Algorithmes 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

Séminaire de l'UR Conduite et décision (DCSD),
27 juin 2006

Quelques problèmes issus du monde réel

- Problème d'emploi du temps pour les gardes des infirmières dans les services d'urgences (*Nurse rostering problem*).
- Problème d'emploi du temps pour le service des pompiers.
- Problème d'affectation de créneaux à des compagnies aériennes dans des aéroports.
- Problèmes d'allocation de ressources multi-agents : partage d'une constellation de satellites d'observation de la Terre.

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.
- Ils font intervenir des variables à domaines généralement finis.

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.
- Ils font intervenir des variables à domaines généralement finis.
- Ils font intervenir un certain nombre de contraintes entre ces variables.

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.
- Ils font intervenir des variables à domaines généralement finis.
- Ils font intervenir un certain nombre de contraintes entre ces variables.
- $\rightsquigarrow$ **ceci suggère une modélisation sous forme de problème de satisfaction de contraintes.**

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.
- Ils font intervenir des variables à domaines généralement finis.
- Ils font intervenir un certain nombre de contraintes entre ces variables.
- $\rightsquigarrow$ **ceci suggère une modélisation sous forme de problème de satisfaction de contraintes.**
- En général ils font intervenir de manière plus ou moins explicite la notion **d'équité**.

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.
- Ils font intervenir des variables à domaines généralement finis.
- Ils font intervenir un certain nombre de contraintes entre ces variables.
- $\rightsquigarrow$ **ceci suggère une modélisation sous forme de problème de satisfaction de contraintes.**
- En général ils font intervenir de manière plus ou moins explicite la notion **d'équité**.

Quelques points communs entre ces problèmes

- Ils peuvent être fortement combinatoires $\rightsquigarrow$ difficiles à résoudre.
- Ils font intervenir des variables à domaines généralement finis.
- Ils font intervenir un certain nombre de contraintes entre ces variables.
- $\rightsquigarrow$ **ceci suggère une modélisation sous forme de problème de satisfaction de contraintes.**
- En général ils font intervenir de manière plus ou moins explicite la notion **d'équité**.

Problème : Comment traiter la recherche de compromis équitable et efficace dans le cadre des problèmes de satisfaction de contraintes ?

Plan de l'exposé du jour

1. Formalisation du problème
   - CSP et programmation par contraintes
   - L'équité dans les problèmes de décision collective
   - Leximin et programmation par contraintes
2. Différentes approches de résolution du problème
   - Une méthode de branch-and-bound
   - Trier pour régner
   - Utiliser une contrainte de cardinalité
   - Utiliser les sous-ensembles saturés
3. Application, benchmark et résultats obtenus
   - Description de l'application
   - Le problème simplifié
   - Benchmark et résultats
4. Conclusion

Plan de l'exposé du jour

Le problème de satisfaction de contraintes

Réseau de contraintes [Montanari, 1974]

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 cadre de la programmation par contraintes

La programmation par contraintes est un cadre de résolution pour les problèmes de satisfaction de contraintes fondé sur :

- un algorithme d'exploration de l'arbre de recherche (paramétrable par des heuristiques),
- des mécanismes de propagation de contraintes réagissant à des évènements sur les variables.

Problème posé

Le cadre de la programmation par contraintes et des CSP fournit des outils de modélisation et résolution de problèmes combinatoires avec ou sans critère d'optimisation.

Problème posé

Le cadre de la programmation par contraintes et des CSP fournit des outils de modélisation et résolution de problèmes combinatoires avec ou sans critère d'optimisation.

Problème : Comment introduire une «contrainte» d'équité (qui exprime le fait que des compromis sont souhaitables entre la satisfaction des différents agents) ?

Décision collective et *welfarism*

Soit un problème pour lequel on doit choisir une alternative parmi un ensemble $\mathcal{S}$ d'alternatives possibles concernant n agents.

Décision collective et *welfarism*

Soit un problème pour lequel on doit choisir une alternative parmi un ensemble $\mathcal{S}$ d'alternatives possibles concernant n agents.
Hypothèse : La décision collective ne dépend que de la donnée, pour chaque agent et pour chaque alternative, d'un niveau de satisfaction : l'utilité individuelle.

Décision collective et *welfarism*

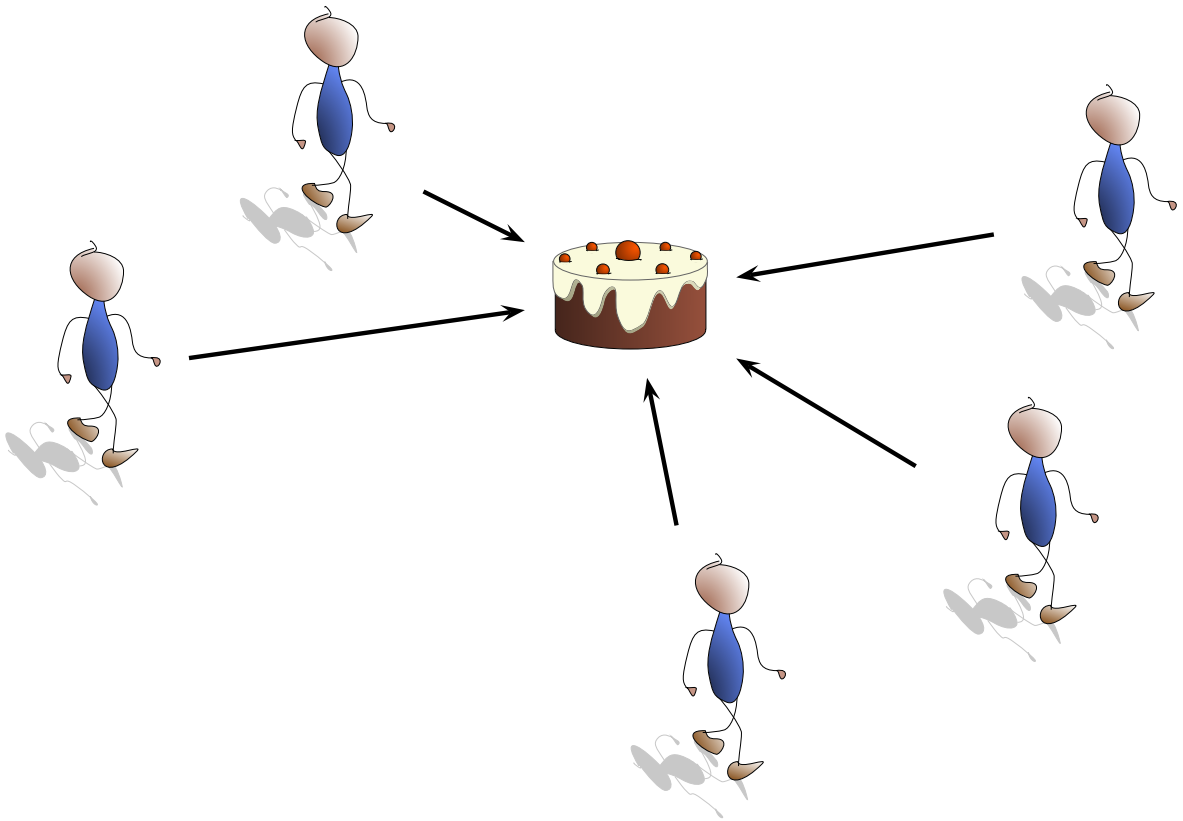
Soit un problème pour lequel on doit choisir une alternative parmi un ensemble $\mathcal{S}$ d'alternatives possibles concernant n agents.
Hypothèse : La décision collective ne dépend que de la donnée, pour chaque agent et pour chaque alternative, d'un niveau de satisfaction : l'utilité individuelle.

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 $u_i : \mathcal{S} \rightarrow \mathbb{N}$.

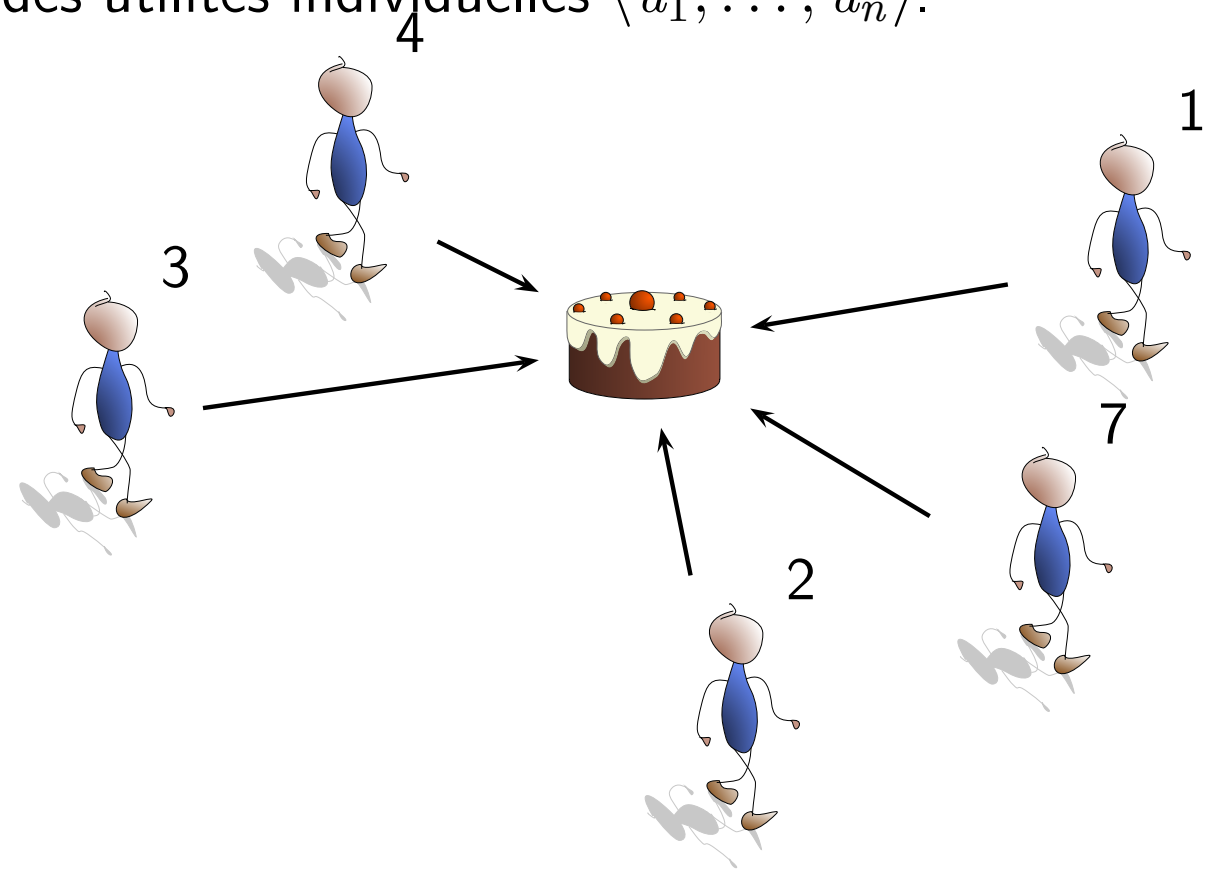
Décision collective et *welfarism*

La théorie du **welfarism** [Moulin, 1988] traite le problème de décision collective en attachant à chaque alternative faisable le vecteur des utilités individuelles $\langle u_1, \ldots, u_n \rangle$.



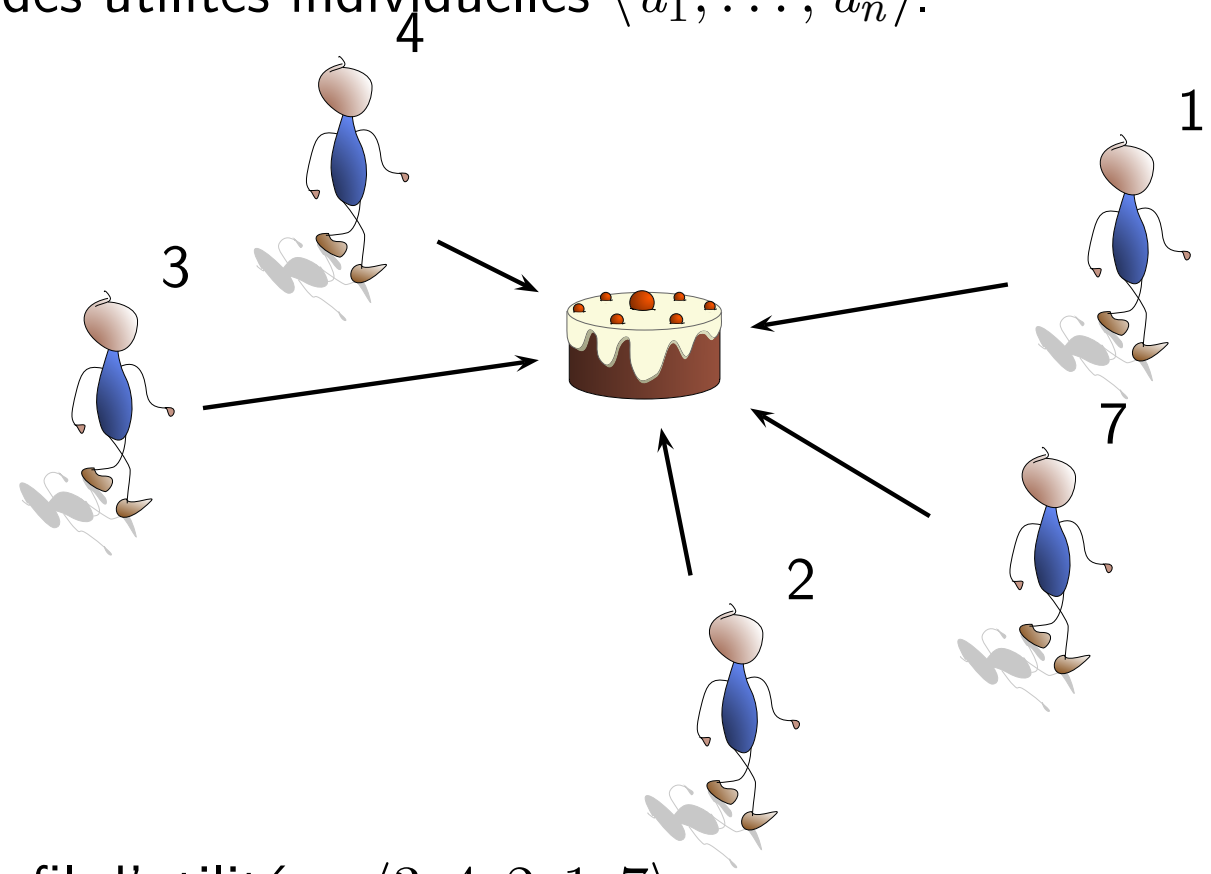
Décision collective et *welfarism*

La théorie du **welfarism** [Moulin, 1988] traite le problème de décision collective en attachant à chaque alternative faisable le vecteur des utilités individuelles $\langle u_1, \ldots, u_n \rangle$.



Décision collective et *welfarism*

La théorie du **welfarism** [Moulin, 1988] traite le problème de décision collective en attachant à chaque alternative faisable le vecteur des utilités individuelles $\langle u_1, \ldots, u_n \rangle$.



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 garantir l'équité et la Pareto-efficacité.

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour garantir l'équité et la Pareto-efficacité.
Problème : Aucune fonction d'agrégation numérique ne peut représenter l'ordre leximin, sauf si l'ensemble des profils d'utilité est fini, auquel cas on peut utiliser les fonctions suivantes :

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour garantir l'équité et la Pareto-efficacité.
Problème : Aucune fonction d'agrégation numérique ne peut représenter l'ordre leximin, sauf si l'ensemble des profils d'utilité est fini, auquel cas on peut utiliser les fonctions suivantes :

- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} n^{u_i}$,

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour garantir l'équité et la Pareto-efficacité.
Problème : Aucune fonction d'agrégation numérique ne peut représenter l'ordre leximin, sauf si l'ensemble des profils d'utilité est fini, auquel cas on peut utiliser les fonctions suivantes :

- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} n^{u_i}$,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} u_i^{-q}$, avec q très grand,

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour garantir l'équité et la Pareto-efficacité.
Problème : Aucune fonction d'agrégation numérique ne peut représenter l'ordre leximin, sauf si l'ensemble des profils d'utilité est fini, auquel cas on peut utiliser les fonctions suivantes :

- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} n^{u_i}$,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} u_i^{-q}$, avec q très grand,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} w_i \times u_i^{\uparrow}$, avec $w_1 >> w_2 >> \cdots >> w_n$ (OWA).

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour garantir l'équité et la Pareto-efficacité.
Problème : Aucune fonction d'agrégation numérique ne peut représenter l'ordre leximin, sauf si l'ensemble des profils d'utilité est fini, auquel cas on peut utiliser les fonctions suivantes :

- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} n^{u_i}$,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} u_i^{-q}$, avec q très grand,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} w_i \times u_i^{\uparrow}$, avec $w_1 >> w_2 >> \cdots >> w_n$ (OWA).

Énoncé du problème

L'ordre social leximin semble donc présenter des propriétés interessantes pour garantir l'équité et la Pareto-efficacité.
Problème : Aucune fonction d'agrégation numérique ne peut représenter l'ordre leximin, sauf si l'ensemble des profils d'utilité est fini, auquel cas on peut utiliser les fonctions suivantes :

- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} n^{u_i}$,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} u_i^{-q}$, avec q très grand,
- $\overrightarrow{u} \mapsto \sum_{i=1}^{n} w_i \times u_i^{\uparrow}$, avec $w_1 >> w_2 >> \cdots >> w_n$ (OWA).

$\rightsquigarrow$ Ceci nous dissuade donc d'exprimer notre problème comme un CSP avec variable objectif.

Problème de satisfaction de contraintes à critère leximin

Leximin-CSP

- **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, $v(\overrightarrow{u}) \preceq_{leximin} \widehat{v}(\overrightarrow{u})$.

Plan de l'exposé du jour

(I) Principe du branch-and-bound

- Algorithme classique de résolution d'un CSP avec variable à maximiser.

(I) Principe du branch-and-bound

- Algorithme classique de résolution d'un CSP avec variable à maximiser.
- Fondé sur l'amélioration successive des solutions trouvées (fournissant une borne inférieure), et sur l'élagage des branches de l'arbre de recherche non optimales.

(I) Principe du branch-and-bound

- Algorithme classique de résolution d'un CSP avec variable à maximiser.
- Fondé sur l'amélioration successive des solutions trouvées (fournissant une borne inférieure), et sur l'élagage des branches de l'arbre de recherche non optimales.

(I) Principe du branch-and-bound

- Algorithme classique de résolution d'un CSP avec variable à maximiser.
- Fondé sur l'amélioration successive des solutions trouvées (fournissant une borne inférieure), et sur l'élagage des branches de l'arbre de recherche non optimales.

Idée : exploiter le principe du branch and bound adapté à un autre préordre que l'ordre classique sur les entiers : le préordre leximin.

(I) Adaptation au leximin-CSP

On utilise une contrainte **Leximin** :

Contrainte Leximin

Soient $\overrightarrow{x}$ un vecteur de variables et λ un vecteur d'entiers de même taille. La contrainte **Leximin** porte sur toutes les variables de $\overrightarrow{x}$ et est satisfaite ssi $\lambda \prec_{leximin} \overrightarrow{x}$.

Utilisation des travaux de [Frisch *et al.*, 2003] introduisant un algorithme de filtrage par GAC en $O(n \log(n))$ pour la contrainte **Multiset Ordering**.

(I) L'algorithme de leximin-branch-and-bound

Algorithme

while $sol(\mathcal{X}, \mathcal{D}, \mathcal{C}) \neq \emptyset$ **do**

$\widehat{v} \leftarrow \mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C})$ $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{Leximin}(\widehat{v}(\overrightarrow{u}), \overrightarrow{u})\}$

end

if $\widehat{v} \neq \mathbf{null}$ **then** **return** $\widehat{v}$ **else** **return** **"Inconsistent"**

(II) Introduction de la version triée du vecteur objectif

Idée : Introduire des variables supplémentaires $\langle y_1, \ldots, y_n \rangle$ représentant la version triée du vecteur objectif, et raisonner par maximisations successives.

(II) Introduction de la version triée du vecteur objectif

Idée : Introduire des variables supplémentaires $\langle y_1, \ldots, y_n \rangle$ représentant la version triée du vecteur objectif, et raisonner par maximisations successives.

1. Maximiser y_1 : $\widehat{y_1}$.

(II) Introduction de la version triée du vecteur objectif

Idée : Introduire des variables supplémentaires $\langle y_1, \ldots, y_n \rangle$ représentant la version triée du vecteur objectif, et raisonner par maximisations successives.

1. Maximiser y_1 : $\widehat{y_1}$.
2. Maximiser y_2 sous la contrainte $y_1 = \widehat{y_1}$: $\widehat{y_2}$.

(II) Introduction de la version triée du vecteur objectif

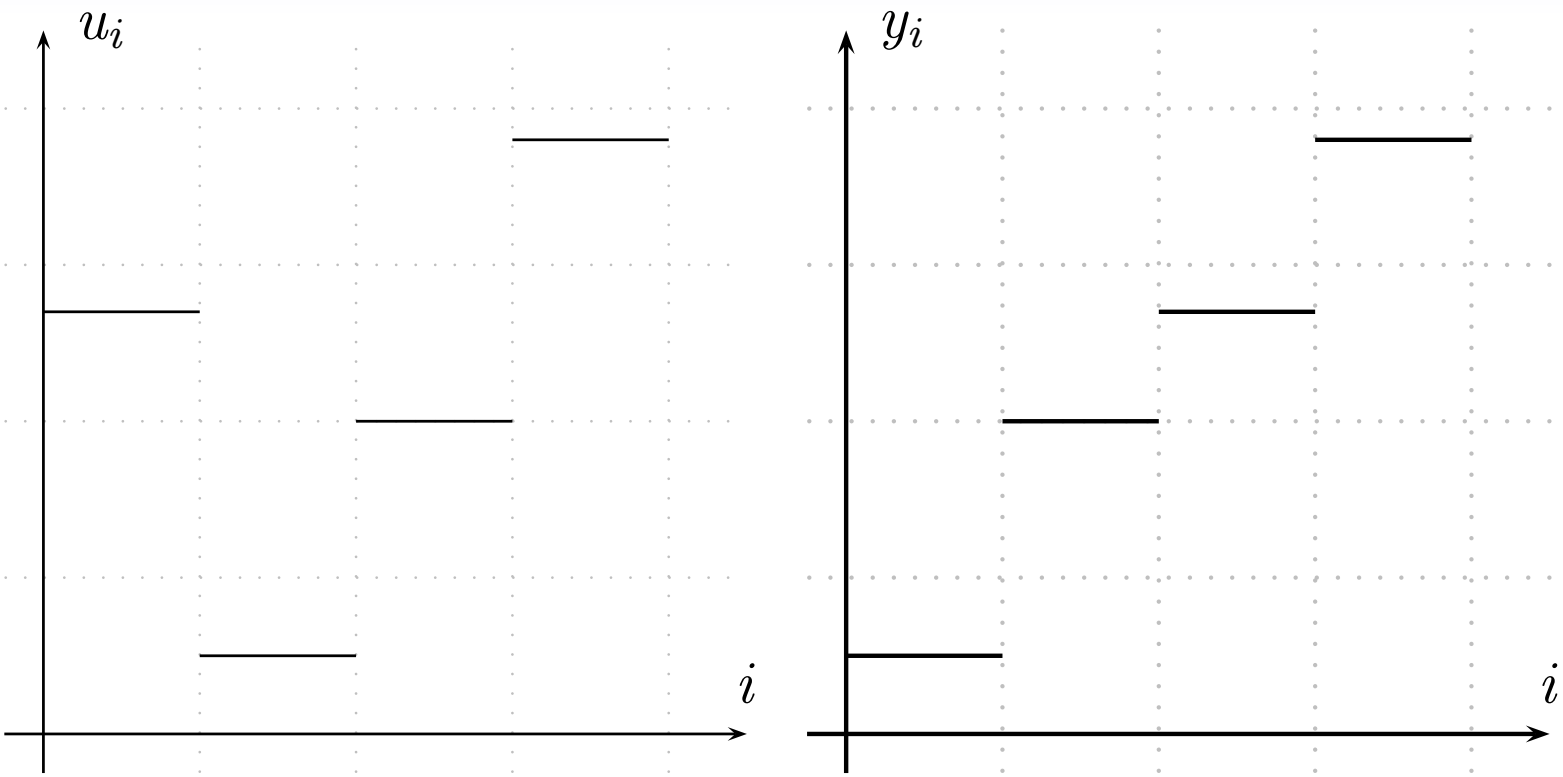
Idée : Introduire des variables supplémentaires $\langle y_1, \ldots, y_n \rangle$ représentant la version triée du vecteur objectif, et raisonner par maximisations successives.

1. Maximiser y_1 : $\widehat{y_1}$.
2. Maximiser y_2 sous la contrainte $y_1 = \widehat{y_1}$: $\widehat{y_2}$.

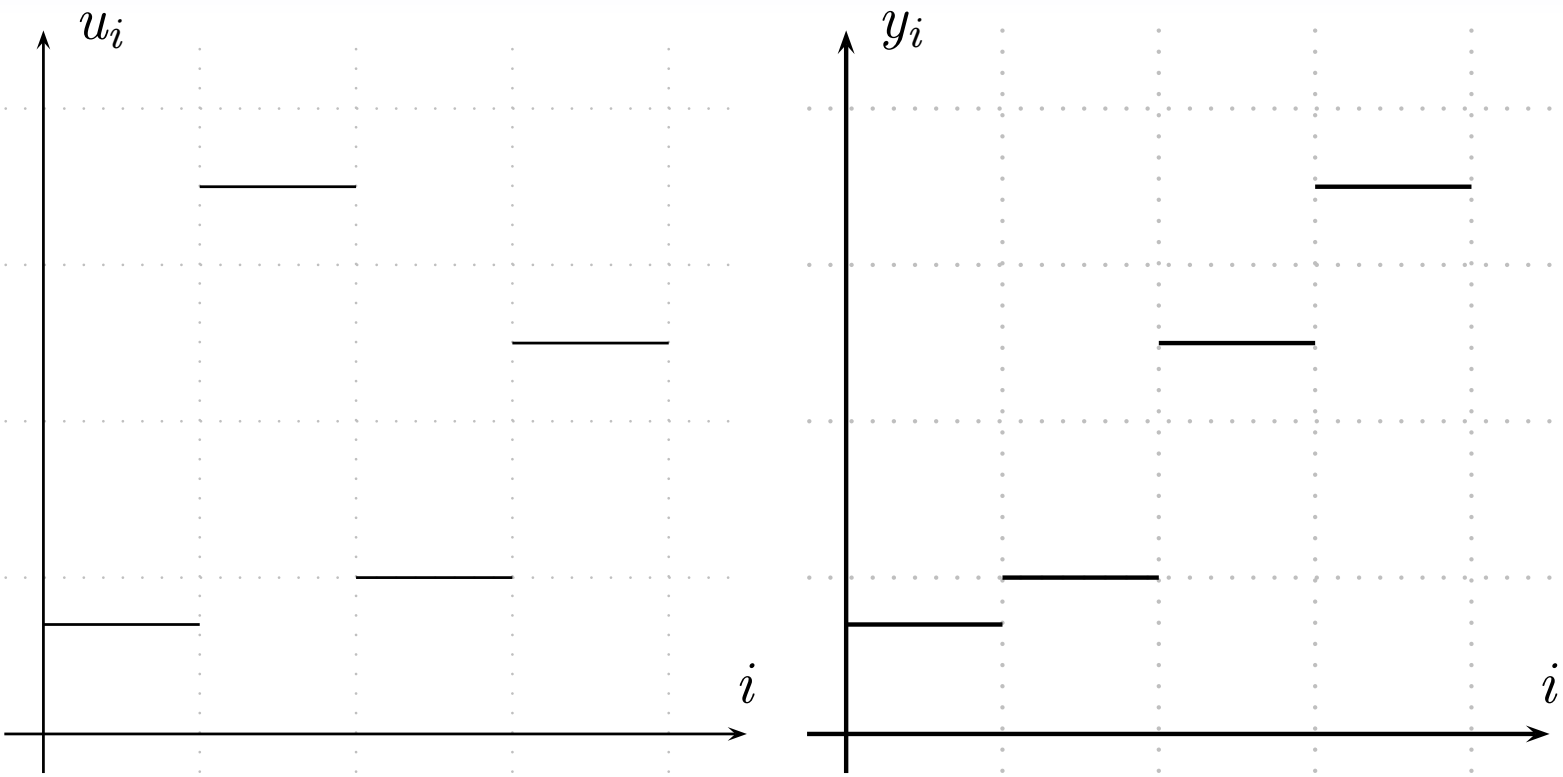
$$\vdots$$

n. Maximiser y_n sous les contraintes $y_1 = \widehat{y_1}$, ..., $y_{n-1} = \widehat{y_{n-1}}$.

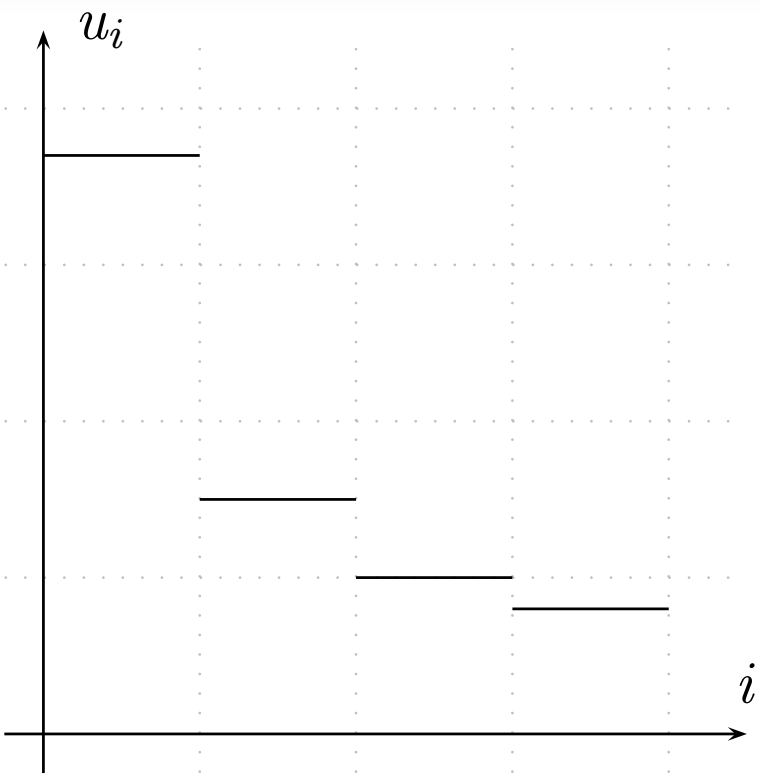
(II) Illustration du principe de l'algorithme

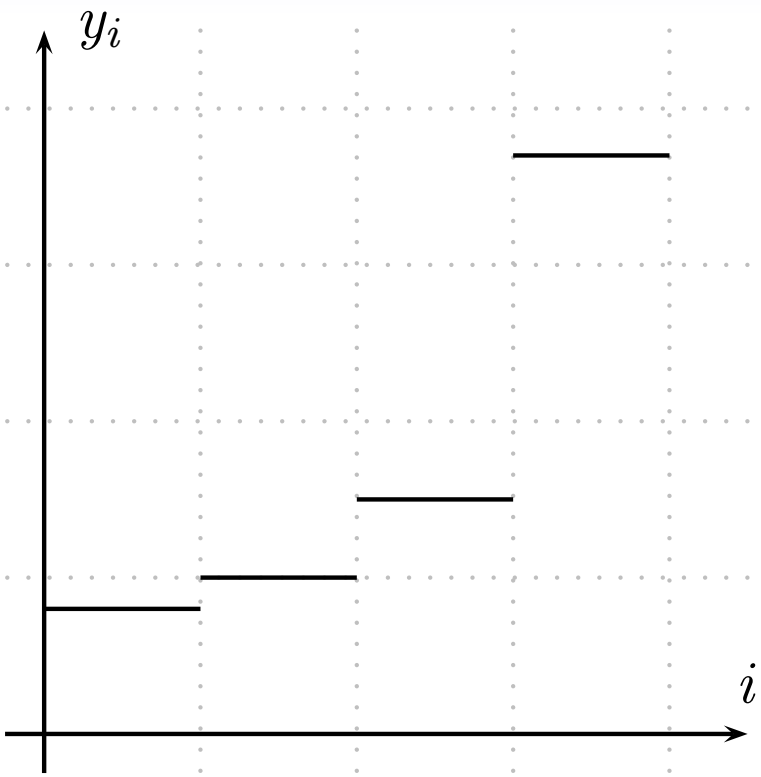


(II) Illustration du principe de l'algorithme

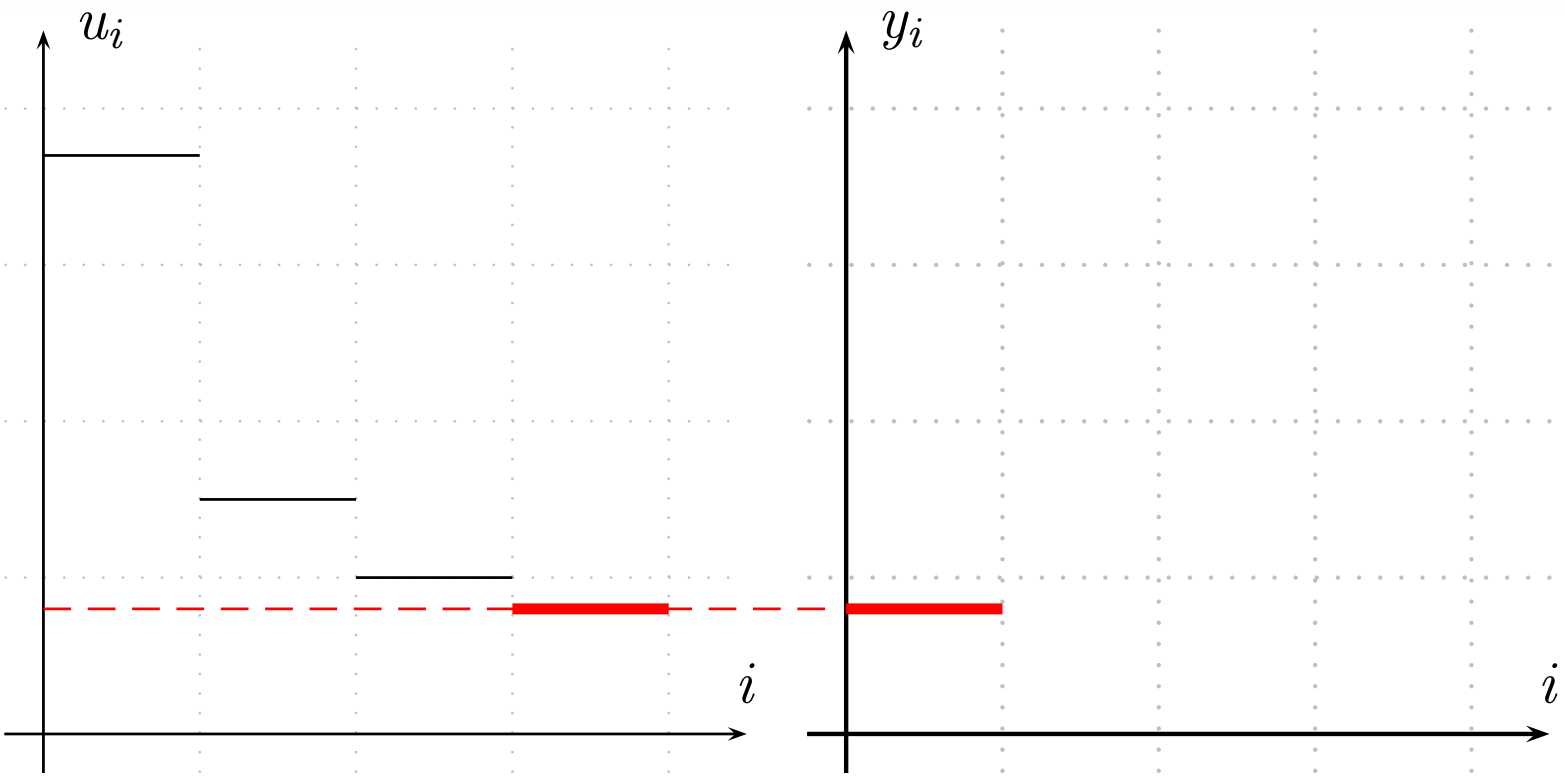


(II) Illustration du principe de l'algorithme

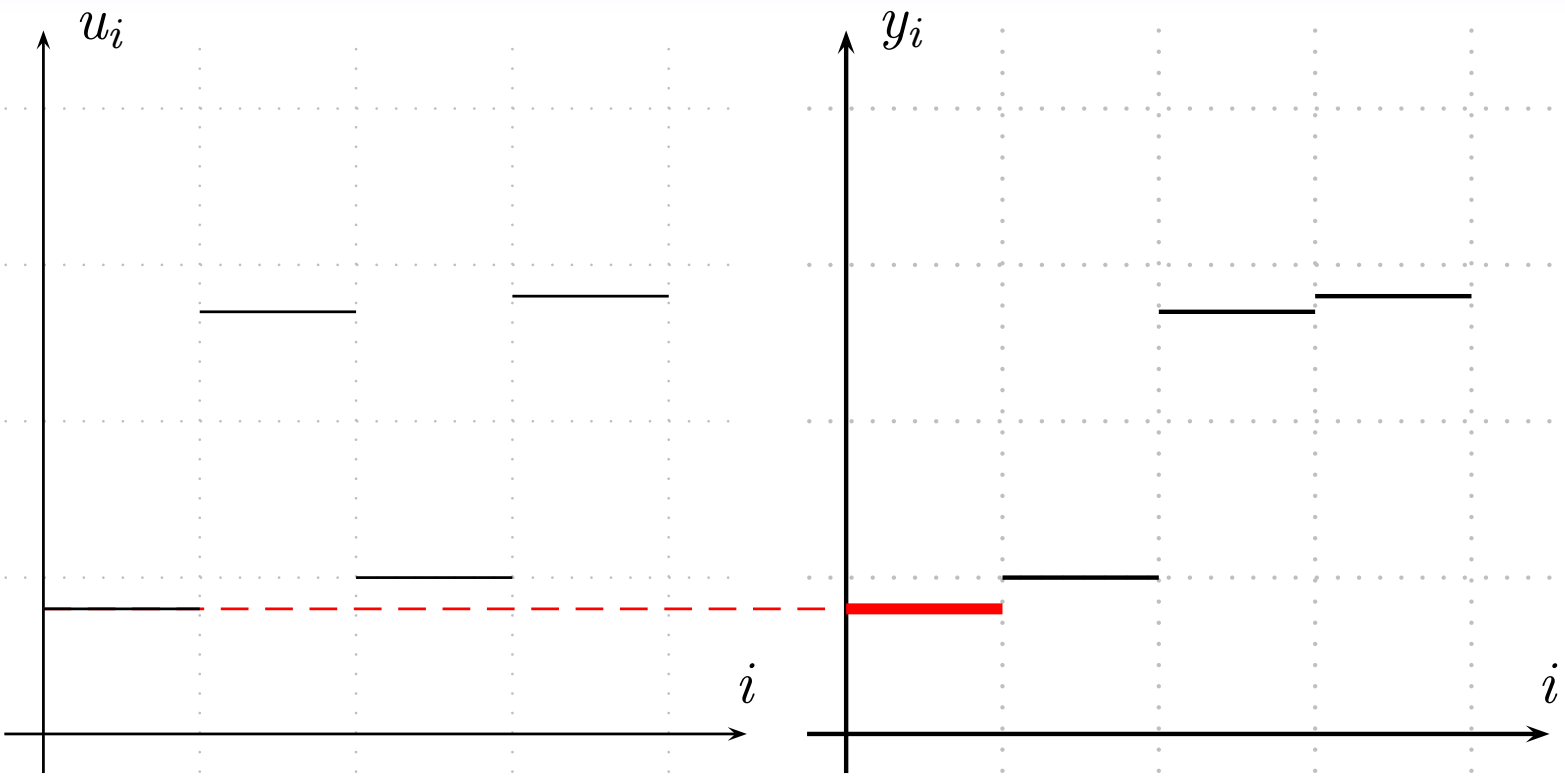




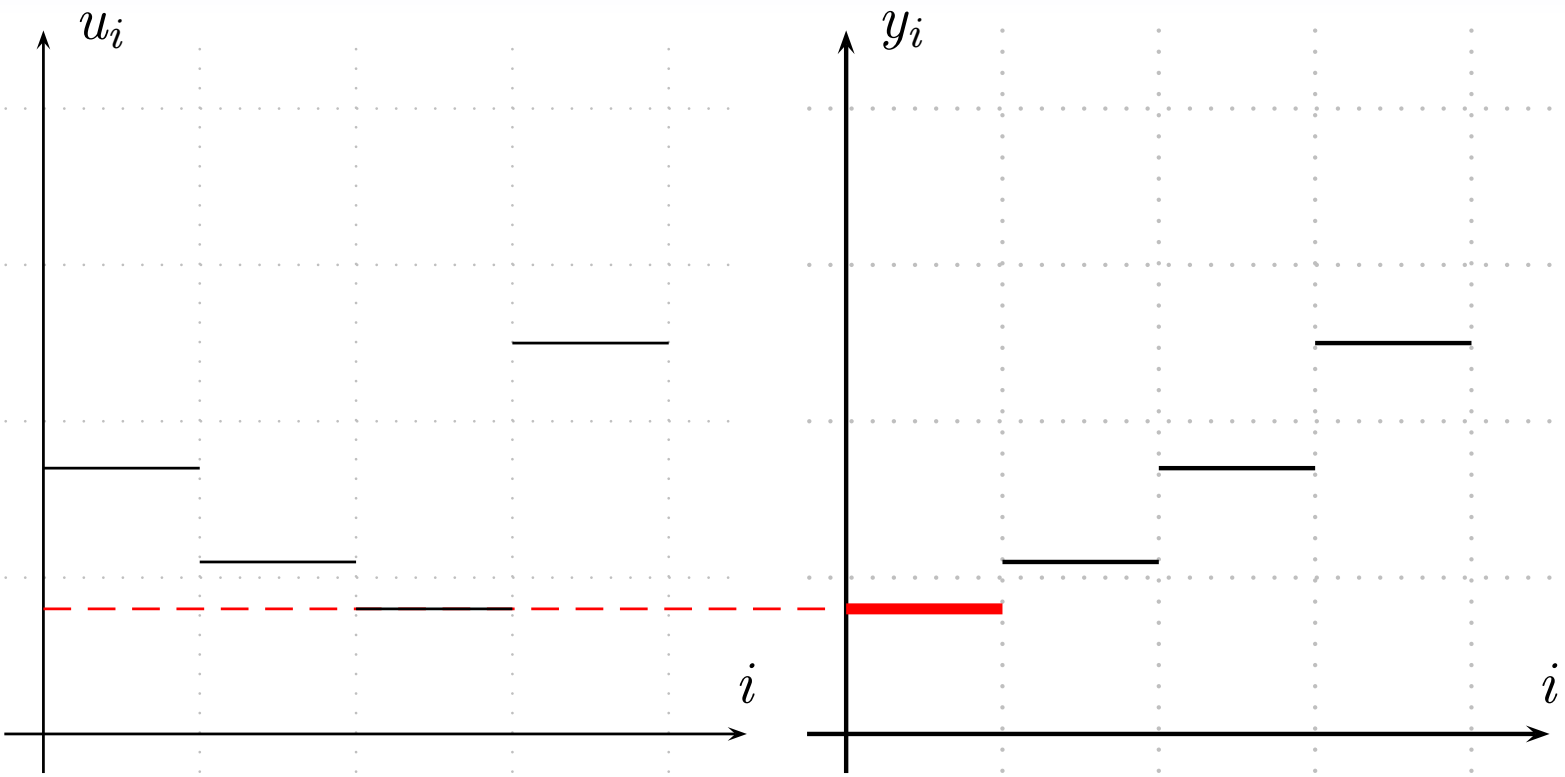
(II) Illustration du principe de l'algorithme



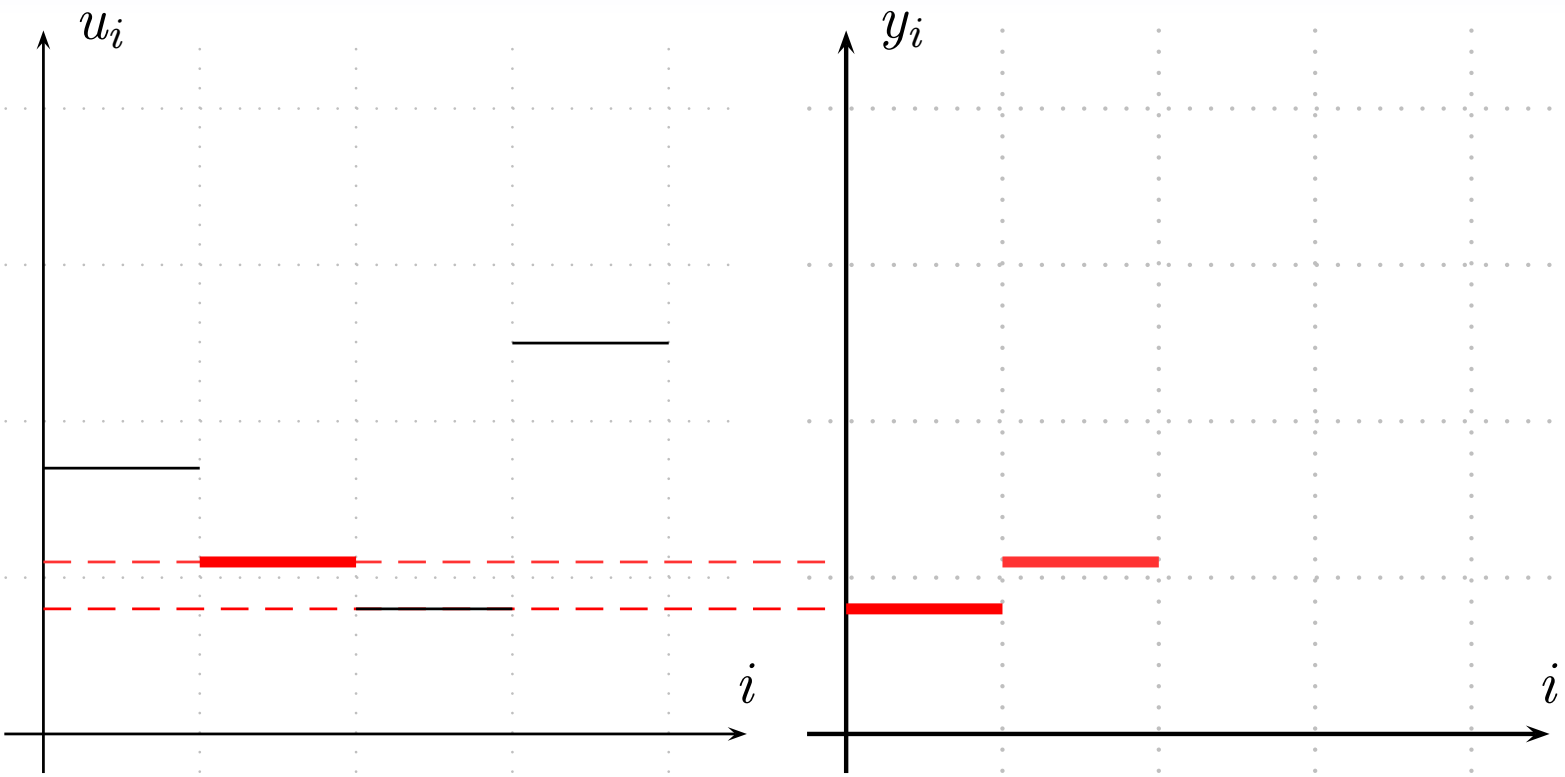
(II) Illustration du principe de l'algorithme



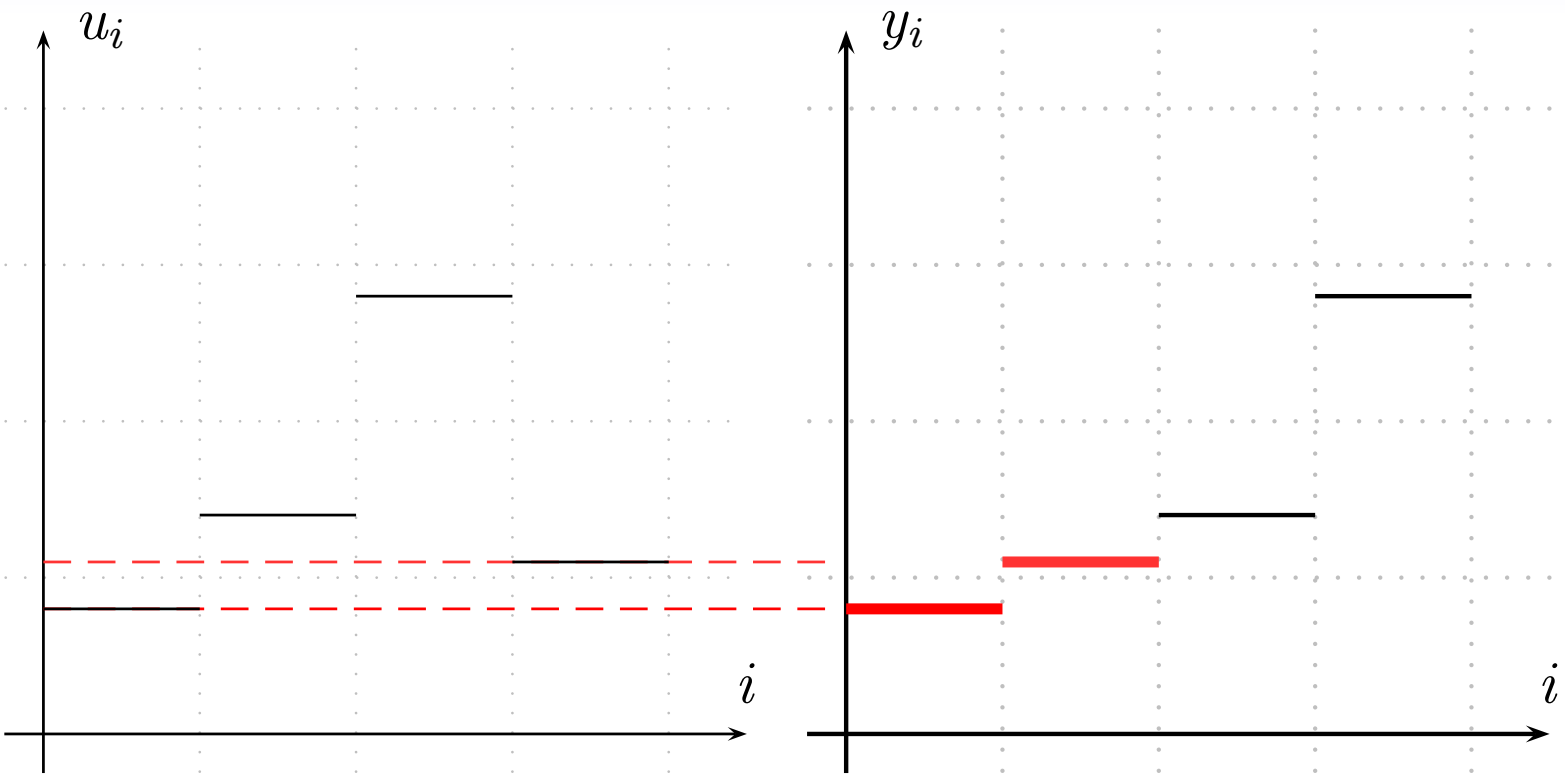
(II) Illustration du principe de l'algorithme



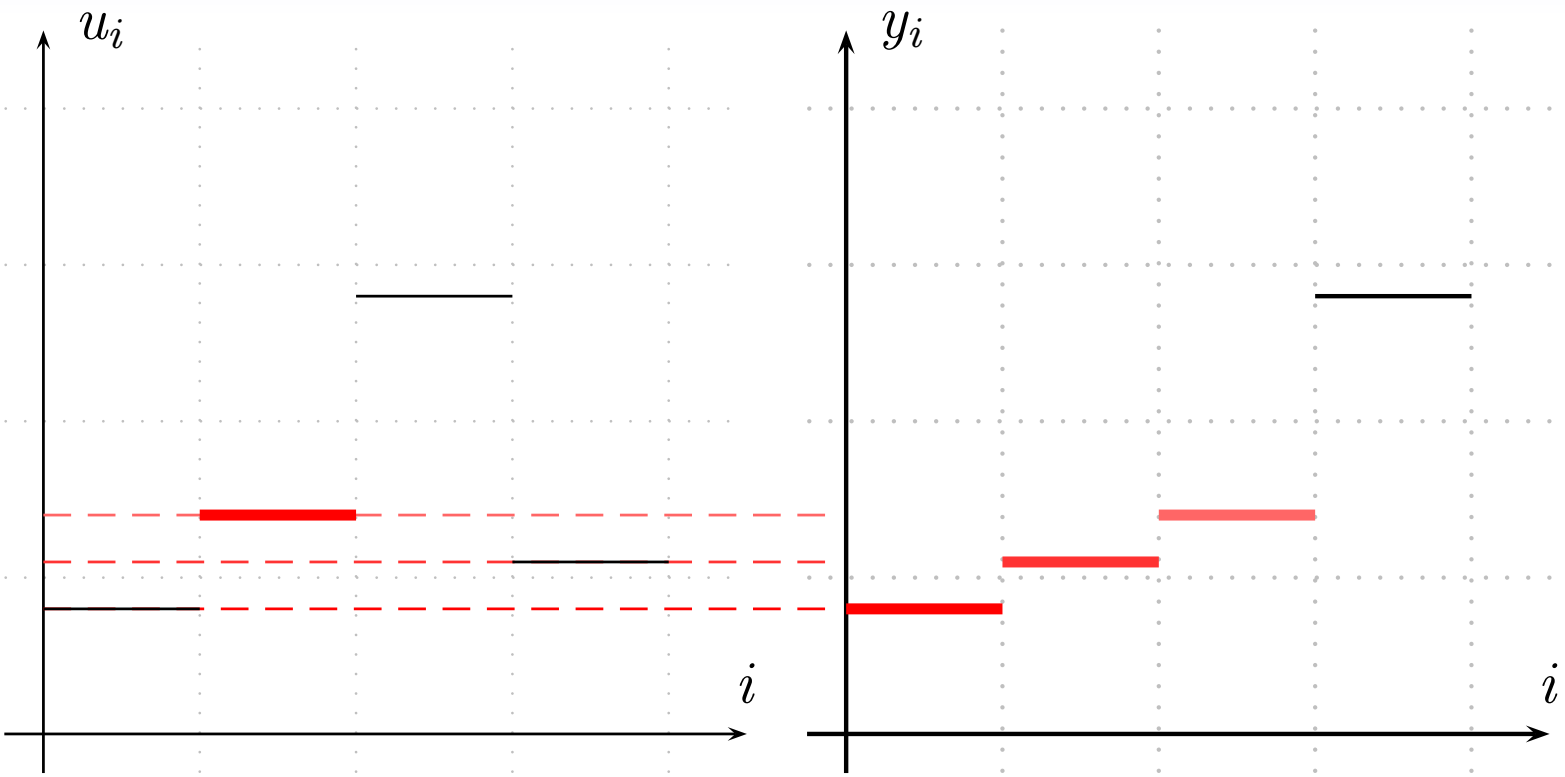
(II) Illustration du principe de l'algorithme



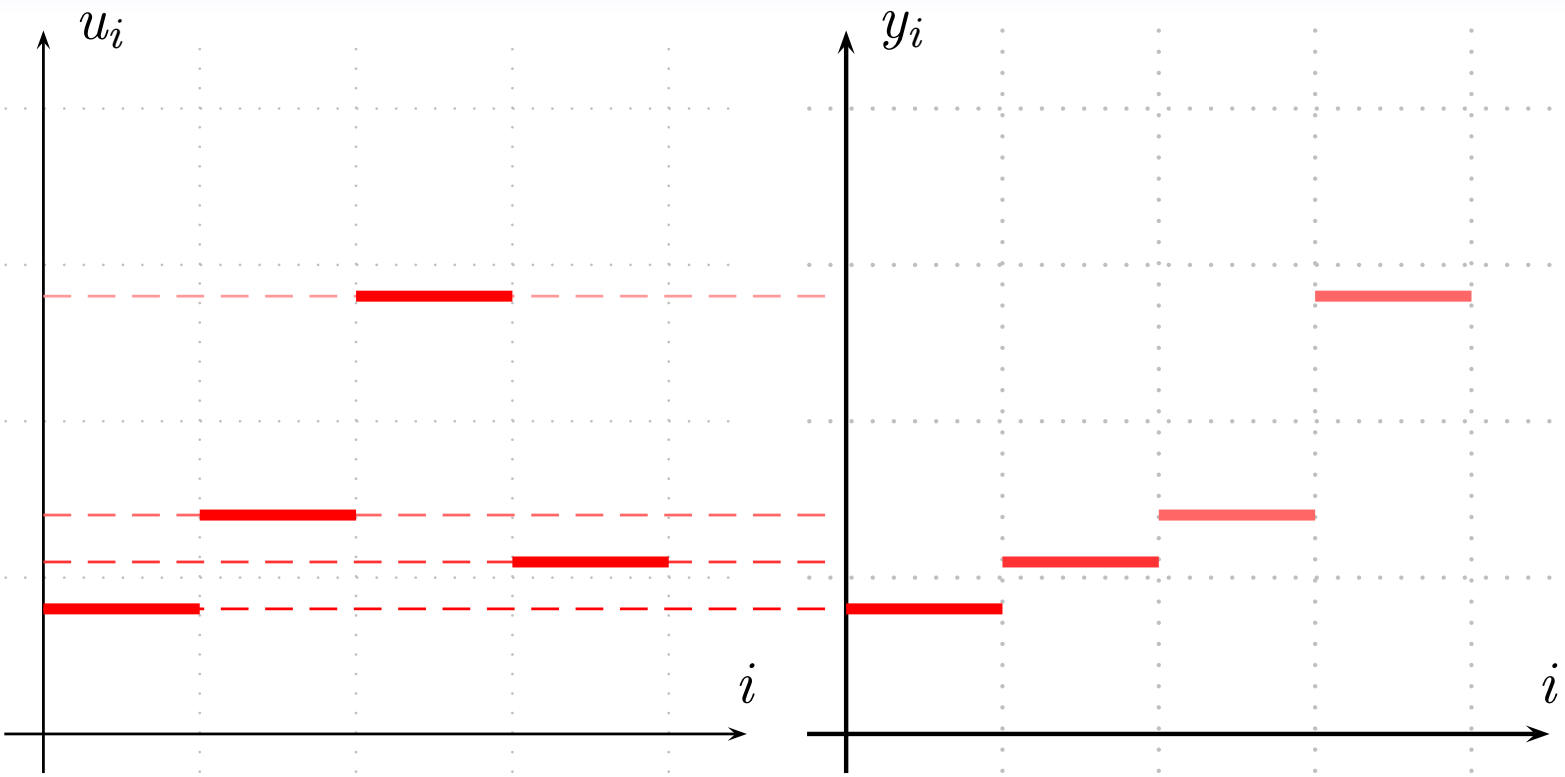
(II) Illustration du principe de l'algorithme



(II) Illustration du principe de l'algorithme



(II) Illustration du principe de l'algorithme



(II) Une contrainte de tri

Comment assurer que $\langle y_1, \ldots, y_n \rangle$ est la version triée du vecteur objectif ?

(II) Une contrainte de tri

Comment assurer que $\langle y_1, \ldots, y_n \rangle$ est la version triée du vecteur objectif ?
Utilisation d'une contrainte :

Contrainte Sort

Soient $\overrightarrow{x}$ et $\overrightarrow{x'}$ deux vecteurs de variables de même longueur. La contrainte $\mathbf{Sort}(\overrightarrow{x}, \overrightarrow{x'})$ porte sur les variables de $\overrightarrow{x}$ et de $\overrightarrow{x'}$ et n'autorise que les tuples pour lesquels les valeurs de $\overrightarrow{x'}$ correspondent à la version triée dans l'ordre croissant des valeurs de $\overrightarrow{x}$.

Utilisation des travaux de [Bleuzen-Guernalec and Colmerauer, 1997] introduisant un algorithme de filtrage de bornes en $O(n \log(n))$ pour la contrainte **Sort**, et des travaux plus récents de [Mehlhorn and Thiel, 2000].

(II) L'algorithme de leximin-branch-and-bound

Algorithme

if $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **"Inconsistent" then**
  return "Inconsistent"
end
$(\mathcal{X}', \mathcal{D}', \mathcal{C}') \leftarrow$introduire les variables $\langle y_1, \ldots, y_n \rangle$ dans $(\mathcal{X}, \mathcal{D}, \mathcal{C})$
for $i \leftarrow 1$ **to** n **do**
  $\widehat{v} \leftarrow \mathbf{maximize}((\mathcal{X}', \mathcal{D}', \mathcal{C}'), y_i)$ $d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$
end
return $\widehat{v}_{\downarrow \mathcal{X}}$

(III) Une reformulation de la contrainte de tri

Proposition

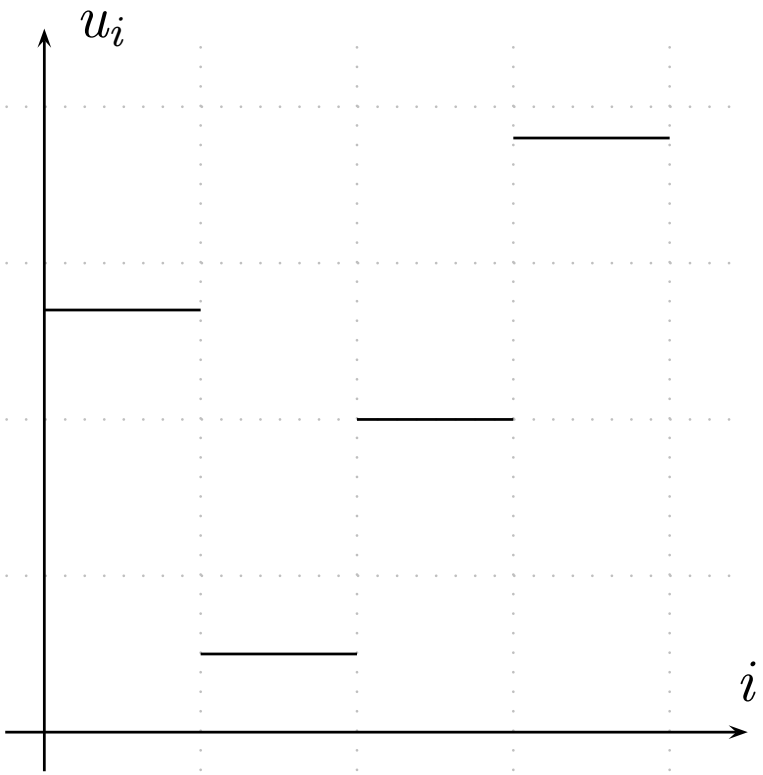
$\langle y_1, \ldots, y_n \rangle$ est la version triée dans l'ordre croissant de $\langle u_1, \ldots, u_n \rangle$ ssi :

- y_1 est la valeur maximale telle que tous les u_i sont plus grands que y_1 ;
- y_2 est la valeur maximale telle que $n-1$ valeurs parmi les u_i sont plus grands que y_2, sachant que tous les u_i sont plus grands que y_1 ;

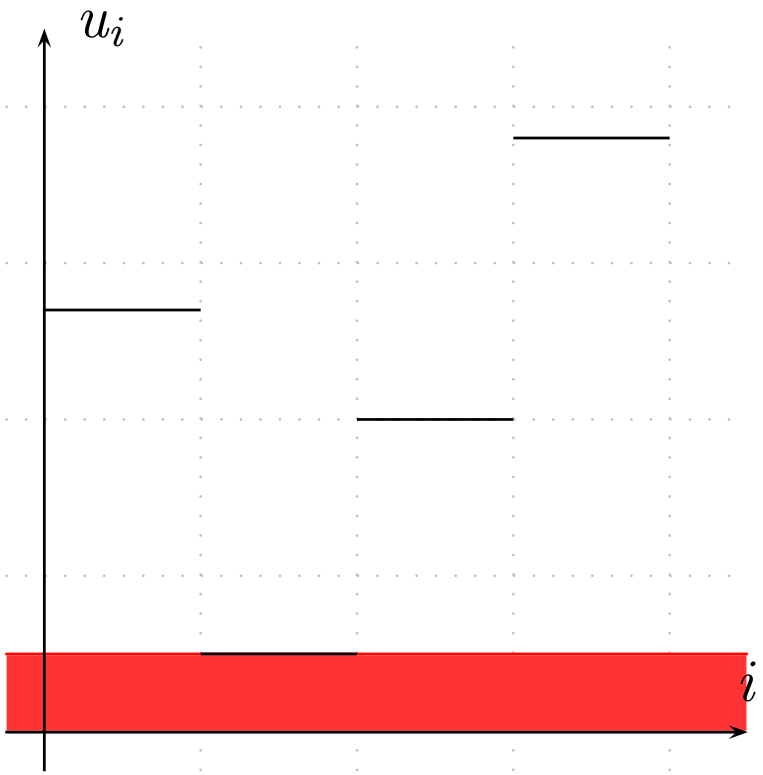
$$\vdots$$

- y_n est la valeur maximale telle que 1 valeur parmi les u_i est plus grande que y_n, sachant que 2 valeurs parmi les u_i sont plus grandes que y_{n-1}, ..., et que tous les u_i sont plus grands que y_1.

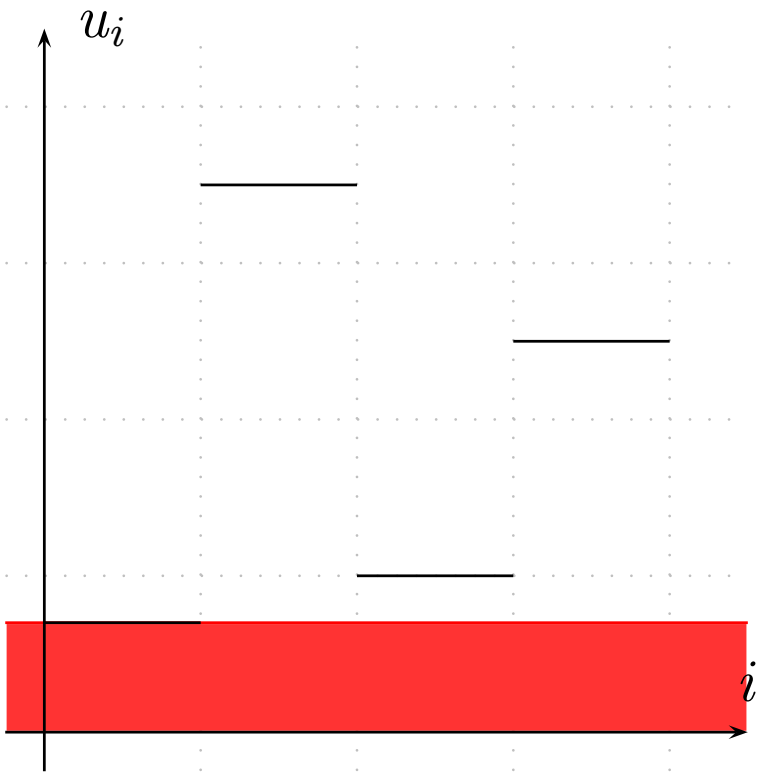
(III) Une reformulation de la contrainte de tri



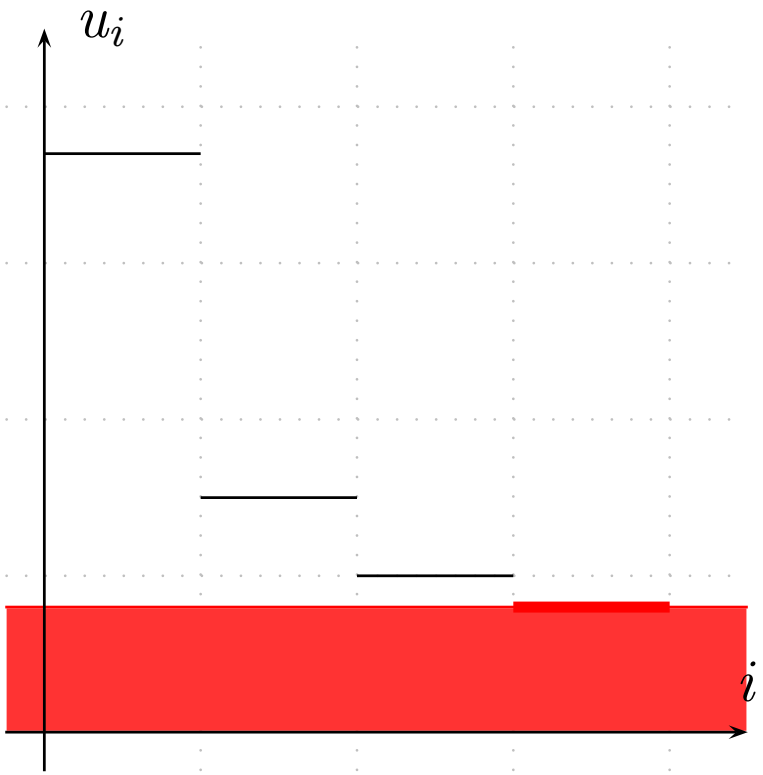
(III) Une reformulation de la contrainte de tri



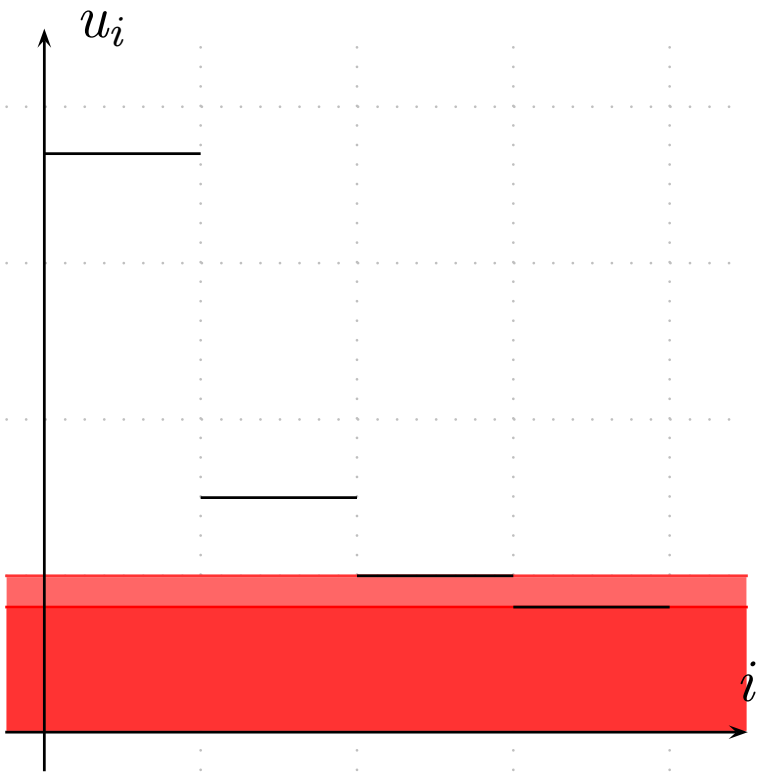
(III) Une reformulation de la contrainte de tri



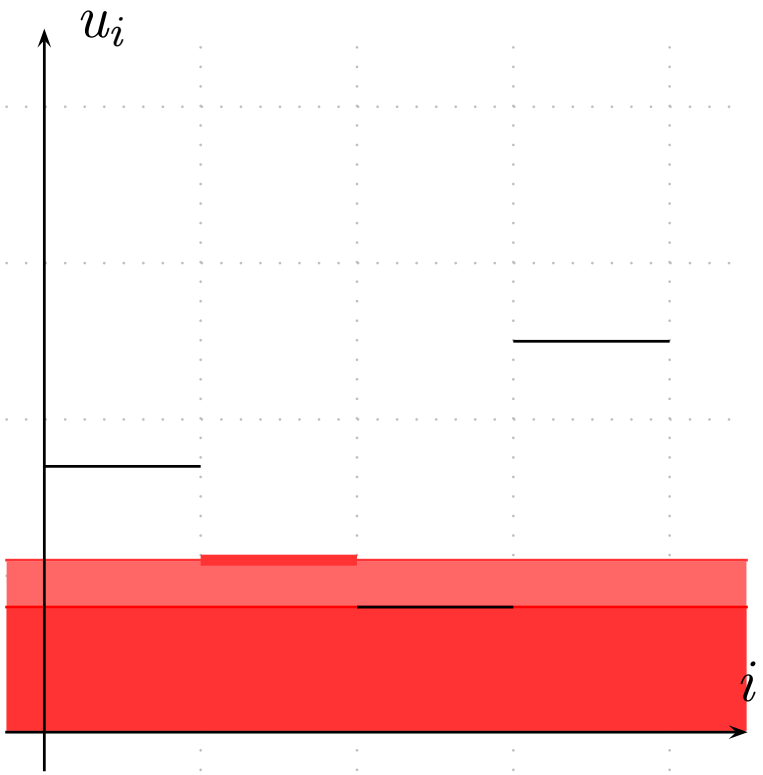
(III) Une reformulation de la contrainte de tri



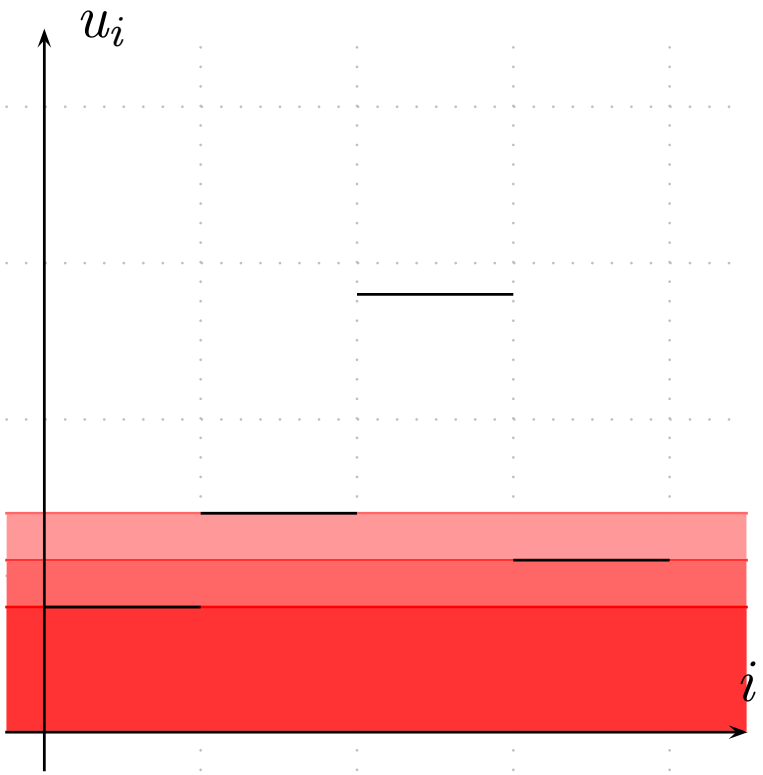
(III) Une reformulation de la contrainte de tri



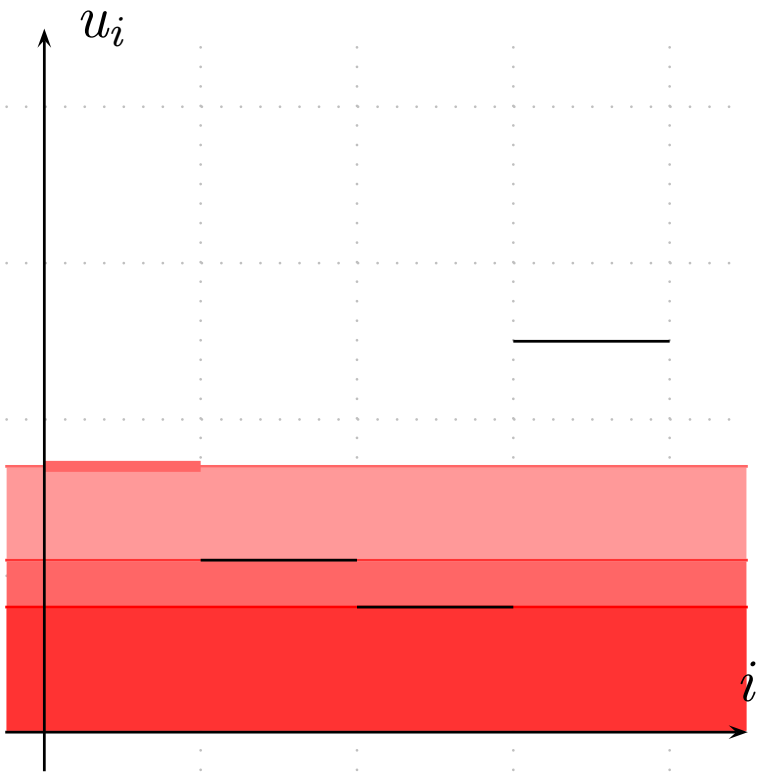
(III) Une reformulation de la contrainte de tri



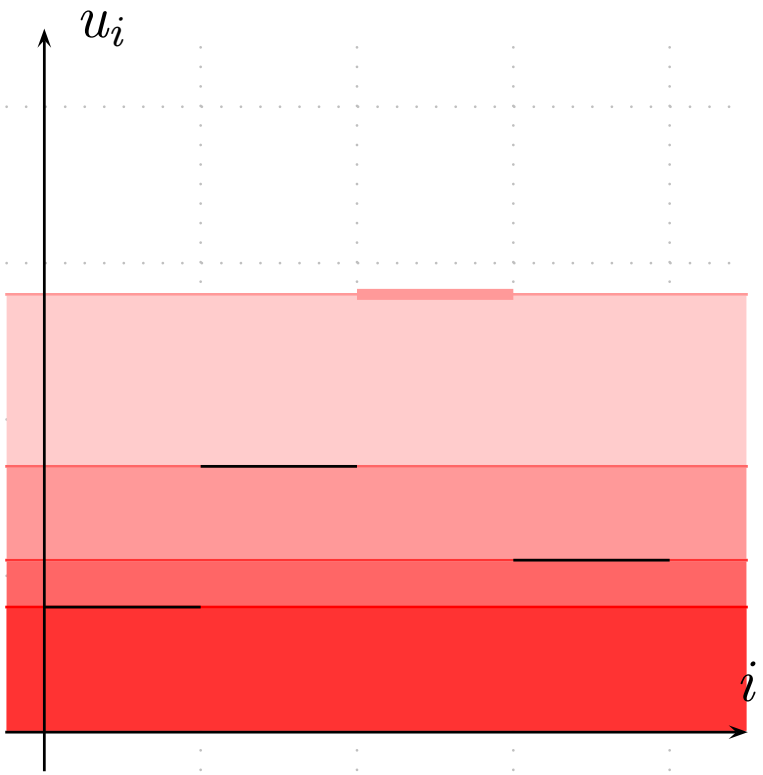
(III) Une reformulation de la contrainte de tri



(III) Une reformulation de la contrainte de tri



(III) Une reformulation de la contrainte de tri



(III) La contrainte AtLeast

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

Contrainte AtLeast [Van Hentenryck *et al.*, 1992]

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.

(III) 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**

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

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

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

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

fin

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

(III) 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**

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

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

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

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

fin

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

(III) 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 $\widehat{v}_{\downarrow \mathcal{X}}$

(III) 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 $\widehat{v}_{\downarrow \mathcal{X}}$

(III) 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 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 y_i**

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

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

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

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

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**

(IV) Principe de l'algorithme [Dubois and Fortemps, 1999]

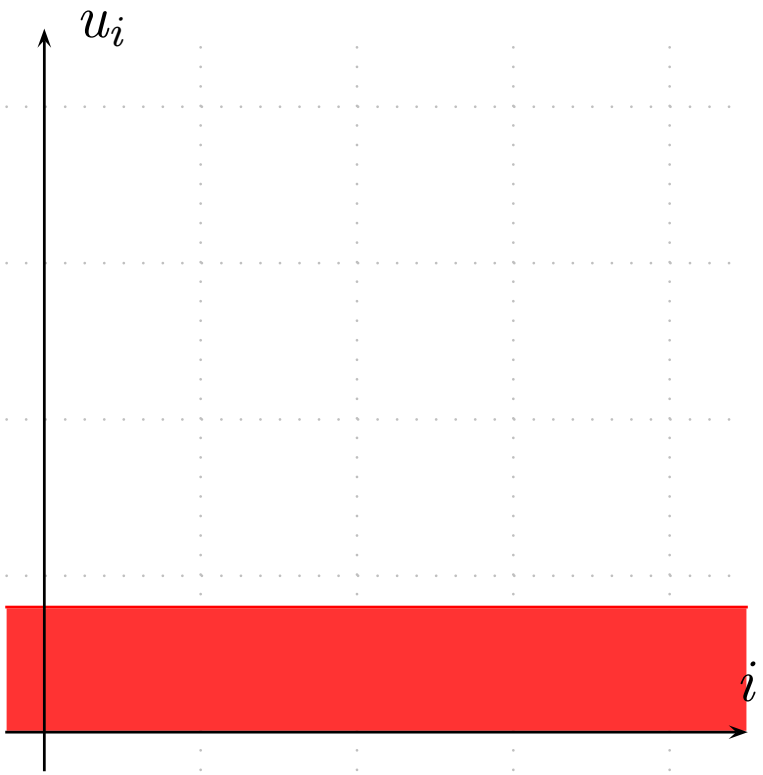
Sous-ensemble saturé

Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un réseau de contraintes, $\overrightarrow{u}$ un ensemble de variables objectif, et α la valeur maximale telle que il existe une solution affectant tous les u_i à une valeur supérieure à α. Un sous-ensemble $\mathcal{S}$ de variables de $\overrightarrow{u}$ est dit **saturé** s'il existe une solution telle que tous les u_i de $\mathcal{S}$ sont instanciés à α, et tous les u_i qui ne sont pas dans $\mathcal{S}$ à une valeur strictement supérieure à α.

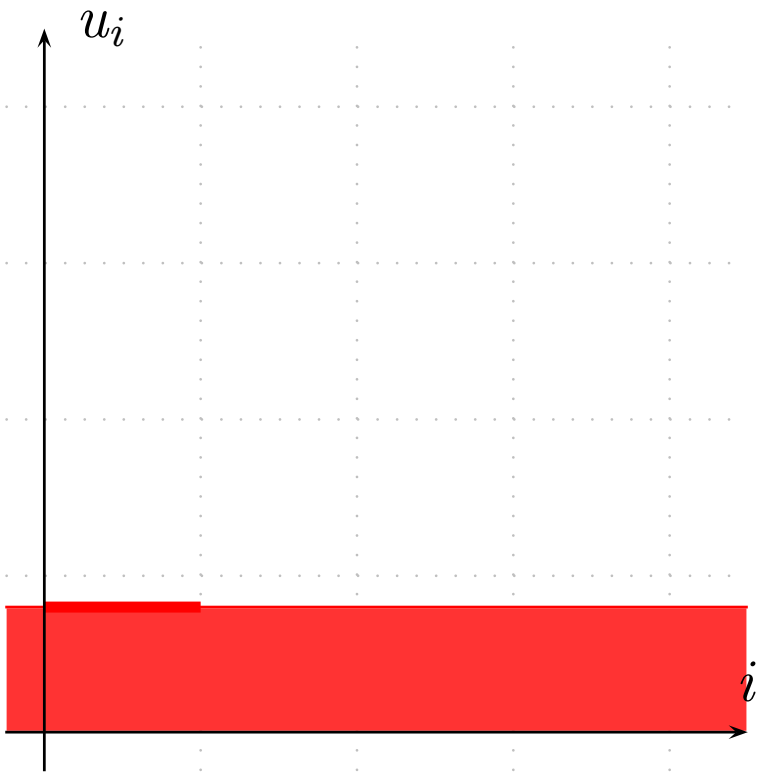
(IV) Principe de l'algorithme [Dubois and Fortemps, 1999]

- À chaque étape de l'algorithme, on maximise la composante courante y_i du leximin (voir algorithmes précédents).
- Pour chaque sous-ensemble saturé minimal pour la cardinalité, on fixe les u_j de ce sous-ensemble à y_i.
- Pour chacun de ces sous-ensembles, on recommence le processus sur le nouveau réseau de contraintes créé.

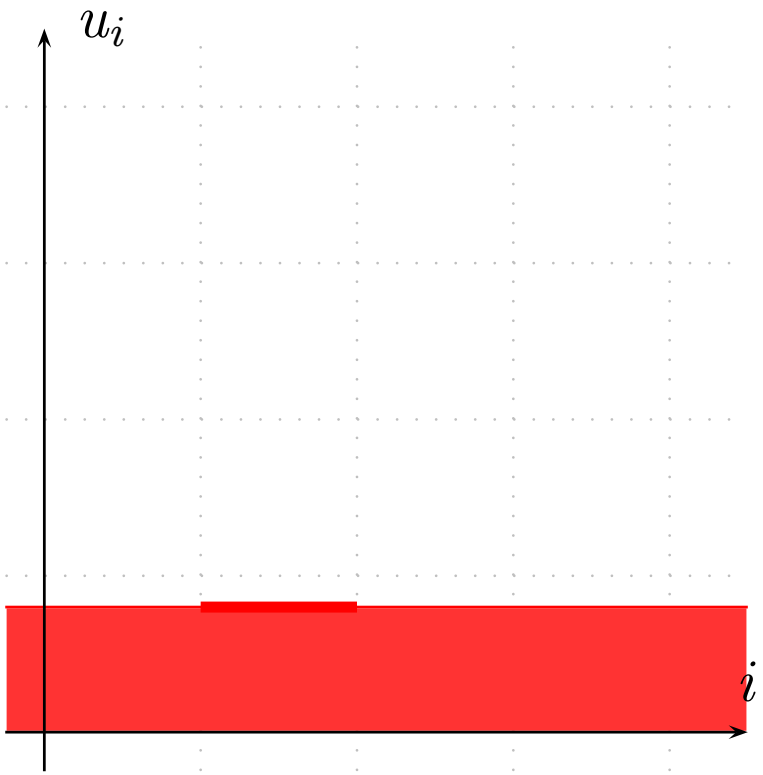
(IV) Illustration de l'algorithme



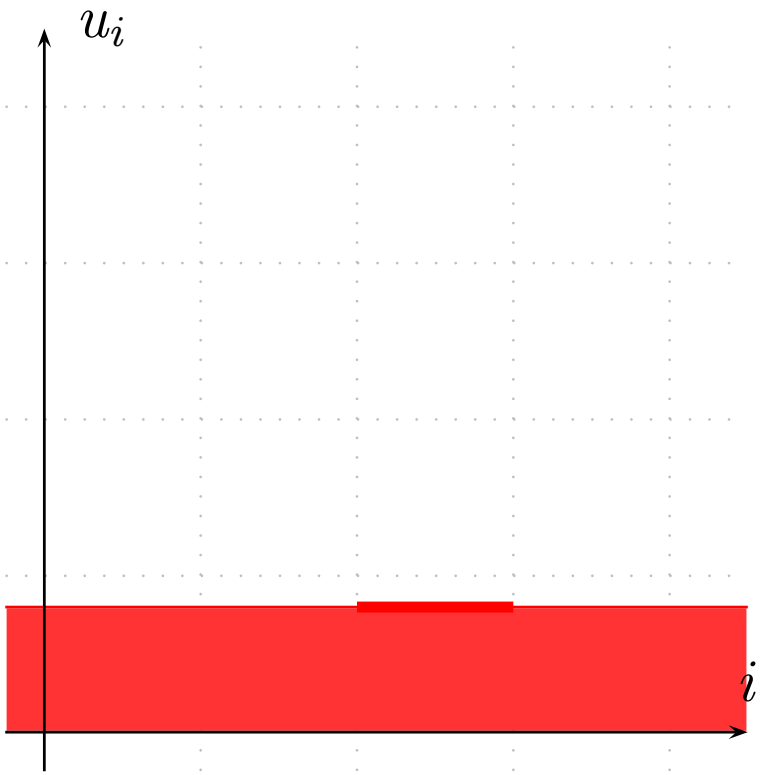
(IV) Illustration de l'algorithme



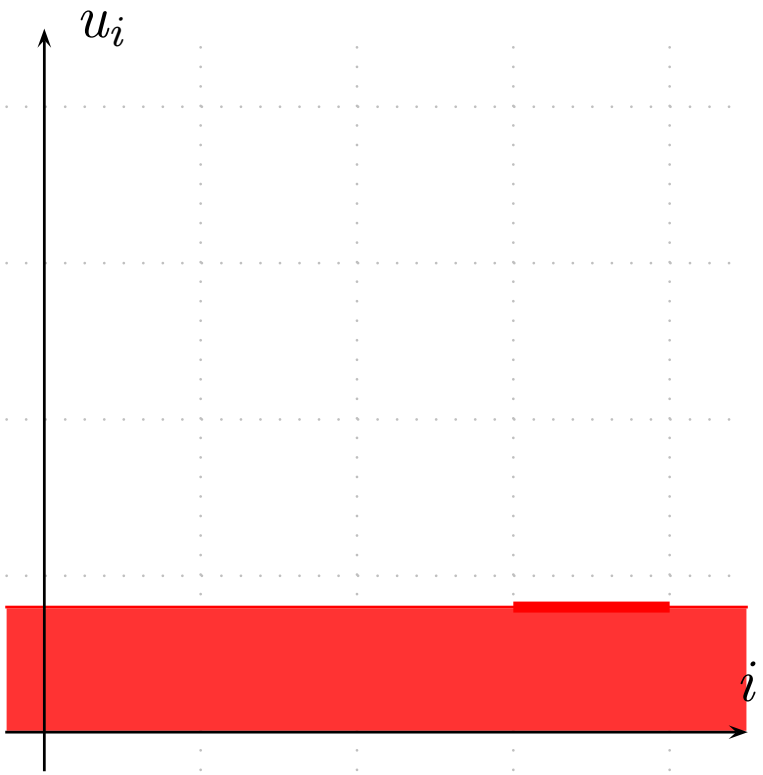
(IV) Illustration de l'algorithme



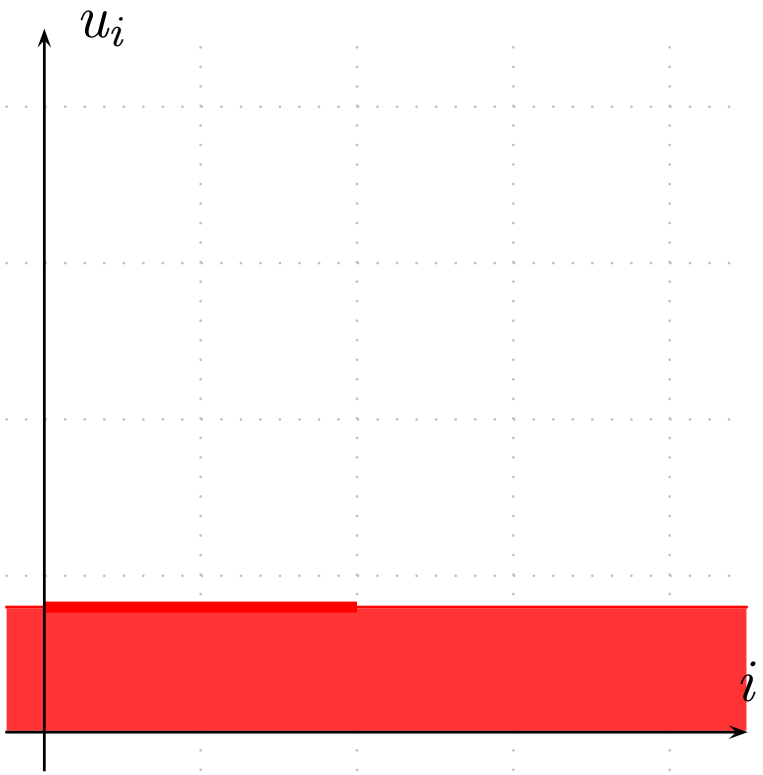
(IV) Illustration de l'algorithme



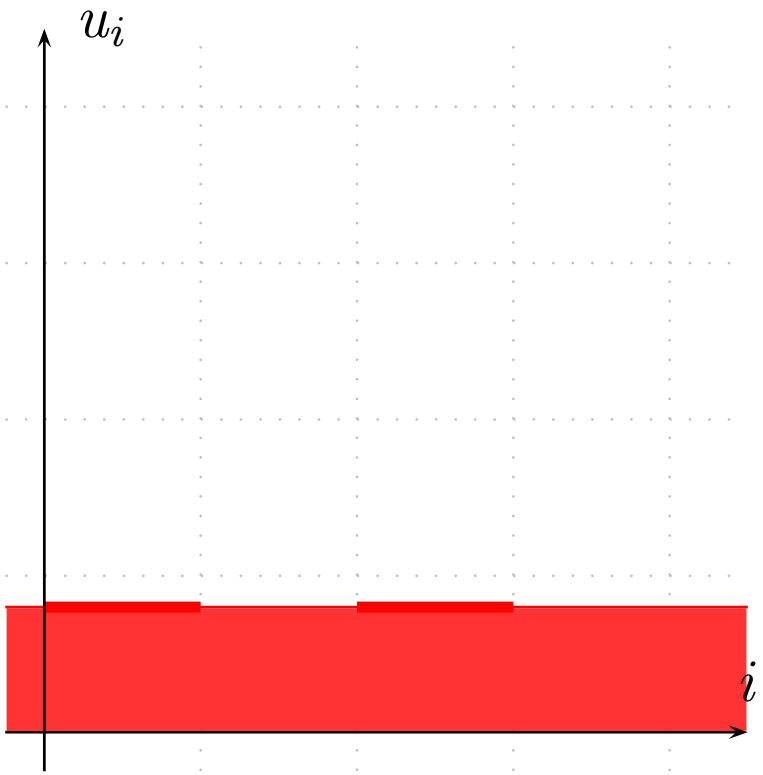
(IV) Illustration de l'algorithme



(IV) Illustration de l'algorithme



(IV) Illustration de l'algorithme



Inconvénient majeur de l'algorithme

Il faut potentiellement calculer tous les sous-ensembles de valeurs de $\overrightarrow{u} \rightsquigarrow$ potentiellement un nombre exponentiel de résolutions successives.

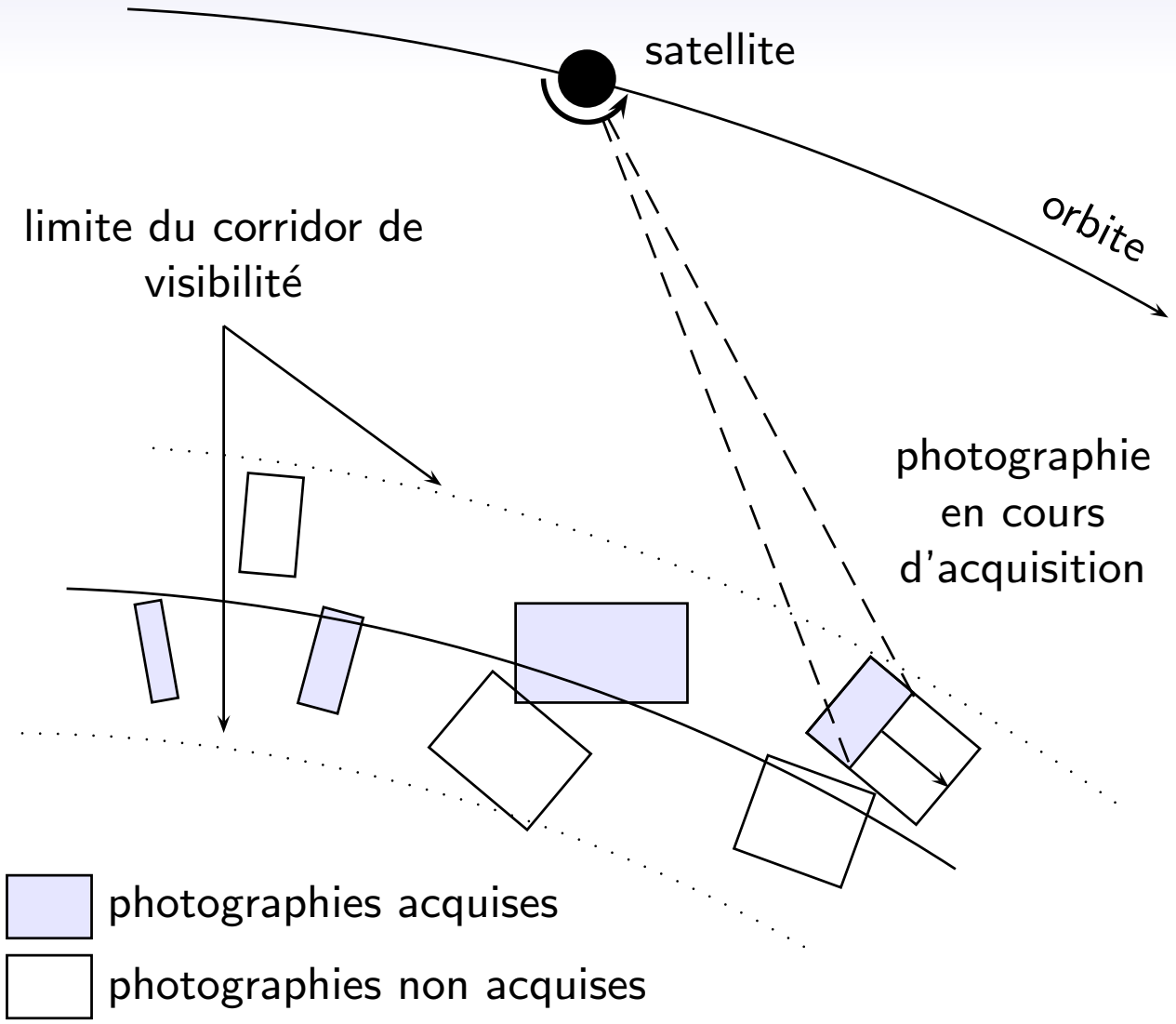
Algorithme à n'utiliser donc que si l'on est certain que les sous-ensembles saturés de cardinalité minimale sont de petite taille.

Plan de l'exposé du jour

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 fondé sur la contrainte **AtLeast** 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 fondé sur la contrainte **AtLeast** 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 :

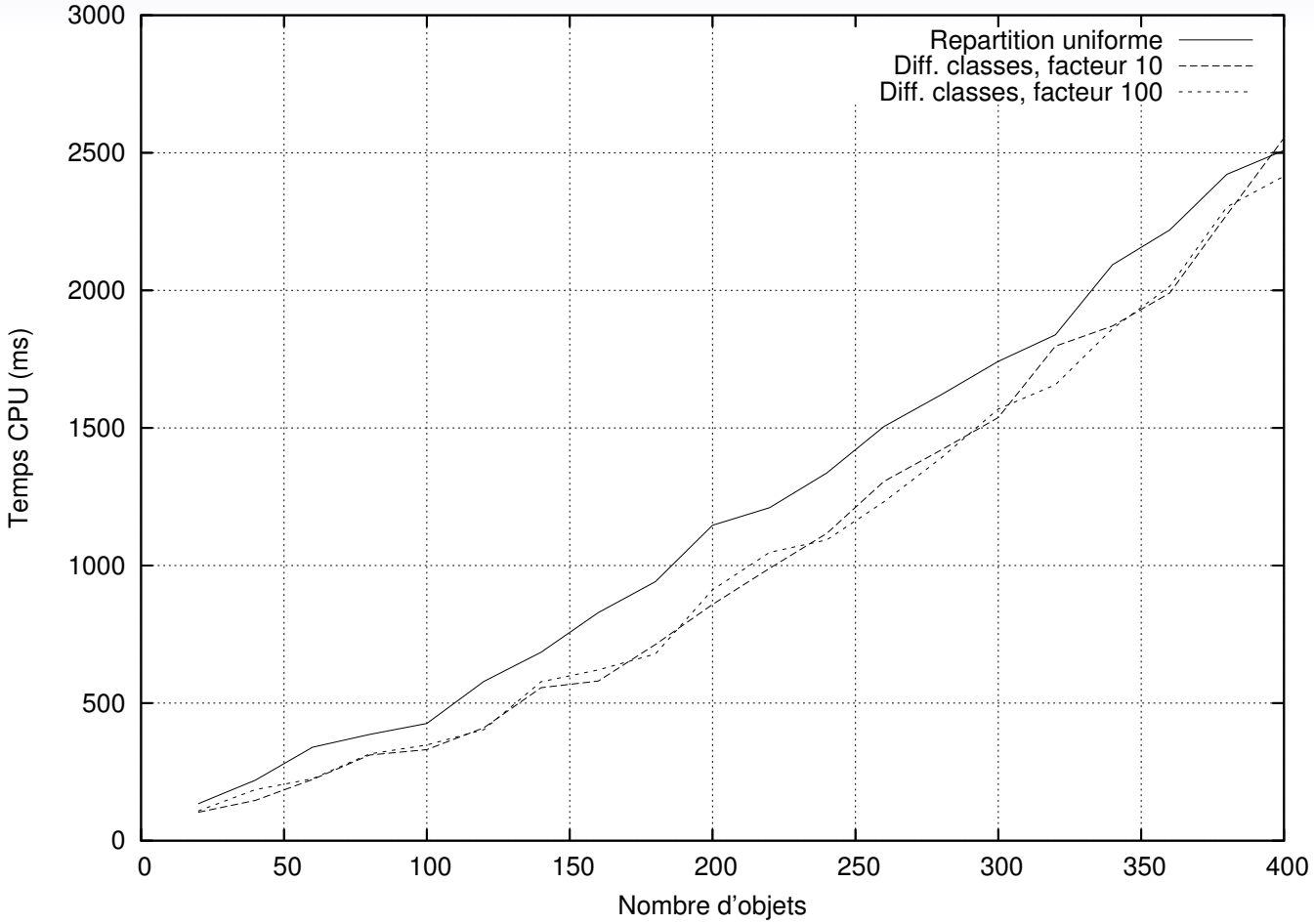
Deux implantations de l'algorithme fondé sur la contrainte **AtLeast** ont été testées :

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

Instance moyenne :

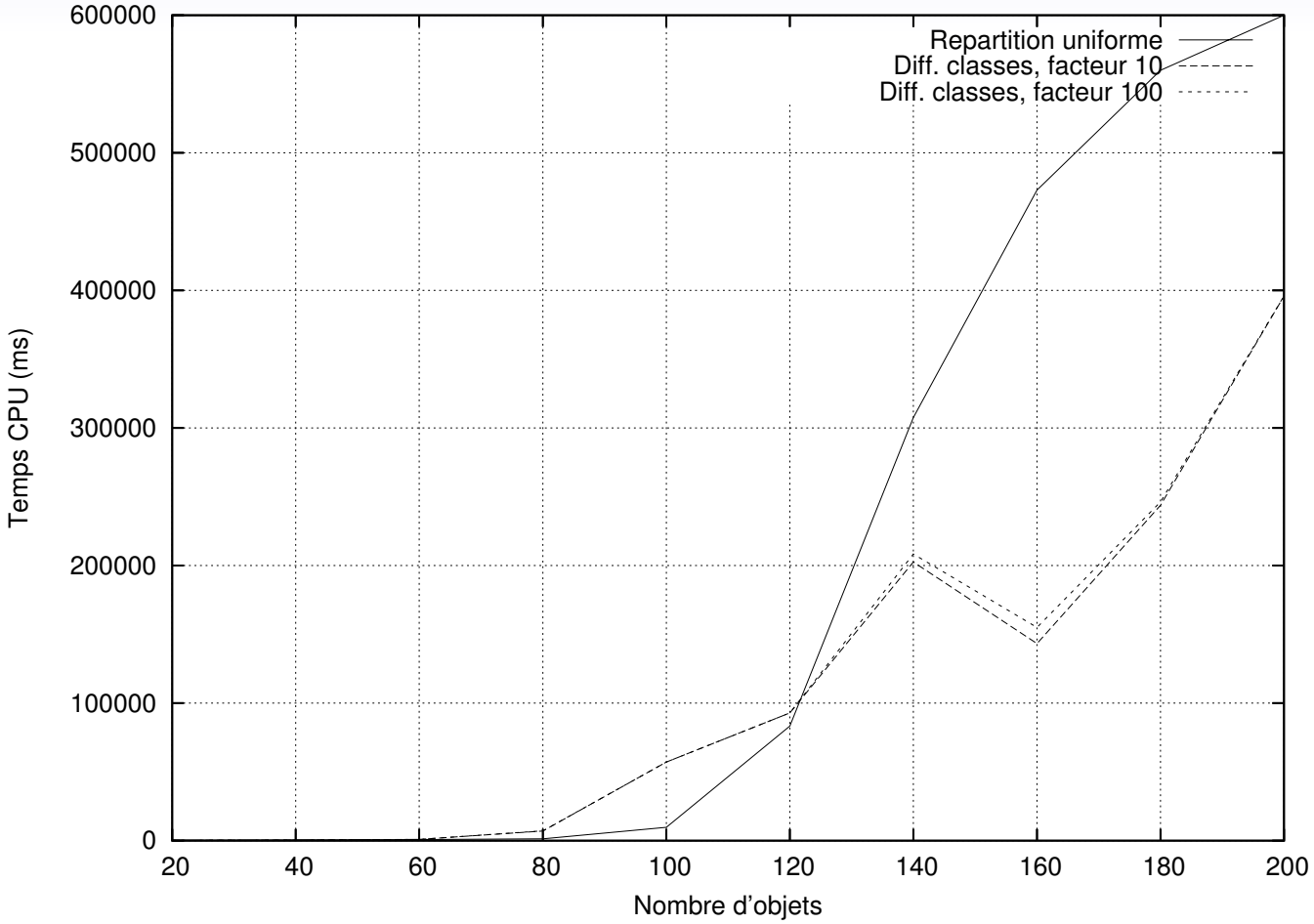
4 agents, droits inégaux, 150 objets, contraintes de volume autorisant 10 objets sur 20 consécutifs.

Résultats



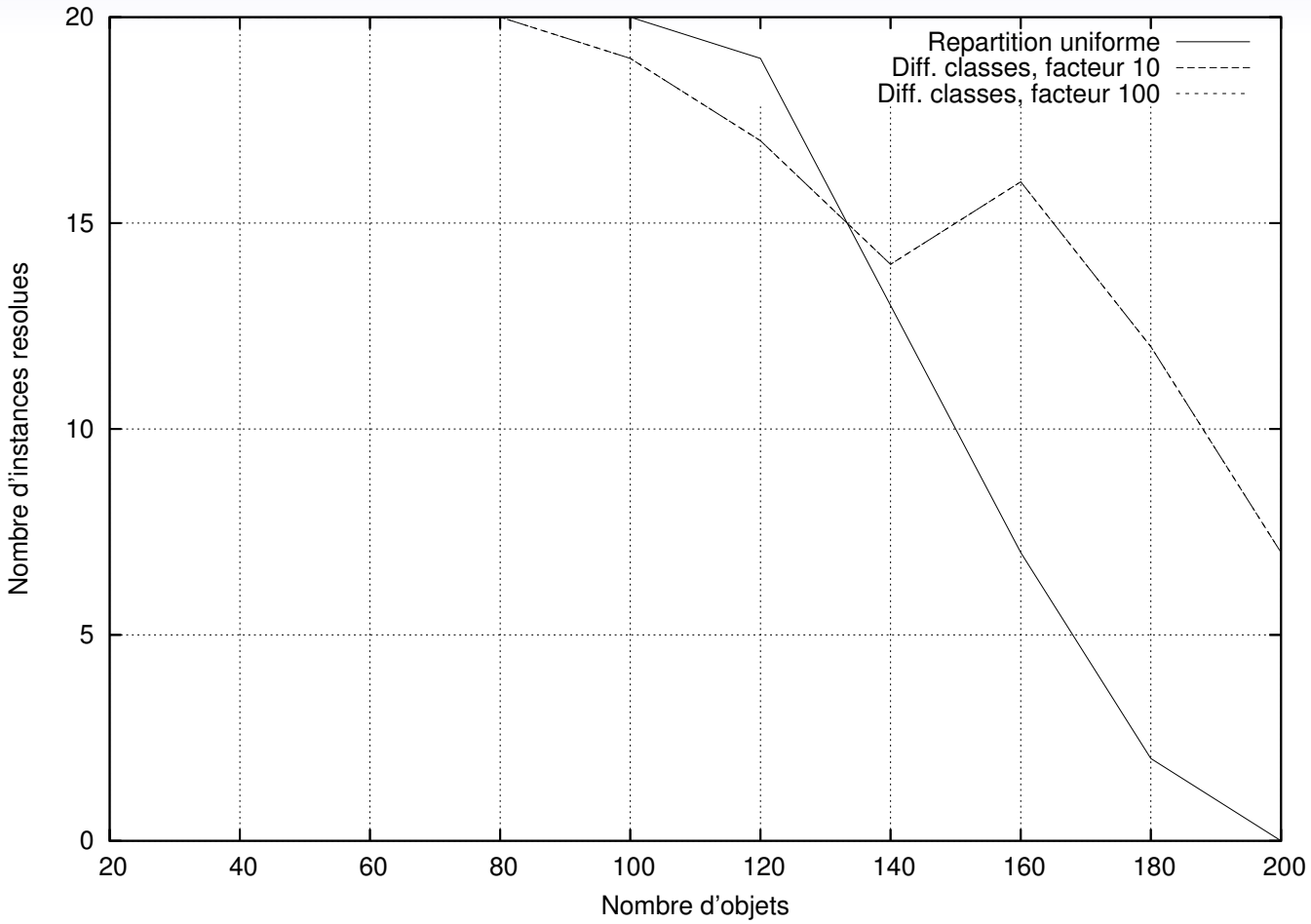
Temps de calcul avec CPLEX.

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.

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.

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.

Les tests des autres algorithmes sont en cours. Quelques constations :

- L'algorithme fondé sur la contrainte Multiset Ordering semble meilleur sur les instances au nombre d'agents plus élevé.

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.

Les tests des autres algorithmes sont en cours. Quelques constations :

- L'algorithme fondé sur la contrainte Multiset Ordering semble meilleur sur les instances au nombre d'agents plus élevé.
- L'algorithme fondé sur la contrainte Sort semble moins bon que les deux précédents.

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.

Les tests des autres algorithmes sont en cours. Quelques constations :

- L'algorithme fondé sur la contrainte Multiset Ordering semble meilleur sur les instances au nombre d'agents plus élevé.
- L'algorithme fondé sur la contrainte Sort semble moins bon que les deux précédents.
- S'il y a de nombreux *ex-aequo* dans le vecteur leximin, l'algorithme de Dubois explose.

Plan de l'exposé du jour

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.
- **Algorithmes :** introduction de plusieurs algorithmes de calcul fondés 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.
- **Algorithmes :** introduction de plusieurs algorithmes de calcul fondés sur la programmation par contraintes.
- **Implantation :** implantation et évaluation des algorithmes en Java avec CHOCO et CPLEX pour l'un d'entre eux.

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.
- **Algorithmes :** introduction de plusieurs algorithmes de calcul fondés sur la programmation par contraintes.
- **Implantation :** implantation et évaluation des algorithmes en Java avec CHOCO et CPLEX pour l'un d'entre eux.
- **Application :** problème d'allocation pour le partage de ressources satellitaires.

Suite de l'étude

- Calcul du leximin-optimal
  - Amélioration de la contrainte de cardinalité utilisée.
- 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.

Kurt Mehlhorn and Sven Thiel.
Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint.
In Rina Dechter, editor, *Proc. of CP'00*, pages 306–319, Singapore, 2000.

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.

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

Merci de votre attention

Transparents (prochainement) disponibles à l'URL :
http://www.cert.fr/dcsd/THESES/sbouveret/ressources/seminaires/seminaire270606.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