A Parametric Propagator for Discretely Convex Pairs of Sum Constraints*

Jean-Noël Monette[1], Nicolas Beldiceanu[2], Pierre Flener[1], and Justin Pearson[1]

[1] Uppsala University, Dept. of Information Technology, 751 05 Uppsala, Sweden
FirstName.LastName@it.uu.se

[2] Mines de Nantes, TASC Team (CNRS/INRIA), 44307 Nantes, France
Nicolas.Beldiceanu@Mines-Nantes.fr

Abstract. We introduce a propagator for abstract pairs of Sum constraints, where the expressions in the sums respect a form of convexity. This propagator is parametric and can be instantiated for various concrete pairs, including Deviation, Spread, and the conjunction of Sum and Count. We show that despite its generality, our propagator is competitive in theory and practice with state-of-the-art propagators.

1 Introduction

Many constraint problems involve a Sum constraint, along with other constraints. It is however well-known that a Sum constraint taken in isolation is not able to perform a lot of pruning since the estimation of the minimum or maximum of a sum does not take other constraints into account. Several authors have studied how to include other constraints (sharing some variables) in the propagator for Sum, either in particular cases (e.g., Spread [9], IncreasingSum [11], and Sum with cliques [12]), or in general (e.g., ObjectiveSum [15]).

In the present work, we focus on a parametric problem, which can be cast as

$$\sum_{i\in[1,n]} f_i(x_i) \leq \overline{f} \tag{1}$$

$$\underline{g} \leq \sum_{i\in[1,n]} g_i(x_i) \leq \overline{g} \tag{2}$$

for any $n \geq 1$. The f_i and g_i are functions from integers to integers and the f_i (resp. g_i) can differ for each i. In this work, $\overline{f}$, $\underline{g}$, and $\overline{g}$ are constants, but Section 5 shows how to use variables instead. In Section 5, we also consider a lower bound $\underline{f}$ on the first sum.

Finding a solution to the conjunction of (1) and (2) is in general NP-complete as it includes as a special case the knapsack problem. There is however a large class of f_i and g_i functions for which either domain consistency or bounds($\mathbb{Z}$) consistency (see, e.g., [19] for definitions) can be achieved in polynomial time.

* This work is supported by grants 2011-6133 and 2012-4908 of the Swedish Research Council (VR). We thank the reviewers for their constructive comments.

C. Schulte (Ed.): CP 2013, LNCS 8124, pp. 529–544, 2013.

In this paper, we present a *parametric propagator* for this class of functions and show how to instantiate it for various functions f_i and g_i. We show that the considered class of problems includes among others the (bounds($\mathbb{Z}$) consistent) constraints DEVIATION [17], SPREAD [9], and WEIGHTEDAVERAGE [3] (with variable weights and constant values) and the (domain consistent) conjunction of LINEAR and COUNT [13]. In several cases, we match the theoretical complexity and practical efficiency of previously published specialised propagators.

Our approach for propagating the conjunction of (1) and (2) contains two parts. First (as discussed in Section 2), we compute a sharp lower bound on $\sum_{i\in[1,n]} f_i(x_i)$ under constraint (2), together with a witnessing assignment. The conjunction is feasible if this lower bound, which we call the *feasibility bound*, is at most $\overline{f}$. To compute this feasibility bound, we introduce new functions derived from the f_i and g_i. We show that the feasibility bound can be greedily computed if the newly introduced functions are *discretely convex*.

In the second part of the propagator (discussed in Section 3), the domain of each variable x_j is filtered by computing for each value u in its domain a sharp lower bound on $\sum_{i\in[1,n]} f_i(x_i)$ under constraint (2) when x_j is assigned u. If this lower bound is larger than $\overline{f}$, then u is removed from the domain of x_j. The lower bound for each pair (x_j, u) is computed *incrementally* from the witnessing assignment for the feasibility bound thanks to the discrete convexity property. We also present an improved propagator for an additional property of f_j and g_j.

The resulting propagator is parametric, depending on the f_i and g_i. The time complexity and the achieved level of consistency depend on the shape of the f_i and g_i and on the values given to the parameters. We study the complexity in Section 4 and give some implementation notes. Afterwards, we present in Section 5 several instantiations of the propagator, including a case study of DEVIATION. Finally, Section 6 presents some experimental results showing that the genericity of our approach is not detrimental to performance.

2 Feasibility Test

Given a variable x, let D_x denote the current domain of that variable. For a function f and value v, we write $f^{-1}(v)$ for the *set* of values $\{u \mid f(u) = v\}$. For a function f and set S, we write $f(S)$ for $\{f(u) \mid u \in S\}$. We use x_i, v_i, f_i to represent single variables, values, and functions, while $\mathbf{x}$, $\mathbf{v}$, $\mathbf{f}$ represent the respective vectors of all variables, values, and functions (e.g., $\mathbf{x} = \langle x_1, x_2, \ldots, x_n\rangle$).

The conjunction of (1) and (2) is satisfiable if and only if the *cost* (i.e., the value of the objective function) of an optimal solution to the following problem is at most $\overline{f}$:

$$\begin{aligned} \text{minimise} \quad & \sum_{i\in[1,n]} f_i(x_i) \\ \text{such that} \quad & \underline{g} \leq \sum_{i\in[1,n]} g_i(x_i) \leq \overline{g} \\ & x_i \in \mathrm{D}_{x_i}, \quad \forall i \in [1,n] \end{aligned} \tag{3}$$

We gradually show in the next sub-sections how to compute greedily this cost, called the *feasibility bound*, together with a witnessing assignment.

2.1 Problem Reformulation

We reformulate problem (3) in two steps. The first step introduces for each i a new function h_i that captures the relation between f_i and g_i. The second step splits the resulting reformulated problem into two subproblems.

First Step. After introducing new variables y_i, so that $y_i = g_i(x_i)$ for each i, we propose the following new problem:

$$\begin{aligned} \text{minimise} \quad & \sum_{i \in [1,n]} h_i(y_i) \\ \text{such that} \quad & \underline{g} \leq \sum_{i \in [1,n]} y_i \leq \overline{g} \\ & y_i \in g_i(\mathrm{D}_{x_i}), \quad \forall i \in [1,n] \end{aligned} \tag{4}$$

where we introduce a new function $h_i \colon g_i(\mathrm{D}_{x_i}) \to f_i(\mathrm{D}_{x_i})$ for each i. This function is defined as $h_i(v) = \min f_i(g_i^{-1}(v)) = \min\{f_i(u) \mid u \in \mathrm{D}_{x_i} \wedge g_i(u) = v\}$, that is $h_i(v)$ is the smallest value of $f_i(x_i)$ that can be attained when $g_i(x_i)$ is equal to v. Note that the definition of h_i depends on the current domain of x_i. We now prove that the feasibility bound can also be computed from problem (4).

Lemma 1. *All optimal solutions to problems (3) and (4) have the same cost.*

Proof. Let $\mathbf{v}$ denote a vector of values for the vector $\mathbf{y}$ of variables. For each value v_i, we choose an arbitrary value u_i in D_{x_i} such that $g_i(u_i) = v_i$ and $f_i(u_i) = h_i(v_i)$. Such a value u_i always exists, by the definition of h_i. Then the vector $\mathbf{u}$ is a feasible solution to problem (3) if and only if $\mathbf{v}$ is a feasible solution to problem (4), and they have the same cost. In addition, any other assignment $\mathbf{u}'$ such that $g_i(u'_i) = v_i$ for each i has a cost larger than or equal to the cost of $\mathbf{u}$ and $\mathbf{v}$, by the definition of h_i. Hence $\mathbf{u}$ is optimal if and only if $\mathbf{v}$ is optimal. $\square$

Second Step. We define a new function, called H, from integers to integers:

$$H(b) = \min \left\{ \sum_{i \in [1,n]} h_i(y_i) \;\middle|\; \sum_{i \in [1,n]} y_i = b \wedge \forall i \in [1,n] : y_i \in g_i(\mathrm{D}_{x_i}) \right\} \tag{5}$$

That is, $H(b)$ is the minimum of the sum of the $h_i(y_i)$ when the sum of the y_i is equal to b. For a given b, we define $\mathbf{w}^b$ to be an assignment of $\mathbf{y}$ such that $b = \sum_{i \in [1,n]} w_i^b$ and $H(b) = \sum_{i \in [1,n]} h_i(w_i^b)$, i.e., an optimal solution to (5). We call $\mathbf{w}^b$ a *witnessing assignment* of b. We propose the following new problem:

$$\begin{aligned} \text{minimise} \quad & H(z) \\ \text{such that} \quad & \underline{g} \leq z \leq \overline{g} \end{aligned} \tag{6}$$

Table 1. Several instantiations of f_i and g_i, and the corresponding h_i. The notation [*cond*] uses the Iverson bracket and is defined to be 1 if *cond* is true, and 0 otherwise.

Common Name	$f_i(u)$	$g_i(u)$	$h_i(v)$
LINEAR	$a_i \cdot u$	0	$\begin{cases} a_i \cdot \min \mathrm{D}_{x_i} & \text{if } a_i > 0 \\ a_i \cdot \max \mathrm{D}_{x_i} & \text{if } a_i \leq 0 \end{cases}$
WEIGHTEDAVERAGE [3]	$a_i \cdot u$	u	$a_i \cdot v$
DEVIATION [17]	$\lvert n \cdot u - n \cdot \mu\rvert$	u	$\lvert n \cdot v - n \cdot \mu\rvert$
SPREAD [9]	$(n \cdot u - n \cdot \mu)^2$	u	$(n \cdot v - n \cdot \mu)^2$
L_p-NORM, $0 < p < +\infty$	$\lvert n \cdot u - n \cdot \mu\rvert^p$	u	$\lvert n \cdot v - n \cdot \mu\rvert^p$
LINEAR and COUNT [13]	$a_i \cdot u$	$[u \in \mathcal{V}]$	$\begin{cases} a_i \cdot \min(\mathrm{D}_{x_i} \setminus \mathcal{V}) & \text{if } v = 0 \wedge a_i > 0 \\ a_i \cdot \max(\mathrm{D}_{x_i} \setminus \mathcal{V}) & \text{if } v = 0 \wedge a_i \leq 0 \\ a_i \cdot \min(\mathrm{D}_{x_i} \cap \mathcal{V}) & \text{if } v = 1 \wedge a_i > 0 \\ a_i \cdot \max(\mathrm{D}_{x_i} \cap \mathcal{V}) & \text{if } v = 1 \wedge a_i \leq 0 \end{cases}$
LINEAR and MAXIMUM	$a_i \cdot u$	$[u \geq m]$	(omitted, similar to previous pair)
MODANDDIV ($a_i > 0$)	$u - a_i \cdot \lfloor u/a_i \rfloor$	$\lfloor u/a_i \rfloor$	$\max(0, \min \mathrm{D}_{x_i} - a_i \cdot v)$

where z is a fresh variable. The feasibility bound can also be computed from problem (6), as the latter has the same optimal cost as problem (4), and thus as problem (3): this is shown by replacing $H(z)$ by its definition (5) in the formulation of problem (6). Problems (4) and (6) are more interesting than problem (3) in three respects. First, it is simpler to reason with only one function per variable (namely h_i) instead of two (namely f_i and g_i). Second, the domain D_{y_i}, which is equal to $g_i(\mathrm{D}_{x_i})$, might be much smaller than D_{x_i}. Third, introducing H allows us to compute the feasibility bound in two steps: (i) construct H from the h_i, and (ii) find an optimal solution to (6). This can be done greedily if all h_i are discretely convex.

Definition 1. *A function $f\colon A \to B$, where $A, B \subseteq \mathbb{Z}$, is* discretely convex *if*
1. *A is an interval, and*
2. *$\forall v \in A\colon (v-1) \in A \wedge (v+1) \in A \Rightarrow 2 \cdot f(v) \leq f(v-1) + f(v+1)$.*

The notion of discrete convexity is an adaptation of the usual convexity from the reals to the integers. This notion has been studied in depth, for instance in [7]. It is also related to the notion of submodular functions on sets [4].

The first condition in Definition 1 restricts in some cases the application of our approach to domains with no holes. This is discussed further in Section 5.1.

Table 1 presents the f_i, g_i, and h_i for several pairs. The h_i are convex for all those examples. Before providing algorithms, we need to introduce some notions.

2.2 Deltas, Segments, Slopes, Breakpoints, Reasoning on Infinity

Let $f\colon A \to B$ be a function with $A, B \subseteq \mathbb{Z}$. Given some value v in A, we call *right delta* (resp. *left delta*) the increase of f when v increases (resp. decreases) by 1. Formally: $\Delta^+(f, v) = f(v+1) - f(v)$ and $\Delta^-(f, v) = f(v-1) - f(v)$; the value of $\Delta^+(f, v)$ (resp. $\Delta^-(f, v)$) is $+\infty$ when $v+1$ (resp. $v-1$) is not in A.

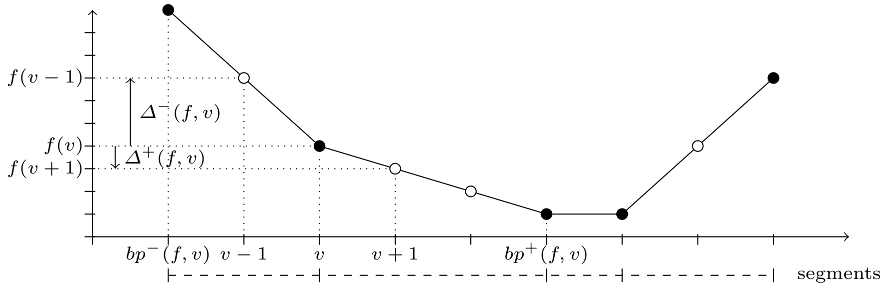

Fig. 1. Illustration of the notions of Section 2.2. Filled points are at breakpoints.

A *segment* of f is a maximal interval $[\ell, u]$ of its domain where the (right or left) delta is constant. Formally: $\Delta^+(f, v) = \Delta^+(f, v+1)$ for all $v \in [\ell, u-1]$, with $\ell \leq u$, $\Delta^+(f, \ell-1) \neq \Delta^+(f, \ell)$, and $\Delta^+(f, u-1) \neq \Delta^+(f, u)$. The endpoints ℓ and u of a segment $[\ell, u]$ of f are called *breakpoints* of f. The *length* of a segment $[\ell, u]$ is $u - \ell$. The *slope* of a segment $[\ell, u]$ is $\Delta^+(f, \ell)$. Hence the slope of a function is constant inside any of its segments and changes at its breakpoints.

The domain of f can be uniquely partitioned into its segments, and each value of the domain belongs to one or two segments. For a value v, the *breakpoint on the right* of v, denoted by $\mathrm{bp}^+(f, v)$, is u if v is in some segment $[\ell, u]$ with $u \neq v$, and otherwise undefined, denoted by $+\infty$. Similarly, $\mathrm{bp}^-(f, v)$ denotes the *breakpoint on the left* of v, if any, otherwise $-\infty$.

Let f be a discretely convex function. For any two contiguous segments, the slope of the former is smaller than the slope of the latter, hence no two segments have the same slope. Also, $\Delta^+(f, v) = +\infty$ only for the largest value v in A, as A is an interval, and $\Delta^-(f, v) = +\infty$ only for the smallest value v in A.

Figure 1 illustrates these notions on a discretely convex function.

The basic properties of $+\infty$ and $-\infty$ used in our algorithms are, for any $v \in \mathbb{Z}$: $-\infty < v < +\infty$, $v + (+\infty) = +\infty$, $v + (-\infty) = -\infty$, $v - (-\infty) = +\infty$, $v - (+\infty) = -\infty$, $\min(v, +\infty) = v$, and $v/+\infty = 0$.

2.3 Characterisation of the *H* Function

When the h_i are discretely convex, problem (6) is easy to solve by greedy search, because H is then also discretely convex and can be calculated efficiently.

Before proving those claims, we need to study the relationship between $H(b)$, $H(b+1)$, and $H(b-1)$, and their respective witnessing assignments. For any j and $k \neq j$, the sum $w_1^b + \cdots + (w_j^b + 1) + \cdots + (w_k^b - 1) + \cdots + w_n^b$ equals b, and hence by definition of H (since w_i^b are the values that minimise $H(b)$), we have $H(b) \leq h_1(w_1^b) + \cdots + h_j(w_j^b + 1) + \cdots + h_k(w_k^b - 1) + \cdots + h_n(w_n^b)$.

Rearranging and cancelling out common terms gives

$$h_k(w_k^b) - h_k(w_k^b - 1) \leq h_j(w_j^b + 1) - h_j(w_j^b) \tag{7}$$

If h_j is discretely convex, then we have that $h_k(w_k^b) - h_k(w_k^b - 1) \leq h_j(w_j^b + 1) - h_j(w_j^b) \leq h_j(w_j^b + 2) - h_j(w_j^b + 1)$. Thus $h_1(w_1^b) + \cdots + h_j(w_j^b + 1) + \cdots + h_k(w_k^b) + \cdots + h_n(w_n^b) \leq h_1(w_1^b) + \cdots + h_j(w_j^b + 2) + \cdots + h_k(w_k^b - 1) + \cdots + h_n(w_n^b)$, so that adding two to any single w_j^b and reducing another w_k^b by one to arrive at the sum $b+1$ will have a higher cost than simply adding one to a single w_j^b. Because each h_i is discretely convex, this is true for any increment larger than one. Hence it is possible to find a witnessing assignment $\mathbf{w}^{b+1}$ for $b+1$ from a witnessing assignment $\mathbf{w}^b$ for b by increasing any suitable w_i^b by one. Similarly it is possible to find a $\mathbf{w}^{b-1}$ by subtracting one from any suitable w_i^b.

Lemma 2. *H is discretely convex whenever each h_i is discretely convex.*

Proof. The domain of each h_i is an interval $[\ell_i, u_i]$, so that the domain of H is the interval $\left[\sum_{i\in[1,n]} \ell_i, \sum_{i\in[1,n]} u_i\right]$. We need to show that $H(b) - H(b-1) \leq H(b+1) - H(b)$. If w_i^b is a witnessing assignment for some b then by the discussion above there are some k and j such that $H(b-1) = h_1(w_1^b) + \cdots + h_k(w_k^b - 1) + \cdots + h_n(w_n^b)$ and $H(b+1) = h_1(w_1^b) + \cdots + h_j(w_j^b + 1) + \cdots + h_n(w_n^b)$. Therefore $H(b) - H(b-1) = h_k(w_k^b) - h_k(w_k^b - 1)$ and $H(b+1) - H(b) = h_j(w_j^b + 1) - h_j(w_j^b)$ and by (7) $H(b) - H(b-1) \leq H(b+1) - H(b)$. Hence H is discretely convex. □

We now show how to calculate H efficiently by giving a characterisation of its minimum and segments. Here, for any set S and function f, the expression $\operatorname{argmin}_{i\in S} f(i)$ returns *one* (arbitrary) value $i \in S$ that minimises $f(i)$.

Lemma 3. *A witnessing assignment $\mathbf{w}^{b^*}$ of a value b^* that minimises H is such that $w_i^{b^*} = \operatorname{argmin}_{v_i \in g_i(\mathrm{D}_{x_i})} h_i(v_i)$.*

Proof. If $\mathbf{w}^{b^*}$ is a witnessing assignment of b^*, then b^* is equal to $\sum_{i\in[1,n]} w_i^{b^*}$ and $H(b^*) = \sum_{i\in[1,n]} h_i(w_i^{b^*})$. Since each $w_i^{b^*} = \operatorname{argmin}_{y_i \in g_i(\mathrm{D}_{x_i})} h_i(y_i)$ corresponds to the minimum value obtainable by h_i, it is not possible to reduce the value $\sum_{i\in[1,n]} h_i(w_i^{b^*})$ by picking a different value for any $w_i^{b^*}$. □

There exist potentially several $\mathbf{w}^{b^*}$ that minimise H. The correctness of our approach does not depend on a particular choice for those values.

We now characterise the segments of H.

Lemma 4. *If $\mathbf{w}^b$ is a witnessing assignment for b, then $\Delta^+(h_i, w_i^b) \geq \Delta^+(H, b)$ and $\Delta^-(h_i, w_i^b) \geq \Delta^-(H, b)$ for all $i \in [1, n]$.*

Proof. If b is increased by one, then one of the w_i^b must be increased by one as discussed previously. To reach the minimum value for $b+1$, one needs to increase the value of a variable y_k that has the smallest $\Delta^+(h_k, w_k^b)$. So the increase of H, namely $\Delta^+(H, b)$, is equal to $\Delta^+(h_k, w_k^b)$, which is smaller than or equal to $\Delta^+(h_i, w_i^b)$ for any other i. A similar argument is used for a decrease of b. □

Lemma 5. *The length of each segment of H is equal to the sum of the lengths of the segments in the h_i functions with the same slope.*

Proof. As in the proof of Lemma 4, $\Delta^+(H,b)$ is equal to a minimal $\Delta^+(h_k, w_k^b)$. If one wants to increase b by more than one, the increase per unit stays constant as long as there is at least one variable with slope equal to $\Delta^+(H,b)$. This defines a segment of slope $\Delta^+(H,b)$, whose length is equal to the sum of the lengths of the segments of all h_i functions with the same slope. □

We can use Lemmas 3 and 5 to construct H efficiently. Section 4 presents two ways to implement this construction in practice.

2.4 Computing the Feasibility Bound and a Witnessing Assignment

We can now show a case when problem (6) can be solved in a greedy way.

Theorem 1. *Problem (6) can be solved greedily if each h_i is discretely convex.*

Proof. If each function h_i is discretely convex, then the function H is also discretely convex (by Lemma 2) and can be constructed from the h_i (by Lemmas 3 and 5). Finding the minimum of a discretely convex function under some bound constraints can be done greedily, as a local minimum of a discretely convex function is also a global minimum (see, e.g., Theorem 2.2 in [7]). □

Given the function H, problem (6) can be solved by first finding b^* minimising H, and then greedily increasing or decreasing b^* if b^* is not in $[\underline{g}, \overline{g}]$. In addition, it is useful for the filtering to compute the witnessing assignment $\mathbf{w}^{b^*}$ of b^*.

Thanks to Lemma 4, this can be achieved as in Algorithm 1. From now on, we simply write $\mathbf{w}$ to refer to $\mathbf{w}^{b^*}$. An assignment $\mathbf{w}$ that minimises the value of H without considering the bounds of b is initially constructed (lines 2–4). If b is in $[\underline{g}, \overline{g}]$, then the initial assignment is the final one. Otherwise the assignment is iteratively modified in order to satisfy the bounds of b. We assume $b < \underline{g}$ in line 5 (the case $b > \overline{g}$ is symmetrical and not shown). Then some w_i must be increased until b is equal to $\underline{g}$. This is done in two steps. In lines 6–10, the segment of H where $\underline{g}$ lies is found. Its slope is stored in $\Delta^{\max}$, and the distance between $\mathrm{bp}^-(H, \underline{g})$ and $\underline{g}$ is stored in *slack*. Those two values allow us then to modify each w_i separately (lines 11–17). For each i, first w_i is moved from breakpoint to breakpoint of h_i while the slope of the segment is smaller than $\Delta^{\max}$. Next, if the slope of the segment on the right of w_i is equal to $\Delta^{\max}$, then w_i is moved further on this segment, without exceeding the remaining slack (line 15).

The algorithm returns the witnessing assignment $\mathbf{w}$ (line 20), or "null" if the constraint is unsatisfiable (line 8), which triggers propagator failure and happens if there exists no value in the domains of the h_i such that $b \in [\underline{g}, \overline{g}]$.

3 Domain Filtering

To filter the domain of a variable, we extend the reasoning presented in Section 2.1. Indeed, variable x_j can take the value u if the cost of an optimal solution to the following problem is smaller than or equal to $\overline{f}$:

Algorithm 1. Greedy algorithm to compute a witnessing assignment

function GetWitnessLowerBound($\mathbf{h}, H, \underline{g}, \overline{g}$)
  for all $i \in [1, n]$ **do**
    $w_i := \operatorname{argmin}_{v \in g_i(\mathrm{D}_{x_i})} h_i(v)$
  $b := \sum_{i \in [1,n]} w_i$
initial bound
  if $b < \underline{g}$ **then**
    while $\Delta^+(H, b) < +\infty$ **and** $\mathrm{bp}^+(H, b) < \underline{g}$ **do**
      $b := \mathrm{bp}^+(H, b)$
    if $\Delta^+(H, b) = +\infty \wedge b < \underline{g}$ **then return** null
    $\Delta^{\max} := \Delta^+(H, b)$
    $slack := \underline{g} - b$
sharp bound
    for all $i \in [1, n]$ **do**
      while $\Delta^+(h_i, w_i) < \Delta^{\max}$ **do**
        $w_i := \mathrm{bp}^+(h_i, w_i)$
      if $\Delta^+(h_i, w_i) = \Delta^{\max}$ **and** $slack > 0$ **then**
        $w' := \min\left(\mathrm{bp}^+(h_i, w_i),\ w_i + slack\right)$
        $slack := slack - w_i + w'$
        $w_i := w'$
modifying $\mathbf{w}$
  else if $b > \overline{g}$ **then**
    [analogous algorithm]
  return w

$$
\begin{aligned}
\text{minimise} \quad & f_j(u) + \sum_{i \neq j \in [1,n]} f_i(x_i) \\
\text{such that} \quad & \underline{g} \leq g_j(u) + \sum_{i \neq j \in [1,n]} g_i(x_i) \leq \overline{g} \\
& x_i \in \mathrm{D}_{x_i}, \quad \forall i \neq j \in [1, n]
\end{aligned} \tag{8}
$$

Problem (8) resembles problem (3) but x_j is fixed to u. Hence we can use the same reformulation as in Section 2.1. We introduce the following new function:

$$
H_j(b) = \min \left\{ \sum_{i \neq j \in [1,n]} h_i(y_i) \;\middle|\; \sum_{i \neq j \in [1,n]} y_i = b \wedge \forall i \neq j \in [1, n] : y_i \in g_i(\mathrm{D}_{x_i}) \right\}
$$

That is, $H_j(b)$ is similar to $H(b)$ in (5) but it only uses the functions h_i for i different from j. The optimal cost of problem (8) is the optimal cost of the following new problem:

$$
\begin{aligned}
\text{minimise} \quad & f_j(u) + H_j(z) \\
\text{such that} \quad & \underline{g} \leq g_j(u) + z \leq \overline{g}
\end{aligned} \tag{9}
$$

where value u is given and z is the only variable. The result of the following lemma can be used to compute H_j.

Lemma 6. *The function H_j is discretely convex if all h_i are convex. The value b^*_j that minimises H_j is equal to the value b^* that minimises H minus the value v^* that minimises h_j. The length of each segment of H_j is equal to the length of the linear segment of H of the same slope minus the length of the linear segment of h_j of the same slope (if any).*

The proof (omitted for space reasons) of this lemma uses similar arguments to the ones of Lemmas 2 to 5. We show hereafter two ways to use H_j to filter the domains. The first way is applicable in general (provided H_j is discretely convex). The second way makes use of an additional property of f_j and g_j.

3.1 Filtering in the General Case

As several values u of x_j can have the same image v through g_j, the set of values in D_{x_j} that are consistent with constraints (1) and (2) can be partitioned as:

$$\bigcup_{v \in g_j(\mathrm{D}_{x_j})} \left\{ u \;\middle|\; g_j(u) = v \wedge f_j(u) \leq \overline{f} - \min_{\underline{g} \leq z+v \leq \overline{g}} H_j(z) \right\}$$

That is, for each v, we have the set of values u in $g_j^{-1}(v)$ such that the optimal cost of problem (9) is no larger than $\overline{f}$, hence which are consistent. The domain of x_j can be made domain consistent by filtering the following unary constraint for each value $v \in g_j(\mathrm{D}_{x_j})$:

$$g_j(x_j) = v \Rightarrow f_j(x_j) \leq \overline{f} - \min_{\underline{g} \leq z+v \leq \overline{g}} H_j(z) \tag{10}$$

The function H_j being discretely convex, one can compute $\min_{\underline{g} \leq z+v \leq \overline{g}} H_j(z)$ (which is independent from a particular u) incrementally from a value v to $v+1$. In addition, if v is equal to w_j, the value of y_j in the witnessing assignment $\mathbf{w}$ computed in Section 2.4, then $H_j(\sum_{i \neq j \in [1,n]} w_i) + h_j(w_j) = H(\sum_{i \in [1,n]} w_i)$. This leads to Algorithm 2, which is used to filter the domain of x_j for the values v larger than w_j. This algorithm traverses h_j and H_j. The only complication is that in some cases (captured by the Boolean variable dec_b defined in lines 6 and 11) reaching an optimal solution to $\min_{\underline{g} \leq z+v \leq \overline{g}} H_j(z)$ involves decrementing b, which is the current value of z (line 9). Domain filtering according to constraint (10) takes place in lines 5 and 10. The algorithm ends when the optimal cost of problem (9) for $v+1$ is larger than $\overline{f}$ (line 7). A complementary algorithm is used for the values smaller than w_j. Algorithm 2 achieves domain consistency provided the h_i are discretely convex. Section 5.1 discusses more precisely the link between the shape of the h_i and the consistency level.

3.2 Filtering in a Special Case

We now present a special case to avoid useless computation. Let us define $k_j(v) = \max f_j(g_j^{-1}(v))$, that is $k_j(v)$ is the largest value $f_j(u)$ for u such that $g_j(u) = v$. The function k_j is similar to h_j but the 'max' operator replaces the 'min' one.

Algorithm 2. Filtering algorithm for values larger than w_j (general case)

function ForwardFilter($j, \mathbf{h}, \mathbf{w}, H, \overline{f}$)
  $H_j := \text{computeHj}(H, h_j)$
  $b := \sum_{i\in[1,n]} w_i - w_j$
  $v := w_j$
  Filter($g_j(x_j) = v \Rightarrow f_j(x_j) \leq \overline{f} - H_j(b)$)
  $dec_b := b + v \geq \overline{g} \vee \Delta^-(H_j, b) < 0$
  while $H_j(b) + h_j(v) + ($**if** dec_b **then** $\Delta^-(H_j, b)$ **else** $0) + \Delta^+(h_j, v) \leq \overline{f}$ **do**
    $v := v + 1$
    if dec_b **then** $b := b - 1$
    Filter($g_j(x_j) = v \Rightarrow f_j(x_j) \leq \overline{f} - H_j(b)$)
    $dec_b := b + v \geq \overline{g} \vee \Delta^-(H_j, b) < 0$
  Filter($g_j(x_j) \leq v$)

If $h_j(v) \geq k_j(v-1)$ for any value v larger than $v^* = \operatorname{argmin}_{u\in g_j(\mathrm{D}_{x_j})} h_j(u)$ and $h_j(v) \geq k_j(v+1)$ for any v smaller than v^*, then there exists a value $v^{\max}$ such that for all values $v \in g_j(\mathrm{D}_{x_j})$ smaller than $v^{\max}$ (but larger than or equal to w_j), all values $u \in g_j^{-1}(v)$ are consistent, and for all v larger than $v^{\max}$, there is no consistent u. We then need not consider all values but only find $v^{\max}$ and filter according to the two constraints $g_j(x_j) \leq v^{\max}$ and $g_j(x_j) = v^{\max} \Rightarrow f_j(x_j) \leq \overline{f} - \min_{\underline{g}\leq z+v^{\max}\leq\overline{g}} H_j(z)$. A similar argument holds for a $v^{\min}$.

Finding $v^{\max}$ amounts to computing the largest value v such that $h_j(v) + \min_{\underline{g}\leq z+v\leq\overline{g}} H_j(z) \leq \overline{f}$. As h_j and H_j are both convex, this problem can be solved by incrementally increasing v until the bound is reached. Algorithm 3 presents the steps to find $v^{\max}$. This algorithm is very similar to Algorithm 2, but it does not need to iterate over all the values v, only over the ones that are at a breakpoint of h_j or H_j. The increment is stored in ℓ (lines 6, 11, and 12).

An example of the special case is when g_j is the identity function. Then g_j is injective. Hence $h_j = k_j$ and, by convexity, h_j is non-decreasing right of v^* and non-increasing left of v^*.

4 A Parametric Propagator and Its Complexity

Our propagator is generic in the sense that it works correctly for any functions f_i and g_i that respect the condition of Theorem 1. However, we call it a *parametric* propagator, because rather than resorting to a fully generic implementation, we use hook functions and procedures that need to be provided. This allows us to get a lower time complexity. The parameters to provide for an instantiation are shown in Table 2: they are used in Algorithms 1 to 3. We now study the time and space complexity of our propagator, based on a few implementation notes.

Feasibility Test. We implement the H function as a linked list of segments, plus two integers for the values b^* and $H(b^*)$. The value of $H(b)$ is never queried for arbitrary values of b, but only for b^* and for incrementally modified values of b,

Algorithm 3. Filtering algorithm for values larger than w_j (special case)

1: **function** FORWARDFILTER($j, \mathbf{h}, \mathbf{w}, H, \overline{f}$)
2: $\quad H_j :=$ COMPUTEHJ(H, h_j)
3: $\quad b := \sum_{i \in [1,n]} w_i - w_j$
4: $\quad v := w_j$
5: $\quad dec_b := b + v \geq \overline{g} \vee \Delta^-(H_j, b) < 0$
6: $\quad \ell := \min \left\{ b - \mathrm{bp}^-(H_j, b),\ \mathrm{bp}^+(h_j, v) - v,\ \textbf{if } dec_b \textbf{ then } +\infty \textbf{ else } \overline{g} - b - v \right\}$
7: $\quad$ **while** $H_j(b) + h_j(v) + \ell \cdot ((\textbf{if } dec_b \textbf{ then } \Delta^-(H_j, b) \textbf{ else } 0) + \Delta^+(h_j, v)) \leq \overline{f}$ **do**
8: $\qquad v := v + \ell$
9: $\qquad$ **if** dec_b **then** $b := b - \ell$
10: $\qquad dec_b := b + v \geq \overline{g} \vee \Delta^-(H_j, b) < 0$
11: $\qquad \ell := \min\left\{ b - \mathrm{bp}^-(H_j, b),\ \mathrm{bp}^+(h_j, v) - v,\ \textbf{if } dec_b \textbf{ then } +\infty \textbf{ else } \overline{g} - b - v \right\}$
12: $\quad \ell := (\overline{f} - H_j(b) - h_j(v)) / (\Delta^+(h_j, v) + (\textbf{if } dec_b \textbf{ then } \Delta^-(H_j, b) \textbf{ else } 0))$
13: $\quad v := v + \ell$
14: $\quad$ FILTER$(g_j(x_j) \leq v)$
15: $\quad$ FILTER$(g_j(x_j) = v \Rightarrow f_j(x_j) \leq \overline{f} - H_j(b))$

Table 2. Parameters to instantiate

Functions	Procedures
$\mathrm{argmin}_{v \in g_i(\mathrm{D}_{x_i})} h_i(v)$	FILTER$(g_i(x_i) \leq v)$
$\Delta^+(h_i, v)$	FILTER$(g_i(x_i) \geq v)$
$\Delta^-(h_i, v)$	FILTER$(g_i(x_i) = v \Rightarrow f_i(x_i) \leq u)$
$\mathrm{bp}^+(h_i, v)$	
$\mathrm{bp}^-(h_i, v)$	

so that $H(b)$ can also be computed incrementally. This is also true for h_i, and is reflected by the absence of $h_i(u)$ from the parameters in Table 2. Using that linked list and some bookkeeping, the computation of $H(b)$, $\Delta^+(H, b)$, $\Delta^-(H, b)$, $\mathrm{bp}^+(H, b)$, and $\mathrm{bp}^-(H, b)$ can be performed in constant time for all values of b used in the algorithms.

Constructing the linked list of H can be done in various ways. A first way is to traverse each function h_i in turn and to build H incrementally by traversing the linked list in parallel. This takes $\mathcal{O}(n \cdot (s(h) \cdot p + s(H)))$ time, where $s(h)$ is the maximum number of segments among the h_i functions, $s(H)$ is the number of segments of H, and p is the highest complexity of the parametric functions. A second way is to collect all the segments from all the functions in a list, to sort this list, and to construct H by traversing the list. This takes $\mathcal{O}(n \cdot s(h) \cdot (p + \log(n \cdot s(h))))$ time and is asymptotically better than the first way when $s(H) > s(h) \cdot \log(n \cdot s(h))$.

Algorithm 1 computes a witnessing assignment in $\mathcal{O}(s(H) + n \cdot s(h))$ time. This is dominated by the prior construction of H, as $s(H) \leq n \cdot s(h)$.

Filtering. We implement Algorithm 2 to run in $\mathcal{O}(r(h) \cdot c)$ time, where $r(h) = \left| g_j(\mathrm{D}_{x_j}) \right|$ and c is the highest complexity of the procedures in Table 2. The segments of H_j are computed on the fly from h_j and H. The sum in line 3 of

Table 3. Time complexity of the different versions of the propagator

Propagator	Time complexity
Traversing, general case	$\mathcal{O}(n \cdot (s(h) \cdot p + s(H) + r(h) \cdot c)$
Sorting, general case	$\mathcal{O}(n \cdot (s(h) \cdot p + s(h) \cdot \log(n \cdot s(h)) + r(h) \cdot c))$
Traversing, special case	$\mathcal{O}(n \cdot (s(h) \cdot p + s(H) + c))$
Sorting, special case	$\mathcal{O}(n \cdot (s(h) \cdot p + s(h) \cdot \log(n \cdot s(h)) + s(H) + c))$

Algorithm 2 is actually provided by our implementation of H, so it need not be recomputed each time. Algorithm 3 takes $\mathcal{O}(s(h) + s(H) + c)$ time.

The Whole Propagator. The time complexity of our propagator is obtained by multiplying the filtering complexity by n (the number of variables) and adding the complexity of computing H. Table 3 summarises this for the different versions of the propagator. Note that $s(h) \leq r(h) \leq |\mathrm{D}_x|$ and $s(H) \leq n \cdot s(h)$.

The space complexity of our propagator is $\mathcal{O}(n + s(H))$, as we need to store a constant amount of information (namely w_i) for each variable and the whole function H (which amounts to a constant amount for each of its segments). The functions h_i and H_j are not stored explicitly.

5 Instantiating the Parametric Propagator

We now show how our propagator can be used for particular pairs of constraints.

Note that if h_i is a linear function, then $-h_i$ is also discretely convex. This means that one can put a lower bound $\underline{f}$ on $\sum_{i \in [1,n]} f_i(x_i)$ and run the propagator twice, first with constraint (1) being $\sum_{i \in [1,n]} f_i(x_i) \leq \overline{f}$, then with constraint (1) being $-\sum_{i \in [1,n]} f_i(x_i) \leq -\underline{f}$.

Our propagator can also be extended to handle variables as the upper and lower bounds of the constraints. In such a case, the largest values in the domains of $\overline{f}$ and $\overline{g}$, and the smallest values in the domains of $\underline{f}$ and $\underline{g}$ are used in the propagator. In addition, the other bound of each variable can be constrained by the H function. Only bounds($\mathbb{Z}$) consistency can be achieved on those variables.

5.1 Instantiations and Consistency

We now discuss for which functions f_i and g_i our propagator can be used and how it affects the consistency of the propagator. The required discrete convexity of the h_i functions puts a strong restriction on the shape of the g_i. Recall that $g_i(\mathrm{D}_{x_i})$ must be an interval by the first condition in Definition 1. Note that the discrete convexity must be respected for *all* D_{x_i} that arise during the search.

If D_{x_i} can be any set of integers, then the only instantiations of g_i satisfying the first condition of Definition 1 are those whose image contains only two values, which must be consecutive. We call these *characteristic functions.* In such a case, the second condition of Definition 1 is always respected and the f_i can be any (integer) functions.

If D_{x_i} can only be an interval, then the class of g_i functions satisfying the first condition of Definition 1 is more general, namely all functions where

$$|g_i(u) - g_i(u+1)| \leq 1 \quad \forall u, u+1 \in \mathrm{D}_{x_i} \tag{11}$$

If there are holes in a domain D_{x_i}, then D_{x_i} can be relaxed to the smallest containing interval without losing the correctness of the approach. Some propagation may be lost, but this compromise is often acceptable for global constraints. In particular, we do not achieve domain consistency, but bounds($\mathbb{Z}$) consistency.

Among others, the identity function respects equation (11). If g_i is the identity function, then f_i must be discretely convex, because $h_i = f_i$. For other instantiations of g_i satisfying (11), the restrictions on f_i are varying.

5.2 Example Instantiations

We now show that many existing (pairs of) constraints fit our parametric problem, optionally extended with a lower bound $\underline{f}$ and with variable bounds. Table 1 presents several instantiations of f_i and g_i, together with the derived h_i. We discuss below various constraints and their time complexity. The concrete complexities are derived from the complexities in Table 3 by replacing $s(h)$, $s(H)$, $r(h)$, p, and c by suitable values derived from the h_i.

If $g_i(u) = 0$ for all i, then the second constraint vanishes and we can use our propagator for a single SUM constraint, e.g., a linear inequation. Our parametric propagator is however too general for this simple case, as it runs in $\mathcal{O}(n \cdot \log n)$ time, while a dedicated bounds($\mathbb{Z}$) consistent propagator runs in $\mathcal{O}(n)$ time [6].

The case $g_i(u) = u$ covers many interesting constraints already presented in the literature. In particular, it covers the bounds($\mathbb{Z}$) consistent propagators for the statistical constraints DEVIATION and SPREAD with a fixed rational mean. Interestingly, it can be generalised to any L_p-norm, with $p > 0$ (except $L_{+\infty}$). One can also give a different penalty for deviations over and under the average. The time complexity of our propagator is $\mathcal{O}(n)$ for DEVIATION, which matches the best published propagator [17]. For SPREAD (and higher norms), the time complexity of our propagator is $\mathcal{O}(n \cdot d)$, with $d = \left|\cup_{i\in[1,n]} \mathrm{D}_{x_i}\right|$. This is incomparable to the complexity $\mathcal{O}(n \cdot \log n)$ of the best published propagator [9]. Note that our propagator achieves bounds($\mathbb{Z}$) consistency, which has only been achieved very recently in the case of SPREAD [18].

As an example, we show in Table 4 the instantiation of the parameters for DEVIATION (symmetric parameters are omitted). For DEVIATION, h_i has (up to) three segments, joining at the breakpoints $\lfloor \mu \rfloor$ and $\lceil \mu \rceil$.

The case $g_i(u) = u$ and $f_i(u) = a_i \cdot u$ can be used to model a restricted version of the WEIGHTEDAVERAGE constraint [3], where the weight are variables, the values are constants, and the average must take an integer value. The time complexity of our bounds($\mathbb{Z}$) consistent propagator is $\mathcal{O}(n \cdot \log n)$, though the dedicated propagator runs in $\mathcal{O}(n)$ time.

If g_i is a characteristic function, then f_i can be any function. A characteristic function may be used to count, as is the case of the COUNT family of constraints

Table 4. Expressions for instantiating a propagator for Deviation. The conditions are not always mutually exclusive and are to be evaluated in top-down order.

Parameter	Instantiation
$\text{argmin}_{v \in g_i(\mathrm{D}_{x_i})} h_i(v)$	$\begin{cases} \lceil\mu\rceil & \text{if } \min \mathrm{D}_{x_i} \le \mu \le \max \mathrm{D}_{x_i} \wedge \lceil\mu\rceil - \mu < \mu - \lfloor\mu\rfloor \\ \lfloor\mu\rfloor & \text{if } \min \mathrm{D}_{x_i} \le \mu \le \max \mathrm{D}_{x_i} \wedge \lceil\mu\rceil - \mu \ge \mu - \lfloor\mu\rfloor \\ \min \mathrm{D}_{x_i} & \text{if } \mu < \min \mathrm{D}_{x_i} \\ \max \mathrm{D}_{x_i} & \text{if } \mu > \max \mathrm{D}_{x_i} \end{cases}$
$\Delta^+(h_i, v)$	$\begin{cases} +\infty & \text{if } v = \max \mathrm{D}_{x_i} \\ -n & \text{if } v < \lfloor\mu\rfloor \\ n \cdot (\lceil\mu\rceil + \lfloor\mu\rfloor) - 2 \cdot n \cdot \mu & \text{if } v = \lfloor\mu\rfloor \wedge \lfloor\mu\rfloor \ne \lceil\mu\rceil \\ n & \text{if } v \ge \lceil\mu\rceil \end{cases}$
$\text{bp}^+(h_i, v)$	$\begin{cases} +\infty & \text{if } v = \max \mathrm{D}_{x_i} \\ \min(\max \mathrm{D}_{x_i},\ \lfloor\mu\rfloor) & \text{if } v < \lfloor\mu\rfloor \\ \lceil\mu\rceil & \text{if } v = \lfloor\mu\rfloor \wedge \lfloor\mu\rfloor \ne \lceil\mu\rceil \\ \max \mathrm{D}_{x_i} & \text{if } v \ge \lceil\mu\rceil \end{cases}$
Filter$(g_i(x_i) \le v)$	Filter$(x_i \le v)$
Filter$(g_i(x_i) = v \Rightarrow f_i(x_i) \le u)$	Filter$(\lvert n \cdot v - n \cdot \mu\rvert > u \Rightarrow x_i \ne v)$

(e.g., Among [1,2]). But characteristic functions can also be used to represent the Maximum constraint. Indeed, the constraint $m = \max_{i \in [1,n]} x_i$ can be decomposed as $\forall i \in [1,n] : m \ge x_i \wedge \sum_{1 \in [1,n]} (\textbf{if } x_i \ge m \textbf{ then } 1 \textbf{ else } 0) \ge 1$. Table 1 gives the definition of h_i for Linear and Exactly, in which case our propagator is domain consistent and runs in $\mathcal{O}(n \cdot (\log n + p + c))$ time, as does the dedicated propagator presented in [13].

Many other pairs can be instantiated. Note that the f_i or g_i functions can differ for each i, i.e., one can mix in the same sum terms of different forms (e.g., some linear and some quadratic), as long as each function h_i is discretely convex.

6 Experimental Evaluation

To show that the genericity of our propagator is not detrimental not only to asymptotic complexity (as seen in Section 5) but also to performance, we propose a small experiment to compare custom propagators with instantiations of our parametric propagator. We selected the Deviation [17] and Spread [18] constraints as their bounds($\mathbb{Z}$)-consistent propagators are freely available in the distribution of OscaR [8]. We performed the comparison on the 100 instances of the Balanced Academic Curriculum Problem (BACP) that were introduced in [16],[1] modelled as in the OscaR distribution (we only slightly modified the search heuristic to make it deterministic, so that the search trees are the same).

[1] They are available from `http://becool.info.ucl.ac.be/resources/bacp`

For DEVIATION, we used the 44 instances that are solved to optimality in more than 1 second (to avoid measurement errors) but less than 12 hours (3 instances timed out). When using our parametric propagator, the time to solve an instance is on average only 7% longer than when using the custom propagator (with a standard deviation of 5%). The numbers of nodes in the search tree and calls to the propagator are *exactly* the same for both propagators due to their common level of consistency and the deterministic search procedure.

For SPREAD, we used the 33 instances that are solved to optimality in more than 1 second but less than 12 hours (2 instances timed out). When using our parametric propagator, the time to solve an instance is on average 28% *shorter* than when using the custom propagator (with a standard deviation of 10%). Again, the numbers of nodes in the search tree and calls to the propagator are *exactly* the same for both propagators. This improvement is explained by a different algorithmic approach, which is in our favour when the domains of the variables are small, as is the case for the BACP instances.

Our Java implementation is available at `http://www.it.uu.se/research/group/astra/software/convexpairs` and a package for replication at `http://recomputation.org` [5].

7 Conclusion, Related Work, and Future Work

We have studied how to propagate pairs of SUM constraints that respect a discrete convexity condition. From this condition, we have derived a parametric propagator, which can be instantiated to be competitive with previously published propagators, often matching their time complexity, despite its generality.

Our approach of first computing a feasibility bound and then incrementally adapting it is not new and has been used in the design of several propagators. Among others, this is the case for the constraints covered by our own propagator. However, the novelty of our work is that for the first time we abstract from the details of each constraint to focus on their common properties. This is close in spirit to what has been done with SEQBIN [10] for another class of constraints.

When the g_i are characteristic functions, our conjunction of sum constraints can be represented using COSTGCC [14]. However, this requires the explicit representation of all variable-value pairs and induces a larger time complexity than our propagator. On the other hand, COSTGCC can handle more than one counting constraint in one propagator.

There are a number of open questions we plan to address in the future. Can we *automatically* generate the instantiation of the parameters from the definitions of the f_i and g_i? Can we make an *incremental* propagator that has a better time complexity along a branch of the search tree? Can we extend the approach to functions that take more than one argument, say $f_i(x_i, y_i)$ for variables y_i distinct from each other, or $f_i(x_i, y)$ for a shared variable y? Can we deal with more than two sum constraints in one propagator? Beside when there are holes in the domains, when is it correct and useful to use a relaxation of h_i when this function is not discretely convex?

References

1. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical and Computer Modelling 20(12), 97–123 (1994)
2. Bessière, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: Among, common and disjoint constraints. In: Hnich, B., Carlsson, M., Fages, F., Rossi, F. (eds.) CSCLP 2005. LNCS (LNAI), vol. 3978, pp. 29–43. Springer, Heidelberg (2006)
3. Bonfietti, A., Lombardi, M.: The weighted average constraint. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 191–206. Springer, Heidelberg (2012)
4. Fujishige, S.: Submodular Functions and Optimization. In: Annals of Discrete Mathematics, 2nd edn., Elsevier (2005)
5. Gent, I.P.: The recomputation manifesto. CoRR, abs/1304.3674 (2013)
6. Harvey, W., Schimpf, J.: Bounds consistency techniques for long linear constraints. In: Proceedings of TRICS 2002, the Workshop on Techniques foR Implementing Constraint programming Systems, pp. 39–46 (2002)
7. Murota, K.: Recent developments in discrete convex analysis. In: Cook, W., Lovász, L., Vygen, J. (eds.) Research Trends in Combinatorial Optimization, pp. 219–260. Springer (2009)
8. OscaR Team. OscaR: Scala in OR (2012), https://bitbucket.org/oscarlib/oscar
9. Pesant, G., Régin, J.-C.: SPREAD: A balancing constraint based on statistics. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 460–474. Springer, Heidelberg (2005)
10. Petit, T., Beldiceanu, N., Lorca, X.: A generalized arc-consistency algorithm for a class of counting constraints. In: IJCAI 2011, pp. 643–648. AAAI Press (2011), revised edition available at http://arxiv.org/abs/1110.4719
11. Petit, T., Régin, J.-C., Beldiceanu, N.: A $\Theta(n)$ bound-consistency algorithm for the increasing sum constraint. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 721–728. Springer, Heidelberg (2011)
12. Puget, J.-F.: Improved bound computation in presence of several clique constraints. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 527–541. Springer, Heidelberg (2004)
13. Razakarison, N., Beldiceanu, N., Carlsson, M., Simonis, H.: GAC for a linear inequality and an atleast constraint with an application to learning simple polynomials. In: SoCS 2013, AAAI Press (2013)
14. Régin, J.-C.: Cost-based arc consistency for global cardinality constraints. Constraints 7(3-4), 387–405 (2002)
15. Régin, J.-C., Petit, T.: The objective sum constraint. In: Achterberg, T., Beck, J.C. (eds.) CPAIOR 2011. LNCS, vol. 6697, pp. 190–195. Springer, Heidelberg (2011)
16. Schaus, P.: Solving balancing and bin-packing problems with constraint programming, PhD Thesis, Université catholique de Louvain, Belgium (2009)
17. Schaus, P., Deville, Y., Dupont, P.: Bound-consistent deviation constraint. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 620–634. Springer, Heidelberg (2007)
18. Schaus, P., Régin, J.-C.: Bound-consistent spread constraint, application to load balancing in nurse to patient assignments (submitted)
19. Schulte, C., Stuckey, P.J.: When do bounds and domain propagation lead to the same search space? ACM Transactions on Programming Languages and Systems 27(3), 388–425 (2005)