New Constraint Programming Approaches For The Computation Of Leximin-Optimal Solutions In Constraint Networks

Sylvain Bouveret and Michel Lemaître

Office National d'Études et de Recherches Aérospatiales (ONERA)
Centre National d'Études Spatiales (CNES)
Institut de Recherche en Informatique de Toulouse (IRIT)
Toulouse – France

20th International Joint Conference on Artificial Intelligence
Hyderabad, January 11, 2007

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.

. . . are combinatorial collective decision making problems under admissibility constraints, involving directly or indirectly the concept of **fairness**.

Initial question

How can we handle fairness requirements in this kind of constraint satisfaction problems ?

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.

. . . are combinatorial collective decision making problems under admissibility constraints, involving directly or indirectly the concept of **fairness**.

Initial question

How can we handle fairness requirements in this kind of constraint satisfaction problems ?

Outline

Outline

Constraint networks

Constraint network [Montanari, 1974]

A constraint network is based on :

- a set of variables $\mathscr{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_p}\}$;
- a set of domains $\mathscr{D} = \{\mathscr{D}_{\mathbf{x_1}}, \ldots, \mathscr{D}_{\mathbf{x_p}}\}$;
- a set of constraints $\mathscr{C}$, with, for all $c \in \mathscr{C}$:
  - $X(c)$ 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 $(\mathscr{X}, \mathscr{D}, \mathscr{C})$.
Is there a complete consistent instantiation v of $(\mathscr{X}, \mathscr{D}, \mathscr{C})$ *?*

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

CSP with objective variable

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

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

The Constraint Satisfaction Problem

Classical CSP

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

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

CSP with objective variable

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

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

Fairness requirements ?

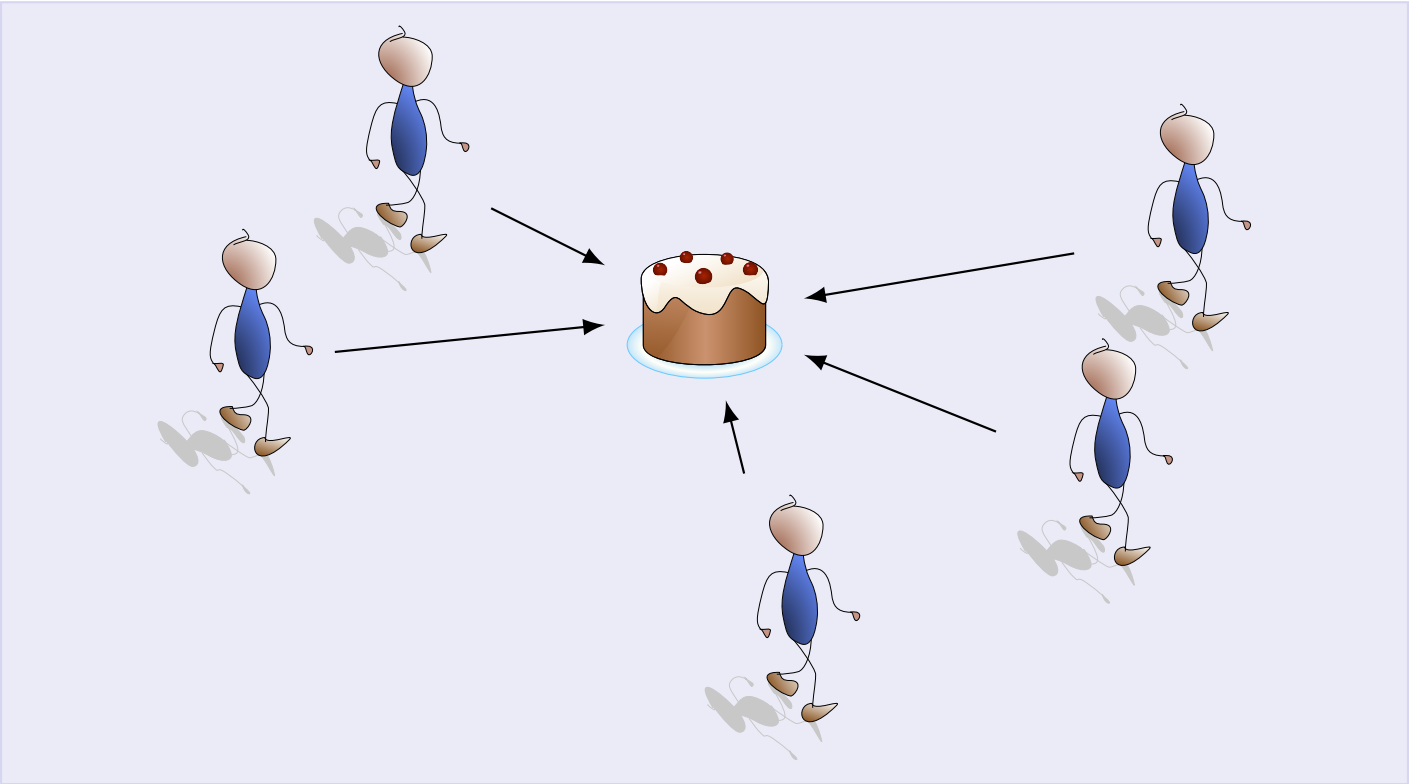
Hypotheses:

- The preferences are numerical.
- The quality of a solution is measured only on the basis of the agents utilities (**Welfarism**).

Fairness requirements ?

Hypotheses:

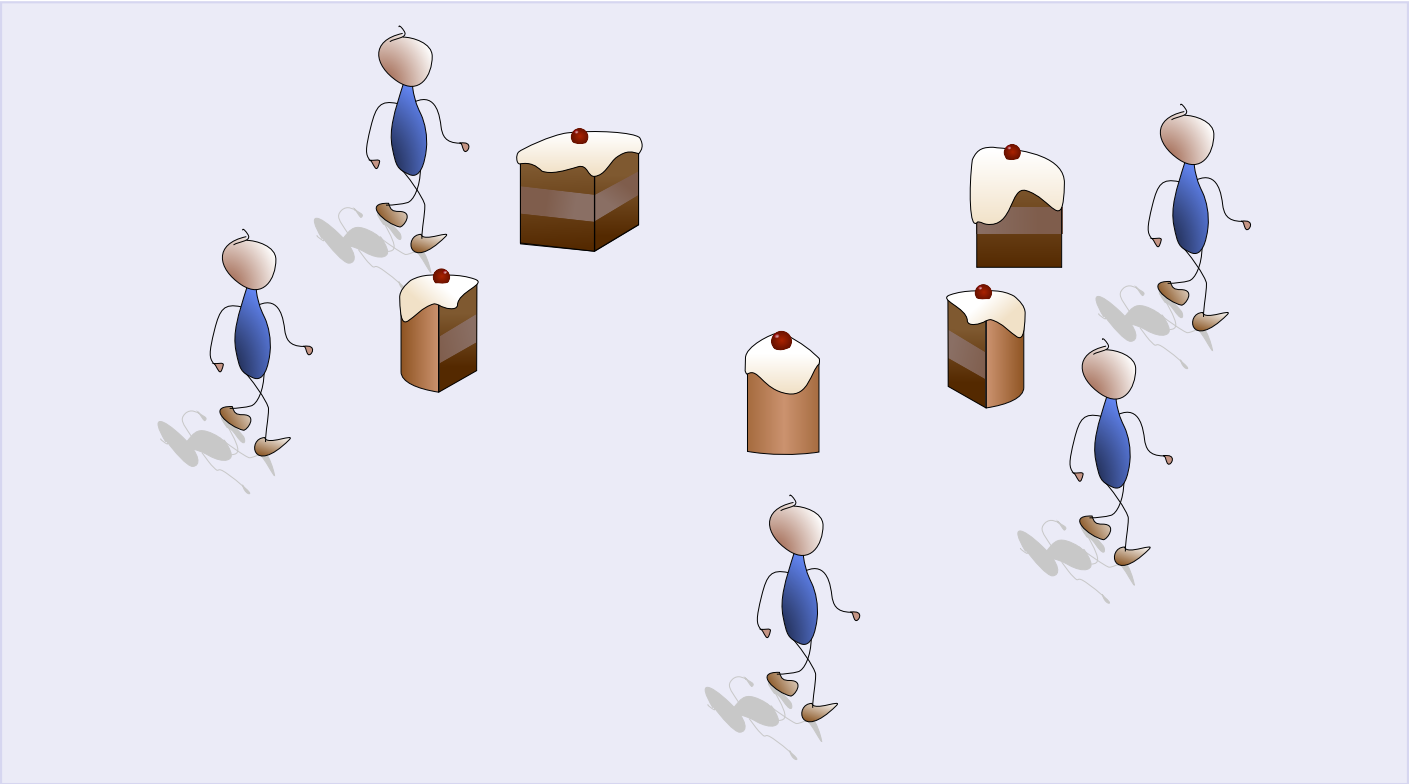
- The preferences are numerical.
- The quality of a solution is measured only on the basis of the agents utilities (**Welfarism**).



Fairness requirements ?

Hypotheses:

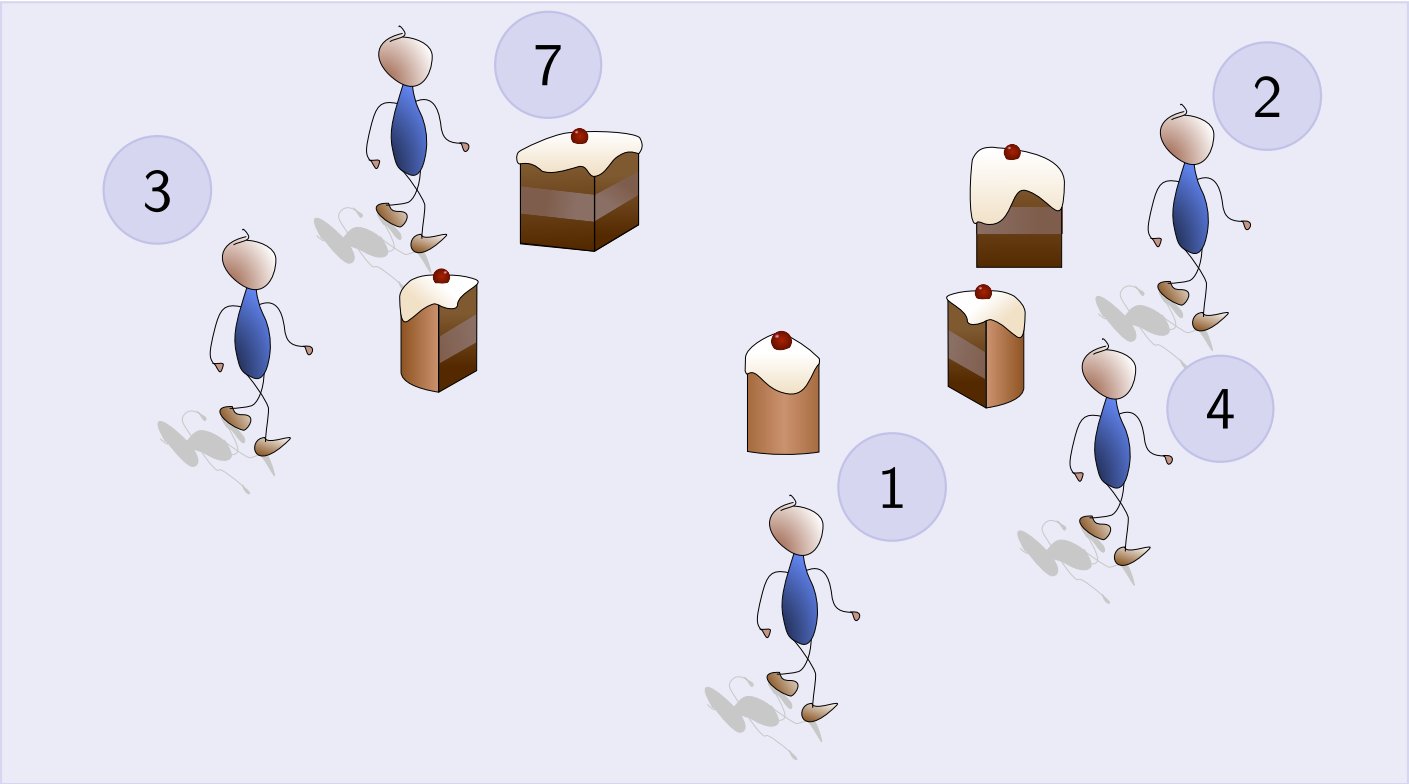
- The preferences are numerical.
- The quality of a solution is measured only on the basis of the agents utilities (**Welfarism**).



Fairness requirements ?

Hypotheses:

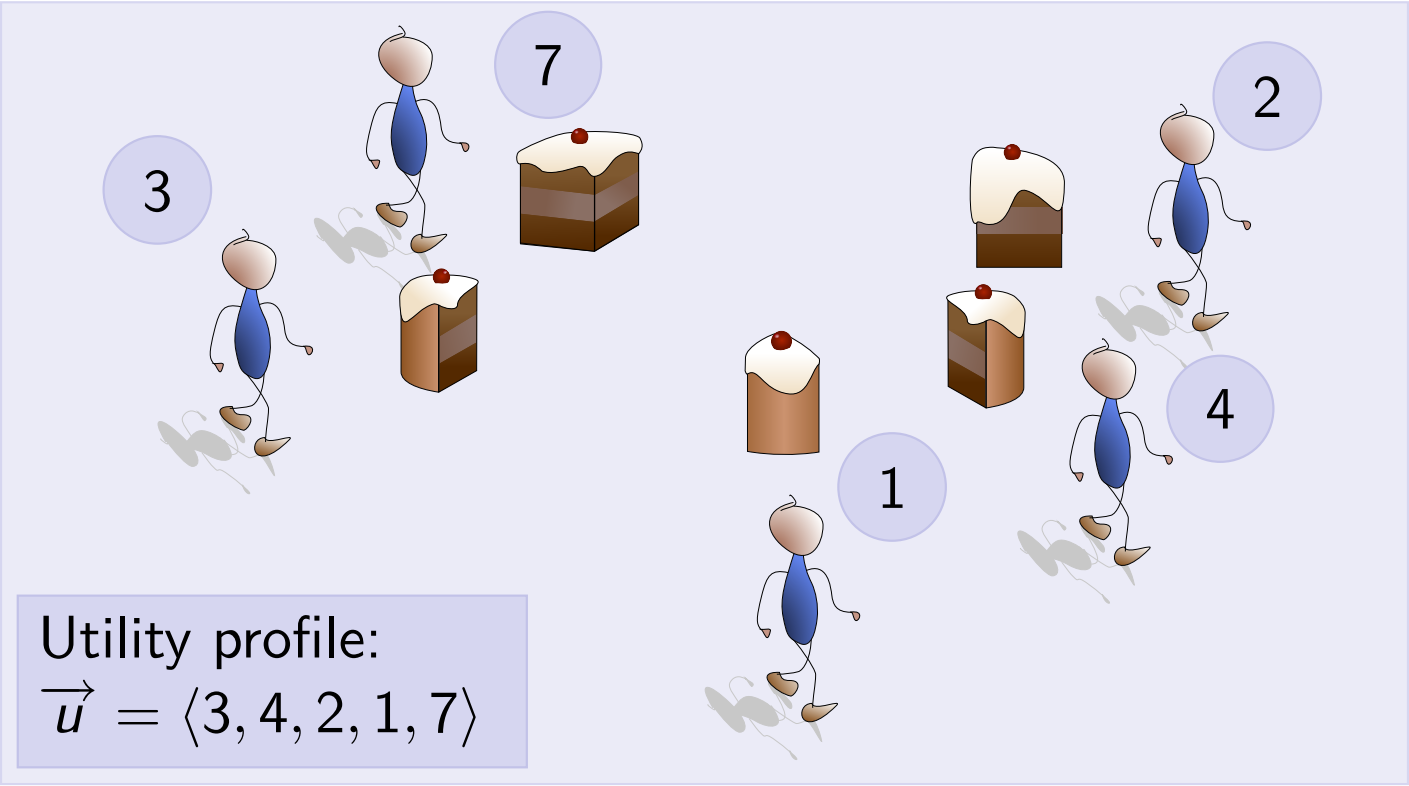
- The preferences are numerical.
- The quality of a solution is measured only on the basis of the agents utilities (**Welfarism**).



Fairness requirements ?

Hypotheses:

- The preferences are numerical.
- The quality of a solution is measured only on the basis of the agents utilities (**Welfarism**).



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 $(\mathscr{X}, \mathscr{D}, \mathscr{C})$.
Is there a complete consistent instantiation v of $(\mathscr{X}, \mathscr{D}, \mathscr{C})$?

CSP with objective variable

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

Leximin-CSP (as a multi-objective CSP)

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

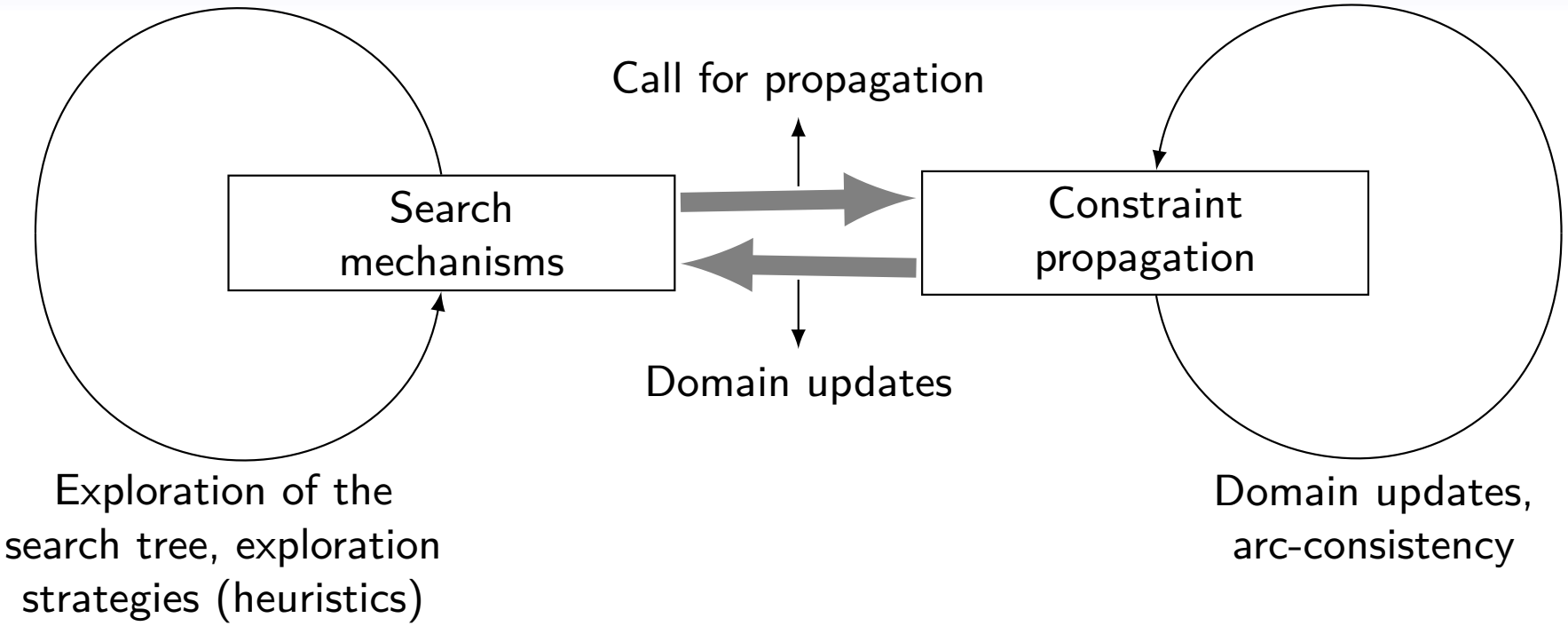
Outline

Constraint Programming

Constraint programming provides a flexible and efficient tool for implementing and solving CSPs.

- **Our approach:** use this tool as a "black box" for solving leximin-CSPs.
- **Aims:**
  - develop generic algorithms.
  - benefit from using of a powerful framework and of its algorithmics.

Constraint Programming



What we can do:

- Set up the problem (declare variables, domains, constraints).
- Implement new constraint propagation algorithms.
- Make calls to functions **solve** or **maximize** (black boxes).

Algorithm 1

Sort and conquer

Sorted version of the objective vector

Initial idea

Maximize the objective vector using the leximin preorder $\Leftrightarrow$ maximize the successive components of the **ordered** objective vector.

$\rightsquigarrow$ We have to introduce the sorted version of the objective vector:

- **A vector of variables** $(\mathbf{y_1}, \ldots, \mathbf{y_n})$.
- **A constraint Sort**$(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{y}})$ ([Mehlhorn and Thiel, 2000] (filtering in time $O(n \log(n))$).

Mehlhorn, K. and Thiel, S. (2000).
Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint.
In Dechter, R., editor, *Proc. of CP'00*, pages 306–319, Singapore.

Sorted version of the objective vector

Initial idea

Maximize the objective vector using the leximin preorder $\Leftrightarrow$ maximize the successive components of the **ordered** objective vector.

$\rightsquigarrow$ We have to introduce the sorted version of the objective vector:

- **A vector of variables** $(\mathbf{y_1}, \ldots, \mathbf{y_n})$.
- **A constraint Sort**$(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{y}})$ ([Mehlhorn and Thiel, 2000] (filtering in time $O(n \log(n))$).

1. Maximize $\mathbf{y_1}$: $\widehat{y_1}$.
2. Maximize $\mathbf{y_2}$ under the constraint $\mathbf{y_1} = \widehat{y_1}$: $\widehat{y_2}$.

$\vdots$

n. Maximize $\mathbf{y_n}$ under the constraints $\mathbf{y_1} = \widehat{y_1}, \ldots, \mathbf{y_{n-1}} = \widehat{y_{n-1}}$.

Sorted version of the objective vector

Initial idea

Maximize the objective vector using the leximin preorder $\Leftrightarrow$ maximize the successive components of the **ordered** objective vector.

$\rightsquigarrow$ We have to introduce the sorted version of the objective vector:

- **A vector of variables** $(\mathbf{y_1}, \ldots, \mathbf{y_n})$.
- **A constraint Sort**$(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{y}})$ ([Mehlhorn and Thiel, 2000] (filtering in time $O(n \log(n))$).

1. Maximize $\mathbf{y_1}$: $\widehat{y_1}$.
2. Maximize $\mathbf{y_2}$ under the constraint $\mathbf{y_1} = \widehat{y_1}$: $\widehat{y_2}$.

$\vdots$

n. Maximize $\mathbf{y_n}$ under the constraints $\mathbf{y_1} = \widehat{y_1}, \ldots, \mathbf{y_{n-1}} = \widehat{y_{n-1}}$.

Sorted version of the objective vector

Initial idea

Maximize the objective vector using the leximin preorder $\Leftrightarrow$ maximize the successive components of the **ordered** objective vector.

$\rightsquigarrow$ We have to introduce the sorted version of the objective vector:

- **A vector of variables** $(\mathbf{y_1}, \ldots, \mathbf{y_n})$.
- **A constraint Sort**$(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{y}})$ ([Mehlhorn and Thiel, 2000] (filtering in time $O(n \log(n))$).

1. Maximize $\mathbf{y_1}$: $\widehat{y_1}$.
2. Maximize $\mathbf{y_2}$ under the constraint $\mathbf{y_1} = \widehat{y_1}$: $\widehat{y_2}$.

$\vdots$

n. Maximize $\mathbf{y_n}$ under the constraints $\mathbf{y_1} = \widehat{y_1}, \ldots, \mathbf{y_{n-1}} = \widehat{y_{n-1}}$.

Sorted version of the objective vector

Initial idea

Maximize the objective vector using the leximin preorder $\Leftrightarrow$ maximize the successive components of the **ordered** objective vector.

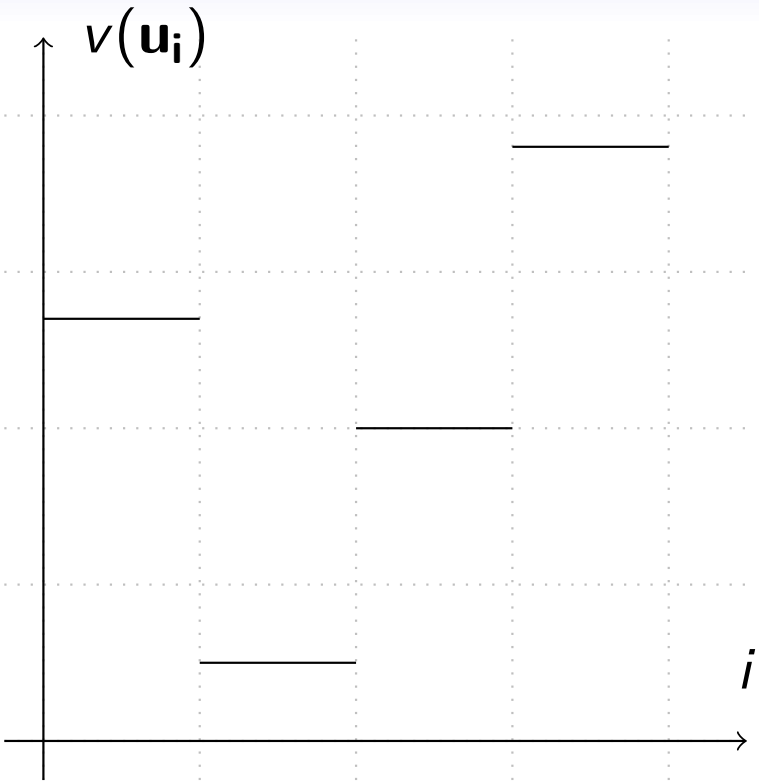
$\rightsquigarrow$ We have to introduce the sorted version of the objective vector:

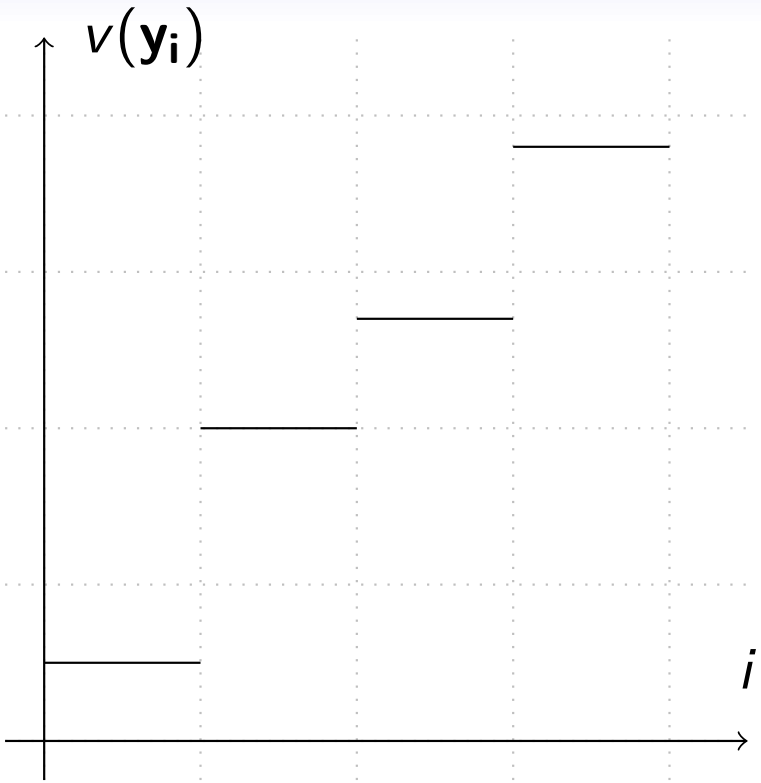
- **A vector of variables** $(\mathbf{y_1}, \ldots, \mathbf{y_n})$.
- **A constraint Sort**$(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{y}})$ ([Mehlhorn and Thiel, 2000] (filtering in time $O(n \log(n))$).

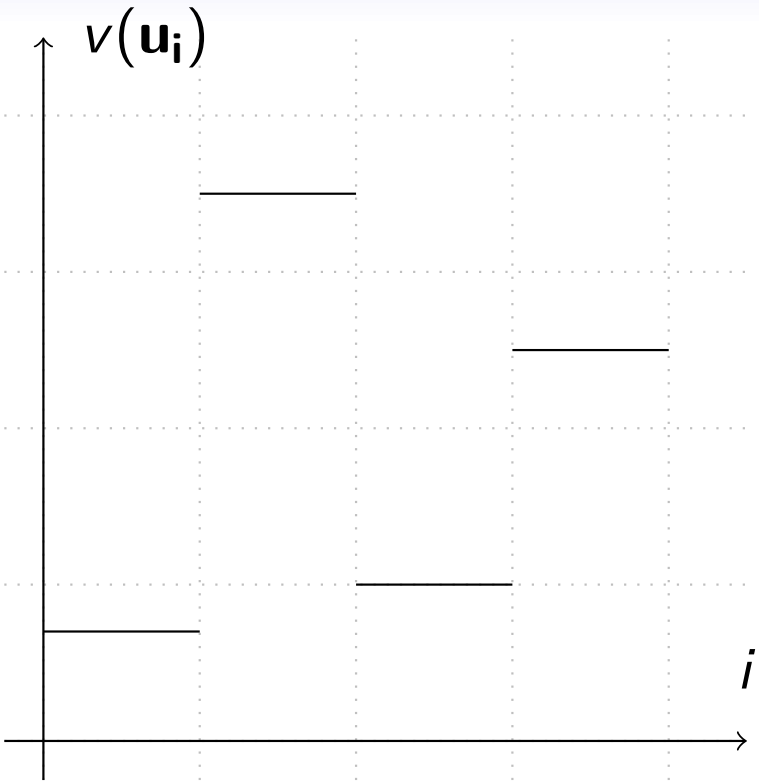
1. Maximize $\mathbf{y_1}$: $\widehat{y_1}$.
2. Maximize $\mathbf{y_2}$ under the constraint $\mathbf{y_1} = \widehat{y_1}$: $\widehat{y_2}$.

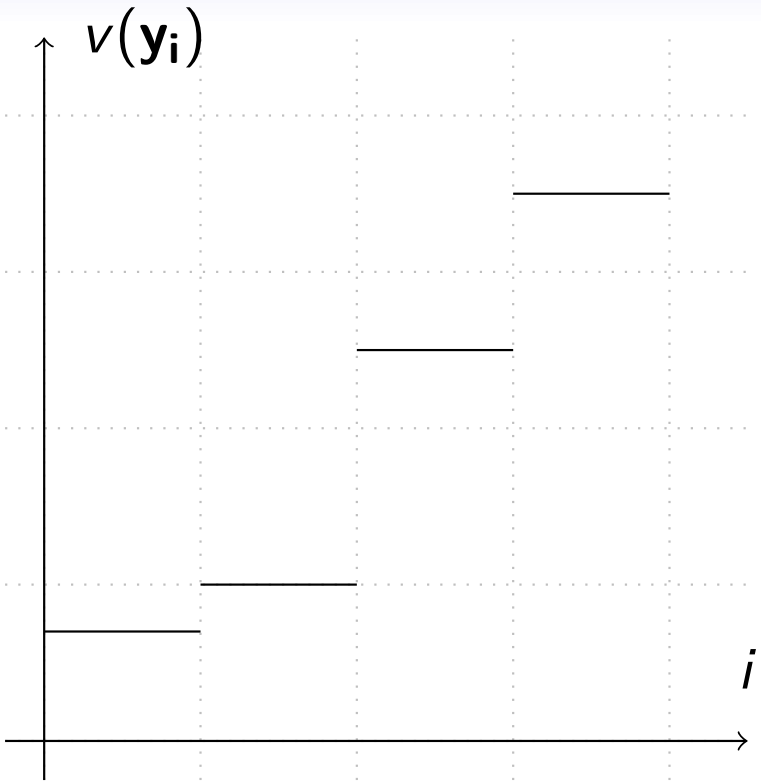
$\vdots$

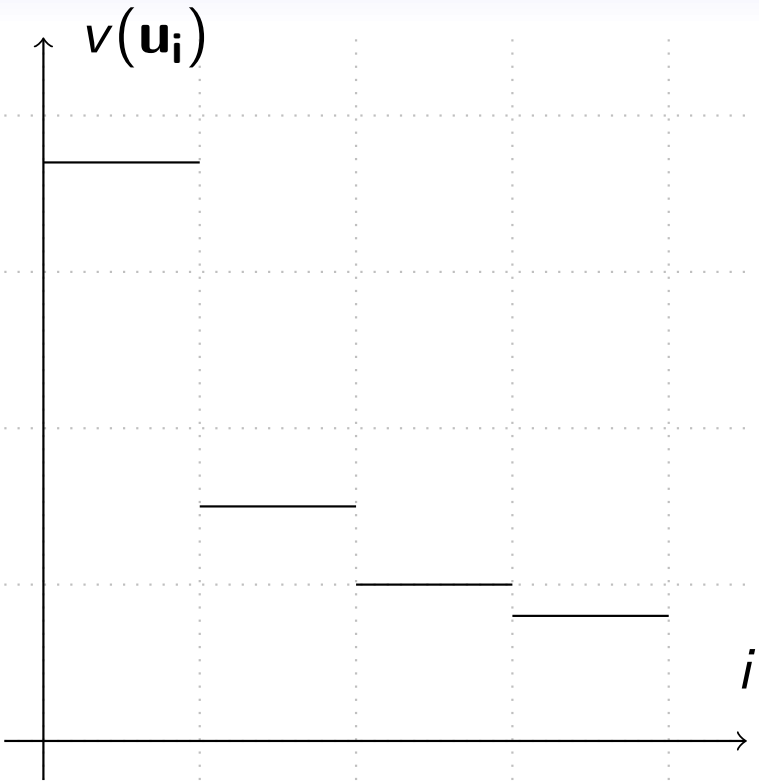
n. Maximize $\mathbf{y_n}$ under the constraints $\mathbf{y_1} = \widehat{y_1}, \ldots, \mathbf{y_{n-1}} = \widehat{y_{n-1}}$.

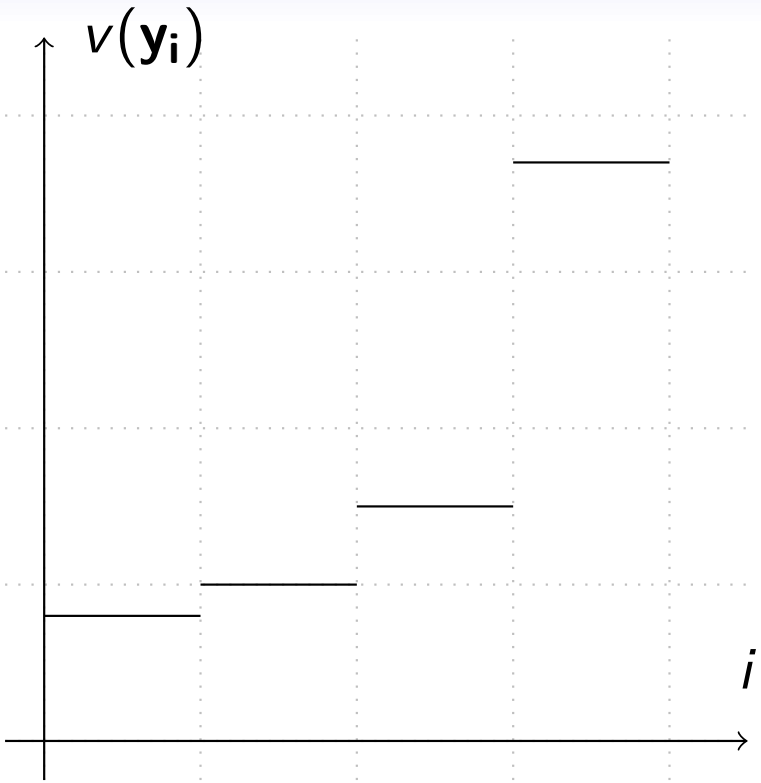

$v(\mathbf{u_i})$
i

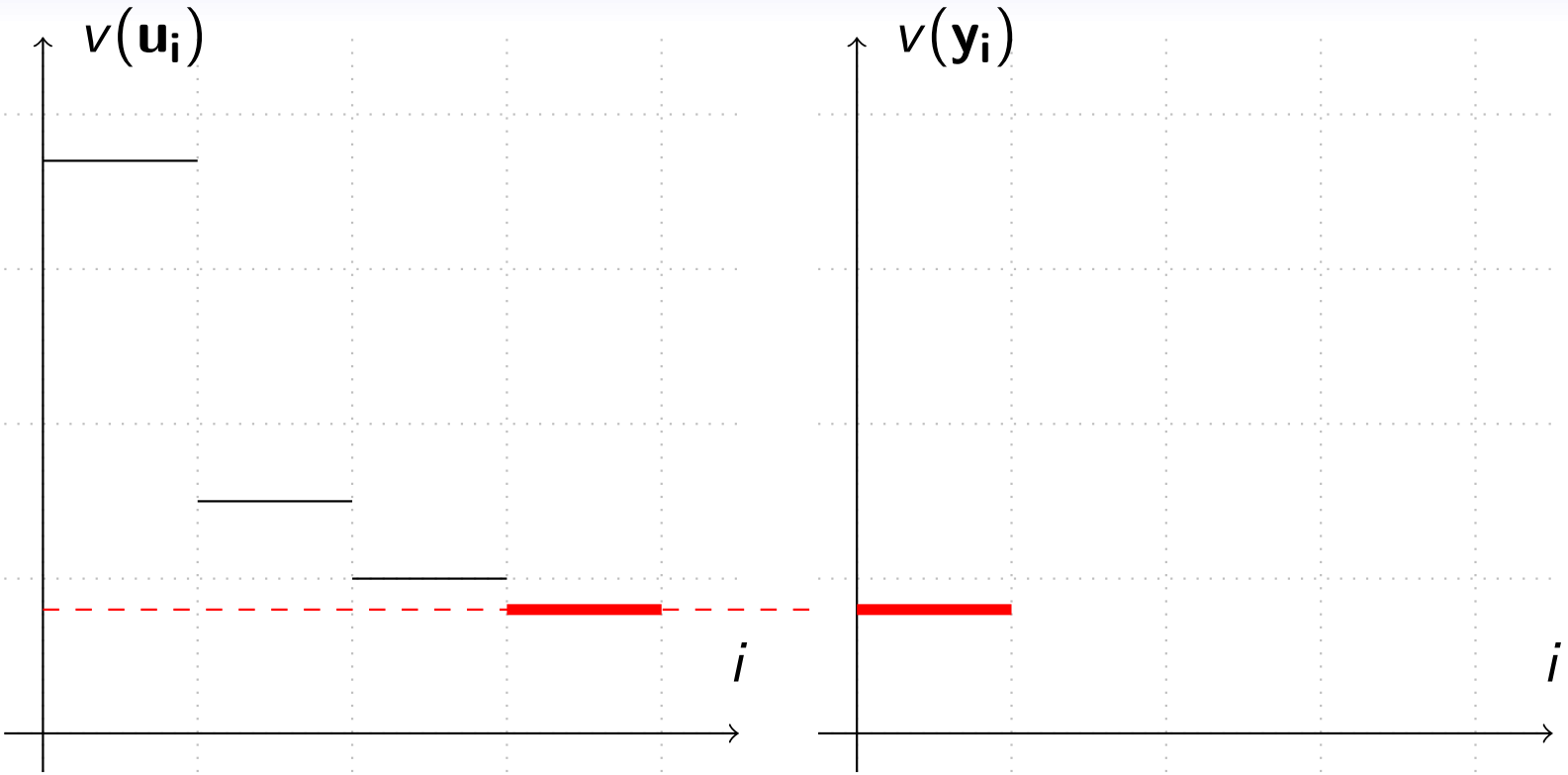

$v(\mathbf{y_i})$
i

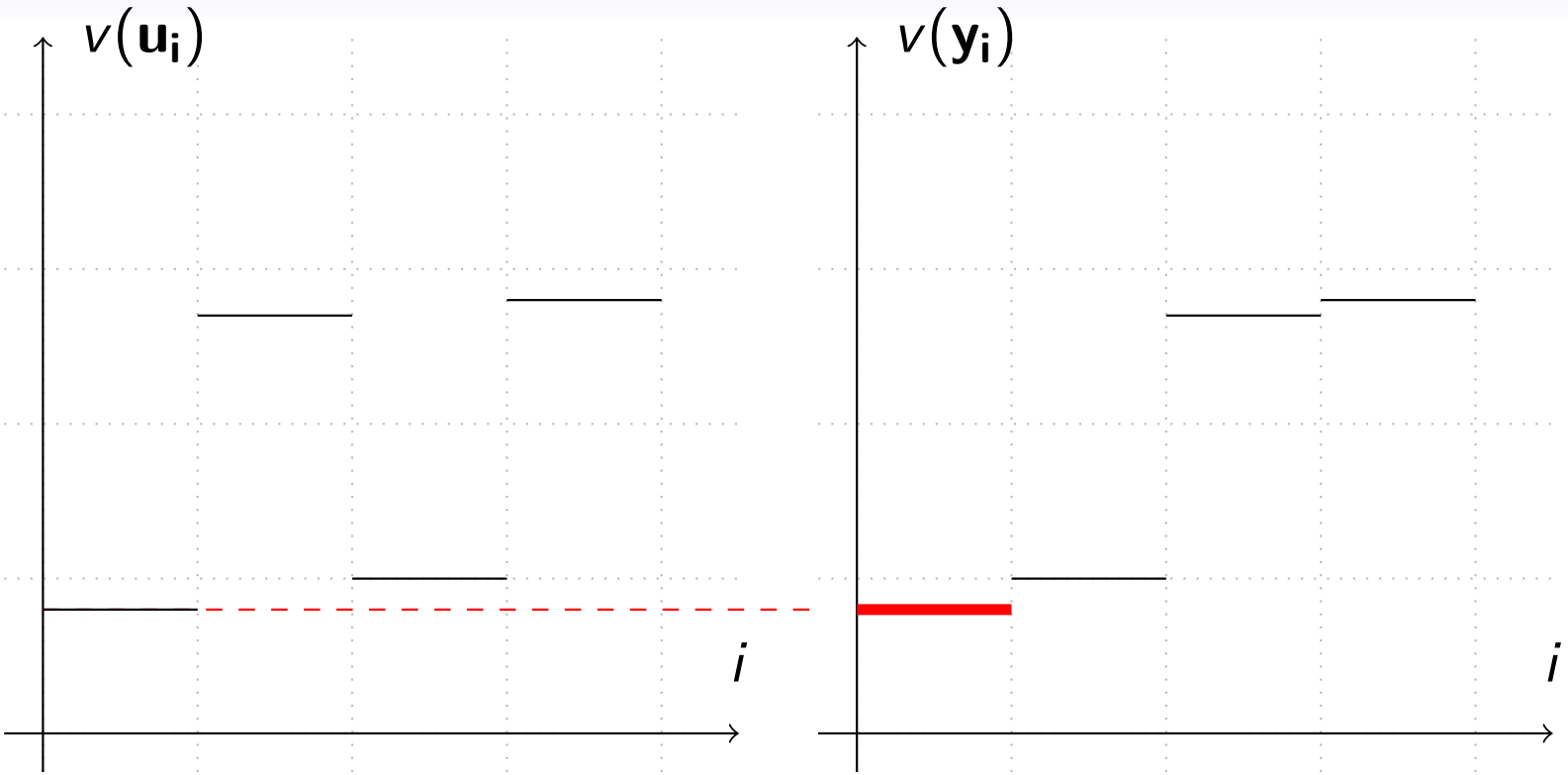

$v(\mathbf{u_i})$
i

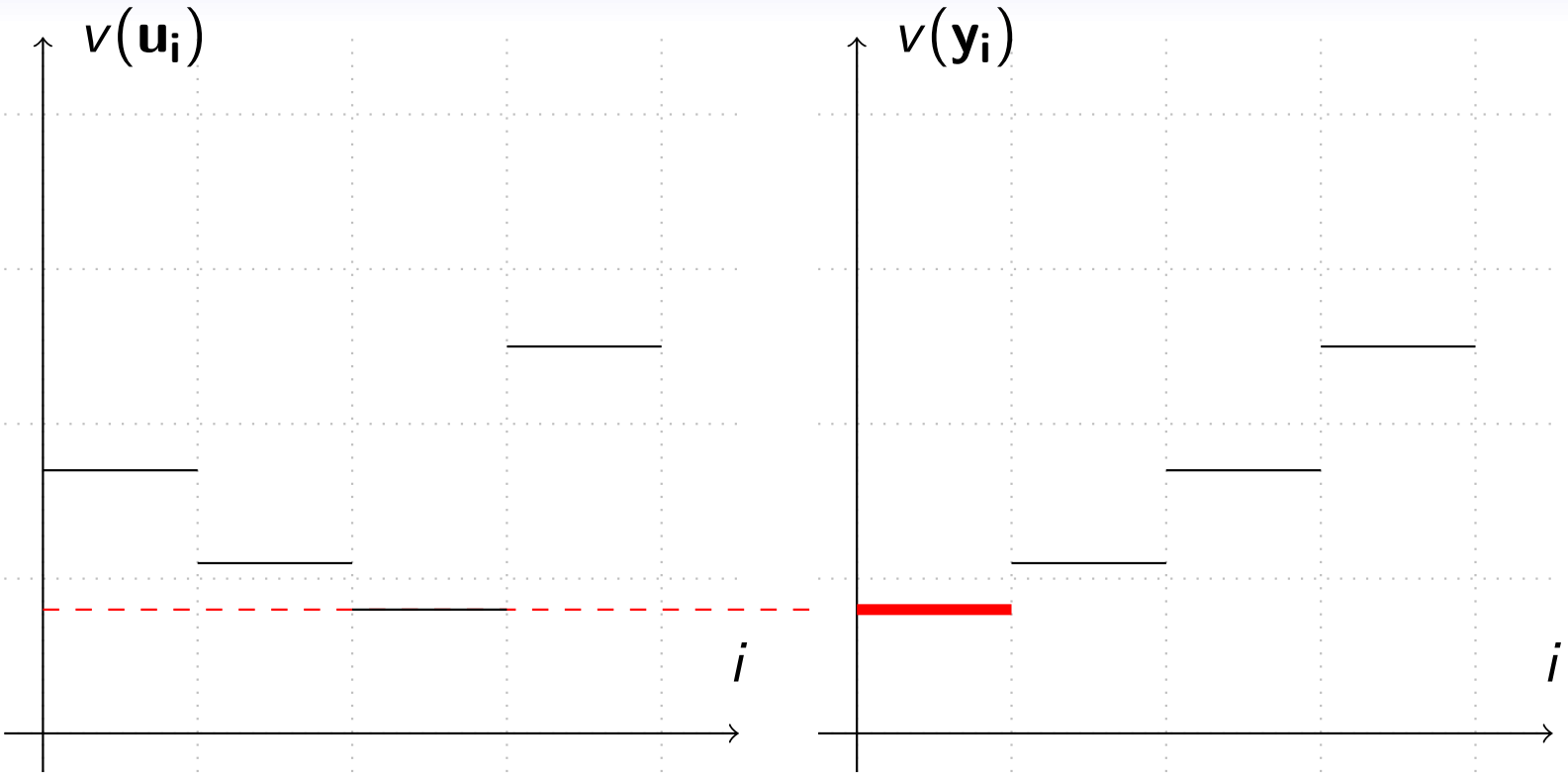

$v(\mathbf{y_i})$
i

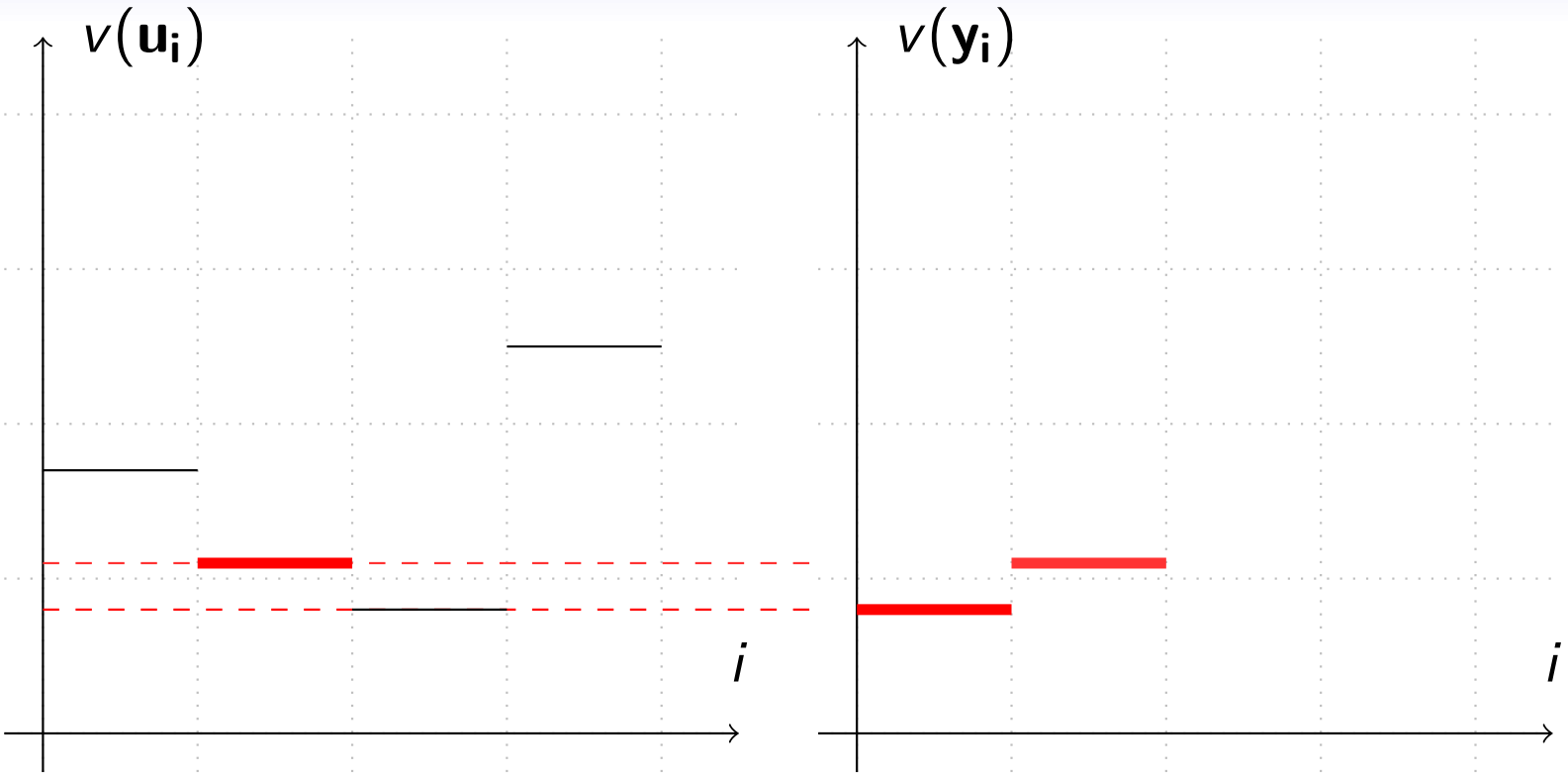

$v(\mathbf{u_i})$
i

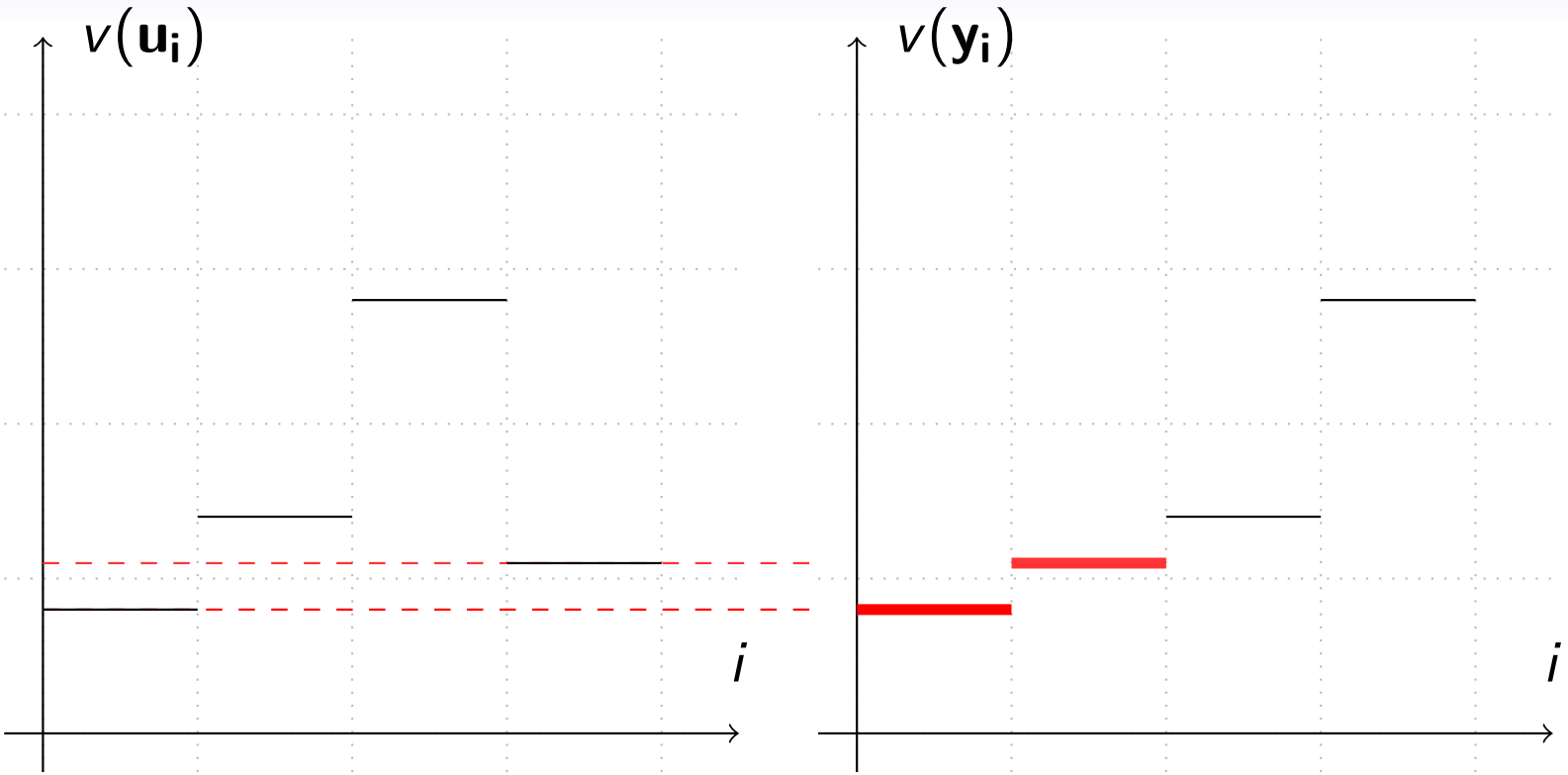

$v(\mathbf{y_i})$
i

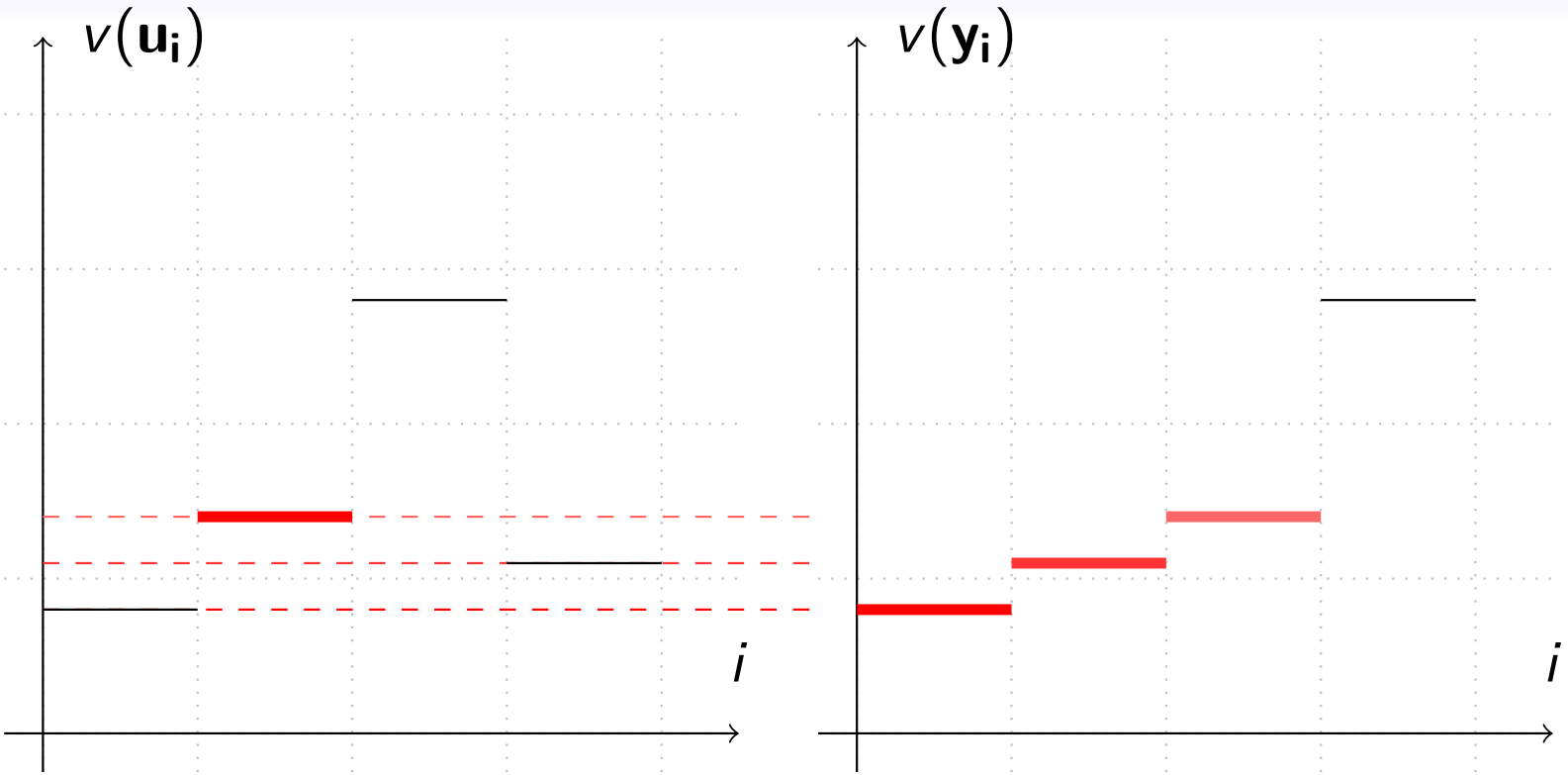

$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i

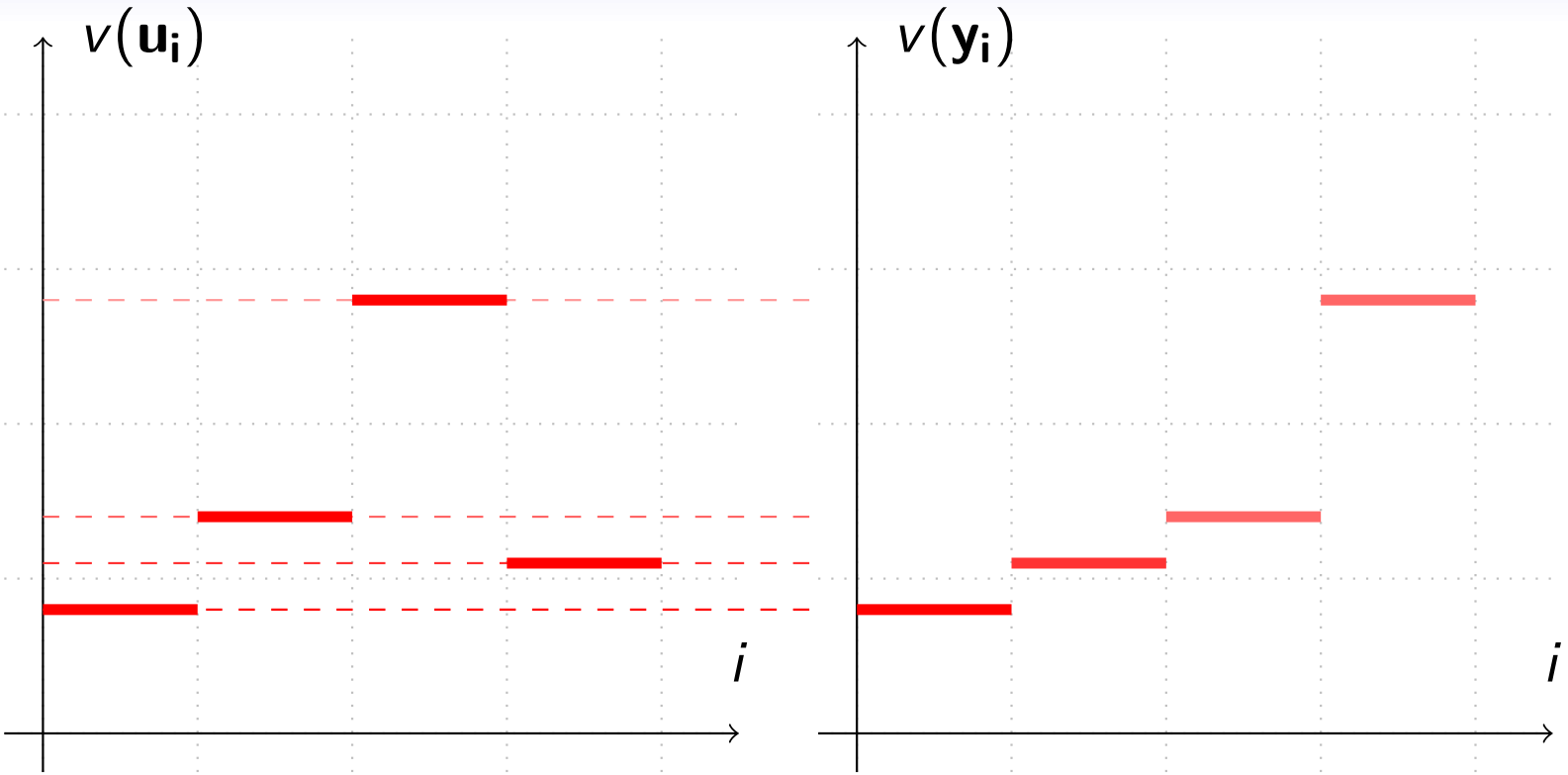

$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i

Algorithm 2

Using cardinality combinators. . .

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.eq to y_1;
- y_2 is the maximal value such that at least $n-1$ values among the u_i are g.eq to y_2;

$\vdots$

- y_n is the maximal value such that at least 1 value among the u_i is g.eq to y_n.

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.eq to y_1;
- y_2 is the maximal value such that at least $n-1$ values among the u_i are g.eq to y_2;

$\vdots$

- y_n is the maximal value such that at least 1 value among the u_i is g.eq to y_n.

An alternative definition of sorting...

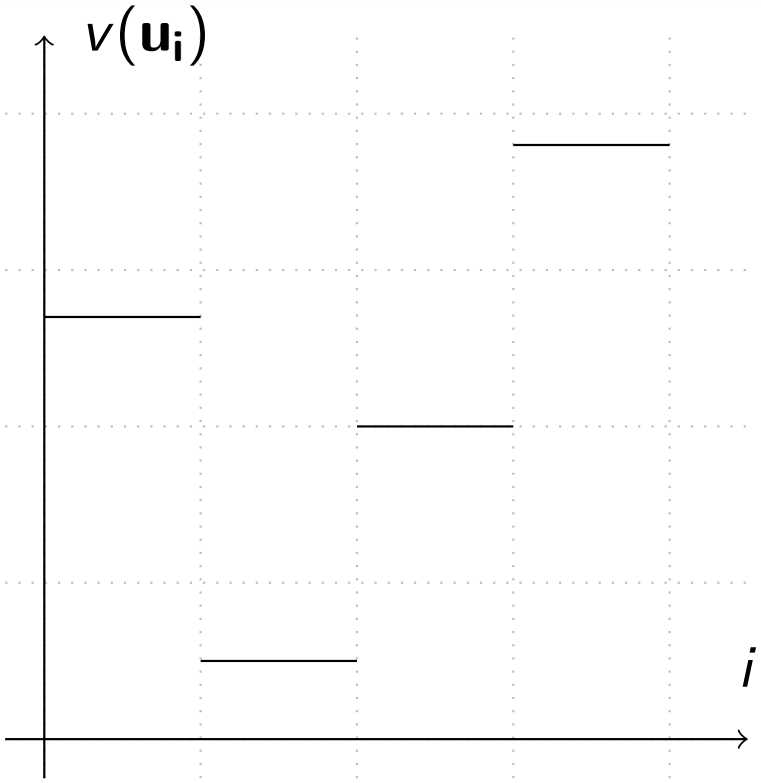
Proposition

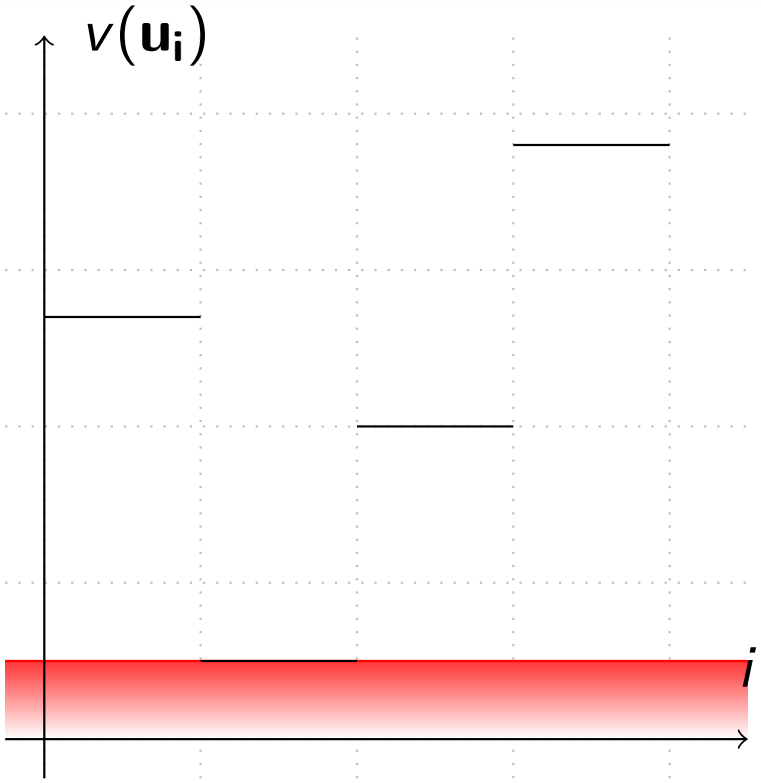
$\langle y_1, \ldots, y_n \rangle$ is the permutation of $\langle u_1, \ldots, u_n \rangle$ sorted in non-decreasing order if and only if:

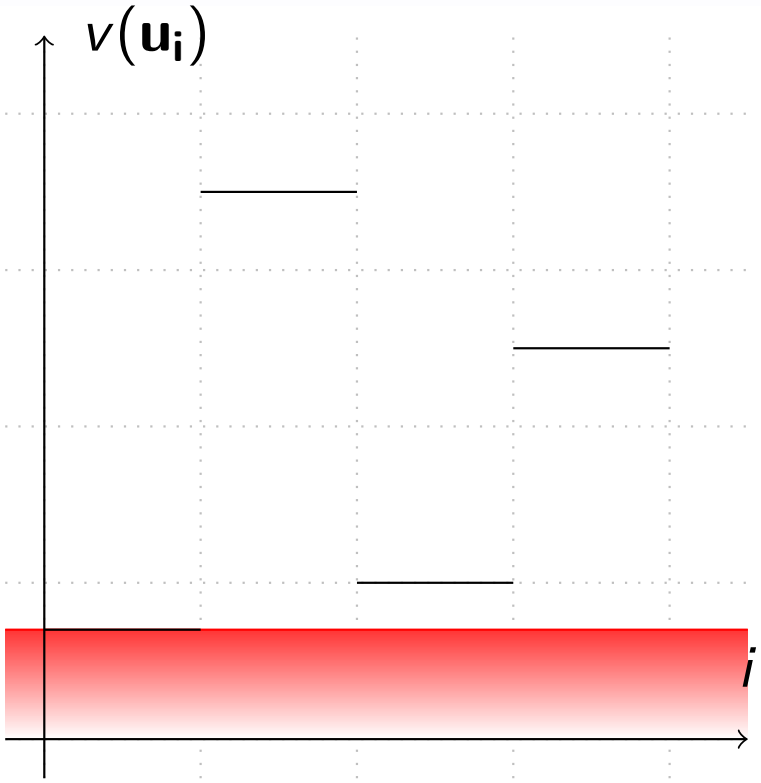
- y_1 is the maximal value such that all the u_i are g.eq to y_1;
- y_2 is the maximal value such that at least $n-1$ values among the u_i are g.eq to y_2;

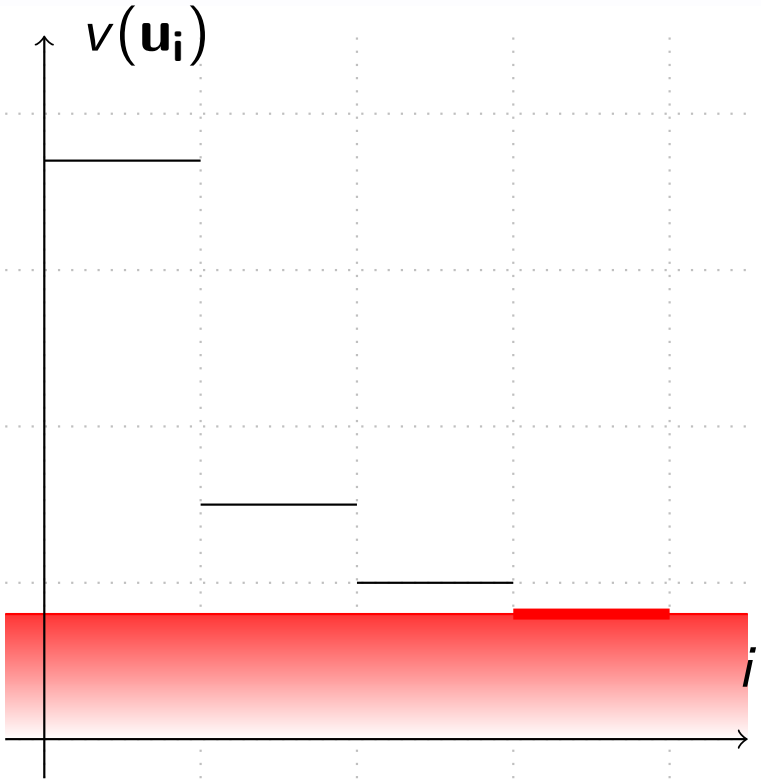
$$\vdots$$

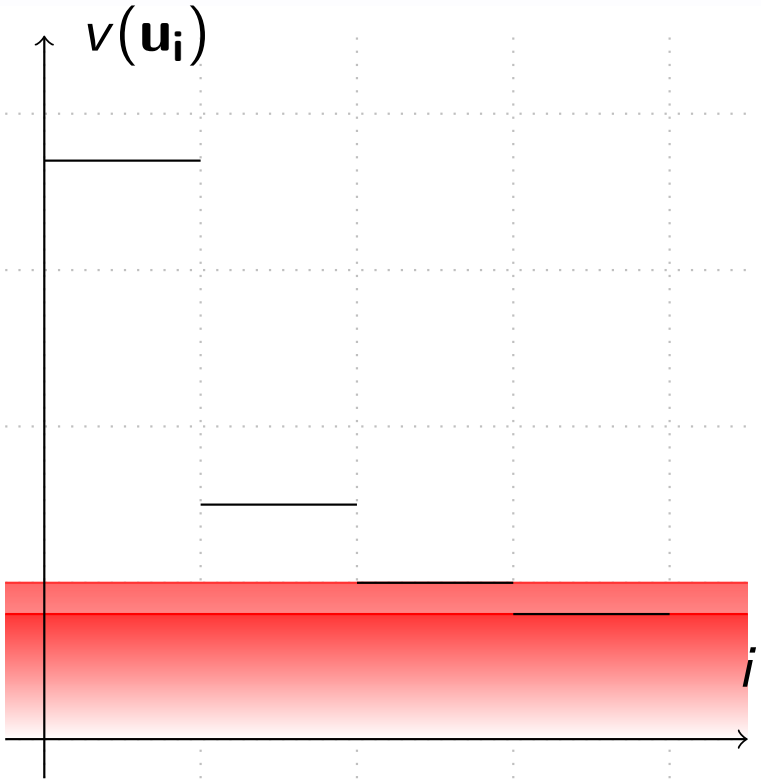
- y_n is the maximal value such that at least 1 value among the u_i is g.eq to y_n.

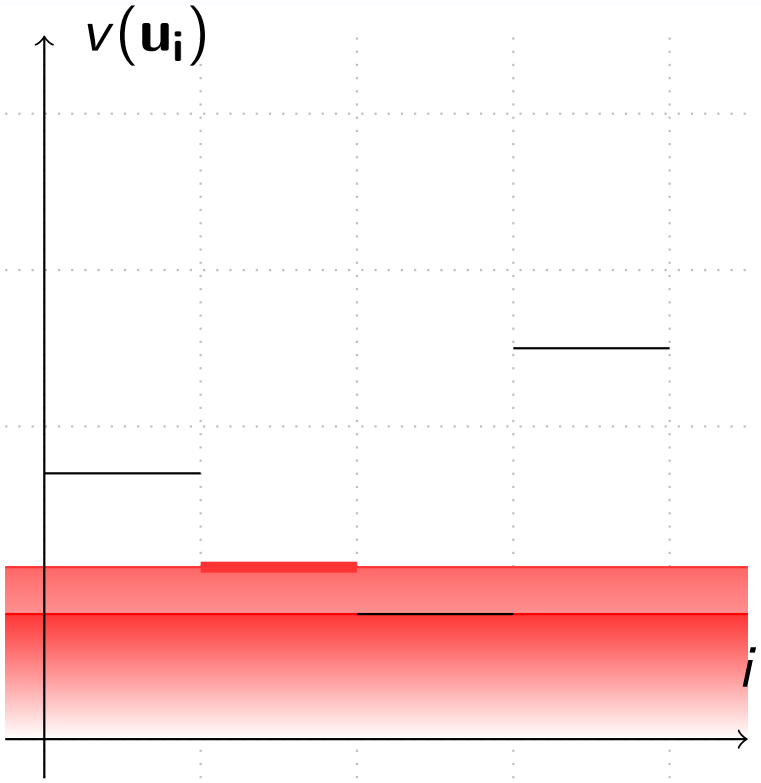

$v(\mathbf{u_i})$
i

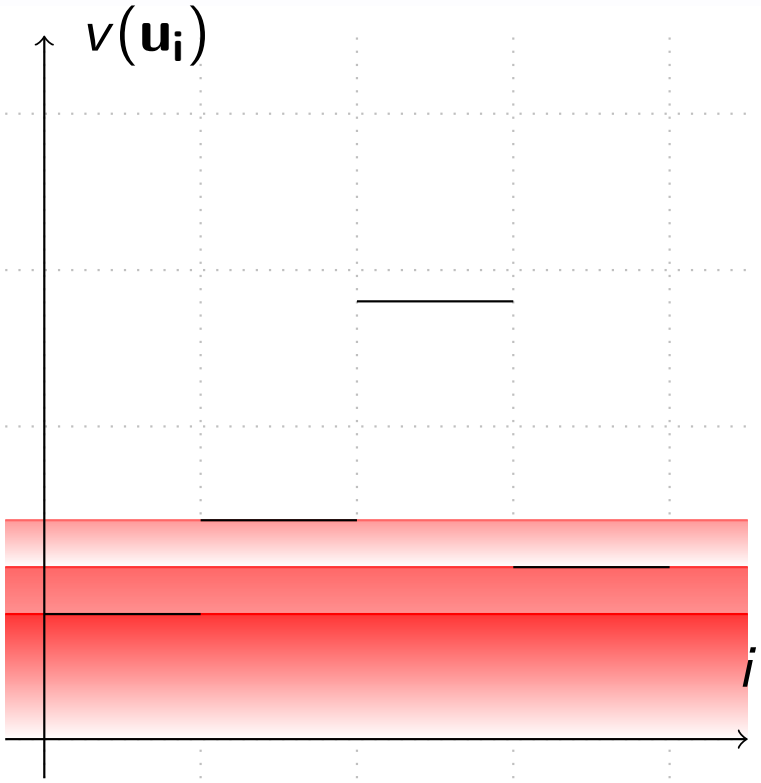

$v(\mathbf{u_i})$
i

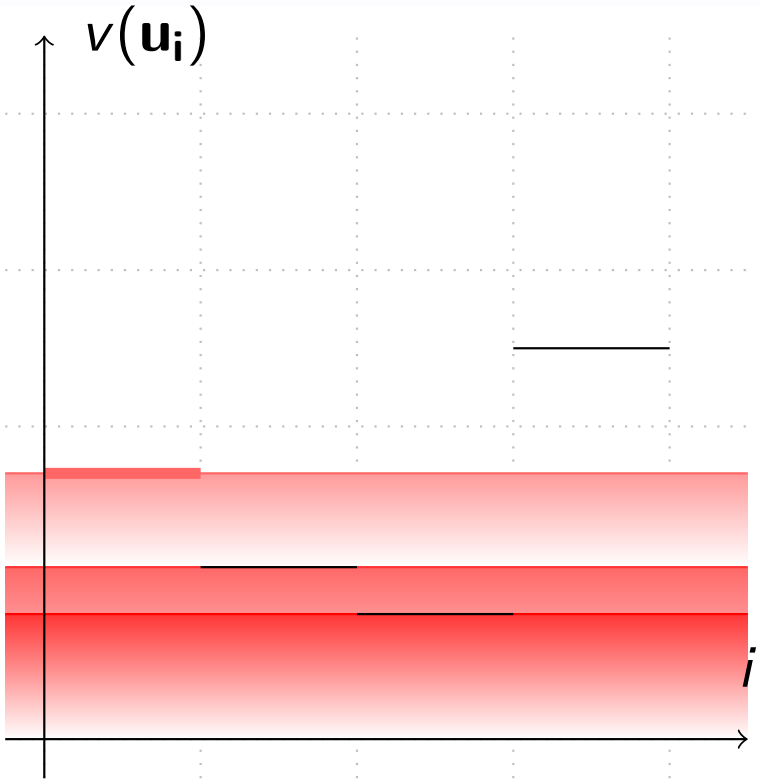

$v(\mathbf{u_i})$
i

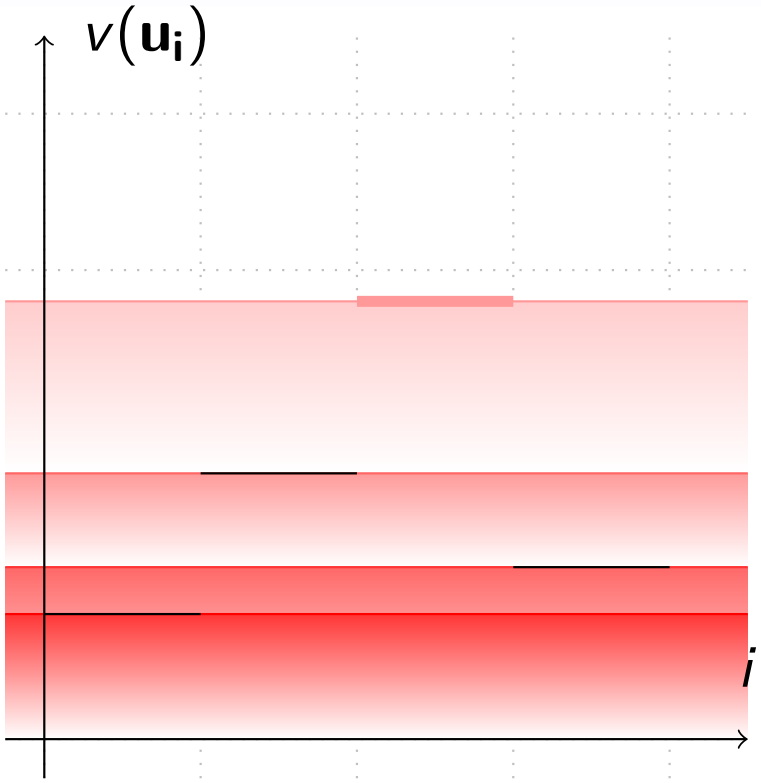

$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
i

The meta-constraint AtLeast

" y_i is the maximal value such that at least $n - i + 1$ values among the u_i are g.e.q than y_i "

$\rightsquigarrow$ a particular cardinality meta-constraint [Van Hentenryck et al., 1992]:

$$\textbf{AtLeast}(\{\mathbf{y_i} \geq u_1, \ldots, \mathbf{y_i} \geq u_n\}, n - i + 1)$$

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

The meta-constraint AtLeast

" y_i is the maximal value such that at least $n - i + 1$ values among the u_i are g.e.q than y_i "

$\rightsquigarrow$ a particular cardinality meta-constraint [Van Hentenryck et al., 1992]:

$$\textbf{AtLeast}(\{\mathbf{y_i} \geq u_1, \ldots, \mathbf{y_i} \geq u_n\}, n - i + 1)$$

- A specific filtering algorithm running in $O(n)$.
- A possible implementation using linear constraints.

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

Algorithm 3

A branch-and-bound-like algorithm

A branch-and-bound-like algorithm

The classical branch-and-bound (integral criterion):

- A branching algorithm (exploration of the search tree).
- A lower bound on the criterion to maximize.
- An upper bound and a pruning mechanism ($ub \leq lb$).

Our algorithm (vectorial criterion with leximin preorder):

- Branching algorithm given by the constraint solver (call to **solve**).
- Lower bound: the objective vector of the last solution found.
- Pruning mechanism given by a filtering procedure associated to the leximin preorder (we reject every solution whose objective vector is leximin-lower than the lower bound).

A branch-and-bound-like algorithm

The classical branch-and-bound (integral criterion):

- A branching algorithm (exploration of the search tree).
- A lower bound on the criterion to maximize.
- An upper bound and a pruning mechanism ($ub \leq lb$).

Our algorithm (vectorial criterion with leximin preorder):

- Branching algorithm given by the constraint solver (call to **solve**).
- Lower bound: the objective vector of the last solution found.
- Pruning mechanism given by a filtering procedure associated to the leximin preorder (we reject every solution whose objective vector is leximin-lower than the lower bound).

A branch-and-bound-like algorithm

The classical branch-and-bound (integral criterion):

- A branching algorithm (exploration of the search tree).
- A lower bound on the criterion to maximize.
- An upper bound and a pruning mechanism ($ub \leq lb$).

Our algorithm (vectorial criterion with leximin preorder):

- Branching algorithm given by the constraint solver (call to **solve**).
- Lower bound: the objective vector of the last solution found.
- Pruning mechanism given by a filtering procedure associated to the leximin preorder (we reject every solution whose objective vector is leximin-lower than the lower bound).

A branch-and-bound-like algorithm

The classical branch-and-bound (integral criterion):

- A branching algorithm (exploration of the search tree).
- A lower bound on the criterion to maximize.
- An upper bound and a pruning mechanism ($ub \leq lb$).

Our algorithm (vectorial criterion with leximin preorder):

- Branching algorithm given by the constraint solver (call to **solve**).
- Lower bound: the objective vector of the last solution found.
- Pruning mechanism given by a filtering procedure associated to the leximin preorder (we reject every solution whose objective vector is leximin-lower than the lower bound).

A branch-and-bound-like algorithm

The classical branch-and-bound (integral criterion):

- A branching algorithm (exploration of the search tree).
- A lower bound on the criterion to maximize.
- An upper bound and a pruning mechanism ($ub \leq lb$).

Our algorithm (vectorial criterion with leximin preorder):

- Branching algorithm given by the constraint solver (call to **solve**).
- Lower bound: the objective vector of the last solution found.
- Pruning mechanism given by a filtering procedure associated to the leximin preorder (we reject every solution whose objective vector is leximin-lower than the lower bound).

A branch-and-bound-like algorithm

The classical branch-and-bound (integral criterion):

- A branching algorithm (exploration of the search tree).
- A lower bound on the criterion to maximize.
- An upper bound and a pruning mechanism ($ub \leq lb$).

Our algorithm (vectorial criterion with leximin preorder):

- Branching algorithm given by the constraint solver (call to **solve**).
- Lower bound: the objective vector of the last solution found.
- Pruning mechanism given by a filtering procedure associated to the leximin preorder (we reject every solution whose objective vector is leximin-lower than the lower bound).

A constraint Leximin

We use a constraint **Leximin**: **Leximin**$(\overrightarrow{\lambda}, \overrightarrow{\mathbf{x}})$ (the vector $\overrightarrow{\mathbf{x}}$ must be leximin-greater than the integer vector $\overrightarrow{\lambda}$)

This constraint is based on the constraint **Multiset Ordering**, introduced in [Frisch et al., 2003] (filtering 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.

Algorithm 4

Using cardinality-minimal critical subsets

Leximin and critical subsets

- The algorithm comes from the litterature on flexible CSP [Dubois and Fortemps, 1999].
- It is based on the search for critical subsets of components of the objective vector (*i.e.* conditioning the minimax value).
- Major drawback: can potentially perform an exponential number of calls to **solve**.

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

Leximin and critical subsets

- The algorithm comes from the litterature on flexible CSP [Dubois and Fortemps, 1999].
- It is based on the search for critical subsets of components of the objective vector (*i.e.* conditioning the minimax value).
- Major drawback: can potentially perform an exponential number of calls to **solve**.

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

Leximin and critical subsets

- The algorithm comes from the litterature on flexible CSP [Dubois and Fortemps, 1999].
- It is based on the search for critical subsets of components of the objective vector (*i.e.* conditioning the minimax value).
- Major drawback: can potentially perform an exponential number of calls to **solve**.

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

Outline

Implementation

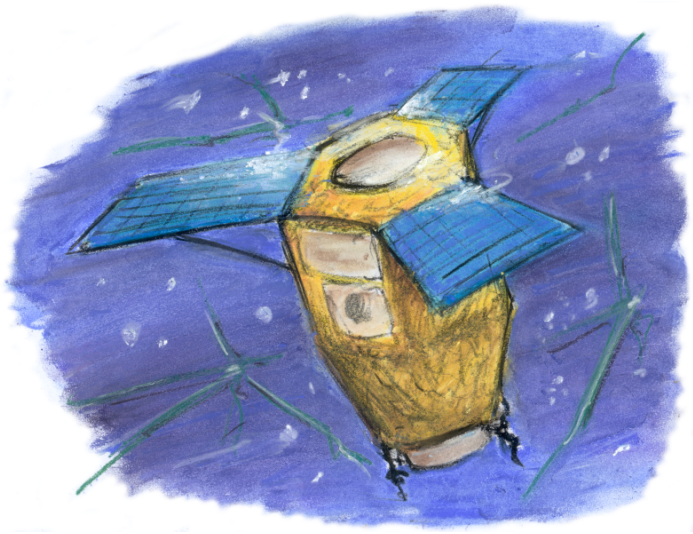
Implementation of the algorithms:
The four algorithms have been implemented using Choco [F. Laburthe and the OCRE project team, 2000] in Java.

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`.

Our 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 problem with (linear) consumption and volume resources.



Instance generator:

We implemented a random instance generator for our simplified problem.

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

General tendency of the results

- The algorithm based on the **AtLeast** constraint is often the most efficient, followed by the algorithm based on the **Sort** constraint (which has similar running times, except when the number of agents increases).
- The algorithm inspired by the **Multiset ordering** constraint is efficient on the instances of our resource allocation problem with a high number of agents.
- The algorithm from [Dubois and Fortemps, 1999] explodes when the number of equal components in the leximin-optimal vector increases.

Summary

- **Problem studied:** 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 four algorithms (the last one coming from flexible CSP) based on the CP framework.
- **Implementation:** Implementation and testing of the algorithms in Java with Choco.

Summary

- **Problem studied:** 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 four algorithms (the last one coming from flexible CSP) based on the CP framework.
- **Implementation:** Implementation and testing of the algorithms in Java with Choco.

Summary

- **Problem studied:** 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 four algorithms (the last one coming from flexible CSP) based on the CP framework.
- **Implementation:** Implementation and testing of the algorithms in Java with Choco.

Summary

- **Problem studied:** 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 four algorithms (the last one coming from flexible CSP) based on the CP framework.
- **Implementation:** Implementation and testing of the algorithms in Java with Choco.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, ...).
  - Applying the algorithms to other practical fields and applications.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, ...).
  - Applying the algorithms to other practical fields and applications.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, ...).
  - Applying the algorithms to other practical fields and applications.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, ...).
  - Applying the algorithms to other practical fields and applications.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, . . .).
  - Applying the algorithms to other practical fields and applications.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, . . .).
  - Applying the algorithms to other practical fields and applications.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by encoding the leximin preorder by a collective utility function) – possibly using the WCSP framework.
  - Combining some of the four algorithms together to get better results.
  - Designing and implementing approximation algorithms.
- **Possible extensions:**
  - More general (and softer) modeling of fairness (*e.g.* OWA, inequality indices, . . .).
  - Applying the algorithms to other practical fields and applications.

This is the end.

For more information:
michel.lemaitre@cert.fr
sylvain.bouveret@cert.fr
http://www.cert.fr/dcsd/THESES/sbouveret

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