Constraints (2015) 20:362–380
DOI 10.1007/s10601-015-9191-0

New filtering for AtMostNValue and its weighted variant: A Lagrangian approach

Hadrien Cambazard[1,2] · Jean-Guillaume Fages[3]

Published online: 21 April 2015

Abstract The AtMostNValue global constraint, which restricts the maximum number of distinct values taken by a set of variables, is a well known NP-Hard global constraint. The weighted version of the constraint, AtMostWValue, where each value is associated with a weight or cost, is a useful and natural extension. Both constraints occur in many industrial applications where the number and the cost of some resources have to be minimized. This paper introduces a new filtering algorithm based on a Lagrangian relaxation for both constraints. This contribution is illustrated on problems related to facility location, which is a fundamental class of problems in operations research and management sciences. Preliminary evaluations show that the filtering power of the Lagrangian relaxation can provide significant improvements over the state-of-the-art algorithm for these constraints. We believe it can help to bridge the gap between constraint programming and linear programming approaches for a large class of problems related to facility location.

Keywords Global constraint · Filtering algorithm · Lagrangian relaxation · At most n values

1 Introduction

Many operations research and management science applications include software components to minimize the number and/or the cost of resource units that are necessary

✉ Hadrien Cambazard
hadrien.cambazard@grenoble-inp.fr

Jean-Guillaume Fages
jg.fages@cosling.com

[1] Univ. Grenoble Alpes, G-SCOP, 38000 Grenoble, France

[2] CNRS, G-SCOP, 38000 Grenoble, France

[3] COSLING S.A.S., 44307 Nantes Cedex 3, France

to run the business. Typical examples are plant location problems such as the uncapacitated facility location problem [11] or personnel scheduling problems [10]. Mixed integer linear programming and column generation are often used in these application fields, providing excellent results. Probably due to a lack of lower bounding techniques, constraint-programming (CP) approaches are not competitive. We investigate stronger filtering algorithms to fill the gap between integer linear programming (ILP) and CP on these problems.

This paper focuses on the AtMostNValue global constraint, which restricts the maximum number of distinct values taken by a set of variables, and its weighted variant, the AtMostWValue. These constraints have been introduced in Beldiceanu et al. [1, 2]. Filtering these constraints has been proved to be NP-hard [4]. Therefore, unless P=NP, no polynomial time algorithm can perform a complete filtering for these constraints. The design of an efficient filtering algorithm for the AtMostNValue constraint has been widely investigated by the CP community [4] and remains an active topic [5, 12, 16]. The state-of-the-art algorithm is a graph-based algorithm relying on minimum independent set relaxation [4]. It provides a very good trade-off between filtering and runtime. However, the resulting filtering may not be sufficient to deal with the most difficult problems. Nevertheless, it has been shown [4] that embedding the linear program associated with AtMostNValue into a global constraint could increase the filtering power of the model, often at the expense of runtime. This paper stems from this observation and suggests an alternative between those two state-of-the-art propagators.

The main contribution of this paper is a new filtering procedure, based on a Lagrangian relaxation, for both AtMostNValue and AtMostWValue.

Lagrangian relaxation (LR) is often seen as a class of algorithms to reformulate and solve an integer *linear* program [17]. An active area of research seeks to generalize theses algorithms to nonlinear optimization problems [27] and CP might be an appropriate framework to extend and benefit from Lagrangian relaxation in a more general context [15]. Our present goal is different and aims at embedding LR as a generic filtering mecanism for constraint solvers. The use of LR for propagating NP-hard global constraints is not new but has mostly been performed in very specific and applicative contexts [7, 8, 25, 26]. Sellman et al. [25] presents LR as a way to design powerful propagation by taking into account several constraints together. This idea is applied to a knapsack constraint coupled with a maximum weighted stable set constraint. Later, propators relying on Lagrangian relaxation are designed for the Multi-cost-regular and Weighted-circuit global constraints [3, 13, 21]. Our present work extends this line of research as we believe that many global constraints (and in particuler NP-hard global constraints involving costs) could be propagated using Lagrangian relaxation in a relatively generic manner [6]. This paper is investigating this idea for AtMostNValue and AtMostWValue.

The algorithm proposed in this paper has several advantages : First, from a software engineering point of view, it is simple to implement and it does not require any connection with a linear solver. Second, it provides a good tradeoff between filtering and runtime. Third, as opposed to the graph-based algorithm, it can be used to propagate the AtMostWValue global constraint for which no simple and efficient filtering algorithm exists. Thus, it is relevant to include it in a CP solver. Several design options are discussed and empirically evaluated. The practical impact of these contributions is evaluated on the p-median and a discrete facility location problem. Results show that the Lagrangian propagator we introduce for both AtMostNValue and AtMostWValue provides significant improvement over a CP approach, up to being competitive with an ILP approach.

2 Technical context

2.1 Notations

In this paper, we consider a set $\mathcal{X} = \{X_1, ..., X_n\}$ of n integer variables, taking their value in $\mathcal{V} = \{1, ..., m\}$, and an integer variable N taking its value in $[1, n]$. The global constraint ensuring that the number of distinct values taken by $\mathcal{X}$ is bounded by N is called AtMostNValue (see definition 1). We use the notation $dom(X)$ to represent the domain of the variable X, *i.e.,* its potential values. For any $X_i \in \mathcal{X}$, we have $dom(X_i) \subseteq \mathcal{V}$. Furthermore, for any $X_i \in \mathcal{X}$, let $\underline{X_i}$ and $\overline{X_i}$ respectively denote the lower and the upper bound of X_i. Therefore, $dom(X_i) \subseteq [\underline{X_i}, \overline{X_i}]$.

Definition 1 Given a set of integer variables $\mathcal{X}$ and an integer variable N, the AtMostNValue($\mathcal{X}$, N) constraint states that at most N distinct values are taken by $\mathcal{X}$, *i.e.,* $|\{j | \exists X_i \in \mathcal{X}, X_i = j\}| \leq N$.

We refer to the weighted extension of the constraint as AtMostWValue (see definition 2), but it has various names in the literature [2]. Therefore, N stands for *number*, whereas W stands for *weight*. To avoid any confusion, we introduce a cost variable W, which takes its value in $\mathbb{R}^+$, to represent the limit.

Definition 2 Given a set of integer variables $\mathcal{X}$, a variable $W \in R^+$ and a weight function $w : \mathcal{V} \rightarrow \mathbb{R}^+$, the AtMostWValue($\mathcal{X}$, W, w) constraint states that W is greater or equal to the sum of weights associated with all distinct values taken by $\mathcal{X}$, *i.e.,* $\sum_{\{j \in \mathcal{V} | \exists X_i \in \mathcal{X}, X_i = j\}} w(j) \leq W$.

Notice that we consider non-negative weights in the present paper for the sake of simplicity. Moreover, AtMostNValue($\mathcal{X}$, N) is a restricted case of AtMostWValue($\mathcal{X}$, N, w_1), where $w_1 : j \in \mathcal{V} \mapsto 1$. Therefore, we will focus on the more general AtMostWValue.

2.2 Explicit representation of used values

Most often, it is convenient to represent explicitly the distinct values which are taken by $\mathcal{X}$ with a set of binary variables $\mathcal{Y} = \{Y_1, ..., Y_m\}$. This may ease the declaration of other constraints of the model as well as the declaration of the search strategy. For instance, a common branching scheme is a two-stage approach where the values to be used are decided before starting their assignment to the variables in $\mathcal{X}$, *i.e.,* branching on variables in $\mathcal{Y}$ before variables in $\mathcal{X}$. $\mathcal{Y}$ is introduced to express reasonings that can not necessarily be expressed on $\mathcal{X}$. We can propagate that a particular value $j \in \mathcal{V}$ must be used, without having any variable in $\mathcal{X}$ assigned to j, because that information is encoded in the domain of Y_j. Therefore, we add $\mathcal{Y}$ to the signature of both global constraints.

Let us assume, without loss of generality, that values range from 1 to m. The channeling between $\mathcal{X}$ and $\mathcal{Y}$ requires to propagate the following relations:

$$\forall j \in [1, m], \qquad Y_j = 1 \Leftrightarrow \exists X_i \in \mathcal{X} | X_i = j$$

This is not straightforward to encode in CP solvers as they generally do not support existential constraints. Nevertheless, it can be implemented in a declarative way by introducing

a variable N_j for every value j, to count the number of variables in $\mathcal{X}$ that take value j, and adding the following constraint network to the model:

$$\bigwedge_{j\in[1,m]} \text{Occurrence}(j, \mathcal{X}, N_j) \wedge (Y_j = 1 \Leftrightarrow N_j > 0)$$

However, by computing the exact number of occurrences of every value in $\mathcal{X}$, this approach is performing unnecessary computations. Therefore, we implemented a more efficient *ad-hoc* propagator, for every value j, to propagate the equivalence $Y_j = 1 \Leftrightarrow \exists X_i \in \mathcal{X} | X_i = j$ directly, without introducing new variables. In the rest of the paper, we assume that this channeling between $\mathcal{X}$ and $\mathcal{Y}$ is part of both AtMostNValue and AtMostWValue. The following example will be used throughout the paper to illustrate ideas:

Example 1 Consider the 0/1 variables $\mathcal{Y} = \{Y_1, Y_2, Y_3, Y_4, Y_5\}$ ($dom(Y_i) = \{0, 1\}$ for every $i \in [1, 5]$) as well as $\mathcal{X} = \{X_1, X_2, X_3\}$ and variable N with the domains:

$$dom(X_1) = \{1, 2\},\ \ dom(X_2) = \{2, 3\},\ \ dom(X_3) = \{4, 5\},\ \ dom(N) = [0, 2]$$

Enforcing Generalized Arc-Consistency (GAC) for AtMostNValue($\mathcal{X}$, $\mathcal{Y}$, N) would, in particular, forbid the use of values 1 and 3 *i.e.* remove 1 from $dom(Y_1)$ and $dom(Y_3)$. Let us briefly explain why value 1 is forbidden. Two distinct values are used by X_2 and X_3 since $dom(X_2) \cap dom(X_3) = \emptyset$. Moreover value 1 does not belong to $dom(X_2)$ or $dom(X_3)$. Thus, using 1 would lead to at least 3 distinct values which is forbidden since $\overline{N} = 2$. The domains of N and the variables in $\mathcal{X}$ after GAC are thus as follows:

$$dom(X_1) = \{2\},\ \ dom(X_2) = \{2\},\ \ dom(X_3) = \{4, 5\},\ \ dom(N) = \{2\}\blacktriangle$$

2.3 The linear relaxation of AtMostWValue

This section extends the approach by Bessiere er al. [4], to filter AtMostNValue with its linear relaxation. Firstly, we simply extend it to AtMostNValue by introducing weights in the objective function of the linear program. Secondly, we introduce the $\mathcal{Y}$ variables in the CP model to allow back-propagation performed with reduced-cost based filtering. The approach proposed by [4] is detailed in Section 3.4.

Given the current state of the domains of variables in $\mathcal{X}$ and $\mathcal{Y}$, we would like to compute the linear relaxation of AtMostWValue. For this purpose, we introduce continuous variables $\{y_1, ..., y_m\}$ for each value. The linear program associated with AtMostWValue is referred to as LP_0:

$$\begin{array}{lll} (LP_0) & \text{Minimize :} & w = \sum_{1\le j\le m} w_j y_j \\ & \text{Subject to :} & \sum_{j\in dom(X_i)} y_j \ge 1 \qquad \forall i \in [1, n] \\ & & \underline{y_j} \in [\underline{Y_j}, \overline{Y_j}] \qquad \forall j \in [1, m] \end{array}$$

The optimum w^* of the above linear program is a valid lower bound for W. Moreover, this lower bound can be back-propagated to $\mathcal{Y}$ by applying reduced cost-based filtering [14]. Note that grounded variables can be removed from the formulation, leaving two types of constraints $\sum_{j\in dom(X_i)} y_j \ge 1$ and $y_j \le 1$. Moreover the constraints $y_j \le 1$ can be omitted as well from the model when assuming positive costs. A feasible solution with y_j strictly greater than 1 can be improved by setting y_j to 1 without questioning its feasibility, so that the $y_j \le 1$ constraints are satisfied in any optimal solution.

Let us denote by α_i^*, the optimal value of the dual variable related to the i-th constraint $\sum_{j \in dom(X_i)} y_j \geq 1$. The linear reduced cost of y_j, obtained at the optimum point, is equal to:

$$r_j^* = w_j - \sum_{\{i | j \in dom(X_i)\}} \alpha_i^* \tag{1}$$

Reduced-cost filtering, for all ungrounded variables of $\mathcal{Y}$, is performed once LP_0 has been solved, by using the optimal values y_j^* and r_j^*:

$$w^* + (1 - y_j^*) r_j^* > \overline{W} \implies Y_j \neq 1$$

$$w^* - y_j^* r_j^* > \overline{W} \implies Y_j \neq 0$$

The domain reductions on $\mathcal{Y}$ are propagated to $\mathcal{X}$ through the channeling constraints.

Example 2 (Continued) Since none of the Y_j variables is grounded, LP_0 is as follow:

$$\begin{array}{lll} \text{Minimize :} & w = y_1 + y_2 + y_3 + y_4 + y_5 & \\ \text{Subject to :} & y_1 + y_2 \geq 1 & \\ & y_2 + y_3 \geq 1 & \\ & y_4 + y_5 \geq 1 & \\ & y_j \geq 0 & \forall j \in [1, 5] \end{array}$$

The optimal value is 2 and $w^* = 2$. Moreover the tuple $(0, 1, 0, 0, 1)$ for $(y_1^*, \ldots, y_5^*)$ is an optimal solution of LP_0. It is feasible and the solution $(0, 1, 1)$ for $(\alpha_1^*, \alpha_2^*, \alpha_3^*)$ is a feasible dual solution with the same objective value: $\alpha_1^* + \alpha_2^* + \alpha_3^* = 2 = w^*$. The reduced costs $(r_1^*, \ldots, r_5^*)$ are therefore equal to $(1, 0, 0, 0, 0)$ since $r_1^* = 1 - \alpha_1^* = 1$, $r_2^* = 1 - \alpha_1^* - \alpha_2^* = 0$, $r_3^* = 1 - \alpha_2^* = 0$, ... As a result, value 1 is filtered from $dom(Y_1)$ because of the following rule:

$$w^* + (1 - y_1^*) r_1^* = 2 + 1 * 1 = 3 > \overline{N} = 2$$

Notice that value 1 is not filtered from $dom(Y_3)$ with this approach since $r_3^* = 0$. ▲

3 A new filtering algorithm for both AtMostNValue and AtMostWValue

Lagrangian relaxation (see *e.g.* [28]) is a very important technique in operation researchs that moves the "complicating constraints" into the objective function with a penalty term. It is an important practical tool for many structured problems and has been used to solve facility location problems for a long time [22]. We describe in this section how to solve LP_0 with Lagrangian relaxation.

For this purpose, we relax the constraints of LP_0 by adding them to the objective function. More precisely, for every $i \in [1, n]$, the constraint $\sum_{j \in dom(X_i)} y_j \geq 1$ is removed from the constraints set. Instead, we add the term $\lambda_i (1 - \sum_{j \in dom(X_i)} y_j)$ into the objective function to take into account the constraint violation or over-satisfaction. The Lagrangian subproblem LS_0 is defined as follow:

$$\begin{array}{lll} (LS_0) \quad \text{Minimize :} & w(\lambda) = \sum_{1 \leq j \leq m} w_j y_j + \sum_{1 \leq i \leq n} \lambda_i (1 - \sum_{j \in dom(X_i)} y_j) & \\ \text{Subject to :} & y_j \in \{0, 1\} & \forall j \in [1, m] \end{array}$$

The subproblem is therefore to find the optimal y for a given set of non-negative λ multipliers. For $w^*(\lambda)$ to be a lower bound of the original problem, the λ coefficients must

belong to $\mathbb{R}^+$. Hence when all constraints are satisfied, $w^*(\lambda)$ is always smaller or equal to w^* and provides a valid lower bound. Solving the Lagrangian relaxation LR_0 of AtMostWValueconsists in finding the Lagrangian multipliers vector λ for which the Lagrangian subproblem optimum is maximal. LR_0 is also called the Lagrangian dual.

$$\begin{array}{lll} (LR_0) & \text{Maximize :} & w^*(\lambda) \\ & \text{Subject to :} & \lambda_i \in \mathbb{R}^+ \qquad \forall i \in [1, n] \end{array}$$

The objective function of LR_0 is known to be a piecewise linear concave function and can thus be optimized with a subgradient procedure updating the λ values to move toward the optimum point.

Example 3 (Continued) The Lagrangian subproblem for our example is as follow:

$$\begin{array}{ll} \text{Minimize : } w(\lambda_1, \lambda_2, \lambda_3) = & y_1(1-\lambda_1) + y_2(1-\lambda_1-\lambda_2) \\ & + y_3(1-\lambda_2) + y_4(1-\lambda_3) + y_5(1-\lambda_3) \\ & +(\lambda_1+\lambda_2+\lambda_3) \\ \text{Subject to :} & y_j \in \{0,1\} \qquad \forall j \in [1,5] \end{array}$$

The optimal value $w^*(\lambda_1, \lambda_2, \lambda_3)$ is a lower bound of N as long as $(\lambda_1, \lambda_2, \lambda_3)$ is a vector of non-negative values. For instance, $w^*(0.3, 0.3, 0.3)$ is easily computed by setting all y_j^* to 0 (since the coefficients in front of each y_j are all non-negative). So $w^*(0.3, 0.3, 0.3) = 0.9$ proving that $\underline{N} = 1$. A vector $(2, 2, 2)$ would lead to $w^*(2, 2, 2) = 11 - 12 = -1$ by setting all y_j^* to 1 (each coefficient is now negative). $(0.6, 0.6, 0.8)$ is a better solution for LR_0 and proves that $\underline{N} = 2$ since $w^*(0.6, 0.6, 0.8) = 1.8$ (all y_j^* are set to 0 except $y_2^* = 1$). Finally $(1, 1, 1)$ is an optimal solution of LR_0 because $w^*(1, 1, 1) = 2$ by setting all y_j^* to 1. ▲

Since any non-negative λ vector provides a lower bound, filtering can be performed from the optimal solution of LS_0 for any value of λ. A propagator filtering according to the Lagrangian relaxation of AtMostWValue is thus made of three main components: An algorithm that solves the Lagrangian subproblem (Section 3.1), an algorithm that filters variable domains from a subproblem solution (Section 3.2) and an algorithm to manage the subgradient optimization (Section 3.3).

Algorithm 1 shows the general filtering procedure for AtMostWValue. Note that a filtering step on W and variables of $\mathcal{Y}$ is applied at every iteration.

Algorithm 1 Outline of the AtMostWValue LR-based propagator

```
for (i ∈ [1, n]) do
    λ_i ← 0 // Lagrangian multipliers initialization
k ← 0
while (not subgradient.isStopCriterionMet()) do
    SolveSubProblem()
    FilterFromSubProblem()
    subgradient.UpdateMultipliers(k)
    k ← k + 1
```

3.1 Solving the Lagrangian subproblem

The Lagrangian subproblem LS_0 consists of computing a subset of values that minimizes the objective function which depends on the weight function, the Lagrangian multipliers and the violation/satisfaction of the relaxed constraints. Since there are no hard constraints

linking the y variables, this problem can be solved by inspection with Algorithm 2. For each value y_j, the quantity q_j represents the variation of the objective function for setting y_j to 1:

$$q_j = w_j - \sum_{\{i \mid j \in dom(X_i)\}} \lambda_i \tag{2}$$

The time complexity of the algorithm is given by the sum of domain sizes in $\mathcal{X}$, which is bounded by $O(nm)$. Computing the q_j values is the time-consuming part of the algorithm. Since this is repeated at every iteration (see Algorithm 1), it is important to perform it in the current domain sizes $O(\sum_{i \in [1,n]} |dom(X_i)|)$.

Algorithm 2 Solving the Lagrangian subproblem of ATMOSTWVALUE

```
Function SolveSubProblem()
// Compute the objective variation resulting from setting each y_j to 1 (q_j).
q ← new double[m]
for (int j ∈ [1, m]) do
    q_j ← w_j
for (int i ∈ [1, n]) do
    for (int j ∈ dom(X_i)) do
        q_j ← q_j − λ_i
// compute the subproblem optimum
double LB ← Σ_{1≤i≤n} λ_i
for (int j ∈ [1, m]) do
    if (Y_j = 1 ∨ (¬Y_j = 1 ∧ q_j < 0)) then
        y_j ← 1
        LB ← LB + q_j
    else
        y_j ← 0
return LB
```

3.2 Filtering from the Lagrangian subproblem

Each Lagrangian subproblem optimum provides a valid lower bound for W. Furthermore, as for a linear relaxation, this lower bound can be back-propagated to $\mathcal{Y}$ by applying reduced cost-based filtering. Here, reduced costs are given by the q_j of Algorithm 2. The entire filtering procedure is given by Algorithm 3.

Example 4 (Continued) Consider the vector $(0.6, 0.6, 0.8)$ previously mentionned for $(\lambda_1, \lambda_2, \lambda_3)$. We had $w^*(0.6, 0.6, 0.8) = 1.8$ by setting all y_j^* to 0 except $y_2^* = 1$. The reduced cost of y_1 is equal to $q_1 = (1 - 0.6) = 0.4$. Value 1 is thus filtered from $dom(Y_1)$ because $1.8 + 0.4 = 2.2 > \overline{N} = 2$. Similarly $q_3 = (1 - 0.6) = 0.4$ and value 1 is filtered from $dom(Y_3)$. ▲

Notice that in this case, the Lagrangian reduced costs (2) and the linear reduced costs (1) are identical if the λ are equal to α^* *ie* if they reached the same optimal solution.

Example 5 (Continued) The vector $(0, 1, 1)$ for $(\alpha_1^*, \alpha_2^*, \alpha_3^*)$ given as an optimal solution of the dual of LP_0 is also an optimal solution of LR_0. We can check that $w^*(0, 1, 1) = 2$ by setting all y_j^* to 0. Although this solution is optimal for LR_0, the filtering performed from

this solution does not detect that value 3 is forbidden as opposed to the non optimal solution (0.6, 0.6, 0.8). ▲

Algorithm 3 Filtering AtMostWValue from its subproblem optimum

```
Function FilterFromSubProblem()
// Filter the objective lower bound
W ← max(W, LB) //LB is the current subproblem optimum (see Algorithm 2)
// Filter the 𝒴 variables
for (int j ∈ [1, m]) do
    // detect forbidden values j
    if (y_j = 0 ∧ q_j + LB > W̄) then
        Ȳ_j ← 0
    else
        // detect mandatory values j
        if (y_j = 1 ∧ LB − q_j > W̄) then
            Y_j ← 1
```

3.3 Solving the Lagrangian relaxation

A simple and popular approach for solving the Lagrangian dual LR_0 is to use a subgradient algorithm. The subgradient algorithm is made for solving a piecewise linear convex function. A subgradient is the direct generalization of the classic gradient used for continuously differentiable convex function [28]. The use of a subgradient algorithm to solve the Lagrangian dual stems from Held and Karp [18]. Propagators designed with Lagrangian relaxation [3, 13, 21] have so far used simple subgradient procedures very similar to the ones presented in the next section. However, [25] has a single multiplier and therefore implements a specific method to maximize one-dimensional concave functions based on the golden section.

3.3.1 Convergence process

Different subgradient algorithms exist [29] and may lead to different performances. Such a choice should not be seen as a defect (choice implies configuration) but as an improvement opportunity, to best fit the instance we are trying to solve. Therefore, from a software engineering point of view, the propagator architecture allows to use different (possibly *ad hoc*) subgradient algorithms.

In this paper, we consider three classic subgradient algorithms in the form of Algorithm 4. They are respectively called *Harmonic, Geometric* and *Newton*. These algorithms are defined by their μ^k function (see Table 1), which is often called the step-size rule. Note that, as long as the multipliers remain non-negative, any update algorithm (possibly random) is correct *i.e.* provides a valid lower bound when the subproblem is solved to optimality. However, convergence towards the best possible λ can be ensured if for instance $\mu_k \to 0$ and $\sum_k \mu_k \to +\infty$ as $k \to +\infty$ (see [28] for other convergence criteria).

Algorithm 4 subgradient algorithm

```
Function UpdateMultipliers(int k)
for (i ∈ [1, n]) do
    λ_i ← max(0, λ_i + μ^k(1 − Σ_{j∈dom(X_i)} y_j))
```

Table 1 Three different subgradient algorithms

	Harmonic	*Geometric*	*Newton*
$\mu^0 =$	1	10^3	5
$\mu^k =$	$\frac{\mu^0}{k}$	$\mu^0 \times 0.95^k$	$\frac{\mu^0}{2^{\lfloor k/10 \rfloor}} \times \frac{(\overline{W} - LB^k)}{\sum_{i \in [1,n]} \gamma_i (1 - \sum_{j \in dom(X_i)} y_j)^2}$ with $\gamma_i = \begin{cases} 0 \text{ if } \lambda_i = 0 \wedge (1 - \sum_{j \in dom(X_i)} y_j) \leq 0 \\ 1 \text{ otherwise} \end{cases}$

The *Harmonic* update satisfies the conditions as the harmonic series is divergent. In practice, we often encounter numerical issues and we also state a maximum limit of the number of iterations so that real convergence is never guaranteed. Let us first illustrate a simple geometric subgradient approach.

Example 6 (Continued) We consider a simple step-size rule $\mu^k = 1 \times (0.8)^k$. The following table shows the first 4 iterations on our running example:

k	μ^k	λ_1	λ_2	λ_3	$w^*(\lambda_1, \lambda_2, \lambda_3)$	Filtering detected
0	-	0	0	0	0	
1	0.8	0.8	0.8	0.8	1.8	$Y_2 \neq 0$
2	0.64	0.8	0.8	1.44	1.56	
3	0.512	0.8	0.8	0.928	1.9279	$Y_1 \neq 1, Y_3 \neq 1$
4	0.4096	0.8	0.8	1.3376	1.6624	
...	...	...	...	...	...	
		0.8	0.8	1.001	1.999	

Initially all λ_i are null and the optimal solution $(y_1^*, \ldots, y_5^*)$ of LS_0 is $(0, 0, 0, 0, 0)$. At the first iteration, $\mu^1 = 0.8$ so the λ vector is updated with algorithm 4 to $(0 + 0.8, 0 + 0.8, 0 + 0.8) = (0.8, 0.8, 0.8)$. The optimal solution of LS_0 becomes $(0, 1, 0, 0, 0)$ and the lower bound is now 1.8. Setting y_2 to 0 would raise the bound to $0.8 + 0.8 + 0.8 = 2.4$ ($q_2 = -0.6$) so that value 2 is detected as mandatory at iteration 1. In the solution $(0, 1, 0, 0, 0)$ of LS_0, constraints $y_1 + y_2 \geq 1$ and $y_2 + y_3 \geq 1$ are satisfied without slacks but $y_4 + y_5 \geq 1$ is violated by 1 unit so the subgradient is 1 for λ_3. λ is thus updated to $(0.8 + 0, 0.8 + 0, 0.8 + 1 \times 0.64) = (0.8, 0.8, 1.44)$ at iteration 2. Notice that the value of $w^*(\lambda_1, \lambda_2, \lambda_3)$ is typically non monotonous. ▲

Harmonic and *Geometric* do not depend on past iteration results as they are completely defined with a parameter μ^0 and the iteration number k. In contrast, *Newton* is more elaborated. This adaptation of the subgradient procedure of [18] has been successfully used in [3, 13]. It involves a coefficient, which is based on a parameter μ^0, that is updated (divided by $2^{\lfloor k/10 \rfloor}$) during the convergence process. As μ^k will rapidly converge towards 0, it triggers the stopping criterion sooner. Moreover, μ^k depends on the distance between the lower bound of the previous iteration and an estimation of the optimum from above. In other words, the step size is adjusted according to how close we are from an estimated optimum. Finally, it also depends on the constraint violations in the previous iteration. The γ function enables to count constraint violations, with strictly positive multipliers, and not constraint over-satisfactions. This gives the first iterations a big step size. Overall, *Newton* has initially large, but rapidly decreasing, step sizes, so that it can get close to the optimum quickly and stop soon.

3.3.2 Parameters and implementation

We stop the subgradient algorithm when the change of the λ values is not significant anymore (this threshold is set to 10^{-4} in our experiments) indicating that the subgradient procedure might be stalled. In addition, a maximum number of iterations (set to 10^3 in our experiments) is also used as a stopping criterion. Finally, to prevent numerical issues (e.g. overflows) from happening, we bound the maximum value of a multiplier with a constant.

It is often beneficial to start with relevant λ values rather than the zero values used by Algorithm 1for initialization. The best λ found are therefore stored at the end of the subgradient algorithm and used as a starting point for the next call of the algorithm. Moreover, these values are also restored upon backtracking.

Finally, the infeasible values detected are marked during the execution of the subgradient and the removal from the domains is only done after LR_0 has been solved.

3.4 Stronger filtering

An alternative to reduced costs filtering is to perform a simple form of shaving, or singleton consistency, by running the Lagrangian relaxation for every variable-value pair. Note that this technique brings a significant overhead and can rarely be used for domains of reasonable size. As a compromise, we follow the procedure proposed by Bessiere et al. in [4] for AtMostNValue with the linear relaxation. More precisely, they first solve the linear relaxation to obtain a lower bound for N. Then, for each value v such that $y_v^* = 0$ in the optimal solution of LP_0, they enforce $y_v = 1$ and solve $LP_0 \cup \{y_v = 1\}$ optimally again. Value v is removed from all domains of $\mathcal{X}$ if the optimal value of the $LP_0 \cup \{y_v = 1\}$ is greater than $\overline{N}$. It is restricted to the detection of forbidden values but not the mandatory ones since [4] do not represent the use of values explicitely with the $\mathcal{Y}$ variables.

We apply the same procedure (restricted to the filtering of forbidden values) as it can be done similarly for AtMostWValue and performed with either LP_0 or LR_0 to compute the lower bound. We call *singleton* this optional propagator configuration.

3.5 Comparing the LR_0 and LP_0 filtering power

It is known that the best bound w_{LR}^* achieved by a Lagrangian relaxation (LR) of an integer linear program (ILP) is at least as good as the optimal value w_{LP}^* of the linear relaxation (LP) of ILP so we have in this case $w_{LP}^* \leq w_{LR}^* \leq w_{ILP}^*$. It is also known, as a consequence of a theorem of Geoffrion [17], that $w_{LP}^* = w_{LR}^*$ if the Lagrangian subproblem is an integral linear program *i.e.* such that its linear relaxation is integral. LS_0 presented in Section 3 meets this criteria. y_j remains integral in any optimal solution even if we relax $y_j \in \{0, 1\}$ into $y_j \geq 0$ and $y_j \leq 1$. The lower bounds propagated for N and W (in AtMostNValue and AtMostWValue) by both approaches are therefore equal in theory (*i.e.* assuming LR_0 is solved optimally). Let us now consider the two options proposed for filtering $\mathcal{Y}$:

– *reduced-cost*: we already noticed that the linear reduced costs have the same definition than the Lagrangian ones. So if LR_0 and LP_0 have reached the same optimal solution *i.e.* the optimal λ are equal to α^* then the filtering performed at the optimal point is the same. We can however have many optimal solutions which makes the filtering incomparable. Furthermore, as explained in [24], by filtering at every iteration of the subgradient algorithm (not just the last one), the algorithm can detect infeasible values that would not be detected at one optimal λ. An illustration of this situation is given

in Example 5. So, assuming that LR_0 and LP_0 reach the same optimal multipliers, LR_0 would filter theoretically more infeasible values than LP_0, which only filters once from the optimal linear reduced costs.
– *singleton*: in this case, the filtering performed by LP_0 is in theory equal to the one of LR_0 since the bounds are the same. In practice, due to numerical and convergence issues of LR_0, the filtering of LP_0 may be stronger.

4 Empirical evaluation

Setup The experiments ran as a single thread on a Dual Quad Core Xeon CPU, 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64. The constraint programming solver used is Choco 3 [23] and the linear solver solver is version 12.5 of CPLEX.

Algorithms In the following we denote by:

– *CP*: the constraint programming model where the propagator of [4] for AtMostNValue is used.
– *CP+NVAL(s) / CP+WVAL(s) with s* ∈ {*Harmonic, Geometric, Newton*}: the constraint programming model using our Lagrangian propagator respectively for AtMostNValue and AtMostWValue with subgradient *s*. *Newton* is used by default if *s* is not given.
– *CP+NVAL(LP)*: the constraint programming model where propagation of AtMostNValue is done by LP_0.
– *ILP*: an integer linear programming model used as a baseline.
– *ILP+H*: the ILP model using the specific branching heuristic implemented for the CP approaches.

Metrics For a given algorithm, we denote by:

– *Cpu*: the cpu times in seconds,
– *Fails*: the number of failures for a CP approach.
– *Iters*: the total number of iterations (over the complete search-tree) performed by a subgradient algorithm.
– *#Feas* / *#Solv* / *#Best*: the number of times the algorithm respectively finds a feasible solution, achieves the proof of optimality, finds a best known solution.

Benchmark

– The dominating queens problem: a benchmark used by [4] that helps analyzing and understanding the behaviour of the various algorithms.
– The p-median problem: a benchmark related to facility location and taken from the discrete location problems benchmark library. Timelimit is set to 300s.
– A variant of the facility location problem: randomly generated instances. Timelimit is set to 60s.

Table 2 Comparison of various propagators for AtMostNValue on the dominating queens

Instance			CP+NVAL(*Harmonic*)			CP+NVAL(*Geometric*)			CP+NVAL(*Newton*)		
n	v	Feas	Cpu	Fails	Iters	Cpu	Fails	Iters	Cpu	Fails	Iters
6	3	Yes	0.2	7	25k	0.2	10	13k	**0.1**	9	**3k**
7	4	Yes	0.5	52	128k	0.4	81	80k	0.1	31	**9k**
8	5	Yes	1.0	86	210k	0.9	158	149k	**0.2**	79	**17k**
8	4	No	23.3	**1796**	2767k	14.4	2172	1352k	**5.1**	6260	**299k**
9	5	Yes	14.7	**862**	1426 k	8.9	910	604k	**3.4**	1593	**157k**

4.1 AtMostNValue: The dominating queens problem

This benchmark was originally used in [4] for comparing various propagation algorithms of the AtMostNValue constraint. The problem is to place v queens on a chessboard such that every square can be attacked by one of the queens (according to chess rules) and no two queens can attack each other. Hence there is nowhere to hide from the queens. The CP model boils down to a single AtMostNValue constraint. For a chessboard of size n, the model is made of variables $X_1, \ldots, X_{n^2}$ for each square and value j is in the domain of X_i if square j can be attacked from square i. The initial domain of every variable X_i is thus given by the set of squares $D_i \subset \{1, \ldots, n^2\}$ that a queen located at square i may attack. A queen is considered to be attacking the square on which it is located. The CP model is simply:

$$\begin{aligned} &\textsc{AtMostNValue}([X_1, \ldots X_{n^2}], v) \\ &X_i \in D_i \qquad\qquad\qquad\qquad \forall i = 1 \ldots n^2 \end{aligned}$$

Branching is performed **lexicographically** on the X variables and values are assigned from the lower to the upper bound. This heuristic is static (does not depend on the current state of the domains) to fairly compare the filtering of the various approaches. We compare the behaviour of propagators for AtMostNValue including the one of [4] (CP), three variants of the LR based propagation that only differ in the subgradient used (*Harmonic*, *Geometric*, *Newton*) and the LP based propagation presented in Section 2.3. The results are shown in Tables 2 and 3 for the instances used by [4].

1. Propagators based on LR outperform both LP based propagation and the state-of-the-art propagator presented in [4].
2. The propagation performed by LR can be stronger than the one done by LP (see the discussion in Section 3.5).

Table 3 Comparison of various propagators for ATMOSTNVALUE on the dominating queens

Instance			CP		CP+NVAL(LP)	
n	v	Feas	Cpu	Fails	Cpu	Fails
6	3	Yes	0.2	15	0.5	9
7	4	Yes	0.1	353	4.4	93
8	5	Yes	0.7	2275	18.8	259
8	4	No	155.6	1074789	222.8	2596
9	5	Yes	186.8	920666	153.7	884

Table 4 Propagators performing a form of singleton consistency on values 1 of the $\mathcal{Y}$ variables

Instance			CP+NVAL(*Harmonic*)			CP+NVAL(*Geometric*)			CP+NVAL(*Newton*)			CP+NVAL(LP)	
n	v	Feas	Cpu	Fails	Iters	Cpu	Fails	Iters	Cpu	Fails	Iters	Cpu	Fails
6	3	Yes	0.1	0	39k	0.1	0	18k	0.04	0	3k	0.1	0
7	4	Yes	3.7	4	759k	1.5	4	234k	0.36	4	54k	0.8	**3**
8	5	Yes	7.5	3	1095k	3.7	5	401k	1.05	8	104k	2.0	**2**
8	4	No	11.5	21	1474k	7.5	22	744k	13.53	276	1317k	**4.0**	21
9	5	Yes	14.0	5	1412k	6.9	5	525k	26.21	407	2017k	**4.8**	5

3. The *Harmonic* subgradient procedure is very slow and tends to perform more iterations than *Geometric* which in turns performs more iterations than *Newton*. Overall, *Newton* offers the best tradeoff between filtering and runtime here.

Table 4 shows the results obtained when all propagators perform a simple form of singleton consistency on the $\mathcal{Y}$ variables, described in Section 3.4. We assign each Y_j in turn to 1 and recompute the lower bound to check the consistency of value j.

1. The singleton procedure dramatically reduces the number of fails. This points out that, even though the relaxations are already elaborated, reduced costs are not that good and even more filtering may be achieved.
2. We can check that LP now captures all the filtering whereas LR can miss some of it due to occasional convergence issues. Moreover, LP is very fast as CPLEX is very incremental when changing the bounds of the variables one by one.
3. The best gradient is no longer *Newton* as it cannot afford to miss any filtering. The cost of propagation per node is now so high that one should make the most of it. Thus, slower gradients, converging better, are more effective in this case.

4.2 Facility location benchmark

We now consider a class of problem related to facility location. The general problem is to open at most p facilities for serving n clients at minimum cost. The set of facilities is chosen out of a discrete set of m facilities. A client i can only access to a subset $P(i)$ of the m facilities and the cost for serving client i by facility j is denoted c_{ij}. A cost must also

Table 5 Results of ILP approaches on P-Median instances

Instance			ILP			ILP+H		
n	m	Class	#Feas	#Solv	#Best	#Feas	#Solv	#Best
100	100	A	30	30	30	30	29	29
100	100	B	30	30	30	30	22	26
100	100	C	27	11	22	27	0	11

be paid for opening facility j and is denoted w_j. The ILP model used as a baseline is the following:

$$
\begin{array}{lll}
\text{(ILP)} \quad \text{Minimize :} & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{j=1}^{m} w_j y_j & \\
\text{Subject to :} & \sum_{j\in P(i)} x_{ij} = 1 & \forall i \in [1,n] \\
\sum_{j=1}^{m} y_j \leq p & & \\
& x_{ij} \leq y_j & \forall i \in [1,n], \forall j \in [1,m] \\
& x_{ij} \in \{0,1\} & \forall i \in [1,n], \forall j \in [1,m] \\
& y_j \in \{0,1\} & \forall j \in [1,m]
\end{array}
$$

Notice that if the y_j variables are known, then the remaining problem is simply to assign each client to its cheapest opened facility. From a linear programming point of view, this amounts to a simple flow problem and $x_{ij} \in \{0,1\}$ can be relaxed into $0 \leq x_{ij} \leq 1$. In fact, $x_{ij} \geq 0$ is enough. This greatly affect the performances of CPLEX by informing the branching to operate on the y variables solely.

We use two specific versions of this problem for benchmarking: the p-median problem (Section 4.2.1) where all fixed costs are null ($w_j = 0$ for all $j \in [1,m]$) and a restriced facility location problem (Section 4.2.2) where connection costs are null ($c_{ij} = 0$ for all $i \in [1,n]$ and $j \in [1,m]$).

4.2.1 AtMostNValue*: The p-median problem*

The benchmark is taken from the discrete location problems benchmark library[1] and features 90 instances split in three classes A, B, C of 30 instances each. The instances have a large duality gap for the linear relaxation making them hard to handle with LP based methods.

$$
\begin{array}{lll}
\text{Minimize } \sum_{i=1}^{n} C_i & & \\
(1) \quad \text{AtMostNValue}([X_1,\ldots X_n],[Y_1,\ldots Y_m],N) & & \\
(2) \quad \text{Element}(X_i,[c_{i1},\ldots,c_{im}],C_i) & & \forall i \in [1,n] \\
(3) \quad \sum_{i=1}^{m} Y_i = N & & \\
X_i \in P(i) & & \forall i \in [1,n] \\
C_i \geq 0 & & \forall i \in [1,n] \\
Y_j \in \{0,1\} & & \forall j \in [1,m] \\
N \in [1,p] & &
\end{array}
$$

Dominance As mentionned before, once the $\mathcal{Y}$ variables have been assigned (facilities are known), then the remaining problem is simply to assign each client X_i to its cheapest opened facility. In other words, dominance relations hold between the values of a given X_i. To model the fact that value a dominates b, we add redundant constraints. For all $i \in [1,n]$ and $\{a,b\} \in P(i)$ such that $c_{ia} < c_{ib} \vee (c_{ia} = c_{ib} \wedge a < b)$

$$(4) \qquad Y_a = 1 \implies X_i \neq b$$

This is implemented in a dedicated *dominance breaking* propagator which removes value b from X_i if $Y_a = 1, c_{ia} \leq c_{ib}$ and $\{a,b\}$ are still in the current domain of X_i ($a \in dom(X_i)$

[1] http://www.math.nsc.ru/AP/benchmarks/UFLP/Engl/uflp_dg_eng.html

Table 6 Results of CP approaches on P-Median instances

Instance			CP			CP+NVAL(*Newton*)				
n	m	Class	#Feas	#Solv	#Best	#Feas	#Solv	#Best	Avg_G	Max_G
100	100	A	1	0	1	30	19	26	1.2	17.1
100	100	B	10	0	0	30	7	15	2.2	9.1
100	100	C	0	0	0	23	0	4	9.5	25.6

and $b \in dom(X_i)$). This approach avoids the use of an *a priori* initial ordering of the values which can be counter-productive with the search heuristic.

Branching The branching is performed on the $\mathcal{Y}$ variables (X_i variables are grounded by propagation once $\mathcal{Y}$ are known due to constraint (4)). The variable Y_v is chosen first if v is a value that belongs to a variable X_i of minimum domain size (*first-fail* principle). To break the ties between values, we simply select the 'cheapest' in average, *ie* the value v minimizing $\frac{\sum_{i|v\in X_i} c_{iv}}{|\{i|v\in X_i\}|}$. Note that ILP+H is the ILP approach where the same heuristic is implemented using callbacks of CPLEX.

The timelimit is set to 300s. We also report in Tables 5 and 6, for the Lagrangian based approach, the average (Avg_G) and maximum (Max_G) gap (as a percentage) to the best known solution computed as $\frac{100*(UB-BestKnown)}{BestKnown}$ over all instances where a feasible solution has been found.

1. The CP model using the standard propagator for AtMostNValue is struggling to find feasible solutions. It is completely outperformed by the new propagator.
2. ILP is more efficient than ILP+H showing that the default branching of the MIP (based on pseudo-costs) is more powerful than the dedicated heuristic.
3. Our CP model does not propagate a strong lower bound on the objective (assignment costs are only handled using Element), does not start the search with any upper bound (as opposed the MIP approaches when they manage to identify a feasible solution in pre-solve) and is thus not expected to be competitive with ILP at this stage. Nonetheless, propagation on AtMostNValue is surprisingly effective and the CP approach generally provides solutions close to the best known.
4. On the hardest class C, ILP and ILP+H fails to find a feasible solution on 3 different instances each. The best value reported by each algorithm on the corresponding 6 instances is shown in Table 7.

Table 7 Highlight on the six instances where ILP or ILP+H fails to find a feasible solution.The value of the best solution found is shown for each approach

	ILP	ILP+H	CP(NVAL)
333PM_GapC	-	157	**147**
633PM_GapC	-	152	**145**
733PM_GapC	**137**	-	157
1833PM_GapC	**154**	-	175
2633PM_GapC	**132**	-	137
3033PM_GapC	-	**153**	167

Table 8 Results comparing four approaches for instances generated in the class (50, 50, 0.1)

	ILP	CP		CP+NVAL		CP+WVAL		CP+NVAL+WVAL	
	Cpu	Cpu	Fails	Cpu	Fails	Cpu	Fails	Cpu	Fails
Median	0.04	11.63	112026	0.22	481	10.12	9326	0.12	100
StDev	0.02	17.74	199847	1.32	2725	25.61	25765	0.16	80
#Best	20	20		20		16		20	
#Solv	20	18		20		15		20	

4.2.2 AtMostWValue*: A restricted facility location problem*

We consider here the facility location problem presented earlier where all c_{ij} are set to 0. Random instances are generated with parameters (n, m, d) where d is the probability to let client i access facility j, *ie* $j \in P(i)$. p is set to $\lfloor \frac{n}{4} \rfloor$ and the fixed costs w_i are taken uniformly in [1000, 1300]. We consider classes with $n, m \in \{50, 80, 100\}$ and $d \in \{0.1, 0.2\}$. 20 random instances are generated in each class. The constraint model is as follows:

$$
\begin{array}{lll}
\text{Minimize } W = \sum_{i=1}^{m} w_i Y_i & & \\
(1) \quad \text{AtMostNValue}([X_1, \ldots X_n], [Y_1, \ldots Y_m], N) & & \\
(2) \quad \sum_{i=1}^{m} Y_i = N & & \\
\quad X_i \in P(i) & & \forall i \in [1, n] \\
\quad Y_j \in \{0, 1\} & & \forall j \in [1, m] \\
\quad N \in [1, p] & &
\end{array}
$$

The AtMostWValue constraint can be added as a redundant constraint to strengthen the lower bound of the objective function:

$$(3)\,\text{AtMostWValue}([X_1, \ldots X_n], [Y_1, \ldots Y_m], [w_1, \ldots, w_m], W)$$

Note also that constraint (3) can entirely replace the objective function and (2) is a useful redundant constraint (in particular for the CP approach).

4.3 Comparing CP models

We compare four approaches including an ILP formulation, the CP model given above, and the three possible extended models using our Lagrangian based propagator. The approach CP + NVAL + WVAL is thus stating both global constraints.

Branching The branching is performed first on N (from its lower to its upper bound) then on the $\mathcal{Y}$ variables, lexicographically (the heuristic is therefore static).

Table 9 reports the cpu times and number of failures (median and deviation) for 20 random instances generated in the class (50, 50, 0.1).

Significant improvements are observed for the model using both global constraints: the median numbers of failures are divided by more than 1000 when compared to the standard propagation of AtMostNValue.

4.4 Comparing with ILP

To ensure further scalability, we improve the model with ideas similar to the ones already introduced for the p-median.

Table 9 Results for larger instances of the restricted facility location problem

Instances		ILP	ILP+H	CP		CP+NVAL+WVAL	
		Cpu	Cpu	Cpu	Fails	Cpu	Fails
(80,80,0.1)	Median	0.43	1.55	60.00	117632	6.99	754
	Stdev	0.34	2.11	0.00	16734	5.87	728
	#Best	20	20	19		20	
	#Solv	20	20	0		20	
(80,80,0.2)	Median	3.22	12.47	60.00	85196	32.92	3628
	Stdev	2.26	12.03	0.00	7359	16.13	1970
	#Best	20	20	20		20	
	#Solv	20	20	0		16	
(100,100,0.1)	Median	3.60	29.52	60.00	49370	60.00	4264
	Stdev	3.10	22.12	0.00	2813	11.68	969
	#Best	20	17	11		20	
	#Solv	20	16	0		7	
(100,100,0.2)	Med	46.75	60.00	60.00	38808	60.00	4671
	Stdev	23.16	20.75	0.00	1315	8.72	756
	#Best	15	11	5		20	
	#Solv	13	4	0		1	
(100,100,0.5)	Med	55.65	60.00	60.00	46518	60.00	6337
	Stdev	21.78	19.11	8.24	7938	11.43	1418
	#Best	12	17	20		20	
	#Solv	10	7	2		4	

Symmetry: In this particular case, all values of the $\mathcal{X}$ variables are symmetrical (all c_{ij} are null). For every $i \in [1, n]$ and for all $\{(a, b) \mid a < b\} \in P(i)$, we can add:

$$(3) \qquad Y_a = 1 \implies X_i \neq b$$

As explained in Section 4.2.1, it is implemented as a specific symmetry breaking constraint.

Branching The branching on $\mathcal{Y}$ variables is improved. Variable Y_v is chosen first if v is a value that belongs to a variable X_i of minimum domain size (*first-fail* principle) and ties are broken by preferring the value v with smallest average cost $\frac{w_v}{|\{i \mid v \in X_i\}|}$.

The timelimit is set to 60s. Table 9 shows the results.

As the size grows, the strenghtened CP model is able to find better solutions than the ILP or ILP+H but is less effective to perform the proof of optimality.

5 Conclusion and future work

This paper has introduced a simple and powerful filtering algorithm for both AtMostNValue and AtMostWValue. The algorithm relies on a network of channeling constraints and a Lagrangian relaxation whose subproblem can be solved very efficiently by inspection. We have illustrated its practical relevance on several benchmarks. The proposed algorithms significantly increase the filtering power of both constraints, while remaining profitable in terms of propagation runtime.

This work can be extended in several ways. From a technical point of view, the management of the subgradient has potential for progress. We plan to investigate different techniques, such as bundle methods [29] as well as the use of simple machine learning components [19] to select the parameters. We also believe that it is worth to generalize this work for assignment costs *i.e.* costs, related to the use of a value j by a given variable X_i. Many problems involving costs related to the $\mathcal{Y}$ variables also involve costs related to the $\mathcal{X}$ variables and the connections with the work of [9, 20] should be investigated.

References

1. Beldiceanu, N., & Carlsson, M. (2001). Pruning for the minimum constraint family and for the number of distinct values constraint family. In Walsh, T. (Ed.) *Principles and Practice of Constraint Programming – CP 2001,* volume 2239 of *Lecture Notes in Computer Science,* (pp. 211–224). Berlin Heidelberg: Springer.
2. Beldiceanu, N., Carlsson, M., & Thiel, S. (2002). Cost-filtering algorithms for the two sides of the sum of weights of distinct values constraint. *Technical report – T2002-14*: Swedish Institute of Computer Science.
3. Benchimol, P., Van Hoeve, W.J., Régin, J.-C., Rousseau, L.-M., & Rueher, M. (2012). Improved filtering for weighted circuit constraints. *Constraints*, *17*(3), 205–233.
4. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., & Walsh, T. (2006). Filtering algorithms for the nvalue constraint. *Constraints*, *11*(4), 271–293.
5. Bessiere, C., Katsirelos, G., Narodytska, N., Quimper, C.-G., & Walsh, T. (2010). Decomposition of the nvalue constraint. In Cohen, D. (Ed.) *Principles and Practice of Constraint Programming – CP 2010,* volume 6308 of *Lecture Notes in Computer Science,* (pp. 114–128). Berlin Heidelberg: Springer.
6. Cambazard, H. Np-hard contraints involving costs: examples of applications and filtering. In *Dixièmes Journées Francophones de Programmation par Contraintes – JFPC.* 2014. Exposé invité.
7. Cambazard, H., O'Mahony, E., & O'Sullivan, B. (2012). A shortest path-based approach to the multileaf collimator sequencing problem. *Discrete Applied Mathematics*, *160*(1–2), 81–99.
8. Cambazard, H., & Penz, B. (2012). A constraint programming approach for the traveling purchaser problem. In Milano, M. (Ed.) *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings,* volume 7514 of *Lecture Notes in Computer Science,* (pp. 735–749): Springer.
9. Cooper, M.C., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., & Werner, T. (2010). Soft arc consistency revisited. *Artificial Intelligence*, *174*(7–8), 449–478.
10. Van den Bergh, J., Belieën, J., De Bruecker, P., Demeulemeester, E., & De Boeck, L. (2013). Personnel scheduling: a literature review. *European Journal of Operational Research*, *226*(3), 367–385.
11. Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location. *Operations Research*, *26*(6), 992–1009.
12. Fages, J.-G., & Lapègue, T. (2014). Filtering atmostnvalue with difference constraints: application to the shift minimisation personnel task scheduling problem. *Artificial Intelligence*, *212*(0), 116–133.
13. Fages, J.-G., Lorca, X., & Rousseau, L.-M. (2014). The salesman and the tree: the importance of search in CP. *Constraints*, 1–18.
14. Focacci, F., Lodi, A., & Milano, M. (1999). Cost-based domain filtering. In Jaffar, J. (Ed.) *Principles and Practice of Constraint Programming – CP'99,* volume 1713 of *Lecture Notes in Computer Science,* (pp. 189–203). Berlin Heidelberg: Springer.

15. Fontaine, D., Michel, L.D., & Van Hentenryck, P. Constraint-based lagrangian relaxation. In O'Sullivan, B. (Ed.) *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings,* volume 8656 of *Lecture Notes in Computer Science,* (pp. 324–339) (p. 2014). Berlin: Springer.
16. Gaspers, S., & Szeider, S. (2011). Kernels for global constraints, *CoRR*. arXiv:1104.2541.
17. Geoffrion, A.M. (1974). Lagrangean relaxation for integer programming. In Balinski, M.L. (Ed.) *Approaches to integer programming,* volume 2 of *mathematical programming studies,* (pp. 82–114). Berlin Heidelberg: Springer.
18. Held, M., & Karp, R.M. (1971). The traveling-salesman problem and minimum spanning trees: part II. *Mathematical Programming*, *1*(1), 6–25.
19. Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC - instance-specific algorithm configuration. In *ECAI, volume 215 of Frontiers in Artificial Intelligence and Applications,* (pp. 751–756): IOS Press.
20. Lee, J.H.M., & Leung, K.L. (2012). Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. *Journal of Artificial Intelligence Research*, *43*(1), 257–292.
21. Menana, J., & Demassey, S. (2009). Sequencing and counting with the multicost-regular constraint. In Van Hoeve, W.J., & Hooker, J.N. (Eds.) *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings,* volume 5547 of *Lecture Notes in Computer Science,* (pp. 178–192): Springer.
22. Narula, S.C., Ogbu, U.I., & Samuelsson, H.M. (1977). An algorithm for the p-median problem. *Operations Research*, *25*(4), 709–713.
23. Prud'homme, C., Fages, J.-G., & Lorca, X. (2014). *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.
24. Sellmann, M. (2004). Theoretical foundations of cp-based lagrangian relaxation. In Wallace, M. (Ed.) *Principles and Practice of Constraint Programming – CP 2004,* volume 3258 of *Lecture Notes in Computer Science,* (pp. 634–647). Berlin Heidelberg: Springer.
25. Sellmann, M., & Fahle, T. (2003). Constraint programming based lagrangian relaxation for the automatic recording problem. *Annals OR*, *118*(1–4), 17–33.
26. Slusky, M.R., & Van Hoeve, W.J. (2013). A lagrangian relaxation for golomb rulers. In Gomes, C.P., & Sellmann, M. (Eds.) *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings,* volume 7874 of *Lecture Notes in Computer Science,* (pp. 251–267): Springer.
27. Wah, B.W., & Wu, Z. (1999). The theory of discrete lagrange multipliers for nonlinear discrete optimization. In Jaffar, J. (Ed.) *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings,* volume 1713 of *Lecture Notes in Computer Science,* (pp. 28–42): Springer.
28. Wolsey, L.A. (1998). *Integer programming.* Wiley-Interscience series in discrete mathmatics and optimization. New York: Wiley.
29. Zhao, X., & Luh, P.B. (2002). New bundle methods for solving lagrangian relaxation dual problems. *Journal of Optimization Theory and Applications*, *113*(2), 373–397.