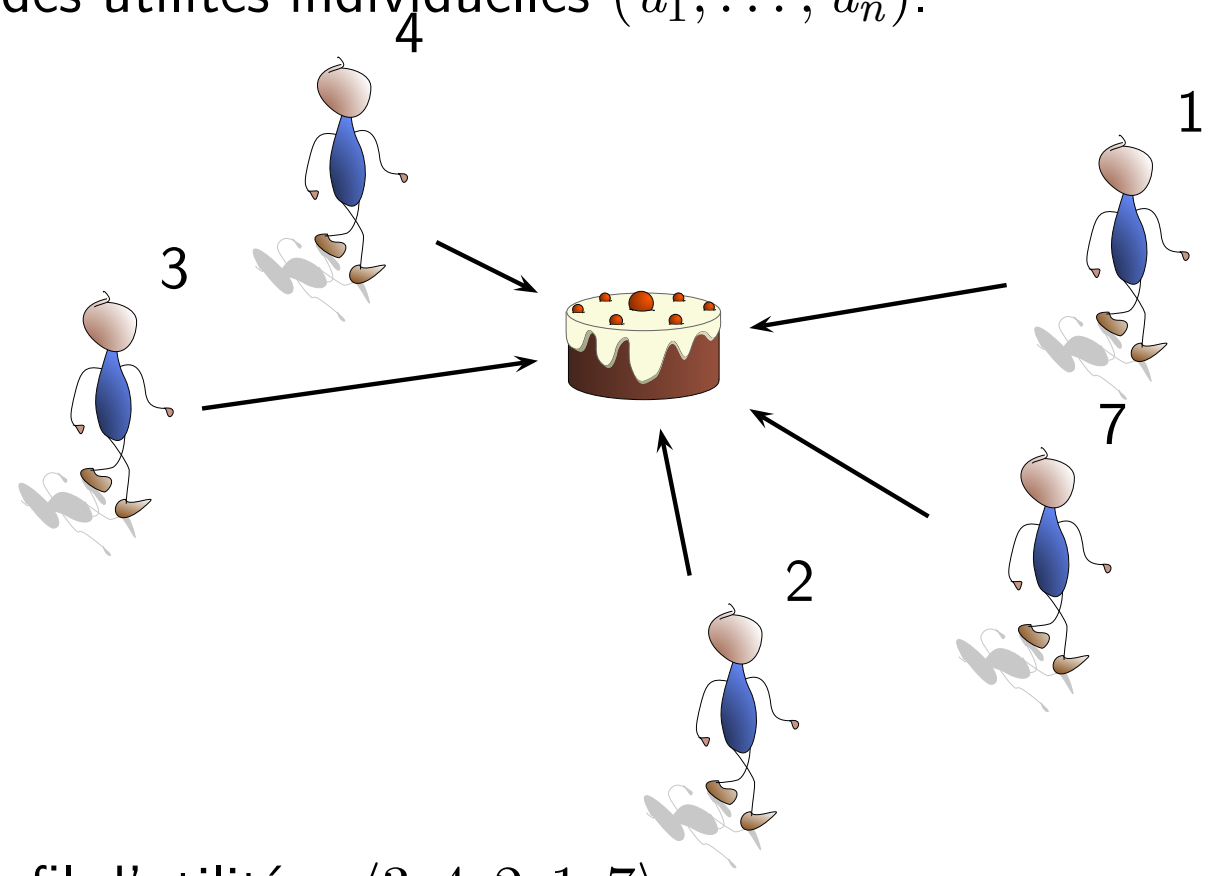
Problème de partage et *welfarism*

La théorie du **bien-être social** [Mou88] traite le problème de décision collective en attachant à chaque alternative faisable le vecteur des utilités individuelles $(u_1, \ldots, u_n)$.



Problème de partage et *welfarism*

La théorie du **bien-être social** [Mou88] traite le problème de décision collective en attachant à chaque alternative faisable le vecteur des utilités individuelles $(u_1, \ldots, u_n)$.



Profil d'utilités : $\langle 3, 4, 2, 1, 7 \rangle$

Ordre de bien-être social

Pour comparer les profils d'utilités, on utilise un **ordre de bien-être social**.

Ordre de bien-être social

Pour comparer les profils d'utilités, on utilise un **ordre de bien-être social**.

Ordre de bien-être social (SWO)

Un **ordre de bien-être social** est un préordre $\preceq$ sur $\mathbb{N}^n$.

Ordre de bien-être social

Pour comparer les profils d'utilités, on utilise un **ordre de bien-être social**.

Ordre de bien-être social (SWO)

Un **ordre de bien-être social** est un préordre $\preceq$ sur $\mathbb{N}^n$.

Exemples

- Ordre social représenté par la fonction d'utilité collective utilitariste.
- Ordre social représenté par la fonction d'utilité collective égalitariste.
- Ordre social leximin.

Fonctions d'utilité et ordres sociaux classiques

- Ordre social utilitariste.
- Ordre social égalitariste.
- Ordre social leximin.

Fonctions d'utilité et ordres sociaux classiques

- **Ordre social utilitariste.**
- Ordre social égalitariste.
- Ordre social leximin.

Ordre social utilitariste [Harsanyi]

$\overrightarrow{u} \preceq \overrightarrow{v} \Leftrightarrow \sum_{i=1}^{n} u_i \leq \sum_{i=1}^{n} v_i$.

Caractéristiques

Les agents sont des "producteurs" d'utilité.
Elle est indifférente aux inégalités entre les agents $\rightsquigarrow$ elle peut mener à des partages très inéquitables.

Fonctions d'utilité et ordres sociaux classiques

- **Ordre social utilitariste.**
- Ordre social égalitariste.
- Ordre social leximin.

Ordre social utilitariste [Harsanyi]

$\overrightarrow{u} \preceq \overrightarrow{v} \Leftrightarrow \sum_{i=1}^{n} u_i \leq \sum_{i=1}^{n} v_i$.

CUF utilitariste et équité

$\langle 10, 10, 10, 10 \rangle \preceq \langle 41, 0, 0, 0 \rangle$, alors que $\langle 10, 10, 10, 10 \rangle$ est plus équitable que $\langle 41, 0, 0, 0 \rangle$.

Fonctions d'utilité et ordres sociaux classiques

- Ordre social utilitariste.
- **Ordre social égalitariste.**
- Ordre social leximin.

CUF égalitariste [Rawls]

$\overrightarrow{u} \preceq \overrightarrow{v} \Leftrightarrow \min_{i=1}^{n} u_i \leq \min_{i=1}^{n} v_i$.

Caractéristiques

Elle ne prend en considération que le moins heureux des agents $\rightsquigarrow$ vocation équitable.

Fonctions d'utilité et ordres sociaux classiques

- Ordre social utilitariste.
- **Ordre social égalitariste.**
- Ordre social leximin.

CUF égalitariste [Rawls]

$\overrightarrow{u} \preceq \overrightarrow{v} \Leftrightarrow \min_{i=1}^{n} u_i \leq \min_{i=1}^{n} v_i$.

Caractéristiques

Elle ne prend en considération que le moins heureux des agents $\rightsquigarrow$ vocation équitable.
En revanche, elle ne satisfait pas le principe d'unanimité (effet de noyade).

Fonctions d'utilité et ordres sociaux classiques

- Ordre social utilitariste.
- **Ordre social égalitariste.**
- Ordre social leximin.

CUF égalitariste [Rawls]

$\overrightarrow{u} \preceq \overrightarrow{v} \Leftrightarrow \min_{i=1}^{n} u_i \leq \min_{i=1}^{n} v_i$.

CUF égalitariste et Pareto-efficacité

$\langle 1,1,1,1 \rangle \sim \langle 1,1000,1000,1000 \rangle$, alors que $\langle 1,1,1,1 \rangle$ et $\langle 1,1000,1000,1000 \rangle$ sont très différents !

Fonctions d'utilité et ordres sociaux classiques

- Ordre social utilitariste.
- Ordre social égalitariste.
- **Ordre social leximin.**

SWO leximin

Pour un vecteur $\overrightarrow{x}$, on note $\overrightarrow{x^{\uparrow}}$ la version triée dans l'ordre non-décroissant de $\overrightarrow{x}$.
$\overrightarrow{u} \succ_{leximin} \overrightarrow{v} \Leftrightarrow \exists k$ tel que $\forall i \leq k,\ u_i^{\uparrow} = v_i^{\uparrow}$ et $u_{k+1}^{\uparrow} > v_{k+1}^{\uparrow}$.

Caractéristiques

Il prend en considération tous les agents dans l'ordre de leur niveau de satisfaction $\rightsquigarrow$ vocation équitable.
Satisfait le principe d'unanimité.

Fonctions d'utilité et ordres sociaux classiques

- Ordre social utilitariste.
- Ordre social égalitariste.
- **Ordre social leximin.**

SWO leximin

Pour un vecteur $\overrightarrow{x}$, on note $\overrightarrow{x^{\uparrow}}$ la version triée dans l'ordre non-décroissant de $\overrightarrow{x}$.
$\overrightarrow{u} \succ_{leximin} \overrightarrow{v} \Leftrightarrow \exists k$ tel que $\forall i \leq k,\ u_i^{\uparrow} = v_i^{\uparrow}$ et $u_{k+1}^{\uparrow} > v_{k+1}^{\uparrow}$.

Leximin-optimal et Pareto-efficacité

$\langle 1,1,1,1 \rangle \prec \langle 1,1000,1000,1000 \rangle$ (car le deuxième indice dans le vecteur ordonné est discriminant).

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour notre problème.

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour notre problème.
Nous reformulons donc notre problème de partage comme suit :

Problème de partage avec critère leximin

Étant donnés :

- un ensemble d'agents ;
- un ensemble d'objets indivisibles (ressource) ;
- un ensemble de fonctions d'utilité individuelles ;
- un ensemble de contraintes d'admissibilité ;

Quel est le partage qui maximise l'ordre leximin sur les profils d'utilités ?

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour notre problème.
Nous reformulons donc notre problème de partage comme suit :

Problème de partage avec critère leximin

Étant donnés :

- un ensemble d'agents ;
- un ensemble d'objets indivisibles (ressource) ;
- un ensemble de fonctions d'utilité individuelles ;
- un ensemble de contraintes d'admissibilité ;

Quel est le partage qui maximise l'ordre leximin sur les profils d'utilités ?

$\rightsquigarrow$ Nous exprimerons ce problème dans le cadre de la programmation par contraintes.

Le problème de satisfaction de contraintes

Réseau de contraintes [Mon74]

Un réseau de contraintes est formé :

- d'un ensemble de variables $\mathcal{X} = \{x_1, \ldots, x_p\}$;
- d'un ensemble de domaines $\mathcal{D} = \{d_{x_1}, \ldots, d_{x_p}\}$;
- d'un ensemble contraintes $\mathcal{C}$, avec pour tout $C \in \mathcal{C}$:
  - $X(C)$ scope de la contrainte,
  - $R(C)$ ensemble de tuples autorisés par la contrainte.

Problème de satisfaction de contraintes (CSP)

Étant donné un réseau de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, existe-t-il une instanciation complète cohérente de ce réseau ?

Cadre employé dans la résolution de problèmes combinatoires divers : emplois du temps, planification, allocation de fréquences. . .

Le problème de satisfaction de contraintes avec objectif

Déclinaison du CSP en problème d'optimisation :

CSP avec variable objectif

Étant donnés un réseau de contraintes $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ et une variable objectif $o \in \mathcal{D}$ telle que $d_o \subset \mathbb{N}$, quelle est la valeur de d_o maximale faisant partie d'une instanciation complète cohérente du réseau ?

Le problème Leximin-Optimal

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

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

Principe de l'algorithme

Reformulation de la condition de leximin-optimalité sur un vecteur objectif $\overrightarrow{u}$:

Le vecteur leximin-optimal est $\langle 4, 7, 10 \rangle \Leftrightarrow$

Principe de l'algorithme

Reformulation de la condition de leximin-optimalité sur un vecteur objectif $\overrightarrow{u}$:

Le vecteur leximin-optimal est $\langle 4, 7, 10 \rangle \Leftrightarrow$

1. 4 est la valeur maximale de y_1 telle qu'il existe une instanciation complète cohérente v telle que :
   - tous les $v(u_i)$ valent au moins y_1 ;

Principe de l'algorithme

Reformulation de la condition de leximin-optimalité sur un vecteur objectif $\overrightarrow{u}$:

Le vecteur leximin-optimal est $\langle 4, 7, 10 \rangle \Leftrightarrow$

1. 4 est la valeur maximale de y_1 telle qu'il existe une instanciation complète cohérente v telle que :
   - tous les $v(u_i)$ valent au moins y_1 ;
2. 7 est la valeur maximale de y_2 telle qu'il existe une instanciation complète cohérente v telle que :
   - tous les $v(u_i)$ valent au moins 4,
   - au moins 2 valeurs parmi les $v(u_i)$ valent au moins y_2 ;

Principe de l'algorithme

Reformulation de la condition de leximin-optimalité sur un vecteur objectif $\overrightarrow{u}$:

Le vecteur leximin-optimal est $\langle 4, 7, 10 \rangle \Leftrightarrow$

1. 4 est la valeur maximale de y_1 telle qu'il existe une instanciation complète cohérente v telle que :
   - tous les $v(u_i)$ valent au moins y_1 ;
2. 7 est la valeur maximale de y_2 telle qu'il existe une instanciation complète cohérente v telle que :
   - tous les $v(u_i)$ valent au moins 4,
   - au moins 2 valeurs parmi les $v(u_i)$ valent au moins y_2 ;
3. 10 est la valeur maximale de y_3 telle qu'il existe une instanciation complète cohérente v telle que :
   - tous les $v(u_i)$ valent au moins 4,
   - au moins 2 valeurs parmi les $v(u_i)$ valent au moins 7 ;
   - au moins 1 valeur parmi les $v(u_i)$ valent au moins y_3 ;

La contrainte AtLeast

L'algorithme est fondée sur la méta-contrainte **AtLeast**

Contrainte **AtLeast** [HSD92]

Soit Γ un ensemble de p contraintes, et $k \in [\![1, p]\!]$ un entier. Alors la méta-contrainte $\mathbf{AtLeast}(\Gamma, k)$ 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.

Description de l'algorithme

Algorithme

si $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **«Incohérent» alors retourner «Incohérent»**

$\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$;

$\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;

$\mathcal{C}' \leftarrow \mathcal{C}$;

pour $i \leftarrow 1$ **à** n **faire**

$\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\quad \widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$\quad d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$\quad d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

retourner $\widehat{v}_{\downarrow \mathcal{X}}$

Description de l'algorithme

Algorithme

si $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$**«Incohérent» alors retourner «Incohérent»**

$\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$; **Introduction des variables** $\mathbf{y_i}$

$\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;

$\mathcal{C}' \leftarrow \mathcal{C}$;

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

retourner $\widehat{v}_{\downarrow \mathcal{X}}$

Description de l'algorithme

Algorithme

si $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **«Incohérent» alors retourner «Incohérent»**

$\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$; **Introduction des variables** $\mathbf{y_i}$

$\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;

$\mathcal{C}' \leftarrow \mathcal{C}$;

pour $i \leftarrow 1$ **à** n **faire** **Introduction d'une contrainte sur** $\mathbf{y_i}$

$\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\quad \widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$\quad d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$\quad d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

retourner $\widehat{v}_{\downarrow \mathcal{X}}$

Description de l'algorithme

Algorithme

si $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **«Incohérent» alors retourner «Incohérent»**

$\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$; **Introduction des variables $\mathbf{y_i}$**

$\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;

$\mathcal{C}' \leftarrow \mathcal{C}$;

pour $i \leftarrow 1$ **à** n **faire** **Introduction d'une contrainte sur $\mathbf{y_i}$**

$\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\quad \widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$; **Calcul de $\mathbf{y_i}$**

$\quad d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$\quad d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

retourner $\widehat{v}_{\downarrow \mathcal{X}}$

Description de l'algorithme

Algorithme

si $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$**«Incohérent» alors retourner «Incohérent»**

$\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$; **Introduction des variables $\mathbf{y_i}$**

$\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;

$\mathcal{C}' \leftarrow \mathcal{C}$;

pour $i \leftarrow 1$ **à** n **faire** **Introduction d'une contrainte sur $\mathbf{y_i}$**

$\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\quad \widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$; **Calcul de $\mathbf{y_i}$**

$\quad d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$; **Mise à jour des domaines**

$\quad d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

retourner $\widehat{v}_{\downarrow \mathcal{X}}$

Déroulement de l'algorithme sur un exemple

Un problème de partage d'objets

- Problème d'allocation à 3 agents et 3 objets.
- Contraintes :
  1. un agent a droit à un et un seul objet,
  2. un objet ne doit pas être attribué à plus d'un agent.
- Une utilité est associée à chaque couple $(agent, objet)$, selon le tableau :

objets \ agents	a_1	a_2	a_3
o_1	3	3	3
o_2	5	9	7
o_3	7	8	1

Déroulement de l'algorithme sur un exemple

Algorithme :

$\mathcal{X}' \leftarrow \mathcal{X} \cup \{y_1, \ldots, y_n\}$;
$\mathcal{D}' \leftarrow \mathcal{D} \cup \{[\![m, M]\!], \ldots, [\![m, M]\!]\}$;
$\mathcal{C}' \leftarrow \mathcal{C}$;

Réseau initial :

$\mathcal{X}' = \{a_1, a_2, a_3, u_1, u_2, u_3\} \cup \{\mathbf{y_1}, \mathbf{y_2}, \mathbf{y_3}\}$

$\mathcal{D}' = \{\{o_1, o_2, o_3\}, \{o_1, o_2, o_3\}, \{o_1, o_2, o_3\}, [\![3, 7]\!], [\![3, 9]\!], [\![1, 7]\!]\}$
$\cup \{[\![\mathbf{1}, \mathbf{9}]\!], [\![\mathbf{1}, \mathbf{9}]\!], [\![\mathbf{1}, \mathbf{9}]\!]\}$

$\mathcal{C}' = \{\mathbf{alldifferent}(\{a_1, a_2, a_3\})\}$

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	{1,. . .,9}	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	{1,. . .,9}	{1,. . .,9}	{1,. . .,9}
o_2	o_1	o_3	5	3	1	{1,. . .,9}	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	{1,. . .,9}	{1,. . .,9}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	{1,. . .,9}	{1,. . .,9}	{1,. . .,9}
o_3	o_2	o_1	7	9	3	{1,. . .,9}	{1,. . .,9}	{1,. . .,9}

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	**1**	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	**{1,2,3}**	{1,. . .,9}	{1,. . .,9}
o_2	o_1	o_3	5	3	1	**1**	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	**{1,2,3}**	{1,. . .,9}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	**{1,2,3}**	{1,. . .,9}	{1,. . .,9}
o_3	o_2	o_1	7	9	3	**{1,2,3}**	{1,. . .,9}	{1,. . .,9}

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
$\mathbf{o_1}$	$\mathbf{o_3}$	$\mathbf{o_2}$	**3**	**8**	**7**	{1,2,**3**}	**{1,. . .,9}**	**{1,. . .,9}**
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
$\mathbf{o_2}$	$\mathbf{o_3}$	$\mathbf{o_1}$	**5**	**8**	**3**	{1,2,**3**}	**{1,. . .,9}**	**{1,. . .,9}**
$\mathbf{o_3}$	$\mathbf{o_1}$	$\mathbf{o_2}$	**7**	**3**	**7**	{1,2,**3**}	**{1,. . .,9}**	**{1,. . .,9}**
$\mathbf{o_3}$	$\mathbf{o_2}$	$\mathbf{o_1}$	**7**	**9**	**3**	{1,2,**3**}	**{1,. . .,9}**	**{1,. . .,9}**

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	**3**	{**3**,. . .,9}	{1,. . .,9}
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	**3**	{**3**,. . .,9}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	**3**	{**3**,. . .,9}	{1,. . .,9}
o_3	o_2	o_1	7	9	3	**3**	{**3**,. . .,9}	{1,. . .,9}

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	3	**{3,. . .,7}**	{1,. . .,9}
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	3	**{3,. . .,5}**	{1,. . .,9}
o_3	o_1	o_2	7	3	7	3	**{3,. . .,7}**	{1,. . .,9}
o_3	o_2	o_1	7	9	3	3	**{3,. . .,7}**	{1,. . .,9}

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	**o_3**	**o_2**	**3**	**8**	**7**	**3**	{3,. . .,**7**}	**{1,. . .,9}**
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	3	{3,. . .,5}	{1,. . .,9}
o_3	**o_1**	**o_2**	**7**	**3**	**7**	**3**	{3,. . .,**7**}	**{1,. . .,9}**
o_3	**o_2**	**o_1**	**7**	**9**	**3**	**3**	{3,. . .,**7**}	**{1,. . .,9}**

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	3	**7**	**{7,8,9}**
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	3	{3,. . .,5}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	3	**7**	**{7,8,9}**
o_3	o_2	o_1	7	9	3	3	**7**	**{7,8,9}**

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	3	7	**{7,8}**
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	3	{3,. . .,5}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	3	7	**7**
o_3	o_2	o_1	7	9	3	3	7	{7,8,9}

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	3	7	{7,8}
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	3	{3,. . .,5}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	3	7	7
$\mathbf{o_3}$	$\mathbf{o_2}$	$\mathbf{o_1}$	**7**	**9**	**3**	**3**	**7**	{7,8,**9**}

Déroulement de l'algorithme sur un exemple

Algorithme :

pour $i \leftarrow 1$ **à** n **faire**

$\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;

$\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;

$d_{y_{i+1}} \leftarrow [\![\widehat{v}(y_i), M]\!]$;

fin

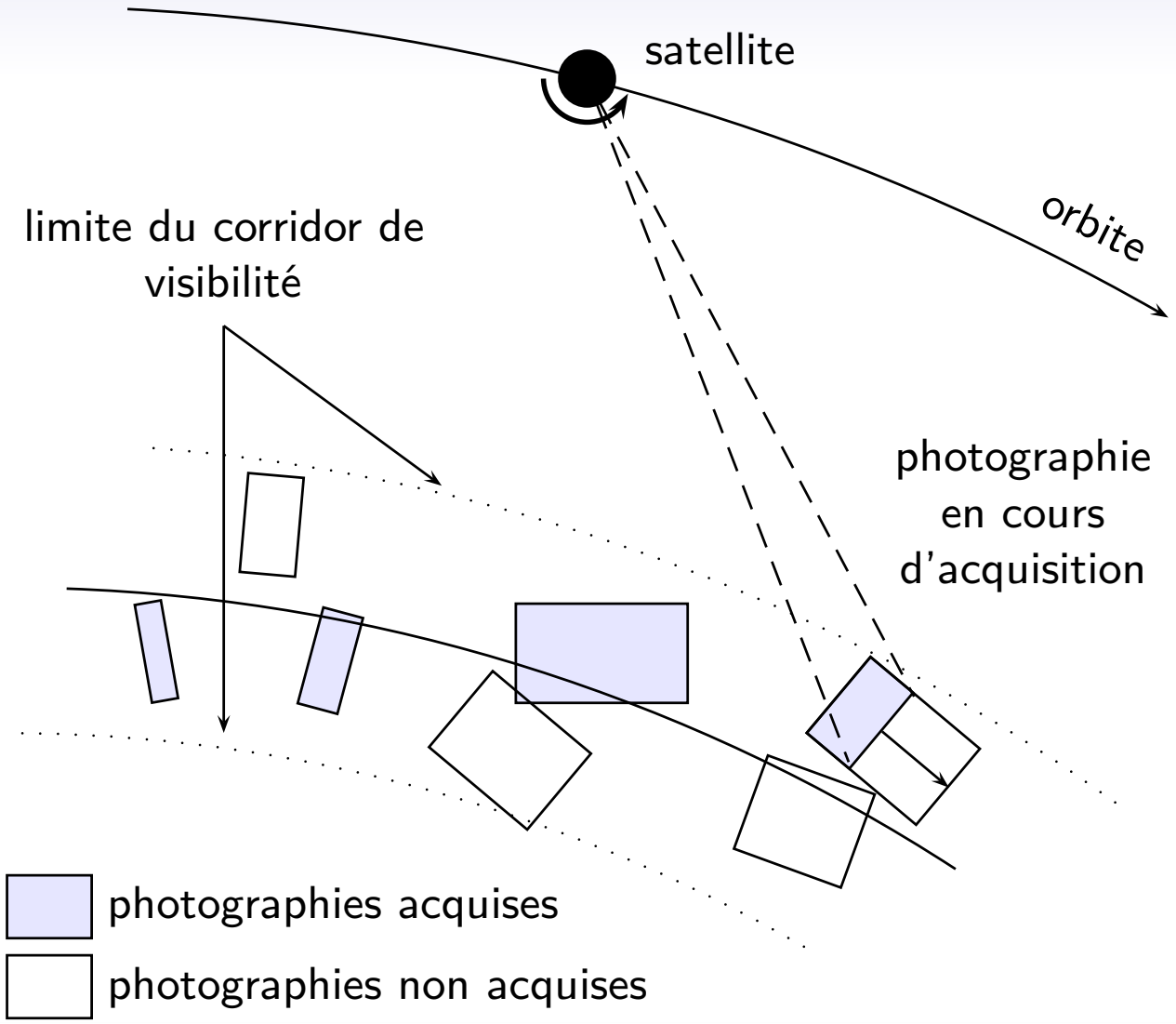
a_1	a_2	a_3	u_1	u_2	u_3	y_1	y_2	y_3
o_1	o_2	o_3	3	9	1	1	{1,. . .,9}	{1,. . .,9}
o_1	o_3	o_2	3	8	7	3	7	{7,8}
o_2	o_1	o_3	5	3	1	1	{1,. . .,9}	{1,. . .,9}
o_2	o_3	o_1	5	8	3	3	{3,. . .,5}	{1,. . .,9}
o_3	o_1	o_2	7	3	7	3	7	7
o_3	o_2	o_1	7	9	3	3	7	**9**

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

Une constellation de satellites...

- Une constellation de satellites d'observation de la Terre co-financée par plusieurs pays (en raison de son coût).

Une constellation de satellites...



Une constellation de satellites...

- Chaque agent (agences civiles et militaires de chaque pays — entre 3 et 6) envoie des demandes d'images à prendre, simples ou complexes (stéréo, tri-stéréo...) — plusieurs centaines.
- Chaque jour, le Centre de Programmation sélectionne les demandes qui seront satisfaites le lendemain et allouées aux agents.
- Contraintes : contraintes physiques (fenêtres temporelles, temps de transition, images particulières — stéréo, ...—, mémoire, énergie,...).
- L'exploitation doit être :
  - **efficace** $\rightsquigarrow$ la constellation ne doit pas être sous-exploitée,
  - **équitable** $\rightsquigarrow$ chaque agent attend un « retour sur investissement » en rapport avec sa contribution financière.

Simplification du problème

Données du problème simplifié :

- un ensemble d'agents $\mathcal{A}$;
- un ensemble d'objets $\mathcal{O}$ (ensemble des images demandées) ;
- préférences des agents exprimées comme un ensemble de poids w_{io} (préférences additives) ;
- des contraintes :
  - contraintes physiques approximées par des contraintes de volume généralisé,
  - droits inégaux des agents approximés par des contraintes de consommation.

Simplification du problème

Énoncé du problème :
Trouver une affectation des objets aux agents satisfaisant toutes les contraintes et dont le profil d'utilités est non dominé au sens de l'ordre leximin.

Simplification du problème

Énoncé du problème :
Trouver une affectation des objets aux agents satisfaisant toutes les contraintes et dont le profil d'utilités est non dominé au sens de l'ordre leximin.

La version problème de décision de ce problème est **NP**-complète.

Implantation du problème simplifié

Générateur d'instances :

Un générateur d'instances aléatoires paramétrable du problème simplifié a été codé en Java. Il est disponible en ligne :

`http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`

Implantation du problème simplifié

Générateur d'instances :
Un générateur d'instances aléatoires paramétrable du problème simplifié a été codé en Java. Il est disponible en ligne :

`http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`

Implantations :
Deux implantations de l'algorithme ont été testées :

- en Java avec CHOCO ;
- en Java avec CPLEX.

Implantation du problème simplifié

Générateur d'instances :
Un générateur d'instances aléatoires paramétrable du problème simplifié a été codé en Java. Il est disponible en ligne :

`http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`

Implantations :
Deux implantations de l'algorithme ont été testées :

- en Java avec CHOCO ;
- en Java avec CPLEX.

La méta-contrainte **AtLeast** peut être transformée en contraintes linéaires : $\mathbf{AtLeast}(\{x_1 \geq y, \ldots, x_n \geq y\}, k) \Leftrightarrow \{x_1 + \delta_1 \overline{y} \geq y, \ldots, x_n + \delta_n \overline{y} \geq y, \sum_{i=1}^{n} \delta_i \leq n - k\}$, avec $\{\delta_1, \ldots, \delta_n\}$ variables 0–1.

Implantation du problème simplifié

Générateur d'instances :

Un générateur d'instances aléatoires paramétrable du problème simplifié a été codé en Java. Il est disponible en ligne :

`http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`

Implantations :

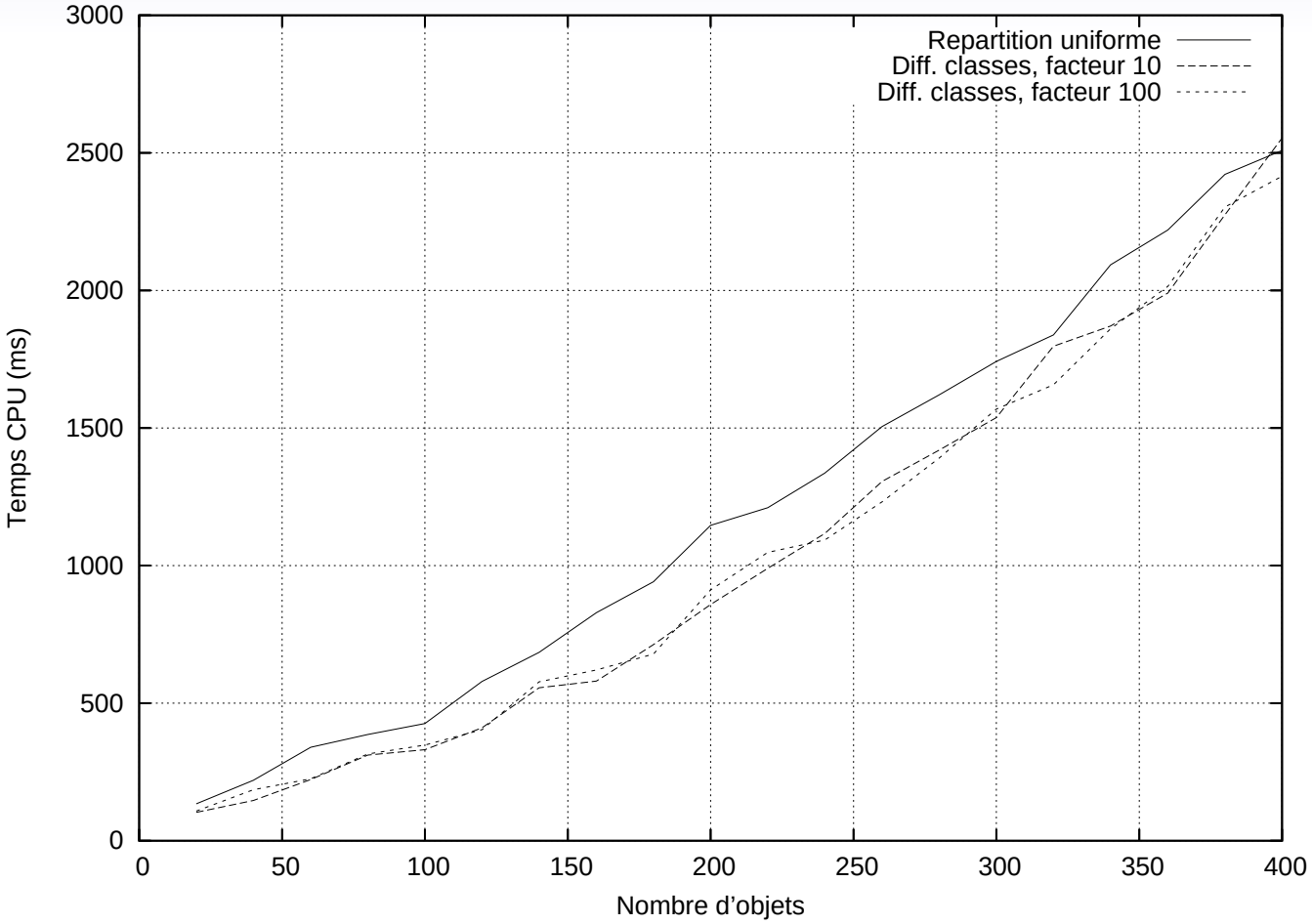
Deux implantations de l'algorithme ont été testées :

- en Java avec CHOCO ;
- en Java avec CPLEX.

Instance moyenne :

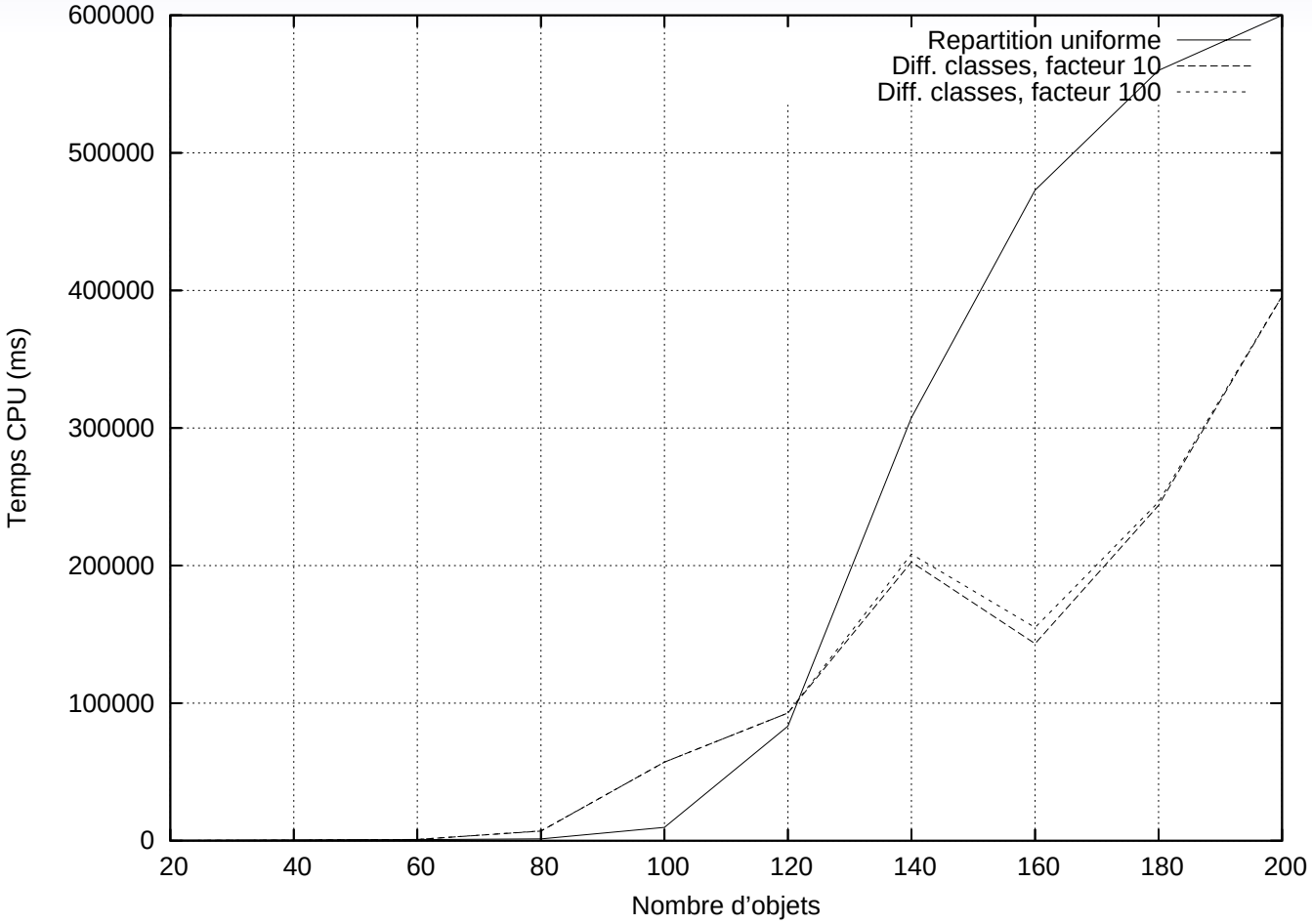
4 agents, droits inégaux, 150 objets, contraintes de volume autorisant 10 objets sur 20 consécutifs.

Résultats



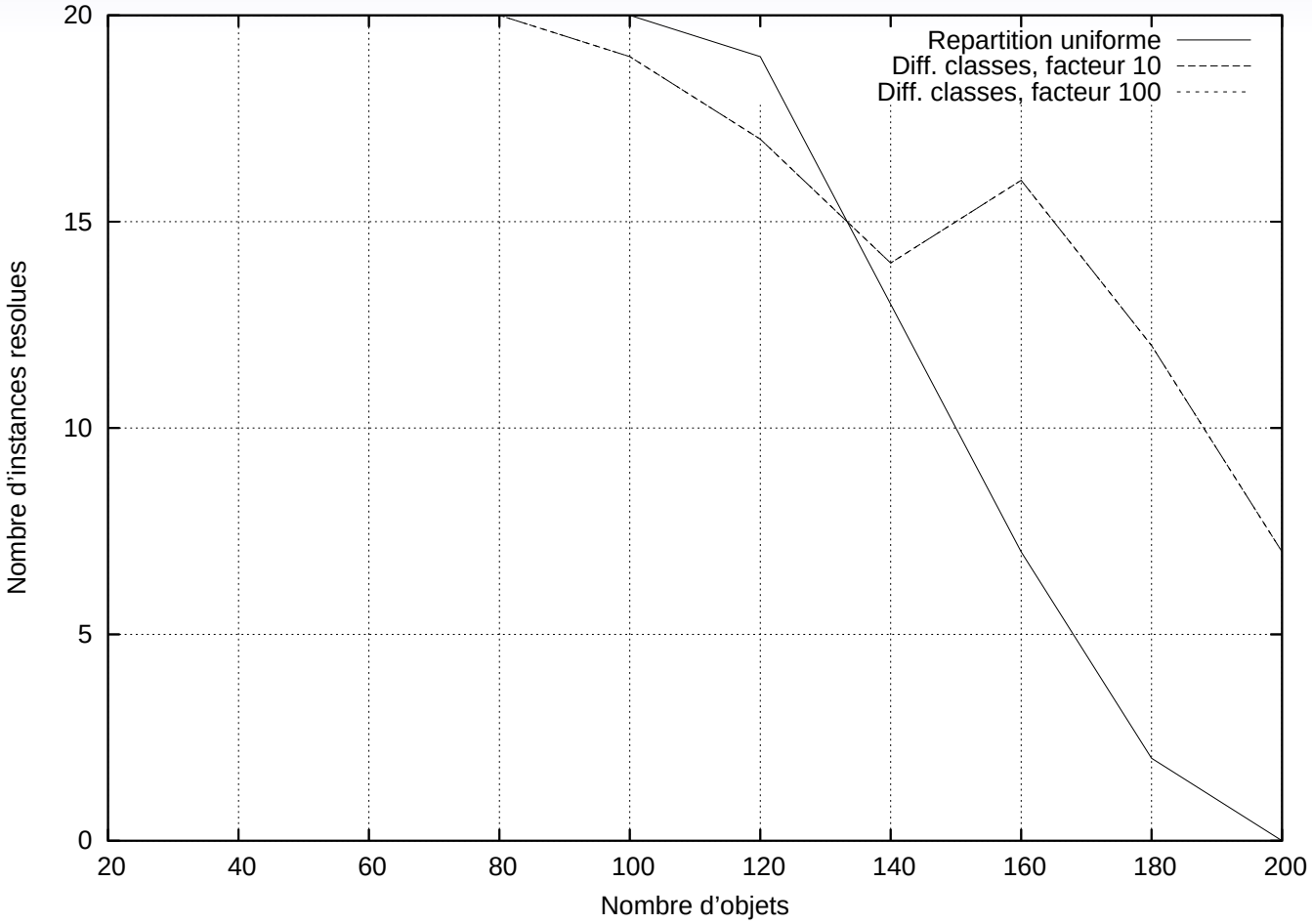
Temps de calcul avec CPLEX.

Résultats



Temps de calcul avec CHOCO.

Résultats



Nombre d'instances résolues en moins de 10 minutes avec CHOCO.

Remarques sur les résultats

- Notre implantation fondée sur CPLEX est plus efficace que celle fondée sur CHOCO sur toutes les instances.

Remarques sur les résultats

- Notre implantation fondée sur CPLEX est plus efficace que celle fondée sur CHOCO sur toutes les instances.
- Cependant, CPLEX peut commettre des erreurs (d'approximation).

Remarques sur les résultats

- Notre implantation fondée sur CPLEX est plus efficace que celle fondée sur CHOCO sur toutes les instances.
- Cependant, CPLEX peut commettre des erreurs (d'approximation).
- Les instances à répartition de poids uniformes sont plus difficiles à résoudre en moyenne.

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

Conclusions générales

- **Problème traité :** calcul d'une instantiation leximin-optimale d'un réseau de contraintes.

Conclusions générales

- **Problème traité :** calcul d'une instantiation leximin-optimale d'un réseau de contraintes.
- **Motivation :** l'ordre leximin garantit certaines propriétés d'équité et d'efficacité pour les problèmes de décision collective.

Conclusions générales

- **Problème traité :** calcul d'une instantiation leximin-optimale d'un réseau de contraintes.
- **Motivation :** l'ordre leximin garantit certaines propriétés d'équité et d'efficacité pour les problèmes de décision collective.
- **Algorithme :** introduction d'un algorithme de calcul fondé sur la programmation par contraintes.

Conclusions générales

- **Problème traité :** calcul d'une instantiation leximin-optimale d'un réseau de contraintes.
- **Motivation :** l'ordre leximin garantit certaines propriétés d'équité et d'efficacité pour les problèmes de décision collective.
- **Algorithme :** introduction d'un algorithme de calcul fondé sur la programmation par contraintes.
- **Implantation :** implantation et évaluation de l'algorithme en Java avec CHOCO et CPLEX.

Conclusions générales

- **Problème traité :** calcul d'une instantiation leximin-optimale d'un réseau de contraintes.
- **Motivation :** l'ordre leximin garantit certaines propriétés d'équité et d'efficacité pour les problèmes de décision collective.
- **Algorithme :** introduction d'un algorithme de calcul fondé sur la programmation par contraintes.
- **Implantation :** implantation et évaluation de l'algorithme en Java avec CHOCO et CPLEX.
- **Application :** problème d'allocation pour le partage de ressources satellitaires.

Suite de l'étude

- Autres méthodes de calcul du leximin-optimal :
  - Utilisation d'un *branch-and-bound* avec une contrainte du type *Multiset Ordering* [FHK+03].
    Implantation déjà réalisée, et résultats intéressants.
  - Utilisation d'une contrainte de « tri » de variables [BGC97].
    Implantation déjà réalisée, tests en cours.
  - Utilisation d'algorithmes issus des CSP flous [DF99].
- Autres approches de l'équité (familles de fonctions d'utilité collective paramétrées).
- Application de l'algorithme à d'autres problèmes.

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.

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

A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh.
Multiset ordering constraints.
In *Proc. of IJCAI'03*, Acapulco, Mexico, august 2003.

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

U. Montanari.

Networks of constraints : Fundamental properties and applications to picture processing.
Information Sciences, 7 :95–132, 1974.

H. Moulin.
Axioms of Cooperative Decision Making.
Cambridge University Press, 1988.

Merci de votre attention

Transparents (prochainement) disponibles à l'URL :
http://www.cert.fr/dcsd/THESES/sbouveret/francais/ressources/JFPC06/pres_jfpc06.pdf
ou par mail :
sylvain.bouveret@cert.fr
michel.lemaitre@cert.fr

Générateur de problèmes disponible à l'URL :
http://www.cert.fr/dcsd/THESES/sbouveret/benchmark