Local Alterations of the Lagrange Multipliers for Enhancing the Filtering of the AtMostNValue Constraint

Frédéric Berthiaume and Claude-Guy Quimper[0000−0002−5899−0217]

Université Laval, Québec, Canada
frederic.berthiaume.1@ulaval.ca, claude-guy.quimper@ift.ulaval.ca

Abstract. The reduced cost filtering is a technique that consists in filtering a constraint using the reduced cost of a linear program that encodes this constraint. Sellmann [16] shows that while doing a Lagrangian relaxation of a constraint, suboptimal Lagrange multipliers can provide more filtering than optimal ones. Boudreault and Quimper [5] make an algorithm that locally altered the Lagrange multipliers for the WeightedCircuit constraint to enhance filtering and achieve a speedup of 30%. We seek to design an algorithm like Boudreault and Quimper, but for the AtMostNValue constraint. Based on the work done by Cambazard and Fages [7] on this constraint, we use a subgradient algorithm which takes into consideration the reduced cost to boost the Lagrange multipliers in the optimal filtering direction. We test our methods on the dominating queens and the p-median problem. On the first, we record a speedup of 71% on average. On the second, there are three classes of instances. On the first two, we have an average speedup of 33% and 8%. On the hardest class, we find up to 13 better solutions than the previous algorithm on the 30 instances in the class.

Keywords: Global constraint · Lagrangian relaxation · N values.

1 Introduction

Global constraints contributed to the success of constraint programming. These constraints involve a non-fixed number of variables [2]. A filtering algorithm prunes the values from the variable domains. To aid the filtering process, Focacci et al. [9] introduced a technique called *reduced cost filtering*. It studies the variation of the objective function after a consistent variable change of values. There is also Lagrangian relaxation, which is a technique to rewrite the problem without some constraints while penalizing when they are violated. During the process of Lagrangian relaxation, Sellmann [16] showed that the suboptimal Lagrange multipliers can provide better filtering than the optimal ones. Boudreault and Quimper [5] proposed an algorithm for the WeightedCircuit constraint where they locally alter the Lagrange multipliers to enhance the filtering.

Cambazard and Fages [7] treated the Lagrangian relaxation of the sub constraint AtMostNValue. We propose to augment their algorithm based on a

Lagrangian relaxation by locally altering the Lagrangian multipliers to enhance the filtering. We use the dominating queens problem and the facility location problem in order to test our algorithm.

Section 2 introduces the notation and defines the NVALUE constraint leading to the starting point of this paper, the filtering algorithm of the ATMOSTNVALUE constraint based on a Lagrangian relaxation. Section 2.5 proposes an addition to the previous algorithm. Section 3 presents this addition along with the theoretical justification. Section 4 presents the experiments.

2 Background

2.1 Notation

We note column and row vectors with the symbols $\boldsymbol{v}$ and $\boldsymbol{v}^\top$. The vectors $\mathbf{0}$ and $\mathbf{1}$ are the vectors with only zeros and ones. The columns and rows of a matrix A are noted $\boldsymbol{a_{cj}}$ and $\boldsymbol{a_{ri}}^\top$ respectively. So, $\boldsymbol{a_{ri}}$ is the column vector that, when transposed, gives the i-th row of A. The components of A are written a_{ij}.

We note the gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ with $\boldsymbol{\nabla} f$. For a *piece-wise* continuous linear function $f : \mathbb{R}^n \to \mathbb{R}$, we note one of its *subgradient* with $\tilde{\boldsymbol{\nabla}} f$. Briefly, a *subdifferential* is the generalization of a *derivative*. For example, the function $f(x) = |x|$ is differentiable everywhere except at $x = 0$. Although, the function $f(x) = |x|$ does not admit a derivative at $x = 0$, it admits a *subdifferential* at $x = 0$ labeled $\partial f(0)$. $\partial f(x_0)$ is the set of all v that satisfies $f(x) - f(x_0) \geq v(x - x_0)$; for $f(x) = |x|$, $\partial f(0) = [-1, 1]$. One may view an ordinary derivative as a subdifferential with a unique value. The subdifferential of $f(\boldsymbol{x})$ at $\boldsymbol{x_0}$, $\partial f(\boldsymbol{x_0})$, is the set of all vectors $\boldsymbol{v}$, the subgradients, that satisfies $f(\boldsymbol{x}) - f(\boldsymbol{x_0}) \geq \boldsymbol{v}^\top(\boldsymbol{x} - \boldsymbol{x_0})$. The symbol $\tilde{\boldsymbol{\nabla}} f(\boldsymbol{x})$ represents an element of this set. If there is ambiguity on which variables the gradient (or subgradient) is taken, we specify them with $\boldsymbol{\nabla}_x$ (or $\tilde{\boldsymbol{\nabla}}_x$).

We use two types of variables that are fundamentally different while related. There are constraint satisfaction problem (*CSP*) variables, written in uppercase (X), and each has a set of *values*, called a domain ($\mathrm{dom}(X)$). There are linear program (LP) variables that we write as vectors ($\boldsymbol{x}$).

2.2 The NVALUE constraint

Definition 1. *The* NVALUE *constraint bounds the number of distinct values taken by a set of variables. It is written* $\text{NVALUE}([X_1, \ldots, X_n], N)$*, where* N *is the cardinality variable. The constraint is satisfied when* $N = |\{X_1, \ldots, X_n\}|$.

Enforcing domain consistency on the NVALUE constraint is *NP-hard* [4]. The constraint can be decomposed into two other constraints: ATLEASTNVALUE and ATMOSTNVALUE. Enforcing domain consistency on the former can be done in polynomial time [4, 14]. Enforcing domain consistency on the ATMOSTNVALUE is *NP-hard* [4]. We will only consider this constraint. The ATMOSTNVALUE constraint is satisfied when $|\{X_1, \ldots, X_n\}| \leq N$.

2.3 The LP of the AtMostNValue constraint

Bessiere et al. [4] present three approaches to propagate the NValue constraint. The first one, an algorithm proposed by Beldiceanu [1], enforces bounds consistency and is based on interval graphs. The second approach is based on the independence number of a graph. The third approach is a LP formulation of the AtMostNValue constraint. Bessiere et al. show that the LP formulation offers a better estimation of the lower bound of N.

In the CSP, each value $j \in \bigcup_{i=1}^{n} \mathrm{dom}(X_i) = \{1, ..., m\}$ is paired with a Boolean variable Y_j. The CSP variable Y_j encodes whether the value j belongs to $\{X_1, ..., X_n\}$. In the LP that encodes the AtMostNValue constraint, each of the CSP variable Y_j has an analog LP variable y_j. If Y_j is instantiated, then y_j takes the same value as Y_j. Otherwise, the LP variables $\boldsymbol{y}$ are free and are assigned while solving the LP of the AtMostNValue constraint:

$$\min_{\boldsymbol{y}} \quad h(\boldsymbol{y}) = \mathbf{1}^\top \boldsymbol{y}, \quad \text{s.t.} \quad \mathrm{A}\boldsymbol{y} \geq \mathbf{1}, \quad \boldsymbol{y} \leq \mathbf{1}, \quad \boldsymbol{y} \geq \mathbf{0}, \tag{1}$$

where $a_{ij} = 1$ if $j \in \mathrm{dom}(X_i)$ and $a_{ij} = 0$ otherwise. The i-th row $\boldsymbol{a_{ri}}$ encodes in a vector the set $\mathrm{dom}(X_i)$. These constraints ensure that for every variable X_i, there is at least one value $j \in \mathrm{dom}(X_i)$ for which $y_j = 1$.

2.4 The propagator of the AtMostNValue constraint

Cambazard and Fages [7] designed a propagator for the AtMostNValue constraint. The algorithm takes as input the domains of the following CSP variables: the integer variables $\{X_1, ..., X_n\}$, the Boolean variables $\{Y_1, ..., Y_m\}$ and the cardinality variable N. This algorithm is based on a **Lagrangian relaxation** [8, 11, 13] of the LP (1) and estimates a lower bound on N.

A Lagrangian relaxation is a twofold process. First, there is a transformation from an initial constrained optimization problem $\mathcal{P}$ into another $\mathcal{P}'$. The *relaxed* problem $\mathcal{P}'$ is defined on a subset of constraints $C' \subset C$. Each constraint $c_i \in C \setminus C'$ is paired with a *Lagrange multiplier* in the new objective function, a *Lagrangian* function, $f'(\boldsymbol{x}, \boldsymbol{\lambda})$. The Lagrangian function depends on the original objective function $f(\boldsymbol{x})$ and each relaxed constraint weighted by its Lagrange multipliers. The transformation of (1), done by Cambazard and Fages, is

$$\min_{y} \quad h'(\boldsymbol{y}, \boldsymbol{\lambda}) = \mathbf{1}^\top \boldsymbol{y} + \boldsymbol{\lambda}^\top (\mathbf{1} - \mathrm{A}\boldsymbol{y}), \quad \text{s.t.} \quad \boldsymbol{y} \leq \mathbf{1}, \quad \boldsymbol{y} \geq \mathbf{0}, \quad \boldsymbol{\lambda} \geq \mathbf{0}. \tag{2}$$

The constraints that are relaxed in (2) are those that ensure that at least one value from every domain $\mathrm{dom}(X_i)$ is selected.

The Lagrange multiplier λ_i associated to the i-th constraint must be positive to penalize the function when the constraint $\boldsymbol{a_{ri}}^\top \boldsymbol{y} \geq 1$ is violated. Constraints violation result in an increase of the Lagrangian function, which is opposite to the original objective of minimization.

The second phase of a Lagrangian relaxation is the optimization of the Lagrangian function, where one tries to solve $\max_{\boldsymbol{\lambda}}(\min_{\boldsymbol{y}} h'(\boldsymbol{y}, \boldsymbol{\lambda}))$. This translates

into trying to find the greatest lower bound of $h'(\boldsymbol{y}, \boldsymbol{\lambda})$. Before addressing this phase, we must clarify what happens with $\min_{\boldsymbol{y}} h'(\boldsymbol{y}, \boldsymbol{\lambda})$.

The LP (2) is *solved* when the LP variables y_j are fixed. We first need the coefficients q_j coupled with the LP variables y_j. After rewriting the function $h'(\boldsymbol{y}, \boldsymbol{\lambda})$ as $h'(\boldsymbol{y}, \boldsymbol{\lambda}) = \left(\mathbf{1} - \mathrm{A}^\top \boldsymbol{\lambda}\right)^\top \boldsymbol{y} + \boldsymbol{\lambda}^\top \mathbf{1}$, the coefficient vector is :

$$\boldsymbol{q}(\boldsymbol{\lambda}) = \mathbf{1} - \mathrm{A}^\top \boldsymbol{\lambda}. \tag{3}$$

The coefficient $q_j(\boldsymbol{\lambda})$ is the reduced cost of y_j, the next section explains the meaning of the reduced cost. The LP variables y_j can be fixed either from their CSP variables counterpart Y_j or the value of their coefficient in $h'(\boldsymbol{y}, \boldsymbol{\lambda})$:

$$\text{if} \quad |\operatorname{dom}(Y_j)| = 1 \quad \text{then} \quad y_j = Y_j \quad \text{else,} \quad y_j(\boldsymbol{\lambda}) = \begin{cases} 1 \text{ if } q_j(\boldsymbol{\lambda}) < 0 \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

The LP variables $\boldsymbol{y}$ associated with unfixed CSP variables Y_j can be viewed as functions of $\boldsymbol{\lambda}$, because they depend on the sign of $q_j(\boldsymbol{\lambda})$. Afterwards the problem $\min_{\boldsymbol{y}} h'(\boldsymbol{y}, \boldsymbol{\lambda})$ is considered solved for the current $\boldsymbol{\lambda}$. To enhance that the LP variables $\boldsymbol{y}$ are fixed, we write the current solution as $h''(\boldsymbol{\lambda}) = \min_{\boldsymbol{y}} h'(\boldsymbol{y}, \boldsymbol{\lambda})$.

The initial Lagrange multipliers $\boldsymbol{\lambda}$ generally need improvement to give the greater relaxed lower bound on N and to respect the relaxed constraints. The $\boldsymbol{\lambda}$ must be updated with care because $\boldsymbol{\nabla} h''(\boldsymbol{\lambda})$ is not continuous.

Definition 2. *$L_c(f) = \{\boldsymbol{x} \mid f(\boldsymbol{x}) = c\}$ is the level set c of function f.*

Lemma 1. *$h''(\boldsymbol{\lambda})$ is continuous $\forall \boldsymbol{\lambda}$, but not $\boldsymbol{\nabla} h''(\boldsymbol{\lambda})$.*

Proof. $h''(\boldsymbol{\lambda}) = \boldsymbol{q}(\boldsymbol{\lambda})^\top \boldsymbol{y} + \mathbf{1}^\top \boldsymbol{\lambda}$. The LP variables $\boldsymbol{y}$ depend on $\boldsymbol{\lambda}$ (4) and are discontinuous functions of $\boldsymbol{\lambda}$. The term $q_j(\boldsymbol{\lambda}) y_j$ is similar to a *RELU function* ($\max\{0, x\}$), which has no derivative at $x = 0$ (from the left the derivative is 0 and 1 from the right). The term $q_j(\boldsymbol{\lambda}) y_j$ has no derivative on $L_0(q_j)$ (Definition 2). So each term in $\boldsymbol{q}(\boldsymbol{\lambda})^\top \boldsymbol{y}$ has no derivative on its $L_0(q_j)$. Thus, $h''(\boldsymbol{\lambda})$ is not derivable, but continuous, on $\bigcup_{j=1}^{m} L_0(q_j)$. □

Hence the Lagrange multipliers are updated according to the *subgradient* of $h''(\boldsymbol{\lambda})$ instead of the traditional gradient. For a given $\boldsymbol{\lambda}$, $\tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda}) = \mathbf{1} - \mathrm{A}\boldsymbol{y}$ is a subgradient of $h''(\boldsymbol{\lambda})$ [10]. Then step-sizes α_k are carefully chosen to update the $\boldsymbol{\lambda}$ with the following equation

$$\lambda_i^{(k+1)} = \max(0, \lambda_i^{(k)} + \alpha_k [\tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda})]_i), \tag{5}$$

Studies of step-sizes rules are listed here [3, 6, 10, 15]. The *max* operator in (5) ensures that $\boldsymbol{\lambda} \geq \mathbf{0}$. Here $[\tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda})]_i$, which is equal to $1 - \boldsymbol{a_{ri}}^\top \boldsymbol{y}$, represents the i-th constraint satisfaction. If the i-th component is 1, it means that the CSP variable X_i has no value in its domain. This procedure is repeated until the optimal solution is found or some other convergence criterion is satisfied.

Reduced cost filtering (introduced by Focacci et al. [9]) is a technique to filter values from CSP variable domains based on their contribution in the

Harmonic	Geometric	Newton
$\alpha_k = 1/k$	$\alpha_k = 10^3(0.95)^k$	$\alpha_k = \frac{5}{2^{\lfloor k/10 \rfloor}} \frac{\max(\text{dom}(N)) - h''(\boldsymbol{\lambda})_k}{\sum_{i=1}^{n} \gamma_i (\tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda})_i)^2}$ $\gamma_i = \begin{cases} 1 & \text{if } \lambda_i > 0 \vee \tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda})_i = 1 \\ 0 & \text{otherwise} \end{cases}$

Table 1: The three step-size rules and their parameters used in [7]. The denominator in the *Newton* column is an analog of the squared norm of $\tilde{\boldsymbol{\nabla}}_{\lambda} h(\boldsymbol{y}, \boldsymbol{\lambda})$.

current state of the solution. In the LP (2), the contribution of the LP variable y_j to the solution is its coefficient $q_j(\boldsymbol{\lambda})$. When

$$h''(\boldsymbol{\lambda}) + |q_j(\boldsymbol{\lambda})| = h''(\boldsymbol{\lambda}) + |1 - \boldsymbol{a_{cj}}^{\top} \boldsymbol{\lambda}| > \max(\text{dom}(N)) \tag{6}$$

the complementary value of the LP variable y_j, $1 - y_j$, can be filtered from the domain of the CSP variable Y_j. This is because the LHS of (6) is the relaxed lower bound on N, if the LP variable y_j was fixed to the value $1 - y_j$. If, (6) is satisfied, it means that $1 - y_j$ leads to $\min(\text{dom}(N)) > \max(\text{dom}(N))$, which is impossible. The $|q_j(\boldsymbol{\lambda})|$ is the reduced cost of the LP variables y_j.

Algorithm 1, designed by Cambazard and Fages [7], filters the constraint AtMostNValue. To perform the transformation from (1) to (2), the LP variables $\boldsymbol{y}$ (not the CSP variables Y_j) and $\boldsymbol{\lambda}$ are initialized at line 1 of Algorithm 1. On lines 2-14, the problem $\min_{\boldsymbol{y}} h'(\boldsymbol{y}, \boldsymbol{\lambda})$ is solved by fixing the LP variables y_j according to (4). The relaxed lower bound on N, $h''(\boldsymbol{\lambda})$, is computed at line 8. On lines 9-10 the reduced cost filtering phase is performed for each non-instantiated LP variable y_j. Line 10 flags the value for future filtering. As Sellman [16] explains in Section *A disturbing example*, filtering values during a Lagrangian relaxation lead to inconsistencies because the current problem, $\min_{\boldsymbol{y}} h'(\boldsymbol{y}, \boldsymbol{\lambda})$, might change too much between consecutive iterations $k \rightarrow k + 1$. Thus, they are filtered at the end of the Algorithm (line 16). Finally, the Lagrangian optimization is ended if the maximum number of iterations is reached or if the change of $\boldsymbol{\lambda}$ between two consecutive iterations is too small. This last condition represents a situation where the algorithm converges to an optimal $\boldsymbol{\lambda}$, for the current state of the CSP variables at the start of Algorithm 1. Cambazard and Fages compared three types of step-size (Table 1) to update the Lagrange multipliers in (5).

2.5 Which Lagrange multipliers are the most effective?

Sellmann [16] presents a dichotomy between *getting the best lower bound on LP like* (2) and *filtering many values from the variables' domain*. The main result is the *Filtering Penalties theorem* which justifies checking between iteration steps of (5) for inconsistent values (Algorithm 1 lines 9-10).

From this result, Boudreault and Quimper [5] presented a new algorithm for the WeightedCircuit constraint, where they model the constraint as a LP, relax it using a Lagrangian relaxation, and locally modify the Lagrange multipliers to augment the filtering. Their algorithm filters more values and is faster than the state-of-the-art algorithm for this constraint.

Algorithm 1: AtMostNValue LR-based propagator ($\{X_1, \ldots, X_n\}$, $\{Y_1, \ldots, Y_m\}$, N, IterMax)

$\boldsymbol{y} \leftarrow \underbrace{[0, \ldots, 0]}_{m}$, $\boldsymbol{\lambda} \leftarrow \underbrace{[0.0, \ldots, 0.0]}_{n}$, $\boldsymbol{q} \leftarrow \underbrace{[0.0, \ldots, 0.0]}_{m}$, $k \leftarrow 0$
repeat
  $\boldsymbol{q} \leftarrow \mathbf{1} - \mathrm{A}^\top \boldsymbol{\lambda}$
  for $j \leftarrow 1$ **to** m **do**
    if $(\min(\mathrm{dom}(Y_j)) = 1)$ **or** $((\max(\mathrm{dom}(Y_j)) = 1)$ **and** $(q_j < 0))$ **then**
      $y_j \leftarrow 1$
    else $y_j \leftarrow 0$
  $h''(\boldsymbol{\lambda}) \leftarrow \boldsymbol{q}^\top \boldsymbol{y} + \mathbf{1}^\top \boldsymbol{\lambda}$
  for $j \leftarrow 1$ **to** m **do**
    if $(h''(\boldsymbol{\lambda}) + |q_j| > \max(\mathrm{dom}(N)))$ **then** Flag value $1 - y_j$ in $\mathrm{dom}(Y_j)$
  $\tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda}) \leftarrow \mathbf{1} - \mathrm{A}\boldsymbol{y}$
  Update α_k // With Table 1
  $\boldsymbol{\lambda'} \leftarrow \boldsymbol{\lambda}$, $\boldsymbol{\lambda} \leftarrow \max(\mathbf{0}, \boldsymbol{\lambda} + \alpha_k \tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda}))$, $k \leftarrow k + 1$
until $k = \mathrm{IterMax}$ **or** $\max_{1 \leq i \leq n} |\lambda_i - \lambda'_i| \leq 0.0001$
// The local alterations algorithm (Algorithm 2) is called here
Filter out all the flag values

3 Improving the filtering of AtMostNValue

We present how Lagrange multipliers can be altered to increase the level of filtering. First, we write the quantity $h''(\boldsymbol{\lambda}) + |q_j(\boldsymbol{\lambda})|$ as a function.

Definition 3. *The **forced lower bound function**, associated with the LP variable y_j, is given in* (7) *and its (sub)gradient with respect to $\boldsymbol{\lambda}$ in* (8)

$$\rho_j(\boldsymbol{\lambda}) = h''(\boldsymbol{\lambda}) + |q_j(\boldsymbol{\lambda})| = h''(\boldsymbol{\lambda}) + |1 - \boldsymbol{a_{cj}}^\top \boldsymbol{\lambda}|, \tag{7}$$

$$\tilde{\boldsymbol{\nabla}} \rho_j(\boldsymbol{\lambda}) = \tilde{\boldsymbol{\nabla}} h''(\boldsymbol{\lambda}) + \tilde{\boldsymbol{\nabla}} |q_j(\boldsymbol{\lambda})| = \mathbf{1} - \mathrm{A}\boldsymbol{y} - \mathrm{sgn}(q_j(\boldsymbol{\lambda}))\boldsymbol{a_{cj}}. \tag{8}$$

We introduce Algorithm 2 that locally alters the Lagrange multipliers in a way to enhance the filtering of Algorithm 1 for the AtMostNValue constraint. It performs several short Lagrangian optimization processes. Each Lagrangian optimization process has a different forced lower bound function $\rho_j(\boldsymbol{\lambda})$ as the objective function. Only a subset of forced lower bound functions are chosen for this second optimization process. The choice is based on how close $\rho_j(\boldsymbol{\lambda})$ is from $\max(\mathrm{dom}(N))$. If the difference is under a threshold τ, the algorithm is triggered. We optimize the *local* Lagrange multipliers $\bar{\boldsymbol{\lambda}}$ with these steps

$$\bar{\boldsymbol{\lambda}}^{(\boldsymbol{k+1})} = \max(\mathbf{0}, \quad \bar{\boldsymbol{\lambda}}^{(\boldsymbol{k})} + \beta_k \tilde{\boldsymbol{\nabla}} \rho_j(\bar{\boldsymbol{\lambda}}^{(\boldsymbol{k})})), \tag{9}$$

$$\beta_k = \omega_k \frac{(\max(\mathrm{dom}(N)) + \eta) - \rho_j(\bar{\boldsymbol{\lambda}}^{(\boldsymbol{k})})}{||\tilde{\boldsymbol{\nabla}}_{\bar{\lambda}} \rho_j(\bar{\boldsymbol{\lambda}}^{(\boldsymbol{k})})||^2}, \quad \omega_k \in (0, 2), \quad \eta > 0. \tag{10}$$

We chose a *Newton* type for the step size β_k with a *constant* coefficient ω_k as opposed to the variable coefficient of Table 1 where $\omega_k = \frac{5}{2^{\lfloor k/10 \rfloor}}$. The second parameter η prevents a null step size β_k at the start of the optimization. It happens

Algorithm 2: Algorithm to enhance filtering

```
1  for j = 1..m if |dom(Y_j)| ≠ 1 and max(dom(N)) − ρ_j(λ) < τ) do
2      y' ← y, λ̄ ← λ, q' ← q, h''(λ)' ← h''(λ), k ← 0
3      ∇̃ρ_j(λ̄) ← 1 − Ay − sgn(q_j(λ))a_cj
4      repeat
5          Update β_k // With equation (10)
6          λ̄' ← λ̄, λ̄ ← max(0, λ̄ − β_k ∇̃ρ_j(λ̄))
7          if (max |λ̄_i − λ̄'_i| > 0.0001) then
8              q' ← 1 − A^⊤ λ̄
9              for j' ← 1 to m do
10                 if (dom(Y_j') = {1}) or ((|dom(Y_j')| ≠ 1) and (q_j' < 0)) then
11                     y'_j' ← 1
12                 else y'_j' ← 0
13             h''(λ̄) ← q^⊤ y' + 1^⊤ λ̄
14             if (ρ_j(λ̄) > max(dom(N))) then Flag the value 1 − y_j in dom(Y_j)
15             ∇̃ρ_j(λ̄) ← 1 − Ay'
16             if ρ_j(λ̄) < max(dom(N)) then ∇̃ρ_j(λ̄) ← ∇̃ρ_j(λ̄) − sgn(q_j(λ))a_cj
17         k ← k + 1
18     until k = numberOfSteps or max_{1≤i≤n} |λ̄_i − λ̄'_i| ≤ 0.0001 or
         ρ_j(λ̄)) > max(dom(N))
```

whenever $\rho_j(\bar{\boldsymbol{\lambda}}^{(\boldsymbol{k})}) = \max(\mathrm{dom}(N))$. A certain number of steps (numberOfSteps) is allowed to optimize the local Lagrange multipliers. Otherwise, Algorithm 2 is very similar to Algorithm 1. The order of some operations is different because we want to start by updating the $\bar{\boldsymbol{\lambda}}$. The other difference is the line 16 of Algorithm 2. Here there is a choice of subgradient used to guide the search. The details are explained in section 3.1.

3.1 The theory

This section justifies the process used in Algorithm 2. The algorithm overcomes many difficulties and particularities of the problem. The forced lower bound functions (7) are neither convex nor concave. The first term, $h''(\boldsymbol{\lambda})$, is concave and the second term, $|q_j(\boldsymbol{\lambda})|$, is convex. This is a problem because subgradient methods converge to a local optimum. In Algorithm 1, the concavity of $h''(\boldsymbol{\lambda})$ saved the situation; here the situation requires some creativity because $\boldsymbol{\nabla}\rho_j(\boldsymbol{\lambda})$ might not lead to a global optimal solution. Algorithm 2 overcomes this major issue by choosing between two directions for the optimization of $\bar{\boldsymbol{\lambda}}$ (Algorithm 2 lines 15-16). To understand why this change overcomes the concavity issue, we start by analyzing some propriety of $\rho_j(\boldsymbol{\lambda})$ and $|q_j(\boldsymbol{\lambda})|$.

Theorem 1. *$\rho_j(\boldsymbol{\lambda})$ is concave on both side of $L_0(q_j)$.*

Proof. Given two concave functions $f_1(x)$ and $f_2(x)$, the function $f(x) = f_1(x) + f_2(x)$ is concave. The function $|q_j(\boldsymbol{\lambda})|$ is a convex function, but on each side of

$L_0(q_j)$ it is a linear function, which is both convex and concave. Since the sum of two concave function is concave, $\rho_j(\boldsymbol{\lambda})$ is concave on each side of $L_0(q_j)$. □

Theorem 1 alone is not enough to justify the optimization of $\rho_j(\boldsymbol{\lambda})$ with a subgradient. From the perspective of $\mathbb{R}^n$, there still exist two regions that provide local maxima of $\rho_j(\boldsymbol{\lambda})$, one on each side of $L_0(q_j)$. But if a value can be filtered from $\text{dom}(Y_j)$, then one of the two local optimal regions is a global maximum of $\rho_j(\boldsymbol{\lambda})$. The rest of this section is dedicated to explaining how the properties of $\rho_j(\boldsymbol{\lambda})$ and Theorem 1 can be used to find a global maximum of $\rho_j(\boldsymbol{\lambda})$ and justify a subgradient method on $\rho_j(\boldsymbol{\lambda})$. In the following, we assume that it is possible to filter one value from $\text{dom}(Y_j)$.

Before studying how to find a global maximum of $\rho_j(\boldsymbol{\lambda})$, we introduce some concepts. Since we consider both sides of $L_0(q_j)$ and we assume that we can filter one of the values from $\text{dom}(Y_j)$, we need to define a side of $L_0(q_j)$ where y_j has the correct value and a side where it does not. We say that a vector $\boldsymbol{\lambda}$ *generates* $\boldsymbol{y}$ if $\boldsymbol{y}$ is obtained from $\boldsymbol{\lambda}$ using (4).

Definition 4. *$\boldsymbol{y^*}$ is an optimal solution of the LP (1), given the current state of the CSP variables at the start of Algorithm 1, but without insurance that $\boldsymbol{y^*}$ can be generated by some $\boldsymbol{\lambda}$ in the LP (2).*

In Definition 4, the optimal solution is *optimal* with regard to the original function $h(\boldsymbol{y})$ and the current state of the CSP variables Y_k. Meaning $\boldsymbol{y^*}$ minimizes the number of values used to cover the integer variables $X_1, ..., X_n$. The following lemmas show why there is no certainty, yet, that $\boldsymbol{y^*}$ can be generated.

Lemma 2. *Given $\boldsymbol{y^*}$, if there is $y^*_j = 1$ and there exists another $j' \neq j$ with $\boldsymbol{a_{cj'}} = \boldsymbol{a_{cj}}$, then $y^*_{j'} = 0$.*

Proof. If $\boldsymbol{a_{cj'}} = \boldsymbol{a_{cj}}$, with $j \neq j'$, it means that for every CSP variable X_i with j in its domain, X_i also has j'. Because $y^*_j = 1$, the value j is used to cover a subset of $\{X_1, \ldots, X_n\}$, and every variable with j and j' in their domain will either be fixed to j or another value in their domain, but never j'. Choosing j' and j only increases the number of values used in the solution. Thus $y^*_{j'} = 0$. □

Lemma 3. *If in $\boldsymbol{y^*}$, there is $y^*_j = 1$ and there exists another $j' \neq j$ with $\boldsymbol{a_{cj'}} = \boldsymbol{a_{cj}}$, then there exists no $\boldsymbol{\lambda}$ that generates this solution.*

Proof. From Lemma 2, $y^*_{j'} = 0$. Suppose there exist a $\boldsymbol{\lambda}$ that generates $\boldsymbol{y^*}$. Then for q_j and $q_{j'}$ the equations that generate $y^*_j = 1$ and $y^*_{j'} = 0$ are $1 - \boldsymbol{a_{cj}}^\top \boldsymbol{\lambda} < 0$ and $1 - \boldsymbol{a_{cj'}}^\top \boldsymbol{\lambda} \geq 0$. Since $\boldsymbol{a_{cj'}} = \boldsymbol{a_{cj}}$, the system has no solution. □

From Lemma 3, if we remove duplicated columns, we ensure that an optimal solution $\boldsymbol{y^*}$ can be generated by some $\boldsymbol{\lambda}$. In practice, we did not observe that phenomena often. From now on, we assume that the columns are unique and that the optimal solution $\boldsymbol{y^*}$ can be generated by some Lagrange multipliers $\boldsymbol{\lambda^*}$. We can compute the gradient of $h''(\boldsymbol{\lambda})$ in the region that produces $\boldsymbol{y^*}$. We use this gradient to explain why we can change the search direction in Algorithm 2 (line 16) and eventually get to a global maximum of $\rho_j(\boldsymbol{\lambda})$.

Definition 5. *The vector* $\nabla h''(\boldsymbol{\lambda}^*) = \mathbf{1} - \mathrm{A}\boldsymbol{y}^*$ *is the gradient of* h'' *for* $\boldsymbol{y}^*$.

Normally the gradient of a function at an extremum is zero, but because of how we assign values to the LP variable y_k (4), it is not always the case.

Lemma 4. $\nabla h''(\boldsymbol{\lambda}^*)$ *is not always* $\mathbf{0}$.

Proof. Whenever a variable X_i has in its domain more than one value in common with the values of the optimal solution $|\operatorname{dom}(X_i) \cap \{j \mid y_j^* = 1\}| = \boldsymbol{a_{ri}}^\top \boldsymbol{y}^* > 1$, then the gradient at that index is non-zero $[\nabla h''(\boldsymbol{\lambda}^*)]_i = 1 - \boldsymbol{a_{ri}}^\top \boldsymbol{y}^* < 0$. □

Once the CSP variables $X_1, \ldots, X_n$ are all fixed to a value, then $\nabla h''(\boldsymbol{\lambda}^*) = 0$, but until that moment, it is not. We have all the information from the contribution of $h''(\boldsymbol{\lambda})$ to $\rho_j(\boldsymbol{\lambda})$. We analyze the last part of $\rho_j(\boldsymbol{\lambda})$, which is $|q_j(\boldsymbol{\lambda})|$.

Lemma 5. $|q_j(\boldsymbol{\lambda})|$ *has no derivative on* $L_0(q_j)$ *because for* $\boldsymbol{\lambda}$ *and* $\boldsymbol{\lambda'}$ *on two different sides of* $L_0(q_j)$ *we have* $\nabla|q_j(\boldsymbol{\lambda})| = -\nabla|q_j(\boldsymbol{\lambda'})|$.

Proof. First, we note that $\forall\ \boldsymbol{\lambda''} \notin L_0(q_j)$, $\nabla|q_j(\boldsymbol{\lambda''})| = \operatorname{sgn}(q_j(\boldsymbol{\lambda''}))\nabla(1 - \boldsymbol{a_{cj}}^\top \boldsymbol{\lambda''}) = -\operatorname{sgn}(q_j(\boldsymbol{\lambda''}))\boldsymbol{a_{cj}}$. Let $q_j(\boldsymbol{\lambda}) < 0$ and $q_j(\boldsymbol{\lambda'}) > 0$. Then, we have $\nabla|q_j(\boldsymbol{\lambda})| = +\boldsymbol{a_{cj}}$ and $\nabla|q_j(\boldsymbol{\lambda'})| = -\boldsymbol{a_{cj}}$, which completes the proof. □

From Lemmas 1 and 5, $\rho_j(\boldsymbol{\lambda})$ inherits the same set of points as $h''(\boldsymbol{\lambda})$ where $\nabla\rho_j(\boldsymbol{\lambda})$ does not exist, namely $\bigcup_{j'=1}^{m} L_0(q_{j'})$. Now we have all the tools to justify the use of a subgradient method for $\rho_j(\boldsymbol{\lambda})$.

Theorem 1 ensures that each gradient of $\rho_j(\boldsymbol{\lambda})$ on either of the two sides of $L_0(q_j)$ points toward the local maximum of $\rho_j(\boldsymbol{\lambda})$ of that side. On the side of $L_0(q_j)$ where $y_j = y_j^*$, the optimal region of $h''(\boldsymbol{\lambda})$ is present. From the definition of $\rho_j(\boldsymbol{\lambda})$, for any $\boldsymbol{\lambda}$ we have the following relation $\rho_j(\boldsymbol{\lambda}) \geq h''(\boldsymbol{\lambda})$. Which means that for any $\boldsymbol{\lambda}^*$ in the optimal region of $h''(\boldsymbol{\lambda})$, we have $\rho_j(\boldsymbol{\lambda}^*) \geq h''(\boldsymbol{\lambda}^*)$. In fact, for any $\boldsymbol{\lambda}^* \notin L_0(q_j)$ we have $\rho_j(\boldsymbol{\lambda}^*) > h''(\boldsymbol{\lambda}^*)$ which means filtering whenever $h''(\boldsymbol{\lambda}^*) = \max(\operatorname{dom}(N))$. The gradient of $\rho_j(\boldsymbol{\lambda})$ on this side of $L_0(q_j)$ points toward a $\boldsymbol{\lambda}$ that maximizes $\rho_j(\boldsymbol{\lambda}) > h''(\boldsymbol{\lambda}^*)$.

For the case where $y_j \neq y_j^*$, we need two other theorems to explain how we can find a direction to leave this local maximum of $\rho_j(\boldsymbol{\lambda})$ and move toward a global maximum. We show that there exist a region on the side where $y_j \neq y_j^*$ of $L_0(q_j)$, for which the gradient of $\rho_j(\boldsymbol{\lambda})$ is equal to the gradient $\nabla h''(\boldsymbol{\lambda}^*)$. This region corresponds to the set of global maxima on this side of $L_0(q_j)$.

Theorem 2. *Given* $\boldsymbol{y}^*$, *let* $\boldsymbol{\lambda}^*$ *be one of the vectors that generates* $\boldsymbol{y}^*$, *let* $\boldsymbol{\lambda}$ *be one of the vectors that generates* $\boldsymbol{y}$ *where* $\forall k \neq j, y_k = y_k^*$ *and* $y_j \neq y_j^*$. *Then* $\nabla\rho_j(\boldsymbol{\lambda}) = \nabla h''(\boldsymbol{\lambda}^*)$.

Proof. We can write the vector $\boldsymbol{y}$ generated as $\boldsymbol{y} = \boldsymbol{y}^* + \operatorname{sgn}(q_j(\boldsymbol{\lambda}^*))\hat{\boldsymbol{u}}_{\boldsymbol{j}}$, where $\hat{\boldsymbol{u}}_{\boldsymbol{j}}$ is a unit vector with only the j-th entry equal to one and the other entries equal to zero. For $\boldsymbol{\lambda}$, the gradient of h'' is

$$\begin{aligned}\nabla h''(\boldsymbol{\lambda}) &= \mathbf{1} - \mathrm{A}(\boldsymbol{y}^* + \operatorname{sgn}(q_j(\boldsymbol{\lambda}^*))\hat{\boldsymbol{u}}_{\boldsymbol{j}}) = \mathbf{1} - \mathrm{A}\boldsymbol{y}^* - \operatorname{sgn}(q_j(\boldsymbol{\lambda}^*))\boldsymbol{a_{cj}},\\ &= \nabla h''(\boldsymbol{\lambda}^*) + \nabla|q_j(\boldsymbol{\lambda}^*)| = \nabla h''(\boldsymbol{\lambda}^*) - \nabla|q_j(\boldsymbol{\lambda})|\end{aligned}$$

where we change the last term of the second equation into a more intuitive object. We use Lemma 5 to change the last term to $-\boldsymbol{\nabla}|q_j(\boldsymbol{\lambda})|$. The gradient of ρ_j for $\boldsymbol{\lambda}$ is

$$\boldsymbol{\nabla}\rho_j(\boldsymbol{\lambda}) = (\boldsymbol{\nabla}h''(\boldsymbol{\lambda}^*) - \boldsymbol{\nabla}|q_j(\boldsymbol{\lambda})|) + \boldsymbol{\nabla}|q_j(\boldsymbol{\lambda})| = \boldsymbol{\nabla}h''(\boldsymbol{\lambda}^*). \qquad \square$$

Theorem 2 justifies when should Algorithm 2 change the subgradient on line 16. From Theorem 1, $\rho_j(\boldsymbol{\lambda})$ is concave on either side of $L_0(q_j)$, therefore for each of these two regions, the subgradient method converges to an optimal in each region. If y_j^* can only be 0 or only be 1 in all possible optimal solutions, then the value $1 - y_j^*$ must be filtered. The region of Lagrange multipliers that generates $\boldsymbol{y}$ from Theorem 2 is the local optimal region of $\rho_j(\boldsymbol{\lambda})$ on the side of $L_0(q_j)$ where $y_j \neq y_j^*$. In the next theorem, we formally explain why Algorithm 2 uses two subgradients.

Theorem 3. *If value $1-y_j^*$ needs to be filtered from $\mathrm{dom}(Y_j)$, then in the region where $\boldsymbol{\lambda}$ generates $y_j = 1 - y_j^*$ but every other $y_k = y_k^*$, following $\boldsymbol{\nabla}h''(\boldsymbol{\lambda})$ bring the next $\boldsymbol{\lambda'}$ in the optimization closer to $L_0(q_j)$ and the filtering zone.*

Proof. The direction that brings $y_j \neq y_j^*$ to a sate $y_j = y_j^*$ is $-\boldsymbol{\nabla}|q_j(\boldsymbol{\lambda})|$, Lemma 5. Since $h''(\boldsymbol{\lambda})$ is concave then every region that is not the optimal region points toward this zone. So $(-\boldsymbol{\nabla}|q_j(\boldsymbol{\lambda})|)^\top(\boldsymbol{\nabla}h''(\boldsymbol{\lambda})) \geq 0$. Hence the next $\boldsymbol{\lambda}$ in the optimization is closer to $L_0(q_j)$ and the optimal region. The optimal region is in the filtering zone because, $\forall\boldsymbol{\lambda}$ we have $\rho_j(\boldsymbol{\lambda}) \geq h''(\boldsymbol{\lambda})$, we can conclude that for any $\boldsymbol{\lambda^*}$ which generates $\boldsymbol{y^*}$ we have filtering. $\square$

Algorithm 2 can alternate between the two sides of $L_0(q_j)$. This situation is due to a step-size that is to big in (9). Eventually, when the step-size is small enough, the algorithm stops cycling. We conclude that Algorithm 2 always approaches the filtering zone.

4 Experimentations and results

We implemented Algorithm 1, denoted LR^0, based on the code that Cambazard and Fages [7] shared to us. We implemented our approach which we denote LR^+. It consists of the implementation of Algorithms 1 and 2. The code is accessible on GitHub[1]. We fixed the threshold to 0.4 and the number of steps to 40. The experiments ran on a MacBook Pro with M2 chip and with 8 Gb of RAM.

We test the algorithms on two problems: the dominating queens problem and the p-median problem. The instances for the second problem come from the discrete locations problem library [12] "*instances with a large duality gap*". There are three classes of instances (easier to harder): A, B and C. There are thirty instances within each class.

For all the experiences, we have chosen our constant step-size for our experiment: $\omega_k = 1.97$ and $\eta = 0.0001$ to determine the step size β_k in equation (10).

[1] `https://github.com/frbert3/LocalAlterationsAlgorithmAtMostNValue.git`

		Harmonic				Geometric				Newton			
n/v	F	Nodes	Fails	Iters	Time	Nodes	Fails	Iters	Time	Nodes	Fails	Iters	Time
7/4	Y LR^0	142	112	170k	0.165	166	135	84k	0.089	176	145	19k	0.026
	LR^+	**31**	**4**	**19k**	**0.024**	**31**	**4**	**6k**	**0.011**	**31**	**4**	**2k**	**0.005**
8/5	Y LR^0	267	225	329k	0.301	362	320	186k	0.197	283	241	33k	0.045
	LR^+	**41**	**6**	**26k**	**0.040**	**76**	**41**	**24k**	**0.028**	**41**	**6**	**3k**	**0.007**
8/4	N LR^0	2k	2k	2.4M	3.497	3k	3k	1.6M	2.461	2k	2k	279k	0.765
	LR^+	**1.2k**	**1.2k**	**1.5M**	**2.066**	**1k**	**1k**	**698k**	**1.177**	**1k**	**1k**	**146k**	**0.362**
9/5	Y LR^0	1k	953	1.1M	2.188	1k	969	544k	1.315	956	909	113k	0.443
	LR^+	**490**	**443**	**576k**	**0.880**	**561**	**514**	**304k**	**0.556**	**441**	**394**	**57k**	**0.167**
10/5	Y LR^0	113k	113k	133M	300	195k	195k	107M	300	944k	944k	91M	300
	LR^+	**9.4k**	**9.4k**	**11M**	**26.9**	**761**	**725**	**471k**	**3.711**	**844**	**808**	**121k**	**2.59**
11/5	Y LR^0	32k	32k	38M	102.2	54k	54k	30M	83.1	83k	83k	7.34M	37.4
	LR^+	**4k**	**4k**	**5.6M**	**28.7**	**3.4k**	**3.4k**	**2.1M**	**11.6**	**4k**	**4k**	**461k**	**9.61**

Table 2: Dominating queens instances on $n \times n$ chessboards with v queens. If the instance is *feasible* we wrote Y and if not N. The time is in seconds.

4.1 The Dominating Queens problem

The dominating queens problem consists of placing the minimal number v of chess queens on a chessboard of dimensions $n \times m$ in order that every square is occupied or attacked by at least one queen. We focus on the case where $n = m$ which has more symmetries and is harder to solve.

We generate a chessboard with squares numbered from 1 to n^2 from the top-left corner to the bottom-right corner. We associate each square i with an integer variable X_i whose domain is the set of squares a queen would be able to attack from position i. We also associate a boolean variable Y_i to each square, representing whether there is a queen on the i-th square. The variable N has for domain $\{0, \ldots, v\}$. There are $2n^2 + 1$ variables in total. The objective is to minimize N. Therefore the model is

$$\textbf{Minimize} \quad N \tag{11}$$

$$\textsc{AtMostNValue}(\mathcal{X}, \mathcal{Y}, N) \tag{12}$$

$$(Y_j = 1 \Leftrightarrow \exists X_i \in \mathcal{X} \mid X_i = j) \quad \forall j \in \{1, \ldots, n^2\}, \tag{13}$$

(12) ensures that all squares are covered by the N queens. (13) ensures that if $Y_j = 1$ then there is at least one variable with $X_i = j$. Branching on the variables in $\mathcal{X}$ is performed lexicographically. A timeout of 300 seconds is fixed.

Table 2 contains the results of 6 instances of the *dominating queens problem*, five are feasible and one is not. The 4 metrics are *Nodes*, the numbers of explored nodes; *Fails*, the failures during the filtering, *Time*, the solving time in seconds; and *Iters*, the number of iterations done in the Lagrangian relaxation.

LR^0 does not find the solution for the 10/5 instance within the time limit. This instance is excluded from the next averages. LR^+ explores 71.8% fewer nodes, does 77.7% fewer failures, does 73.3% fewer iterations in the Lagrangian

relaxation, and is 71.2% faster than LR^0. The local alterations algorithm significantly reduces the number of explored nodes. For the three-step size rules, the algorithm seems to reduce the nodes and fails metrics by closely the same amount for a given instance. We use a *Newton* type of step size, so it is not surprising that the additional filtering reduces those metrics in a similar way.

4.2 The p-median problem

The p-median problem is a variant of the facility location problem which consists of opening *at most* p facilities on a territory to deserve n clients at a minimal cost. In this variation, the problem focuses on disposing the p facilities in a way to minimize the total transportation cost from the facilities to the clients.

We use the same model as Cambazard and Fages [7]. There are m candidate facility locations. Each client X_i can be served by a subset of facilities, noted $P(i) \subset \{1, \ldots, m\}$. We associate a Boolean variable Y_j to each facility location j to represent if we open the facility j or not. The transportation cost of the client X_i for the facility j is contained in the matrix $\mathrm{C}_{n \times m}$, with entries ∞ if $j \notin P(i)$. The variables T_{Ci} represent the transportation costs for the clients X_i. Their values are subject to an $\textsc{Element}(Index, table, Value)$ constraint, where $Index$ and $Value$ are variables and $table$ are constants. This constraint ensures that $Value = table[Index]$.

$$\textbf{Minimize} \quad \textstyle\sum_{i=1}^{n} T_{Ci} \tag{14}$$

$$\textsc{AtMostNValue}(\mathcal{X}, \mathcal{Y}, N) \tag{15}$$

$$\textsc{Element}(X_i, [c_{i1}, ..., c_{im}], T_{Ci}) \quad \forall i \in \{1, \ldots, n\} \tag{16}$$

$$\textstyle\sum_{j=1}^{m} Y_j = N \tag{17}$$

$$\text{if} \quad ((c_{ia} < c_{ib}) \vee ((c_{ia} = c_{ib}) \wedge (a < b)) \text{ then } Y_a = 1 \Rightarrow X_i \neq b \tag{18}$$

$$X_i \in P(i), \quad T_{Ci} \geq 0 \quad \forall i \in \{1, \ldots, n\}, \quad N \in [1, p] \tag{19}$$

The constraints (18) (there is one for each facility) help to choose which clients are served by which facility: the cheapest facility. We fix a timeout of 300 seconds.

We use two search heuristics : *standard* and *cost*. Both heuristics branch on the facilities Y_j first, because once these variables are fixed, the constraints (18) filter the variables X_i. The heuristic branch on the client variables X_i second applying the first fail principle, starting with the smallest value in the domains.

The *standard* heuristic follows the first fail principle for the facilities variables starting with the smallest domain size.

The *cost* based heuristic, introduced by Cambazard and Fages [7], is also based on the first fail principle. It branches on the facility variable Y_j which belongs to the client X_i who is served by the smallest number of facilities, the minimum domain size. It breaks ties with the cheapest facility j in average, $\frac{\sum_{i|j \in \mathrm{dom}(X_i)} c_{ij}}{|\{i|j \in \mathrm{dom}(X_i)\}|}$.

Table 3 shows the results for the p-median problem. The column *Optimal* reports the number of times the value of the optimal solution is found, without

necessarily proving that the solution is optimal. The optimal value is given in the benchmark. The column *Proof optimal* is the number of times the solution is proven optimal. The column *# of better solutions* is the number of times that the algorithm finds a solution with a smaller objective value than other algorithm finds. The column *Av. time gain of* LR^+ *on* LR^0 is the average percentage speedup by LR^+ over LR^0 for the instances proven optimal. The column *Av. node reduction from* LR^0 *to* LR^+ is the average percentage of nodes not explored by LR^+ compared to LR^0 for the instances on which we prove the optimality.

We see in Table 3 that for the *easy* instances class A, only LR^+ achieves to find all the optimal solutions. It is also on this class of instances that we record the greatest speedup on instances for which both algorithms prove the optimality. For class A, the speedups of the standard and cost-based heuristics are 28.4% (in average 18 seconds) and 33.2% (in average 17 seconds).

Figure 1 presents a visual comparison of LR^0 and LR^+ on the time to solve the instances from the classes A and B. Figure 1a shows the instances that LR^0 and LR^+ achieve the proof of optimality. Figure 1b displays all instances.

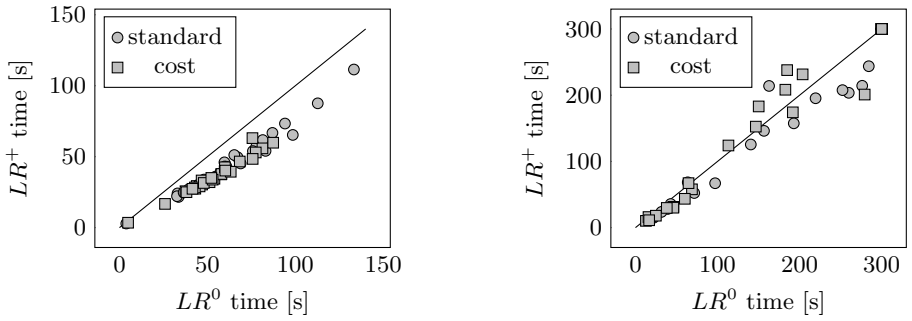
Figure 1a shows that for class A, LR^+ is always faster than LR^0. We stress that LR^+ finds solutions faster and with fewer nodes than LR^0.

For instances of class B, both LR^0 and LR^+ find the same number of optimal solutions and achieve, on the same instances, the proof of optimality. With the standard heuristic, LR^+ finds two better solutions than LR^0 when neither can find the optimal solution. The average speedup for the standard and cost-based heuristics are 15.6% (in average 20 seconds) and 8.10% (in average 9 seconds).

Figure 1b shows that when using a more accurate search heuristic, as the cost based heuristic, LR^+ becomes less effective. In other words, LR^+ wastes time at filtering values for which the heuristic has no intention to branch on. Nevertheless, LR^+ is still faster than LR^0 and explores 72.6% fewer nodes with the standard heuristic (in average 94764 fewer nodes) and 71.3% fewer nodes with the cost based heuristic (which represents in average 85251 fewer nodes).

Problem	Heuristics	Optimal		Proof optimal		# of better solutions		Av. time gain of LR^+ on LR^{0*}	Av. node reduction from LR^0 to LR^{+*}
		LR^0	LR^+	LR^0	LR^+	LR^0	LR^+	[%]	[%]
A	standard	27	27	27	27	0	**1**	28.4	71.5
	cost	29	**30**	27	27	0	**1**	33.2	72.0
B	standard	28	28	18	18	0	**2**	15.6	72.6
	cost	28	28	18	18	0	0	8.10	71.3
C	standard	5	**6**	0	0	0	**8**	–	–
	cost	8	**12**	0	0	0	**13**	–	–

Table 3: Results of the three classes of problem of the facility location problem. Each class has 30 instances and the solutions are known.
* *On the instances for which we were able to prove the optimality.*



(a) P-MGAP A (b) P-MGAP B

Fig. 1: Time comparison of LR^0 and LR^+

On the hardest instances in class C, no algorithms are able to prove the optimality within the time limit. Although, with both search heuristics, LR^+ finds the optimal solution of more instances than LR^0. Furthermore, LR^+ finds better solutions than LR^0 for 8 and 13 instances.

Difference between the problems The main difference between the results on the dominating queens problem and the p-median problem is the average speedup. The speedup is less significant in the latter. The p-median problem is more complex than the dominating queen problem because there are more constraints and more variables. Also, those are instances randomly generated, the matrix A has no structure. In contrast with the very structured matrix A in the dominating queens problem. This might affect the convergence of the algorithm.

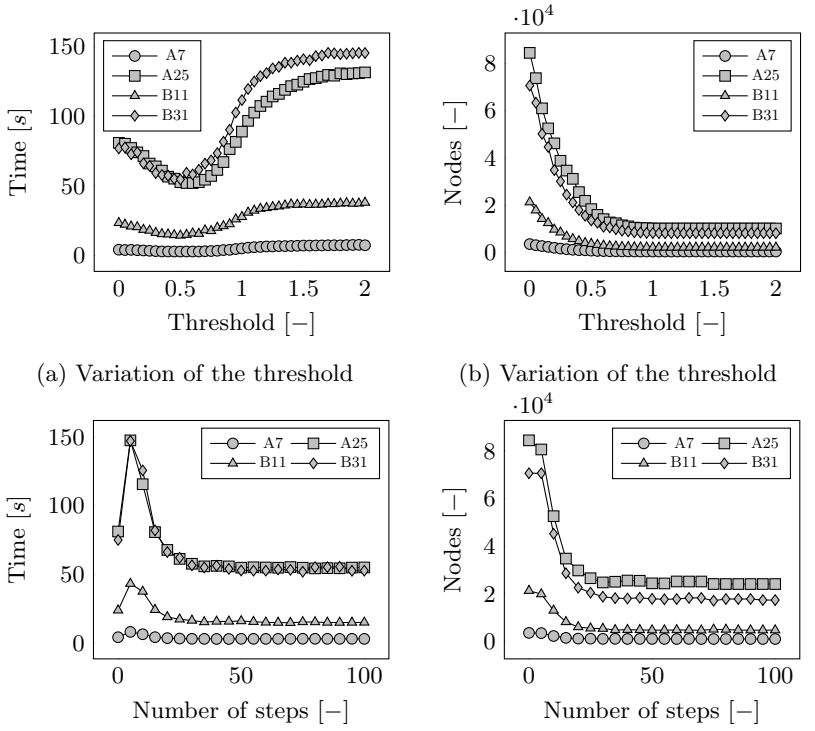
4.3 Determining parameters

Experimental data helped us fix the parameters' values. Figure 2 shows the solving time and number of nodes of four instances of the p-median problem (A7, A25, B11 and B31) in function of the threshold and number of steps.

The solving time curves (Figure 2a) follow the same trend. When the threshold lies in $[0, 0.25]$, the solving time diminishes because LR^+ starts filtering values. In $[0.25, 0.5]$, LR^+ filters even more values, but beyond 0.5 it considers too many values and wastes time. All curves share a minimum around 0.50.

In Figure 2b, we see that when the threshold augments, the number of explored nodes decreases almost exponentially until it stops to a value. After 0.40, the number of explored nodes starts to decrease much slowly.

In Figure 2c, we see that the solving time stops decreasing after 40 steps. The variation in time induced by the number of steps between LR^0 (the first point) and LR^+ is less than the variation induced by the threshold. It is not surprising. The threshold dictates when to trigger LR^+ and the number of steps dictates how much time is spent in LR^+. We see that for a number of steps between 1 and 15, the time increases, because we are not converging fast enough to a maximum. We lose time by trying to filter values without enough steps.



(a) Variation of the threshold

(b) Variation of the threshold

(c) Variation of the number of steps

(d) Variation of the number of steps

Fig. 2: Variation of the *threshold* and *number of steps* parameters

In Figure 2d, we see a tendency similar to the variation of the threshold. Again the variation, induced by the number of steps, of the number of explored nodes is less than the variation induced by the threshold. Nevertheless, we can infer that a greater number of steps helps filtering more values.

5 Conclusion

We conceived a Lagrangian relaxation based algorithm for the AtMostNValue constraint that uses local alterations of the multipliers to enhance the filtering. This new filtering algorithm leads to an average speedup of 71% on the dominating queens problem and average speedups of 33% and 8% on the instances of class A and class B of the p-median problem.

Acknowledgement We thank Hadrien Cambazard for sharing with us his code for the AtMostNValue. It served as a thorough guide for the present work.

References

1. Beldiceanu, N.: Pruning for the minimum constraint family and for the number of distinct values constraint family. In: Principles and Practice of Constraint Programming. pp. 211–224 (2001)
2. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog (2010)
3. Bertsekas, D.P.: On the goldstein-levitin-polyak gradient projection method. IEEE Transactions on automatic control **21**(2), 174–184 (1976)
4. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Filtering algorithms for the nv alue constraint. Constraints **11**, 271–293 (2006)
5. Boudreault, R., Quimper, C.G.: Improved cp-based lagrangian relaxation approach with an application to the tsp. In: IJCAI. pp. 1374–1380 (2021)
6. Boyd, S., Xiao, L., Mutapcic, A.: Subgradient methods. lecture notes of EE392o, Stanford University, Autumn Quarter **2004**, 2004–2005 (2003)
7. Cambazard, H., Fages, J.G.: New filtering for atmostnvalue and its weighted variant: A lagrangian approach. Constraints **20**(3), 362–380 (2015)
8. Fisher, M.L.: An applications oriented guide to lagrangian relaxation. Interfaces **15**(2), 10–21 (1985)
9. Focacci, F., Lodi, A., Milano, M.: Cost-based domain filtering. In: International conference on principles and practice of constraint programming. pp. 189–203 (1999)
10. Fumero, F.: A modified subgradient algorithm for lagrangean relaxation. Computers & Operations Research **28**(1), 33–52 (2001)
11. Held, M., Wolfe, P., Crowder, H.P.: Validation of subgradient optimization. Mathematical programming **6**(1), 62–88 (1974)
12. Kochetov, Y., Ivanenko, D.: Computationally difficult instances for the uncapacitated facility location problem. Metaheuristics: Progress as real problem solvers pp. 351–367 (2005)
13. Lemaréchal, C.: Lagrangian relaxation. Computational combinatorial optimization: optimal or provably near-optimal solutions pp. 112–156 (2001)
14. Petit, T., Régin, J.C., Bessiere, C.: Specific filtering algorithms for over-constrained problems. In: Principles and Practice of Constraint Programming. pp. 451–463 (2001)
15. Polyak, B.T.: Minimization of unsmooth functionals. USSR Computational Mathematics and Mathematical Physics **9**(3), 14–29 (1969)
16. Sellmann, M.: Theoretical foundations of cp-based lagrangian relaxation. In: Principles and Practice of Constraint Programming. pp. 634–647 (2004)