A Θ(n) Bound-Consistency Algorithm for the Increasing Sum Constraint

Thierry Petit[1], Jean-Charles Régin[2], and Nicolas Beldiceanu[1]

[1]TASC team (INRIA/CNRS), Mines de Nantes, France.
[2] Université de Nice-Sophia Antipolis, I3S UMR 6070, CNRS, France.
{nicolas.beldiceanu,thierry.petit}@mines-nantes.fr, jcregin@gmail.com

Abstract. Given a sequence of variables $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$, we consider the IncreasingSum constraint, which imposes $\forall i \in [0, n-2]$ $x_i \leq x_{i+1}$, and $\sum_{x_i \in X} x_i = s$. We propose an $\Theta(n)$ bound-consistency algorithm for IncreasingSum.

1 Introduction

Many problems involve *sum constraints*, for instance optimization problems. In this paper we consider a specialization of the sum constraint enforcing that an objective variable should be equal to a sum of a set of variables. Given a sequence of variables $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ and a variable s, we propose an $\Theta(n)$ BC algorithm for the IncreasingSum constraint, which imposes that $\forall i \in [0, n-2], x_i \leq x_{i+1} \wedge \sum_{x_i \in X} x_i = s$. IncreasingSum is a special case of the InequalitySum constraint [4], which represents a sum constraint with a graph of binary inequalities.[1]

IncreasingSum is useful for breaking symmetries in some problems. For instance, in bin packing problems some symmetries can be broken by ordering bins according to their use. For each bin i we introduce a variable x_i giving the sum of the heights of the items assigned to i. We can explicitly state that the sum of the x_i's is equal to the sum of the heights of all the items.

2 Sum Constraints in CP

This section discusses the time complexity for filtering sum constraints. Given $x_i \in X$, we denote by $D(x_i)$ the domain of x_i, $min(x_i)$ its minimum value and $max(x_i)$ its maximum value. We say that an assignment $A(X)$ of values to a set of integer variables in X is valid iff each value assigned to $x_i \in X$, denoted by $A(x_i)$, is such that $A(x_i) \in D(x_i)$ (the domain of x_i).

We first recall the usual definitions of GAC and BC.

[1] BC can be achieved on InequalitySum in $O(n \cdot (m + n \cdot log(n)))$ time complexity, where m is the number of binary inequalities (arcs of the graph) and n is the number of variables.

Definition 1 (GAC, BC). *Given a variable x_i and a constraint $C(X)$ such that $x_i \in X$, Value $v \in D(x_i)$*

- *has a* support *on $C(X)$ iff there exists a valid assignment $A(X)$ satisfying C with $A(x_i) = v$.*
- *has a* bounds-support *on $C(X)$ iff there exists an assignment $A(X)$ satisfying C with $A(x_i) = v$ and such that $\forall x_j \in X$, $x_j \neq x_i$, we have $A(x_j) \in [min(D(x_j)), max(D(x_j))]$.*

$C(X)$ is Generalized Arc-Consistent (GAC) *iff $\forall x_i \in X$, $\forall v \in D(x_i)$, v has a support on $C(X)$. $C(X)$ is* Bounds-Consistent (BC) *iff $\forall x_i \in X$, $min(D(x_i))$ and $max(D(x_i))$ have a bound-support on $C(X)$.*

Given a set X of integer variables and an integer k, we denote by $\sum_{=} k$ the problem consisting of determining whether there exists an assignment of values to variables in X such that $\sum_{x_i \in X} x_i = k$, or not. This problem is NP-Complete [4, p. 7]: The SubsetSum problem [2, p. 223], which is NP-Complete, is a particular instance of the feasibility check of a constraint $\sum_{x_i \in X} x_i = k$.

When we consider an objective variable s instead of an integer k, performing GAC on $\sum_{x_i \in X} x_i = s$ is NP-Hard since one has to check the consistency of all values in $D(s)$, which corresponds to the $\sum_{=} k$ problem.

Conversely, enforcing BC on $\sum_{x_i \in X} x_i = s$ is in P as well as achieving BC on $\sum_{x_i \in X, a_i \in \mathbb{N}} a_i \cdot x_i \leq s$ [3].

In practice the constraint $\sum_{x_i \in X} x_i = s$ is generally associated with some additional constraints on variables in X. Next section presents a BC algorithm for a sum with increasing variables. This constraint may occur in problems involving sum constraints on symmetrical variables.

3 BC Linear Algorithm for Increasing Sum

Given a sequence of variables $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ and a variable s, this section presents an $\Theta(n)$ algorithm for enforcing BC on the constraint IncreasingSum$(X, s) = \forall i \in \{0, 1, \ldots n-2\}, x_i \leq x_{i+1} \wedge \sum_{x_i \in X} x_i = s$. Following Definition 1, and since we consider an algorithm achieving BC, **this section ignores holes in the domains of variables**.

Definition 2. *Let $x_i \in X$ be a variable. $D(x_i)$ is $\leq$-consistent iff there exists two assignments $A(X)$ and $A'(X)$ such that $A(x_i) = min(x_i)$ and $A'(x_i) = max(x_i)$ and $\forall j \in [0, n-2]$, $A(x_j) \leq A(x_{j+1})$ and $A'(x_j) \leq A'(x_{j+1})$. X is $\leq$-consistent iff $\forall x_i \in X$, $D(x_i)$ is $\leq$-consistent.*

W.l.o.g., **from now we consider that X is $\leq$-consistent**. In practice we can ensure $\leq$-consistency of X in $\Theta(n)$ by traversing X so as to make for each $i \in [0, n-2]$ the bounds of the variables x_i and x_{i+1} consistent with the constraint $x_i \leq x_{i+1}$. After making X $\leq$-consistent, it is easy to evaluate a lower bound and an upper bound of the sum s.

Lemma 1. *Given* IncreasingSum(X, s)*, the intervals* $[min(s), \sum_{x_i \in X} min(x_i)[$ *and* $]\sum_{x_i \in X} max(x_i), max(s)]$ *can be removed from* $D(s)$*.*

Proof. $\sum_{x_i \in X} min(x_i) \leq \sum_{x_i \in X} x_i \leq \sum_{x_i \in X} max(x_i)$. □

Lemma 1 does not ensure that IncreasingSum is BC, since we can have $min(s) > \sum_{x_i \in X} min(x_i)$ and $max(s) < \sum_{x_i \in X} max(x_i)$. In this case, bounds of variables in X may not be consistent, and some additional pruning needs to be performed. Next example highlights this claim.

Example 1. We consider IncreasingSum(X, s), $D(s) = \{28, 29\}$ and the sequence $X = \langle x_0, x_1, \ldots, x_5 \rangle$. We denote by <u>sum</u> the minimum value of the sum of variables in X.

$D(x_0) = \{$ 2, 3, 4, 5, 6 $\}$, <u>sum</u> if $x_0 = 6 : 28 + \mathbf{9} = 37$
$D(x_1) = \{$ 4, 5, 6, 7 $\}$, <u>sum</u> if $x_1 = 7 : 28 + \mathbf{9} = 37$
$D(x_2) = \{$ 4, 5, 6, 7 $\}$, <u>sum</u> if $x_2 = 7 : 28 + \mathbf{6} = 34$
$D(x_3) = \{$ 5, 6, 7 $\}$, <u>sum</u> if $x_3 = 7 : 28 + \mathbf{3} = 31$
$D(x_4) = \{$ 6, 7, 8, 9 $\}$, <u>sum</u> if $x_4 = 9 : 28 + \mathbf{5} = 33$
$D(x_5) = \{$ 7, 8, 9 $\}$, <u>sum</u> if $x_5 = 9 : 28 + \mathbf{2} = 30$.

For all $x_i \in X$, $min(x_i)$ is consistent since $\sum_{x_i \in X} min(x_i) = 28 = min(s)$, and $max(x_i)$ is not consistent. The increase in the sum corresponding to $max(x_i)$ (the bold values) is computed by considering that values assigned to variables having an index greater than i should be at least equal to $max(x_i)$. For instance, if $x_0 = 6$ then $\underline{sum} = 28 + 9 = 37$ with $9 = 4 + 2 + 2 + 1 + 0 + 0$, where 4 is the increase with respect to x_0, 2 the increase with respect to x_1, and so on.

Conversely, once s has been updated thanks to Lemma 1, all values between $min(s)$ and $max(s)$ are bound-consistent with IncreasingSum.

Property 1. Given IncreasingSum(X, s), if $min(s) \geq \sum_{x_i \in X} min(x_i)$, $max(s) \leq \sum_{x_i \in X} max(x_i)$ and $min(s) \leq max(s)$ then $\forall v \in D(s)$ there exists an assignment $A(X)$ such that $\sum_{x_i \in X} A(x_i) = v$.

Proof. Let $\delta \geq 0$ such that $v \in D(s)$ and $v = \sum_{x_i \in X} min(x_i) + \delta$. If $\delta = 0$ then the property holds. Assume the property is true for $\delta = k$: there exists an assignment $A(X)$ with $\sum_{x_i \in X} A(x_i) = \sum_{x_i \in X} min(x_i) + k$. We prove that it remains true for $\delta = k + 1$, that is, $v = \sum_{x_i \in X} min(x_i) + k + 1$. First, if $v > \sum_{x_i \in X} max(x_i)$ the property holds (the condition is violated). Otherwise, consider $A(X)$. We have not $\forall i \in [0, n-1], A(x_i) = max(x_i)$ since $v \leq \sum_{x_i \in X} max(x_i)$. Therefore, consider the greatest index $i \in [0, n-1]$ such that $A(x_i) < max(x_i)$. All $x_j \in X$ such that $j > i$ (if $i = n-1$ no such x_j exists) satisfy by definition $A(x_j) = max(x_j)$. Variables in X are range variables, thus $A(x_i) + 1 \in D(x_i)$. X is $\leq$-consistent: if $i < n - 1$ then $A(x_i) + 1 \leq A(x_{i+1})$. Moreover, if $i < n - 1$, $A(x_{i+1}) = max(x_{i+1})$ by definition of i. In all cases, ($i < n-1$ or $i = n-1$), assignment $A'(X)$ such that $A'(x_i) = A(x_i) + 1$ is such that $\sum_{x_i \in X} A'(x_i) = \sum_{x_i \in X} min(x_i) + k + 1 = v$. The Property holds. □

Once Property 1 is satisfied, we have to focus on bounds of variables in X. We restrict ourself to the maximum values in domains. The case of minimum values is symmetrical. We consider also that $D(s)$ is not empty after applying Lemma 1, which entails that no domain of a variable in X can become empty, *i.e.*, we have at least one feasible solution for IncreasingSum.

In Example 1, all maximum values of domains should be reduced. For all x_i in X, if we assign $max(x_i)$ to x_i the overload on $min(s)$ (bold values in Example 1) is too big, *i.e.*, $max(s)$ is exceeded. To reduce the upper bound of a variable x_i, we search for the greatest value v in $D(x_i)$ which leads to a value of s less than or equal to $max(s)$.

Notation 1 *Given a value $v \in D(x_j)$, we denote by $bp(X, j, v)$* (break point) *the minimum value of the sum $\sum_{x_i \in X} x_i$ of an assignment $A(X)$ satisfying for each $i \in [0, n-2]$ the constraint $x_i \leq x_{i+1}$ and such that $x_j = v$.*

To compute this quantity we introduce the notion of *last intersecting index*, which allows to split $\sum_{x_i \in X} x_i$ in three sub-sums that can be evaluated independently.

Definition 3. *Given* IncreasingSum(X, s)*, let $i \in [0, n-1]$ be an integer. The* last intersecting index *$last_i$ of variable x_i is equal either to the greatest index in $[i+1, n-1]$ such that $max(x_i) > min(x_{last_i})$, or to i if no integer k in $[i+1, n-1]$ is such that $max(x_i) > min(x_k)$.*

Property 2. Given IncreasingSum(X, s), let $i \in [0, n-1]$ be an integer and $v \in D(x_i)$, $bp(X, i, v) =$

$$\left(\sum_{k \in [0,\ldots,i-1]} min(x_k)\right) + bp(\langle x_i, \ldots, x_{last_i}\rangle, i, v) + \left(\sum_{k \in [last_i+1,\ldots,n-1]} min(x_k)\right)$$

Proof. By Definition 3, any variable x_k in $\{x_0, \ldots, x_{i-1}\} \cup \{x_{last_i+1}, \ldots, x_{n-1}\}$ can be assigned to its minimum $min(x_k)$ within an assignment of X where: (1) x_i is assigned to v, and (2) this assignment satisfies $\forall k \in [0, n-2], x_k \leq x_{k+1}$. □

From Property 2, we know that to check the feasibility of the upper bound of x_i we have to compute $bp(\langle x_i, \ldots, x_{last_i}\rangle, i, max(x_i))$.

Property 3. Given IncreasingSum(X, s), let $i \in [0, n-1]$ and $last_i$ be the last intersecting index of x_i, $bp(\langle x_i, \ldots, x_{last_i}\rangle, i, max(x_i)) = \sum_{k \in [i, last_i]} max(x_i)$.

Proof. By Definition 3, $last_i$ is the *greatest* index, greater than i, such that $min(x_{last_i}) < max(x_i)$, or i if no such an index exists. All variables x_k in $\langle x_i, \ldots, x_{last_i}\rangle$ are such that $min(x_k) \leq max(x_i)$, thus assigning $max(x_i)$ to x_i implies assigning a value greater than or equal to $max(x_i)$ to any x_k such that $k \in [i+1, last_i]$, in order to satisfy $\forall l \in [i+1, last_i]$ the constraint $x_{l-1} \leq x_l$. Since X is $\leq$-consistent, for each $k \in [i, last_i]$ $max(x_i) \in D(x_k)$ and the minimum increase due to x_k compared with $\sum_{x_k \in [i, last_i]} min(x_k)$ if $x_i = max(x_i)$ is $max(x_i) - min(x_k)$. □

From Property 3 we obtain a consistency check for the maximum value of x_i. We use the following notations:

– $margin = max(s) - \sum_{k\in[0,n-1]} min(x_k)$; we consider $\sum_{k\in[0,n-1]} min(x_k)$ because our goal is here to reduce upper bounds of domains of variables in X according to $max(s)$.
– $\Delta_i = \sum_{k\in[i,last_i]}(max(x_i) - min(x_k))$; Δ_i is the minimum increase with respect to $\sum_{k\in[0,n-1]} min(x_k)$ under the hypothesis that x_i is fixed to $max(x_i)$.

Lemma 2. *Given* IncreasingSum(X, s) *and* $i \in [0, n-1]$*, if* $\Delta_i > margin$ *then* $max(x_i)$ *is not consistent.*

Proof. Obvious from Property 3. □

We now present our BC algorithm. Algorithm 1 prunes the maximum values in domains of variables in a $\leq$-consistent sequence X, using an incremental computation of Δ_i, starting from the last variable x_{n-1} and considering at each step the valid last intersection index. When the condition of Lemma 2 is satisfied, that is, $\Delta_i > margin$, Algorithm 1 calls the procedure FilterMaxVar$(x_i, last_i, \Delta_i, margin)$ to decrease $max(x_i)$. This procedure is described later.

Algorithm 1: FilterMaxVars(X, s)

```
1  minsum := 0;
2  for i = 0 to n − 1 do  minsum := minsum + min(x_i);
3  margin := max(s) − minsum; i := n − 1; last_i := i; Δ_i := max(x_i) − min(x_i);
4  while i ≥ 0 do
5      if Δ_i ≤ margin then
6          oldmax := max(x_i);
7          i := i − 1 ;
8          if i ≥ 0 then
9              while (min(x_{last_i}) ≥ max(x_i)) ∧ (last_i > i) do
10                 Δ_i := Δ_i − (oldmax − min(x_{last_i}));
11                 last_i := last_i − 1;
12             Δ_i := Δ_i + max(x_i) − min(x_i) − (last_i − i) · (oldmax − max(x_i));
13     else  (last_i, Δ_i) := FilterMaxVar(x_i, last_i, Δ_i, margin);
14     if i > 0 ∧ max(x_{i−1}) > max(x_i)) then  max(x_{i−1}) := max(x_i);
```

Figure 1 illustrates with an example of the incremental update of Δ_i (lines 9-12 of Algorithm 1) when $\Delta_i < margin$ and i is decremented by one.

We now describe how the procedure FilterMaxVar$(x_i, last_i, \Delta_i, margin)$ can be implemented to obtain a time complexity linear in the number of variables for Algorithm 1. We thus consider that the condition of Lemma 2 is satisfied, that is, $\Delta_i > margin$. It is required to reduce $max(x_i)$.

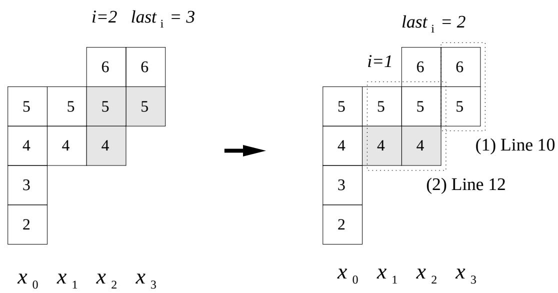

Fig. 1: Execution of Algorithm 1 with $margin = 4$ and 4 variables such that $D(x_0) = [2,5], D(x_1) = [4,5], D(x_2) = [4,6], D(x_3) = [5,6]$. On the left side, the current index is $i = 2$, $last_2 = 3$ and we have $\Delta_2 = 3$ (bolded values). Since $\Delta_2 < margin$ no pruning is performed and the algorithm moves to the next variable ($i = 1$). The right side shows that: (1) Δ_1 is first updated by removing the contributions computed with the previous maximum value of x_i ($oldmax = max(x_2)$) at the variable indexed by the previous last intersecting index $last_2 = 3$ (line 10 of Algorithm 1), and then $last_i$ is decreased (line 11). (2) According to the new $last_1 = 2$, Δ_1 is increased by the contribution of x_1, while the exceed over $max(x_2)$ of variables indexed between $i = 1$ and $last_1 = 2$ is removed from Δ_1 (line 12).

Our aim is then to update x_i and update both $last_i$ and Δ_i while preserving the property that the time complexity of Algorithm 1 is linear in the number of variables. The principle is the following.

Algorithm 2: FilterMaxVar($x_i, last_i, \Delta_i, margin$)

while $\Delta_i > margin$ **do**
  $steps := min(\lceil \frac{\Delta_i - margin}{last_i - i + 1} \rceil, max(x_i) - min(x_{last_i}));$
  $D(x_i) := D(x_i) \setminus \,]max(x_i) - steps, max(x_i)]$;
  $\Delta_i := \Delta_i - (last_i - i + 1) \cdot (steps)$;
  while $(min(x_{last_i}) \geq max(x_i)) \wedge (last_i > i)$ **do** $last_i := last_i - 1;$
return $(last_i, \Delta_i)$;

If we assume that all variables $\langle x_i, x_{i+1}, \ldots, x_{last_i} \rangle$ will be assigned the same value then the minimum number of horizontal slices to remove (each slice corresponding to a same value, that can potentially be assigned to each variable in $\langle x_i, x_{i+1}, \ldots, x_{last_i} \rangle$) in order to absorb the exceed $\Delta_i - margin$ is equal to $\lceil \frac{\Delta_i - margin}{last_i - i + 1} \rceil$. Then, two cases are possible.

1. If $\lceil \frac{\Delta_i - margin}{last_i - i + 1} \rceil$ is strictly less (strictly since one extra slice is reserved for the common value assigned to $x_i, x_{i+1}, \ldots, x_{last_i}$, that is, the new maximum of x_i) than the number of available slices between $min(x_{last_i})$ and $max(x_i)$, namely $max(x_i) - min(x_{last_i}) + 1$, then removing $]max(x_i) - \lceil \frac{\Delta_i - margin}{last_i - i + 1} \rceil, max(x_i)]$ gives the feasible upper bound of x_i.

2. Otherwise, the quantity $q = max(x_i) - \lceil \frac{\Delta_i - margin}{last_i - i + 1} \rceil$ is not necessarily a feasible upper bound of x_i. In this case we decrease $max(x_i)$ down to $\min(x_{last_i})$, that is, we consider the number of available slices consistent with the current $last_i$. Then we update $last_i$ and Δ_i and we repeat the process.

Algorithm 2 implements these principles. It takes as arguments the variable x_i, the last intersecting index $last_i$ of x_i, Δ_i and $margin$. It prunes the max of x_i and returns the updated pair $(last_i, \Delta_i)$. Figure 2 depicts an example where the pruning of x_i requires more than one step.

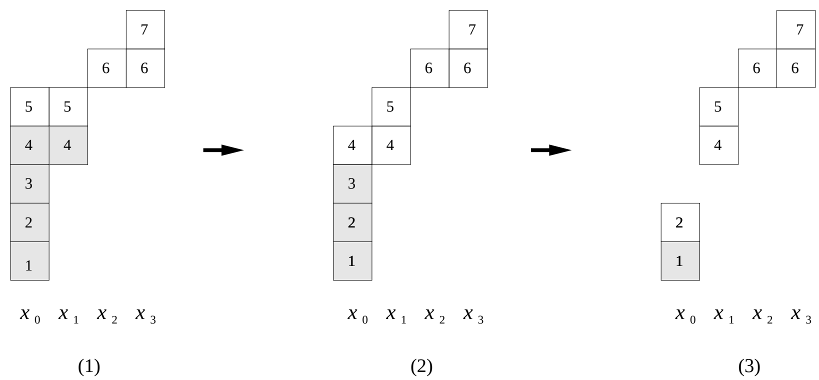

Fig. 2: Execution of Algorithm 2 with $i = 0$, $margin = 1$, $\Delta_i = 5$, and $last_0 = 1$. $D(x_0) = [1, 5], D(x_1) = [4, 5], D(x_2) = [6, 6], D(x_3) = [6, 7]$. (1) $\Delta_0 > margin$ so we compute $\lceil \frac{\Delta_0 - margin}{1-0+1} \rceil = 2$, which is not strictly less than $max(x_0) - min(x_1) + 1 = 2$, so $steps = max(x_0) - min(x_1) = 1$ and several phases may be required to prune x_0. (2) $D(x_0) := D(x_0) \setminus]5 - 1, max(x_0)] = [1, 4]$. $\Delta_0 = \Delta_0 - (1 - 0 + 1) * 1 = 3$. $(min(x_1) \geq max(x_0)) \wedge (1 > 0)$ so $last_0 = 1 - 1 = 0$. (3) $\Delta_0 > margin$ so we compute $\lceil \frac{\Delta_0 - margin}{0-0+1} \rceil = \lceil \frac{3-1}{0-0+1} \rceil = 2$, which is strictly less than $max(x_0) - min(x_0) + 1 = 4$. $D(x_0) := D(x_0) \setminus]4 - 2, max(x_0)] = [1, 2]$, and we have $\Delta_i = margin = 1$.

With respect to time complexity, recall $\leq$-consistency of X can be achieved in $\Theta(n)$ before runing Algorithm 1 by traversing the sequence and ensuring for each $i \in [0, n-2]$ that bounds of variables are consistent with $x_i \leq x_{i+1}$. Therefore, the time complexity of for achieving BC is linear in the number of variables, since the following proposition holds with respect to Algorithm 1.

Proposition 1. *Time complexity of Algorithm 1 is $\Theta(n)$.*

Proof. An invariant of both Algorithm 2 and Algorithm 1 is that during the whole pruning of X, the index $last_i$ only decreases. Moreover, in Algorithm 2, if $steps = max(x_i) - min(x_{last_i}) + 1$ then $last_i$ decreases, otherwise $\Delta_i = margin$ and the algorithm ends. Thus, the cumulative time spent in the loop of line 5 in Algorithm 2 as well as the loop of lines 8-9 in Algorithm 1 is in n, the number of variables in X. Therefore, time complexity of Algorithm 1 is $O(n)$. Since to reduce domains of all the variables in X we have at least to update each of them, this time complexity is optimum. The proposition holds. $\square$

Furthermore, if minimum values of domains of variables in X are pruned after maximum values, there is no need to recompute those maximum values: increasing the lower bound $min(x_i)$ of a variable x_i leads to a diminution of *margin* and exactly the same diminution in Δ_i. Therefore, applying a second time Algorithm 1 cannot lead to more pruning. The reasoning is symmetrical if maximum values are filtered after minimum values. As a consequence, BC can be achieved in three phases: the first one to ensure $\leq$-consistency of X and adjust the bounds of s, the second one for maximum values in domains of variables in X, and the third one for minimum values in in domains of variables in X.

4 Conclusion and Future Work

We presented a $\Theta(n)$ BC algorithm for IncreasingSum(X, s), where $X = \langle x_0, x_1, \ldots, x_n \rangle$ is a sequence of variables and s is a variable. This constraint can be used in problems with variable symmetries involved in a sum. A Choco [1] implementation is available.

IncreasingSum can be used to enforce BC on the following generalization: $\forall i \in [0, n-2], x_i \leq x_{i+1}+cst \wedge \sum_{x_i \in X} x_i = s$, where cst is a constant. Indeed, we can add n additional variables X', one additional variable s' and $n+1$ mapping constraints: $\forall i \in [0, n-1], x'_i = x_i + cst \cdot i$ and $s' = s + \sum_{i \in [1,n-1]} i \cdot k$. Then enforcing BC on IncreasingSum(X', s') also enforces BC on variables in X and s since we use only mapping (equality) constraints. Time complexity remains $\Theta(n)$ because we add $O(n)$ variables.

With respect to GAC on IncreasingSum, Property 1 is not true when variables in X may have some holes in their domains. For instance, consider a sequence X of three variables with $D(x_0) = D(x_1) = D(x_2) = \{1, 3\}$ and a variable s with domain $D(s) = \{3, 6, 9\}$. Values 3 and 9 in $D(s)$ are consistent with IncreasingSum(X, s) while value 6 in $D(s)$ is not consistent with IncreasingSum(X, s). From this remark, enforcing GAC may require a check in $O(d^n)$ per value in s. A solution to IncreasingSum corresponds to a "sorted" solution of the SubsetSum problem, which does not make that problem easier.

References

1. Choco: An open source Java CP library, documentation manual. `http://www.emn.fr/z-info/choco-solver/`, 2011.
2. M. R. Garey and D. S. Johnson. Computers and intractability : A guide to the theory of NP-completeness. *W.H. Freeman and Company*, ISBN 0-7167-1045-5, 1979.
3. W. Harvey and J. Schimpf. Bounds Consistency Techniques for Long Linear Constraints. In *CP'02 Workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*, pages 39–46, 2002.
4. J.-C. Régin and Michel Rueher. Inequality-sum: a global constraint capturing the objective function. *RAIRO - Operations Research*, 39:123–139, 2005.