# Comparison of two constraint programming algorithms for computing leximin-optimal allocations

Sylvain Bouveret and Michel Lemaître

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

Workshop on Modeling and Solving Problems with Constraints
ECAI'06, Riva del Garda, Italy
29th August 2006

# Fairness in combinatorial problems. . .

Many real-world combinatorial problems. . .

- Nurse rostering problem.
- Balanced timetables.
- Fair allocation of airport and airspace resources (to several airlines).
- Fair share of Earth Observation Satellites.

. . . imply human **agents** and thus involve directly or indirectly the concept of **fairness**.

All of these problems are combinatorial collective decision making problems, under admissibility constraints, and fairness requirements.

*How can we model and solve this kind of problems in a Constraint Programming framework ?*

# Fairness in combinatorial problems. . .

Many real-world combinatorial problems. . .

- Nurse rostering problem.
- Balanced timetables.
- Fair allocation of airport and airspace resources (to several airlines).
- Fair share of Earth Observation Satellites.

. . . imply human **agents** and thus involve directly or indirectly the concept of **fairness**.
All of these problems are combinatorial collective decision making problems, under admissibility constraints, and fairness requirements.

*How can we model and solve this kind of problems in a Constraint Programming framework ?*

# Fairness in combinatorial problems. . .

Many real-world combinatorial problems. . .

- Nurse rostering problem.
- Balanced timetables.
- Fair allocation of airport and airspace resources (to several airlines).
- Fair share of Earth Observation Satellites.

. . . imply human **agents** and thus involve directly or indirectly the concept of **fairness**.

All of these problems are combinatorial collective decision making problems, under admissibility constraints, and fairness requirements.

*How can we model and solve this kind of problems in a Constraint Programming framework ?*

# Outline

1. Formalizing and modeling the problem
   - CSP and constraint programming
   - Fairness in collective decision making
   - Leximin and Constraint Programming
2. Two constraint programming algorithms
   - Using a set of cardinality constraints
   - Using a multiset ordering constraint
3. Application, tests and results
   - Benchmark and results

# Outline

# Constraint networks

## Constraint network [Montanari, 1974]

A constraint network is based on :

- a set of variables $\mathcal{X} = \{x_1, \ldots, x_p\}$ ;
- a set of domains $\mathcal{D} = \{d_{x_1}, \ldots, d_{x_p}\}$ ;
- a set of constraints $\mathcal{C}$, with, for all $C \in \mathcal{C}$ :
  - $X(C)$ the scope of the constraint,
  - $R(C)$ the set of allowed tuples of the constraint.

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

# The Constraint Satisfaction Problem

## Classical CSP

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.
*Is there a complete consistent instantiation $v$ of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ ?*

$\rightsquigarrow$ **NP**-complete.

## CSP with objective variable

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and an objective variable $o \in \mathcal{X}$, such that $d_o \subset \mathbb{N}$.
*What is the maximal value $\alpha$ of $d_o$ such that there is a complete consistent instantiation $\widehat{v}$ with $\widehat{v}(o) = \alpha$ ?*

$\rightsquigarrow$ **NP**-complete (for the associated decision problem).

*How to introduce a fairness "constraint", expressing the need for compromises between the satisfaction of the different agents ?*

# The Constraint Satisfaction Problem

## Classical CSP

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.
*Is there a complete consistent instantiation $v$ of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ ?*

$\rightsquigarrow$ **NP**-complete.

## CSP with objective variable

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and an objective variable $o \in \mathcal{X}$, such that $d_o \subset \mathbb{N}$.
*What is the maximal value $\alpha$ of $d_o$ such that there is a complete consistent instantiation $\widehat{v}$ with $\widehat{v}(o) = \alpha$ ?*

$\rightsquigarrow$ **NP**-complete (for the associated decision problem).

*How to introduce a fairness "constraint", expressing the need for compromises between the satisfaction of the different agents ?*

# The Constraint Satisfaction Problem

## Classical CSP

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.
*Is there a complete consistent instantiation $v$ of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ ?*

$\rightsquigarrow$ **NP**-complete.

## CSP with objective variable

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and an objective variable $o \in \mathcal{X}$, such that $d_o \subset \mathbb{N}$.
*What is the maximal value $\alpha$ of $d_o$ such that there is a complete consistent instantiation $\widehat{v}$ with $\widehat{v}(o) = \alpha$ ?*

$\rightsquigarrow$ **NP**-complete (for the associated decision problem).

*How to introduce a fairness "constraint", expressing the need for compromises between the satisfaction of the different agents ?*

# Individual utility and collective decision

Given a collective decision making problem, the preferences of an agent regarding the set of admissible decisions $\mathcal{S}$ is given by a **utility function**:

## Individual utility function

Given an agent $a_i$ and the set of admissible decisions $\mathcal{S}$, the individual utility function of $a_i$ is a function $u_i : \mathcal{S} \rightarrow \mathbb{N}$.

The quality of a collective decision is measured using its associated **utility profile**, that is, the vector of its associated utilities.
"[Welfarism] judges a collective action on the basis of the utility levels enjoyed by the individual agents, and on those levels only" [Moulin, 1988]

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

# Individual utility and collective decision

Given a collective decision making problem, the preferences of an agent regarding the set of admissible decisions $\mathcal{S}$ is given by a **utility function**:

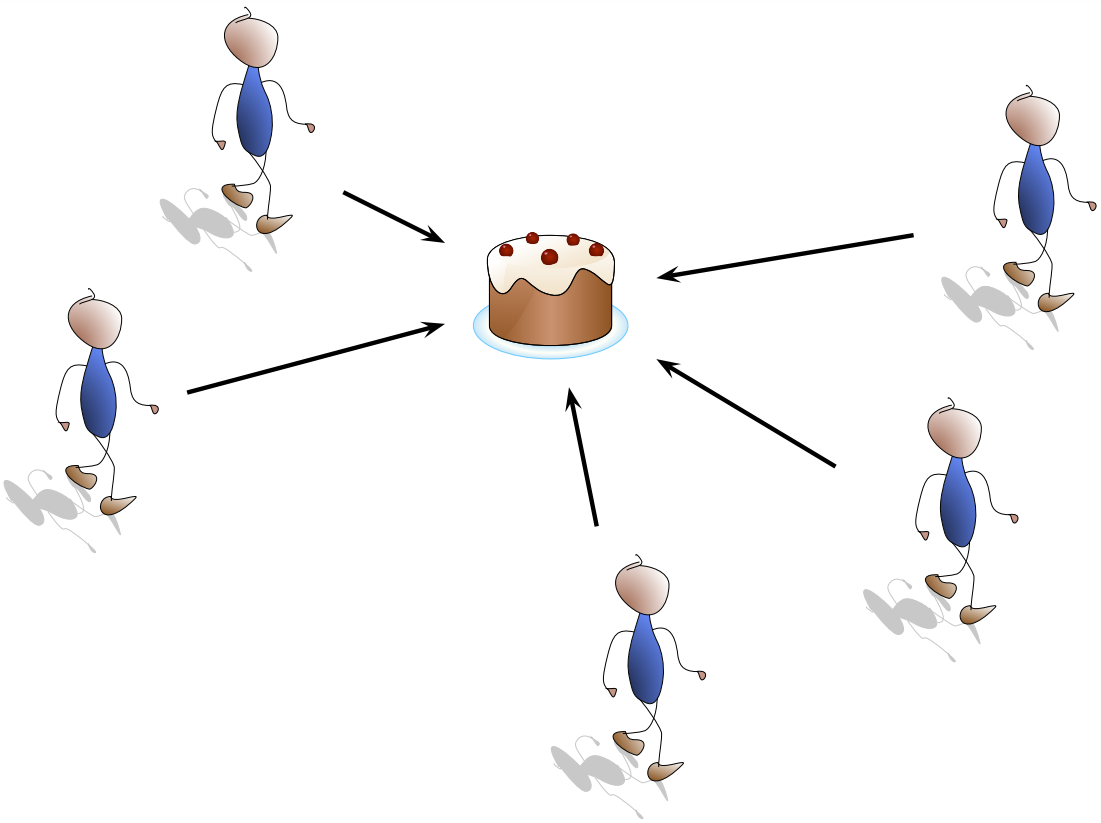
## Individual utility function

Given an agent $a_i$ and the set of admissible decisions $\mathcal{S}$, the individual utility function of $a_i$ is a function $u_i : \mathcal{S} \rightarrow \mathbb{N}$.

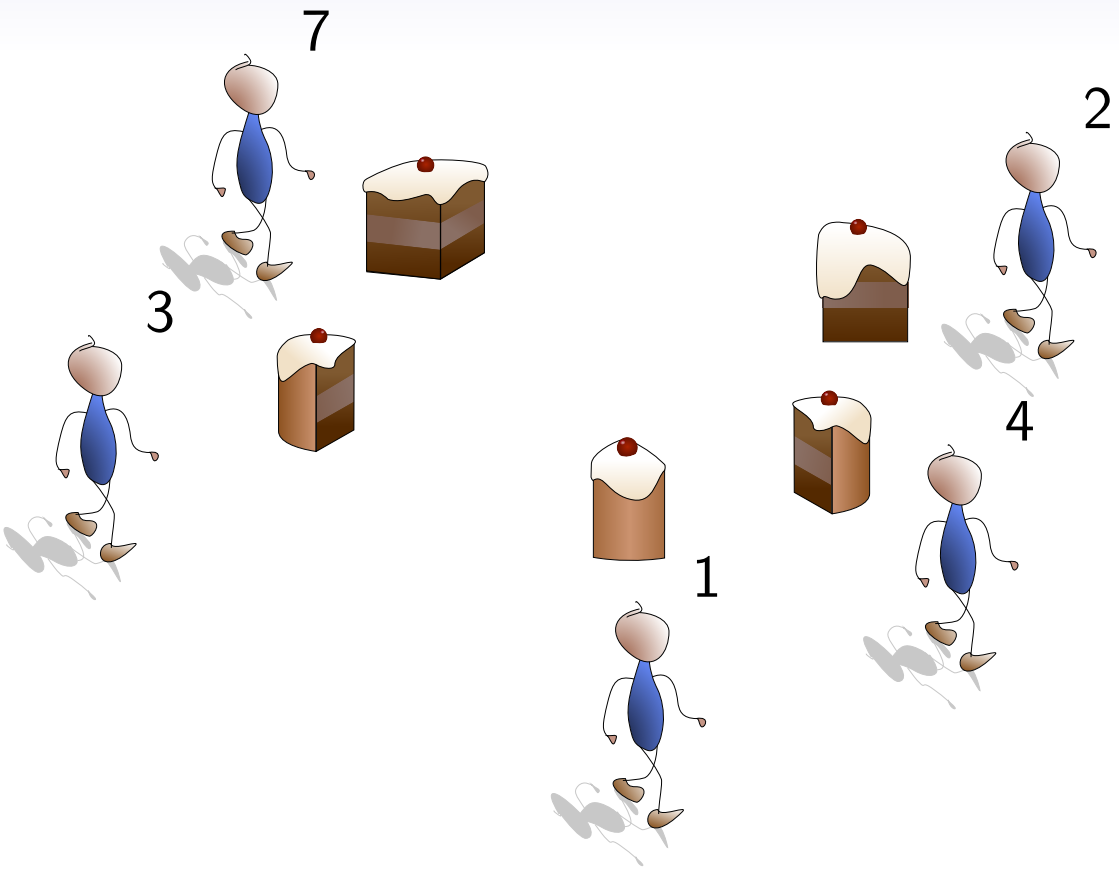
The quality of a collective decision is measured using its associated **utility profile**, that is, the vector of its associated utilities.
"[Welfarism] judges a collective action on the basis of the utility levels enjoyed by the individual agents, and on those levels only" [Moulin, 1988]

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

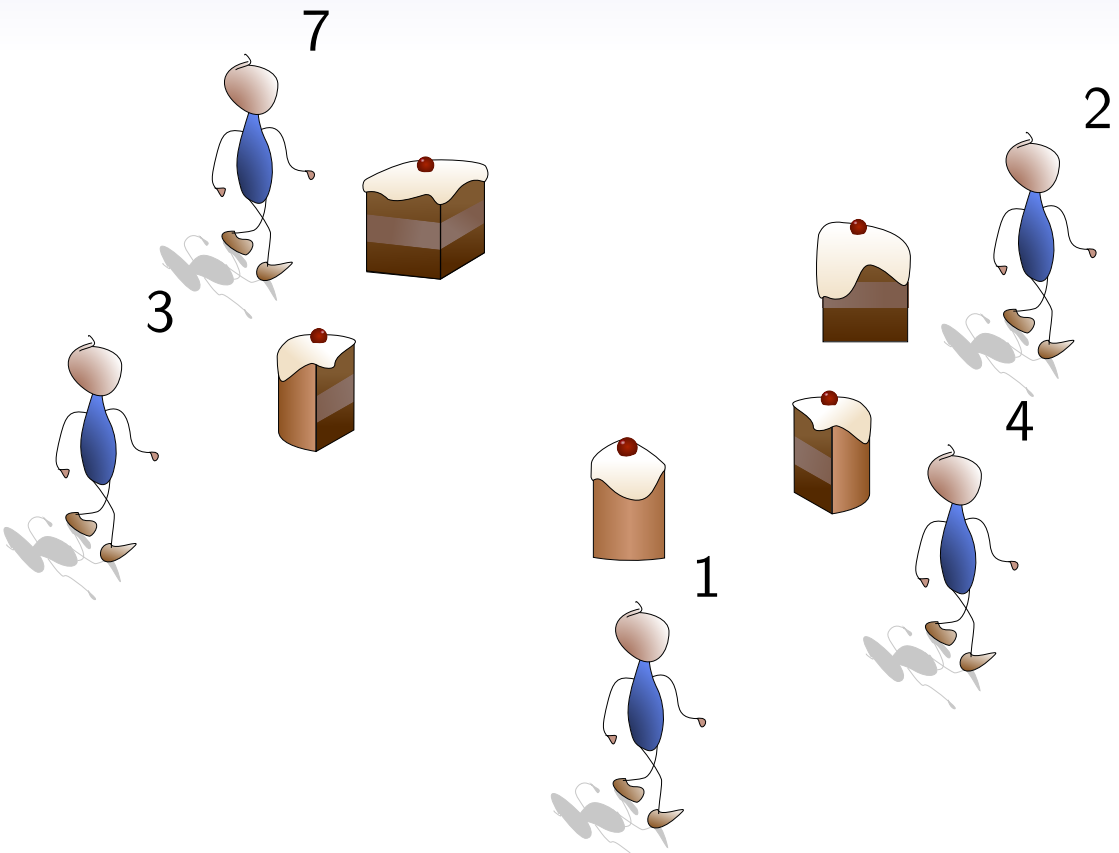
# Decision making and *welfarism*



# Decision making and *welfarism*



# Decision making and *welfarism*




Utility profile: $\overrightarrow{u} = \langle 3, 4, 2, 1, 7 \rangle$

# Social Welfare Ordering

We make use of a **Social Welfare Ordering** to compare the utility profiles.

## Social Welfare Ordering

A **Social Welfare Ordering** is a preorder $\preceq$ on $\mathbb{N}^n$.

## Examples

- Classical utilitarian Social Welfare Ordering.
- Egalitarian Social Welfare Ordering.
- Leximin Social Welfare Ordering.

# Social Welfare Ordering

We make use of a **Social Welfare Ordering** to compare the utility profiles.

## Social Welfare Ordering

A **Social Welfare Ordering** is a preorder $\preceq$ on $\mathbb{N}^n$.

## Examples

- Classical utilitarian Social Welfare Ordering.
- Egalitarian Social Welfare Ordering.
- Leximin Social Welfare Ordering.

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

## Classical utilitarian SWO [Harsanyi]

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

## Features

The agents are utility "producers".
It is indifferent to inequalities between agents $\rightsquigarrow$ it can lead to very unfair decisions.

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

**Classical utilitarian SWO [Harsanyi]**

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

**Fairness and utilitarian SWO**

$\langle 10, 10, 10, 10 \rangle \preceq \langle 41, 0, 0, 0 \rangle$, whereas $\langle 10, 10, 10, 10 \rangle$ is more equitable than $\langle 41, 0, 0, 0 \rangle$.

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

**Egalitarian SWO [Rawls]**

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

**Features**

It only takes the least satisfied agent into account $\rightsquigarrow$ natural inclination to fairness.

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

**Egalitarian SWO [Rawls]**

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

**Features**

It only takes the least satisfied agent into account $\rightsquigarrow$ natural inclination to fairness.

**On the other hand, it can lead to non Pareto-optimal decisions (drowning effect).**

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

**Egalitarian SWO [Rawls]**

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

**Egalitarian SWO and Pareto-efficiency**

$\langle 1,1,1,1 \rangle \sim \langle 1000,1,1000,1000 \rangle$, whereas $\langle 1,1,1,1 \rangle$ and $\langle 1000,1,1000,1000 \rangle$ are very different !

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

## Leximin SWO

Let $\overrightarrow{x}$ be a vector. We write $\overrightarrow{x^{\uparrow}}$ the sorted version of $\overrightarrow{x}$.
$\overrightarrow{u} \succ_{leximin} \overrightarrow{v} \Leftrightarrow \exists k$ such that $\forall i \leq k,\ u_i^{\uparrow} = v_i^{\uparrow}$ and $u_{k+1}^{\uparrow} > v_{k+1}^{\uparrow}$.
**In other words, a lexicographic comparison on the sorted vectors.**

## Performing a leximin comparison...

Two vectors to compare: $\overrightarrow{u} = \langle 4, 10, 3, 5 \rangle$ and $\overrightarrow{v} = \langle 4, 3, 6, 6 \rangle$.

- We sort the vectors: $\begin{cases} \overrightarrow{u}^{\uparrow} = \langle 3, 4, 5, 10 \rangle \\ \overrightarrow{v}^{\uparrow} = \langle 3, 4, 6, 6 \rangle \end{cases}$
- We compare the sorted vectors lexicographically: $\overrightarrow{u}^{\uparrow} \prec_{lexico} \overrightarrow{v}^{\uparrow}$

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

## Leximin SWO

Let $\overrightarrow{x}$ be a vector. We write $\overrightarrow{x^{\uparrow}}$ the sorted version of $\overrightarrow{x}$.
$\overrightarrow{u} \succ_{leximin} \overrightarrow{v} \Leftrightarrow \exists k$ such that $\forall i \leq k,\ u_i^{\uparrow} = v_i^{\uparrow}$ and $u_{k+1}^{\uparrow} > v_{k+1}^{\uparrow}$.
**In other words, a lexicographic comparison on the sorted vectors.**

## Features

It takes all agents into account, in the order of their satisfaction level $\rightsquigarrow$ natural inclination to fairness.
It both refines the order induced by the egalitarian SWO and the Pareto order.

# Classical Social Welfare Orderings

- Classical utilitarian SWO.
- Egalitarian SWO.
- Leximin SWO.

## Leximin SWO

Let $\overrightarrow{x}$ be a vector. We write $\overrightarrow{x^{\uparrow}}$ the sorted version of $\overrightarrow{x}$.
$\overrightarrow{u} \succ_{leximin} \overrightarrow{v} \Leftrightarrow \exists k$ such that $\forall i \leq k,\ u_i^{\uparrow} = v_i^{\uparrow}$ and $u_{k+1}^{\uparrow} > v_{k+1}^{\uparrow}$.
**In other words, a lexicographic comparison on the sorted vectors.**

## Leximin SWO and Pareto-efficiency

$\langle 1, 1, 1, 1 \rangle \prec \langle 1000, 1, 1000, 1000 \rangle$ (the second value of the ordered vectors is discriminatory).

# The Constraint Satisfaction Problem

## Classical CSP

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.
*Is there a complete consistent instantiation $v$ of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ ?*

## CSP with objective variable

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and an objective variable $o \in \mathcal{X}$, such that $d_o \subset \mathbb{N}$.
*What is the maximal value $\alpha$ of $d_o$ such that there is a complete consistent instantiation $\widehat{v}$ with $\widehat{v}(o) = \alpha$ ?*

## Leximin-CSP (as a multi-objective CSP)

**Given :** A constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a vector of variables $\overrightarrow{u} = \langle u_1, \ldots, u_n \rangle$ ($\forall i,\ u_i \in \mathcal{X}$ and $d_{u_i} \in \mathbb{N}$) called **objective vector**.
*What is the leximin-optimal vector $\langle \alpha_1, \ldots, \alpha_n \rangle$ of $\langle d_{u_1}, \ldots, d_{u_n} \rangle$ such that there is a complete consistent instantiation $\widehat{v}$ with $\widehat{v}(u_i) = \alpha_i$ forall $i$ ?*

# Outline

# An alternative definition of sorting

## Proposition

$\langle y_1, \ldots, y_n \rangle$ is the permutation of $\langle u_1, \ldots, u_n \rangle$ sorted in non-decreasing order if and only if:

- $y_1$ is the maximal value such that all the $u_i$ are g.e.q than $y_1$;
- $y_2$ is the maximal value such that at least $n-1$ values among the $u_i$ are g.e.q than $y_2$ and all the $u_i$ are g.e.q than $y_1$;

$\vdots$

- $y_n$ is the maximal value such that at least 1 value among the $u_i$ is g.e.q than $y_n$, at least 2 values among the $u_i$ are g.e.q than $y_{n-1}$, $\ldots$, and all the $u_i$ are g.e.q than $y_1$.

# An alternative definition of sorting

## Proposition

$\langle y_1, \ldots, y_n \rangle$ is the permutation of $\langle u_1, \ldots, u_n \rangle$ sorted in non-decreasing order if and only if:

- $y_1$ is the maximal value such that all the $u_i$ are g.e.q than $y_1$;
- $y_2$ is the maximal value such that at least $n-1$ values among the $u_i$ are g.e.q than $y_2$ and all the $u_i$ are g.e.q than $y_1$;

$\vdots$

- $y_n$ is the maximal value such that at least 1 value among the $u_i$ is g.e.q than $y_n$, at least 2 values among the $u_i$ are g.e.q than $y_{n-1}$, $\ldots$, and all the $u_i$ are g.e.q than $y_1$.

# An alternative definition of sorting

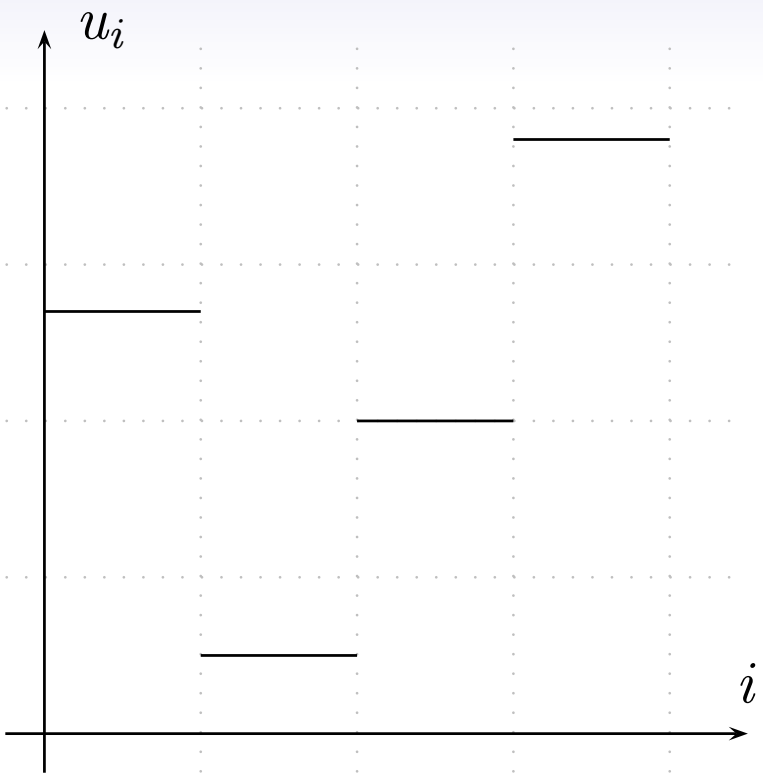
## Proposition

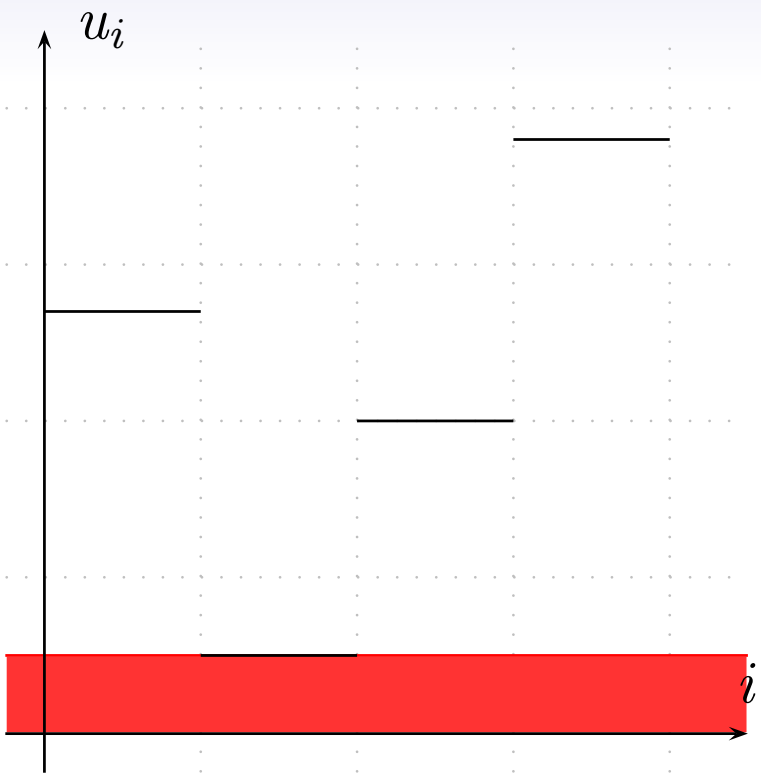
$\langle y_1, \ldots, y_n \rangle$ is the permutation of $\langle u_1, \ldots, u_n \rangle$ sorted in non-decreasing order if and only if:

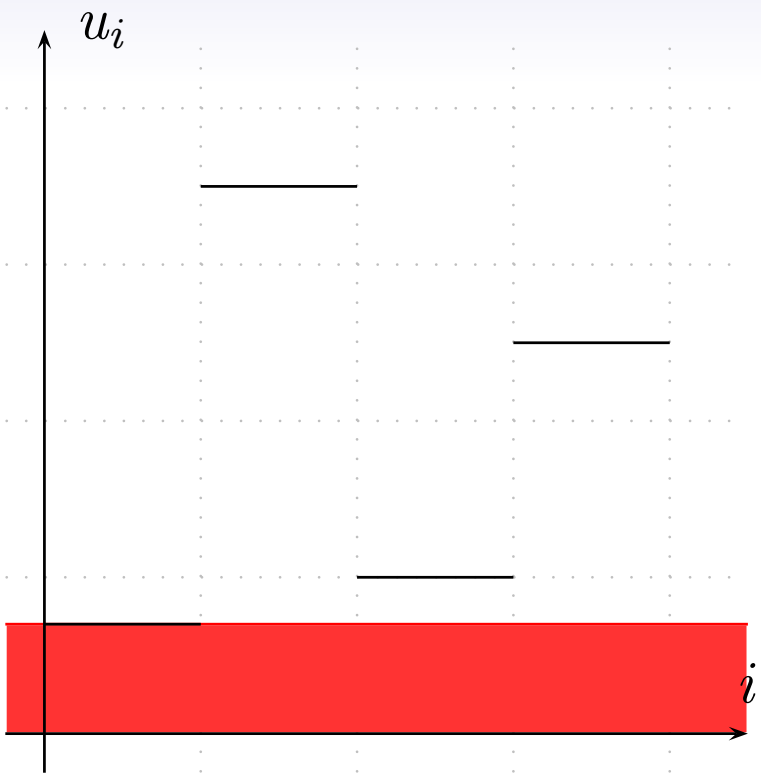
- $y_1$ is the maximal value such that all the $u_i$ are g.e.q than $y_1$;
- $y_2$ is the maximal value such that at least $n-1$ values among the $u_i$ are g.e.q than $y_2$ and all the $u_i$ are g.e.q than $y_1$;

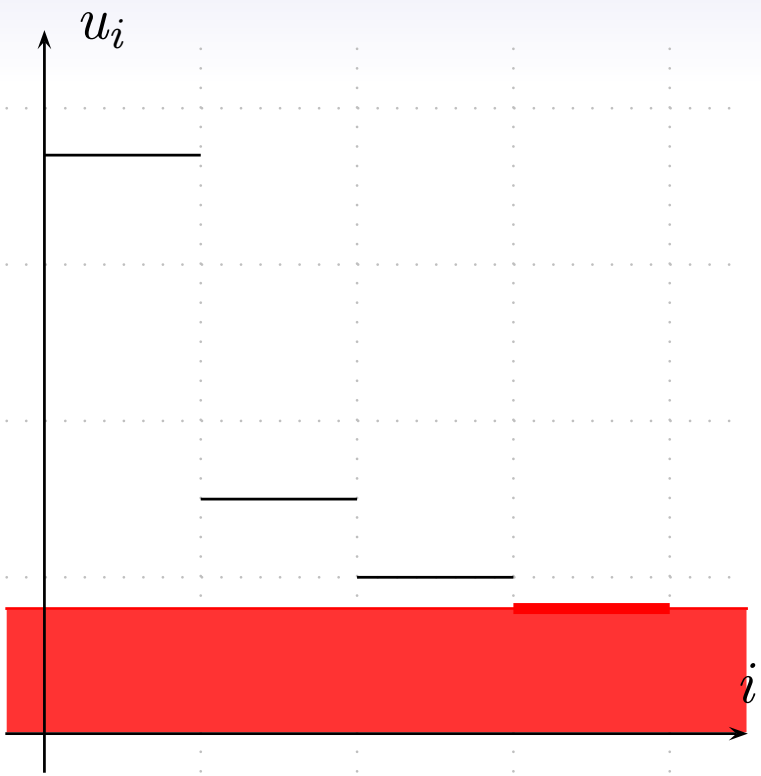
$$\vdots$$

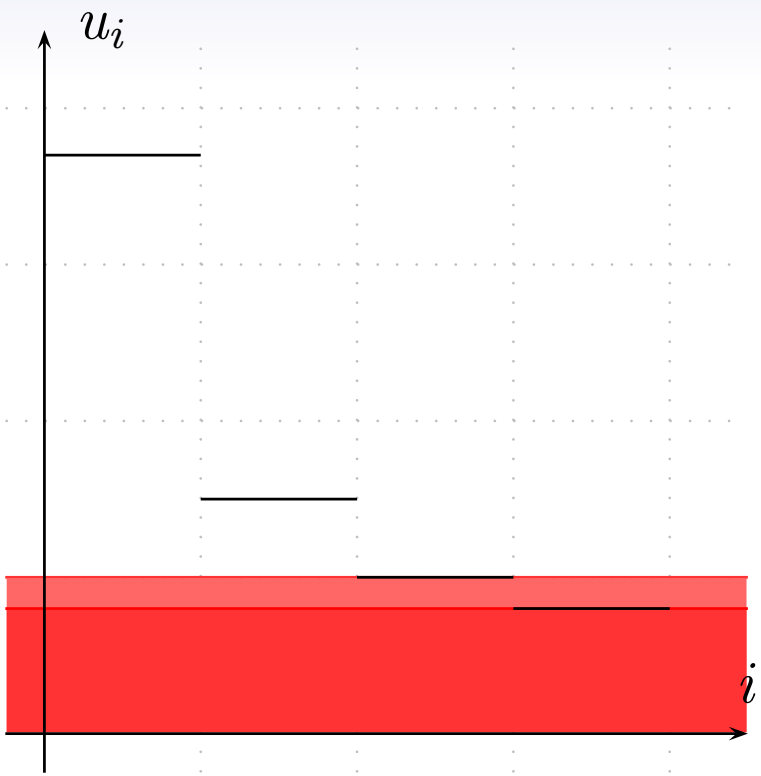
- $y_n$ is the maximal value such that at least 1 value among the $u_i$ is g.e.q than $y_n$, at least 2 values among the $u_i$ are g.e.q than $y_{n-1}$, $\ldots$, and all the $u_i$ are g.e.q than $y_1$.

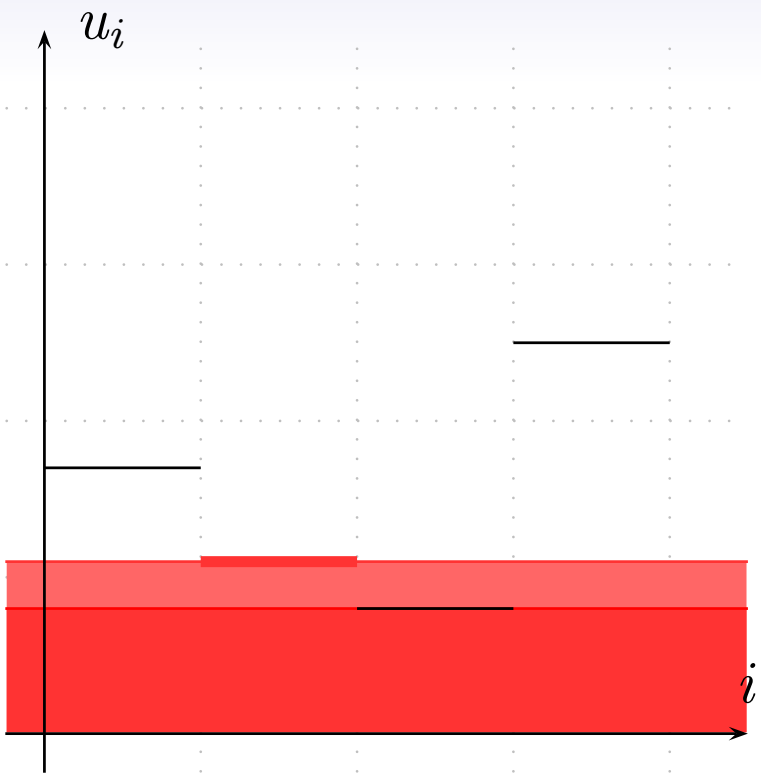

$u_i$
$i$

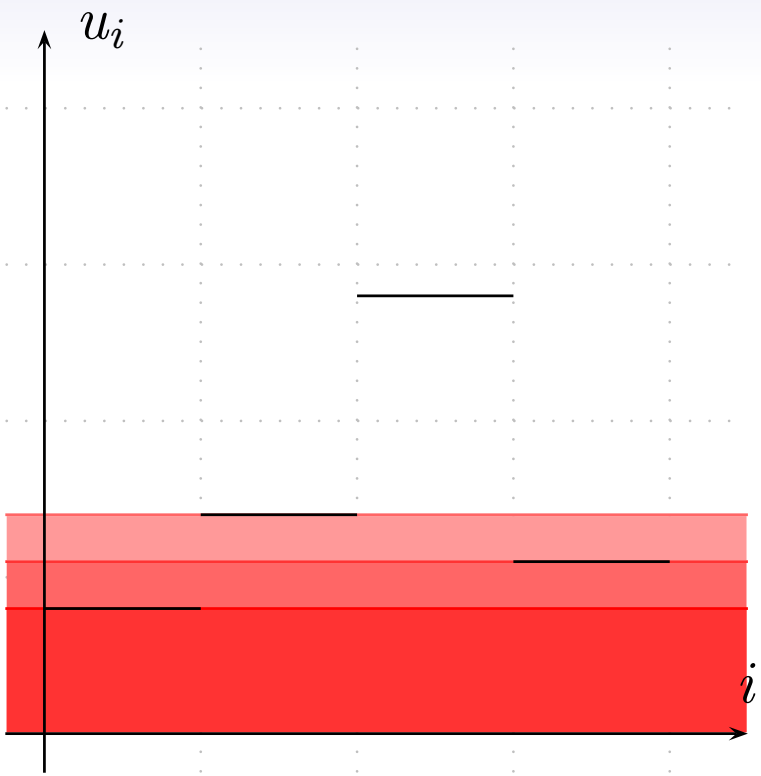

$u_i$
$i$

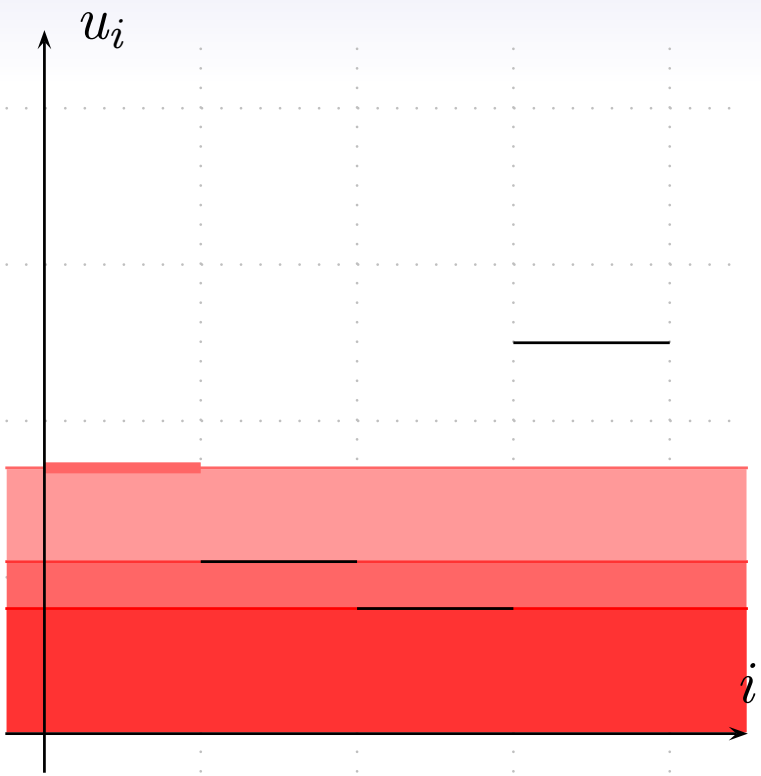

$u_i$
$i$

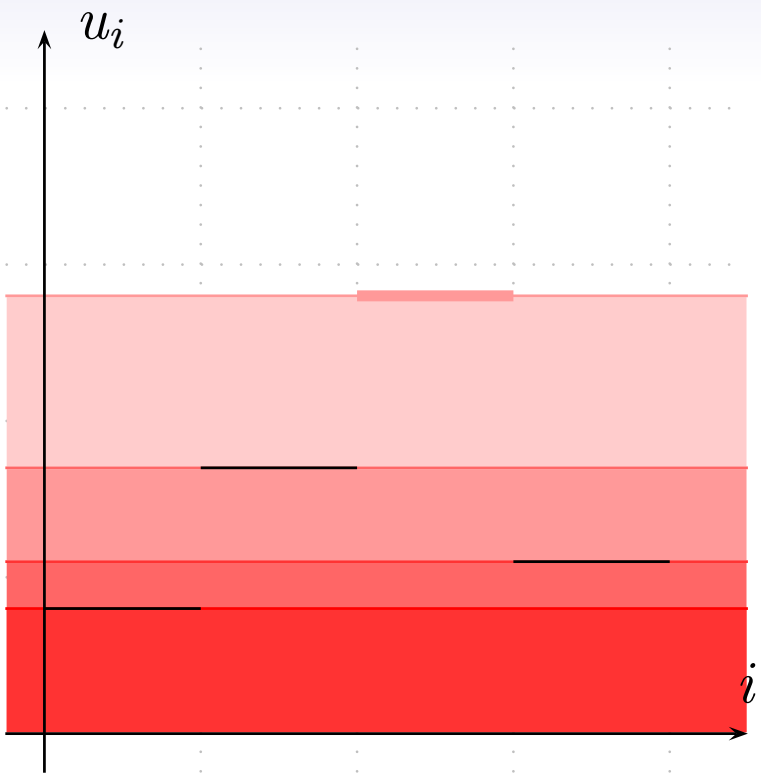

$u_i$
$i$


$u_i$
$i$


$u_i$
$i$


$u_i$
$i$


$u_i$
$i$


$u_i$
$i$

# The meta-constraint AtLeast

The algorithm is based on the meta-constraint **AtLeast**

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

Let $\Gamma$ be a set of $p$ constraints, and $k \in [\![1, p]\!]$ be an integer. The meta-constraint $\mathbf{AtLeast}(\Gamma, k)$ is the constraint that holds on the union of the scopes of the constraints in $\Gamma$, and that allows a tuple if and only if at least $k$ constraints from $\Gamma$ are satisfied.

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

# Algorithm description

**Input:** $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle u_1, \ldots, u_n \rangle \in \mathcal{X}^n$ objective variables (utilities of agents)
**Output:** The leximin-optimal instantiation or "Inconsistent"

## Algorithm

**if** $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **"Inconsistent" then return "Inconsistent"**

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

**for** $i \leftarrow 1$ **to** $n$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;
  $\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;
  $d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;
**end**
**return** $\widehat{v}_{\downarrow \mathcal{X}}$

# Algorithm description

**Input:** $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle u_1, \ldots, u_n \rangle \in \mathcal{X}^n$ objective variables (utilities of agents)
**Output:** The leximin-optimal instantiation or "Inconsistent"

## Algorithm

**if** $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **"Inconsistent" then return "Inconsistent"**

**Introduction of the variables $\mathbf{y_i}$**

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

**for** $i \leftarrow 1$ **to** $n$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;
  $\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;
  $d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;
**end**
**return** $\widehat{v}_{\downarrow \mathcal{X}}$

# Algorithm description

**Input:** $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle u_1, \ldots, u_n \rangle \in \mathcal{X}^n$ objective variables (utilities of agents)
**Output:** The leximin-optimal instantiation or "Inconsistent"

## Algorithm

**if** $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **"Inconsistent" then return "Inconsistent"**

**Introduction of the variables $\mathbf{y_i}$**

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

**Introduction of a constraint on $\mathbf{y_i}$**

**for** $i \leftarrow 1$ **to** $n$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;
  $\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$;
  $d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;
**end**
**return** $\widehat{v}_{\downarrow \mathcal{X}}$

# Algorithm description

**Input:** $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle u_1, \ldots, u_n \rangle \in \mathcal{X}^n$ objective variables (utilities of agents)
**Output:** The leximin-optimal instantiation or "Inconsistent"

## Algorithm

**if** $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **"Inconsistent" then return "Inconsistent"**

**Introduction of the variables $\mathbf{y_i}$**

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

**Introduction of a constraint on $\mathbf{y_i}$**

**for** $i \leftarrow 1$ **to** $n$ **do**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{AtLeast}(\{u_1 \geq y_i, \ldots, u_n \geq y_i\}, n - i + 1)\}$;
  $\widehat{v} \leftarrow \mathbf{maximize}(y_i, (\mathcal{X}, \mathcal{D}, \mathcal{C}))$; **Computation of $\mathbf{y_i}$**
  $d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$;
**end**
**return** $\widehat{v}_{\downarrow \mathcal{X}}$

# Algorithm description

**Input:** $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle u_1, \ldots, u_n \rangle \in \mathcal{X}^n$ objective variables (utilities of agents)
**Output:** The leximin-optimal instantiation or "Inconsistent"

## Algorithm

**if** $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ **"Inconsistent" then return "Inconsistent"**

**Introduction of the variables $\mathbf{y_i}$**

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

**Introduction of a constraint on $\mathbf{y_i}$**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

$d_{y_i} \leftarrow \{\widehat{v}(y_i)\}$; **Updating of the domains**

**end**

**return** $\widehat{v}_{\downarrow \mathcal{X}}$

# Running of the algorithm on an example

## A sharing problem

- An allocation problem with 3 agents and 3 objects.
- Constraint: the allocation has to be a matching (*i.e.* one different object per agent).
- The utility functions of the agents are defined by a set of weights $w(a_i, o_j)$, the utility function of the agent $a_i$ being $u_i = \sum_{o_j | a_i \leftarrow o_j} w(a_i, o_j)$.
- The weights are the following:

| objects \ agents | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|
| $o_1$ | 3 | 3 | 3 |
| $o_2$ | 5 | 9 | 7 |
| $o_3$ | 7 | 8 | 1 |

# Running of the algorithm on an example

**Algorithm :**

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

**Initial constraint network:**

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

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

$\mathcal{C}' = \{\mathbf{alldifferent}(\{a_1, a_2, a_3\}), u_i = \sum_{o_j} (a_i = o_j)\ \forall i\}$

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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} |

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

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

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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}** |

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

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

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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} |

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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}** |

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

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

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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}** |

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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**} |

# Running of the algorithm on an example

**Algorithm :**

**for** $i \leftarrow 1$ **to** $n$ **do**

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

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

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

**end**

| $a_1$ | $a_2$ | $a_3$ | $u_1$ | $u_2$ | $u_3$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|---|---|
| $o_1$ | $o_2$ | $o_3$ | 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** |

# A second algorithm

We introduce a new constraint:

**Constraint Leximin**

Let $\overrightarrow{x}$ be a vector of variables and $\overrightarrow{\lambda}$ be a vector of integers. The constraint $\mathbf{Leximin}(\overrightarrow{\lambda}, \overrightarrow{x})$ concerns every variables belonging to $\overrightarrow{x}$, and allows a tuple $\overrightarrow{v(x)}$ if and only if $\overrightarrow{\lambda} \prec_{leximin} \overrightarrow{v(x)}$.

This constraint is based on the multiset ordering constraint, introduced in [Frisch et al., 2003] (algorithm to enforce GAC in $O(n \log(n))$).

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

# Description of the algorithm

## Algorithm

$v \leftarrow \mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C})$
**while** $v \neq$ **"Inconsistent"** **do**
  $\widehat{v} \leftarrow v$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{Leximin}(\widehat{v}(\overrightarrow{u}), \overrightarrow{u})\}$
  $v \leftarrow \mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C})$
**end**
**if** $\widehat{v} \neq \mathbf{null}$ **then** **return** $\widehat{v}$
**else** **return** **"Inconsistent"**

# Outline

# Implementation

## Implementation of the algorithms:

The two algorithms have been implemented using Choco[F. Laburthe and the OCRE project team, 2000] in Java, and the first one has also been implemented using CPLEX.

**Remark**

Our specific constraint **AtLeast** can be encoded with a set of linear constraints: $\mathbf{AtLeast}(\{x_1 \geq y, \ldots, x_n \geq y\}, k) \Leftrightarrow \{x_1 + \delta_1 \overline{y} \geq y, \ldots, x_n + \delta_n \overline{y} \geq y, \sum_{i=1}^{n} \delta_i \leq n - k\}$, with $\{\delta_1, \ldots, \delta_n\}$ 0–1 variables.

**F. Laburthe and the OCRE project team (2000).**
CHOCO: Implementing a CP kernel.
In *Proceedings of TRICKS'2000, Workshop on techniques for implementing Constraint Programming systems*, Singapore.
http://sourceforge.net/projects/choco.

# Implementation

## Implementation of the algorithms:

The two algorithms have been implemented using Choco[F. Laburthe and the OCRE project team, 2000] in Java, and the first one has also been implemented using CPLEX.

**Remark**

Our specific constraint **AtLeast** can be encoded with a set of linear constraints: $\mathbf{AtLeast}(\{x_1 \geq y, \ldots, x_n \geq y\}, k) \Leftrightarrow$ $\{x_1 + \delta_1 \overline{y} \geq y, \ldots, x_n + \delta_n \overline{y} \geq y, \sum_{i=1}^{n} \delta_i \leq n - k\}$, with $\{\delta_1, \ldots, \delta_n\}$ 0–1 variables.

**F. Laburthe and the OCRE project team (2000).**
CHOCO: Implementing a CP kernel.
In *Proceedings of TRICKS'2000, Workshop on techniques for implementing Constraint Programming systems*, Singapore.
http://sourceforge.net/projects/choco.

# A real world application

The algorithms have been tested using a (very) simplified model extracted from a real world application:

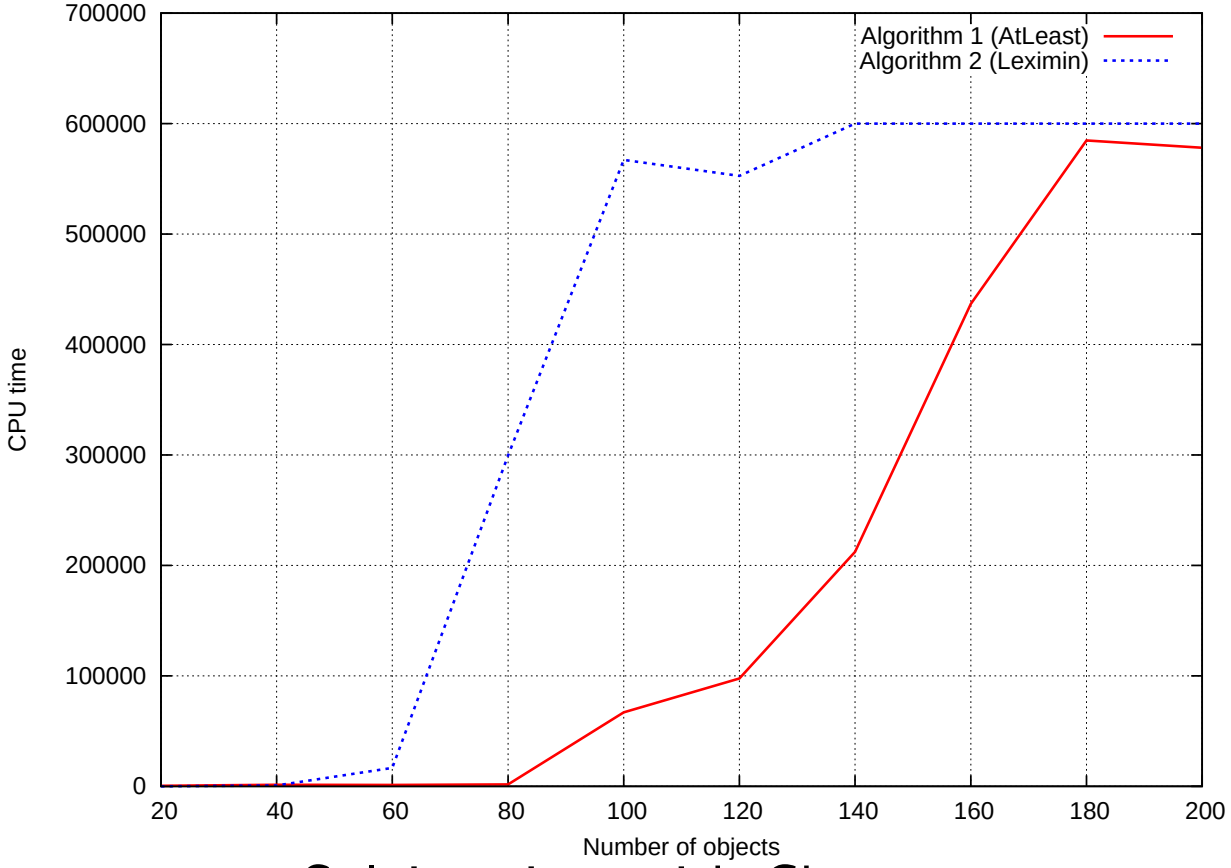
- Fair share of a constellation of Earth Observation Satellites cofunded by several countries.
- Our simplified model is a multiagent resource allocation with consumption and volume resources.

© CNES - Mars 2003 / illustration Pierre CARRIL

**Instance generator:**
We implemented a random instance generator for our simplified problem.
Available online: `http://www.cert.fr/dcsd/THESES/sbouveret/benchmark`

# Some results

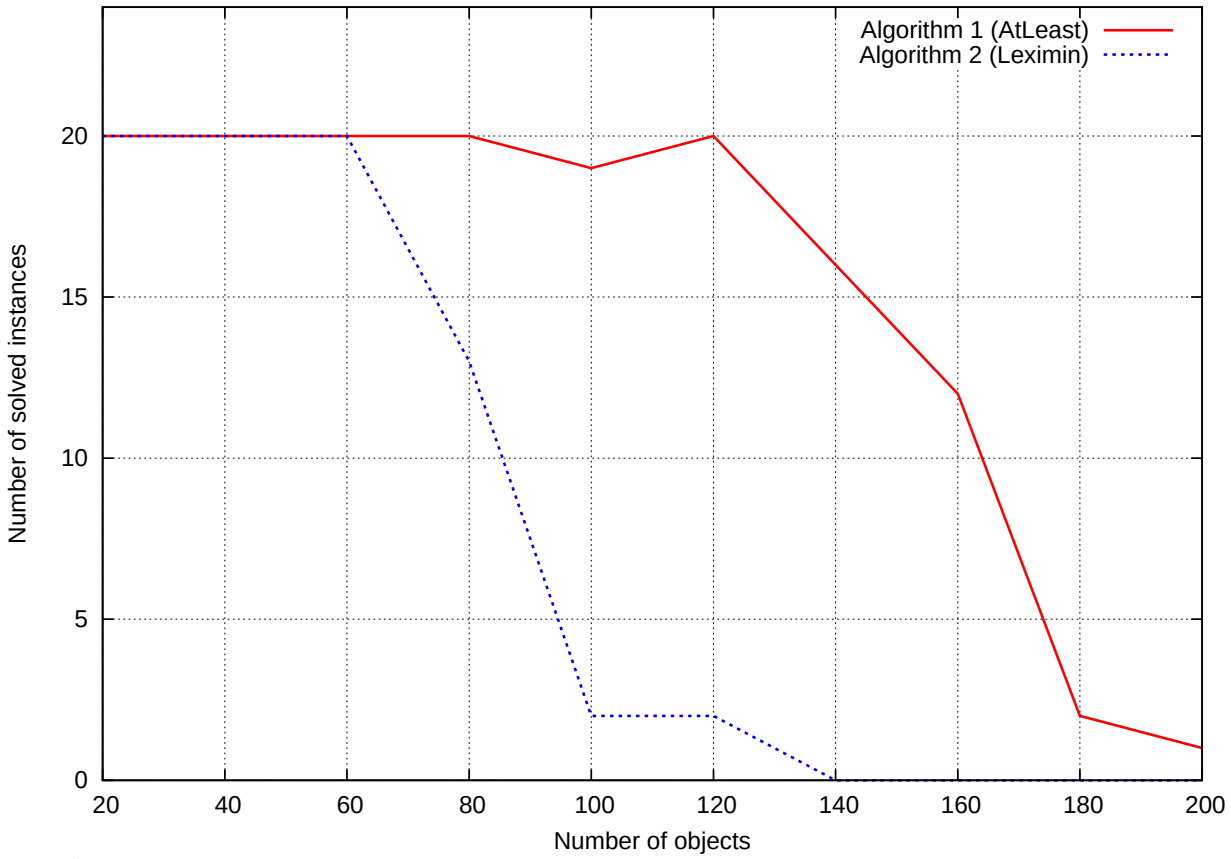
**Instance:** 4 agents, $x$ objects and volume constraints allowing 10 objects out of 20 consecutive ones, weights uniformly distributed.




Solving time with Choco.

# Some results

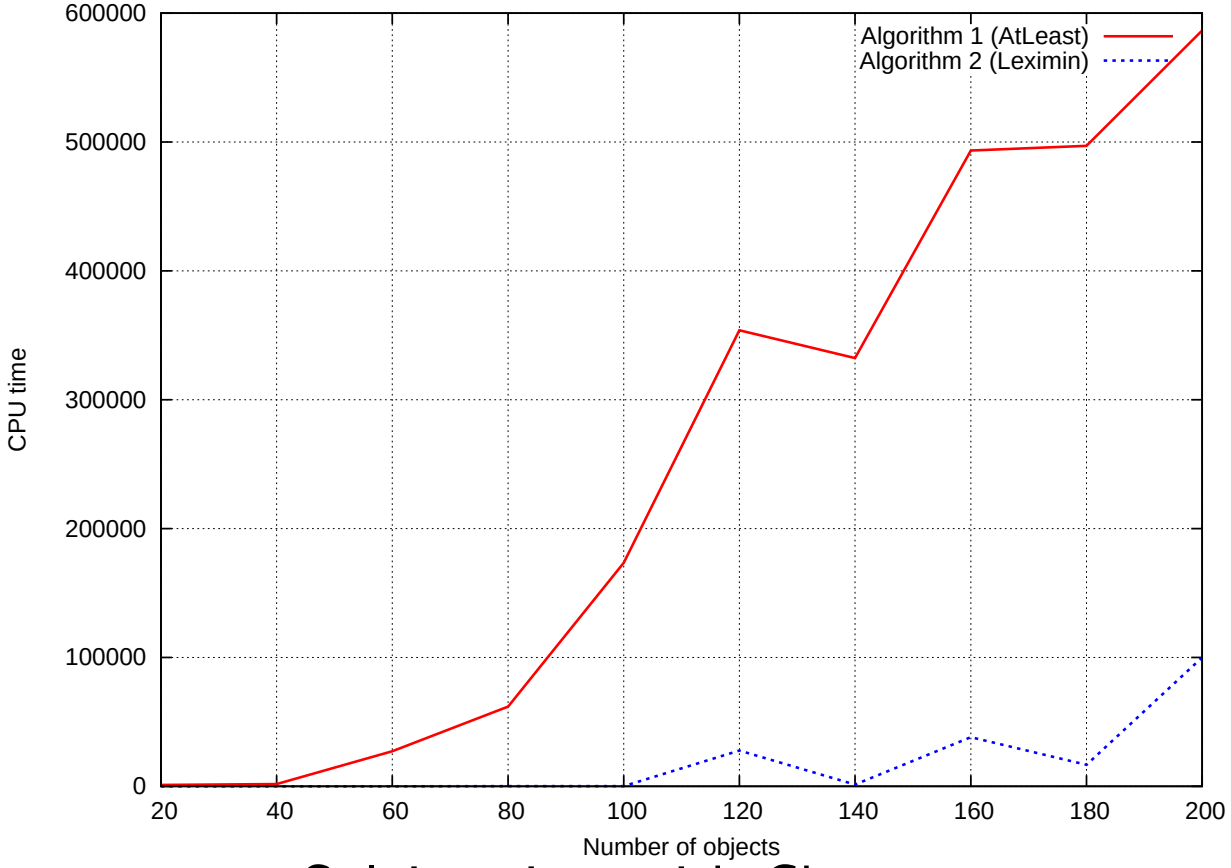
**Instance:** 4 agents, $x$ objects and volume constraints allowing 10 objects out of 20 consecutive ones, weights uniformly distributed.




Number of instances solved in less than 10 minutes with Choco.

# Some results

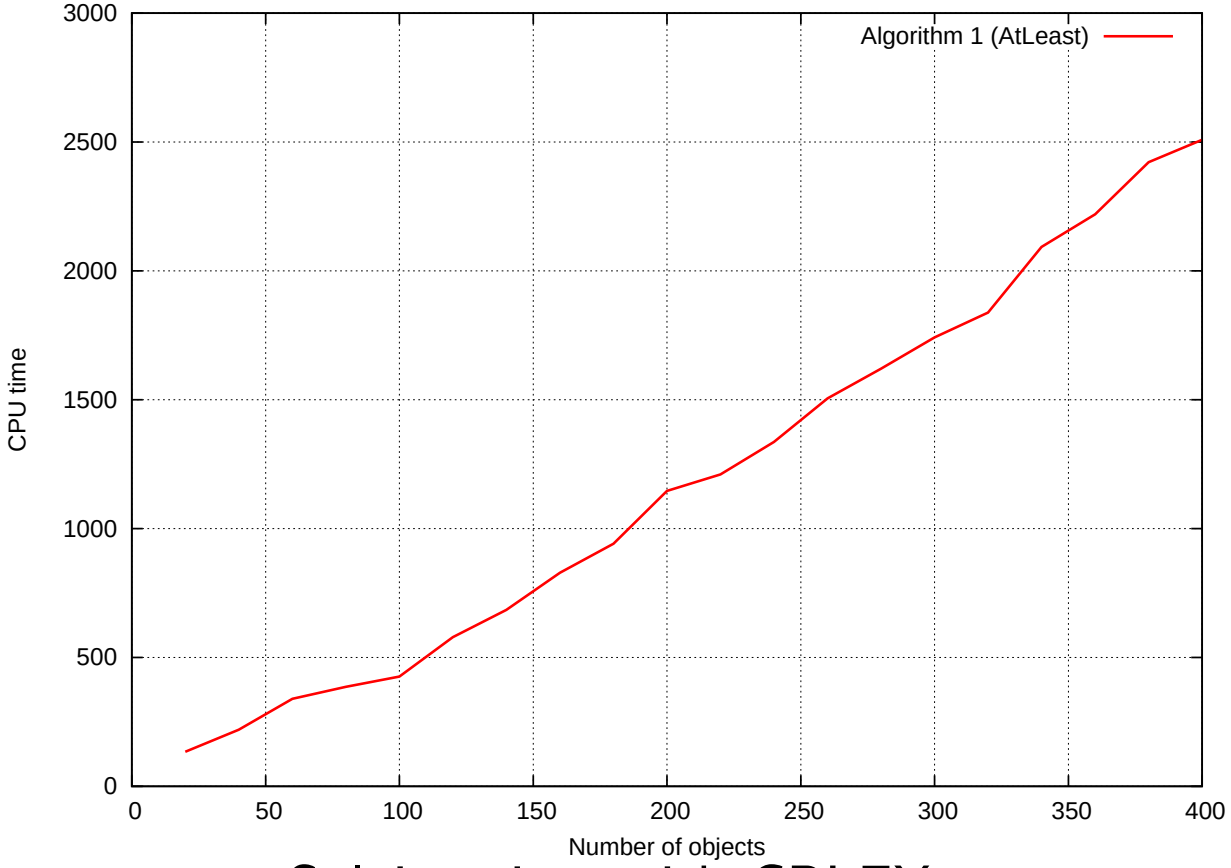
**Instance:** 10 agents, $x$ objects and volume constraints allowing 10 objects out of 20 consecutive ones, weights non-uniformly distributed.




Solving time with Choco.

# Some results

**Instance:** 4 agents, $x$ objects and volume constraints allowing 10 objects out of 20 consecutive ones, weights uniformly distributed.




Solving time with CPLEX.

# Summary of the results

- None of the two algorithms clearly outperforms the other one.
- The second algorithm seems to be better on the instance with more agents.
- CPLEX's solving times are far lower than Choco's ones, but CPLEX may make some computation errors (probably due to approximations).

It thus suggests that the best algorithm to use depends on the kind of instance to solve (*e.g.* number of agents).

# Summary

- **Studied problem:** computation of a leximin-optimal allocation of a constraint network.
- **Justification :** the leximin preorder ensures some interesting properties of fairness and efficiency for collective decision making problems.
- **Algorithms :** introduction of two algorithms (the first one being original) based on the CP framework.
- **Implementation :** implementation and testing of the algorithms in Java with Choco (and CPLEX for the first one).
- **Results :** None of the two algorithms is clearly better than the other one.

# Summary

- **Studied problem:** computation of a leximin-optimal allocation of a constraint network.
- **Justification :** the leximin preorder ensures some interesting properties of fairness and efficiency for collective decision making problems.
- **Algorithms :** introduction of two algorithms (the first one being original) based on the CP framework.
- **Implementation :** implementation and testing of the algorithms in Java with Choco (and CPLEX for the first one).
- **Results :** None of the two algorithms is clearly better than the other one.

# Summary

- **Studied problem:** computation of a leximin-optimal allocation of a constraint network.
- **Justification :** the leximin preorder ensures some interesting properties of fairness and efficiency for collective decision making problems.
- **Algorithms :** introduction of two algorithms (the first one being original) based on the CP framework.
- **Implementation :** implementation and testing of the algorithms in Java with Choco (and CPLEX for the first one).
- **Results :** None of the two algorithms is clearly better than the other one.

# Summary

- **Studied problem:** computation of a leximin-optimal allocation of a constraint network.
- **Justification :** the leximin preorder ensures some interesting properties of fairness and efficiency for collective decision making problems.
- **Algorithms :** introduction of two algorithms (the first one being original) based on the CP framework.
- **Implementation :** implementation and testing of the algorithms in Java with Choco (and CPLEX for the first one).
- **Results :** None of the two algorithms is clearly better than the other one.

# Summary

- **Studied problem:** computation of a leximin-optimal allocation of a constraint network.
- **Justification :** the leximin preorder ensures some interesting properties of fairness and efficiency for collective decision making problems.
- **Algorithms :** introduction of two algorithms (the first one being original) based on the CP framework.
- **Implementation :** implementation and testing of the algorithms in Java with Choco (and CPLEX for the first one).
- **Results :** None of the two algorithms is clearly better than the other one.

# Current and future work

- **Improvement of the algorithms:**
  - Working on specific heuristics.
  - Implementing filtering algorithms for our specific **AtLeast** constraint.
- **Implementing new algorithms:**
  - An algorithm using the sorting global constraint [Bleuzen-Guernalec and Colmerauer, 1997].
  - The algorithm from [Dubois and Fortemps, 1999].

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

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

# Current and future work

- **Improvement of the algorithms:**
  - Working on specific heuristics.
  - Implementing filtering algorithms for our specific **AtLeast** constraint.
- **Implementing new algorithms:**
  - An algorithm using the sorting global constraint [Bleuzen-Guernalec and Colmerauer, 1997].
  - The algorithm from [Dubois and Fortemps, 1999].

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

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

# Current and future work

- **Improvement of the algorithms:**
  - Working on specific heuristics.
  - Implementing filtering algorithms for our specific **AtLeast** constraint.
- **Implementing new algorithms:**
  - An algorithm using the sorting global constraint [Bleuzen-Guernalec and Colmerauer, 1997].
  - The algorithm from [Dubois and Fortemps, 1999].

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

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

# Current and future work

- **Improvement of the algorithms:**
  - Working on specific heuristics.
  - Implementing filtering algorithms for our specific **AtLeast** constraint.
- **Implementing new algorithms:**
  - An algorithm using the sorting global constraint [Bleuzen-Guernalec and Colmerauer, 1997].
  - The algorithm from [Dubois and Fortemps, 1999].

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

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

# Current and future work

- **Improvement of the algorithms:**
  - Working on specific heuristics.
  - Implementing filtering algorithms for our specific **AtLeast** constraint.
- **Implementing new algorithms:**
  - An algorithm using the sorting global constraint [Bleuzen-Guernalec and Colmerauer, 1997].
  - The algorithm from [Dubois and Fortemps, 1999].

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

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

# Current and future work

- **Improvement of the algorithms:**
  - Working on specific heuristics.
  - Implementing filtering algorithms for our specific **AtLeast** constraint.
- **Implementing new algorithms:**
  - An algorithm using the sorting global constraint [Bleuzen-Guernalec and Colmerauer, 1997].
  - The algorithm from [Dubois and Fortemps, 1999].

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

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

# Current and future work

- **Possible extensions:**
  - More general and softer modeling of fairness (*e.g.* OWA).
  - Applying the algorithms to other practical fields and applications.

# Current and future work

- **Possible extensions:**
  - More general and softer modeling of fairness (*e.g.* OWA).
  - Applying the algorithms to other practical fields and applications.

# Current and future work

- **Possible extensions:**
  - More general and softer modeling of fairness (*e.g.* OWA).
  - Applying the algorithms to other practical fields and applications.

# The end.

Contacts:
sylvain.bouveret@cert.fr
michel.lemaitre@cert.fr

Random instance generator available online:
http://www.cert.fr/dcsd/THESES/sbouveret/benchmark