Finding Leximin-Optimal Solutions using Constraint Programming

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

1st International Workshop on Computational Social Choice
Amsterdam, 8 december 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.

. . . 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

1. **Modeling the problem**
   - Constraint Satisfaction Problems
   - The leximin criterion
2. **Solving the problem**
   - Sort and Conquer
   - Using cardinality combinators
   - A branch-and-bound-like algorithm
   - Using cardinality-minimal critical subsets
3. **Implementing the problem**
   - Fair combinatorial auctions
   - Results

Outline

1. **Modeling the problem**
   - Constraint Satisfaction Problems
   - The leximin criterion
2. **Solving the problem**
   - Sort and Conquer
   - Using cardinality combinators
   - A branch-and-bound-like algorithm
   - Using cardinality-minimal critical subsets
3. **Implementing the problem**
   - Fair combinatorial auctions
   - Results

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 $(\mathscr{X}, \mathscr{D}, \mathscr{C})$ *?*

$\leadsto$ **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* $\widehat{v}(\mathbf{o}) = \alpha$ *?*

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

CSP and collective decision making problems

Combinatorial collective decision making problems can be naturally represented in the CSP framework, by introducing **utility variables**.

CSP and collective decision making problems

A resource allocation problem

- An allocation problem with 3 agents and 3 objects.
- Constraint: One object cannot be given to more than one 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

CSP and collective decision making problems

A resource allocation problem

- Variables: $\mathscr{X} = \{o_{1,1}, o_{1,2}, o_{1,3}, \dots, o_{3,3}, u_1, u_2, u_3\}$.
- Domains: $\mathscr{D} = \{\{0,1\}, \dots, \{0,1\}, [\![0,15]\!], [\![0,20]\!], [\![0,11]\!]\}$.
- Constraints: $\mathscr{C} = \{u_1 = 3o_{1,1} + 5o_{1,2} + 7o_{1,3}, u_2 = \dots, u_3 = \dots, \forall i, \sum_{i=1}^{3} o_{i,j} \geq 1\}$

Which criterion shall we optimize ?

- **Question:** which criterion shall we optimize to ensure fairness and Pareto-efficiency requirements ?
- **Our answer:** The leximin criterion seems to be well-suited.

Which criterion shall we optimize ?

- **Question:** which criterion shall we optimize to ensure fairness and Pareto-efficiency requirements ?
- **Our answer:** The leximin criterion seems to be well-suited.

Which criterion shall we optimize ?

- **Question:** which criterion shall we optimize to ensure fairness and Pareto-efficiency requirements ?
- **Our answer:** The leximin criterion seems to be well-suited.

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 $u_{k+1}^{\uparrow} > v_{k+1}^{\uparrow}$.
This is a lexicographical comparison over sorted vectors.

Which criterion shall we optimize ?

- **Question:** which criterion shall we optimize to ensure fairness and Pareto-efficiency requirements ?
- **Our answer:** The leximin criterion seems to be well-suited.

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

Which criterion shall we optimize ?

- **Question:** which criterion shall we optimize to ensure fairness and Pareto-efficiency requirements ?
- **Our answer:** The leximin criterion seems to be well-suited.

Features

This SWO both refines the egalitarian SWO and the Pareto relation $\rightsquigarrow$ it inherits of the fairness features of egalitarism, while overcoming the drowning effect.

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

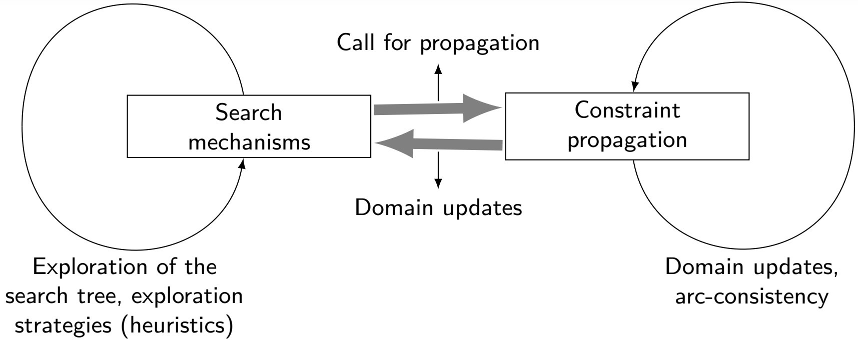
1. Modeling the problem
   - Constraint Satisfaction Problems
   - The leximin criterion
2. **Solving the problem**
   - Sort and Conquer
   - Using cardinality combinators
   - A branch-and-bound-like algorithm
   - Using cardinality-minimal critical subsets
3. Implementing the problem
   - Fair combinatorial auctions
   - Results

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 under 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 under using the leximin preorder ⇔ 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 under 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 $\mathbf{y_1} = \widehat{y_1}$, ..., $\mathbf{y_{n-1}} = \widehat{y_{n-1}}$.

Sorted version of the objective vector

Initial idea

Maximize the objective vector under 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 under using the leximin preorder ⇔ maximize the successive components of the **ordered** objective vector.

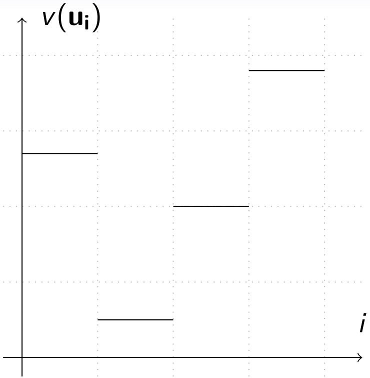
$\leadsto$ We have to introduce the sorted version of the objective vector:

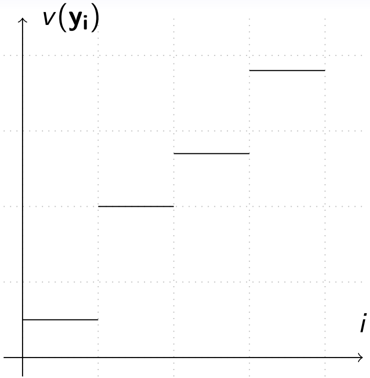
- **A vector of variables** $(\mathbf{y_1}, \dots, \mathbf{y_n})$.
- **A constraint Sort**$(\overrightarrow{\mathbf{u}}, \overrightarrow{\mathbf{y}})$ ([Mehlhorn and Thiel, 2000] (filtering in time $O(n \log(n))$).

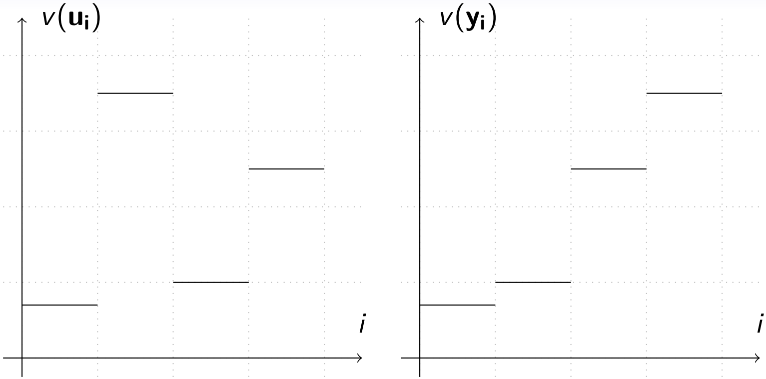
1. Maximize $\mathbf{y_1}$: $\widehat{y_1}$.
2. Maximize $\mathbf{y_2}$ under the constraint $\mathbf{y_1} = \widehat{y_1}$: $\widehat{y_2}$.

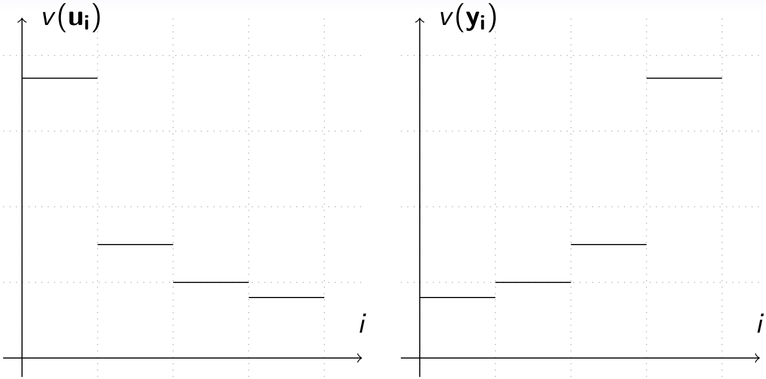
⋮

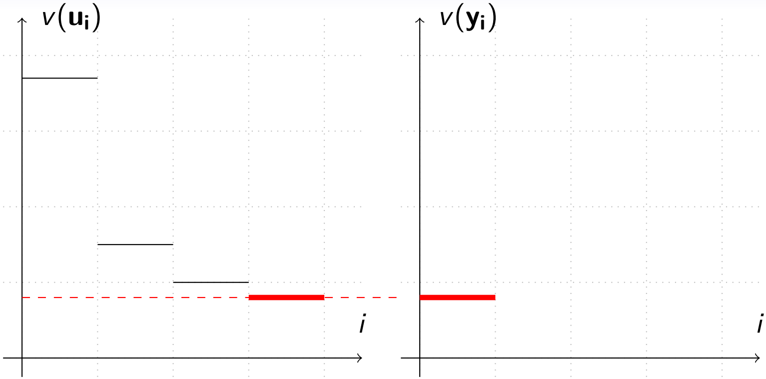
n. Maximize $\mathbf{y_n}$ under the constraints $\mathbf{y_1} = \widehat{y_1}, \dots, \mathbf{y_{n-1}} = \widehat{y_{n-1}}$.

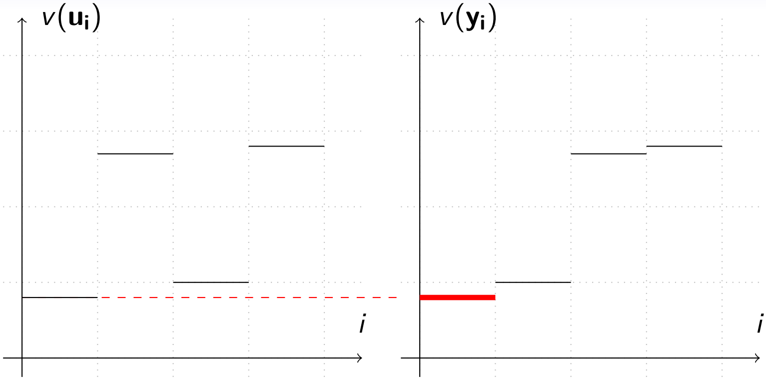

$v(\mathbf{u_i})$
i

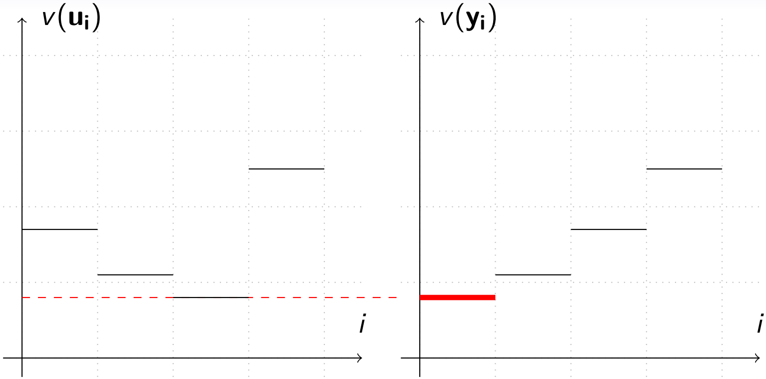

$v(\mathbf{y_i})$
i

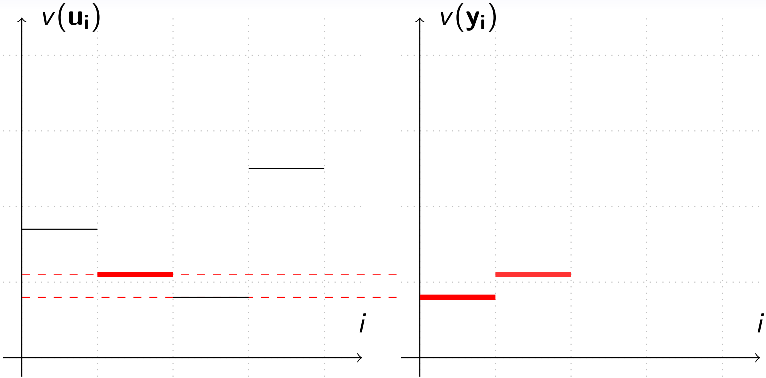

v(ui)
i
v(yi)
i


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

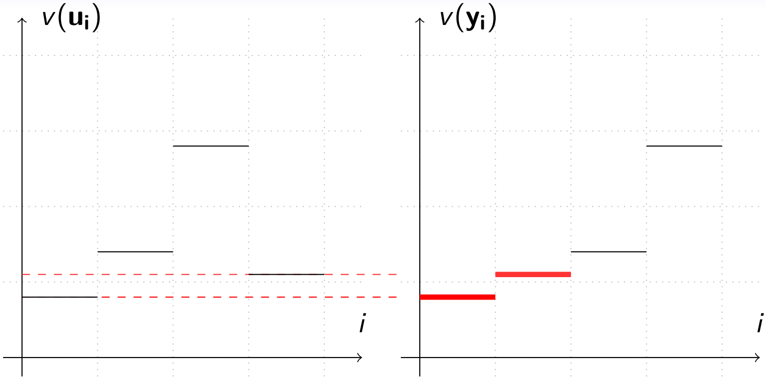

$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i

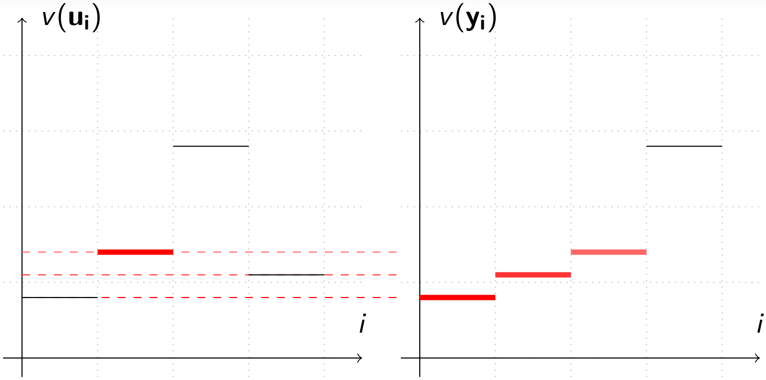

$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


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

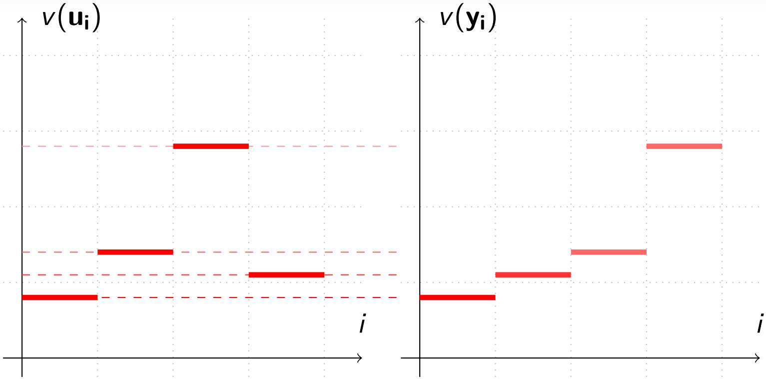


$v(\mathbf{u_i})$
i
$v(\mathbf{y_i})$
i


$v(\mathbf{u_i})$
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...

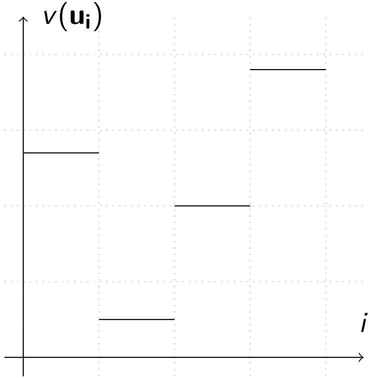
Proposition

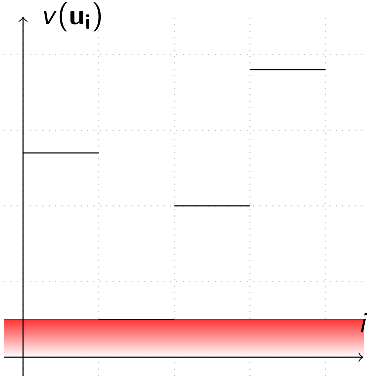
$\langle y_1, \ldots, y_n \rangle$ is the permutation of $\langle u_1, \ldots, u_n \rangle$ sorted in non-decreasing order if and only if:

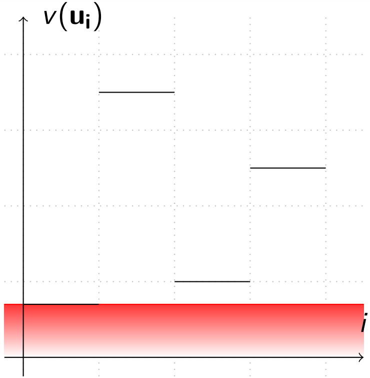
- y_1 is the maximal value such that all the u_i are g.eq to y_1;
- y_2 is the maximal value such that at least $n-1$ values among the u_i are g.eq to y_2;

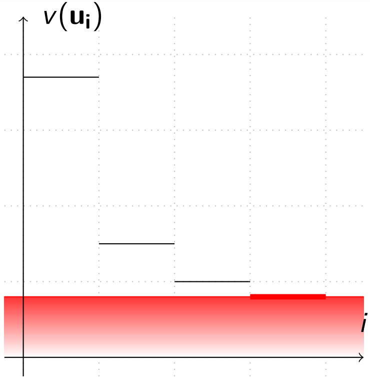
$$\vdots$$

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

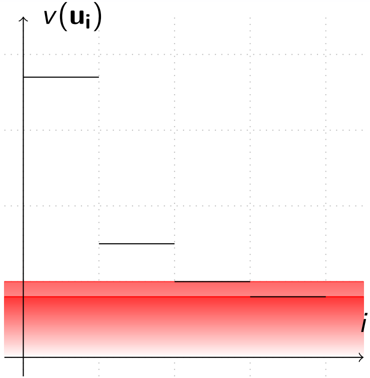

$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
i

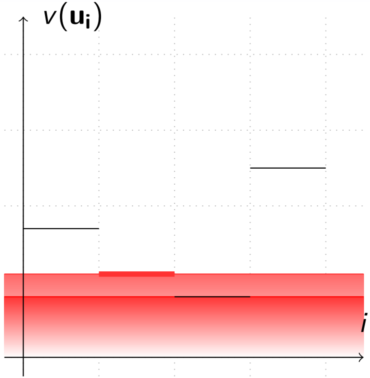

$v(\mathbf{u_i})$
i

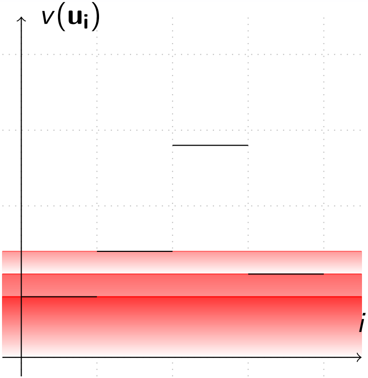


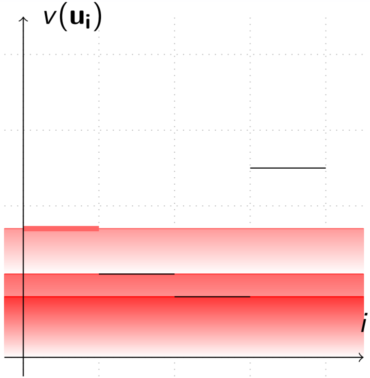
$v(\mathbf{u_i})$
i

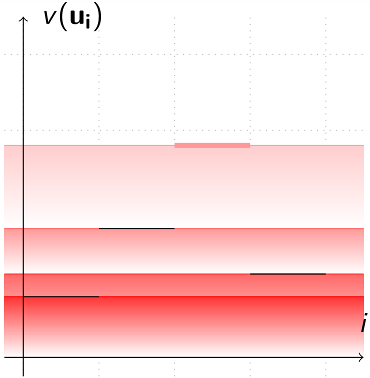


$v(\mathbf{u_i})$
i


$v(\mathbf{u_i})$
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 $\rightsquigarrow$ use of a particular cardinality meta-constraint [Van Hentenryck et al., 1992]:

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

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 $\rightsquigarrow$ use of 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 of 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 of 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 of 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 of 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 of 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 of 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 resolutions.

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 resolutions.

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 resolutions.

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

Outline

Combinatorial auctions

Combinatorial auctions [Cramton et al., 2006]

- a set of agents $\mathcal{A}$;
- a set of objects $\mathcal{O}$;
- each agent bids on **bundles of items** (a bid being a set of objects associated to a price).

What is the set of non-intersecting bids maximizing the sum of the prices ?

Cramton, P., Shoham, Y., and Steinberg, R., editors (2006).
Combinatorial Auctions.
MIT Press.

Fair combinatorial auctions

Fair combinatorial auctions

- a set of agents $\mathcal{A}$;
- a set of objects $\mathcal{O}$;
- each agent bids on **bundles of items** (a bid being a set of objects associated to a price).
- we make the assumption that the utility of an agent is equal to the sum of the prices of her selected bids.

What is the set of non-intersecting bids maximizing the leximin over the utility profiles ?

A random instance generator with realistic bids for combinatorial auction problems exists: CATS (`http://cats.stanford.edu`).

Implementation

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

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.

General tendency of the results

- The algorithm based on the meta-constraint **AtLeast** seems to be the most efficient one. . .
- . . . followed by the algorithm based on the constraint **Sort**.
- The algorithm from [Dubois and Fortemps, 1999] is completely inefficient.
- Solving the Winner Determination Problem using Constraint Programming with our model is not a good idea.

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[1].

[1] We also used CPLEX with the one based on the cardinality combinator

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[1].

[1] We also used CPLEX with the one based on the cardinality combinator

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[1].

[1] We also used CPLEX with the one based on the cardinality combinator

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[1].

[1] We also used CPLEX with the one based on the cardinality combinator

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

Future work

- **More about leximin-optimality:**
  - Comparing our algorithms to a numerical approach of leximin (*i.e* by translating the leximin preorder to 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 modeling of fairness (*e.g.* OWA), see [Ogryczak, 2006].
  - Applying the algorithms to other practical fields and applications.

Ogryczak, W. (2006).
Bicriteria models for fair resource allocation.
In Endriss, U. and Lang, J., editors, *Proc. of COMSOC'06*, pages 380–393, Amsterdam.

This is the end.

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