Bound-Consistent Deviation Constraint

Pierre Schaus, Yves Deville, Pierre Dupont

Department of Computing Sciences and Engineering
Université catholique de Louvain, Belgium
{pierre.schaus, yves.deville, pierre.dupont}@uclouvain.be

Abstract. Deviation is a recent constraint to balance a set of variables with respect to a given mean. We show that the propagators recently introduced are not bound-consistent when the mean is rational. We introduce bound-consistent propagators running in linear time with respect to the number of variables. We evaluate the improvement in terms of efficiency and pruning obtained with the new propagators on the Balanced Academic Curriculum Problem.

1 Introduction

Global constraints to obtain a balanced assignment on a set of variables has not received much attention up to now. Some possible applications for such constraints are the following: fairly distribute the night and weekend shifts in physician scheduling in emergency rooms [3], balance the tardiness of tasks in a scheduling problem, balance the violations among soft global constraints, balance the load of work between periods in a timetabling problem [2], and generate spatially balanced scientific experiments [5].

The constraint `deviation` has been recently introduced in [7]. This constraint guarantees an assignment on a set of variables to be balanced around a given mean. More precisely `deviation` constrains a set of variables to present a given mean and constrains the sum of deviations to this mean. A closely related constraint using a different measure of balance is `spread` [4, 6]. The propagators for `spread` run in quadratic time with respect to the number of variables against linear time for `deviation` . The semantic of `deviation` is given in the following definition.

Definition 1. *Given n finite domain variables $\mathcal{X} = (X_1, X_2, ..., X_n)$, one integer value s and one finite domain variable Δ,* `deviation`$(\mathcal{X}, s, \Delta)$ *holds if and only if*

$$\sum_{i=1}^{n} X_i = s \quad \text{and} \quad \Delta \geq \sum_{i=1}^{n} |n \cdot X_i - s|.$$

In other words, `deviation`$(\mathcal{X}, s, \Delta)$ is the conjunction of two constraints. One sum constraint enforcing the sum of the variables to be equal to s and one deviation constraint enforcing the sum of absolute deviations of $n \cdot X_i$ to the sum s to be less than or equal to Δ [1]. Another formulation is that the mean

[1] Bound-consistency is $\mathcal{NP}$-Complete when it is constrained to be equal to Δ [7]

(or average) of variables X_i must be equal to s/n and the sum of deviations to this mean must be smaller than Δ/n. The definition of `deviation` might seem restrictive since the sum is fixed. However in many practical applications the sum is known: one often needs to distribute (weighted) items into categories (nurses, shifts,...) and balance the loads of the categories.

The domain of a variable A is denoted $Dom(A)$, the maximum and minimum values in $Dom(A)$ are denoted $A^{\min}$ and $A^{\max}$ respectively.

Two propagators can be imagined for the `deviation` constraint:

1. Increasing of $\Delta^{\min}$ given domains of variables in $\mathcal{X}$ and value s.
2. Narrowing of $Dom(X_i)$ given the values $\Delta^{\min}$, s and the domains $Dom(X_j)$ with $X_j \in \mathcal{X}$ and $i \neq j$.

This paper gives bound-consistent filtering algorithms for both propagators running in linear time $\Theta(n)$.

Section 2 motivates the need for new propagators by explaining the weaknesses of the bounds computed in [7]. The improved bound-consistent ones considered in this paper are introduced. Section 3 and 4 give linear time filtering algorithms for propagators 1 and 2 respectively. Finally, Section 5 experiments the improvement made by the new propagators on the Balanced Academic Curriculum Problem.

2 Weakness of existing propagators

This section starts with some notations useful for the rest of the paper. Then the weaknesses of the bounds computed in [7] are explained and the improved bound-consistent ones considered in this paper are introduced.

An integer interval between integer numbers a and b is denoted $[a..b] \subseteq \mathbb{Z}$ while the rational interval is denoted $[a, b] \subseteq \mathbb{Q}$. An assignment on the variables $\mathcal{X} = (X_1, X_2, ..., X_n)$ is denoted by the tuple x and the ith entry of this tuple by $x[i]$. We denote by $s^{\downarrow}$ the largest multiple of n from s not larger than s: $s^{\downarrow} = \lfloor s/n \rfloor \cdot n$ and by $s^{\uparrow}$ the value $s^{\downarrow} + n$. The rational interval domain of X_i is $I_i^{\mathbb{Q}} = [X_i^{\min}, X_i^{\max}]$ and its integer interval domain is $I_i^{\mathbb{Z}} = [X_i^{\min} \,..\, X_i^{\max}]$.

Definition 2 (Bound Consistency). *A global constraint $C(X_1, ..., X_k)$ is bound-consistent if and only if the minimum value and maximum value of every variable X_i with $i \in [1..k]$ has a support in the constraint assuming the other variables $X_{j \neq i}$ take their value from $[X_j^{\min}..X_j^{\max}]$.*

Filtering algorithms from [7] are simple and efficient (run-time in $\Theta(n)$). However, for integer finite domains, these algorithms are bound-consistent only when $s \bmod n = 0$ that is when the mean s/n is an integer. The reason is the relaxing assumption that the domains are rational intervals instead of integer intervals when computing the bounds. Definition 3 gives the expressions of the computed bounds in [7] and the bound-consistent ones considered in this paper.

Definition 3. *$\underline{\Delta}^{\mathbb{Q}}$ denotes the minimal sum of deviations with rational interval domains:*

$$\underline{\Delta}^{\mathbb{Q}} = \min_{x}\{\sum_{i=1}^{n} |n \cdot x[i] - s| \;\big|\; \forall i :\; x[i] \in I_i^{\mathbb{Q}} \text{ and } \sum_{i=1}^{n} x[i] = s\}. \tag{1}$$

$\underline{\Delta}^{\mathbb{Z}}$ denotes the minimal sum of deviations with integer interval domains obtained by substituting $\mathbb{Q}$ by $\mathbb{Z}$ in equation (1).

$\overline{X}_i^{\mathbb{Q}}$ denotes the maximal consistent value for X_i with rational interval domains:

$$\overline{X}_i^{\mathbb{Q}} = \max_{x} \{\, x[i] \;\big|\; \forall j :\; x[j] \in I_j^{\mathbb{Q}} \text{ and } \sum_{j=1}^{n} x[j] = s \text{ and} \sum_{j=1}^{n} |n \cdot x[j] - s| \leq \Delta^{\max}\}. \tag{2}$$

$\overline{X}_i^{\mathbb{Z}}$ denotes the maximal consistent value for X_i with integer interval domains obtained by substituting $\mathbb{Q}$ by $\mathbb{Z}$ in equation (2).

Corresponding definitions for $\underline{X}_i^{\mathbb{Q}}$ and $\underline{X}_i^{\mathbb{Z}}$ are obtained by replacing maximization over x by minimization in equation (2).

The two propagators described in [7] filtering Δ and $\mathcal{X}$ apply respectively the filtering rules

$$\Delta^{\min} \leftarrow \max(\Delta^{\min}, \underline{\Delta}^{\mathbb{Q}}) \quad \text{and} \tag{3}$$

$$Dom(X_i) \leftarrow Dom(X_i) \cap [\underline{X}_i^{\mathbb{Q}}, \overline{X}_i^{\mathbb{Q}}] \quad \forall i \in [1..n]. \tag{4}$$

These filtering rules are bound-consistent if the domains of the X_i's are rational intervals $[X_i^{\min}, X_i^{\max}]$. When the domains of the X_i's are integer intervals $[X_i^{\min}..X_i^{\max}]$, the corresponding bound-consistent filtering rules are obtained by substituting $\mathbb{Q}$ by $\mathbb{Z}$ in equations (3) and (4). Nevertheless, rules (3) and (4) can be used for integer domains as well since they are obtained by relaxing the domains to rational intervals. The relations between the bounds are $\underline{X}_i^{\mathbb{Z}} \geq \underline{X}_i^{\mathbb{Q}}$, $\overline{X}_i^{\mathbb{Z}} \leq \overline{X}_i^{\mathbb{Q}}$ and $\underline{\Delta}^{\mathbb{Z}} \geq \underline{\Delta}^{\mathbb{Q}}$. In the particular case of $s \bmod n = 0$, the bounds are completely equivalent. As illustrated in the two following examples, the relaxing assumption of rational interval domains can lead to miss some possible filtering with respect to a bound-consistent filtering.

Example 1 (Filtering of Δ). Assume two variables $\mathcal{X} = (X_1, X_2)$ with domains $[-5..5]$ and a sum constraint $s = 1$. Obviously $\underline{\Delta}^{\mathbb{Q}} = 0$ is obtained with the tuple $x = (0.5, 0.5)$ while $\underline{\Delta}^{\mathbb{Z}} = 2$ is obtained with the tuple $x = (1, 0)$ or $x = (0, 1)$.

Example 2 (Filtering of $\mathcal{X}$). Assume ten variables with domains $[-5..5]$, a sum constraint $s = 7$ and a maximum sum of deviations $\Delta^{\max} = 42$. One can see that $\overline{X}_i^{\mathbb{Q}} = \frac{7}{10} + \frac{21}{10} = 2.8$ and $\underline{X}_i^{\mathbb{Q}} = \frac{7}{10} - \frac{21}{10} = -1.4$. This solution is obtained

if eight variables are assigned to the mean 7/10 and the other two are as far as possible from the mean that is one above the mean and the other below the mean at an equal distance $\frac{21}{10}$. For this configuration, the maximum deviation $\Delta^{\max} = 42$ is reached. When only integer assignments are permitted, the result is $\overline{X}_i^{\mathbb{Z}} = 1$ and $\underline{X}_i^{\mathbb{Z}} = 0$. Indeed, for an assignment composed of seven values 1 and three values 0, the maximal deviation is reached ($\Delta^{\max} = 42$). Clearly there is no other integer assignment with a lower deviation. Hence the filtering of [7] would achieve $Dom(X_i) = [-1..2]$ while a bound-consistent filtering would give $Dom(X_i) = [0..1]$.

3 A bound-consistent lower bound for the deviation.

The previous section shows in Example 1 that when every domain overlaps the mean, the lower bound for the deviation computed by propagators in [7] is equal to 0 since every variable can be assigned to the mean s/n. This lower bound is not bound-consistent when the mean is rational (when $s \bmod n \neq 0$). Next theorem gives a lower bound for Δ that can be computed in constant time and greater than 0 in this case.

Theorem 1. *A lower bound for the deviation Δ is:*

$$0 \leq 2 \cdot (n - s \bmod n) \cdot (s \bmod n) \leq \underline{\Delta}^{\mathbb{Z}}.$$

Proof. This lower bound is obtained by enlarging every domain $Dom(X_i)$ such that s/n gets inside: $\forall i \in [1..n] : s/n \in [X_i^{\min}, X_i^{\max}]$. Then in an assignment of minimum deviation, every variable are either assigned to $s^{\downarrow}$ or to $s^{\uparrow} = s^{\downarrow} + n$. If we denote by y the number of variables $(n \cdot X_i)$ assigned to $s^{\downarrow}$, the sum constraint can be written: $y \cdot s^{\downarrow} + (n-y) \cdot (s^{\downarrow} + n) = s \cdot n$. Hence $y = n - (s - s^{\downarrow}) = n - s \bmod n$. Using this, a lower bound of $\underline{\Delta}^{\mathbb{Z}}$ is $(n - s \bmod n) \cdot (s \bmod n) + (s \bmod n) \cdot (n - s \bmod n) = 2 \cdot (n - s \bmod n) \cdot (s \bmod n)$. □

The lower bound introduced in Theorem 1 is bound-consistent only if every domain overlaps the mean s/n. The remaining of this section introduces a linear time algorithm to compute a valid assignment satisfying the sum constraint and minimizing the sum of deviations in the general case when the domains do not necessarily overlap the mean. More formally the algorithm computes a tuple x satisfying the relation [2]:

$$\operatorname*{argmin}_x \{ (\sum_{i=1}^{n} |n \cdot x[i] - s|) \big| \forall i : \ x[i] \in I_i^{\mathbb{Z}} \text{ and } \sum_{i=1}^{n} x[i] = s \}.$$

To alleviate notations, the tuple $n \cdot x$ is used instead of x. Note that $n \cdot x$ corresponds to an integer assignment only if it is composed of values which are multiple of n. The algorithm executes in two phases: a greedy part followed by a repair part.

[2] $\operatorname{argmin}_x f(X)$ is the set of x such that $f(x)$ is minimal.

– Greedy: The sum constraint is dropped. Each $n \cdot x[i]$ is set to the closest multiple of n from s in $Dom(n \cdot X_i)$.
– Repair: If the sum constraint is satisfied that is $\sum_{i=1}^{n} x[i] = s$, then $n \cdot x$ is a solution to the problem. Otherwise the sum is larger or smaller than s. We consider the larger case: $\sum_{i=1}^{n} x[i] > s$ (the other case is similar). Then some entries of $n \cdot x$ must be decreased until the sum constraint is satisfied. An entry $n \cdot x[i] = s^{\uparrow} > n \cdot X_i^{\min}$ is called an *overlapping* entry. The choice of the entries to decrease is important. Decreasing an entry which is smaller than s by n results in an augmentation by n of the sum of deviations. But decreasing an overlapping entry by n (that is from $n \cdot x[i] = s^{\uparrow}$ to $s^{\downarrow}$) only increases the sum of deviations by $(2 \cdot (s \bmod n) - n)$ (see Figure 1). This last quantity is smaller or equal to n. Consequently, all overlapping entries are first considered in any order to be decreased by n to satisfy the sum constraint. If the sum constraint is not yet satisfied after this operation, the following property holds:

$$\forall i : n \cdot x[i] \leq s \text{ or } n \cdot X_i^{\min} \geq s.$$

In other words, each entry $n \cdot x[i]$ lies either on the lower bound of the corresponding variable domain or lies below s and can if necessary be further decreased. Consequently every entry below s, not yet on its lower bound, can be decreased at most to its lower bound $(n \cdot X_i^{\min})$. This results in an augmentation of the sum of deviations equal to the amount of the decreasing. These entries are used to satisfy the sum constraint. They are decreased maximally in an arbitrary order until the sum constraint is satisfied.

The greedy part is achieved by iterating once over the variables. There are at most n overlapping variables candidates to a repair. Finally, there are at most n variables needed to be further decreased to satisfy the sum constraint. Hence the total complexity is $\Theta(n)$ to compute the bound-consistent lower bound $\underline{\Delta}^{\mathbb{Z}}$.

Lemma 1. *The greedy + repair algorithm computes an assignment x such that $\sum_{i=1}^{n} x[i] = s$ and $\sum_{i=1}^{n} |n \cdot x[i] - s| = \underline{\Delta}^{\mathbb{Z}}$.*

Proof. It can be verified that tuple x after the greedy part until the termination of the algorithm satisfies the following invariant:

$$x \in \min_{y} \{ (\sum_{i=1}^{n} |n \cdot y[i] - s|) \;\; \Big| \;\; \sum_{i=1}^{n} y[i] = \sum_{i=1}^{n} x[i] \text{ and } \forall j : \; y[j] \in I_j^{\mathbb{Q}} \}.$$

Since each modification of x make the sum over x strictly closer to s and since the algorithm terminates whenever the sum is equal to s, the correctness follows. □

Example 3 (Computing $\underline{\Delta}^{\mathbb{Z}}$). Assume six variables with domain bounds represented on Figure 2 and given in the following table:

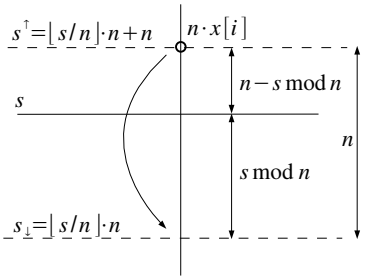

Fig. 1. Decreasing of an overlapping variable by n. The horizontal plain line represents the sum constraint s. The horizontal dashed lines are placed at $s^{\uparrow}$ and $s^{\downarrow}$.

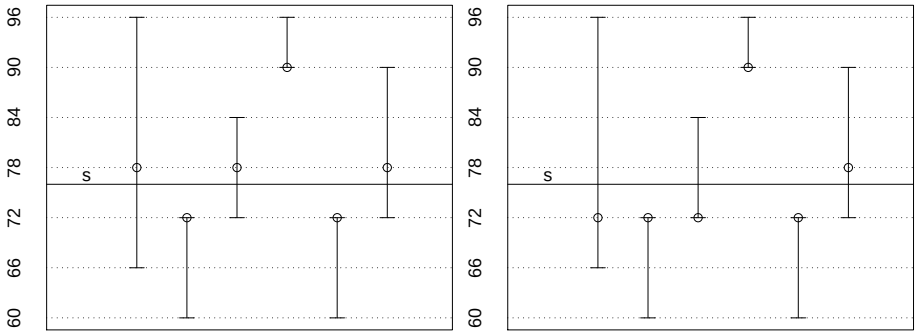

Fig. 2. Illustration of Example 3 to compute $\underline{\Delta}^{\mathbb{Z}}$. Horizontal lines represents the multiples of $n = 6$. On the left, the result of the greedy part and on the right the result of the repair part are represented with symbol $\circ$

i	1	2	3	4	5	6
$X_i^{\max}$	16	12	14	16	12	15
$X_i^{\min}$	11	10	12	15	10	12
$n \cdot X_i^{\max}$	96	72	84	96	72	90
$n \cdot X_i^{\min}$	66	60	72	90	60	72

The sum constraint is $s = 76$. After completion of the greedy part, the tuple $n \cdot x$ is equal to $(78, 72, 78, 90, 72, 78)$. An illustration of $n \cdot x$ is given on the left of Figure 2 (symbols $\circ$). For this tuple $\sum_{j=1}^{n} n \cdot x[j] = 468 > 456$. Since the sum is too high, some entries of $n \cdot x$ must be decreased. First candidates are overlapping entries $n \cdot x[1]$, $n \cdot x[3]$ and $n \cdot x[6]$. The decrease by $n = 6$ of any two of them is sufficient to satisfy the sum constraint. The right of Figure 2 shows the final tuple $n \cdot x$. The value of $\underline{\Delta}^{\mathbb{Z}}$ is then $\sum_{j=1}^{n} |n \cdot x[j] - s| = 32$.

4 Bound-consistent lower and upper bounds for X_i

This section explains how to compute $\overline{X}_i^{\mathbb{Z}}$ the maximum value in $I_i^{\mathbb{Z}}$ consistent with $\texttt{deviation}(\mathcal{X}, s, \Delta)$. Note that computing $\underline{X}_i^{\mathbb{Z}}$ is a similar problem symmetric with respect to s. The previous section gives an algorithm to find the minimum deviation in linear time. A shaving process using this algorithm can be sketched:

- Assign X_i successively to increasing values of its extended domain $I_i^{\mathbb{Z}}$.
- For each value compute $\underline{\Delta}^{\mathbb{Z}}$.
- $\overline{X}_i^{\mathbb{Z}}$ is the largest value in $I_i^{\mathbb{Z}}$ with $\underline{\Delta}^{\mathbb{Z}} \leq \Delta^{\max}$.

The complexity of this shaving procedure is $\mathcal{O}(e \cdot n)$ for X_i where e is the the size of the largest domain over $\mathcal{X}$ and $\mathcal{O}(e \cdot n^2)$ for all variables in $\mathcal{X}$.

A better algorithm is possible to lower the complexity to $\Theta(n)$. Indeed, for each variable X_i, it is possible to compute a function over the domain interval $I_i^{\mathbb{Z}}$ giving for each value the minimum deviation if X_i were assigned to that value. As shown in Subsection 4.1, this function has a simple analytical form composed of two contiguous increasing linear functions. Given this function, $\overline{X}_i^{\mathbb{Z}}$ is found in constant time by intersecting it with the horizontal line at $\Delta^{\max}$ (see Figure 3). Subsection 4.2 gives an algorithm to compute the function for every variable in $\Theta(n)$.

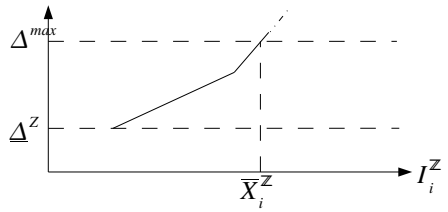

Fig. 3. Computation of $\overline{X}_i^{\mathbb{Z}}$ on basis of the minimum deviation function defined on $I_i^{\mathbb{Z}}$.

4.1 Function of the minimum deviation on $I_i^{\mathbb{Z}}$

The computation of the function giving the minimum deviation on the domain of X_i is conceptually based on any assignment m_i on $\mathcal{X}$ which maximizes the ith entry among all the assignments of minimal sum of deviations $\underline{\Delta}^{\mathbb{Z}}$:

$$m_i \in \underset{n \cdot x}{\operatorname{argmax}}\{x[i] \big| \forall j \neq i : \; x[j] \in I_j^{\mathbb{Z}} \;\text{ and }\; \sum_{j=1}^{n} |n \cdot x[j] - s| = \underline{\Delta}^{\mathbb{Z}} \;\text{ and }\; \sum_{j=1}^{n} x[j] = s\}.$$

Any assignment with the ith entry larger than $m_i[i]$ has a deviation larger than the deviation of m_i. If $m_i[i] \geq n \cdot X_i^{\max}$, then $\overline{X}_i^{\mathbb{Z}} = X_i^{\max}$. We now assume $m_i[i] < n \cdot X_i^{\max}$.

The minimum deviation function on $[m_i[i], n \cdot X_i^{\max}]$ can take different forms following the value $m_i[i]$. Three cases are possible for $m_i[i]$ given in Property 1.

Property 1.

- If $m_i[i] < s^{\downarrow}$ then $m_i[i] = n \cdot X_i^{\max}$.
- If $m_i[i] = s^{\downarrow}$ then $\forall j \neq i$: either $m_i[j] = n \cdot X_i^{\min}$ or $m_i[j] \leq s^{\downarrow}$.
- If $m_i[i] \geq s^{\uparrow}$ then $\forall j \neq i$: either $m_i[j] = n \cdot X_i^{\min}$ or $m_i[j] \leq s^{\uparrow}$.

Property 1 can be verified starting from an assignment obtained from the greedy+ repair algorithm from Section 3 and then by increasing the ith entry as much as possible while keeping the sum constraint satisfied and the deviation unchanged. Each case from Property 1 is considered in turn in the next three paragraphs giving the evolution of the minimum deviation on $I_i^{\mathbb{Z}}$ for each case.

Case $m_i[i] < s^{\downarrow}$:

In this case, $n \cdot \overline{X}_i^{\mathbb{Z}} = m_i[i]$ because the entry $m_i[i]$ cannot be increased since it is already to its maximum possible value.

Case $m_i[i] = s^{\downarrow}$:

If $m_i[i]$ is increased by n, the only entries which can be decreased are below $s^{\downarrow}$ (Property 1). Consequently when $m_i[i]$ is increased by n the deviation increases by $n - (s - s^{\downarrow}) + (s^{\uparrow} - s)$. Term n represents the decrease of an entry below $s^{\downarrow}$ and the term $-(s - s^{\downarrow}) + (s^{\uparrow} - s)$ represents the increase by n of $m_i[i]$. If $m_i[i]$ is further increased by n, the deviation increases by $2 \cdot n$. Indeed, $m_i[i] \geq s^{\uparrow}$ and the other entries candidate to be decreased are below $s^{\downarrow}$.

Example 4. This example considers 4 variables with domains given in next table:

i	1	2	3	4
$X_i^{\max}$	7	5	6	7
$X_i^{\min}$	3	0	5	5
$n \cdot X_i^{\max}$	28	20	24	28
$n \cdot X_i^{\min}$	12	0	20	20

The sum constraint is $s = 17$. Hence $s^{\downarrow} = 16$ and $s^{\uparrow} = 20$. The assignment $m_1 = (16, 12, 20, 20)$ is represented on Figure 4 with symbols $\circ$. The deviation of this assignment is 12. If $m_1[1]$ is increased by 4 that is from 16 to 20, the deviation increases by $-(s - s^{\downarrow}) + (s^{\uparrow} - s) = -1 + 3$. For the sum constraint $s = 17$ to remain satisfied, another entry must be decreased by 4. The only possible entry is $m_1[2]$ making the deviation increase by 4. The deviation of m_1 is thus increased from 12 to 18 when $m_1[1]$ is set to 20 (represented by the symbols $\triangle$ on the Figure 4). If $m_1[1]$ is further increased, the deviation is increased by $2 \cdot 4 = 8$ at every step. Hence when $m_1[1]$ is increased to 28 the deviation is 34 (represented by the symbol $\bullet$)

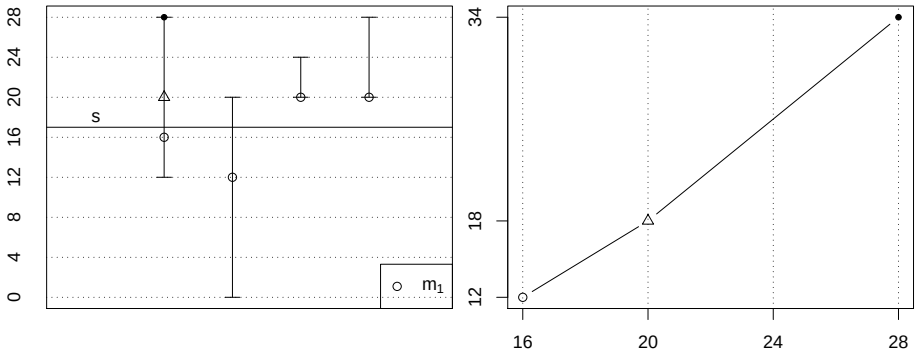

Fig. 4. Figure of Example 4. On the left is the representation of m_1 with symbols $\circ$ and the successive values of $m_1[1]$. On the right is the evolution of the deviation with the successive values of $m_1[1]$.

Case $m_i[i] \geq s^{\uparrow}$:

If $m_i[i]$ is increased by n the deviation of m_i increases by n. For the sum constraint to remain satisfied, another entry must also be decreased by n. To keep the deviation of m_i minimal, priority must be given to entries $m_i[j] = s^{\uparrow} > n \cdot X_j^{\min}$. Indeed, the decrease of such an entry induces a smaller increase in the deviation than for an entry under s. The whole effect on the deviation is an augmentation of $n - (s^{\uparrow} - s) + (s - s^{\downarrow}) = 2.(s - s^{\downarrow})$. Note that if only entries $n \cdot X_j^{\min} < m_i[j] \leq s^{\downarrow}$ are available, the deviation augments by $2 \cdot n$. This reasoning makes it possible to predict the evolution of the deviation in $\Theta(1)$ on basis of two information's about m_i:

- $m_i[i]$.
- $o_i = \#\{m_i[j] | j \neq i$ and $m_i[j] = s^{\uparrow}$ and $m_i[j] > n \cdot X_j^{\min}\}$. This number corresponds to the number of entries in m_i that can be decreased by n causing an augmentation of the deviation of only $-(s^{\uparrow} - s) + (s - s^{\downarrow})$.

The minimum deviation increases by $2 \cdot (s - s^{\downarrow})$ every n during o_i steps. After that it increases by $2 \cdot n$ every n.

Example 5. This example considers 4 variables with domains given in next table:

i	1	2	3	4
$X_i^{\max}$	10	5	6	2
$X_i^{\min}$	3	4	3	0
$n \cdot X_i^{\max}$	40	20	24	8
$n \cdot X_i^{\min}$	12	16	12	0

The sum constraint is $s = 17$. Hence $s^{\downarrow} = 16$ and $s^{\uparrow} = 20$. Assignment $m_1 = (20, 20, 20, 8)$ is represented on Figure 5 with symbol $\circ$. The deviation of this assignment is 18 and $o_1 = 2$ because of the second and third entries. The evolution of the deviation is given on the Figure 5.

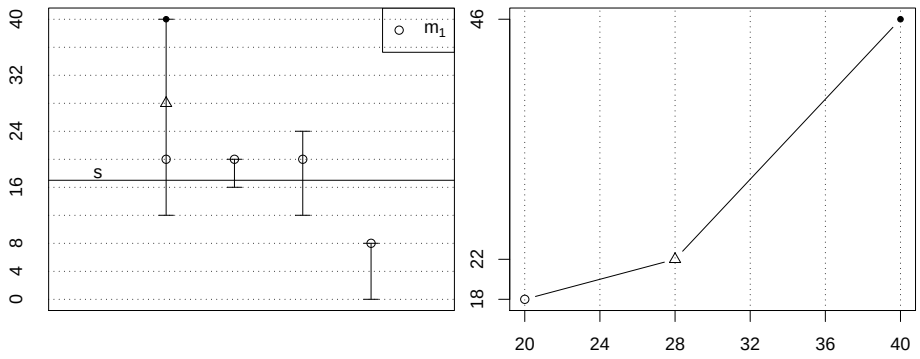

Fig. 5. Figure of Example 5. On the left is the representation of m_1 and the successive values of $m_1[1]$. On the right is the evolution of the deviation with the successive values of $m_1[1]$.

4.2 Computation of the evolution of the minimum deviation for every variable

The previous subsection explains how the minimum deviation on $I_i^{\mathbb{Z}}$ evolves starting from a special assignment called m_i. We briefly summarize the possible cases of evolution of the deviation when $m_i[i]$ is incremented by n.

- $m_i[i] < s^{\downarrow}$: The deviation can not increase anymore.
- $m_i[i] = s^{\downarrow}$: The deviation increases first by $n - (s - s^{\downarrow}) + (s^{\uparrow} - s)$ the first time $m[i]$ is increased by n. Then it increases by $2 \cdot n$ every increase by n of $m_i[i]$.
- $m_i[i] \geq s^{\uparrow}$: The deviation increases by $2 \cdot (s - s^{\downarrow})$ every n during o_i steps. After that it increases by $2 \cdot n$ every increase by n of $m_i[i]$.

The only necessary information to predict the evolution of the deviation is the entry $m_i[i]$ and the counter o_i. To simplify the notations we denote by $m[i]$ the entry $m_i[i]$ and by $o[i]$ the counter o_i. Algorithm 1 computes $m[i]$ and $o[i]$ for $1 \leq i \leq n$ in $\Theta(n)$. The algorithm assumes that the `deviation` constraint is consistent.

- Lines 4-5 do a greedy assignment for each variable multiple of n closest from s inside its domain.
- Lines 9-13 consider the case when the sum constraint is (by chance) respected after the greedy assignment.
- Lines 15-21 try to make the sum constraint satisfied by moving assignment of variables which overlap the value s.
- Lines 22-23 update the sets *overlaps* and *overlaps*$(s^{\uparrow})$ after the possible modifications in lines 15-23.
- Lines 24-31 consider the case where the sum constraint could be satisfied after modifications of lines 15-23.

Algorithm 1: Compute m and o

Data: $\mathcal{X}$ and s such that $s \in [\sum_{i=1}^{n} X_i^{\min}, \sum_{i=1}^{n} X_i^{\max}]$
Result: m and o

nx, m, o integer arrays of size n
$sum \leftarrow 0$ `/* ` $\sum_{i=1}^{n} nx[i]$ `*/`
$s^* \leftarrow (s - s^{\downarrow}) \leq (s^{\uparrow} - s)\ ?\ s^{\downarrow}\ :\ s^{\uparrow}$ `/* the multiple of` n `closest to` s `*/`
for $i \leftarrow 1$ **to** n **do**
    Set $nx[i]$ to the multiple of n closest to s in $[n \cdot X_i^{\min}, n \cdot X_i^{\max}]$
$overlaps \leftarrow \{i \mid nx[i] = s^{\uparrow} > n \cdot X_i^{\min}$ or $nx[i] = s^{\downarrow} < n \cdot X_i^{\max}\}$
$overlaps(s^{\uparrow}) \leftarrow \{i \in overlaps \mid nx[i] = s^{\uparrow}\}$
$sum \leftarrow \sum_{i=1}^{n} nx[i]$
if $sum = n \cdot s$ **then**
    for $i \leftarrow 1$ **to** n **do**
        $m[i] \leftarrow nx[i]$
        if $i \in overlaps(s^{\uparrow})$ **then** $o[i] \leftarrow \#overlaps(s^{\uparrow}) - 1$
        else $o[i] \leftarrow \#overlaps(s^{\uparrow})$
else
    if *(* $sum > n \cdot s$ and $s^* = s^{\uparrow}$ *)* or *(* $sum < n \cdot s$ and $s^* = s^{\downarrow}$ and $s^* \neq s$ *)* **then**
        $\delta \leftarrow sum > n \cdot s\ ?\ -n\ :\ n$
        for $i \in overlaps$ **do**
            if $sum = n \cdot s$ **then** break
            else
                $nx[i] \leftarrow nx[i] + \delta$
                $sum \leftarrow sum + \delta$
        $overlaps \leftarrow \{i \mid nx[i] = s^{\uparrow} > n \cdot X_i^{\min}$ or $nx[i] = s^{\downarrow} < n \cdot X_i^{\max}\}$
        $overlaps(s^{\uparrow}) \leftarrow \{i \in overlaps \mid nx[i] = s^{\uparrow}\}$
    if $sum = n \cdot s$ **then**
        for $i \leftarrow 1$ **to** n **do**
            if $i \in overlaps$ and $\#overlaps(s^{\uparrow}) > 0$ **then**
                $m[i] = s^{\uparrow}$
                $o[i] = \#overlaps(s^{\uparrow}) - 1$
            else
                $m[i] = nx[i]$
                $o[i] = \#overlaps(s^{\uparrow})$
    else if $sum > n \cdot s$ **then**
        for $i \leftarrow 1$ **to** n **do**
            $m[i] = nx[i]$
            $o[i] = 0$
    else `/* ` $sum < n \cdot s$ `*/`
        for $i \leftarrow 1$ **to** n **do**
            $m[i] = nx[i] + n \cdot s - sum$
            if $n \cdot X_i^{\min} < s < n \cdot X_i^{\max}$ and $\#overlaps(s^{\uparrow}) > 0$ **then**
                $o[i] = \#overlaps(s^{\uparrow}) - 1$
            else $o[i] = \#overlaps(s^{\uparrow})$

– Lines 32-35 and 36-41 holds respectively when the sum is too large or too low even after the modifications of lines 15-23. If the sum is too large, some entries must be decreased. It is implicitly assumed that entries $j \neq i$ can be potentially decreased. Hence $m[i] = nx[i]$ and $o[i]$ is 0 because, all other entries are already at their minimum or under s. If the sum it too small, $m[i]$ is obtained by increasing $nx[i]$ such that the sum is satisfied. If the ith entry was overlapping, $o[i]$ is the current number of overlapping entries minus one.

It can be seen that Algorithm 1 has a time complexity of $\Theta(n)$. Indeed, in all cases a constant number of operations is performed for each variable.

Example 6. This example considers the following domains:

i	1	2	3	4	5	6
$X_i^{\max}$	16	11	14	14	12	15
$X_i^{\min}$	11	9	12	13	10	12
$n \cdot X_i^{\max}$	96	66	84	84	72	90
$n \cdot X_i^{\min}$	66	54	72	78	60	72

The sum constraint is $s = 74$, $n \cdot s = 444$ and $s^* = s^{\downarrow} = 72$.

– Lines 4-8: After the greedy assignment, $nx = (72, 66, 72, 78, 72, 72)$. The sum is 432 which is smaller than 444. Hence the condition to execute lines 9-13 is not satisfied. We have also $overlaps = \{1, 3, 6\}$ and $overlaps(s^{\uparrow}) = \phi$.
– Lines 15-23 will result in $nx = (78, 66, 78, 78, 72, 72)$. The sum is now 444, $overlaps = \{1, 3, 6\}$ and $overlaps(\text{sup}) = \{1, 3\}$.
– Since $sum = n.s$ is satisfied, lines 24-32 are executed next. Entries 1, 3 and 6 satisfy the if statement line 26 while entries 2, 4 and 5 does not. Hence results are $m = (78, 66, 78, 78, 72, 78)$ and $o = (1, 2, 1, 2, 2, 1)$

5 Experimental results

This section compares the existing propagators from [7] with the presented bound-consistent propagators on the Balanced Academic Curriculum Problem (BACP). We also give an expression of the minimum difference between two deviation values and experiment the usage of this difference to speed up the Branch and Bound search. All experiments were performed on an Intel® Pentium® M 1.86GHz with 1GB of memory and with the Gecode 1.3.1 Solver.

The objective of the BACP is to assign courses to periods while balancing the workload of periods and respecting prerequisites relations between pair of courses. The CP model we consider to solve BACP is precisely the one introduced in [1]. The objective function in [1] is to minimize the maximum load over periods. In contrast, our objective function is to minimize the deviation of loads of periods from the mean load.

The search performed to solve BACP is a DFS Branch and Bound search. Hence, each time a solution is found, the next solution is constrained to have a smaller deviation. More information can be given on the next solution to be found. Indeed the smallest difference δ between two possible deviation values is

$\delta = \min\{2 \cdot s \bmod n \ , \ 2 \cdot (n - s \bmod n)\}$ when $s \bmod n \neq 0$.
$\delta = 2 \cdot n$ when $s \bmod n = 0$.

The expression $\min\{2 \cdot s \bmod n \ , \ 2 \cdot (n - s \bmod n)\}$ can be understood easily. The value $2 \cdot s \bmod n$ corresponds to the move represented by arrows labeled 1 on Figure 6 while the value $2 \cdot (n - s \bmod n)$ corresponds to the move represented with arrows labeled 2. The value δ can be used to speed-up the Branch and Bound search. Indeed, if a solution is found with a deviation of Δ, the next solution can be constrained to present a deviation less than or equal to $\Delta - \delta$.

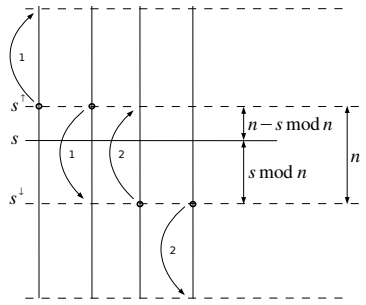

Fig. 6. Illustration of the smallest distance between two possible deviations.

Three real instances are available on CSPLIB. These instances are summarized in the following table:

n (#periods)	#courses	#prerequisites	s (total load)	$s \bmod n$
8	46	38	133	5
10	42	34	134	4
12	66	65	204	0

We report here the instances of 8 and 10 periods because the instance with 12 periods presents an integer mean load ($s \bmod n = 0$). Hence proposed propagators, as checked experimentally, behave exactly as the ones from [7] on the instance with 12 periods.

The two instances were solved with 4 configurations:

- the propagators proposed in [7] (Deviation),
- the bound-consistent propagators (BC Deviation),
- the propagators proposed in [7] with the lower bound from Theorem 1 and the value δ during the Branch and Bound search (Deviation*) and
- the bound-consistent propagators and the value δ during the Branch and Bound search (BC Deviation*).

Table 1 gives statistics on the last 5 bounds found during the Branch and Bound search for each configuration. The reported statistics are the time and the

8 Periods					8 Periods				
	Deviation		BC Deviation			Deviation*		BC Deviation*	
Bound	Time(ms)	# L.N.	Time(ms)	# L.N.	Bound	Time(ms)	# L.N.	Time(ms)	# L.N.
50	20	447	40	**441**	62	20	365	20	**320**
46	30	454	40	**443**	56	20	370	20	**325**
40	30	456	40	**445**	50	20	375	20	**330**
36	40	559	40	**450**	40	20	378	20	**332**
30	40	561	40	**451**	30	30	483	20	**337**
proof	∞	∞	40	**1517**	proof	30	1644	20	**1172**
10 Periods					10 Periods				
84	50	857	60	857	84	40	855	50	**770**
76	50	862	60	862	76	40	860	60	**822**
64	50	864	60	864	64	40	862	60	**826**
56	70	1060	60	**891**	56	60	1058	60	**853**
48	18370	368486	60	**896**	48	24660	574539	60	**858**
proof	∞	∞	60	**3288**	proof	24660	2021299	60	**3191**

Table 1. Instances of 8 and 10 periods. Statistics about the last 5 bounds during the B&Bound. The time and the number of leaf nodes (# L.N.) are given for each bound.

number of leaf nodes explored so far (# L.N.). The last line gives the statistics to prove the optimality of the last bound found.

It appears from Table 1 that the new propagators become really useful when the upper bound $\Delta^{\max}$ becomes tight. Moreover, the new propagators permit to prove optimality of the last bound within less than one second for both instances while it not possible in a reasonable time (not finished after 20 minutes) with existing propagators from [7]. The usage of the lower bound and the δ value permits to prove the optimality of the last bound with propagators from [7]. We can also see that even with the bound consistent propagators, the number of explored nodes is decreased: $1,172 < 1,517$ for 8 periods and $3,191 < 3,288$ for 10 periods.

The objective of the next experiments is to study on more instances the gain obtained with the new propagators. We generated 500 instances from the original 8 periods instance. For each instance, the weight given to each course is a random integer in [1..5] and 30 prerequisites relations are randomly chosen out the 38. Each instance was solved with the four configurations. The time limit given is of 5 seconds. Table 2 gives the number of unsolved instances.

	$s \bmod n = 0$	$s \bmod n \neq 0$
Nb instances	55	445
Deviation	0	391
BC Deviation	0	0
Deviation*	0	229
BC Deviation*	0	0

Table 2. Comparison of the propagators on 500 randomized versions of the 8 periods instance: number of unsolved instances.

It appears that all instances can be easily solved with the bound-consistent propagators. This is not the case with propagators from [7] since 301 instances remain unsolved. The use of the lower bound from Theorem 1 and the δ value permits to solve 162 additional instances with propagators from [7]. The value $s \bmod n$ has a strong influence on the old propagators but it does not influence the new propagators. All the 55 instances with $s \bmod n = 0$ could be solved with all configurations. This is not surprising since propagators proposed in [7] are also bound-consistent in this case.

6 Conclusion

The `deviation` constraint recently introduced in [7] guarantees an assignment on a set of variables to be balanced around a given mean. It constrains the set of variables to present a given mean and the deviation with respect to this mean. The main advantage of the propagators proposed in [7] is their simplicity. However, these propagators are bound-consistent only when the mean is an integer. We experiment on the Balanced Academic Curriculum Problem (BACP) that instances are very difficult to solve with propagators from [7] when this property does not hold. We give a simple lower bound on the deviation which can be found in constant time and help to solve some additional problems when the sum is not a multiple of n. Our main contributions are bound-consistent propagators for any rational mean value running in linear time. In contrast with propagators presented in [7], bound-consistent propagators solve efficiently any instance of the BACP from CSPLIB. Finally, we give a way to speedup the Branch and Bound search when the objective is to minimize the deviation.

Acknowledgment The authors wish to thank Grégoire Dooms for various interesting discussions, especially for the Theorem 1 which was the starting point of this article. Tkanks to the reviewers for their helpful comments. This research is supported by the Walloon Region, project Transmaze (516207).

References

1. Toby Walsh Brahim Hnich, Zeynep Kiziltan. Modelling a balanced academic curriculum problem. *Proceedings of CP-AI-OR*, 2002.
2. S. Manzano C. Castro. Variable and value ordering when solving balanced academic curriculum problem. *Proc. of the ERCIM WG on constraints*, 2001.
3. M.l Gendreau, J. Ferland, B. Gendron, N. Hail, B. Jaumard, S. Lapierre, G. Pesant, and P. Soriano. Physician scheduling in emergency rooms. *Proc. of PATAT*, 2006.
4. Jean-Charles Regin Gilles Pesant. Spread: A balancing constraint based on statistics. *Lecture Notes in Computer Science*, 3709:460–474, 2005.
5. C. Gomes, M. Sellmann, C. van Es, and H. van Es. The challenge of generating spatially balanced scientific experiment designs. *Proc. of CP-AI-OR*, 2004.
6. P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. Simplication and extension of spread. *3th Workshop on Constraint Propagation And Implementation*, 2006.
7. P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. The deviation constraint. *Proceedings of CP-AI-OR*, 4510:269–284, 2007.