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
sylvain.bouveret@onera.fr and michel.lemaitre@onera.fr

Abstract

We study the problem of computing a leximin-optimal solution of a constraint network. This problem is highly motivated by fairness and efficiency requirements in many real-world applications implying human agents. We compare several generic algorithms which solve this problem in a constraint programming framework. The first one is entirely original, and the other ones are partially based on existing works adapted to fit with this problem.

1 Introduction

Many advances have been done in recent years in modeling and solving combinatorial problems with constraint programming (CP). These advances concern, among others, the ability of this framework to deal with human reasoning schemes, such as, for example, the expression of preferences with soft constraints. However, one aspect of importance has only received a few attention in the constraints community to date: the way to handle fairness requirements in multiagent combinatorial problems.

The seek for fairness stands as a subjective but strong requirement in a wide set of real-world problems implying human agents. It is particularly relevant in crew or worker timetabling and rostering problems, or the optimization of long and short-term planning for firemen and emergency services. Fairness is also ubiquitous in multiagent resource allocation problems, like, among others, bandwidth allocation among network users, fair share of airspace and airport resources among several airlines or Earth observing satellite scheduling and sharing problems [Lemaître *et al.*, 1999].

In spite of the wide range of problems concerned by fairness issues, it often lacks a theoretical and generic approach. In many Constraint Programming and Operational Research works, fairness is only enforced by specific heuristic local choices guiding the search towards supposed equitable solutions. However, a few works may be cited for their approach of this fairness requirement. [Lemaître *et al.*, 1999] make use of an Earth observation satellite scheduling and sharing problem to investigate three ways of handling fairness among agents in the context of constraint satisfaction. More recently [Pesant and Régin, 2005] proposed a new constraint based on statistics, which enforces the relative balance of a given set of variables, and can possibly be used to ensure a kind of equity among a set of agents. Equity is also studied in Operational Research, with for example [Ogryczak and Śliwiński, 2003], who investigate a way of solving linear programs by aggregating multiple criteria using an Ordered Weighted Average Operator (OWA) [Yager, 1988]. Depending on the weights used in the OWA, this kind of aggregators can provide equitable compromises.

Microeconomy and Social Choice theory provide an important literature on fairness in collective decision making. From this theoretical background we borrow the idea of representing the agents preferences by *utility* levels, and we adopt the *leximin* preorder on utility profiles for conveying the fairness and efficiency requirements. Being a refinement of the *maximin* approach[1], it has an inclination to fairness, while avoiding the so-called *drowning effect* of this approach.

Apart from the fact that it conveys and formalizes the concept of equity in multiagent contexts, the leximin preorder is also a subject of interest in other contexts, such as fuzzy CSP [Fargier *et al.*, 1993], and symmetry-breaking in constraint satisfaction problems [Frisch *et al.*, 2003].

This contribution is organized as follows. Section 2 gives a minimal background in social choice theory and justifies the interest of the leximin preorder as a fairness criterion. Section 3 defines the search for leximin-optimality in a constraint programming framework. The main contribution of this paper is Section 4, which presents three algorithms for computing leximin-optimal solutions, the first one being entirely original, and the other ones adapted from existing works. The proposed algorithms have been implemented and tested within a constraint programming system. Section 5 presents an experimental comparison of these algorithms.

2 Background on social choice theory

We first introduce some notations. Calligraphic letters (*e.g.* $\mathcal{X}$) will stand for sets. Vectors will be written with an arrow (*e.g.* $\overrightarrow{x}$), or between brackets (*e.g.* $\langle x_1, \ldots, x_n \rangle$). $f(\overrightarrow{x})$ will be used as a shortcut for $\langle f(x_1), \ldots, f(x_n) \rangle$. Vector $\overrightarrow{x}^{\uparrow}$ will stand for the vector composed by each element of $\overrightarrow{x}$ rearranged in increasing order. We will write $x_i^{\uparrow}$ the i^{th} component of vector $\overrightarrow{x}^{\uparrow}$. Finally, the interval of integers between k and l will be written $[\![k, l]\!]$.

[1] Trying to maximize the utility of the unhappiest agent.

2.1 Collective decision making and welfarism

Let $\mathcal{N}$ be a set of n agents, and $\mathcal{S}$ be a set of admissible alternatives concerning all of them, among which a benevolent arbitrator has to choose one. The most classical model describing this situation is *welfarism* (see *e.g.* [Keeney and Raiffa, 1976; Moulin, 1988]): the choice of the arbitrator is made on the basis of the utility levels enjoyed by the individual agents and on those levels only. Each agent $i \in \mathcal{N}$ has an individual utility function u_i that maps each admissible alternative $s \in \mathcal{S}$ to a numerical index $u_i(s)$. We make here the classical assumption that the individual utilities are comparable between the agents[2]. Therefore each alternative s can be attached to a single *utility profile* $\langle u_1(s), \ldots, u_n(s)\rangle$. According to welfarism, comparing two alternatives is performed by comparing their respective utility profiles.

A standard way to compare individual utility profiles is to aggregate each of them into a *collective utility* index, standing for the collective welfare of the agents community. If g is a well-chosen aggregation function, we thus have a collective utility function uc that maps each alternative s to a collective utility level $uc(s) = g(u_1(s), \ldots, u_n(s))$. An optimal alternative is one of those maximizing the collective utility.

2.2 The leximin preorder as a fairness and efficiency criterion

The main difficulty of equitable decision problems is that we have to reconcile the contradictory wishes of the agents. Since generally no solution fully satisfies everyone, the aggregation function g must lead to fair and Pareto-efficient[3] compromises.

The problem of choosing the right aggregation function g is far beyond the scope of this paper. We only describe the two classical ones corresponding to two opposite points of view on social welfare[4]: classical utilitarianism and egalitarianism. The rule advocated by the defenders of classical utilitarianism is that the best decision is the one that maximizes the sum of individual utilities (thus corresponding to $g = +$). However this kind of aggregation function can lead to huge differences of utility levels among the agents, thus ruling out this aggregator in the context of equitable decisions. From the egalitarian point of view, the best decision is the one that maximizes the happiness of the least satisfied agent (thus corresponding to $g = \min$). Whereas this kind of aggregation function is particularly well-suited for problems in which fairness is essential, it has a major drawback, due to the idempotency of the $\min$ operator, and known as "drowning effect" in the community of fuzzy CSP (see *e.g.*[Dubois and Fortemps, 1999]). Indeed, it leaves many alternatives indistinguishable, such as for example the ones with utility profiles $\langle 0, \ldots, 0\rangle$ and $\langle 1000, \ldots, 1000, 0\rangle$, even if the second one appears to be much better than the first one. In other words, the $\min$ aggregation function can lead to non Pareto-optimal decisions, which is not desirable.

The leximin preorder is a well-known refinement of the order induced by the $\min$ function that overcomes this drawback. It is classically introduced in the social choice literature (see [Moulin, 1988]) as the social welfare ordering that reconcile egalitarianism and Pareto-efficiency, and also in fuzzy CSP [Fargier *et al.*, 1993]. It is defined as follows:

Definition 1 (leximin preorder [Moulin, 1988]) *Let $\overrightarrow{x}$ and $\overrightarrow{y}$ be two vectors of $\mathbb{N}^n$. $\overrightarrow{x}$ and $\overrightarrow{y}$ are said leximin-indifferent (written $\overrightarrow{x} \sim_{leximin} \overrightarrow{y}$) if and only if $\overrightarrow{x}^{\uparrow} = \overrightarrow{y}^{\uparrow}$. The vector $\overrightarrow{y}$ is leximin-preferred to $\overrightarrow{x}$ (written $\overrightarrow{x} \prec_{leximin} \overrightarrow{y}$) if and only if $\exists i \in [\![0, n-1]\!]$ such that $\forall j \in [\![1, i]\!]$, $x_j^{\uparrow} = y_j^{\uparrow}$ and $x_{i+1}^{\uparrow} < y_{i+1}^{\uparrow}$. We write $\overrightarrow{x} \preceq_{leximin} \overrightarrow{y}$ for $\overrightarrow{x} \prec_{leximin} \overrightarrow{y}$ or $\overrightarrow{x} \sim_{leximin} \overrightarrow{y}$. The binary relation $\preceq_{leximin}$ is a total preorder.*

In other words, the leximin preorder is the lexicographic preorder over ordered utility vectors. For example, we have $\langle 4, 1, 5, 1\rangle \prec_{leximin} \langle 2, 2, 1, 2\rangle$.

A known result is that no collective utility function can represent the leximin preorder[5], unless the set of possible utility profiles is finite. In this latter case, it can be represented by the following non-linear functions: $g_1 : \overrightarrow{x} \mapsto -\sum_{i=i}^{n} n^{-x_i}$ (adapted for leximin from a remark in [Frisch *et al.*, 2003]) and $g_2 : \overrightarrow{x} \mapsto -\sum_{i=1}^{n} x_i^{-q}$, where $q > 0$ is large enough [Moulin, 1988]. Using this kind of functions has however a main drawback: it rapidly becomes unreasonable when the upper bound of the possible values of $\overrightarrow{x}$ increases. Moreover, it hides the semantics of the leximin preorder and hinders the computational benefits we can possibly take advantage of.

In the following, we will use the leximin preorder as a criterion for ensuring fairness and Pareto-efficiency, and we will be seeking the non-dominated solutions in the sense of the leximin preorder. Those solutions will be called leximin-optimal. This problem will be expressed in the next section in a CP framework.

3 Constraint programming and leximin-optimality

The constraint programming framework is an effective and flexible tool for modeling and solving many different combinatorial problems such as planning and scheduling problems, resource allocation problems, or configuration problems. This paradigm is based on the notion of *constraint network* [Montanari, 1974]. A constraint network consists of a set of variables $\mathcal{X} = \{X_1, \ldots, X_p\}$ (in the following, variables will be written with uppercase letters), a set of associated domains $\mathcal{D} = \{\mathcal{D}_{X_1}, \ldots, \mathcal{D}_{X_p}\}$, where $\mathcal{D}_{X_i}$ is the set of possible values for X_i, and a set of constraints $\mathcal{C}$, where each $c \in \mathcal{C}$ specifies a set of allowed tuples $\mathcal{R}(c)$ over a set of variables $\mathcal{X}(c)$. We make the additional assumption that all the domains are in $\mathbb{N}$, and we will use the following notations: $\underline{X} = \min(\mathcal{D}_X)$ and $\overline{X} = \max(\mathcal{D}_X)$.

[2] In other words, they are expressed using a common utility scale.

[3] A decision is Pareto-efficient if and only if we cannot strictly increase the satisfaction of an agent unless we strictly decrease the satisfaction of another agent. Pareto-efficiency is generally taken as a basic postulate in collective decision making.

[4] Compromises between these two extremes are possible. See *e.g.* [Moulin, 2003, page 68] or [Yager, 1988] (*OWA aggregators*).

[5] In other words there is no g such that $\overrightarrow{x} \preceq_{leximin} \overrightarrow{y} \Leftrightarrow g(\overrightarrow{x}) \leq g(\overrightarrow{y})$. See [Moulin, 1988].

An instantiation v of a set $\mathcal{S}$ of variables is a function that maps each variable $X \in \mathcal{S}$ to a value $v(X)$ of its domain $\mathcal{D}_X$. If $\mathcal{S} = \mathcal{X}$, this instantiation is said to be complete, otherwise it is partial. If $\mathcal{S}' \subsetneq \mathcal{S}$, the projection of an instantiation of $\mathcal{S}$ over $\mathcal{S}'$ is the restriction of this instantiation to $\mathcal{S}'$ and is written $v_{\downarrow \mathcal{S}'}$. An instantiation is said to be consistent if and only if it satisfies all the constraints. A complete consistent instantiation of a constraint network is called a solution. The set of solutions of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is written $sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$. We will also write $v[X \leftarrow a]$ the instantiation v where the value of X is replaced by a.

Given a constraint network, the problem of determining whether it has a solution is called a *Constraint Satisfaction Problem* (CSP) and is NP-complete. The CSP can be classically adapted to become an optimization problem in the following way. Given a constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and an *objective* variable $O \in \mathcal{X}$, find the value m of $\mathcal{D}_O$ such that $m = \max\{v(O) \mid v \in sol(\mathcal{X}, \mathcal{D}, \mathcal{C})\}$. We will write $max(\mathcal{X}, \mathcal{D}, \mathcal{C}, O)$ for the subset of those solutions that maximize the objective variable O.

Expressing a collective decision making problem with a numerical collective utility criterion as a CSP with objective variable is straightforward: consider the collective utility as the objective variable, and link it to the variables representing individual utilities with a constraint. However this cannot directly encode our problem of computing a leximin-optimal solution, which is a kind of multicriteria optimization problem. We introduce formally the MaxLeximinCSP problem as follows :

Definition 2 (MaxLeximinCSP problem)
Input: *a constraint network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; a vector of variables $\overrightarrow{U} = \langle U_1, \ldots, U_n \rangle \in \mathcal{X}^n$, called an* objective vector.
Output: *"Inconsistent" if $sol(\mathcal{X}, \mathcal{D}, \mathcal{C}) = \emptyset$. Otherwise a solution $\widehat{v}$ such that $\forall v \in sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$, $v(\overrightarrow{U}) \preceq_{leximin} \widehat{v}(\overrightarrow{U})$.*

We describe in the next section several generic constraint programming algorithms that solve this problem. The first one is entirely original, and the other ones are based on existing works that are adapted to fit with our problem.

4 Proposed algorithms

4.1 Using a cardinality combinator

Our first algorithm is based on an iterative computation of the components of the leximin-optimal vector. It first computes the maximal value y_1 such that there is a solution v with $\forall i$, $y_1 \leq v(U_i)$, or in other words $\sum_i (y_1 \leq v(U_i)) = n$, where by convention the value of $(y_1 \leq v(U_i))$ is 1 if the inequality is satisfied and 0 otherwise[6]. Then, after having fixed this value for y_1, it computes the maximal value y_2 such that there is a solution v with $\sum_i (y_2 \leq v(U_i)) \geq n - 1$, and so on until the maximal value y_n such that there is a solution v with $\sum_i (y_n \leq v(U_i)) \geq 1$.

To enforce the constraint on the y_i, we make use of the meta-constraint **AtLeast**, derived from a *cardinality combinator* introduced by [Van Hentenryck *et al.*, 1992], and present in most of CP systems:

[6]This convention is inspired by the constraint modeling language OPL [Van Hentenryck, 1999].

Definition 3 (Meta-constraint AtLeast) *Let Γ be a set of p constraints, and $k \in [\![1, p]\!]$ be an integer. The meta-constraint* $\mathbf{AtLeast}(\Gamma, k)$ *holds on the union of the scopes of the constraints in Γ, and allows a tuple if and only if at least k constraints from Γ are satisfied.*

Due to its genericity, this meta-constraint cannot provide very efficient filtering procedures. In our case where the constraints of Γ are linear, this meta-constraint is simply a counting constraint, and bound-consistency can be achieved in $O(n)$. The specific meta-constraint **AtLeast** can also be implemented with a set of linear constraints [Garfinkel and Nemhauser, 1972, p.11], by introducing n 0–1 variables $\{\Delta_1, \ldots, \Delta_n\}$, and a set of linear constraints $\{X_1 + \Delta_1 \overline{Y} \geq Y, \ldots, X_n + \Delta_n \overline{Y} \geq Y, \sum_{i=1}^{n} \Delta_i \leq n - k\}$.

Our first approach for computing a leximin-optimal solution is presented in algorithm 1.

Algorithm 1: Computation of a leximin-optimal solution using a cardinality combinator.

input : A const. network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle U_1, \ldots, U_n \rangle \in \mathcal{X}^n$
output: A solution to the MaxLeximinCSP problem

1 **if** $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ *"Inconsistent"* **return** *"Inconsistent"*;
2 $(\mathcal{X}_0, \mathcal{D}'_0, \mathcal{C}_0) \leftarrow (\mathcal{X}, \mathcal{D}, \mathcal{C})$;
3 **for** $i \leftarrow 1$ **to** n **do**
4 $\quad \mathcal{X}_i \leftarrow \mathcal{X}_{i-1} \cup \{Y_i\}$;
5 $\quad \mathcal{D}_i \leftarrow \mathcal{D}'_{i-1} \cup \{\mathcal{D}_{Y_i}\}$ with $\mathcal{D}_{Y_i} = [\![\min_j(\underline{U_j}), \max_j(\overline{U_j})]\!]$;
6 $\quad \mathcal{C}_i \leftarrow \mathcal{C}_{i-1} \cup \{\mathbf{AtLeast}(\{Y_i \leq U_1, \ldots, Y_i \leq U_n\}, n - i + 1)\}$;
7 $\quad \widehat{v}_{(i)} \leftarrow \mathbf{maximize}(\mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i, Y_i)$;
8 $\quad \mathcal{D}'_i \leftarrow \mathcal{D}_i$ with $\mathcal{D}_{Y_i} \leftarrow \{\widehat{v}_{(i)}(Y_i)\}$;
9 **return** $\widehat{v}_{(n) \downarrow \mathcal{X}}$;

The functions **solve** and **maximize** (the detail of which is the concern of solving techniques for constraints satisfaction problems) of lines 1 and 7 respectively return one solution $v \in sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$ (or "Inconsistent" if such a solution does not exist), and an optimal solution $\widehat{v} \in max(\mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i, Y_i)$ (or "Inconsistent" if $sol(\mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i) = \emptyset$). We assume – contrary to usual constraint solvers – that these two functions do not modify the input constraint network.

The following example illustrates the behavior of the algorithm. It is a simple resource allocation problem, where 3 objects must be allocated to 3 agents, with the following constraints: each agent must get one and only one object, and one object cannot be allocated to more than one agent (i.e. a perfect matching agent/objects). A utility is associated with each pair (agent,object) with respect to the array above.

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

This problem has 6 feasible solutions (one for each permutation of $[\![1, 3]\!]$), producing the 6 utility profiles shown in the columns of the following array:

	p_1	p_2	p_3	p_4	p_5	p_6
u_1	3	3	5	5	7	7
u_2	9	8	3	8	3	9
u_3	1	7	1	3	7	3

The algorithm runs in 3 steps: **Step 1:** After having introduced one variable Y_1, we look for the maximal value $\widehat{y_1}$ of Y_1 such that each (**at least** 3) agent gets at least Y_1. We find $\widehat{y_1} = 3$. The variable Y_1 is fixed to this value, implicitly removing profiles p_1 and p_3. **Step 2:** After having introduced one variable Y_2, we look for the maximal value $\widehat{y_2}$ of Y_2 such that **at least** 2 agents get at least Y_2. We find $\widehat{y_2} = 7$. The variable Y_2 is fixed to this value, implicitly removing profile p_4. **Step 3:** After having introduced one variable Y_3, we look for the maximal value $\widehat{y_3}$ of Y_3 such that **at least** 1 agent gets at least Y_3. We find $\widehat{y_3} = 9$. Only one instantiation maximizes Y_3: p_6. Finally, the returned leximin-optimal allocation is: $a_1 \leftarrow o_3$, $a_2 \leftarrow o_2$ and $a_3 \leftarrow o_1$.

Proposition 1 *If the two functions* **maximize** *and* **solve** *are both correct and both halt, then algorithm 1 halts and solves the* MaxLeximinCSP *problem.*

In the next proofs, we will write sol_i and sol'_i for respectively $sol(\mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i)$ and $sol(\mathcal{X}_i, \mathcal{D}'_i, \mathcal{C}_i)$. We will also write $(sol_i)_{\downarrow \mathcal{X}_j}$ and $(sol'_i)_{\downarrow \mathcal{X}_j}$ for the same sets of solutions projected on $\mathcal{X}_j$ (with $j < i$). We can notice that $sol_0 = sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$, and that $\forall i,\ sol'_i \subseteq sol_i$.

Lemma 1 *Let $\overrightarrow{x}$ be a vector of size n. At least $n - i + 1$ components of $\overrightarrow{x}$ are greater than or equal to $x_i^{\uparrow}$.*

The proof of this useful lemma is obvious, so we omit it.

Lemma 2 *If $sol_0 \neq \emptyset$ then $\widehat{v}_{(n)}$ is well-defined and not equal to "Inconsistent".*

Proof: Let $i \in [\![1, n]\!]$, suppose that $sol'_{i-1} \neq \emptyset$, and let $v_{(i)} \in sol'_{i-1}$. Then extending $v_{(i)}$ by instantiating Y_i to $\min_j(\underline{U_j})$ leads to a solution of $(\mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i)$ (only one constraint has been added and it is satisfied by the latter instantiation). Therefore $sol_i \neq \emptyset$ and, if **maximize** is correct, $\widehat{v}_{(i)} \neq$ "Inconsistent" and $\widehat{v}_{(i)} \in sol'_i$. So, $sol'_i \neq \emptyset$. It proves lemma 2 by induction. ■

Lemma 3 *If $sol_0 \neq \emptyset$, then $(\widehat{v}_{(n)})_{\downarrow \mathcal{X}_i} \in sol_i, \forall i \in [\![0, n]\!]$.*

Proof: We have $sol'_i \subseteq sol_i$, and $(sol_{i+1})_{\downarrow \mathcal{X}_i} \subseteq sol'_i$ (since from $(\mathcal{X}_i, \mathcal{D}'_i, \mathcal{C}_i)$ to $(\mathcal{X}_{i+1}, \mathcal{D}_{i+1}, \mathcal{C}_{i+1})$ we just add a constraint). More generally, $(sol'_i)_{\downarrow \mathcal{X}_j} \subseteq (sol_i)_{\downarrow \mathcal{X}_j}$, and $(sol_{i+1})_{\downarrow \mathcal{X}_j} \subseteq (sol'_i)_{\downarrow \mathcal{X}_j}$, as soon as $j \leq i$. Hence, $(\widehat{v}_{(n)})_{\downarrow \mathcal{X}_i} \in (sol'_n)_{\downarrow \mathcal{X}_i} \subseteq (sol_n)_{\downarrow \mathcal{X}_i} \subseteq \cdots \subseteq (sol_{i+1})_{\downarrow \mathcal{X}_i} \subseteq sol'_i \subseteq sol_i$. ■

Lemma 4 *If $sol_0 \neq \emptyset$, $\widehat{v}_{(n)}(\overrightarrow{y})$ is equal to $\widehat{v}_{(n)}(\overrightarrow{U})^{\uparrow}$.*

Proof: For all $i \in [\![1, n]\!]$, $(\widehat{v}_{(n)})_{\downarrow \mathcal{X}_i}$ is a solution of sol_i by lemma 3. By lemma 1, $(\widehat{v}_{(n)})_{\downarrow \mathcal{X}_i}[Y_i \leftarrow \widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow}]$ satisfies the cardinality constraint of iteration i, and is then a solution of sol_i. By definition of function **maximize**, we thus have $\widehat{v}_{(i)}(Y_i) \geq \widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow}$. Since $\widehat{v}_{(i)}(Y_i) = \widehat{v}_{(n)}(Y_i)$, we have $\widehat{v}_{(n)}(Y_i) \geq \widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow}$.

Since $\widehat{v}_{(n)}$ is a solution of sol_n, at least $n - i + 1$ numbers from vector $\widehat{v}_{(n)}(\overrightarrow{U})$ are greater than or equal to $\widehat{v}_{(n)}(Y_i)$. At least the $n - i + 1$ greatest numbers from $\widehat{v}_{(n)}(\overrightarrow{U})$ must then be greater than or equal to $\widehat{v}_{(n)}(Y_i)$. These components include $\widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow}$, which leads to $\widehat{v}_{(n)}(Y_i) \leq \widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow}$, proving the lemma. ■

We can now put things together and prove proposition 1.

Proof of proposition 1: If $sol(\mathcal{X}, \mathcal{D}, \mathcal{C}) = \emptyset$, and if **solve** is correct, then algorithm 1 obviously returns "Inconsistent". Otherwise, following lemma 2, it outputs an instantiation $(\widehat{v}_{(n)})_{\downarrow \mathcal{X}}$ which is, according to lemma 3, a solution of $(\mathcal{X}_0, \mathcal{D}_0, \mathcal{C}_0) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$.

Suppose that there is a $v \in sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$ such that $v(\overrightarrow{U}) \succ_{leximin} \widehat{v}_{(n)}(\overrightarrow{U})$. Then following definition 1, $\exists i \in [\![1, n]\!]$ such that $\forall j < i$, $v(\overrightarrow{U})_j^{\uparrow} = \widehat{v}_{(n)}(\overrightarrow{U})_j^{\uparrow}$ and $v(\overrightarrow{U})_i^{\uparrow} > \widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow}$. Let $v_{(i)}^+$ be the extension of v respectively instantiating $Y_1, \ldots Y_{i-1}$ to $\widehat{v}_{(n)}(Y_1), \ldots \widehat{v}_{(n)}(Y_{i-1})$ and Y_i to $v(\overrightarrow{U})_i^{\uparrow}$. Following lemma 4, $\forall j$, $\widehat{v}_{(n)}(Y_j) = \widehat{v}_{(n)}(\overrightarrow{U})_j^{\uparrow}$. By gathering all the previous equalities, we have $\forall j < i$ $v_{(i)}^+(Y_j) = \widehat{v}_{(n)}(Y_j) = v(\overrightarrow{U})_j^{\uparrow} = (v_{(i)}^+(\overrightarrow{U}))_j^{\uparrow}$. We also have $v_{(i)}^+(Y_i) = v(\overrightarrow{U})_i^{\uparrow} = (v_{(i)}^+(\overrightarrow{U}))_i^{\uparrow}$. By lemma 1, $\forall j \leq i$ at least $n - j + 1$ numbers from $(v_{(i)}^+(\overrightarrow{U}))$ are greater than or equal to $v_{(i)}^+(Y_j)$, proving that $v_{(i)}^+$ satisfies all the cardinality constraints at iteration i. Since it also satisfies each constraint in $\mathcal{C}$ and maps each variable of $\mathcal{X}_i$ to one of its possible values, it is a solution of sol_i, and $v_{(i)}^+(Y_i) = v(\overrightarrow{U})_i^{\uparrow} > \widehat{v}_{(n)}(\overrightarrow{U})_i^{\uparrow} = \widehat{v}_{(i)}(Y_i)$. It contradicts the definition of **maximize**, proving the proposition 1. ■

4.2 Using a sorting constraint

Our second algorithm is directly based on the definition 1 of the leximin preorder, which involves the sorted version of the objective vector. This can be naturally expressed in the CP paradigm by introducing a vector of variables $\overrightarrow{Y}$ and enforcing the constraint $\mathbf{Sort}(\overrightarrow{U}, \overrightarrow{Y})$ which is defined as follows:

Definition 4 (Constraint Sort) *Let $\overrightarrow{X}$ and $\overrightarrow{X}'$ be two vectors of variables of the same length, and v be an instantiation. The constraint $\mathbf{Sort}(\overrightarrow{X}, \overrightarrow{X}')$ holds on the set of variables being either in $\overrightarrow{X}$ or in $\overrightarrow{X}'$, and is satisfied by v if and only if $v(\overrightarrow{X}')$ is the sorted version of $v(\overrightarrow{X})$ in increasing order.*

This constraint has been particularly studied in two works, which both introduce a filtering algorithm for enforcing bound consistency on this constraint. The first algorithm comes from [Bleuzen-Guernalec and Colmerauer, 1997] and runs in $O(n \log n)$ (n being the size of $\overrightarrow{X}$). [Mehlhorn and Thiel, 2000] designed a simpler algorithm that runs in $O(n)$ plus the time required to sort the interval endpoints of $\overrightarrow{X}$, which can asymptotically be faster than $O(n \log n)$.

Algorithm 2: Computation of a leximin-optimal solution using a sorting constraint.

input : A const. network $(\mathcal{X}, \mathcal{D}, \mathcal{C})$; $\langle U_1, \ldots, U_n \rangle \in \mathcal{X}^n$
output: A solution to the MaxLeximinCSP problem

if $\mathbf{solve}(\mathcal{X}, \mathcal{D}, \mathcal{C}) =$ *"Inconsistent"* **return** *"Inconsistent"*;
$\mathcal{X}' \leftarrow \mathcal{X} \cup \{Y_1, \ldots, Y_n\}$;
$\mathcal{D}' \leftarrow \mathcal{D} \cup \{\mathcal{D}_{Y_1}, \ldots, \mathcal{D}_{Y_n}\}, \mathcal{D}_{Y_i} = [\![\min_j(\underline{U_j}), \max_j(\overline{U_j})]\!]$;
$\mathcal{C}' \leftarrow \mathcal{C} \cup \{\mathbf{Sort}(\overrightarrow{U}, \overrightarrow{Y})\}$;
for $i \leftarrow 1$ **to** n **do**
  $\widehat{v}_{(i)} \leftarrow \mathbf{maximize}(\mathcal{X}', \mathcal{D}', \mathcal{C}', Y_i)$;
  $\mathcal{D}_{Y_i} \leftarrow \{\widehat{v}_{(i)}(Y_i)\}$;
return $\widehat{v}_{(n)\downarrow \mathcal{X}}$;

Proposition 2 *If the two functions* **maximize** *and* **solve** *are both correct and both halt, then algorithm 2 halts and solves the* MaxLeximinCSP *problem.*

Proof: If $sol(\mathcal{X},\mathcal{D},\mathcal{C}) = \emptyset$ and if **solve** is correct, then algorithm 2 obviously returns "Inconsistent". We will suppose in the following that $sol(\mathcal{X},\mathcal{D},\mathcal{C}) \neq \emptyset$ and we will use the following notations: sol_i and sol'_i are the sets of solutions of $(\mathcal{X}',\mathcal{D}',\mathcal{C}')$ respectively at the beginning and at the end of iteration i.

We have obviously $\forall i \in [\![1, n-1]\!]$ $sol_{i+1} = sol'_i$, which proves that if $sol_i \neq \emptyset$, then the call to **maximize** at line 6 does not return "Inconsistent", and $sol_{i+1} \neq \emptyset$. Thus $\widehat{v}_{(n)}$ is well-defined, and obviously $(\widehat{v}_{(n)})_{\downarrow\mathcal{X}}$ is a solution of $(\mathcal{X},\mathcal{D},\mathcal{C})$.

We note $\widehat{v} = \widehat{v}_{(n)}$ the instantiation computed by the last **maximize** in algorithm 2. Suppose that there is an instantiation $v \in sol(\mathcal{X},\mathcal{D},\mathcal{C})$ such that $\widehat{v}(\overrightarrow{U}) \prec_{leximin} v(\overrightarrow{U})$. We define v^+ the extension of v that instantiates each y_i to $v(\overrightarrow{U})^{\uparrow}_i$. Then, due to constraint **Sort**, $\widehat{v}(\overrightarrow{Y})$ and $v^+(Y)$ are the respective sorted version of $\widehat{v}(\overrightarrow{U})$ and $v^+(U)$. Following definition 1, there is an $i \in [\![0, n-1]\!]$ such that $\forall j \in [\![1, i]\!]$, $\widehat{v}(Y_j) = v^+(Y_j)$ and $\widehat{v}(Y_{i+1}) < v^+(Y_{i+1})$. Due to line 7, we have $\widehat{v}(Y_{i+1}) = \widehat{v}_{(n)}(Y_{i+1}) = \widehat{v}_{(i+1)}(Y_{i+1})$. Thus v^+ is a solution in $max(\mathcal{X}',\mathcal{D}',\mathcal{C}',Y_{i+1})$ with objective value $v^+_{(i+1)}(Y_{i+1})$ strictly greater than $\widehat{v}_{(i+1)}(Y_{i+1})$, which contradicts the hypothesis about **maximize**. ■

4.3 Using a multiset ordering constraint

Our third algorithm computing a leximin-optimal solution is perhaps the most intuitive one. It proceeds in a pseudo branch and bound manner: it computes a first solution, then tries to improve it by specifying that the next solution has to be better (in the sense of the leximin preorder) than the current one, and so on until the constraint network becomes inconsistent. This approach is based on the following constraint:

Definition 5 (Constraint Leximin) *Let $\overrightarrow{X}$ be a vector of variables, $\overrightarrow{\lambda}$ be a vector of integers, and v be an instantiation. The constraint* $\mathbf{Leximin}(\overrightarrow{\lambda}, \overrightarrow{X})$ *holds on the set of variables belonging to $\overrightarrow{X}$, and is satisfied by v if and only if $\overrightarrow{\lambda} \prec_{leximin} \overrightarrow{v(X)}$.*

Although this constraint does not exist in the literature, the work of [Frisch *et al.*, 2003] introduces an algorithm for enforcing generalized arc-consistency on a quite similar constraint: the multiset ordering constraint, which is, in the context of multisets, the equivalent of a leximax[7] constraint on vectors of variables. At the price of some slight modifications, the algorithm they introduce can easily be used to enforce the latter constraint **Leximin**.

Proposition 3 *If the function* **solve** *is correct and halts, then algorithm 3 halts and solves the* MaxLeximinCSP *problem.*

The proof is rather straightforward, so we omit it.

4.4 Other approaches

In the context of fuzzy constraints, two algorithms dedicated to the computation of leximin-optimal solutions have been published by [Dubois and Fortemps, 1999]. These algorithms work by enumerating, at each step, all the subsets of fuzzy constraints (corresponding to our agents) having a property connected to the notion of consistency degree.

[Ehrgott, 2000, p. 162] describes two very simple algorithms for solving the closely related "Lexicographic Max-Ordering" problem (that could be called "leximax-optimal" in our terms). However, they do not seem realistic in the context of combinatorial problems, since they are based on an enumeration of all utility profiles.

Algorithm 3: Computation of a leximin-optimal solution in a branch and bound manner.

input : A const. network $(\mathcal{X},\mathcal{D},\mathcal{C})$; $\langle U_1,\ldots,U_n\rangle \in \mathcal{X}^n$
output: A solution to the MaxLeximinCSP problem

1 $\widehat{v} \leftarrow$ **null**; $v \leftarrow$ **solve**$(\mathcal{X},\mathcal{D},\mathcal{C})$;
2 **while** $v \neq$ *"Inconsistent"* **do**
3 $\quad \widehat{v} \leftarrow v$;
4 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{Leximin}(\widehat{v}(\overrightarrow{U}), \overrightarrow{U})\}$;
5 $\quad v \leftarrow$ **solve**$(\mathcal{X},\mathcal{D},\mathcal{C})$;
6 **if** $\widehat{v} \neq$ **null then return** $\widehat{v}$ **else return** *"Inconsistent"*;

5 Experimental results

The algorithms 1, 2, 3 and the first algorithm proposed in [Dubois and Fortemps, 1999] have been implemented and tested using the constraint programming tool CHOCO [Laburthe, 2000]. So as to test them on realistic instances, we have extracted, from a real-world problem, a simplified multiagent resource allocation problem. In this problem, the resource is a set of objects $\mathcal{O}$, that must be allocated to some agents under volume and consumption constraints. The individual utility functions are specified by a set of weights $w_{a,o}$ (one per pair $(agent, object)$): given an allocation of the objects, the individual utility of an agent i is the sum of the weights $w_{i,o}$ of the objects o that she receives. The weights can be generated uniformly or can be concentrated around some powers of 10, so as to simulate some kind of priorities[8].

We have developed a customizable generator of random instances, available online[9]. We tested our algorithms on several instances, with very different characteristics, leading to very different kind of problems. Here is a brief description of each kind of instances appearing in table 1 (by default, the weights are non-uniformly distributed, and the constraints are of medium tightness):

(1) 10 agents, 100 objects. (2) 4 agents, 100 objects. (3) 20 agents, 40 objects. (4) 10 agents, 100 objects, low-tightness constraints. (5) 10 agents, 100 objects, hard-tightness constraints. (6) 10 agents, 30 objects, uniform weights (with low values), hard-tightness constraints. (7) 4 agents, 150 objects.

The results from table 1 show that algorithm 1 has the best running times with most of the instances, followed by algorithm 2 which is almost as fast, but is less efficient when the number of agents increases (instances of kind 3), whereas algorithm 3 is better on this kind of instances. As expected, the algorithm from [Dubois and Fortemps, 1999] explodes when

[7] The leximax is based on an increasing reordering of the values, instead of a decreasing one for leximin.

[8] Approximating the conditions of our real-world application.

[9] http://www.cert.fr/dcsd/THESES/sbouveret/benchmark/

kind	Algorithm 1 (**AtLeast**)				Algorithm 2 (**Sort**)				Algorithm 3 (**Leximin**)				[Dubois and Fortemps, 1999]			
	avg	min	max	N%	avg	min	max	N%	avg	min	max	N%	avg	min	max	N%
1	**0.7**	0.6	1.8	100	0.9	0.7	1.1	100	3	0.2	34.5	100	9	8.8	9.6	100
2	**0.6**	0.2	17.7	100	0.7	0.2	19.1	100	6.9	1.3	43.9	100	2.5	2	18.5	100
3	20.2	1.5	117	100	97.9	2	551	100	**16.7**	0.4	99.2	100	600	600	600	0
4	**0.8**	0.7	1.2	100	0.9	0.8	1	100	2	0.5	8	100	9.1	8.9	9.2	100
5	4.2	0.8	57.9	100	**3.9**	0.8	83.5	100	6.4	0.2	186.6	100	12.4	9.1	53.7	100
6	2.1	0.6	4.3	100	2.1	0.7	4.3	100	**0.7**	0.1	1.2	100	218.2	47.5	457.4	100
7	**101**	0.3	600	92	103	0.3	600	92	320	4.3	600	60	155.2	2.4	600	84

Table 1: CPU times (in seconds) and percentage of instances solved within 10 minutes for each algorithm. Each algorithm has been run on 50 instances of each kind, on a 1.6GHz Pentium M PC under Linux.

the number of equal components in the leximin-optimal vector increases (kinds (3) and (6)).

These results must however be considered with care, since they are subject to our implementation of the filtering algorithms. In particular, not every optimizations given in [Mehlhorn and Thiel, 2000] for the constraint **Sort** have been implemented yet. Moreover, the running times are highly affected by the variable choice heuristics. In our tests, we used the following particular heuristics, that are specially efficient: choose as the next variable to instantiate the one that will the most increase the lowest objective value (in our application problem we first allocate the objects that have the highest weight for the currently least satisfied agent).

6 Conclusion

The leximin preorder cannot be ignored when dealing with optimization problems in which some kind of fairness must be enforced between utilities of agents or equally important criteria. This paper brings a contribution to the computation of leximin-optimal solutions of combinatorial problems. It describes, in a constraint programming framework, three generic algorithms solving this problem. The first one, based on a cardinality combinator, is entirely new, and gives slightly better results than two algorithms based on the sort and leximin constraints.

References

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

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

[Ehrgott, 2000] M. Ehrgott. *Multicriteria Optimization*. Number 491 in Lecture Notes in Economics and Mathematical Systems. Springer, 2000.

[Fargier *et al.*, 1993] H. Fargier, J. Lang, and T. Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proc. of EUFIT'93*, Aachen, 1993.

[Frisch *et al.*, 2003] A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Multiset ordering constraints. In *Proc. of IJCAI'03*, February 2003.

[Garfinkel and Nemhauser, 1972] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. Wiley, 1972.

[Keeney and Raiffa, 1976] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives*. J. Wiley & Sons, 1976.

[Laburthe, 2000] F. Laburthe. CHOCO : implementing a CP kernel. In *Proc. of TRICS'2000, Workshop on techniques for implementing CP systems*, Singapore, 2000. http://sourceforge.net/projects/choco.

[Lemaître *et al.*, 1999] M. Lemaître, G. Verfaillie, and N. Bataille. Exploiting a Common Property Resource under a Fairness Constraint: a Case Study. In *Proc. of IJCAI-99*, pages 206–211, Stockholm, 1999.

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

[Montanari, 1974] U. Montanari. Network of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.*, 7:95–132, 1974.

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

[Moulin, 2003] H. Moulin. *Fair division and collective welfare*. MIT Press, 2003.

[Ogryczak and Śliwiński, 2003] W. Ogryczak and T. Śliwiński. On solving linear programs with the ordered weighted averaging objective. *Europen Journal of Operational Research*, (148):80–91, 2003.

[Pesant and Régin, 2005] G. Pesant and J-C. Régin. SPREAD: A balancing constraint based on statistics. In *Proc. of CP'05*, Sitges, Spain, 2005.

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

[Van Hentenryck, 1999] P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.

[Yager, 1988] R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Trans. on Syst., Man, and Cybernetics*, 18:183–190, 1988.