GAC for a Linear Inequality and an Atleast Constraint with an Application to Learning Simple Polynomials

N. Razakarison
ENS Cachan, France

N. Beldiceanu
TASC team (CNRS/INRIA)
Mines de Nantes, France

M. Carlsson
SICS, Sweden

H. Simonis*
4C, University College Cork
Ireland

Abstract

We provide a filtering algorithm achieving GAC for the conjunction of constraints $\texttt{atleast}(b, \langle x_0, x_1, \ldots, x_{n-1}\rangle, \mathcal{V})$ and $\sum_{i=0}^{n-1} a_i \cdot x_i \leq c$, where the `atleast` constraint enforces b variables out of $x_0, x_1, \ldots, x_{n-1}$ to be assigned to a value in the set $\mathcal{V}$. This work was motivated by learning simple polynomials, i.e. finding the coefficients of polynomials in several variables from example parameter and function values. We additionally require that coefficients be integers, and that most coefficients be assigned to zero or integers close to 0. These problems occur in the context of learning constraint models from sample solutions of different sizes. Experiments with this more global filtering show an improvement by several orders of magnitude compared to handling the constraints in isolation or with `cost_gcc`, while also out-performing a 0/1 MIP model of the problem.

1 Introduction

Considering conjunction of constraints is a way of getting better propagation, and the last years have witnessed significant research on how to come up with efficient filtering algorithms that handle a conjunction of two well known constraints. This was for instance the case for combining the `alldifferent` constraint with `precedences` (Bessière et al. 2011) or with the `sum` constraint (Beldiceanu et al. 2012). This was also the case for combining the `sum` constraint with difference constraints (Régin and Rueher 2000), or with a chain of `precedences` (Petit, Régin, and Beldiceanu 2011), or with a covering set of clique constraints in the context of 0/1 variables (Puget 2004). Initially motivated by learning simple polynomials in the context of learning generic constraint models (Beldiceanu and Simonis 2012), i.e. polynomials that have a large number of small coefficients (in absolute value) and parameters in $\mathbb{N}$, this paper addresses the question of finding an efficient generalized arc-consistency (GAC) filtering algorithm for the conjunction of a linear inequality and an `atleast` constraint. More precisely, given the constraints $\texttt{atleast}(b, \langle x_0, x_1, \ldots, x_{n-1}\rangle, \mathcal{V})$ and $\sum_{i=0}^{n-1} a_i \cdot x_i \leq c$ $(b, n \in \mathbb{N}, a_i, c, \in \mathbb{Z}, \mathcal{V} \in \mathcal{P}(\mathbb{Z}), n \neq 0)$, where x_i $(0 \leq i < n)$ are domain variables (see Section 3), and `atleast` enforces at least b variables from $x_0, x_1, \ldots, x_{n-1}$ be assigned to a value in set $\mathcal{V}$,[1] this paper provides a GAC filtering algorithm. Note that only enforcing bounds consistency on a single linear inequality already yields GAC as noted in (Zhang and Yap 2000). A filtering algorithm achieves GAC for a given constraint ctr if and only if for every variable var of ctr there exists at least one solution for ctr such that var can be assigned to any value in its domain, and every other variable var' of ctr to a value in its domain (Bessière 2006).

Note that, by complementing the set of values $\mathcal{V}$, we can replace an `atmost` constraint by the `atleast` constraint and still use the same filtering algorithm. Also, by inverting the signs of the coefficients a_i, the same algorithm can propagate $\geq$ instead of $\leq$.

Section 2 gives an intuition of the methodology used for systematically deriving the GAC filtering algorithm from a feasible lower bound of the sum of the variables of the linear inequality subject to the `atleast` constraint. Section 3 provides a necessary and sufficient condition for the conjunction of a linear inequality and an `atleast` constraint, which is based on a sharp lower bound of the sum $\sum_{i=0}^{n-1} a_i \cdot x_i$ subject to the `atleast` constraint. We call this conjunction `linear_atleast`. Section 4 shows how to compute the increase of this sharp lower bound when a variable x_i is assigned to a value u. Section 5 derives a GAC filtering algorithm from the results presented in the preceding sections. Section 6 describes related work and Section 7 presents the problem of learning simple polynomials in the context of learning generic constraint models. Before we conclude, Section 8 evaluates our algorithm on randomized instances of our application problem and compare it with a reformulation as well as with `cost_gcc` (Régin 2002).

2 Intuition

In order to get a GAC filtering algorithm we first focus on getting a necessary and sufficient condition for the feasibility of the conjunction of a linear inequality and an `atleast`

*Supported by EU FET grant ICON (project number 284715).

[1] In the context of learning simple polynomials $x_0, x_1, \ldots, x_{n-1}$ represent the coefficients of the polynomial to learn; the set $\mathcal{V}$ of the `atleast` constraint is set to a small interval around 0 since we want to control the minimum number of coefficients set to a value around 0.

constraint. This condition is based on a sharp lower bound of the sum of the linear term that also considers the atleast constraint.

In a second step we concentrate on defining for every variable-value pair (var, val) the increase, i.e. the *regret*, of the previous lower bound when var is assigned to val (i.e., the reduced cost introduced by (Focacci, Lodi, and Milano 1999)). We come up with a number of mutually disjoint cases on the pair (var, val).

In a third step, since it is obviously too expensive to enumerate all possible variable-value pairs for filtering, we instead group together consecutive values of a given variable that correspond to the same regret case. We find out all possible sequences of regret cases in order to identify maximum size intervals of values corresponding to the same regret case.

In a fourth step, given a variable and its maximum intervals, we have for each such interval one linear function that, given a value val in the interval, returns the increase of the lower bound, i.e. see functions f and g in Example 1 and see Lemma 2 where these functions are defined. From the maximum value of the right hand side of the linear inequality and from the function defining the regret on an interval of values we directly compute the intervals of infeasible values in the corresponding interval.

Example 1. *We informally illustrate the previous steps on the conjunction of constraints* $x_0 \in [3,10], x_1 \in [0,1] \cup [5,9]$, $x_2 \in [0,3] \cup [6,9]$, $x_0 + 2 \cdot x_1 - x_2 \leq 5$, $\texttt{atleast}(2, \langle x_0, x_1, x_2 \rangle, \{4,6\})$. *The conjunction of constraints admits four solutions* $\langle x_0 = 4, x_1 = 0, x_2 = 6 \rangle$, $\langle x_0 = 4, x_1 = 1, x_2 = 6 \rangle$, $\langle x_0 = 6, x_1 = 0, x_2 = 6 \rangle$, $\langle x_0 = 6, x_1 = 1, x_2 = 6 \rangle$. *We focus on the filtering of the domain of variable* x_0 *(GAC), which reduces its domain to* $\{4,6\}$*, i.e., it creates a hole in the domain of* x_0.

- *Ignoring the* atleast *constraint, the lower bound of* $x_0 + 2 \cdot x_1 - x_2$ *is equal to* -6*, where depending on the sign of their coefficients, variables* x_0*,* x_1*,* x_2 *are assigned to 3, 0, 9, respectively. But to satisfy the* atleast *constraint we need two values in the set* $\{4,6\}$*. The feasible lower bound* $\ell = -2$ *is obtained by reassigning* x_0 *from* 3 *to 4 and* x_2 *from* 9 *to* 6*, which leads to the smallest possible increase.*
- *For a value* $u \in \{4,6\}$ *the increase of the feasible lower bound* $\ell = -2$ *when reassigning* x_0 *to* u *is equal to* $f(u) = u - 4$*. This stems from the fact that* x_0 *was already assigned to 4 in* ℓ *and that we do not change the number of variables assigned to a value in* $\{4,6\}$*. For a value* u *in* $\{3,5,7..10\}$ *the increase of* $\ell = -2$ *when reassigning* x_0 *to* u *is* $g(u) = (u-4) + 12$*, "*$u - 4$*" since we reassign* x_0 *from 4 to* u*, "*$+12$*" since we reassign, in the term* $2 \cdot x_1$*, variable* x_1 *from 0 to the smallest value in* $\mathrm{dom}(x_1) \cap \{4,6\}$*, i.e. 6, to still satisfy the* atleast *constraint.*
- *Enforcing* $\ell + f(u) \leq 5$ *on the set* $\{4,6\}$ *does not lead to any filtering, while enforcing* $\ell + g(u) \leq 5$ *on* $\{3,5,7..10\}$ *leads to removing* $\{3,5,7..10\}$ *from* x_0.

3 Necessary and Sufficient Condition for Feasibility

In order to evaluate the feasibility of the conjunction of the constraints $\sum_{i=0}^{n-1} a_i \cdot x_i \leq c$ and atleast, we need to evaluate the minimum value of the left hand side of the inequality subject to the atleast constraint and check that it does not exceed c. For this purpose, we first introduce some notation:

- A *domain variable* x is a variable that ranges over a finite set of integers in $\mathbb{Z}$ denoted by $\mathrm{dom}(x)$; $\underline{x}$ and $\overline{x}$ respectively denote the minimum and maximum value of $\mathrm{dom}(x)$.
- Let $\mathcal{V}_i$ denote the set $\mathcal{V} \cap \mathrm{dom}(x_i)$.
- Let $\mathcal{X}^+_{out}$ (resp. $\mathcal{X}^-_{out}$) denote the set of indices i $(0 \leq i < n)$ such that $a_i \geq 0$ (resp. $a_i < 0$) and $\mathcal{V}_i = \varnothing$.
- Let $\mathcal{X}^+_{in}$ (resp. $\mathcal{X}^-_{in}$) denote the set of indices i $(0 \leq i < n)$ such that $a_i \geq 0$ (resp. $a_i < 0$) and $\mathcal{V}_i \neq \varnothing$. Let $\underline{v_i}$ (resp. $\overline{v_i}$) denote the smallest (resp. largest) value of $\mathcal{V}_i$.
- For each element i of $\mathcal{X}^+_{in}$ (resp. $\mathcal{X}^-_{in}$) we introduce the quantity $\delta_i = a_i \cdot (\underline{v_i} - \underline{x_i})$ (resp. $\delta_i = a_i \cdot (\overline{v_i} - \overline{x_i})$), which represents the minimum increase of the term $\sum_{i \in \mathcal{X}^+_{in}} a_i \cdot \underline{x_i}$ (resp. $\sum_{i \in \mathcal{X}^-_{in}} a_i \cdot \overline{x_i}$) wrt. assigning x_i to a value in $\mathcal{V}_i$ rather than to $\underline{x_i}$ (resp. $\overline{x_i}$).
- Given the set of δ_i $(i \in \mathcal{X}^+_{in} \cup \mathcal{X}^-_{in})$, let $\sigma_0, \sigma_1, \ldots, \sigma_{|\mathcal{X}^+_{in}| + |\mathcal{X}^-_{in}| - 1}$ denote the δ_i sorted in increasing order and increasing index i in case of ties.
- For $i \in \mathcal{X}^+_{in} \cup \mathcal{X}^-_{in}$, let p_i denote the position of δ_i in the sequence σ, and let Δ_i denote the sum $\sum_{k=0}^{i-1} \sigma_k$. For $i \in \mathcal{X}^+_{out} \cup \mathcal{X}^-_{out}$, let $p_i = n$.

Lemma 1. *Assuming that each variable* x_i $(0 \leq i < n)$ *can be assigned to any value in its domain, a necessary and sufficient condition for the feasibility of* $\texttt{atleast}(b, \langle x_0, x_1, \ldots, x_{n-1} \rangle, \mathcal{V}) \wedge \sum_{i=0}^{n-1} a_i \cdot x_i \leq c$ *consists of the following two conditions:*

1. $|\mathcal{X}^+_{in}| + |\mathcal{X}^-_{in}| \geq b$.
2. $\ell = \sum_{i \in \mathcal{X}^+_{in} \cup \mathcal{X}^+_{out}} a_i \cdot \underline{x_i} + \sum_{i \in \mathcal{X}^-_{in} \cup \mathcal{X}^-_{out}} a_i \cdot \overline{x_i} + \Delta_b \leq c$.

Proof. First observe that $|\mathcal{X}^+_{in}| + |\mathcal{X}^-_{in}| \geq b$ must hold in order to satisfy the atleast constraint. Assuming that $|\mathcal{X}^+_{in}| + |\mathcal{X}^-_{in}| \geq b$ holds, we now show how to build an assignment that both satisfies the atleast constraint and minimizes the sum $\sum_{i=0}^{n-1} a_i \cdot x_i$.

The minimum value of the sum $\sum_{i=0}^{n-1} a_i \cdot x_i$ is achieved by setting those x_i with a positive or zero (resp. negative) coefficient a_i to their minimum (resp. maximum) value. This leads to the quantity $s = \sum_{i \in \mathcal{X}^+_{in} \cup \mathcal{X}^+_{out}} a_i \cdot \underline{x_i} + \sum_{i \in \mathcal{X}^-_{in} \cup \mathcal{X}^-_{out}} a_i \cdot \overline{x_i}$. However we may have to correct (increase) this quantity in order to handle the fact that at least b variables from $x_0, x_1, \ldots, x_{n-1}$ must be assigned to a value in $\mathcal{V}$. For each variable x_i that can be eventually assigned to a value in $\mathcal{V}$ (i.e., $x_i | i \in \mathcal{X}^+_{in} \cup \mathcal{X}^-_{in}$) we have introduced the quantity δ_i for representing the minimum increase of s that can be achieved by assigning x_i to a value in $\mathcal{V}$. The

smallest possible increase assuming that at least b variables must be assigned to a value in $\mathcal{V}$ is achieved by adding up the b smallest δ_i, which leads to the feasible lower bound $\ell = \sum_{i \in \mathcal{X}^+_{in} \cup \mathcal{X}^+_{out}} a_i \cdot \underline{x_i} + \sum_{i \in \mathcal{X}^-_{in} \cup \mathcal{X}^-_{out}} a_i \cdot \overline{x_i} + \Delta_b$. If ℓ exceeds c, the conjunction of the two constraints cannot be satisfied, otherwise we have an assignment where the lower bound is achieved. □

4 Computing the Regret of a Variable-Value Pair

Given a variable-value pair (x_i, u) $(0 \leq i < n, u \in \mathrm{dom}(x_i))$, the *regret* of this pair, denoted by $r(x_i, u)$, is defined as the increase of the sharp lower bound ℓ introduced in Condition 2 of Lemma 1 when x_i is assigned to u. It is equal to $+\infty$ when the assignment $x_i = u$ is not feasible subject to the `atleast` constraint.

Lemma 2. *The regret $r(x_i, u)$ assuming that variable x_i is assigned to u $(0 \leq i < n)$ is defined by a set of linear functions given by the following table:*

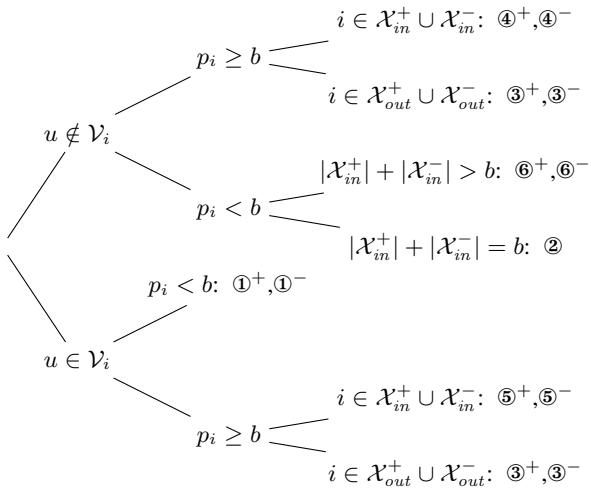
case	*condition*	*regret $r(x_i, u)$*
①$^+$	$u \in \mathcal{V}_i \wedge i \in \mathcal{X}^+_{in} \wedge p_i < b$	$a_i \cdot (u - \underline{v_i})$
①$^-$	$u \in \mathcal{V}_i \wedge i \in \mathcal{X}^-_{in} \wedge p_i < b$	$a_i \cdot (u - \overline{v_i})$
②	$u \notin \mathcal{V}_i \wedge p_i < b \wedge$ $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert = b$	$+\infty$
③$^+$	$i \in \mathcal{X}^+_{out}$	$a_i \cdot (u - \underline{x_i})$
③$^-$	$i \in \mathcal{X}^-_{out}$	$a_i \cdot (u - \overline{x_i})$
④$^+$	$u \notin \mathcal{V}_i \wedge i \in \mathcal{X}^+_{in} \wedge p_i \geq b$	$a_i \cdot (u - \underline{x_i})$
④$^-$	$u \notin \mathcal{V}_i \wedge i \in \mathcal{X}^-_{in} \wedge p_i \geq b$	$a_i \cdot (u - \overline{x_i})$
⑤$^+$	$u \in \mathcal{V}_i \wedge i \in \mathcal{X}^+_{in} \wedge p_i \geq b$	$a_i \cdot (u - \underline{x_i}) - \sigma_{b-1}$*
⑤$^-$	$u \in \mathcal{V}_i \wedge i \in \mathcal{X}^-_{in} \wedge p_i \geq b$	$a_i \cdot (u - \overline{x_i}) - \sigma_{b-1}$
⑥$^+$	$u \notin \mathcal{V}_i \wedge i \in \mathcal{X}^+_{in} \wedge p_i < b \wedge$ $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert > b$	$a_i \cdot (u - \underline{v_i}) + \sigma_b$
⑥$^-$	$u \notin \mathcal{V}_i \wedge i \in \mathcal{X}^-_{in} \wedge p_i < b \wedge$ $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert > b$	$a_i \cdot (u - \overline{v_i}) + \sigma_b$

* For convenience we assume that σ_{-1} is defined and equal to 0.

Proof. We first prove that the conditions attached to cases ①$^+$ to ⑥$^-$ are mutually exclusive and cover all cases. Finally we prove that the corresponding regrets are sharp.
[MUTUALLY EXCLUSIVE]: To prove that cases ①$^+$ to ⑥$^-$ are mutually exclusive we show that the cases ①$^+$, ②, ③$^+$, ④$^+$, ⑤$^+$ and ⑥$^+$ are mutually exclusive, the cases corresponding to negative a_i being similar. Finally its is clear that a case where a_i is positive or zero is not compatible with a case where a_i is negative. In the context of positive or zero a_i, the next table provides for each pair of cases (j, k) the two mutually exclusive subconditions $\frac{cond_j}{cond_k}$ respectively associated with cases j and k, where each subcondition is a subpart of the condition of its corresponding case.

	②	③$^+$	④$^+$	⑤$^+$	⑥$^+$
①$^+$	$\frac{u \in \mathcal{V}_i}{u \notin \mathcal{V}_i}$	$\frac{i \in \mathcal{X}^+_{in}}{i \in \mathcal{X}^+_{out}}$	$\frac{u \in \mathcal{V}_i}{u \notin \mathcal{V}_i}$	$\frac{p_i < b}{p_i \geq b}$	$\frac{u \in \mathcal{V}_i}{u \notin \mathcal{V}_i}$
②	-	$\frac{p_i < b}{i \in \mathcal{X}^+_{out}}$	$\frac{p_i < b}{p_i \geq b}$	$\frac{u \notin \mathcal{V}_i}{u \in \mathcal{V}_i}$	$\frac{\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert = b}{\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert > b}$
③$^+$	-	-	$\frac{i \in \mathcal{X}^+_{out}}{i \in \mathcal{X}^+_{in}}$	$\frac{i \in \mathcal{X}^+_{out}}{i \in \mathcal{X}^+_{in}}$	$\frac{i \in \mathcal{X}^+_{out}}{i \in \mathcal{X}^+_{in}}$
④$^+$	-	-	-	$\frac{u \notin \mathcal{V}_i}{u \in \mathcal{V}_i}$	$\frac{p_i \geq b}{p_i < b}$
⑤$^+$	-	-	-	-	$\frac{u \in \mathcal{V}_i}{u \notin \mathcal{V}_i}$

[COVER ALL CASES]: Given the elementary subconditions $u \in \mathcal{V}_i$, $u \notin \mathcal{V}_i$, $p_i < b$, $p_i \geq b$, $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert = b$, $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert > b$, $i \in \mathcal{X}^+_{in} \cup \mathcal{X}^-_{in}$ and $i \in \mathcal{X}^+_{out} \cup \mathcal{X}^-_{out}$ found in the conditions describing cases ①$^+$ to ⑥$^-$,[2] the next tree shows how all possible combinations of elementary conditions are covered, i.e. for every node of the tree, the disjunction of the elementary conditions attached to all its children corresponds to true.



[SHARPNESS]: W.l.o.g. we omit cases ①$^-$, ③$^-$, ④$^-$, ⑤$^-$ and ⑥$^-$, which are respectively similar to cases ①$^+$, ③$^+$, ④$^+$, ⑤$^+$ and ⑥$^+$. We successively consider each remaining case.

- In case ①$^+$, x_i corresponds to a variable with a positive or zero coefficient a_i that was assigned to $\underline{v_i}$ in the assignment attached to ℓ. Since this variable remains assigned to a value u in $\mathcal{V}_i$ this does not affect the number of variables assigned to values from $\mathcal{V}_i$ in the lower bound ℓ. Consequently the regret is equal to $\delta_i = a_i \cdot (u - \underline{v_i})$.
- In case ②, x_i corresponds to a variable that must be assigned to a value in $\mathcal{V}_i$ in order to satisfy the `atleast` constraint. Since we cannot reassign it to any value u outside $\mathcal{V}_i$ the regret is equal to $+\infty$.
- In case ③$^+$, x_i corresponds to a variable that cannot be assigned to a value in $\mathcal{V}_i$ and that has a positive or zero coefficient a_i. Such a variable was assigned to its minimum value in the assignment attached to ℓ. If we reassign it to a new value u this does not affect the number of variables assigned to values from $\mathcal{V}_i$ in the lower bound ℓ. Consequently the regret is equal to $\delta_i = a_i \cdot (u - \underline{x_i})$.
- In case ④$^+$, x_i corresponds to a variable that was not assigned to a value belonging to $\mathcal{V}_i$ in the assignment attached to ℓ (even if it could have been) and that has a positive or zero coefficient a_i. Such a variable was assigned to its minimum value in the lower bound ℓ. If we reassign it to a new value u that also does not belong to $\mathcal{V}_i$, this does not affect the number of variables assigned to values

[2] W.l.o.g. we group $i \in \mathcal{X}^+_{in}$ and $i \in \mathcal{X}^-_{in}$ together, and do the same for $i \in \mathcal{X}^+_{out}$ and $i \in \mathcal{X}^-_{out}$.

from $\mathcal{V}_i$ in the lower bound ℓ. Consequently the regret is equal to $\delta_i = a_i \cdot (u - \underline{x_i})$.

- In case ⑤$^+$, x_i corresponds to a variable that was not assigned to a value belonging to $\mathcal{V}_i$ in the assignment attached to ℓ (even if it could have been) and that has a positive or zero coefficient a_i. Such a variable was assigned to its minimum value in the lower bound ℓ. Now if we reassign it to a value from $\mathcal{V}_i$, this increases by one the number of variables assigned to values from $\mathcal{V}_i$ in the lower bound ℓ. Consequently the regret is equal to the increase $\delta_i = a_i \cdot (u - \underline{x_i})$ minus σ_{b-1}. The last term σ_{b-1} comes from the fact that we can reset the variable corresponding to the b smallest δ to its minimum or maximum value depending on the sign of its coefficient.
- In case ⑥$^+$, x_i corresponds to a variable with a positive or zero coefficient a_i that was assigned to $\underline{v_i}$ in the assignment attached to ℓ. Now if we reassign it to a value u that does not belong to $\mathcal{V}_i$, this decreases by one the number of variables assigned to values from $\mathcal{V}_i$ in the lower bound ℓ. Consequently the regret is equal to the increase $a_i \cdot (u - \underline{v_i})$ due to the fact that we switch x_i from $\underline{v_i}$ to u plus the increase σ_b due to the fact that we have to assign one extra variable to a value from $\mathcal{V}_i$. The term σ_b comes from the fact that we select the variable corresponding to the $b+1$ smallest δ in order to minimize the new lower bound.

□

We note $r_k(x_i, u)$ $k \in \{①^+,①^-,②,③^+,③^-,④^+,④^-,⑤^+,⑤^-,⑥^+,⑥^-\}$ the regret associated with case k.

5 Filtering Algorithm

This section provides a filtering algorithm (see Algorithms 1 and 2) that is directly based on the lower bound introduced in Lemma 1 and on the regret introduced in Lemma 2. To filter $\mathrm{dom}(x_i)$ $(0 \leq i < n)$, we first need to characterize how the cases ①$^+$ to ⑥$^-$ introduced in Lemma 2 are structured with respect to an interval of values $[\underline{x_i}, \overline{x_i}]$.

Lemma 3.
Given an interval of values $[\underline{x_i}, \overline{x_i}]$, cases ①$^+$ to ⑥$^-$ can only follow one of the following four mutually exclusive patterns shown by the following table plus four symmetrical exclusive patterns where $^+$ is replaced by $^-$:

pattern	*condition*	*sequence of cases*
$\mathbf{a}^+$	$i \in \mathcal{X}^+_{out}$	③$^+$ over $\underline{x_i}$ … $\overline{x_i}$
$\mathbf{b}^+$	$i \in \mathcal{X}^+_{in} \wedge$ $p_i \geq b$	④$^+$⑤$^+$④$^+$ … ⑤$^+$④$^+$ over $\underline{x_i}$ … $\overline{x_i}$
$\mathbf{c}^+$	$i \in \mathcal{X}^+_{in} \wedge$ $p_i < b \wedge$ $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert > b$	⑥$^+$①$^+$⑥$^+$ … ①$^+$⑥$^+$ over $\underline{x_i}$ … $\overline{x_i}$
$\mathbf{d}^+$	$i \in \mathcal{X}^+_{in} \wedge$ $p_i < b \wedge$ $\lvert\mathcal{X}^+_{in}\rvert + \lvert\mathcal{X}^-_{in}\rvert = b$	② ①$^+$② … ①$^+$② over $\underline{x_i}$ … $\overline{x_i}$

Proof. The patterns are obtained by first removing the sub-conditions $u \in \mathcal{V}_i$, $u \notin \mathcal{V}_i$ from each condition attached to cases ①$^+$ to ⑥$^-$ and by grouping together the remaining compatible conditions. The patterns are mutually exclusive since their conditions are mutually incompatible. □

Patterns $\mathbf{b}^+$, $\mathbf{c}^+$, $\mathbf{d}^+$ consist of alternating cases switching from condition $u \notin \mathcal{V}_i$ to condition $u \in \mathcal{V}_i$ back and forth. The filtering algorithm consists of the following steps:

- Fail if $|\mathcal{X}^+_{in}| + |\mathcal{X}^-_{in}| < b$.
- Fail if $\ell = \sum_{i \in \mathcal{X}^+_{in} \cup \mathcal{X}^+_{out}} a_i \cdot \underline{x_i} + \sum_{i \in \mathcal{X}^-_{in} \cup \mathcal{X}^-_{out}} a_i \cdot \overline{x_i} + \Delta_b > c$.
- Prune each variable x_i $(0 \leq i < n)$ such that $\underline{x_i} \neq \overline{x_i}$ by considering the patterns $\mathbf{a}^+$, $\mathbf{b}^+$, $\mathbf{c}^+$ and $\mathbf{d}^+$ introduced by the previous lemma, the other patterns $\mathbf{a}^-$, $\mathbf{b}^-$, $\mathbf{c}^-$ and $\mathbf{d}^-$ being similar even if $a_i \neq 0$ in these later patterns:

$\mathbf{a}^+$. In the context of pattern a$^+$ we are in case ③$^+$. Therefore we remove all values u such that $\ell + r_{3^+}(x_i, u) > c$.

$\mathbf{b}^+$. In the context of pattern b$^+$ we are in cases ④$^+$ and ⑤$^+$. Therefore we remove:
- all values $u \notin \mathcal{V}$ such that $\ell + r_{4^+}(x_i, u) > c$,
- all values $u \in \mathcal{V}$ such that $\ell + r_{5^+}(x_i, u) > c$.

$\mathbf{c}^+$. In the context of pattern c$^+$ we are in cases ⑥$^+$ and ①$^+$. Therefore we remove:
- all values $u \notin \mathcal{V}$ such that $\ell + r_{6^+}(x_i, u) > c$,
- all values $u \in \mathcal{V}$ such that $\ell + r_{1^+}(x_i, u) > c$.

$\mathbf{d}^+$. In the context of pattern d$^+$ we are in cases ② and ①$^+$. Since, in case ② the regret $r_2(x_i, u)$ is equal to $+\infty$, we remove:
- all values $u \notin \mathcal{V}$,
- all values $u \in \mathcal{V}$ such that $\ell + r_{1^+}(x_i, u) > c$.

This leads to Algorithm 2 with the following result.

Theorem 1. *Given the two constraints* $\texttt{atleast}(b, \langle x_0, x_1, \ldots, x_{n-1}\rangle, \mathcal{V})$ *and* $\sum_{i=0}^{n-1} a_i \cdot x_i \leq c$, *Algorithm 2 called with* $\mathcal{V}$ *and* $\mathrm{dom}(x_i)$ $(0 \leq i < n)$ *achieves GAC with a worst case time complexity of* $O(n \log n + n \cdot (r + s + t))$ *where* $O(r)$, $O(s)$, $O(t)$ *and* $O(1)$ *are the respective worst case time complexity for (1) checking whether a domain variable intersects* $\mathcal{V}$, *(2) computing the minimum or maximum value of the intersection between a domain variable and* $\mathcal{V}$, *(3) removing all values of the complement of* $\mathcal{V}$ *wrt. an interval from a domain variable, (4) adjusting the minimum or maximum value of a domain variable.*

Proof. (sketch) GAC stems from the fact that lines 2 to 19 implement the necessary and sufficient condition introduced by Lemma 1 and from the fact that lines 20 to 46 use the sharp regret introduced by Lemma 2 to filter the domains of the variables. The worst case time complexity $O(n \log n + n \cdot (r + s + t))$ is made up from: (a) $O(nr)$

the n tests at line 5 of Algorithm 2 for checking whether a domain intersects or not $\mathcal{V}$, (b) $O(ns)$ the $2 \cdot n$ computations at lines 6 and 7 for extracting the minimum and maximum value of the intersection between the domain of a variable and $\mathcal{V}$, (c) $O(n \log n)$ the sort at line 21, (d) $O(nt)$ the at most n calls to `remove_set` in the filtering part (lines 25 to 46). □

6 Related Work

We can use the `cost_gcc` constraint (Régin 2002) to model a conjunction of linear inequality and `atleast` constraint and get GAC. For this purpose, we create a cost matrix indexed by the variables and the values to be assigned. We then set each row associated with a variable to the product of the coefficient of the variable in the linear equality with the corresponding value. Corollary 1 on page 13 of (Régin 2002) gives a complexity that depends on the number of arcs in the flow graph (i.e. the sum of the domain sizes) and that assumes all domain operations to be constant time. The space complexity of using `cost_gcc` is $O(nd)$, for n variables and d values, since we need a cost matrix. Our space complexity is $O(n)$. Section 8 provides an empirical comparison between our propagator and `cost_gcc`.

7 Application to Learning Simple Polynomials

The ModelSeeker (Beldiceanu and Simonis 2012) generates elements of a model from sample solutions. Each element consists of a partition description, which describes systematic subsets of the decision variables, and a global constraint, which is applied to each subset of the variables. For every problem size, different partition generator arguments and derived arguments of the global constraint may be used. Our motivation for the present work is to find simple polynomial functions of maximal degree p, which describe all parameters in terms of some basic parameters, e.g. problem size. While many problems (like n-queens) can be described by a single problem parameter, others will require multiple, independent parameters (BIBDs for example use up to five parameters). Clearly, a solution with few parameters is preferable to a solution with many parameters. Identifying the parameters and finding the right simple polynomials relating the parameters is a key subproblem, especially when you have few examples.

We cannot use standard curve fitting techniques, as we search for "nice" polynomials, with few nonzero, small integer coefficients. Alternatives like the "Method of Differences" (Langley et al. 1987) can find integer solutions, but require $n+2$ samples to determine (arbitrary) coefficients of a polynomial in one variable of order n. It seems difficult to extend the method to search only for polynomials with few non-zero coefficients.

Given parameters I, J and a set of samples K, we assume known parameter values x_{ik} required to explain observed parameters values y_{jk} via a polynomial of M product terms

$$\forall_{k \in K} \forall_{j \in J} : \quad q_j y_{jk} = \sum_{m \in M} c_{mj} \prod_{i \in I} x_{ik}^{m_{mi}} \qquad (1)$$

with unknown, integer values q_j and c_{mj}, and $0 \leq \sum_{i \in I} m_{mi} \leq p$ for all $m \in M$. Note that the product term evaluates to an integer, i.e. the right-hand side is a linear function in variables c_{mj}. Also note that, if we wish, we can restrict a priori the form of the polynomials considered by only collecting some product terms, i.e. considering only simple products of parameters, but not higher exponents. The quotient variable q_j gives us more flexibility for the function we search for, we assume that $\gcd(q_j, c_{mj}) = 1$ for at least one m.

Example 2. *Consider the magic hexagon problem (CSPlib (Gent and Walsh 1999) problem 23; see Fig. 1) where, given solutions for the magic hexagon of order $k \in K = \{3, 4, 5, 6, 7\}$, we want to relate (1) the order x_{1k} of the hexagon, (2) the smallest value x_{2k} used to fill the hexagon and (3) the total sum y_{1k} over the full hexagon. From the parameters $x_1 = [3, 4, 5, 6, 7]$, $x_2 = [1, 3, 6, 21, 2]$ and $y_1 = [190, 777, 2196, 6006, 8255]$ we obtain the simple polynomial $2y_{1k} = 9x_{1k}^4 + 6x_{1k}^2 x_{2k} - 18x_{1k}^3 - 6x_{1k}x_{2k} + 12x_{1k}^2 + 2x_{2k} - 3x_{1k}$, which describes the expected result for all $k \in K$. The data for this problem are taken from Wikipedia (http://en.wikipedia.org/wiki/Magic_hexagon).*

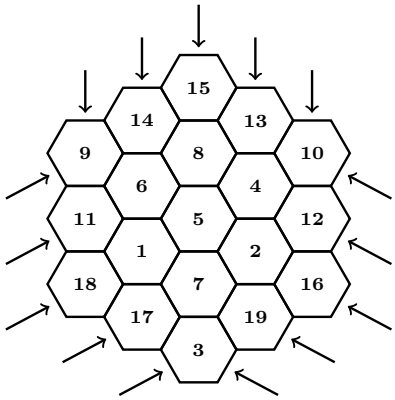


Figure 1: A magic hexagon of order 3, filled by integers 1 through 19. The sum of the integers in each row of cells, in all three directions, is 38. The sum of all integers is 190.

We now provide the simple polynomials learned for a number of classical CSP problems from the parameters mostly taken from the companion report to (Beldiceanu and Simonis 2012) (http://4c.ucc.ie/~hsimonis/modelling/report.pdf). By simple we mean minimizing the following list of criteria in lexicographic order: (1) the number of parameters (one or two), (2) the maximum degree of the polynomials, (3) the number of nonzero coefficients, (4) the absolute value of the largest coefficient, (5) the sum of the absolute value of the coefficients.[3] We provide the interpretation of the parameters that are directly related to the problem and not of the

[3] In the case when only one or two sizes are available the criteria, are reordered by (1), (3), (2), (4) and (5) in order to prevent the discovery of linear expression with big coefficients rather than nonlinear expressions with very small coefficients. This sometimes allows the expected polynomial to be found even if we have a single size; see example magic cube

parameters that are related to generated constraints, which is outside the scope of this paper.

amazon (like n-queen but the queen may also move as a knight):

input $x_0 = [10, 12], x_1 = [9, 11], x_2 = [5, 6], x_3 = [4, 5]$.

output $x_0 = 2x_2, x_1 = 2x_2 - 1, x_3 = x_2 - 1$.

interpretation x_0 is the sample size, x_1 is a constant used in a *smooth* (Beldiceanu, Carlsson, and Rampon 2012) constraint between consecutive columns, x_2 is a constant used in a *smooth* constraint between columns two apart.

bibd (balanced incomplete block designs):

input $x_0 = [49, 60, 112], x_1 = [7, 10, 14], x_2 = [7, 6, 8]$.

output $x_0 = x_2x_1$.

interpretation x_0 is the size of the sample, x_1 is the number of columns, x_2 is the number of rows.

bundesliga (http://www.weltfussball.de/alle_spiele/bundesliga-2010-2011/):

input $x_0 = [612], x_1 = [34], x_2 = [18], x_3 = [17]$.

output $x_0 = 18x_1, x_2 = 18, 2x_3 = x_1$.

interpretation x_0 is the size of the sample, x_1 is the number of days in a complete season, x_2 is the number of teams, and x_3 the length of a half season.

coinsgrid (http://www.svor.ch/competitions/competition2007/AsroContestSolution.pdf):

input $x_0 = [100, 121, 441, 625], x_1 = [10, 11, 21, 25], x_2 = [6, 6, 11, 13], x_3 = [4, 5, 10, 12]$.

output $x_0 = x_1^2, x_2 = x_1 - x_3$.

interpretation x_0 is the size of the sample, x_1 is the size of the squared board, x_2 is the number of zeros in one column, x_3 is the number of ones in one column.

efpa (http://www-circa.mcs.st-and.ac.uk/Preprints/freqpermarrays.pdf) :

input $x_0 = [56, 72, 90], x_1 = [8, 9, 10], x_2 = [7, 8, 9], x_3 = [4, 6, 8], x_4 = [1, 3, 4]$.

output $x_0 = -2x_4 + 10x_3 + 18$, $2x_1 = x_3 + 12$, $2x_2 = x_3 + 10$.

interpretation x_0 is the size of the sample, x_1 is the number of columns of the rectangular board, x_2 is the number of rows of the rectangular board, x_3 tells for any pair of columns from how many positions they should differ at least, x_4 is the maximum value used among a set of consecutive values.

franklin (http://mathworld.wolfram.com/FranklinMagicSquare.html and (van Delft and Botermans 1990, page 95)):

input $x_0 = [130, 1028], x_1 = [64, 256], x_2 = [16, 32], x_3 = [8, 16], x_4 = [4, 8], x_5 = [2, 4]$.

output $4x_0 = x_3^3 + x_3$, $2x_1 = x_3$, $x_2 = 2x_3$, $2x_4 = x_3$, $4x_5 = x_3$.

interpretation x_0 is the sum along a half-row/column of the squared board, x_1 is the number of cells of the squared board, x_2 is the number of cells in one quarter of the squared board, x_3 is the size of the squared board.

kirkman :

input $x_0 = [105, 147, 351], x_1 = [15, 21, 27], x_2 = [7, 7, 13]$.

output $x_0 = x_2x_1$.

interpretation x_0 is the size of a sample, x_1 is the number of schoolgirls, x_2 is the number of days.

magic cube :

input $x_0 = [42], x_1 = [27], x_2 = [3]$.

output $2x_0 = x_2^4 + x_2, x_1 = x_2^3$.

interpretation x_0 is the sum along a direction of the cube, x_1 is the number of cells of the cube, x_2 is the size of the cube.

magic hexagon :

input $x_0 = [190, 777, 2196, 6006, 8255], x_1 = [3, 4, 5, 6, 7], x_2 = [1, 3, 6, 21, 2]$.

output $2x_0 = 9x_1^4+6x_1^2x_2-18x_1^3-6x_1x_2+12x_1^2+2x_2-3x_1$.

interpretation x_0 is the overall sum over the full hexagon, x_1 is the size of the hexagon, x_2 is the smallest value among the consecutive values used in the hexagon.

magic square :

input $x_0 = [34, 870, 7825]$, $x_1 = [16, 144, 625]$, $x_2 = [4, 12, 25]$.

output $2x_0 = x_2^3 + x_2, x_1 = x_2^2$.

interpretation x_0 is the sum along a column, row or diagonal of the square, x_1 is the size of the sample, x_2 is the size of the square.

queen :

input $x_0 = [7, 8, 9], x_1 = [6, 7, 8]$.

output $x_0 = x_1 + 1$.

interpretation x_0 is the size of the squared board and x_1 is a parameter of the *smooth* constraint on adjacent columns.

samuraï (Dürr 2011):

input $x_0 = [4, 9], x_1 = [2, 3]$.

output $x_0 = x_1^2$.

interpretation x_0 is the size of the sample and x_1 is the size of the squared board.

sudoku :

input $x_0 = [81, 256, 625], x_1 = [9, 16, 25], x_2 = [3, 4, 5]$.

output $x_0 = x_2^4, x_1 = x_2^2$.

interpretation x_0 is the total number of cells, x_1 is the size of the big square, x_2 is the size of the small square blocks.

sudoku(1) :

input $x_0 = [144, 400, 900], x_1 = [12, 20, 30], x_2 = [4, 5, 6], x_3 = [3, 4, 5]$.

output $x_0 = x_3^4 + 2x_3^3 + x_3^2, x_1 = x_3^2 + x_3, x_2 = x_3 + 1$.

interpretation x_0 is the total number of cells, x_1 is the size of the big square, x_2 and x_3 are the width and height of the small rectangular blocks. Width and height are linked.

sudoku(2) :

input $x_0 = [144, 400, 1296, 3969]$, $x_1 = [12, 20, 36, 63]$, $x_2 = [4, 5, 6, 9], x_3 = [3, 4, 6, 7]$.

output $x_0 = x_2^2x_3^2, x_1 = x_2x_3$.

interpretation x_0 is the total number of cells, x_1 is the size of the big square, x_2 and x_3 are the width and height of the small rectangular blocks. Width and height are not linked.

8 Experiments

To test the efficiency of our algorithm, we performed some randomized tests on a model based on our application problem. The benchmarks were run on a quad core 2.2 GHz Intel Core i7 MacBookPro machine with 16Gb of memory running MacOS 10.7.5 (using only one processor core), with `linear_atleast` implemented as an addition to SICStus Prolog (Carlsson and et al. 2012). We compare four implementations, System DECOMP, a model with separate linear

and `atleast` constraints, System GLOBAL, using several `linear_atleast` constraints, System CGCC, stating each equality as a single `cost_gcc` constraint, and System MIP, the straightforward 0/1 MIP model of the problem solved with CPLEX 12.4.

We impose two variants of the `atleast` constraint. The first enforces that there be at most four nonzero coefficients, the second states that there is a single "large" (absolute value greater than 5) coefficient. All c_{mj} are between -25 and 25, q_j is 1, 2 or 3. These values match the domain range encountered in our application. We use six samples (x values), which for our cases is sufficient to make the solutions unique. For feasible problems, we randomly generate polynomials with 4 nonzero coefficients, for infeasible problems we use polynomials with 5 nonzero coefficients, and evaluate them for our samples, generating the observations y_{jk}.

In our tests, we search for polynomials with two parameters, varying the maximal degree between 3 and 9. We perform 100 runs for each set, and report the number of variables, the minimum, average, maximum run time (in ms), the standard deviation, and the ratio of average times compared to our new version. For the finite domain models, we use a static variable order by decreasing order of the exponents, and a static (increasing) value order in our search routine to allow a fair comparison of the systems. The MIP model uses the default CPLEX branching method. We do not show results if the longest run needed more than 1000 seconds.

Table 1 shows results for the feasible problems, Table 2 shows results for infeasible problems. The infeasible results are of particular interest: In our application we solve a problem repeatedly with different choices of independent parameters, increasing domain sizes and increasing maximum degree of the polynomial. We therefore often encounter several infeasible problems, before a feasible solution is found.

System	Deg	Vars	Min	Avg	Max	StdDev	Ratio
DECOMP	3	10	0	717	2590	613.88	7.46
GLOBAL	3	10	0	96	340	69.47	1.00
CGCC	3	10	3360	17770	38210	8364.26	185.10
MIP	3	511	9	164	2210	247.70	1.71
DECOMP	4	15	150	8592	26760	7154.63	28.26
GLOBAL	4	15	10	304	930	208.51	1.00
CGCC	4	15	13790	78003	164860	36820.77	256.59
MIP	4	766	22	1218	5891	1410.07	4.01
DECOMP	5	21	1300	58864	153670	42253.77	62.42
GLOBAL	5	21	30	943	2980	722.07	1.00
CGCC	5	21	39190	193683	554560	98938.79	205.39
MIP	5	1072	20	6712	40804	7095.47	7.12
DECOMP	6	28	5560	267582	658470	153104.90	97.87
GLOBAL	6	28	80	2734	7410	1839.16	1.00
MIP	6	1429	122	19549	92924	19015.32	7.15
GLOBAL	7	36	240	5349	15330	3646.60	1.00
MIP	7	1837	561	40280	173112	36476.62	7.53
GLOBAL	8	45	430	7042	19780	4779.71	1.00
MIP	8	2296	370	75005	344072	73742.91	10.65
GLOBAL	9	55	300	18160	43610	8627.43	1.00
MIP	9	2806	2809	196379	880427	160229.00	10.81

Table 1: Results Feasible Problems

System	Deg	Vars	Min	Avg	Max	StdDev	Ratio
DECOMP	3	10	50	2449	9550	2555.77	23.10
GLOBAL	3	10	0	106	420	94.68	1.00
CGCC	3	10	11450	74542	168120	39675.52	703.23
MIP	3	511	8	142	747	136.33	1.34
DECOMP	4	15	820	33045	103880	30271.70	74.93
GLOBAL	4	15	10	441	1940	378.28	1.00
CGCC	4	15	43630	278255	641410	143783.80	630.96
MIP	4	766	26	1517	9349	1968.44	3.44
DECOMP	5	21	3840	199777	570410	162089.1	121.15
GLOBAL	5	21	10	1649	6110	1410.89	1.00
MIP	5	1072	36	7486	27581	6827.52	4.54
GLOBAL	6	28	70	4754	17620	3927.10	1.00
MIP	6	1429	150	20729	77729	18028.52	4.36
GLOBAL	7	36	210	12391	41690	11231.11	1.00
MIP	7	1837	1260	36516	171297	44041.36	2.95
GLOBAL	8	45	210	28214	79860	22136.83	1.00
MIP	8	2296	686	129338	636411	112111.90	4.58
GLOBAL	9	55	810	54200	167370	37568.83	1.00
MIP	9	2806	20369	279949	900200	200853.50	5.16

Table 2: Results Infeasible Problems

The results show a clear improvement of our new constraint over the other finite domain versions. Even though the `cost_gcc` constraint achieves the same consistency, Model CGCC is not competitive, as the overhead of the flow model and cost matrix are too high. The decomposition in Model DECOMP performs better, but still becomes too slow for larger problem sizes. Our model GLOBAL also clearly outperforms Model MIP, even though we are using a default, static variable and value ordering for the finite domain solver. In practical terms, the new version, Model GLOBAL, is powerful enough to handle all problem sizes we are interested in, in a few seconds.

System	Deg	Domain	Min	Avg	Max	StdDev	Ratio
DECOMP	3	-10..10	20	194	420	81.36	7.19
GLOBAL	3	-10..10	0	27	60	14.98	1.00
CGCC	3	-10..10	120	711	1800	349.17	26.33
DECOMP	3	-15..15	0	259	870	299.23	7.85
GLOBAL	3	-15..15	0	33	100	19.53	1.00
CGCC	3	-15..15	530	2896	5990	1245.95	87.76
DECOMP	3	-20..20	10	410	1550	363.34	10.25
GLOBAL	3	-20..20	0	40	110	26.28	1.00
CGCC	3	-20..20	2090	8303	18620	3494.18	207.58
DECOMP	3	-25..25	20	683	2200	600.83	17.08
GLOBAL	3	-25..25	0	40	110	29.29	1.00
CGCC	3	-25..25	3360	17770	38210	8364.26	444.25

Table 3: Results for Increasing Domain Sizes

To confirm our hypothesis that the behavior of the `cost_gcc` model is influenced by the domain size, we performed an additional experiment. Table 3 shows the run times when, for the model with maximal degree 3 (10 variables), we vary the domain sizes between -10..10 and -25..25, but keep all other constraints, i.e. we allow a single large coefficient for the polynomial with the increased range. The decomposition Model DECOMP shows a moderate increase in run times, our new Model GLOBAL with the

1: **procedure** prune($var, \mathcal{V}, limit, a, b, \delta$)
2: **if** $a > 0$ **then**
3: $u \leftarrow \lfloor \frac{limit-b}{a} \rfloor$ // last feasible out-value
4: $u' \leftarrow \lfloor \frac{limit-b+\delta}{a} \rfloor$ // last feasible value
5: remove_set($x[i], [u+1, +\infty] \setminus \mathcal{V}$)
6: adjust_max($x[i], u'$)
7: **else if** $a < 0$ **then**
8: $u \leftarrow \lceil \frac{limit-b}{a} \rceil$ // first feasible out-value
9: $u' \leftarrow \lceil \frac{limit-b+\delta}{a} \rceil$ // first feasible value
10: remove_set($x[i], [-\infty, u-1] \setminus \mathcal{V}$)
11: adjust_min($x[i], u'$)
12: **else if** $b > limit$ **then**
13: remove_set($x[i], \setminus \mathcal{V}$)

Algorithm 1: First remove from variable var all values u such that $u \notin \mathcal{V} \wedge\ a \cdot u + b > limit$. Then remove from variable var all values u such that $a \cdot u + b - \delta > limit$. Called by main algorithm, Algorithm 2, in a context where it can never fail since lines 19 and 23 of Algorithm 2 check the necessary and sufficient condition introduced by Lemma 1 for having at least one solution. adjust_min, adjust_max and remove_set respectively adjust the minimum value of a variable, adjust the maximum value of a variable, and remove a set of values from a variable.

linear_atleast constraints shows nearly no increase at all, while the times for Model CGCC with a single cost_gcc constraint for each equation increase dramatically. This increase is due to the larger flow model and cost matrix that must be considered when the domain sizes increase.

9 Conclusion

The notion of regret and its use in the context of cost-based filtering was originally introduced for dealing with constraints having a cost variable (Focacci, Lodi, and Milano 1999) and used more extensively later on, e.g. (Focacci, Lodi, and Milano 2002; Sellmann, Gellermann, and Wright 2007; Kovács and Beck 2011). This paper shows how to use this notion of regret for providing a GAC filtering algorithm for a conjunction of a linear inequality and an atleast constraint. Experiment shows that this stronger filtering has a key impact on finding simple polynomials or for proving that no solution with a given structure exists, and that it scales much better than using a reformulation or using the cost_gcc constraints, which provide the same filtering. The model also out-performs a 0/1 MIP model, even without a dynamic search routine.

Acknowledgments

We thank the anonymous reviewers for their detailed feedback, which helped improve the paper.

1: **function** filter($b, \mathcal{V}, c, n, a_{[0..n-1]}, x_{[0..n-1]}$) : boolean
2: // 1. computing the lower bound ℓ
3: $in \leftarrow 0$; $\ell \leftarrow 0$;
4: **for** $i = 0$ to $n-1$ **do**
5: **if** $\mathrm{dom}(x[i]) \cap \mathcal{V} \neq \emptyset$ **then**
6: $\underline{v[i]} \leftarrow \min(\mathrm{dom}(x[i]) \cap \mathcal{V})$
7: $\overline{v[i]} \leftarrow \max(\mathrm{dom}(x[i]) \cap \mathcal{V})$
8: $(\mathcal{X}^+_{in}[i], \mathcal{X}^-_{in}[i]) \leftarrow (a[i] \geq 0, a[i] < 0)$
9: $(\mathcal{X}^+_{out}[i], \mathcal{X}^-_{out}[i]) \leftarrow (\textbf{false}, \textbf{false})$
10: **if** $a[i] \geq 0$ **then**
11: $\sigma[in] \leftarrow a[i] \cdot (\underline{v[i]} - \underline{x[i]})$
12: **else**
13: $\sigma[in] \leftarrow a[i] \cdot (\overline{v[i]} - \overline{x[i]})$
14: $index[in] \leftarrow i$; $in \leftarrow in + 1$;
15: **else**
16: $(\mathcal{X}^+_{out}[i], \mathcal{X}^-_{out}[i]) \leftarrow (a[i] \geq 0, a[i] < 0)$
17: $(\mathcal{X}^+_{in}[i], \mathcal{X}^-_{in}[i]) \leftarrow (\textbf{false}, \textbf{false})$
18: $\ell \leftarrow \ell + a[i] \cdot (\textbf{if } a[i] \geq 0 \textbf{ then } \underline{x[i]} \textbf{ else } \overline{x[i]})$
19: **if** $in < b$ **then return false**
20: // 2. filtering $x_{[0..n-1]}$ wrt. the regret
21: sort the pairs $(\sigma[i], index[i])$ $(0 \leq i < in)$ increasing
22: **for** $i = 0$ to $b-1$ **do** $\ell \leftarrow \ell + \sigma[i]$
23: **if** $\ell > c$ **then return false**
24: **for** $i = 0$ to $in - 1$ **do** $p[index[i]] \leftarrow i$
25: **for** $i = 0$ to $n-1$ **do**
26: **if** $\underline{x[i]} \neq \overline{x[i]}$ **then**
27: **if** $\mathcal{X}^+_{out}[i]$ **then**
28: **if** $a[i] > 0$ **then**
29: adjust_max($x[i], \lfloor \frac{c + a[i] \cdot \underline{x[i]} - \ell}{a[i]} \rfloor$)
30: **else if** $\mathcal{X}^+_{in}[i] \wedge p[i] \geq b$ **then**
31: prune($x[i], \mathcal{V}, c, a[i], \ell - a[i] \cdot \underline{x[i]}, \sigma[b-1]$)
32: **else if** $\mathcal{X}^+_{in}[i] \wedge p[i] < b \wedge in > b$ **then**
33: prune($x[i], \mathcal{V}, c, a[i], \ell + \sigma[b] - a[i] \cdot \underline{v[i]}, \sigma[b]$)
34: **else if** $\mathcal{X}^+_{in}[i]$ **then**
35: remove_set($x[i], \setminus \mathcal{V}$)
36: **if** $a[i] > 0$ **then**
37: adjust_max($x[i], \lfloor \frac{c + a[i] \cdot \underline{v[i]} - \ell}{a[i]} \rfloor$)
38: **else if** $\mathcal{X}^-_{out}[i]$ **then**
39: adjust_min($x[i], \lceil \frac{c + a[i] \cdot \overline{x[i]} - \ell}{a[i]} \rceil$)
40: **else if** $\mathcal{X}^-_{in}[i] \wedge p[i] \geq b$ **then**
41: prune($x[i], \mathcal{V}, c, a[i], \ell - a[i] \cdot \overline{x[i]}, \sigma[b-1]$)
42: **else if** $\mathcal{X}^-_{in}[i] \wedge p[i] < b \wedge in > b$ **then**
43: prune($x[i], \mathcal{V}, c, a[i], \ell + \sigma[b] - a[i] \cdot \overline{v[i]}, \sigma[b]$)
44: **else**
45: remove_set($x[i], \setminus \mathcal{V}$)
46: adjust_min($x[i], \lceil \frac{c + a[i] \cdot \overline{v[i]} - \ell}{a[i]} \rceil$)
47: **return** true

Algorithm 2: GAC algorithm for the conjunction atleast($b, \langle x_0, x_1, \ldots, x_{n-1} \rangle, \mathcal{V}$) and $\sum_{i=0}^{n-1} a_i \cdot x_i \leq c$. Patterns $\mathbf{a}^+, \mathbf{b}^+, \mathbf{c}^+, \mathbf{d}^+, \mathbf{a}^-, \mathbf{b}^-, \mathbf{c}^-, \mathbf{d}^-$ correspond resp. to line 29, 31, 33, 35..37, 39, 41, 43, 45..46.

References

Beldiceanu, N., and Simonis, H. 2012. A model seeker: Extracting global constraint models from positive examples. In *Principles and Practice of Constraint Programming (CP 2012)*, volume 7514 of *LNCS*, 141–157. Springer.

Beldiceanu, N.; Carlsson, M.; Petit, T.; and Régin, J.-C. 2012. An $O(n \log n)$ bound consistency algorithm for the conjunction of an *alldifferent* and an inequality between a sum of variables and a constant, and its generalization. In Raedt, L. D.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *20th European Conference on Artificial Intelligence (ECAI'12)*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 145–150. Montpellier, France: IOS Press.

Beldiceanu, N.; Carlsson, M.; and Rampon, J.-X. 2012. Global constraint catalog, 2nd edition (revision a). Technical Report T2012:03, Swedish Institute of Computer Science.

Bessière, C.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2011. The *alldifferent* constraint with precedences. In Achterberg, T., and Beck, J. C., eds., *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'11)*, volume 6697 of *LNCS*, 36–52. Springer.

Bessière, C. 2006. Constraint propagation. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*. Elsevier. chapter 3.

Carlsson, M., and et al. 2012. *SICStus Prolog User's Manual*. SICS, 4.2.1 edition.

Dürr, C. 2011. n samuraïs. Published electronically at http://www.enseignement.polytechnique.fr/informatique/INF580/exams/examen_11.pdf. Taken from 2011 constraint exam at Polytechnique, Palaiseau, France.

Focacci, F.; Lodi, A.; and Milano, M. 1999. Cost-based domain filtering. In Jaffar, J., ed., *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, 189–203. Springer.

Focacci, F.; Lodi, A.; and Milano, M. 2002. Optimization-oriented global constraints. *Constraints* 7(3-4):351–365.

Gent, I. P., and Walsh, T. 1999. CSPlib: A benchmark library for constraints. In Jaffar, J., ed., *Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, 480–481. Springer.

Kovács, A., and Beck, J. C. 2011. A global constraint for total weighted completion time for unary resources. *Constraints* 16(1):100–123.

Langley, P.; Simon, H. A.; Bradshaw, G. L.; and Zytkow, J. M. 1987. *Scientific Discovery*. Cambridge, MA.: MIT Press.

Petit, T.; Régin, J.-C.; and Beldiceanu, N. 2011. A $\theta(n)$ bound-consistency algorithm for the *increasing sum* constraint. In Lee, J. H., ed., *Principles and Practice of Constraint Programming (CP 2011)*, volume 6876 of *LNCS*, 721–728. Springer.

Puget, J.-F. 2004. Improved bound computation in presence of several clique constraints. In Wallace, M., ed., *Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*, 527–541. Springer.

Régin, J.-C., and Rueher, M. 2000. A global constraint combining a sum constraint and difference constraints. In Dechter, R., ed., *Principles and Practice of Constraint Programming (CP'2000)*, volume 1894 of *LNCS*, 384–395. Springer.

Régin, J.-C. 2002. Cost-based arc consistency for global cardinality constraints. *Constraints* 7(3–4):387–405.

Sellmann, M.; Gellermann, T.; and Wright, R. 2007. Cost-based filtering for shorter path constraints. *Constraints* 12(2):207–238.

van Delft, P., and Botermans, J. 1990. *Denk Spiele der Welt*. Hugendubel.

Zhang, Y., and Yap, R. H. C. 2000. Arc consistency on n-ary monotonic and linear constraints. In Dechter, R., ed., *Principles and Practice of Constraint Programming (CP 2000)*, volume 1894 of *LNCS*, 470–483. Springer.