Solving Balancing and Bin-Packing problems with Constraint Programming

Pierre Schaus

Thèse présentée en vue de l'obtention du grade de Docteur en Sciences de l'Ingénieur

August 2009

Ecole polytechnique de Louvain
Département d'Ingénierie Informatique
Université catholique de Louvain
Louvain-la-Neuve
Belgium

This work could not have been achieved without the support of the following persons.

My parents who encourage me to finish this thesis (even by coming in Providence). They are great example for their five children.

My wife Caro who supports me everyday. She's always there to remind me what is the most important in my life and helps me to find a good balance between work, family and friends.

Yves Deville my advisor who gave me many opportunities to participate in conferences and meet very interesting people. Yves, thank you also for giving responsibilities and autonomy to your Phd students and for generating such a good team spirit in the *Be-Cool* group.

Jean-Charles Régin that I met early during my thesis at the Samos summer school. From my point of view, he did one of the most influential works in constraint-programming. One year before my thesis he published a paper that kept me busy during three years. Thank you Jean-Charles for all your advices and input during this work.

Pascal Van Hentenryck that I visited during three months at Providence. It was a great honor for me to work with the father of modern constraint programming, and the creator with Laurent Michel of the most influential optimization systems. This was a wonderful visit from the scientific and also from the relational point of view. I really appreciated how you made my stay at Providence enjoyable even if I was far from Caro and Tom.

Jean-Noël Monette which is my colleague since the beginning. I could not have dreamed of a better person to share my office: he can solve your technical problems because he is very smart and he is always ready to take a break with you. I hope we will have the opportunity to work together for many years. Thank you also to Grégoire Dooms and Stéphane Zampelli that were my models when I started with Jean-Noël.

The member of the jury who accepted to read and comment this thesis. Especially Yves Crama that noticed that some problems solved with dedicated algorithms in this thesis can be seen as minimization of separable convex functions that can be solved with well known algorithms. This remark will probably give new perspectives to extend this work.

Finally I want to thank the *Be-Cool* members, the colleagues of the 3d floor, the *free-runners* of Ingi, Stéphanie Landrain and all other people that I forgot for the pleasant years spent at the INGI department.

Contents

1 Introduction

Constraint Programming (CP) can be summarized with the famous equation[1]:

$$CP = modeling + search.$$

A problem is modeled in a declarative language by stating the variables and the constraints, then a solution is found by exploring a search tree with a backtracking algorithm until a solution is found. An alternative definition of CP [2] that illustrates better the content of this thesis is:

$$CP = filtering + search.$$

Constraints act at each node of the search tree by removing inconsistent values from the domain (act of filtering), pruning the search tree when a domain becomes empty. Constraints only communicate through the domain of the variables (the domain store). Constraint programming can be seen as a decomposing technique where constraints encapsulate sub-problems that can be solved efficiently. This approach to solve combinatorial problems has two advantages:

1. re-usability: constraints are building blocks that can be used in various problems, and
2. flexibility: very efficient OR algorithms (e.g. network flows) can be integrated into the constraints to solve large sub-problems very efficiently.

[1] I think from Pascal Van Hentenryck

[2] This one is from Jean-Charles Régin

This work focus on the filtering algorithm for two types of constraints:

1. Balancing or Fairness Constraints working on the variance and the mean absolute deviation of a vector of variables with a fixed mean, and
2. Bin-Packing Constraints linking the placement of sized/weighted items into bins and the capacity of the bins.

Contributions

Most of the constraints that we study are not new and can be expressed in most of the existing CP solvers by decomposing them into smaller constraints. The contributions of this work are on the filtering algorithms of these constraints. We improve the existing filtering functions of these constraints by making global reasoning rather than letting the system work separately on the decomposition[3].

New Filtering Algorithm:

- Balancing Constraints: When starting this thesis the first paper on balancing constraints introducing `spread` [31] was just published. This constraint considers a variable mean and restricts the variance of a set of variables. It was not implemented and while the filtering of the variance variable was correct, we suspect that the description of the filtering on the set of variables has some problems. We restrict here our attention on the particular case of a fixed mean having many practical applications. This simplification allows us to design and implement for the first time a reasonably simple filtering algorithm for `spread`. We also introduce a new balancing constraint (`deviation`) similar to `spread` but for the mean absolution deviation rather than the variance. This constraint has the advantage to simplify even more the filtering algorithms compared to `spread`.
- Bin-Packing Constraints: We improve a failure detection algorithm introduced by Paul Shaw [52]. We also introduce some preliminary new ideas to filter bin-packing constraints using network

[3]This idea is also not new, the most famous example being the filtering algorithm for the `allDifferent` constraint [37].

flows. Finally we extend the Bin-Packing constraint with a set of precedence relations between items that must be satisfied. We introduce a strong formulation using redundant constraints for this problem as well as a global filtering algorithm.

New Bound-Consistency Notions: Another contribution of this work is a better characterization of the classical bound-consistency notion in CP. We distinguish the $\mathbb{Q}$-bound-consistency and the $\mathbb{Z}$-bound-consistency following that values can take rational or integer values in the relaxation. For constraints like `allDifferent` which are discrete by nature this differentiation has no reason to be. But for `spread` and `deviation` the question arises and the filtering algorithms designed for these two consistencies are quite different and lead to different strengths of filtering in theory and in practice.

Modeling of new problems in CP: Efficient CP models are introduced to solve problems that were not approached by constraint programming:

1. the assembly line balancing problems and the
2. the nurse assignment balancing problem.
3. the soft-distribute constraint, a soft-version of a particular case of the global cardinality constraint with equal lower and upper bounds for the cardinality of each value.

Publications: The content of this thesis led to some publications accepted in refereed international conferences:

- P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. Simplification and extension of the spread constraint. In *Future and Trends of Constraint Programming*, pages 95–99. ISTE, 2007
- P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. The deviation constraint. In *4th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 4510 of *Lecture Notes in Computer Science*, pages 269–284, Brussels, Belgium, 2007. Springer

- P. Schaus, Y. Deville, and P. Dupont. Bound-consistent deviation constraint. In *13th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4741 of *Lecture Notes in Computer Science*, pages 620–634, Providence, RI, USA, September 2007. Springer

- P. Schaus and Y. Deville. A global constraint for bin-packing with precedences: Application to the assembly line balancing problem. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 369–374, Chicago, Illinois, USA, July 2008. AAAI Press

- P. Schaus, P. Van Hentenryck, and J.C. Régin. Scalable load balancing in nurse to patient assignment problems. In *6th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, Lecture Notes in Computer Science, Pittsburgh, Pennsylvania, USA, 2009. Springer

2
CP Background

The chapter introduces Constraint Programming preliminaries useful for the reading of the subsequent chapters.

2.1 Variables, Domains and Constraints

Let X be a finite-domain (discrete) *variable*. The *domain* of X is a set of ordered values that can be assigned to X and is denoted by $Dom(X)$. The minimum (resp. maximum) value of the domain is denoted by $X^{\min} = \min(Dom(X))$ (resp. $X^{\max} = \max(Dom(X))$. 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 $\mathbf{X} = [X_1, X_2, ..., X_n]$ is denoted by the tuple $\mathbf{x}$ and the ith entry of this tuple by $\mathbf{x}[i]$. The extended rational interval domain of X_i is $I_D^{\mathbb{Q}}(X_i) = [X_i^{\min}, X_i^{\max}]$ and its integer interval domain is $I_D^{\mathbb{Z}}(X_i) = [X_i^{\min} .. X_i^{\max}]$.

Let $\mathbf{X} = [X_1, X_2, ..., X_k]$ be a sequence of variables. A *constraint* C on $\mathbf{X}$ is defined as a subset of the Cartesian product of the domains of the variables in $\mathbf{X}$: $C \subseteq Dom(X_1) \times Dom(X_2) \times ... \times Dom(X_k)$. A tuple $\mathbf{x} = (x_1, ..., x_k) \in C$ is called a *solution* to C. A value $v \in Dom(X_i)$ for some $i = 1, ..., k$ is *inconsistent* with respect to C if it does not belong to a tuple of C, otherwise it is *consistent*. C is inconsistent if it does not contain a solution. Otherwise, C is called consistent.

A constraint satisfaction problem, or a *CSP*, is defined by a finite sequence of variables $\mathbf{X} = [X_1, X_2, ..., X_n]$, together with a finite set of constraint $\mathcal{C}$ each on a subset of $\mathbf{X}$. The goal is to find an *assignment*

$X_i := v$ with $v \in Dom(X_i)$ for $i = 1, ..., n$, such that all constraints are satisfied. This assignment is called a solution to the CSP.

2.2 The Search and Pruning Process

The solution process of constraint programming interleaves constraint *propagation*, or propagation in short, and search. The search process essentially consists of an enumeration of all possible variable-value combinations, until a solution is found or it is proved that none exists. We say that this process constructs a search tree. To reduce the exponential number of combinations, constraint propagation is applied to each node of the search tree: Given the current domains and a constraint C, the propagator for C removes domain values that do not belong to a solution to C. This is repeated for all constraints until no more domain value can be removed (*fix point*). The removal of inconsistent domain values is called *filtering*.

It is also possible to make optimization in Constraint Programming by minimizing or maximizing one variable of the problem. The classical Branch and Bound (B&B) scheme is applied to constraint programming. When a solution is found during the search process, a constraint is dynamically added to the problem constraining the next solution to be better. The last solution found when the search is completed is then proved to be optimal.

2.3 Classical Consistency Notions

In order to be effective, filtering algorithms should be efficient, because they are applied many times during the search process. They should furthermore remove as many inconsistent values as possible. If a filtering algorithm for a constraint C removes all inconsistent values from the domains with respect to C, we say that it makes C *domain-consistent*[1]:

Definition 1 (domain-consistency)**.** A constraint $C(X_1, \ldots, X_n) \subseteq Dom(X_1) \times \ldots \times Dom(X_n)$ $(n > 1)$ is domain-consistent if for all $i \in \{1, \ldots, n\}$ and every value $v_i \in Dom(X_i)$, there exists a tuple $(x_1, \ldots, x_n) \in C$ such that $x_i = v_i$.

[1]also called (Generalized) Arc Consistency (GAC)

It is possible to achieve domain-consistency in polynomial time for some constraints such as `AllDiff` but for other constraints such as `SUM` this would be too costly. In such cases a weaker notion of consistency is used such as the bound-consistency:

Definition 2 (bound-consistency)**.** A constraint $C(X_1, ..., X_n)$ $(n > 1)$ is bound-consistent with respect to domains $Dom(X_i)$ if for all $i \in \{1, ..., n\}$ and each value $v_i \in \{X_i^{\min}, X_i^{\max}\}$, there exist values $X_j^{\min} \leq v_j \leq X_j^{\max}$ for all $j \in \{1, ..., n\} - i$ such that $(v_1, ..., v_n) \in C$.

The idea of the bound-consistency is to bound the domain of each variable by an interval and make sure that the end-points of the intervals obey the domain-consistency requirement. If not, the upper and lower bounds of the intervals can be tightened until bounds-consistency is achieved.

2.4 Set Variables

Set variables in CP [11] allow to represent a set rather than a single value. The classical representation of the domain of a set variable $\mathcal{S}$ is represented by two sets $\underline{\mathcal{S}}$ and $\overline{\mathcal{S}}$ with $\underline{\mathcal{S}} \subseteq \overline{\mathcal{S}}$ (subset-bound representation [11]). The lower bound $\underline{\mathcal{S}}$ represents the values that must be in the set, and $\overline{\mathcal{S}}$ represents the values that may figure in the set; *i.e.* $\underline{\mathcal{S}} \subseteq \mathcal{S} \subseteq \overline{\mathcal{S}}$. The subset-bound representation is generally enhanced with a finite domain variable representing the cardinality of the set.

Note that recently, an other promising representation of set domains have been proposed in [13, 17] but it is not yet available in any solver. We refer to [12] for more information about different structured domains that have been proposed in CP.

2.5 Redundant Constraints

Redundant constraints (also called implied or surrogate constraints) are constraints which are implied by the constraints defining the problem. They do not change the set of solutions, and hence are logically redundant. Adding redundant constraints to the model may however further reduce the search space by allowing more pruning either by

- expressing a property of the solutions of problem that doesn't appear in the formulation, or by

- improving the communication between existing constraints of the problem.

Redundant constraints have been successfully applied to various problems such as car sequencing [8]. We refer to [54] for more examples of successful applications using redundant constraints.

3
Balancing Constraints

3.1 Introduction

Balancing constraints appears to be useful in many real applications especially when humans are involved. One often want the actors of the application to be treated equally. For example, in assignment problems, solutions with the workloads distributed fairly should be preferred. In rostering problems, the bad shifts (e.g. night or weekend shifts) should be evenly distributed. A non exhaustive list of applications and problems where the balance property is important is given next.

- The Balanced Academic Curriculum Problem (BACP)[1]: The goal is to assign a period to each course in a way that the prerequisite relationships are satisfied and *the academic load of each period is balanced*. It is quite natural to ask a good repartition of work among the periods of the curriculum such that there is no periods with a huge peak of work and periods with almost nothing to do. This last constraint makes BACP a Constraint Optimization Problem where the objective is to maximize the balancing property.

- The (Vertical) Assembly Line Balancing Problem (ALBP)[2]. A given number of workstations are placed along a conveyor belt.

[1]Problem 30 of CSPLIB (www.csplib.org)

[2]This problem has several variants, see [4] for a detailed classification of all the variants

The workpieces are consecutively launched down the line from station to station until the end of the line. Some operations are performed on any workpieces in each station. The problem is to assign all the operations to the workstations such that the workload of the stations is nearly the same while satisfying various constraints such as precedences between the operations [4]. The balance requirement has a double objective here. First it allows to optimize the speed of the line. Secondly, if a station represents a set of tasks achieved by an operator (manual assembly line), the total work of each station should be balanced such that all the operators have almost the same quantity of work.

- Assigning patients to nurses [30]: This problem is to assign group of infants throughout the shift. The large variation in infant conditions along with several complicating side constraints makes it difficult to develop balanced nurse work loads. Distributing work fairly among nurses is essential for optimal quality of care.

- Balancing the assignment of customers among employees [21]: To handle the orders, a mail order firm assigns its customer to employees such that customers having last names beginning with the same letter are assigned to the same employee to ease the tracking of orders. To be fair from an employee point of view and to minimize the deviation of the processing times of customers, the management wants the number of customers assigned to each employee to be as close as possible to the average number of customers per assignments.

- Nurse rostering [53]: Nurses must be allocated to morning, afternoon and night shifts while satisfying the staffing requirements and various constraints such as the personnel work rules and administrative requirements. As explained by Simonis, balancing the workload between persons or within the schedule for a single person can have a big impact on the perceived quality of the solution.

- Generating Spatially Balanced Scientific Experiment Designs [14]: This problem arises in the design of scientific experiments. For example, in agronomic field experiments, one has to test and compare different soil treatments. Gomes et al. shows that a good design can be given by Latin square which is spatially balanced.

- The completion time variance problem [29] is to find a sequencing of tasks minimizing the sum of the absolute (squared) deviations of the completion times to the average completion time of the set of tasks. This problem is applicable to any service or manufacturing setting where uniformity of services is desired.
- Softening of cardinality constraint: Assume you have n objects to allocate to m resources and you have an idea of the ideal repartition. This ideal repartition might be a preference rather than a hard constraint. A solution to model this situation is to use a global cardinality constraint [34] $\texttt{gcc}([D_1,\ldots,D_m],[X_1,\ldots,X_n])$ where X_i is the resource allocated to object i and D_j is the number of objects allocated to resource j. The desired value for D_j is δ_j and we have that $\sum_{j=1}^{n}\delta_j = n$ to have a valid preference distribution. We introduce variables $Y_j = D_j - \delta_j$. We have the redundant constraint that $\sum_j Y_j = 0$ and we wish that the violation variables Y_i to be well balanced around 0.
- Balancing the violations among soft constraints in over-constrained problems: Most real constrained problems are over-constrained. Petit et al. suggest in [32] a framework where the violation of each constraint is represented by a violation variable. They also express that the violations must be homogeneously distributed.

3.2 Measuring the balance

For all these problems, the perfect balance is in general not possible. In the BACP the overall load of s is known since the loads of each course are part of the description of the problem. A hard balancing constraint would impose all periods to take a same load s/n if n is the number of periods in the curriculum. This often results in an over-constrained problem without solution. One possibility is to relax the hard balancing constraint with respect to some violation measure.

For a set of variables $\mathbf{X} = [X_1, X_2, ..., X_n]$ and a given fixed sum s, a violation measure of the perfect balance property can be defined as the L_p-norm of the vector $[\mathbf{X} - \mathbf{s/n}]$ with $\mathbf{s/n} = [s/n, s/n, ..., s/n]$ such that $\sum_{i=1}^{n} X_i = s$. The L_p-norm of $[\mathbf{X}-\mathbf{s/n}]$ is defined as $(\sum_{i=1}^{n}|X_i - s/n|^p)^{\frac{1}{p}}$ with $p \geq 0$.

Following the scheme proposed by Régin et al. [36] to soften global constraints, we define a violation of the perfect balance constraint as a cost variable L_p in the global balance constraint. The constraint $\texttt{soft-balance}(\mathbf{X}, s, L_p)$ holds if and only if L_p-norm$([\mathbf{X} - \mathbf{s/n}]) = L_p$ and $\sum_{i=1}^{n} X_i = s$.

The interpretation of the violation to the mean for some specific norms is given below.

- L_0: $|\{X_i | i \in [1..n] \wedge X_i \neq s/n\}|$ is the number of values different from the mean.
- L_1: $\sum_{i\in[1..n]} |X_i - s/n|$ is the sum of deviations from the mean.
- L_2: $\sum_{i\in[1..n]} (X_i - s/n)^2$ is the sum of square deviations from the mean.
- L_∞: $\max_{i\in[1..n]} |X_i - s/n|$ is the maximum deviation from the mean.

None of these balance criteria subsumes the others. For instance, the minimization of L_1 does not imply in general a minimization of criterion L_2. This is illustrated on the following example. Assume a constraint problem with four solutions given in Table 3.1. The most balanced solution depends on the chosen norm. Each solution exhibits a mean of 100 but each one optimizes a different norm.

sol. num.	solution	L_0	L_1	L_2	L_∞
1	100 100 100 100 30 170	**2**	140	9800	70
2	60 80 100 100 120 140	4	**120**	4000	40
3	70 70 90 110 130 130	6	140	**3800**	30
4	71 71 71 129 129 129	6	174	5046	**29**

Table 3.1: Illustration showing that no balance criterion defined by the norm L_0, L_1, L_2 or L_∞ subsumes the others. The smallest norm is indicated in bold character. For example, solution 2 is the most balanced according to L_1.

They are several alternatives to the L_p norm to enforce a balance property. One can also minimize:

- the maximum value: $\max_{i\in[1..n]}\{X_i\}$ or

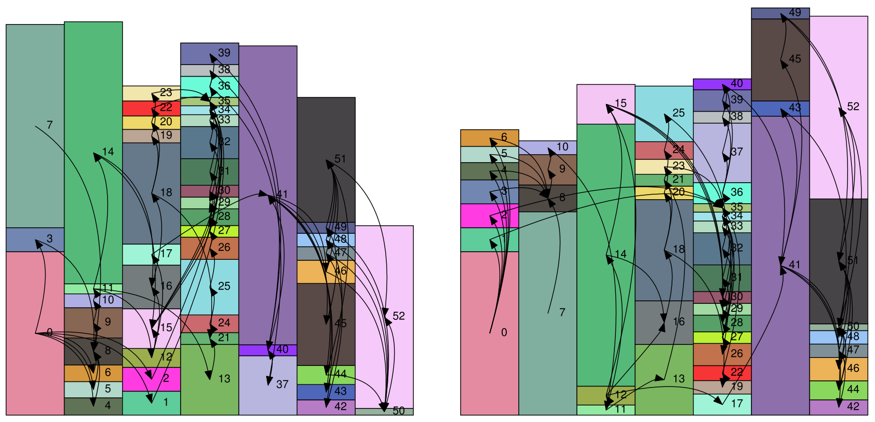


Figure 3.1: minimization of the maximum load

- the maximum range: $\max_{i\in[1..n]}\{X_i\} - \min_{i\in[1..n]}\{X_i\}$.

The minimization of the maximum value is popular to solve the BACP [5, 18] or the SLBP [22]. Unfortunately this can result in very poor quality solutions from the point of view of L_1 or L_2. The Figure 3.1 illustrates this on the instance Hahn with 7 workstations from the benchmark data set of Scholl [48]. The left solution is obtained by minimization of the maximum workload and the right solution by minimization of the L_1 criterion. The maximum value of the left solution is 2336 against 2418 for the right solution and the mean absolute deviation is 298 against 222. On this instance we have a degradation of 3.5% from the point of view of the maximum value criterion but the L_1 criterion is improved by 25.5%.

A constraint for L_0 can easily be implemented using an `atleast`$(i, [X_1, ..., X_n], s/n)$ constraint for $|\{X \in \mathcal{X} | X \neq s/n\}| \leq i$ and a `sum`$([X_1, ..., X_n], s)$ constraint to ensure a sum of s. In statistics, the usual indices to measure the balance of a set of data is either the standard deviation which corresponds to the L_2 criterion or the mean absolute deviation (MAD) corresponding the L_1 criterion. Which one should we prefer ? This old debate is certainly not closed (see for example [15]). The least we can say is that L_2 is more sensitive to outliers than L_1. On practical problems like the simple assembly line balancing problems some studies try to optimize the L_1 criterion [33, 27] and others the L_2 criterion [38]. At the best of our knowledge, neither of these works use exact methods to optimize the balancing with respect to L_1 or L_2.

Section 3.3 presents the related works in CP on balancing constraints. Section 3.4 presents a global constraint and its propagators for the L_2 criterion called `spread`. Then Section 3.5 presents a global constraint and is propagators for the L_1 norm called `deviation`. We study thoroughly in these two sections the `spread` and `deviation` constraints with a fixed mean (or fixed sum). It was not a limitation for the applications we have encountered during this thesis since for every of them the sum is known. For example, in assembly line balancing problems, the total work to be distributed among workstation is given (sum of the durations of the tasks). In rostering problems, the number of shifts to distribute is also given. In the patient to nurses assignment problem, the amount of work for each infant is also known [30].

3.3 Related Work

Balancing Global Constraints: The pioneering work in CP on balancing constraints is the introduction of the `spread` constraint in [31] by Pesant and Régin. It constrains the variance and the mean over a set of variables. The filtering algorithm of the variance variable in [31] was the basis of all the other filtering algorithms that have been developed for `spread` in this work. We are not aware of other works in CP on balancing constraints and their filtering.

Soft Global Constraints: Many real problems are over-constrained. To solve these problems with exact techniques like CP, some constraints need to be relaxed. Petit et al. suggest in [32] a framework where the violation of each constraint is represented by a violation variable. In particular several filtering algorithms of soft global constraints have been suggested (for instance `soft-alldifferent` or the `soft-gcc`, `soft-regular` ... [57, 58]. We consider that balancing constraints fit in this framework since they are soft versions of the perfect balance in which every variable takes a same value.

Resource Allocation Problems: Optimizing the balance with a given sum value can be expressed as

$$\min \sum_{i=1}^{n} f_i(X_i)$$
$$\text{such that : } \sum_{i=1}^{n} X_i = s$$

where $f_i(X_i)$ is a function that measures the violation for variable X_i of not being equal to the mean s/n. These kind of problems are known as resource allocation problems [19] and are well solved when the functions f_i are convex and differentiable.

3.4 Variance Constraint

A global constraint for the variance was first introduced in [31]. Simplifications and extensions of the propagators for `spread` were further developed in [42].

Definition 3. Given finite domain variables $\mathbf{X} = (X_1, X_2, ..., X_n)$, an integer value s and a finite domain variable Δ, $\texttt{spread}(\mathbf{X}, s, \Delta)$ holds if and only if

$$\sum_{i\in[1..n]} X_i = s \quad \text{and} \quad \Delta \geq n \cdot \sum_{i\in[1..n]} |X_i - s/n|^2.$$

A convenient way to compute the sum of square deviations often used in the remaining is

$$n \cdot \sum_{i\in[1..n]} |X_i - s/n|^2 = n \cdot \sum_{i\in[1..n]} X_i^2 - s^2. \tag{3.1}$$

As s is integer, this quantity is integer. This is why it is more convenient to work with $n \cdot \sum_{i\in[1..n]} |X_i - s/n|^2$ than with $\sum_{i\in[1..n]} |X_i - s/n|^2$.

Example 1. Tuple $\mathtt{x} = (4, 6, 2, 5) \in \texttt{spread}([X_1, X_2, X_3, X_4], s = 17, \Delta = 40)$ but $\mathtt{x} = (3, 6, 2, 6) \notin \texttt{spread}([X_1, X_2, X_3, X_4], s = 17, \Delta = 40)$ because $4 \cdot (3^2 + 6^2 + 2^2 + 6^2) - 17^2 = 51 > 40$.

Three questions can occur at this point:

1. What level of consistency should be used to filter `spread`?

2. Why do we use $\Delta \geq n \cdot \sum_{i\in[1..n]} |X_i - s/n|^2$ instead of $\Delta = n \cdot \sum_{i\in[1..n]} |X_i - s/n|^2$ in Definition 3?
3. Why is a global constraint necessary if actually $\texttt{spread}(\mathbf{X}, s, \Delta)$ can be decomposed into two simple arithmetic constraints as suggested by its Definition 3?

Which consistency for `spread`?

Achieving the domain consistency for `spread` would be NP-complete. Indeed a particular case of `spread` is the `sum` constraint which is well known to be NP-complete for the domain consistency. The natural choice is then the bound-consistency filtering. As shown in next example, the bound-consistency definition (see Definition 2) let-us some flexibility of interpretation:

Example 2. Assume 4 variables $X_1, \ldots, X_4$ all with domain $\{0, 1\}$. Consider a `spread` constraint with parameters $\texttt{spread}([X_1, \ldots, X_4], s = 2, \Delta \in [0.. + \infty])$. Clearly only a an assignment with two 0 and two 1 can achieve a sum of 2. It means that Δ can be fixed to $4 * 2 - 4 = 4$. But if we forget the integrality requirement (which is allowed by the definition), the solution $X_1 = X_2 = X_3 = X_4 = 0.5$ is also valid tuple of `spread` and hence Δ can only be restricted to the interval $[0, 4]$.

Since both interpretations are possible because of the sentence *"there exist values $X_j^{\min} \leq v_j \leq X_j^{\max}$ for all $j \in \{1, ..., n\} - i$ such that $(v_1, ..., v_n) \in C$"* from Definition 2, we distinguish the integer case from the rational case in the next definition:

Definition 4 ($\mathbb{Q}$-bound-consistency and $\mathbb{Z}$-bound-consistency)**.** A constraint $C(X_1, ..., X_n)$ $(n > 1)$ is $\mathbb{Q}$-bound-consistent (resp. $\mathbb{Z}$-bound-consistent) with respect to domains $Dom(X_i)$ if for all $i \in \{1, ..., n\}$ and each value $v_i \in \{X_i^{\min}, X_i^{\max}\}$, there exist values $v_j \in I_D^{\mathbb{Q}}(X_j)$ (resp. $v_j \in I_D^{\mathbb{Z}}(X_j)$) for all $j \in \{1, ..., n\} - i$ such that $(v_1, ..., v_n) \in C$.

The idea of the $\mathbb{Q}$-bound-consistency (resp. $\mathbb{Z}$-bound-consistency) is to bound the domain of each variable by an rational (resp. integer) interval and make sure that the end-points of the intervals obey the domain-consistency requirement. If not, the upper and lower bounds of the intervals can be tightened until bounds-consistency is achieved.

We present filtering algorithms for `spread` for both $\mathbb{Q}$ and $\mathbb{Z}$-bound-consistencies.

Why $\Delta \geq n \cdot \sum_{i \in [1..n]} |X_i - s/n|^2$ instead of $\Delta = n \cdot \sum_{i \in [1..n]} |X_i - s/n|^2$?

The answer to this question is simply that it is an NP-Hard problem to achieve a bound consistent filtering on $\Delta^{\max}$. Filtering $\Delta^{\max}$ requires to solve the following optimization problem which is shown to be an NP-Hard in Theorem 1.

$$\overline{\Delta}^{\mathbb{Q}} = \max_{\mathbf{x}} \{ n \cdot \sum_{i \in [1..n]} (\mathbf{x}[i] - s/n)^2 \quad \text{s.t.} \quad \sum_{i \in [1..n]} \mathbf{x}[i] = s \quad \text{and} \quad \forall i \in [1..n] : \mathbf{x}[i] \in I_D^{\mathbb{Q}}(X_i) \} \tag{3.2}$$

Theorem 1. Computing $\overline{\Delta}^{\mathbb{Q}}$ is NP-Hard.

Proof. It is possible to reduce the subset sum problem [10] to the problem of computing $\overline{\Delta}^{\mathbb{Q}}$. This problem is not more difficult than the particular case where $s = 0$:

$$\overline{\Delta}^{\mathbb{Q}} = \max \sum_{i=1}^{n} |n \cdot X_i|$$
$$\text{such that}: \sum_{i=1}^{n} X_i = 0$$
$$X_i \in I_D^{\mathbb{Q}}(X_i),\ 1 \leq i \leq n$$

Given a set of n positive values $\{b_1, ..., b_{n-1}, T\}$, the subset sum problem consists in finding if there exists a set of binary values $\{y_1, ..., y_{n-1}\}, y_i \in \{0,1\}, 1 \leq i < n$ such that $\sum_{i=1}^{n-1} y_i.b_i = T$. The reduction is the following:

- $X_i^{\min} = -\frac{b_i}{2}$ and $X_i^{\max} = \frac{b_i}{2}$ for $1 \leq i < n$.
- $X_n = \frac{\sum_{i=1}^{n-1} b_i}{2} - T$.
- There is a solution to the subset sum problem if and only if $\overline{\Delta}^{\mathbb{Q}} \geq n \cdot \sum_{i=1}^{n-1} \left(\frac{b_i}{2}\right)^2 + n \cdot \left(\frac{\sum_{i=1}^{n-1} b_i}{2} - T\right)^2$. This constraint on the optimal value ensures that the optimal solution is such that $X_i \in \{-\frac{b_i}{2}, \frac{b_i}{2}\}$. The solution to the subset sum problem is then given by $y_i = 1$ if $X_i = \frac{b_i}{2}$ and $y_i = 0$ if $X_i = \frac{-b_i}{2}$.

□

Note that filtering $\Delta^{\max}$ is not really important since we are generally interested in minimizing the variance instead of maximizing it. Hence only a good lower bound is required.

Why is a global filtering for `spread` more efficient than its decomposition?

The necessity of a global filtering rather than a decomposition is illustrated with a two variable example on Figure 3.2. Assume two variables X_1, X_2 with unbounded finite domains and the constraint

$$\texttt{spread}([X_1, X_2], s, \Delta \in [0, \Delta^{\max}]).$$

The shaded circle on Figure 3.2 delimits the set of points such that $2(X_1 - s/n)^2 + 2(X_2 - s/n)^2 \leq \Delta^{\max}$. The diagonal line is the set of points such that $X_1 + X_2 = s$. The unbounded domains for X_1 and X_2 are bound-consistent for the mean constraint. The vertically shaded rectangle defines the domain of X_1 after a bound-consistent filtering for $2(X_1 - s/n)^2 + 2(X_2 - s/n)^2 \leq \Delta^{\max}$. The set of solutions for `spread` is the bold diagonal segment obtained by intersecting the circle surface and the diagonal line. It can be seen on the figure that more filtering is possible. The $\mathbb{Q}$-bound-consistent filtering on the bold diagonal segment leads to a domain for X_1 defined by the diagonally shaded rectangle. In conclusion a $\mathbb{Q}$-bound-consistent filtering for the decomposition leads to $Dom(X_1) = Dom(X_2) = [s - \sqrt{\Delta^{\max}/2}, s + \sqrt{\Delta^{\max}/2}]$ while a $\mathbb{Q}$-bound-consistent filtering for $\texttt{spread}([X_1, X_2], s, \Delta \in [0, \Delta^{\max}])$ leads to domains $Dom(X_1) = Dom(X_2) = [s - \sqrt{\Delta^{\max}}/2, s + \sqrt{\Delta^{\max}}/2]$.

Two propagators can be imagined for the `spread` constraint: Increasing of $\Delta^{\min}$ given domains of variables in $\mathbf{X}$ and the value s, and narrowing of $Dom(X_i)$ given the values $\Delta^{\max}$, s, and the domains $Dom(X_j)$ for $j \in \{1, .., n\} - i$.

3.4.1 Filtering of $\Delta^{\min}$

The filtering of $\Delta^{\min}$ requires to find a good lower bound for it given domains of variables in $\mathbf{X}$ and the value s. Régin and Pesant explain in

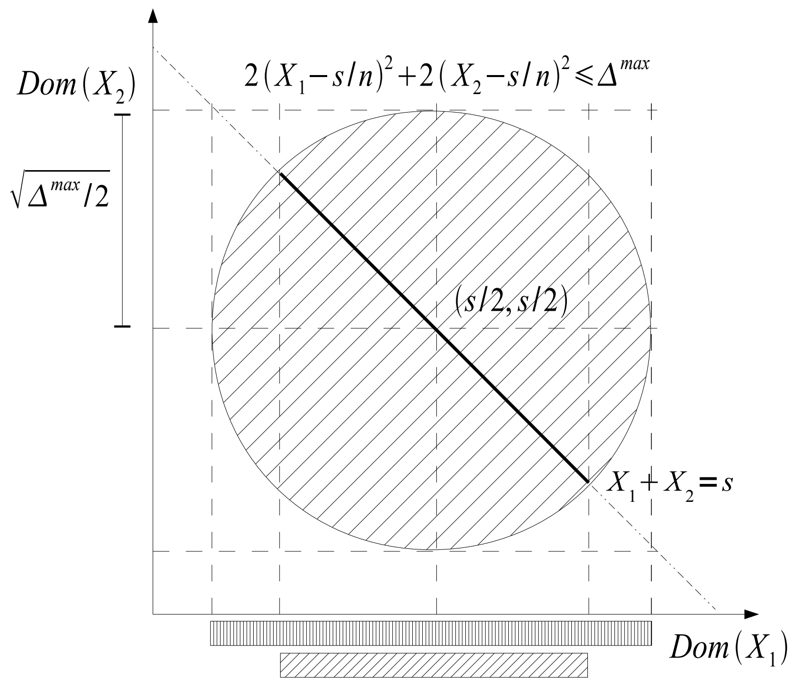


Figure 3.2: Comparison of the global filtering for `spread` with the decomposition for the two variable case.

[31] how to solve the minimization problem:

$$\underline{\Delta}^{\mathbb{Q}} = \min_{\mathbf{x}}\{n \cdot \sum_{i\in[1..n]} (\mathbf{x}[i] - s/n)^2 \quad \text{s.t.} \quad \sum_{i\in[1..n]} \mathbf{x}[i] = s \quad \text{and} \quad \forall i \in [1..n] : \mathbf{x}[i] \in I_D^{\mathbb{Q}}(X_i)\} \tag{3.3}$$

Their algorithm consider that the domains of $\mathbf{X}$ are rational intervals. This lower bound can be strengthened by considering integer intervals:

$$\underline{\Delta}^{\mathbb{Z}} = \min_{\mathbf{x}}\{n \cdot \sum_{i\in[1..n]} (\mathbf{x}[i] - s/n)^2 \quad \text{s.t.} \quad \sum_{i\in[1..n]} \mathbf{x}[i] = s \quad \text{and} \quad \forall i \in [1..n] : \mathbf{x}[i] \in I_D^{\mathbb{Z}}(X_i)\} \tag{3.4}$$

Since $\underline{\Delta}^{\mathbb{Z}} \geq \lceil \underline{\Delta}^{\mathbb{Q}} \rceil$, the filtering rule is $\Delta^{\min} \leftarrow \max(\Delta^{\min}, \underline{\Delta}^{\mathbb{Z}})$. Next example illustrates the difference between $\underline{\Delta}^{\mathbb{Z}}$ and $\underline{\Delta}^{\mathbb{Q}}$.

Example 3. Assume two variables $\mathbf{X} = (X_1, X_2)$ with domains $[-5..5]$ and a sum constraint $s = 1$. Obviously $\underline{\Delta}^{\mathbb{Q}} = 0$ is obtained with the tuple $\mathbf{x} = (0.5, 0.5)$ while $\underline{\Delta}^{\mathbb{Z}} = 1$ is obtained with the tuple $\mathbf{x} = (1, 0)$ or $\mathbf{x} = (0, 1)$.

First we explain in details the $O(n \cdot log(n))$ algorithm from [31] to compute $\underline{\Delta}^{\mathbb{Q}}$ then we adapt this algorithm to compute $\underline{\Delta}^{\mathbb{Z}}$.

The key point of the algorithm given in the next theorem is a clear characterization of an optimal solution to the problem of minimization of the sum of square deviations with a given sum. This theorem says that in an assignment of sum s minimizing the sum of square deviations, there cannot exist two values that can be made closer by moving both of them inside their corresponding domains.

Theorem 2. $\mathbf{x} \in \text{argmin}_{\mathbf{y}}\{n \cdot \sum_{i\in[1..n]}(\mathbf{y}[i] - \frac{s}{n})^2$ s.t. $\sum_{i\in[1..n]} \mathbf{y}[i] = s$ and $\forall i : \mathbf{y}[i] \in I_D^{\mathbb{Q}}(X_i)\} \Leftrightarrow \nexists\ (i, j)$ such that: $i \neq j$, $\mathbf{x}[i] < X_i^{\max}$, $\mathbf{x}[j] > X_j^{\min}$ and $\mathbf{x}[i] < \mathbf{x}[j]$.

Proof. We denote by $opt(\mathbf{x})$ the left member of the bi-conditional symbol and by $\neg p(\mathbf{x})$ the right member.

($\Rightarrow$) $p(\mathbf{x}) \Rightarrow \neg opt(\mathbf{x})$: Assume it is possible to find a pair i, j such that $\mathbf{x}[i] < X_i^{\max}$, $\mathbf{x}[j] > X_j^{\min}$ and $\mathbf{x}[i] < \mathbf{x}[j]$, then it is possible to transform the assignment $\mathbf{x}$ into a assignment $\mathbf{x}'$ also of sum s with $n \cdot \sum_{i\in[1..n]}(\mathbf{x}'[i] - \frac{s}{n})^2 < n \cdot \sum_{i\in[1..n]}(\mathbf{x}[i] - \frac{s}{n})^2$. We define the positive value $\delta = \min\{((\mathbf{x}[j] - \mathbf{x}[i])/2), (X_i^{\max} - \mathbf{x}[i]), (\mathbf{x}[i] - X_i^{\min})\}$ and $\mathbf{x}'[k] = \mathbf{x}[k]\ \forall k \neq i, j$, $\mathbf{x}'[j] = \mathbf{x}[j] - \delta$, $\mathbf{x}'[i] = \mathbf{x}[i] + \delta$. If we define $\Delta(\mathbf{x}) = n \cdot \sum_{i\in[1..n]}(\mathbf{x}[i] - \frac{s}{n})^2 = n \cdot \sum_{i\in[1..n]} \mathbf{x}[i]^2 - s^2$. Then $\Delta(\mathbf{x}) - \Delta(\mathbf{x}') = -n \cdot (2\delta^2 + 2\delta \cdot (\mathbf{x}[i] - \mathbf{x}[j])) \geq 2n\delta^2 > 0$ because $(\mathbf{x}[j] - \mathbf{x}[i]) \geq 2\delta$.

($\Leftarrow$) $\neg opt(\mathbf{x}) \Rightarrow p(\mathbf{x})$: We assume that $\mathbf{x}$ is not optimal and we consider $\mathbf{x}'$ a modified version of $\mathbf{x}$ also of sum s. Without loss of generality, we consider that the k first entries were increased,the l next were decreased and the other unchanged: $(\mathbf{x}'[i] - \mathbf{x}[i]) = \delta_i > 0$ for $i \in [1..k]$, $(\mathbf{x}[i] - \mathbf{x}'[i]) = \delta_i > 0$ for $i \in [k+1..k+l]$ and $\mathbf{x}[i] = \mathbf{x}'[i]$ for $i \in [(k+l+1)..n]$. The sum s is preserved from $\mathbf{x}$ to $\mathbf{x}'$ if $\sum_{i=1}^{k} \delta_i - \sum_{i=k+1}^{k+l} \delta_i = 0$. The difference between the sum of square deviations can be written as:

$$\Delta(\mathbf{x}) - \Delta(\mathbf{x}') = n \cdot (\sum_{i=1}^{k+l} -\delta_i^2 - 2 \cdot \sum_{i=1}^{k} \mathbf{x}[i]\delta_i + 2 \cdot \sum_{i=k+1}^{k+l} \mathbf{x}[i]\delta_i).$$

We show that this difference is strictly negative (i.e. $\mathbf{x}'$ does not improve $\mathbf{x}$) if every increasing entry is larger than every decreasing entry, $\forall i \in [1..k], \forall j \in [k+1..k+l] : \mathbf{x}[i] > \mathbf{x}[j]$. A lower bound on $(\sum_{i=1}^{k} \mathbf{x}[i]\delta_i)$ is $((\min_{i\in[1..k]} \mathbf{x}[i]) \cdot \sum_{i=1}^{k} \delta_i)$ and an upper bound on $(\sum_{i=k+1}^{k+l} \mathbf{x}[i]\delta_i)$ is $((\min_{i\in[1..k]} \mathbf{x}[i]) \cdot \sum_{i=k+1}^{k+l} \delta_i)$. Hence we have: $\Delta(\mathbf{x}) - \Delta(\mathbf{x}') < n \cdot (-\sum_{i=1}^{k+l} \delta_i^2 + 2 \cdot (\min_{i\in[1..k]} \mathbf{x}[i]) \cdot (-\sum_{i=1}^{k} \delta_i + \sum_{i=k+1}^{k+l} \delta_i)) = -n \cdot \sum_{i=1}^{k+l} \delta_i^2 < 0$.

□

An assignment $\mathbf{x}$ such that $\nexists\, i, j$ with $\mathbf{x}[i] < X_i^{\max}$, $\mathbf{x}[j] > X_j^{\min}$ and $\mathbf{x}[i] < \mathbf{x}[j]$ is called a ν-$\mathbb{Q}$centered assignment in [31]:

Definition 5 (ν-$\mathbb{Q}$centered assignment [31])**.** A ν-$\mathbb{Q}$centered assignment $\mathbf{x} \in \mathbb{Q}^n$ on $\mathbf{X}$ with $\nu \in \mathbb{Q}$ is such that $\forall i \in [1..n]$:

$$\mathbf{x}[i] = \begin{cases} X_i^{\max} & \text{if } X_i^{\max} < \nu \\ X_i^{\min} & \text{if } X_i^{\min} > \nu \\ \nu & \text{otherwise.} \end{cases}$$

Corollary 1 ([31])*.* $\mathbf{x} \in \operatorname{argmin}_{\mathbf{y}}\{n \cdot \sum_{i\in[1..n]} (\mathbf{y}[i] - s/n)^2$ s.t. $\sum_{i\in[1..n]} \mathbf{y}[i] = s$ and $\forall i : \mathbf{y}[i] \in I_D^{\mathbb{Q}}(X_i)\}$ if and only if $\mathbf{x}$ is a ν-$\mathbb{Q}$centered assignment of sum s.

The optimal value $\underline{\Delta}^{\mathbb{Q}}$ can be obtained from a tuple which is ν-$\mathbb{Q}$centered and exhibiting a sum of s. Note that by definition of a ν-$\mathbb{Q}$centered assignment, this tuple is completely defined by the value ν. The algorithm from [31] searches for the value ν such that the ν-$\mathbb{Q}$centered assignment exhibits a sum of s.

For a given value ν, the sum of the corresponding ν-$\mathbb{Q}$centered assignment is

$$\sum_{X_i^{\min} > \nu} X_i^{\min} + \sum_{X_i^{\max} < \nu} X_i^{\max} + \sum_{\nu \in I_D^{\mathbb{Q}}(X_i)} \nu. \tag{3.5}$$

In order to simplify the notations, we denote by:

- $R(\nu)$ the (Right) variables $\{X_i$ s.t. $X_i^{\min} > \nu\}$,
- $L(\nu)$ the (Left) variables $\{X_i$ s.t. $X_i^{\max} < \nu\}$ and by
- $M(\nu)$ the (Medium) variables $\{X_i$ s.t. $X_i \notin (R(\nu) \cup L(\nu))\}$.

The cardinality of the sets $R(\nu)$, $L(\nu)$ and $M(\nu)$ are denoted respectively $r(\nu)$, $l(\nu)$ and $m(\nu)$. Let us furthermore define

- $ES(\nu) = \sum_{X_i \in R(\nu)} X_i^{\min} + \sum_{X_i \in L(\nu)} X_i^{\max}$ (Extrema Sum), and
- $ES_{(2)}(\nu) = \sum_{X_i \in R(\nu)} (X_i^{\min})^2 + \sum_{X_i \in L(\nu)} (X_i^{\max})^2$.

When no ambiguity is possible, the ν argument will be dropped.

With the introduced notations, the sum of the ν-$\mathbb{Q}$centered assignment (3.5) can be simply written as $ES + m \cdot \nu$. Our objective to find the lower bound $\underline{\Delta}^{\mathbb{Q}}$ is achieved if we can find a value ν such that $ES + m \cdot \nu = s$.

Non empty intervals such that every domain either completely subsume it or do not overlap it, have the property that for every value ν inside it, the quantities $R(\nu)$, $L(\nu)$, $M(\nu)$ and $ES(\nu)$ are constant. This is more formally stated in Property 1.

Property 1. All intervals $I = [a, b] \subset \mathbb{Q}$ with $a < b$ such that $\forall i \in [1..n]$: $(I \subseteq I_D^{\mathbb{Q}}(X_i)) \vee (I \geq I_D^{\mathbb{Q}}(X_i)) \vee (I \leq I_D^{\mathbb{Q}}(X_i))$ have the property that for every $a < (\nu^1, \nu^2) < b$ the following equalities hold: $R(\nu^1) = R(\nu^2)$, $L(\nu^1) = L(\nu^2)$, $M(\nu^1) = M(\nu^2)$ and $ES(\nu^1) = ES(\nu^2)$.

Proof. Direct consequences of the definitions of R, L, M and ES. $\square$

Property 1 leads naturally to an extension of the definitions of R, L, M and ES to intervals I such as the ones considered in the Property 1:

- $R(I)$ the variables $\{X_i$ s.t. $X_i^{\min} \geq \max(I)\}$,
- $L(I)$ the variables $\{X_i$ s.t. $X_i^{\max} \leq \min(I)\}$,
- $M(I)$ the variables $\{X_i$ s.t. $X_i^{\min} \leq \min(I)$ and $X_i^{\max} \geq \max(I)\}$ and
- $ES(I) = \sum_{X_i \in L(I)} X_i^{\max} + \sum_{X_i \in R(I)} X_i^{\min}$.

Note that $M(I) = \mathbf{X} - (R(I) \cup L(I))\}$.

When ν varies inside such an interval I, the sum ranges in the interval $SI(I) = [ES(I) + m(I) \cdot \min(I), ES(I) + m(I) \cdot \max(I)]$ (SI stands for Sum Interval). If s does not fall in $SI(I)$ then the value ν we are looking for does not lie inside I neither. On the contrary if s belongs to this interval, ν is the solution of the equation $s = ES + m \cdot \nu$ that is $\nu = (s - ES)/m$.

There are at most $2 \cdot n - 1$ intervals I to consider (when all the bounds are different). These are obtained by sorting the set of upper and lower bounds of every variable X_i into increasing order. Any two consecutive values of this sorted sequence form an interval I satisfying Property 1.

Definition 6. Let $B(\mathbf{X})$ be the sorted sequence in non-decreasing order of the set of bounds $\bigcup_i \{X_i^{\min}, X_i^{\max}\}$. Let $\mathcal{I}(\mathbf{X})$ be the set of intervals defined by a pair of two consecutive elements of $B(\mathbf{X})$. The k^{th} interval of $\mathcal{I}(\mathbf{X})$ is denoted by I_k. For an interval $I = I_k$ we define the operator $\text{prev}(I) = I_{k-1}, (k > 1)$ and $\text{succ}(I) = I_{k+1}$.

Example 4 (Building $I(\mathbf{X})$). Let $\mathbf{X} = \{X_1, X_2, X_3\}$ with $Dom(X_1) = [1..3]$, $Dom(X_2) = [2..6]$ and $Dom(X_3) = [3..9]$ then $\mathcal{I}(X) = \{I_1, I_2, I_3, I_4\}$ with $I_1 = [1, 2]$, $I_2 = [2, 3]$, $I_3 = [3, 6]$, $I_4 = [6, 9]$. We have $\text{prev}(I_3) = I_2$ and $\text{succ}(I_3) = I_4$.

Observe that for two consecutive intervals I_k and I_{k+1} taken from $\mathcal{I}(\mathbf{X})$, the sum intervals are also contiguous: $\max(SI(I_k)) = \min(SI(I_{k+1}))$. It is then possible to make an algorithm to filter the lower bound of Δ. Algorithm 8 computes $\underline{\Delta}^{\mathbb{Q}}$ by iterating over the contiguous intervals $SI(I_k)$ until the sum lies inside it.

Algorithm 1: Filtering of Δ

```
   Data: $\mathcal{I}(\mathbf{X})$ and for all $I \in \mathcal{I}(\mathbf{X})$: $m(I)$ and $ES(I)$.
   Result: $\Delta^{\min} \leftarrow \max\{\underline{\Delta}^{\mathbb{Q}}, \Delta^{\min}\}$.
1  forall $I \in \mathcal{I}(X)$ do
2      if $s \in SI(I)$ then
3          $\nu \leftarrow (s - ES(I))/m(I)$
4          $\underline{\Delta}^{\mathbb{Q}} \leftarrow n \cdot \left(ES_{(2)}(I) + m(I) \cdot \nu^2\right) - s^2$
5          $\Delta^{\min} \leftarrow \max\{\underline{\Delta}^{\mathbb{Q}}, \Delta^{\min}\}$
6          break
```

Algorithm 8 executes in $\mathcal{O}(n)$ once $\mathcal{I}(\mathbf{X})$ is computed and that $ES(I)$ and $m(I)$ are available for every $I \in \mathcal{I}(\mathbf{X})$. Unfortunately, computing $\mathcal{I}(\mathbf{X})$ requires to sort the bounds, hence $\mathcal{O}(n \log(n))$ time. Algorithm 8 also needs the values $ES(I)$ and $m(I)$. For a given I, these can be obtained in $\Theta(n)$ by scanning every variable once. This would rise the overall complexity of Algorithm 8 to $\mathcal{O}(n^2)$ since in the worst case, $ES(I)$ and $m(I)$ must be computed for every $I \in \mathcal{I}(\mathbf{X})$. A smarter procedure is possible to compute $ES(I)$ and $m(I)$ in linear time for every $I \in \mathcal{I}(\mathbf{X})$.

Lemma 1 explains how the values $ES(I)$ can be computed in linear time for all $I \in \mathcal{I}(\mathbf{X})$ once $l(I), r(I)$ are given for all $I \in \mathcal{I}(\mathbf{X})$. Finally Algorithm 10 computes $l(I)$, $r(I)$ and $m(I)$ for all $I \in \mathcal{I}(\mathbf{X})$ in linear time too. This last algorithm can be easily understood with the invariant given in the pseudo code.

Lemma 1 ([31]). $ES(I_{k+1}) = ES(I_k) + (p_{k+1} - q_{k+1}) \cdot \max(I_k)$ where $p_{k+1} = l(I_{k+1}) - l(I_k)$ and $q_{k+1} = r(I_k) - r(I_{k+1})$.

Algorithm 2: Compute $\mathcal{I}(\mathbf{X})$ and $l(I), r(I), m(I)$ for all $I \in \mathcal{I}(\mathbf{X})$

```
   Data: The sorted sequence of the set of bounds
         $B(\mathbf{X}) = \langle b_1, b_2, ..., b_k \rangle$ and for each bound $b_i$ the
         information $b_i^+ = |\{X_i | X_i^{\max} = b_i\}|$ and
         $b_i^- = |\{X_i | X_i^{\min} = b_i\}|$.
   Result: $\mathcal{I}(\mathbf{X})$ and for all $I \in \mathcal{I}(\mathbf{X})$: $m(I)$, $l(I)$ and $r(I)$.
1  $lc \leftarrow b_1^+$
2  $rc \leftarrow n - b_1^-$
3  $\mathcal{I} \leftarrow list()$
4  for $i \leftarrow 2$ to $k$ do
       /* invariant:
       $lc = |\{X_j | X_j^{\max} \leq b_{i-1}\}|$ and $rc = |\{X_j | X_j^{\min} \geq b_i\}|$   */
5      $I \leftarrow [b_{i-1}, b_i]$
6      $\mathcal{I}.add(I)$
7      $l(I) = lc$
8      $r(I) = rc$
9      $m(I) = n - l(I) - r(I)$
10     $lc \leftarrow lc + b_i^+$
11     $rc \leftarrow rc - b_i^-$
```

The complete filtering algorithm dominated by a complexity of $\mathcal{O}(n \log(n))$ is:

- sort the bounds ($\mathcal{O}(n \log(n))$),
- compute $\mathcal{I}(\mathbf{X})$ and for all $I \in \mathcal{I}(\mathbf{X})$: $r(I)$, $l(I)$ and $m(I)$ with Algorithm 10 ($\Theta(n)$),
- compute $ES(I)$ for all $I \in \mathcal{I}(\mathbf{X})$ with Lemma 1 ($\mathcal{O}(n)$),
- filter Δ with Algorithm 8 ($\mathcal{O}(n)$).

i	I_i	$R(I_i)$	$L(I_i)$	$M(I_i)$	$ES(I_i)$	$ES_{(2)}(I_i)$	$IS(I_i)$
1	$[1,2]$	x_2, x_3	ϕ	x_1	5	13	$[6,7]$
2	$[2,3]$	x_3	ϕ	x_1, x_2	3	9	$[7,9]$
3	$[3,6]$	ϕ	x_1	x_2, x_3	3	9	$[9,15]$
4	$[6,9]$	ϕ	x_1, x_2	x_3	9	45	$[15,18]$

Table 3.2: Relevant values computed from Example 5.

Example 5 (Computing $\underline{\Delta}^{\mathbb{Q}}$). Variables and domains are from Example 4 and the sum considered is $s = 10$. Relevant values necessary to compute $\underline{\Delta}^{\mathbb{Q}}$ are given in Table 3.2. Since $s \in IS(I_3)$: $\nu = (s - ES)/m = (10-3)/2 = 3.5$ and $\underline{\Delta}^{\mathbb{Q}} = 3 \cdot (ES_{(2)}(I_3) + m(I_3) \cdot 3.5^2) - 10^2 = 0.5$. For $s = 9$, $s \in IS(I_2)$ and $s \in IS(I_3)$. Whichever interval is chosen between I_2 and I_3, the value ν is the same. Consequently the value for $\underline{\Delta}^{\mathbb{Q}}$ is also the same.

The remaining of the section explains how to compute $\underline{\Delta}^{\mathbb{Z}} \geq \lceil \underline{\Delta}^{\mathbb{Q}} \rceil$ that will allow to achieve $\mathbb{Z}$-bound-consistency rather than $\mathbb{Q}$-bound-consistency on `spread`.

As shown in the next theorem, the optimal solution obtained with integer assignments is very similar to the one obtained on rational domains in Theorem 2.

Theorem 3. $\mathbf{x} \in \operatorname{argmin}_{\mathbf{y}}\{n \cdot \sum_{i\in[1..n]}(\mathbf{y}[i] - \frac{s}{n})^2$ s.t. $\sum_{i\in[1..n]} \mathbf{y}[i] = s$ and $\forall i : \mathbf{y}[i] \in I_D^{\mathbb{Z}}(X_i)\}$ if and only if $\sum_{i\in[1..n]} \mathbf{x}[i] = s$ and $\nexists\, i, j$ such that $\mathbf{x}[i] < X_i^{\max}$, $\mathbf{x}[j] > X_j^{\min}$ and $\mathbf{x}[i] + 1 < \mathbf{x}[j]$.

Proof. Similar to proof of Theorem 2:

($\Rightarrow$) take $\delta = 1$.

($\Leftarrow$) All the δ_i's are integer and greater or equal to 1. We show that the difference $(\Delta(\mathbf{x}) - \Delta(\mathbf{x}'))$ is non positive if every increasing entry is at most one smaller than every decreasing entry, $\forall i \in [1..k], \forall j \in [k+1..k+l] : \mathbf{x}[i] + 1 \geq \mathbf{x}[j]$. A lower bound on $(\sum_{i=1}^{k} \mathbf{x}[i]\delta_i)$ is $((\min_{i\in[1..k]} \mathbf{x}[i]) \cdot \sum_{i=1}^{k} \delta_i)$ and an upper bound on $(\sum_{i=k+1}^{k+l} \mathbf{x}[i]\delta_i)$ is $((\min_{i\in[1..k]} \mathbf{x}[i] + 1) \cdot \sum_{i=k+1}^{k+l} \delta_i)$. Hence we have: $\Delta(\mathbf{x}) - \Delta(\mathbf{x}') < n \cdot (-\sum_{i=1}^{k+l} \delta_i^2 + (\min_{i\in[1..k]} \mathbf{x}[i]) \cdot (-2 \cdot \sum_{i=1}^{k} \delta_i + 2 \cdot \sum_{i=k+1}^{k+l} \delta_i) + 2 \cdot \sum_{i=k+1}^{k+l} \delta_i) = n \cdot (-\sum_{i=1}^{k+l} \delta_i^2 + 2 \cdot \sum_{i=k+1}^{k+l} \delta_i) \leq 0.$

□

Definition 7. A $\mathbb{Z}$centered assignment $\mathbf{x}$ is such that $\forall i : \mathbf{x}[i] \in I_D^{\mathbb{Z}}(X_i)$ and $\nexists\, i, j$ such that $\mathbf{x}[i] < X_i^{\max}$, $\mathbf{x}[j] > X_j^{\min}$ and $\mathbf{x}[i] + 1 < \mathbf{x}[j]$.

Algorithm 3 computes a $\mathbb{Z}$centered assignment of sum s. This

Algorithm 3: An inefficient algorithm to compute a $\mathbb{Z}$centered assignment of sum s.

```
   Result: A $\mathbb{Z}$centered assignment $\mathbf{x}$ of sum $s$.
1  $\mathbf{x} \leftarrow$ a valid assignment of sum $s$
2  while $\exists\, i, j$ such that $\mathbf{x}[i] < X_i^{\max}$, $\mathbf{x}[j] > X_j^{\min}$ and
   $\mathbf{x}[i] + 1 < \mathbf{x}[j]$ do
3      $\mathbf{x}[i] \leftarrow \mathbf{x}[i] + 1$
4      $\mathbf{x}[j] \leftarrow \mathbf{x}[j] - 1$
```

algorithm can be easily implemented but is very inefficient. A smarter method is possible. The idea is that it is always possible to distribute the m entries assigned to ν on the integer values $\lfloor \nu \rfloor$ and $\lceil \nu \rceil$ while conserving a sum of s. Clearly, such an assignment is $\mathbb{Z}$centered since the values assigned to ν are distant by at most one after the transformation.

The distribution of the m entries on $\lfloor \nu \rfloor$ and $\lceil \nu \rceil$ while conserving the sum of s is given in the following. The value ν always takes the form of $(s - ES)/m$ and there are exactly m entries assigned to ν. Hence it is always possible to distribute these m entries on $\lfloor \nu \rfloor$ and $\lceil \nu \rceil$ without modifying the sum. The only question to answer is how many of them must be assigned to $\lfloor \nu \rfloor$ and to $\lceil \nu \rceil$? Let us denote by ν^+ the number of entries that must be assigned to $\lceil \nu \rceil$ and by ν^- the number of entries that must be assigned to $\lfloor \nu \rfloor$. Of course $\nu^+ + \nu^- = m$. Looking at the Figure 3.3, the sum is preserved if

$$\nu^- \cdot ((s - ES) \bmod m) = (m - \nu^-) \cdot (m - (s - ES) \bmod m).$$

Hence the repartition on $\lfloor \nu \rfloor$ and $\lceil \nu \rceil$ is:

$$\nu^- = m - (s - ES) \bmod m$$

$$\nu^+ = (s - ES) \bmod m.$$

Algorithm 8 can be modified into Algorithm 4 to achieve a bound-consistent filtering of Δ with the computation of $\underline{\Delta}^{\mathbb{Z}}$.

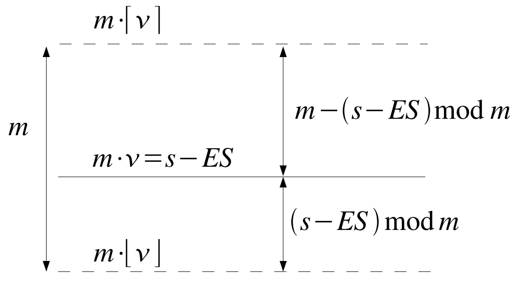


Figure 3.3: Distances between $\nu = (s - ES)/m$, $\lfloor \nu \rfloor$ and $\lceil \nu \rceil$.

Algorithm 4: Bound consistent filtering of Δ

```
Data: $\mathcal{I}(\mathbf{X})$ and for all $I \in \mathcal{I}(\mathbf{X})$: $m(I)$ and $ES(I)$.
Result: $\Delta^{\min} \leftarrow \max\{\underline{\Delta}^{\mathbb{Z}}, \Delta^{\min}\}$.
1 forall $I \in \mathcal{I}(\boldsymbol{X})$ do
2     if $s \in SI(I)$ then
3         $\nu \leftarrow (s - ES(I))/m(I)$
4         $\nu^+ \leftarrow (s - ES(I)) \bmod m(I)$
5         $\nu^- \leftarrow m - (s - ES(I)) \bmod m(I)$
6         $\underline{\Delta}^{\mathbb{Z}} \leftarrow n \cdot \left(ES_{(2)}(I) + \nu^+ \cdot \lceil \nu \rceil^2 + \nu^- \cdot \lfloor \nu \rfloor^2\right) - s^2$
7         $\Delta^{\min} \leftarrow \max\{\underline{\Delta}^{\mathbb{Z}}, \Delta^{\min}\}$
8         break
```

Example 6. Variables are the same as in Example 5 and the sum is $s = 10$. From Example 5, $s \in IS(I_3)$ and $\nu = (s - ES)/m = (10-3)/2 = 3.5$. The repartition of overlapping variables between $\lfloor \nu \rfloor$ and $\lceil \nu \rceil$ is given by $\nu^+ = (s - ES(I)) \bmod m(I) = (10 - 3) \bmod 3 = 1$ and $\nu^- = 1$. Consequently $\underline{\Delta}^{\mathbb{Z}} = 3 \cdot (ES_{(2)}(I_3) + 4^2 + 3^2) - 10^2 = 2$.

3.4.2 Filtering of X

Only the propagation of the upper bound of X_i is considered here since the propagation of its lower bound is a symmetrical problem with respect to the mean s/n. The propagation of X_i is achieved in [42] by computing the largest consistent value assuming the domains of other variables are rational intervals:

$$\overline{X}_i^{\mathbb{Q}} = \max_{\mathbf{x}}\{\mathbf{x}[i] \text{ s.t. } n \cdot \sum_{i\in[1..n]} (x[i]-s/n)^2 \leq \Delta^{\max} \text{ and} \\ \sum_{i\in[1..n]} \mathbf{x}[i] = s \text{ and } \forall j : x[j] \in I_D^{\mathbb{Q}}(X_j)\}. \tag{3.6}$$

The $\mathbb{Z}$-bound-consistent upper bound for X_i is obtained by allowing integer assignments only that is by replacing $\mathbb{Q}$ by $\mathbb{Z}$ in expression (3.6).

Algorithm to compute $\overline{X}_i^{\mathbb{Q}}$

The idea of the algorithm from [31] it to start from the a ν-$\mathbb{Q}$centered assignment $\mathbf{x}$ of sum s found with Algorithm 8. This assignment is the one with minimal sum of square deviations $\underline{\Delta}^{\mathbb{Q}}$. If $\mathbf{x}[i] = X_i^{\max}$, the upper bound of the variable X_i is consistent. Otherwise the optimal value $\overline{X}_i^{\mathbb{Q}}$ is obtained by successively assigning X_i to increasing values until either the minimal sum of square deviations reaches $\Delta^{\max}$ or it is proved that $X_i^{\max}$ is consistent. At that point, $\overline{X}_i^{\mathbb{Q}}$ is equal to the current value considered for X_i. This procedure is valid since the increasing values assigned to X_i range from $\mathbf{x}[i]$ to $X_i^{\max}$ and that the minimal sum of square deviations $\underline{\Delta}^{\mathbb{Q}}$ increases quadratically when the value assigned to X_i increases.

A detailed description of the Algorithm 6 follows. Let us denote by x_i the current value assigned to X_i and by $d = x_i - \mathbf{x}[i]$ the distance of the current value of X_i to the ith entry in the starting ν-$\mathbb{Q}$centered assignment $\mathbf{x}$ of sum s. Let us furthermore denote by $\underline{\Delta}^{\mathbb{Q}'}, SI', ES', m', \nu'$ the modified values if X_i were assigned to $x_i = \mathbf{x}[i] + d$.

The interval I from $\mathcal{I}(\mathbf{X})$ is such that $s \in SI(I)$. It is first assumed that $X_i \in R(I)$ such that $\mathbf{x}[i] = X_i^{\min}$ (by definition of a ν-$\mathbb{Q}$centered assignment). Recall that if $X_i \in M(I)$ then $\mathbf{x}[i] = \nu = (s - ES(I))/m$ and if $X_i \in L(I)$ then $\mathbf{x}[i] = X_i^{\max}$ and no filtering of the upper bound is possible.

The following lemma gives the expression of the quadratic evolution of $\underline{\Delta}^{\mathbb{Q}}$ with d *i.e.* when x_i increases. An illustration of Lemma 2 is given on Figure 3.4.

Lemma 2. if $d \leq s - \min(IS(I))$ then

1. $ES'(I) = ES(I) + d$,

2. $ES'_{(2)}(I) = ES_{(2)} + d^2 + 2d \cdot X_i^{\min}$,
3. $IS'(I) = IS(I) + d$,
4. $\nu' = \nu - d/m$ and
5. $\underline{\Delta}^{\mathbb{Q}'} = n \cdot (ES'_{(2)}(I) + m \cdot \nu'^2) - s^2$
   $\underline{\Delta}^{\mathbb{Q}'} = n \cdot \left(ES_{(2)}(I) + d^2 + 2dX_i^{\min} + m \cdot (\nu - \frac{d}{m})^2\right) - s^2$
   $\underline{\Delta}^{\mathbb{Q}'} = \underline{\Delta}^{\mathbb{Q}} + n \cdot (d^2 + 2dX_i^{\min} + d^2/m - 2d\nu)$.

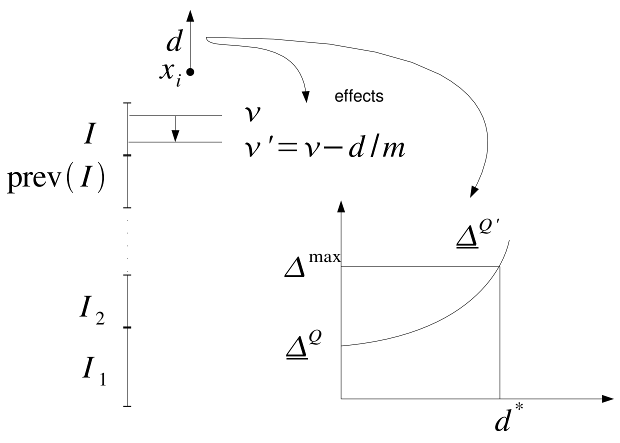


Figure 3.4: Consequences of the increasing of x_i by d : $\underline{\Delta}^{\mathbb{Q}'}$ increases quadratically and ν' decreases by d/m.

From Lemma 2, the minimal sum of square deviations increases quadratically with d. The maximum consistent value for X_i is the non negative solution of the following second degree equation:

$$\underline{\Delta}^{\mathbb{Q}} + n \cdot (d^2 + 2dX_i^{\min} + d^2/m - 2d\nu) = \Delta^{\max}.$$

The non negative solution of this equation is $d^* = \frac{-b+\sqrt{b^2-ac}}{a}$ where $a = n \cdot (1 + 1/m)$, $b = n \cdot (x_i - \nu)$ and $c = \underline{\Delta}^{\mathbb{Q}} - \Delta^{\max}$. This reasoning is valid as long as $d \le s - \min(IS(I))$ otherwise, ν' moves outside the interval I and the modified values in Lemma 2 are no longer valid. If $d^* \le s - \min(IS(I))$ then $\overline{X_i^{\mathbb{Q}}} = \mathbf{x}[i] + d^*$. Otherwise, $\mathbf{x}[i] + s - \min(IS(I))$ is a consistent value for X_i and larger values for x_i must be considered.

The procedure is repeated with the current interval $\text{prev}(I)$ instead of I. Indeed when $X_i \leftarrow x_i = \mathbf{x}[i] + s - \min(IS(I))$, the value ν' is equal to $\min(I) = \max(\text{prev}(I))$ and $s = \min(IS'(I)) = \max(IS'(\text{prev}(I)))$.

The process is repeated until one valid solution of the second degree equation $d^* \leq s - \min(IS(I))$ is found or the current interval considered is equal to I_1.

Until now it was assumed that $X_i \in R(I)$. The case $X_i \in M(I)$ can also lead to the filtering of $X_i^{\max}$. This case can be reduced to the same procedure as for the case $X_i \in R(I)$:

- The x_i starts to increase from ν rather than from $X_i^{\min}$ as previously. Hence, the interval I is conceptually split in two in the value ν: $I' = [\min(I), \nu]$ and $[\nu, \max(I)]$.
- The value $\overline{X_i^{\mathbb{Q}}}$ is computed on this modified configuration by considering that the domain of X_i is now $[\nu, X_i^{\max}]$.
- The same procedure as described previously can be applied since with this modified domain we have that $X_i \in R(I')$.

We say that the interval I is conceptually split because actually we only need to adapt the computation of $m'(I)$, $ES'(I)$ and $ES'_{(2)}$ from Lemma 2. For $x_i = X_i^{\min} + d$ and an interval $I \in \mathcal{I}(\mathbf{X})$ such that $x_i > \min(I)$, if $X_i \in M(I)$ then $m'(I') = m(I) - 1$ where $I' = [\min(I), \min\{\min(I), x_i\}]$. The reason is that $I_D^{\mathbb{Q}}(X_i)$ subsumes I' hence if $X_i \leftarrow x_i$, then X_i belongs to $R(I')$ and not to $M(I')$ anymore. The unified procedure to adapt the values $m'(I)$, $ES'(I)$ and $ES'_{(2)}$ for interval I when $X_i \leftarrow x_i$ with $x_i \geq \min(I)$ is given in Procedure 5.

The algorithm is given in Algorithm 6. In lines 1-5, the current value for x_i is initialized to $X_i^{\min}$ if $X_i \in R(I)$ and to v if $X_i \in L(I)$. In the main loop in lines 6-23, the algorithm tries to discover is there is a value for x_i such that the value of the sum of square deviations reaches the maximum value $\Delta^{\max}$ while keeping the value ν in the current interval I. If this is not possible (line 14) and that the upper bound $X_i^{\max}$ is not yet proved to be consistent (line 21), the procedure is repeated on the previous interval $\text{prev}(I)$ with x_i increased (line 20) such that the current value ν is equal to $\max(I)$ in the next iteration of the loop.

The complexity of the Algorithm 6 is linear in the number of intervals in $\mathcal{I}$ which is smaller than $2n - 1$. It is applied for each variable X_i to make the filtering $X_i^{\max} \leftarrow \min(X_i^{\max}, \lfloor \overline{X_i^{\mathbb{Q}}} \rfloor)$ and a similar procedure

Procedure `getUpdatedValues`$(x_i,\ I)$

```
Data: $\mathbf{x}_i \in I_D^{\mathbb{Q}}(X_i)$ and $x_i \geq \min(I)$, $I \in \mathcal{I}(\mathbf{X})$
Result: Let $I' = [\min(I), \min\{\max(I), x_i\}]$ and
        $\mathbf{X}' = \{X_1, ..., X_{i-1}, X_i \leftarrow x_i, X_{i+1}, ..., X_n\}$. Return the
        values $m(I'), ES(I')$ and $ES_{(2)}(I')$ computed with
        respect to $\mathbf{X}'$.
1 $d \leftarrow x_i - X_i^{\min}$
2 $m' \leftarrow m(I)$
3 $ES' \leftarrow ES(I) + d$
4 $ES'_{(2)} \leftarrow ES_{(2)}(I) + d^2 + 2dX_i^{\min}$
5 if $X_i \in M(I)$ then
6     $m' \leftarrow m' - 1$
7     $ES' \leftarrow ES' + X_i^{\min}$
8     $ES'_{(2)} \leftarrow ES'_{(2)} + (X_i^{\min})^2$
9 return $m', ES', ES'_{(2)}$
```

is used to filter the lower bounds $X_i^{\min}$. Hence the overall complexity of filtering $\mathbf{X}$ is $O(n^2)$. This complexity can actually be improved to $O(n \cdot log(n))$ by doing a dichotomic search on the intervals I rather than a linear search. The dichotomic search is valid since the sum of square deviation increases piecewise quadratically and continuously with the current value x_i.

Once again, the filtering with the value $\overline{X}_i^{\mathbb{Q}}$ is not $\mathbb{Z}$-bound-consistent because it corresponds to an assignment with some variables assigned to ν which might be not integer. Line 16 of the Algorithm 6 computes the $\mathbb{Z}$-bound-consistent upper bound for X_i given $\overline{X}_i^{\mathbb{Q}}$ and the current interval I. We explain in the following the procedure get$\overline{X}_i^{\mathbb{Z}}(x_i,\ I)$ detailed in Procedure 7 running in $O(m)$. The main steps of the procedure are illustrated on Figure 3.5. Line 1, rounds down the value $\overline{X}_i^{\mathbb{Q}}$. This corresponds to the arrow A on the Figure 3.5. Lines 2-5 compute the sum of square deviation corresponding to a $\mathbb{Z}$centered assignment of sum s with X_i assigned to $\overline{X}_i^{\mathbb{Z}}$. We can use the *getUpdatedValues* procedure to do this in constant time. Since the $\mathbb{Q}$centered assignment of sum s has been transformed into a $\mathbb{Z}$centered assignment of sum s, the sum of square deviations might have increased ($\underline{\Delta}^{\mathbb{Z}} > \underline{\Delta}^{\mathbb{Q}} = \Delta^{\max}$). In this case, the current value $\overline{X}_i^{\mathbb{Z}}$ at line 1 is not consistent for X_i and there is

Algorithm 6: Filtering of X_i

```
   Data: $I$ s.t. $s \in IS(I), X_i \in \mathbf{X}$
   Result: $X_i^{\max} \leftarrow \min\{\overline{X}_i^{\mathbb{Q}}, X_i^{\max}\}$
 1 if $X_i \in L(I)$ then return                  /* $X_i^{\max}$ is consistent */
 2 $\nu \leftarrow (s - ES(I))/m(I)$
 3 $x_i \leftarrow X_i^{\min}$
 4 if $X_i \in M(I)$ then
 5     $x_i \leftarrow \nu$
 6 repeat
 7     $m', ES', ES'_{(2)} \leftarrow$ getUpdatedValues($x_i$, $I$)
 8     $IS' \leftarrow ES' + m' \cdot \min(I)$
 9     if $m' > 0$ then
10         $\nu' \leftarrow (s - ES')/m$
11         $\underline{\Delta}^{\mathbb{Q}} \leftarrow n \cdot (ES'_{(2)} + m' \cdot \nu'^2) - s^2$
12         $d_1 \leftarrow s - IS'$
13         $d_2 \leftarrow (-b + \sqrt{b^2 - a \cdot c})/a$    /* $a, b, c$ def. in text */
14         if $d_2 \leq d_1$ then
15             $x_i \leftarrow x_i + d_2$                  /* $x_i = \overline{X}_i^{\mathbb{Q}}$ */
16             $\overline{X}_i^{\mathbb{Z}} \leftarrow$ get$\overline{X}_i^{\mathbb{Z}}$($x_i$, $I$)      /* b-c upper bound */
17             $X_i^{\max} \leftarrow \min\{X_i^{\max}, \overline{X}_i^{\mathbb{Z}}\}$
18             return
19         else
20             $x_i \leftarrow x_i + d_1$
21     if $x_i \geq X_i^{\max}$ then return        /* $X_i^{\max}$ is consistent */
22     $I \leftarrow$ prev($I$)
23 until $I = I_1$
24 $X_i^{\max} \leftarrow \min\{X_i^{\max}, x_i\}$
```

some opportunity to decrease it even more by steps of 1 until a consistent value for $X_i^{\max}$ is found (lines 6-8).

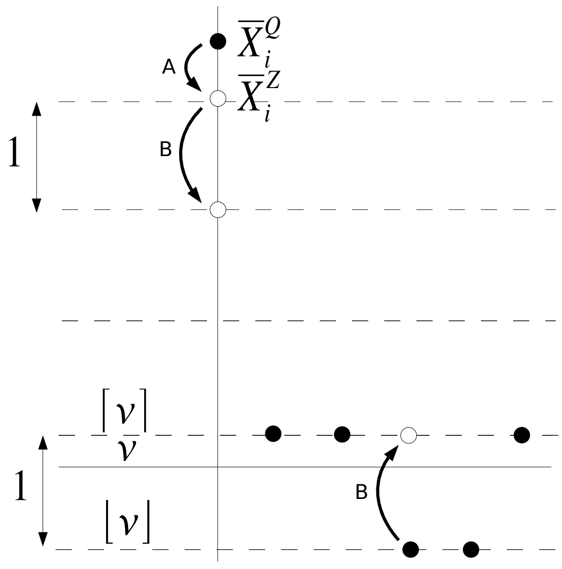


Figure 3.5: Illustration of the main steps of the Procedure 7

Note that no more than m iterations are needed because as shown on the Figure 3.5 with arrows B, each time the $\overline{X}_i^{\mathbb{Z}}$ is decreased by 1, one variable assigned to $\lfloor \nu \rfloor$ must be increased in the $\mathbb{Z}$centered assignment (to maintain the correct sum s). We are guaranteed than within at most m steps the $\mathbb{Z}$centered assignment will corresponds to a $\mathbb{Q}$centered assignment which was proved to exhibit a sum of square deviations $\leq \Delta^{\max}$ in Algorithm 6. It remains to explain how to update in constant time the value $\underline{\Delta}^{\mathbb{Z}}$ in line 7 each time $\overline{X}_i^{\mathbb{Z}}$ is decreased by one. We refer to arrows B of Figure 3.5 for the visual explanation and to the next transformations to obtain the formula.

$$\begin{aligned}
\underline{\Delta}^{\mathbb{Z}'} &\leftarrow n \cdot \Big(ES'_{(2)} + 1 - 2\overline{X}_i^{\mathbb{Z}} + (\nu^+ + 1) \cdot \lceil \nu \rceil^2 + (\nu^- - 1) \cdot \lfloor \nu \rfloor^2) \Big) - s^2 \\
&\leftarrow \underline{\Delta}^{\mathbb{Z}} + n \cdot (1 - 2\overline{X}_i^{\mathbb{Z}} + \lceil \nu \rceil^2 - \lfloor \nu \rfloor^2) \\
&\leftarrow \underline{\Delta}^{\mathbb{Z}} + n \cdot (1 - 2\overline{X}_i^{\mathbb{Z}} + (\lceil \nu \rceil - \lfloor \nu \rfloor) \cdot (\lceil \nu \rceil + \lfloor \nu \rfloor)) \\
&\leftarrow \underline{\Delta}^{\mathbb{Z}} + n \cdot (1 - 2\overline{X}_i^{\mathbb{Z}} + 1 \cdot (2\lceil \nu \rceil - 1)) \\
&\leftarrow \underline{\Delta}^{\mathbb{Z}} + 2n \cdot (\lceil \nu \rceil - \overline{X}_i^{\mathbb{Z}})
\end{aligned}$$

Procedure get$\overline{X}_i^{\mathbb{Z}}$($x_i$, I)

```
1 $\overline{X}_i^{\mathbb{Z}} \leftarrow \lfloor x_i \rfloor$
2 $m', ES', ES'_{(2)} \leftarrow \text{getUpdatedValues}(\overline{X}_i^{\mathbb{Z}}, I)$
3 $\nu^+ \leftarrow (s - ES') \bmod m'$
4 $\nu^- \leftarrow m - \nu^+$
5 $\underline{\Delta}^{\mathbb{Z}} \leftarrow n \cdot \left(ES'_{(2)} + \nu^+ \cdot \lceil \nu \rceil^2 + \nu^- \cdot \lfloor \nu \rfloor^2\right) - s^2$
6 while $\underline{\Delta}^{\mathbb{Z}} > \Delta^{\max}$ do
7     $\underline{\Delta}^{\mathbb{Z}} \leftarrow \underline{\Delta}^{\mathbb{Z}} + 2n \cdot (\lceil \nu \rceil - \overline{X}_i^{\mathbb{Z}})$
8     $\overline{X}_i^{\mathbb{Z}} \leftarrow \overline{X}_i^{\mathbb{Z}} - 1$
9 return $\overline{X}_i^{\mathbb{Z}}$
```

The term $1 - 2\overline{X}_i^{\mathbb{Z}}$ is the resulting modification on $ES'_{(2)}$ caused by the decreasing by one of $\overline{X}_i^{\mathbb{Z}}$, and the $+1$ and -1 applied respectively on ν^+ and ν^- is because one variable assigned to $\lfloor \nu \rfloor$ is increased by one. The other lines are simple algebraic manipulations to simplify the formula.

3.4.3 Experimental Comparison of the $\mathbb{Q}$-bound-consistent and $\mathbb{Z}$-bound-consistent propagators

We propose to experiment the gain obtained with the global constraint developed for `spread` with respect to its decomposition into two arithmetic constraints. Since the design of the $\mathbb{Z}$-bound-consistent propagators complexify even more the filtering algorithms, it is also natural to experiment if it is worth using them. We propose to experiment this on the Balanced Academic Curriculum Problem (BACP). We modified the largest instance available from CSPLib to generate 100 instances from it. This largest instances is composed of 12 periods, 66 courses having a weight between 1 and 5 (credits) and 65 prerequisites relations. We generate a random instance from this one by assigning to each course a random weight between 1 and 5 and by randomly keeping 50 out of the 65 prerequisites. The COMET model is given in Listing 3.1. Lines 6-7 ask for the minimization of the variance variable. Lines 9-14 in the *subject to* block post the constraints of the problem. The global constraint *multiknapsack* links the placement variables x of the courses with the load

Listing 3.1: BACP Comet Model

```
 1 Solver<CP> cp();
 2 var<CP>{int} x[courses](cp,periods);
 3 var<CP>{int} l[periods](cp,0..totCredit);
 4 var<CP>{int} vari(cp,0..System.getMAXINT());
 5 cp.timeLimit(30);
 6 minimize<cp>
 7   vari
 8 subject to{
 9   cp.post(multiknapsack(x,credits,l));
10   cp.post(spread(l,totCredit,vari));
11   forall(i in prerequisites){
12     cp.post(x[prereq[i].a]<x[prereq[i].b]);
13   }
14 }
15 using{
16   while(!bound(x)){
17     selectMin(i in courses:!x[i].bound()) (x[i].getSize()){
18       tryall<cp>(p in periods: x[i].memberOf(p)) by(l[p].getMin())
19         cp.post(x[i]==p);
20       onFailure
21         cp.post(x[i]!=p);
22     }
23   }
24 }
```

variables l of the periods: $l[j] = \sum_i (x[i] = j) \cdot credits[i]$. The branching heuristic implemented in the *using* block line 15-24 selects the course which can be placed in the fewest number of period (line 17), then this course is placed in the least loaded periods first.

The results on the 100 BACP instances are summarized on table 3.3. The first line gives the number of instances solved and proved optimal. One can see that the decomposition cannot solve any instances and the $\mathbb{Z}$-bound-consistent propagators outperform significantly the $\mathbb{Q}$-bound consistent ones. The second line reports the number of instances for which the best objective value among the three approaches is reached. It appears that the gap between the $\mathbb{Q}$-BC and the $\mathbb{Z}$-BC decreases slightly

Table 3.3: Results on the 100 BACP instances.

	decomposition	`spread` $\mathbb{Q}$-BC	`spread` $\mathbb{Z}$-BC
#solved	0	12	74
#best	2	64	100

but remains significant. This means that the $\mathbb{Z}$-BC propagators are not only useful to prove optimality but also to reach a better objective.

3.5 Mean Absolute Deviation Constraint

The definition of the `deviation` constraint is the following:

Definition 8. Given a sequence of finite domain integer variables $\mathbf{X} = [X_1, X_2, ..., X_n]$, one sum value s and one deviation variable Δ, the constraint `deviation`$(\mathbf{X}, s, \Delta)$ holds if and only if

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

Note that:

- We work with the deviation of the scaled variables $|n \cdot X_i - s|$ rather than $|X_i - s/n|$ because the former is always an integer while the latter is not when the sum s is not a multiple of n ($S \mod n \neq 0$).
- The definition considers $\sum_{i \in [1..n]} |n \cdot X_i - S| \leq \Delta$ rather than $\sum_{i \in [1..n]} |n \cdot X_i - S| = \Delta$ because achieving bound-consistency for the latter is $\mathcal{NP}$-Hard.

As for `spread` two kinds of consistencies can be achieved for `deviation` The $\mathbb{Q}$-bound-consistency and the $\mathbb{Z}$-bound-consistency. As the filtering algorithms for these two consistencies are not related as was the case for `spread`, we start with the simplest filtering achieving the $\mathbb{Q}$-bound-consistency in Section 3.5.1 then we explain the $\mathbb{Z}$-bound-consistent filtering algorithm in Section 3.5.2

3.5.1 $\mathbb{Q}$-bound-consistency for `deviation`

Definition 9. For a scaled variable $n \cdot X_i$ and a given value s, the upper bounds on the right and left deviation are respectively

- $\overline{rd}(n \cdot X_i, s) = \max(0, n \cdot X_i^{\max} - s)$ and
- $\overline{ld}(n \cdot X_i, s) = \max(0, s - n \cdot X_i^{\min})$.

The sum of these values over $\mathbf{X} = [X_1, X_2, ..., X_n]$ are respectively

- $\overline{RD}(n \cdot \mathbf{X}, s) = \sum_{i \in [1..n]} \overline{rd}(n \cdot X_i, s)$ and
- $\overline{LD}(n \cdot \mathbf{X}, s) = \sum_{i \in [1..n]} \overline{ld}(n \cdot X_i, s)$.

The same idea holds for the lower bounds on the deviations:

- $\underline{rd}(n \cdot X_i, s) = \max(0, n \cdot X_i^{\min} - s)$.
- $\underline{ld}(n \cdot X_i, s) = \max(0, s - n \cdot X_i^{\max})$.
- $\underline{RD}(n \cdot \mathbf{X}, s) = \sum_{i \in [1..n]} \underline{rd}(n \cdot X_i, s)$.
- $\underline{LD}(n \cdot \mathbf{X}, s) = \sum_{i \in [1..n]} \underline{ld}(n \cdot X_i, s)$.

For a variable $X_i \in \mathbf{X}$ we define:

- $\overline{LD}_i(n \cdot \mathbf{X}, s) = \overline{LD}(n \cdot \mathbf{X}, s) - \overline{ld}(n \cdot X_i, s)$ and
- $\overline{RD}_i(n \cdot \mathbf{X}, s) = \overline{RD}(n \cdot \mathbf{X}, s) - \overline{rd}(n \cdot X_i, s)$.

To alleviate notations, $(n \cdot \mathbf{X}, s)$ are sometimes omitted. For example $\overline{LD}(n \cdot \mathbf{X}, s)$ is simply written $\overline{LD}$.

Example 7. Let $\mathbf{X} = [X_1, X_2, X_3, X_4]$ be four variables with domains $Dom(X_1) = [8, 10]$, $Dom(X_2) = [4, 7]$, $Dom(X_3) = [1, 5]$ and $Dom(X_4) = [3, 4]$. The following table exhibits the quantities introduced in Definition 9 for $s = 20$.

i	$\overline{rd}(4 \cdot X_i, 20)$	$\overline{ld}(4 \cdot X_i, 20)$	$\underline{rd}(4 \cdot X_i, 20)$	$\underline{ld}(4 \cdot X_i, 20)$
1	20	0	12	0
2	8	4	0	0
3	0	16	0	0
4	0	8	0	4
$\sum_i$	**28**	**28**	**12**	**4**
	$\overline{RD}_i(4 \cdot \mathbf{X}, 20)$	$\overline{LD}_i(4 \cdot \mathbf{X}, 20)$	$\underline{RD}_i(4 \cdot \mathbf{X}, 20)$	$\underline{LD}_i(4 \cdot \mathbf{X}, 20)$
1	8	28	0	4
2	20	24	12	4
3	28	12	12	4
4	28	20	12	0

The filtering for `deviation` is based on the next theorem stating that the sum of deviations above and under the mean are equal.

Lemma 3. Let $\mathbf{X} = [X_1, ..., X_n]$. The equality $s = \sum_{i \in [1..n]} X_i$ holds if and only if $\sum_{n \cdot X > s}(n \cdot X - s) = \sum_{n \cdot X < s}(s - X)$.

Proof. $s = \sum_{X \in \mathbf{X}} X$ can be rewritten $0 = \sum_{X \in \mathbf{X}} n \cdot X - s = \sum_{n \cdot X > s}(n \cdot X - s) + \sum_{n \cdot X < s}(n \cdot X - s) + \sum_{n \cdot X = s}(n \cdot X - s) = \sum_{n \cdot X > s}(n \cdot X - s) - \sum_{n \cdot X < s}(s - n \cdot X)$. □

Property 2. Let $\mathbf{X} = [X_1, ..., X_n]$. Whatever the value of s, an assignment on $\mathbf{X}$ satisfies:

- $\sum_{n \cdot X > s}(n \cdot X - s) \in [\underline{RD}(n \cdot \mathbf{X}, s), \overline{RD}(\mathbf{X}, s)]$ and
- $\sum_{n \cdot X < s}(s - n \cdot X) \in [\underline{LD}(n \cdot \mathbf{X}, s), \overline{LD}(\mathbf{X}, s)]$.

Theorem 4. $\texttt{deviation}(\mathbf{X}, s, \Delta)$ is consistent only if the following conditions are satisfied:

1. $\underline{RD}(n \cdot \mathbf{X}, s) \leq \frac{\Delta^{\max}}{2}$
2. $\underline{LD}(n \cdot \mathbf{X}, s) \leq \frac{\Delta^{\max}}{2}$
3. $\overline{RD}(n \cdot \mathbf{X}, s) \geq \frac{\Delta^{\min}}{2}$
4. $\overline{LD}(n \cdot \mathbf{X}, s) \geq \frac{\Delta^{\min}}{2}$
5. $\left[\underline{LD}(n \cdot \mathbf{X}, s), \overline{LD}(n \cdot \mathbf{X}, s)\right] \cap \left[\underline{RD}(n \cdot \mathbf{X}, s), \overline{RD}(n \cdot \mathbf{X}, s)\right] \neq \phi$

Proof. 1. If $\underline{RD}(n \cdot \mathbf{X}, s) > \frac{\Delta^{\max}}{2}$ then $\sum_{n \cdot X > s}(n \cdot X - s) > \frac{\Delta^{\max}}{2}$ (Property 2). Hence $\sum_{i=1}^{n} |n \cdot X_i - s| > \Delta^{\max}$ (by Lemma 3).

2., 3. and 4. similar to 1.

5. Direct consequence of Lemma 3 and Property 2.

$\square$

Filtering of $\Delta^{\min}$

The filtering of Δ to achieve the $\mathbb{Q}$-bound-consistency requires to solve the following optimization problem: the minimization of the sum of deviations from a given sum s allowing rational assignments.

$$\underline{\Delta}^{\mathbb{Q}} = \min_{\mathbf{x}}\{n \cdot \sum_{i\in[1..n]} |\mathbf{x}[i] - s/n| \quad \text{s.t.} \quad \sum_{i\in[1..n]} \mathbf{x}[i] = s \quad \text{and} \quad \forall i \in [1..n] : \mathbf{x}[i] \in I_D^{\mathbb{Q}}(X_i)\} \tag{3.7}$$

The filtering using this value is

$$\Delta^{\min} \leftarrow \max(\Delta^{\min}, \underline{\Delta}^{\mathbb{Q}}).$$

The remaining of this section mainly explains how $\underline{\Delta}^{\mathbb{Q}}$ can be computed in linear time with respect to the number of variables $n = |\mathbf{X}|$. Finding $\overline{\Delta}^{\mathbb{Q}}$ is an $\mathcal{NP}$-Hard problem as proved in [44].

Next Definition characterizes an optimal solution to the problem of finding $\underline{\Delta}^{\mathbb{Q}}$.

Definition 10 (up and down centered assignment)**.** Let $\mathbf{X} = [X_1, ..., X_n]$ and $\mathbf{x} = [x_1, ..., x_n] \in I_D^{\mathbb{Q}}(X_1) \times ... \times I_D^{\mathbb{Q}}(X_n)$ be an assignment on $\mathbf{X}$. Let $\text{sum}(\mathbf{x})$ denote the sum of assigned values: $\text{sum}(\mathbf{x}) = \sum_{i\in[1..n]} x[i]$.

An assignment $\mathbf{x}$ is said to be **up-centered** when:

$$n \cdot \mathbf{x}[i] \begin{cases} = n \cdot X_i^{\min} & \text{if } n \cdot X_i^{\min} \geq \text{sum}(\mathbf{x}) \\ \leq \text{sum}(\mathbf{x}) & \text{otherwise} \end{cases}$$

In other words, each variable with minimum domain value larger than the mean of the assigned values takes its minimum domain value and the other variables take values smaller than the mean of the assigned values.

An assignment $\mathbf{x}$ is said to be **down-centered** when:

$$n \cdot \mathbf{x}[i] \begin{cases} = n \cdot X^{\max} & \text{if } n \cdot X^{\max} \leq \text{sum}(\mathbf{x}) \\ \geq \text{sum}(\mathbf{x}) & \text{otherwise} \end{cases}$$

In other words, each variable with maximum domain value smaller than the mean of the assigned values takes its maximum domain value and the other variables take values larger than the mean of the assigned values.

Example 8. Considering the variables and domains of Example 7, the following assignment is up-centered for $s = 17$:

$$\mathbf{x} = [x_1, x_2, x_3, x_4] = [8, 4, 2, 3]$$

Theorem 5. An assignment is an optimal solution to the problem of finding $\underline{\Delta}^{\mathbb{Q}}$ if and only if it is a down-centered assignment or an up-centered assignment of sum s.

Proof. (if) Given an assignment $\mathbf{x}$ with $\text{sum}(\mathbf{x}) = s$, the only way to decrease the sum of deviations while conserving the sum s is to find a pair of variables X_i, X_j such that $n \cdot \mathbf{x}[i] > s, \mathbf{x}[i] > X_i^{\min}, n \cdot \mathbf{x}[j] < s, \mathbf{x}[j] < X_j^{\max}$ and to decrease $n \cdot \mathbf{x}[i]$ and increase $n \cdot \mathbf{x}[j]$ by the same quantity to make them closer to s. By definition of a left and down centered assignment, it is impossible to find such a pair X_i, X_j. Hence, up-centered and a down-centered assignments are optimal solutions.

(only if) Assume an assignment $\mathbf{x}$ is neither down-centered nor up-centered such that $\text{sum}(\mathbf{x}) = s$. It is possible to find at least two variables $X_i, X_j \in \mathbf{X}$. One with $n \cdot \mathbf{x}[i] > s$ and $\mathbf{x}[i] > X_i^{\min}$ (violation of up-centered) and one with $n \cdot \mathbf{x}[i] < s$ and $\mathbf{x}[j] < X_j^{\max}$ (violation of down-centered). Let us define $\delta = \min(n \cdot \mathbf{x}[i] - \max(n \cdot X_i^{\min}, s), \min(n \cdot X_j^{\max}, s) - n \cdot \mathbf{x}[j]))$. The assignment $\mathbf{x}$ is not optimal since the sum of deviations can be decreased by 2δ by modifying the assignment on X_i and X_j: $n \cdot \mathbf{x}'[i] = n \cdot \mathbf{x}[i] - \delta$ and $n \cdot \mathbf{x}'[j] = n \cdot \mathbf{x}[j] + \delta$. □

Theorem 6. If `deviation` is consistent then

$$\underline{\Delta}^{\mathbb{Q}} = 2 \cdot \max(\underline{LD}\,(n \cdot \mathbf{X}), s), \underline{RD}(n \cdot \mathbf{X}, s))\,.$$

Proof. Assume $\underline{LD} \geq \underline{RD}$, then it is possible to build a down-centered assignment $\mathbf{x}$ with $\text{sum}(\mathbf{x}) = s$ and which is optimal by Theorem 5. For this assignment $\sum_{n \cdot \mathbf{x}[i] < s}(s - n \cdot \mathbf{x}[i]) = \underline{LD}$ (by Definition 9 of $\underline{LD}$). Since $\sum_{n \cdot \mathbf{x}[i] > s}(n \cdot \mathbf{x}[i] - s) = \sum_{n \cdot \mathbf{x}[i] < s}(s - n \cdot \mathbf{x}[i])$ (by Lemma 3), the sum of deviations for this down-centered assignment is $\sum_{i \in [1..n]} |n \cdot \mathbf{x}[i] - s| = 2.\underline{LD}$. The case $\underline{LD} \leq \underline{RD}$ is similar. The assignment is up-centered instead of down-centered. □

Example 9. The variables and domains considered here are the same as those in Example 7. A mean $s = 20$ is considered. Using the computed values $\underline{LD}(n \cdot \mathbf{X}, 20) = 4$ and $\underline{RD}(n \cdot \mathbf{X}, 20) = 12$ from Example 7, it can be deduced that $\underline{\Delta}^{\mathbb{Q}} = 2.\max(4, 12) = 24$. Consequently, filtering on Δ for `deviation`$(\mathbf{X}, s = 20, \Delta \in [0, 28])$ leads to $Dom(D) = [24, 28]$.

Filtering on X

The filtering of $Dom(X_i)$ is based on the computation of the values $\overline{X}_i^{\mathbb{Q}}$ and $\underline{X}_i^{\mathbb{Q}}$:

$\overline{X}_i^{\mathbb{Q}}$ and $\underline{X}_i^{\mathbb{Q}}$ are the optimal values to the following problems:

$$\overline{X}_i^{\mathbb{Q}} = \max(X_i) \quad \text{and} \quad \underline{X}_i^{\mathbb{Q}} = \min(X_i) \tag{3.8}$$

$$\text{such that}: \sum_{j=1}^{n} n \cdot X_j = s \tag{3.9}$$

$$\sum_{j=1}^{n} |n \cdot X_j - s| \leq \Delta^{\max} \tag{3.10}$$

$$X_j \in I_D^{\mathbb{Q}}(X_j), 1 \leq j \leq n, j \neq i \tag{3.11}$$

The filtering rule on the domain of X_i can be simply written:

$$Dom(X_i) \longleftarrow Dom(X_i) \cap [\underline{X}_i^{\mathbb{Q}}, \overline{X}_i^{\mathbb{Q}}] \tag{3.12}$$

Theorem 7. For a variable X_i, assuming the constraint is consistent, the following equalities hold:

$$n \cdot \overline{X}_i^{\mathbb{Q}} = \min\left(\frac{\Delta^{\max}}{2}, \overline{LD}_i(n \cdot \mathbf{X}, s)\right) - \underline{RD}_i(n \cdot \mathbf{X}, s) + s.$$

$$n \cdot \underline{X}_i^{\mathbb{Q}} = -\min\left(\frac{\Delta^{\max}}{2}, \overline{RD}_i(n \cdot \mathbf{X}, s)\right) + \underline{LD}_i(n \cdot \mathbf{X}, s) + s.$$

Proof. Only $\overline{X}_i$ is considered because the proof for $\underline{X}_i^{\mathbb{Q}}$ is symmetrical with respect to s. Two cases can be considered:

- $\overline{LD}_i \leq \frac{\Delta^{\max}}{2}$: By Lemma 3 the deviation above the mean and under the mean must be equal. Hence the optimal solution is such that $\overline{X}_i^{\mathbb{Q}} - s + \underline{RD}_i = \overline{LD}_i$. Constraint (3.10) is not tight in this case.

- $\overline{LD}_i > \frac{\Delta^{\max}}{2}$: By Lemma 3 the constraint (3.9) means that the deviation above the mean and under the mean must be equal. The conjunction of constraint (3.9) with constraint (3.10) means that the deviation of the scaled variables above and under the sum s are equal and at most $\Delta^{\max}/2$. Hence the optimal solution is such that $n \cdot \overline{X}_i^{\mathbb{Q}} - s + \underline{RD}_i = \frac{\Delta^{\max}}{2}$. Constraint (3.10) is tight in this case.

If both cases are considered together, the equality $n \cdot \overline{X}_i^{\mathbb{Q}} - s + \underline{RD}_i =$- $\min(\frac{\Delta^{\max}}{2}, \overline{LD}_i)$ holds at the optimal solution. $\square$

The filtering procedure on $\mathbf{X}$ applies rule (3.12) once on each $X_i \in \mathbf{X}$. This can be achieved in linear time with respect to the number of variables.

Example 10. Variables and domains considered are the same as in Example 7. The constraint considered is `deviation`$(\mathbf{X} = [X_1, X_2, X_3, X_4], s = 20, \Delta \in [0, 28])$. Values $\overline{X}_i^{\mathbb{Q}}$ and $\underline{X}_i^{\mathbb{Q}}$ are:

i	$\overline{X}_i^{\mathbb{Q}}$	$\underline{X}_i^{\mathbb{Q}}$
1	$(\min(14, 28) - 0 + 20)/4 = 8.5$	$(-\min(14, 8) + 4 + 20)/4 = 4$
2	$(\min(14, 24) - 12 + 20)/4 = 5.5$	$(-\min(14, 20) + 4 + 20)/4 = 2.5$
3	$(\min(14, 12) - 12 + 20)/4 = 5$	$(-\min(14, 28) + 4 + 20)/4 = 2.5$
4	$(\min(14, 20) - 12 + 20)/4 = 5.5$	$(-\min(14, 28) + 0 + 20)/4 = 1.5$

Hence filtering rule (3.12) leads to filtered domains: $Dom(X_1) = [8, 8]$, $Dom(X_2) = [4, 5]$, $Dom(X_3) = [3, 5]$ and $Dom(X_4) = [3, 4]$.

3.5.2 $\mathbb{Z}$-bound-consistency for `deviation`

The previously presented filtering algorithm are simple and efficient. However, for integer finite domains, these algorithms are $\mathbb{Z}$-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.

When the domains of the X_i's are integer intervals $[X_i^{\min}..X_i^{\max}]$, the corresponding $\mathbb{Z}$-bound-consistent filtering rules are obtained by substituting $\mathbb{Q}$ by $\mathbb{Z}$ in Equations 3.7, 3.8 and 3.11. Nevertheless, the values $\underline{X}_i^{\mathbb{Q}}$ and $\overline{X}_i^{\mathbb{Q}}$ can be used for integer domains as well since they are obtained by relaxing the domains. 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.

Filtering of $\Delta^{\min}$

As illustrated in the following example, the relaxing assumption of rational interval domains can lead to miss some possible filtering with respect to a $\mathbb{Z}$-bound-consistent filtering on $\Delta^{\min}$.

Example 11 (Filtering of Δ). Assume two variables $\mathbf{X} = (X_1, X_2)$ with domains $[-5..5]$ and a sum constraint $s = 1$. Obviously $\underline{\Delta}^{\mathbb{Q}} = 0$ is obtained with the tuple $\mathbf{x} = [0.5, 0.5]$ while $\underline{\Delta}^{\mathbb{Z}} = 2$ is obtained with the tuple $\mathbf{x} = [1, 0]$ or $\mathbf{x} = [0, 1]$.

Example 11 showed that when every domain overlaps the mean, the lower bound $\underline{\Delta}^{\mathbb{Q}}$ for the deviation 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 8. 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 8 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[3]:

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

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

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.

- 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 3.6). 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 $\mathcal{O}(n)$ to compute the bound-consistent lower bound $\underline{\Delta}^{\mathbb{Z}}$.

Lemma 4. 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:

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

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. □

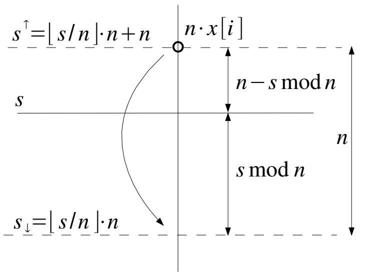

$s^{\uparrow}=\lfloor s/n \rfloor \cdot n+n$
$n \cdot x[i]$
$n - s \bmod n$
s
n
$s \bmod n$
$s_{\downarrow}=\lfloor s/n \rfloor \cdot n$

Figure 3.6: 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 $s^{\downarrow}$.

Example 12 (Computing $\underline{\Delta}^{\mathbb{Z}}$). Assume six variables with domain bounds represented on Figure 3.7 and given in the following table:

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 3.7 (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

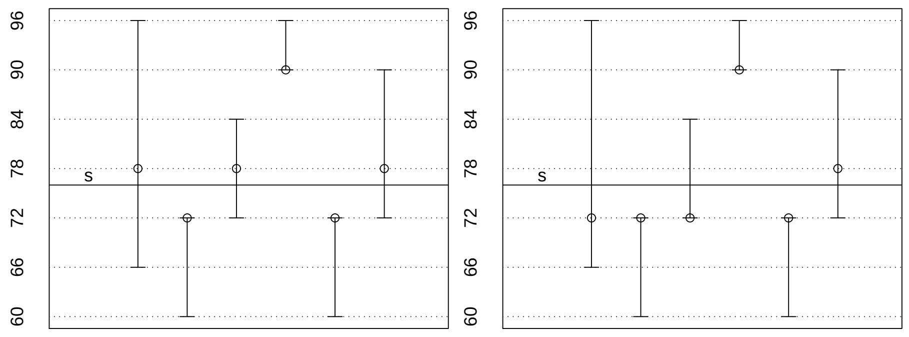


Figure 3.7: Illustration of Example 12 to compute $\underline{\Delta}^{\mathbb{Z}}$. Horizontal lines represents the multiples of $n = 6$. On the left, the result of the greedy part and on the right the result of the repair part are represented with symbol $\circ$

$n \cdot x[6]$. The decrease by $n = 6$ of any two of them is sufficient to satisfy the sum constraint. The right of Figure 3.7 shows the final tuple $n \cdot x$. The value of $\underline{\Delta}^{\mathbb{Z}}$ is then $\sum_{j=1}^{n} |n \cdot x[j] - s| = 32$.

Filtering of X

As illustrated in the following example, the relaxing assumption of rational interval domains can lead to miss some possible filtering with respect to a $\mathbb{Z}$-bound-consistent filtering on $\mathbf{X}$.

Example 13 (Filtering of $\mathbf{X}$). 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 $\mathbb{Q}$-bound consistent filtering would achieve $Dom(X_i) = [-1..2]$ while a $\mathbb{Z}$-bound-consistent filtering would give $Dom(X_i) = [0..1]$.

This section explains how to compute $\overline{X}_i^{\mathbb{Z}}$ the maximum value in $I_D^{\mathbb{Z}}(X_i)$ consistent with $\texttt{deviation}(\mathbf{X}, s, \Delta)$. 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 to increasing values of its extended domain $I_D^{\mathbb{Z}}(X_i)$.
- For each value compute $\underline{\Delta}^{\mathbb{Z}}$.
- $\overline{X}_i^{\mathbb{Z}}$ is the largest value in $I_D^{\mathbb{Z}}(X_i)$ 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 size of the largest domain over $\mathbf{X}$ and $\mathcal{O}(e \cdot n^2)$ for all variables in $\mathbf{X}$.

A better algorithm is possible to lower the complexity to $\mathcal{O}(n)$. Indeed, for each variable X_i, it is possible to compute a function over the domain interval $I_D^{\mathbb{Z}}(X_i)$ giving for each value the minimum deviation if X_i were assigned to that value. We will see that 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 3.8). In the following we show how the two segments composing the function can be computed for every variables in $\mathcal{O}(n)$ allowing the Z-bound consistent filtering of $\mathbf{X}$ in $\mathcal{O}(n)$.

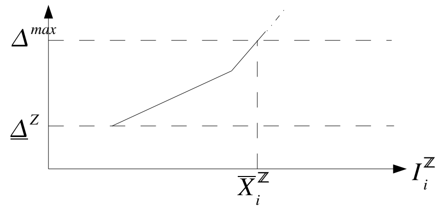


Figure 3.8: Computation of $\overline{X}_i^{\mathbb{Z}}$ on basis of the minimum deviation function defined on $I_D^{\mathbb{Z}}(X_i)$.

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

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

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 3.

Property 3.

- 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 3 can be verified starting from an assignment obtained from the greedy+ repair algorithm from previous section 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 3 is considered in turn in the next three paragraphs giving the evolution of the minimum deviation on $I_D^{\mathbb{Z}}(X_i)$ 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 3). 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 14. 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 3.9 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 3.9). 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 $\bullet$)

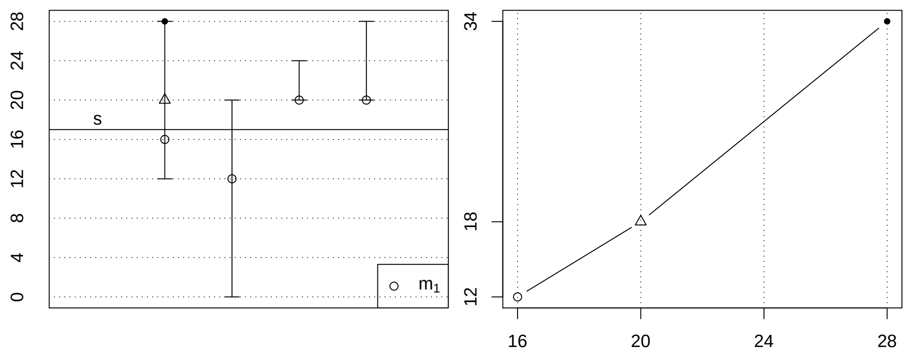


Figure 3.9: Figure of Example 14. On the left is the representation of m_1 with symbols $\circ$ and the successive values of $m_1[1]$. On the right is the evolution of the deviation with the successive values of $m_1[1]$.

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 $\mathcal{O}(1)$ on basis of two information's about m_i:

- $m_i[i]$.
- $o_i = \#\{m_i[j] \big| j \neq i \text{ and } m_i[j] = s^{\uparrow} \text{ and } m_i[j] > n \cdot X_j^{\min}\}$. 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 15. 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 3.10 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 3.10.

Computation of the evolution of the minimum deviation for every variable

The previous subsection explains how the minimum deviation on $I_D^{\mathbb{Z}}(X_i)$ 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.

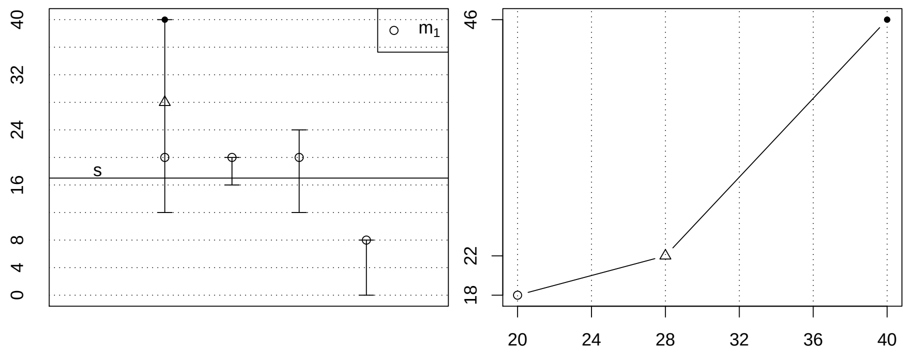


Figure 3.10: Figure of Example 15. On the left is the representation of m_1 and the successive values of $m_1[1]$. On the right is the evolution of the deviation with the successive values of $m_1[1]$.

- $m_i[i] < s^{\downarrow}$: 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 8 computes $m[i]$ and $o[i]$ for $1 \leq i \leq n$ in $\mathcal{O}(n)$. 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.

Algorithm 8: Compute m and o

```
$nx, m, o$ integer arrays of size n;
$sum \leftarrow 0$                                   /* $\sum_{i=1}^{n} nx[i]$ */;
$s^* \leftarrow (s - s^{\downarrow}) \leq (s^{\uparrow} - s)\ ?\ s^{\downarrow}\ :\ s^{\uparrow}$   /* cl. mul.of $n$ 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
                $nx[i] \leftarrow nx[i] + \delta$ ; $sum \leftarrow sum + \delta$ ;
        $overlaps \leftarrow$
                $\{i | nx[i] = s^{\uparrow} > n \cdot X_i^{\min}$ 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 24-31 consider the case where the sum constraint could be satisfied after modifications of lines 15-23.

- 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 8 has a time complexity of $\mathcal{O}(n)$. Indeed, in all cases a constant number of operations is performed for each variable.

Example 16. 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 $o = (1, 2, 1, 2, 2, 1)$

3.5.3 Experimental Comparison of the $\mathbb{Q}$-bound-consistent and $\mathbb{Z}$-bound-consistent propagators

As we did for `spread` in Section 3.4.3, we propose to experiment if the $\mathbb{Z}$-bound-consistent propagators make a big difference compared to the simpler ones for the $\mathbb{Q}$-bound-consistency. Note that the $\mathbb{Q}$-bound-consistent propagators are fairly easy to implement in about 40 lines of C++ code. On the contrary, the $\mathbb{Z}$-bound-consistent ones require about 450 lines of C++ code and are much more difficult to implement. The difference of difficulty of implementation of the $\mathbb{Z}$ and $\mathbb{Q}$-bound-consistent propagators for `deviation` is much more important than the difference of difficulty of the $\mathbb{Z}$ and $\mathbb{Q}$-bound-consistent propagators for `spread`. This is because for `deviation`, the $\mathbb{Z}$-bound-consistent algorithm is completely different than the $\mathbb{Q}$-bound-consistent one and is not simply an improvement as it is the case for `spread`.

One can see on Table 3.4 that the $\mathbb{Z}$-bound-consistent propagators allow to solve significantly more instances. For all the 14 instances that could be solved by the $\mathbb{Q}$-bound-consistent propagators, the mean is either an integer or at a distance of 1/12 from an integer. This is not surprising since the $\mathbb{Z}$ and $\mathbb{Q}$-bound-consistent propagators are completely equivalent when the mean is an integer and are very close in terms of pruning when the mean is almost an integer. Unfortunately there is no reason that all the instances present this property. The second line of the results is also interesting since it shows that, most of the time, the $\mathbb{Q}$-bound-consistent propagators find a solution which is as good as the one obtained with the $\mathbb{Z}$-bound-consistent propagators but the gap of 20 instances remains important. Interestingly the $\mathbb{Z}$-bound-consistent propagators do not obtain 100/100. This can be explained by several reasons. The two instances where the $\mathbb{Q}$-bound-consistent propagators were strictly better than the $\mathbb{Z}$-bound-consistent propagators could not be solved and proved optimal by neither approach. The difference can mainly be explained because the search tree explored is not necessarily the same. The heuristic is non-deterministic hence two different levels of propagation can lead to different search trees. Note also that the size of the search tree explored in the given timeout with the $\mathbb{Z}$-bound-consistent propagators might be smaller because the algorithm, even if linear, is more costly.

Table 3.4: Results on the 100 BACP instances.

	decomposition	`deviation` $\mathbb{Q}$-BC	`deviation` $\mathbb{Z}$-BC
#solved	0	14	69
#best	6	78	98

3.6 Applications using Spread and Deviation

In this section we apply `spread` and `deviation` on two real applications namely the Assembly Line Balancing Problem [4] (ALBP) and the problem of Fair Assignment of Nurses to Patients in a hospital (FANP) [30].

3.6.1 Assembly Line Balancing Problem

In this simplest form, this problem is to assign tasks to a given number of workstations such that the precedences between the tasks are satisfied. The classical objective is to minimize the maximum workload among the stations. This problem is called the Simple Assembly Line Balancing Problem of type II (SALBP2). An alternative objective is to equalize as much as possible the workloads. This problem is called the Vertical SALBP2 (VSALBP2) (see [4] for a detailed classification of assembly line balancing problems).

We are interested in this section to the VSALBP2. For this problem, the L_1 balancing criterion has already been applied on the vertical ALBP in [33] in a heuristic procedure. The L_2 criterion has also been used in [38] with genetics algorithms. To the best of our knowledge, no exact method to optimize the balancing with respect to L_1 or L_2 has never been applied on the VSALBP2. We experiment `spread` and `deviation` on some real instances coming from the Scholl benchmark data set [48] to solve exactly the VSALBP2 with constraint programming.

The heuristic

Very efficient procedures exist for the initialization of a local search procedure for the SALBP2 [50]. We suggest to reuse such a method to build an initial solution and to use this initial solution to drive the depth first search rapidly toward high quality solutions. Before going further into the explanation of the CP-heuristic we explain the algorithm to build a good initial solution.

Let us introduce some vocabulary from [50] specific to the assembly line balancing problems allowing to describe concisely the procedure of initialization:

- c is the cycle time. All the stations have a duration lower or equal to it. SALBP2 try to minimize the cycle time.
- t_j is the task time for $j \in [1..n]$
- t_{sum} is the sum of all task times ($\sum_j t_j$)
- $t_{\max}$ is the maximum task time
- P_j is the set of tasks which must precede task j immediately (predecessors)
- F_j is the set of tasks which must follow task j immediately (successors, followers)
- P_j^* is the set of all tasks which must precede task j
- F_j^* set of all tasks which must follow task j
- m is the number of stations
- $E_j = \lceil (t_j + \sum_{h \in P_j^*} t_h)/c \rceil$ earliest station for task j.
- $L_j = m - \lceil (t_j + \sum_{h \in F_j^*} t_h)/c \rceil + 1$ latest station to which task j can be assigned.
- S_k is the set of tasks which are currently assigned to station $k \in [1..m]$
- $t(S_k)$ is the station time of station k, $t(S_k) \leq c$

The approach to build a good initial solution is to use a dichotomic search method on the cycle time using a greedy procedure as sub-routine to shrink the cycle time interval. The greedy procedure solves a satisfaction problem heuristically: for a given cycle time is there a feasible solution ? We first detail the greedy procedure given in Algorithm 9 then we explain the dichotomic search using it.

The greedy procedure *findFeasible* given in Algorithm 9 assigns the tasks in a topological order from the left most workstation to the right most one passing to the next when no tasks can be assigned to the current one without exceeding the cycle time c. For each of the n iterations, the algorithm pick up a task such that all its direct predecessors have already been placed (line 7 or 12). Since more than one task can satisfy this property, line 7 chose heuristically a task with the largest duration. The idea is that largest task should be preferred earlier to allow more flexibility in the following stations. Indeed, it is preferable to terminate with the smallest tasks and, when actually line 7 places the last task in the station before moving to the next one, it is better to fill it as much as possible. In line 12, the current station is empty and we chose a task with the most successors in the transitive closure of the precedence graph. The idea is that many tasks must still be placed after this one hence we prefer to assign it in an early workstation. Since it is probably not the last one of the station, there is not reason to choose the largest one at this point. The algorithm returns *true* when all the tasks have been assigned. It returns *false* in line 11 when more stations than the available number m are necessary to assign the remaining tasks in C. The algorithm *findFeasible* can easily be randomized by replacing the $\text{argmax}_{k \in C}\{t_k \mid ...\}$ with a $\text{argmax}[p]_{k \in C}\{t_k \mid ...\}$ which means that instead of taking a task with maximum duration, one task is randomly chosen among the p largest ones [4] (we use $p = 5$). Algorithm *findFeasible* is very fast (in $O(n)$) but it can return false negative. We propose to use its randomized version several times to (we hope) reduce the risk of false negative answers. Instead of executing *findFeasible* only once, its randomized version is called several times and $false$ is returned only if all the answers are negative (we call it 5 times). We call this modified procedure *findFeasible**.

We use the binary search from [50] using the *findFeasible** algorithm as subroutine. The search maintains two values: LB (lower bound) and UB (upper bound). The UB value is the smallest cycle time for which we have a solution and LB is the value for which we think there are no solution with a cycle time smaller than it. Initially we set LB to $\lceil \sum_j t_j/m \rceil$ and UB to $\sum_j t_j$ (solution with every task in the same station). The binary search shrinks the interval $[LB, UB]$ dichotomically until $LB = UB$. Each iteration, a value for the cycle time $c = \lfloor (LB + UB)/2 \rfloor$ is tried with the algorithm *findFeasible**. If

[4]This is achieved with the `selectMax[p]` selector in Comet

Algorithm 9: findFeasible

```
   Data:  $c$ the cycle time with $c \geq t_{\max}$
   Result:  $true$ and a feasible solution $sol$ if such a one is found,
            $false$ otherwise
1  $front \leftarrow 1$                      /* the current workstation */
2  $C \leftarrow \{1, ..., n\}$  /* candidate tasks i.e.  not yet placed */
3  $sol[1..n]$          /* station where each task is placed */
4  $j \leftarrow -1$
5  forall $iter \in [1..n]$ do
6      if $\{t_k \mid t(S_{front}) + t_k \leq c \ \wedge \ P_j \cap C = \phi\} \neq \phi$ then
7          $j \leftarrow \text{argmax}_{k \in C}\{t_k \mid t(S_{front}) + t_k \leq c \ \wedge \ P_j \cap C = \phi\}$
8      else
9          $front \leftarrow front + 1$
10         if $front > m$ then
11             return $false$
12         $j \leftarrow \text{argmax}_{k \in C}\{|F_k^*| \mid \ P_j \cap C = \phi\}$
13     $sol[j] \leftarrow front$
14     $C \leftarrow C \setminus \{j\}$
15 return $true, sol$
```

Table 3.5: Results of the initialization on the 302 SALBP2 instances from [48]

	# best	av. dev. (%)	max. dev. (%)
results from [50]	34	2.29	10.67
our results	47	1.56	7.69

*findFeasible** succeeds to find a solution, UB is set to the cycle time of the solution otherwise LB is set to $c+1$.

The results obtained for the initialization on the 302 instances of the benchmarks from [48] are presented on Table 3.5 and are compared with the best results obtained with the initializations from [50]. The reported measures are the same as in 3.5 that are: the number of times the best known solution is reached, the average deviation from the best known solution and the maximum deviation of the best known solution among all the instances. The comparison is very fair since in [50] they experiment 13 different initializations and we report for each measure the best one obtained among the 13. For the three criteria, our results are better. Note that it is also an encouraging results for the local search methods which are highly dependent on the quality of the initialization on this problem.

Recall that we are not interested to solve the SALBP2 problem but the VSALBP2. We are interested in good initializations for the SALBP2 because they are also good for the VSALBP2. Indeed, minimizing the cycle time also tends to equalize the durations of the station. The fragment of Comet code 3.2 gives the heuristic implementation. The decision variables are the vector `ws[]` that is for each tasks the station it is assigned to. The next variable chosen for the instantiation is the one which impacts on most of the tasks (measured with $P_i^* + F_i^*$) in the precedence graph and ties are broken by preferring the tasks with the smallest domain (first fail). For the value heuristic, if the station of the initial solution is still present in the domain of the task, it is first placed into this station (lines 4-7). If on the contrary this station is not present, the stations are tried by increasing load (lines 8-13). The illustration of the heuristic on the placement of the first 11th tasks is illustrated on the Figure 3.11 with a Comet visualization. The precedence graph is on top of the figure with bigger circles for the tasks already placed.

Listing 3.2: Heuristic for the VSALBP

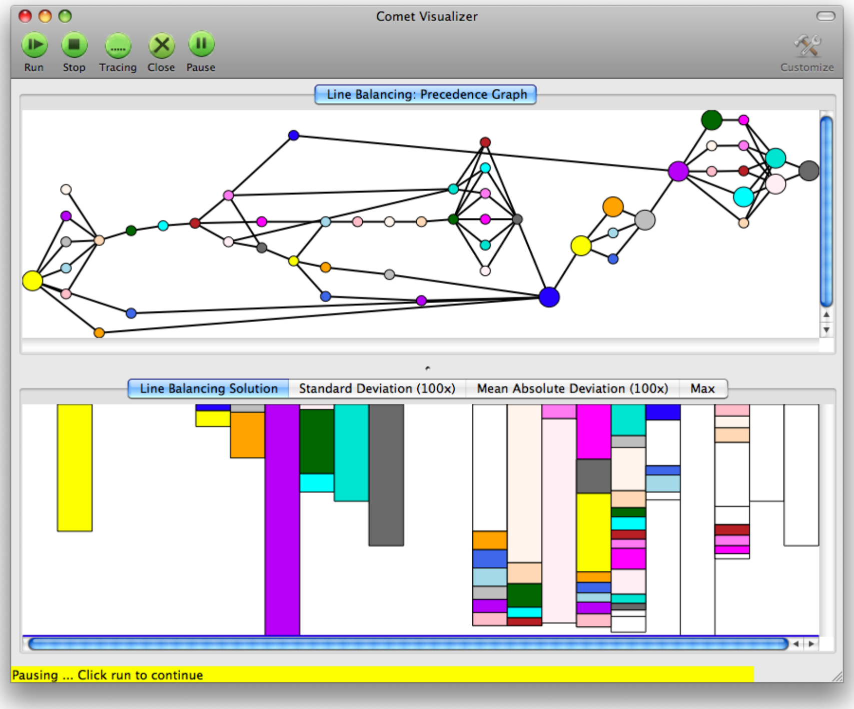
```
while(!bound(ws)){
   selectMax(i in tasks:!ws[i].bound())
          (predAll[i].getSize()+succAll[i].getSize(),−ws[i].getSize()){
      if(ws[i].memberOf(initial[i])){
           try<cp>{cp.post(ws[i]==initial[i]);} |
                    {cp.post(ws[i]!=initial[i]);}
      }
      else{
         tryall<cp>(s in stations) by(load[s].getMin())
             cp.post(ws[i]==s);
         onFailure
             cp.post(ws[i]!=s);
      }
   }
}
```

On the bottom right is the initial solution. When a task is placed, it is removed from the initial solution and moved in the partial CP solution on the bottom left. By construction our heuristic first dives to rebuild exactly the initial solution. What is interesting to see is which task in the precedence graph are first assigned.

As it is shown on an numeric example on Table 3.1, minimizing one balancing criteria does not always mean minimizing the others. Nevertheless we think that the least squares criterion L_2 of `spread` and the mean absolute deviation L_1 of `deviation` are quite similar. One question that occur is: How good is the minimization of L_2 with respect to L_1 and vice versa?

As first experiment, we minimize L_2 with `spread` on the *Hahn* instance with 10 stations from [48] and we report on Figure 3.12, for each solution encountered during the Branch and Bound, respectively the standard deviation[5], the mean absolute deviation and the maximum (called cycle time on this application). Of course, the standard deviation decreases strictly since it is the objective function to minimize. More interesting is that, as expected, the mean absolute deviation and

[5]the standard deviation is illustrated instead of the variance such that it can be illustrated on the same axis as the mean absolution deviation.


Comet Visualizer
Run
Stop
Tracing
Close
Pause
Customize
Line Balancing: Precedence Graph
Line Balancing Solution
Standard Deviation (100x)
Mean Absolute Deviation (100x)
Max

Pausing ... Click run to continue

Figure 3.11: Illustration of the heuristic behavior for the VSALBP2 with a Comet visualization.

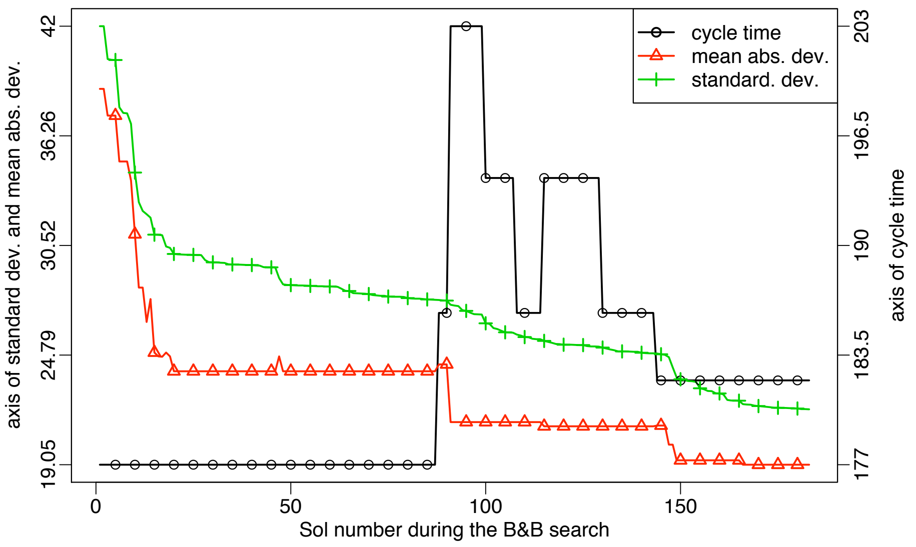


Figure 3.12: Illustration of the evolution of the standard deviation, the mean absolute deviation and the cycle time when minimizing L_2.

the cycle time do not strictly decrease along the solutions. Globally the allure of the mean absolute deviation curve is decreasing but this is not the case for the cycle time. Indeed sometimes the cycle time increases by a large amount during the search. This illustrates the close proximity of `spread` and `deviation` and their difference with respect to the minimization of the maximum. This also illustrates that the VSALBP2 and the SALBP2 are two distinct problems and that SALBP2 can only be considered as a rough approximation of the VSALBP2.

As next experiment we continue to compare `spread` and `deviation` on the VSALBP2. We chose several instances among the largest ones from [48]. The names and the number of tasks of the precedence graphs we will work on are given on Table 3.6. For each of these precedence graphs we solve the problem for 6, 8 and 10 workstations with `spread` and `deviation` with a timeout of 200 seconds and we report on Table 3.7 the standard deviation (sd), the mean absolute deviation (mad), the time, the number of fails and if the objective was proved optimal or not.

The minimization of L_2 gives for 36/36 instances, the solution with the smallest sd, and for 26/36 instances, the minimization of L_1 gives minimum sd. The minimization of L_1 gives for 32/36 instances the

Table 3.6: Names and number of tasks of selected instances from [48]

instance	# tasks
BUXEY	29
GUNTHER	35
HAHN	53
LUTZ1	32
LUTZ2	89
LUTZ3	89
MITCHELL	21
ROSZIEG	25
SAWYER30	30
TONGE70	70
WARNECKE	58
WEE-MAG	75

minimum mad and the minimization of L_2 gives for 35/36 instances the minimum mad. This surprising result is due to the instances that could not be terminated in the due time. The minimization of L_2 is a more radical criterion (more sensitive to outliers). Consequently it generally leads to smaller search trees as can be seen with the number of fails. When a timeout occurs in both search trees of L_1 and L_2, the last solution in the tree of L_2 is of better quality for L_2 but also for L_1 when compared to the last solution in the search tree of L_1. Also because the search tree seems smaller with L_2, two instances could only be solved by L_2 (GUNTHER 10 and LUTZ3 10).

The conclusion of this experiment is that we would prefer using `spread` rather that `deviation` for the VSALBP2 because it produces smaller search trees and also gives excellent results for L_1 (sometimes even better when the search is interrupted by a timeout).

3.6.2 Nurses to patients assignment problem

This paper considers the daily assignment of new born infant patients to nurses in a hospital described in [30]. In this problem, some infants require little attention, while others need significant care. The amount of work required by the infant during one shift is called the *acuity*. A

Table 3.7: Comparison of spread and deviation on VSALBP2 instances

inst. caract.		**Minimization of L_2**					**Minimization of L_1**				
name	st.	opt.	fails	time	sd.	mad.	opt.	fails	time	sd.	mad.
BUXEY	6	1	4	0.2	0.58	0.33	1	3	0.2	0.58	0.33
BUXEY	8	1	5	0.2	0.50	0.50	1	5	0.2	0.50	0.50
BUXEY	10	1	464	1.1	0.80	0.68	1	1720	2.3	0.80	0.68
GUNTHER	6	1	8	0.4	1.61	1.17	1	7	0.4	1.61	1.17
GUNTHER	8	1	11250	11.3	1.58	1.12	1	13128	14.0	1.58	1.12
GUNTHER	10	1	49706	45.7	0.90	0.76	0	209005	200.0	1.27	1.10
HAHN	6	1	164	1.6	4.41	3.83	1	207	1.5	4.71	3.83
HAHN	8	1	89	1.9	22.36	15.69	1	102	1.9	25.45	14.69
HAHN	10	0	165646	200.0	27.69	23.94	0	133012	200.0	30.17	23.94
LUTZ1	6	1	61	0.4	55.12	40.22	1	62	0.4	55.70	40.22
LUTZ1	8	1	328	0.7	57.48	50.00	1	1150	1.2	57.48	50.00
LUTZ1	10	1	135	0.6	73.06	53.60	1	728	0.9	73.06	53.60
LUTZ2	6	1	2	6.5	0.37	0.28	1	2	6.4	0.37	0.28
LUTZ2	8	1	18	7.5	0.48	0.47	1	19	7.6	0.48	0.47
LUTZ2	10	1	171	10.2	0.50	0.50	1	1031	10.6	0.50	0.50
LUTZ3	6	1	480	10.1	0.58	0.33	1	32692	39.5	0.58	0.33
LUTZ3	8	0	195136	200.0	3.08	2.62	0	213777	200.0	3.08	2.62
LUTZ3	10	1	167389	162.7	0.66	0.60	0	201979	200.0	4.65	3.60
MITCHELL	6	1	1	0.1	0.50	0.50	1	1	0.1	0.50	0.50
MITCHELL	8	1	0	0.1	0.60	0.44	1	0	0.1	0.60	0.44
MITCHELL	10	1	39	0.2	1.50	1.20	1	59	0.2	1.50	1.20
ROSZIEG	6	1	1	0.1	0.37	0.28	1	1	0.1	0.37	0.28
ROSZIEG	8	1	9	0.2	0.70	0.56	1	10	0.2	0.70	0.56
ROSZIEG	10	1	111	0.3	1.36	1.00	1	114	0.3	1.36	1.00
SAWYER	6	1	7	0.3	0.58	0.33	1	5	0.3	0.58	0.33
SAWYER	8	1	8	0.3	0.50	0.50	1	13	0.3	0.50	0.50
SAWYER	10	1	1285	2.1	0.80	0.68	1	5605	6.1	0.80	0.68
TONGE	6	0	61377	200.0	0.58	0.33	0	61958	200.0	0.58	0.33
TONGE	8	0	83209	200.0	2.38	2.25	0	55048	200.0	2.54	2.25
TONGE	10	0	54730	200.0	2.05	1.60	0	108290	200.0	3.07	2.40
WARNECKE	6	1	842	4.1	0.00	0.00	1	372	3.2	0.00	0.00
WARNECKE	8	0	122976	200.0	2.96	2.12	0	106093	200.0	3.16	2.12
WARNECKE	10	0	92548	200.0	1.54	1.08	0	90722	200.0	1.54	1.08
WEE-MAG	6	1	21	3.8	0.37	0.28	1	98	3.8	0.37	0.28
WEE-MAG	8	1	5	3.6	0.48	0.47	1	5	3.6	0.48	0.47
WEE-MAG	10	0	100877	200.0	0.94	0.72	0	96489	200.0	1.30	1.10

nurse is in charge of a group of infants and the total amount of acuity is the workload of the nurse during that shift. For ensuring an optimal care quality and perceived fairness for the nurses, it is essential to balance the workload. In addition, the problem features various side constraints:

- A nurse can work in only one zone, but the patients are located in p different zones.
- A nurse cannot be responsible of more than $\mathit{children}^{\max}$ infants.
- The total amount of acuity of a nurse cannot exceed $\mathit{acuity}^{\max}$.

The balance objective and the various constraints make it very difficult to find a good solution in a reasonable time. Since nurses only work in one zone, the number of nurses assigned to each zone has already a huge impact on the quality of the balancing. In [30], the problem was tackled using a MIP model, but the results were not satisfactory. In this paper, we present a series of increasingly sophisticated constraint programming models in order to reach the required solution quality and scalability.

We start by presenting the instances proposed in [30] then we describes the MIP model and its limitations. A first CP model able to solve two-zones instances is introduced. This model is then improved with a two-step approach that first assigns the nurses in each zone and then assign the infants to nurses to balance the load optimally. Finally we show that the second step can be decomposed by zones without losing the optimality guarantees. This final model is instrumental in solving large instances with dozens of zones and hundreds of patients.

Problem Instances

Reference [30] specifies a statistical model to generate instances very similar to their real instances. This statistical model was also used to measure the robustness of their solution technique with respect to the number of nurses, the number of infants, and the number of zones. The model contains a single parameter: the number of zones. The maximum acuity per nurse is fixed to $\mathit{acuity}^{\max} = 105$ and the maximum number of infants per nurse is fixed to $\mathit{children}^{\max} = 3$. The instance generator fixes the number of nurses, the number of infants, the acuity, and the zone of each infant. The different steps to generate an instance are as follows:

- The number of patient in a zone is obtained with a Poisson random variable with mean 3.8 and offset by 10.
- The acuity Y of a patient is obtained by first generating a number $X \sim Binomial(n = 8, p = 0.23)$ and then the number $Y \sim Unif(10 \cdot (X + 1), 10 \cdot (X + 1) + 9)$.
- The number of nurses made available is obtained by solving a First Fit Decreasing (FFD) procedure in each zone. The total number of nurses available is then the sum of the amount of nurses found in each zone by the FFD procedure. The FFD procedure starts by ranking the patients in decreasing acuity. Then the patient with the highest acuity is assigned to the first nurse. The next infants are assigned successively to the first nurse that can accommodate them without violating the maximum acuity and the number of patient constraints.

The MIP Model

In this section, we recall the main variables of the MIP model from [30]. We also describe the limitation of the MIP model and suggest that a CP approach may address them. Due to space reasons, we do not reproduce the entire MIP model but readers can consult [30] for more details. The technical details presented here are sufficient for our purposes. The MIP model contains four family of variables:

1. $X_{ij} = 1$ if infant i is assigned to nurse j and 0 otherwise;
2. $Z_{jk} = 1$ if nurse j is assigned zone k and 0 otherwise;
3. $Y_{k,\max}$ is the maximum acuity of a nurse in zone k;
4. $Y_{k,\min}$ is the minimum acuity of a nurse in zone k.

All these variables are linked with linear constraints to enforce the constraints of the problem. The objective function, we call the *range-sum* criteria, consists of minimizing the sum of the acuity ranges of the p zones, i.e.,

$$\sum_{k=1}^{p} (Y_{k,\max} - Y_{k,\min}).$$

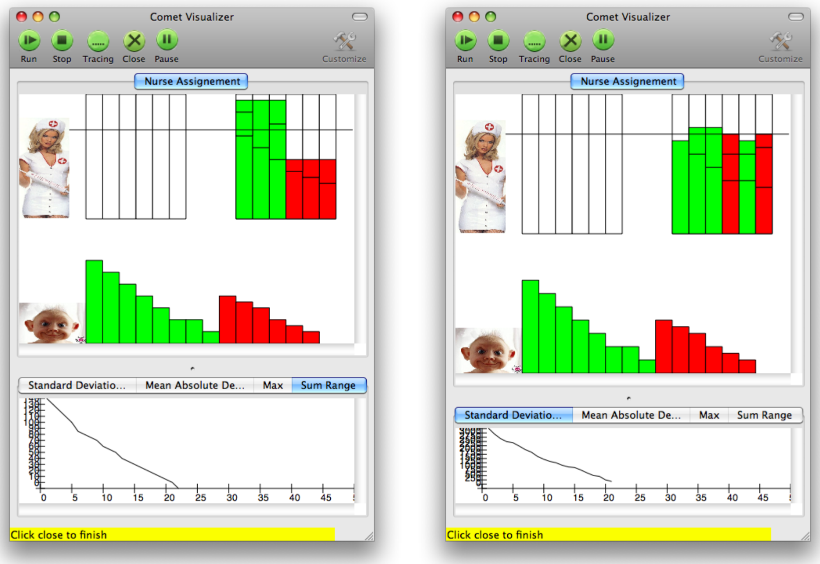


Figure 3.13: Comparison of Two Solutions on a 6 Nurses, 14 Infants, and 2 zones Problem. The left solution is obtained by minimizing the range-sum criteria. The right solution is obtained by minimizing the L_2 norm presented subsequently.

The MIP model has a fundamental limitation: The objective function may produce poorly balanced workloads. It tends to equalize the workload inside the zones but may produce huge differences among the workload of different zones. This is illustrated in Figure 3.13. The workloads are depicted in the upper right corner of each Comet visualization. The left solution is obtained by minimizing the range-sum criteria and the right solution by minimizing the L_2 norm to be defined precisely in the next section. The range-sum objective is minimal on the left because the workloads inside each of the two zones are identical. Unfortunately, nurses in the first zone work twice as much as those in the second zone. The right solution is obtained by minimizing the L_2 norm and is significantly more appealing. *This illustrates clearly that "the high level objective that all nurses should be assigned an equal amount of patient acuity" [30] is not properly captured with the range-sum criteria.*

It is not immediately obvious how to remedy these problems. The L_2 criteria presented in the next section is non-linear and is not easily modeled in a MIP approach. In addition, a CP approach may exploit the combinatorial structure in the bin-packing and the side-constraints, while the MIP relaxation is generally bad for bin-packing like problems. Finally, there are important symmetries that are not removed in their model: For a given solution, the nurses are completely interchangeable.

We now turn to the CP approaches after having reviewed load balancing constraints in constraint programming.

A Basic CP model.

We now present a CP model which addresses all the issues raised for the MIP model. Let m be the number of nurses, n the number of patients, and a_i be the acuity of patient i. The set of patients in zone k is denoted by $\mathcal{P}_k$ and $[\mathcal{P}_1, ..., \mathcal{P}_p]$ forms a partition of $\{1, ..., n\}$, i.e.,

$$\forall 1 \leq k_1, k_2 \leq p : \mathcal{P}_{k_1} \cap \mathcal{P}_{k_2} = \phi \tag{3.13}$$

$$\bigcup_{1 \leq k \leq p} \mathcal{P}_k = \{1, ..., n\}. \tag{3.14}$$

For each patient i, we introduce a decision variable $N_i \in [1..n]$ which represents her/his nurse. The workload of nurse j is the variable $W_j \in [0..\mathit{acuity}^{\max}]$ and the number of patients of nurse j is $C_j \in [1..\mathit{children}^{\max}]$. The complete model in COMET is shown in Listing 3.3. The objective and constraints are expressed as follows. (We use shorter names in the text).

- The objective to minimize is the L_2 norm modeled using the `spread` constraint over the workload variables $[W_1, ..., W_m]$ (lines 12–13 and 15).
- To express that nurses have a total acuity of at most $\mathit{acuity}^{\max}$, we link the variables N_i, W_j, and the acuities with a global packing/-multiknapsack constraint `Pack`$([N_1, ..., N_n], [a_1, ..., a_n], [W_1, ..., W_m])$ [52] (line 16).
- A global cardinality constraint [34] is used to model that a nurse takes care of at most $\mathit{children}^{\max}$ infants for each nurse (line 17).
- The constraint that a nurse can work in at most one zone is modeled by introducing set variables representing the set of nurses

Table 3.8: Patients to Nurses Assignment Problem with 2 zones and minimization of L_2 with `spread`.

m	n	#fails	time(s)	avg workload	sd. workload
11	28	511095	170.2	86.09	2.64
11	29	1126480	302.0	80.27	1.76
10	26	104931	24.7	76.50	2.29
12	30	259147	136.5	83.42	1.93
10	28	2990450	1138.5	91.80	6.84
10	26	779969	206.9	88.40	2.29
12	29	555243	198.2	80.08	2.72
10	27	931858	343.9	90.60	5.33
10	25	1616689	434.5	82.70	7.32
8	22	4160	1.2	87.50	3.12

working in each zone $NS_k = \bigcup_{i \in \mathcal{P}_k} N_i$. The set NS_k is maintained with a global constraint `unionOf`. Then the pairwise empty intersections between the set variables are enforced with a global disjoint constraint. The Comet uses a reformulation with channeling constraints and a global cardinality constraint as explained in [35] p38 or in [2] (line 18).

The search is implemented in the `using` block in lines 20–29. The search dynamically breaks the value symmetries originating from the nurse interchangeability. The patient having the largest acuity is selected first in lines 21–22. Then the search tries to assign a nurse to this patient starting first with those with the smaller load (lines 24–27). The symmetry breaking is implemented by considering already assigned nurses and at most one nurse without any patient yet (a similar technique was used for the steel mill slab problem in [16]). Value `mn` is the maximal index of a nurse already assigned to a patient. The `tryall` statement considers all the nurse indexes until `mn+1` (nurse `mn+1` having currently no patient).

As a first experiment we generate 10 instances with 2 zones as was the case for the real instances studied in [30]. These instances have about 10–15 nurses, 20–30 infants, and cannot be solved by the MIP model from [30] minimizing the range-sum objective. All the instances could be solved optimally with our Comet model in less than 30 minutes (the

Listing 3.3: Patient-Nurse Assignment Model

```
var<CP>{int} nurseOfPatient[patients];
var<CP>{int} acuityByNurse[1..nbNurses](cp,1..105);
var<CP>{int} spreadAcuity(cp,0..System.getMAXINT());
var<CP>{int}[] nurseOfPatientByZone[zones];
forall(i in zones) {
    nurseOfPatientByZone[i] =
        new var<CP>{int}[1..nbPatientsInZone[i]](cp,nurses);
}
int k = 1;
forall(i in zones,j in 1..nbPatientsInZone[i])
    nurseOfPatient[k++] = nurseOfPatientByZone[i][j];
minimize<cp>
   spreadAcuity
subject to {
   cp.post(spread(acuityByNurse,totAcuity,spreadAcuity));
   cp.post(multiknapsack(nurseOfPatient,acuity,acuityByNurse));
   cp.post(cardinality(minNbPatients,nurseOfPatient,maxNbPatients));
   cp.post(pairwiseDisjoint(nurseOfPatientByZone,nurses));
}
using {
  forall(p in patients: !nurseOfPatient[p].bound())
                        by (−acuity[p],nurseOfPatient[p].getSize()) {
    int mn = max(0,maxBound(nurseOfPatient));
    tryall<cp>(n in nurses: n <= mn + 1)
                             by (acuityByNurse[n].getMin())
        cp.label(nurseOfPatient[p],n);
    onFailure
        cp.diff(nurseOfPatient[p],n);
  }
}
```

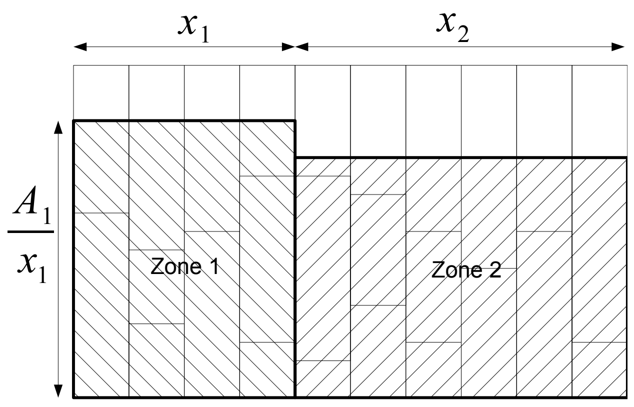


Figure 3.14: Illustration of a solution of the relaxation solved to find the number of nurses in each zone.

time constraint specified in [30] by the hospital to find the assignment). Table 3.8 depicts the experimental results.

A Two-Step CP Model

The basic CP model can solve 2-zones instances but has great difficulty to solve the problem for 3 zones or more. We propose to simplify the resolution by pre-computing the number of nurses that are assigned to each zone. This facilitates the resolution by

1. removing one degree of flexibility which is the number of nurses in each zone.

2. removing the disjointness constraint since the set of nurses that can be assigned to each patient can be pre-computed.

The drawback is that the resulting solution might be sub-optimal if these numbers are not the right ones. Indeed, the number of nurses assigned to each zone has a crucial impact on the quality of the balancing. However, after visualizing some optimal solutions, we observed that the workloads of the nurses are extremely well balanced (almost the same) inside the zones.

This suggested solving a relaxation of the problem to discover a good repartition of the nurses to the zones. The relaxation allows the

acuity of a child in a zone to be distributed among the nurses of that zone (continuous relaxation of the acuity). Since the acuity of a child can be split, the relaxed problem will have an optimal solution where the nurses of a zone have exactly the same workload $\frac{A_k}{x_k}$, i.e., the total acuity $A_k = \sum_{i \in \mathcal{P}_k} a_i$ of the zone k divided by the number of nurses x_k in that zone k. This is schematically illustrated on Figure 3.14 for a two-zone relaxation problem. The reason that the optimal solution of the relaxation will take this configuration is that as long as two workloads can be made closer, the L_2 criterion can be decreased as it was shown in Theorem 2.
Since the optimal solution of the relaxed problem must have a same workload for the nurses inside a same zone, the mathematical formulation of the relaxed problem is the following.

$$\min \quad \sum_{k=1}^{p} x_k \cdot \left(\frac{A_k}{x_k} - \sum_{j=1}^{p} \frac{A_j}{m} \right)^2 \tag{3.15}$$

$$s.t. \quad \sum_{k=1}^{p} x_i = m \tag{3.16}$$

$$x_k \in \mathbb{Z}_0^+ \tag{3.17}$$

The workload of all the nurses of zone k is $\frac{A_k}{x_k}$ and the average workload is $\sum_{j=1}^{p} \frac{A_j}{m}$. Hence the contribution to the L_2 criterion for the x_k nurses of zone k is $x_k \cdot \left(\frac{A_k}{x_k} - \sum_{j=1}^{p} \frac{A_j}{m} \right)^2$. We describe a greedy procedure to solve optimally this optimization problem.

First note that by developing the objective (3.15), it appears that it is equivalent to minimize $\sum_{k=1}^{p} \frac{H_k}{x_k}$ where $H_k = A_k^2$. Hence the reformulated problem to solve is

$$\min f(x_1, ..., x_k) \equiv \sum_{k=1}^{p} \frac{H_k}{x_k} \tag{3.18}$$

$$s.t. \quad \sum_{k=1}^{p} x_i = m \tag{3.19}$$

$$x_k \in \mathbb{Z}_0^+ \tag{3.20}$$

with m integer and $m \geq p$. We can solve this problem optimally with a $O(p + m \cdot \log p)$ greedy procedure:

1. Start with $x_i = 1$ for $i \in [1..p]$.
2. Increment x_k by 1 with $k = \text{argmax}_i\{\frac{H_i}{x_i} - \frac{H_i}{x_i+1}\}$.
3. Repeat previous step until $\sum_{i\in[1..p]} x_i = m$.

The complexity is obtained using a heap data structure to select in $O(\log p)$ the $k = \text{argmax}_i\{\frac{H_i}{x_i} - \frac{H_i}{x_i+1}\}$ at each iteration and it takes a linear time $O(p)$ to initialize it.

We proof in Theorem 9 that the procedure reaches an optimal solution. This proof makes use of the next Lemma 5 and its Corollary 2.

Lemma 5. Let $\mathcal{X} = x_1, ..., x_p$ be an assignation with $\forall i \in [1..p]: \ x_i^m \geq 1$. For $k = \text{argmax}_i\{\frac{H_i}{x_i^m} - \frac{H_i}{x_i^m+1}\}$ we have that $\forall\ \alpha \in \mathbb{Z}^+, i \in [1..p]$:

$$\frac{H_i}{x_i+\alpha} - \frac{H_i}{x_i+\alpha+1} \leq \frac{H_k}{x_k} - \frac{H_k}{x_k+1}$$

Proof. By definition of k, $\forall\ i \in [1..p]$ we have $\frac{H_i}{x_i} - \frac{H_i}{x_i+1} \leq \frac{H_k}{x_k} - \frac{H_k}{x_k+1}$. For a given i we have $\frac{H_i}{x_i+\alpha+1} - \frac{H_i}{x_i+\alpha+2} \leq \frac{H_i}{x_i+\alpha} - \frac{H_i}{x_i+\alpha+1}$. By transitivity the lemma follows. □

Corollary 2. Let us consider 4 different assignations:

1. $\mathcal{X}^m = x_1^m, ..., x_p^m$ be an assignation of $x_1, .., x_m$ with
   - $\sum_{i\in[1..p]} x_i^m = m$ and
   - $\forall i \in [1..p]: \ x_i^m \geq 1$.
2. $\tilde{\mathcal{X}}^m = x_1^m, ..., x_{k-1}^m, x_k^m + 1, x_{k+1}^m, ..., x_p^m$ with $k = \text{argmax}_i\{\frac{H_i}{x_i^m} - \frac{H_i}{x_i^m+1}\}$.
3. $\mathcal{X}^{m+1} = x_1^{m+1}, ..., x_p^{m+1}$ with
   - $\sum_{i\in[1..p]} x_i^{m+1} = m+1$,
   - $x_i^{m+1} > x_i$ for $i \in [1..l]$,
   - $x_i^{m+1} \leq x_i$ for $i \in [l+1..p]$ and
   - $\forall i \in [1..p]: \ x_i^{m+1} \geq 1$.
4. $\tilde{\mathcal{X}}^{m+1} = x_1^{m+1} - 1, x_2^{m+1}, ..., x_p^{m+1}$.

We have that $f(\mathcal{X}^m) - f(\tilde{\mathcal{X}}^m) \geq f(\tilde{\mathcal{X}}^{m+1}) - f(\mathcal{X}^{m+1})$.

Proof. $f(\mathcal{X}^m) - f(\tilde{\mathcal{X}}^m) = \frac{H_k}{x_k^m} - \frac{H_k}{x_k^m+1}$ and $f(\tilde{\mathcal{X}}^{m+1}) - f(\mathcal{X}^{m+1}) = \frac{H_k}{x_1^{m+1}-1} - \frac{H_k}{x_1^{m+1}}$. The corollary is then a direct consequence of Lemma 5 by taking $i = 1$ and $\alpha = x_1^{m+1} - 1 - x_k^m$. □

Theorem 9. The greedy procedure builds an optimal solution to problem (3.18).

Proof. We prove it by taking $m = \sum_{i \in [1..p]} x_i$ as induction parameter. We denote by f^m the optimal objective for parameter $m \geq p$.

- Basis step: clearly, for $m = p$, we have no other choice than assigning $x_i = 1$ for all $i \in [1..p]$ hence this solution has an objective equal to f^p.

- Induction step: assume that the current solution $\mathcal{X}^m = x_1, ..., x_p$ is optimal: $f(\mathcal{X}^m) = f^m$ (induction hypothesis). We prove that modifying the solution into $\tilde{\mathcal{X}}^m = x_1^m, ..., x_{k-1}^m, x_k^m + 1, x_{k+1}^m, ..., x_p^m$ with $k = \text{argmax}_i\{\frac{H_i}{x_i^m} - \frac{H_i}{x_i^m+1}\}$ leads to an optimal solution for $m+1$: $f(\tilde{\mathcal{X}}^m) = f^{m+1}$. We prove it by contradiction: we assume that the solution $\tilde{\mathcal{X}}^m$ is not optimal for $m+1$. It means that there exists another solution $\mathcal{X}^{m+1} = x_1^{m+1}, ..., x_p^{m+1}$ which is optimal: $f(\mathcal{X}^{m+1}) = f^{m+1} < f(\tilde{\mathcal{X}}^m)$. Without loss of generality we assume that $x_1^{m+1} > x_1^m, ..., x_l^{m+1} > x_l^m, x_{l+1}^{m+1} \leq x_{l+1}^m, ..., x_p^{m+1} \leq x_p^m$. Then we can transform this solution into $\tilde{\mathcal{X}}^{m+1} = x_1^{m+1} - 1, x_2^{m+1}, ..., x_p^{m+1}$. By corollary 2 we have that $f(\mathcal{X}^m) - f(\tilde{\mathcal{X}}^m) \geq f(\tilde{\mathcal{X}}^{m+1}) - f(\mathcal{X}^{m+1})$. If we combine it with the contradiction hypothesis $f(\tilde{\mathcal{X}}^m) > f(\mathcal{X}^{m+1})$ we obtain that $f(\tilde{\mathcal{X}}^m) > f(\tilde{\mathcal{X}}^{m+1})$ which contradicts our induction hypothesis that $\mathcal{X}^m$ is optimal.

□

Remarq. As Yves Crama pointed out, the problem 3.18 is a discrete resource allocation problem with separable objective convex function. The algorithm we rediscovered is named `INCREMENT` in p54 of [19] and its correctness is proved differently in [19] using *Generalized Lagrange multiplier method.*

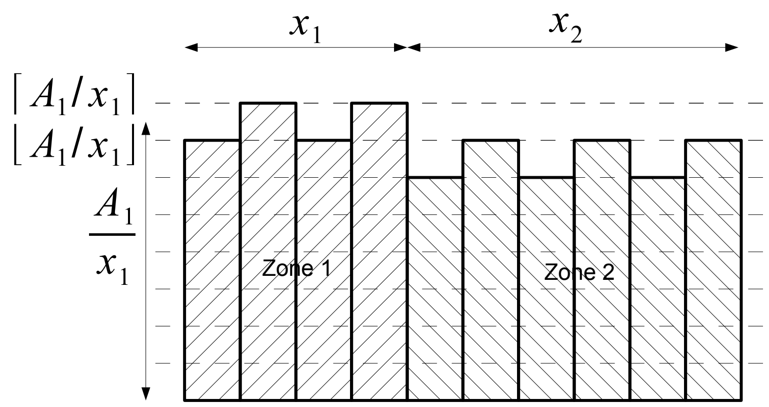


Figure 3.15: Illustration of the Lower Bound on L_2 using the Pre-Computation of the Number of Nurses in Each Zone.

The pre-computation of the number of nurses assigned to each zone can also be used to compute a lower bound on the L_2 criterion. Inside a zone, the average load is $\mu_k = A_k/x_k$. Since the acuity of patients are integers, we can strengthen the lower bound of the objective (3.15) by enforcing the workloads of nurses of zone k to be either $\lfloor \mu_k \rfloor$ or $\lceil \mu_k \rceil$. This is illustrated on Figure 3.15. The total workload of zone k must remain A_k. Hence the distribution of the workload among $\lfloor \mu_k \rfloor$ and $\lceil \mu_k \rceil$ are given respectively by $\alpha_k = A_k + x_k \cdot (1 - \lceil \mu_k \rceil)$ and $\beta_k = x_k - \alpha_k$. The lower bound on the spread variable $\underline{\Delta}^{\mathbb{Z}}$ computed with formula (3.1) is then

$$m \cdot \sum_{k=1}^{p} (\alpha_k \cdot \lceil \mu_k \rceil^2 + \beta_k \cdot \lfloor \mu_k \rfloor^2) - (\sum_{k=1}^{p} A_k)^2. \tag{3.21}$$

The two-step CP model in Comet is given in Listing 3.4. The model assumes that the x_k (named `nbPatientsInZone[k]` in the program) are already computed in the fragment. The main differences with respect to the previous Comet model in Listing 3.3 is the definition of the domains in line 11. The patients of one zone i can only take the nurses that have been affected to this zone defined by the range `nursesOfZone[i]`. The search in the `using` block lines 23–36 is a little bit more complicated: all the patients of one zone are assigned before the patients of the next zone are assigned. The dynamic symmetry breaking scheme is the same

Table 3.9: Patients to Nurses Assignment Problem with 2 zones with precomputation of the number of nurses in each zone

m	n	#fails	time(s)	avg workload	sd. workload	lb. sd.
11	28	25385	4.5	86.09	2.64	2.23
11	29	4916	1.4	80.27	1.76	0.62
10	26	458	0.1	76.50	2.29	2.29
12	30	17558	6.7	83.42	1.93	1.19
10	28	29865	4.8	91.80	6.84	6.81
10	26	3705	1.0	88.40	2.29	1.43
12	29	6115	1.2	80.08	2.72	0.64
10	27	1109	0.4	90.60	5.33	5.22
10	25	3299	0.6	82.70	7.32	6.71
8	22	127	0.0	87.50	3.12	3.04

but adapted to this by zone assignment.

Table 3.9 reports the results obtained on the same 2-zones instances as for Table 3.8 with the pre-computation of the number of nurses assigned to each zone. The last column is the lower bound obtained with equation (3.21). A first observation is that the computation times are greatly reduced. It does not exceed 10 seconds while it exceeded 1000 seconds for the most difficult instances. One can see that our method finds the correct number of nurses since the standard deviation with previous model are exactly the same (hence optimum) as the optimal values in Table 3.8. It is also interesting to see that the lower bound is reasonably close to the optimum values which also validates the approach.

Since the instances with 2 zones can now be solved easily, we tried to solve instances with 3 zones. The results are presented on Table 3.10. Only 6 instances could be solved optimally within 30 minutes when precomputing the number of nurses in each zone.

A Two-Step CP Model with Decomposition

The previous approach can solve easily two-zone problems but has difficulties to solve 3 zones problems and instances with more that 3 zones

Listing 3.4: Two steps Patient-Nurse Assignment Model

```
Solver<CP> cp();
var<CP>{int} nurseOfPatient[1..nbPatients];
var<CP>{int} acuityByNurse[1..nbNurses](cp,1..105);
var<CP>{int} spreadAcuity(cp,0..System.getMAXINT());
var<CP>{int}[] nurseOfPatientByZone[1..nbZones];
range nursesOfZone[zones];
int l=1;
forall(i in zones){
   nursesOfZone[i] = l..l+nbNurseInZone[i]−1;
   nurseOfPatientByZone[i] =
        new var<CP>{int}[1..nbPatientsInZone[i]](cp,nursesOfZone[i]);
   l += nbNurseInZone[i];
}
int k = 1;
forall(i in zones,j in 1..nbPatientsInZone[i])
        nurseOfPatient[k++] = nurseOfPatientByZone[i][j];
minimize<cp> spreadAcuity
subject to {
   cp.post(spread(acuityByNurse,totAcuity,spreadAcuity));
   cp.post(multiknapsack(nurseOfPatient,acuity,acuityByNurse));
   cp.post(cardinality(minNbPatients,nurseOfPatient,maxNbPatients));
}
using {
  forall(i in zones){
    forall(p in nurseOfPatientByZone[i].rng():
                   !nurseOfPatientByZone[i][p].bound())
        by(−acuityByZone[i][p],nurseOfPatientByZone[i][p].getSize()){
        int shift = i==1? 0 : nursesOfZone[i−1].getUp();
        int mn = max(0,maxBound(nurseOfPatientByZone[i]))+shift;
        tryall<cp>(n in nursesOfZone[i]: n <= mn + 1)
                              by (acuityByNurse[n].getMin())
           cp.label(nurseOfPatientByZone[i][p],n);
        onFailure
           cp.diff(nurseOfPatientByZone[i][p],n);
    }
  }
}
```

Table 3.10: Patients to Nurses Assignment Problem with 3 zones with precomputation of the number of nurses in each zone

sol	m	n	#fails	time(s)	avg workload	sd. workload	lb. sd.
1	15	42	19488	5.3	84.20	3.04	2.93
1	18	43	3619310	919.2	79.78	5.84	5.49
0	17	43	9023072	1800.0	81.41	4.75	3.45
1	17	42	483032	106.9	83.82	5.65	5.59
0	18	43	7124370	1800.0	81.00	7.11	4.94
1	14	38	590971	145.2	85.36	3.08	2.16
0	19	48	3786580	1800.0	87.42	3.18	2.30
1	16	44	3888210	839.8	84.88	6.70	6.39
0	19	49	5697272	1800.0	86.00	2.70	1.95
1	17	41	61250	17.3	82.18	3.40	3.07

are intractable. It thus seems natural to decompose the problem by zone and to balance the workload of nurses inside each zone independently rather balancing the workload of all the nurses globally. Interestingly, this decomposition preserves optimality, i.e., it reaches the same solution for the L_2 criterion than the two-step approach of Section 3.6.2 for a given pre-computation of the number of nurses assigned in each zone. In other words, given the pre-computed number of nurses in each zone, it is equivalent to minimize L_2 among all the nurses at once or to minimize L_2 separately inside each zone. We now prove this result formally.

Lemma 6. Minimizing $n \cdot \sum_{i=1}^{x_k}(y_i - A_k/x_k)^2$ such that $\sum_{i=1}^{x_k} y_i = A_k$ is equivalent to minimizing $n \cdot \sum_{i=1}^{x_k}(y_i - (A_k/x_k + c))^2$ such that $\sum_{i=1}^{x_k} y_i = A_k$.

Proof. The first objective can be reformulated from formula (3.1) as $x_k \cdot \sum_{i=1}^{x_k} y_i^2 - A_k^2$. The second one can be reformulated after some algebraic manipulations as $c^2 \cdot x_k^2 + x_k \cdot \sum_{i=1}^{x_k} y_i^2 - A_k^2$. Since they differ only by a constant term, they are equivalent. □

Corollary 3. It is equivalent to minimize L_2 among all the nurses at once or to minimize L_2 separately inside each zone.

Table 3.11: Patients to Nurses Assignment Problem with 3 zones with precomputation of the number of nurses in each zone and decomposition by zone

m	n	#fails	time(s)	avg workload	sd. workload	lb. sd.
15	42	203	0.1	84.20	3.04	2.93
18	43	608	0.1	79.78	5.84	5.49
17	43	8134	1.1	81.41	4.46	3.45
17	42	345	0.1	83.82	5.65	5.59
18	43	24994	3.2	81.00	5.77	4.94
14	38	151	0.0	85.36	3.08	2.16
19	48	3695	0.8	87.42	3.07	2.30
16	44	384	0.1	84.88	6.70	6.39
19	49	2056	0.4	86.00	2.49	1.95
17	41	776	0.2	82.18	3.40	3.07

Proof. This follows directly from Lemma 6. If the minimization of L_2 is achieved globally for all the nurses, the least square L_2 criterion is computed with respect to the global average load of all the nurses that is wrt $\sum_{k=1}^{p} A_k/m$. This corresponds to choosing c in Lemma 6 equal to the difference between the average load in zone k and the global average load: $c = \sum_{k=1}^{p} A_k/m - A_k/x_k$. □

We solved again the 3-zone instances with the decomposition method. The results are given on Table 3.11. One can observe that, as expected, the objectives are the same for the instances that could be solved optimally in Table 3.10. For the remaining ones, the algorithm produces strictly better solutions. The time is also significantly smaller. Figure 3.16 shows a Comet visualization of a solution for a 15 zones instance with 81 nurses and 209 patients. This instance could be solved in only 7 seconds and 10938 fails.

3.6.3 Softening an ideal given distribution

Many combinatorial problems are over-constrained. For solving these with constraint programming some constraints must be relaxed. In particular we explain here how to relax the fact that an array of variables

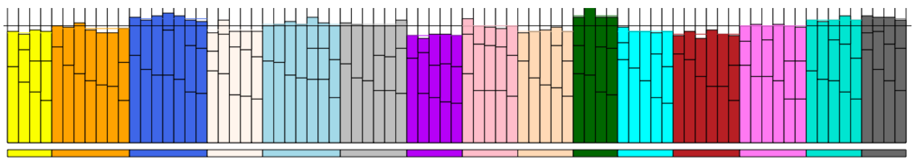


Figure 3.16: Solution for a 15 zones instance.

must follow a given ideal distribution of values. This constraint occurs in rostering problems where each day a set of workers must cover some needs and the number of workers that must be assigned to each type of needs is known in advance.

Example 17. Assume 10 workers that might achieve 5 different activities. The possible activities of each workers according to their skills or availability are respectively

$$\{1,3\},\{1\},\{2,4\},\{1,2\},\{1,2,5\},\{5,3\},\{1,2\},\{3,1\},\{1,4\},\{2\}.$$

The number of workers required for each activity are respectively $2,0,3,1,4$. This small problem is over-constrained since 4 workers are required for activity 5 but only 2 can achieve this activity.

The constraint is defined precisely in next definition:

Definition 11. We call

$$\texttt{soft-distribute}_\mu([X_1,\dots,X_n],[\delta_1,\dots,\delta_m],V)$$

the constraint

$$\sum_{j=1}^{m}\mu(|\#\{i|X_i=j\}-\delta_j|)\le V$$

where the value parameters satisfy $\sum_{j=1}^{m}\delta_j=n$ (valid ideal distribution) and μ is a function measuring the violation.

A possible model for this problem is to use the combination of a global cardinality constraint `gcc` together with a balancing constraint like `spread` or `deviation`:

- The `gcc` constraint computes the cardinalities $D_j=\#\{i|X_i=j\}$,
- We introduce a variable $Y_j=D_j-d_j$. Since we know that $\sum_j Y_j=0$, `spread` or `deviation` with a sum value $s=0$ can be used to balance the violations among the values.

Listing 3.5: Modeling of `soft-distribute` constraint with `spread`

```
Solver<CP> cp();
int d [1..5] = [2,0,3,1,4];
var<CP>{int} Y[1..5](cp,−10..10);
var<CP>{int} D[1..5](cp,0..10);
set{int} domains[1..10] =
  [{1,3},{1},{2,4},{1,2},{1,2,5},{5,3},{1,2},{3,1},{1,4},{2}];
var<CP>{int} X[i in 1..10](cp,domains[i]);
var<CP>{int} viol(cp,0..System.getMAXINT());

minimize<cp>
  viol
subject to{
    cp.post(cardinality(D,X),onDomains);
    forall(i in 1..5){
      cp.post(Y[i]==D[i]−d[i]);
    }
    cp.post(spread(Y,0,viol));
}
using{
  labelFF(X);
}
```

A model with `spread` uses a quadratic violation function $\mu(x) = x^2$ while a model with `deviation` uses the linear violation function $\mu(x) = x$. A complete model for example 17 modeling `soft-distribute` with `spread` is given in Listing 3.5.

Note that using `deviation` to model `soft-distribute` is not really interesting since the violation measured would be the same as with `soft-gcc` computing the sum of over-flow under-flow for each value and efficient domain consistent filtering algorithms exist for this constraint[6] [58, 57].

[6]To the best of our knowledge, this constraint is currently only implemented in COMET using a filtering algorithm described in [46]

4 Bin-Packing Constraints

Many real problems have a bin-packing component (line balancing, rostering...). To tackle such problems with constraint programming, it is important to capture the structure of the bin-packing sub-problem as much as possible to prune efficiently the search space. Paul Shaw introduced a global constraint for bin-packing very efficient both in terms of speed and quality of filtering [52]. It allows to solve problems with a bin-packing component that are intractable without using it.

In some problems, in addition to the bin-packing component, we have precedence constraints between items. The bin-packing with precedences sub-problem is ubiquitous in line balancing problems. We give filtering algorithms for this constraint and show the effectiveness of the approach on standard line balancing benchmarks.

4.1 Bin-Packing Constraint

4.1.1 Related Work

Several work has been achieved in CP on the filtering of the binary knapsack constraints which are closely related to the bin-packing constraint:

- Trick introduced in [55] a pseudo-polynomial time dynamic programming algorithm to achieve domain consistency on a binary knapsack constraint.
- When the capacity is large, Sellmann proposes in [51] to palliate the pseudo-polynomial time by weakening the propagation

strength by dividing down item sizes and bin capacities.

For the bin-packing constraint, to the best of our knowledge, there only exists the filtering introduced by Shaw [52]. Achieving domain consistency for a bin-packing constraint is an NP-complete problem. This is why propagators must make some relaxations on the problem in order to be efficient. The filtering algorithm proposed by Paul Shaw essentially works separately on each bin with a knapsack reasoning and detects non packable bins or non packable items into bins. The relaxation is that an item can be used in more than one bin. Shaw also introduces a failure detection test based on fast bin-packing lower bound algorithm. In the next Section 4.1.2, we recall the filtering introduced by Shaw.

We refer to [7] for an extended bibliography review of the literature on the bin-packing problem and its variants.

4.1.2 Paul Shaw's Global Constraint

The `Pack`$([B_1, ..., B_n], [s_1, ..., s_n], [L_1, ..., L_m])$ constraint is composed of

- n variables $[B_1, ..., B_n]$ representing for each item the bin where it is placed ($Dom(B_i) = \{1, \ldots, m\}$)
- n positive values $[s_1, ..., s_n]$ representing the size of each item.
- a number of available bins m.
- m variables $[L_1, ..., L_m]$ representing the load of each bin ($Dom(L_i) = \{0, \ldots, \sum_i s_i\}$)

The semantic of the constraint is $L_i = \sum_{j=1}^{n} ((B_j = i) \cdot s_j)\ \forall i \in [1..m]$. This constraint can be easily expressed with binary variables X_{ij} and reified constraints: $X_{ij} = 1 \leftrightarrow B_j = i\ \forall i, j \in [1..m] \times [1..n]$, $L_i = \sum_j X_{ij} \cdot s_j$. This simple model is improved with the redundant constraint $\sum_{j=1}^{n} s_j = \sum_{i=1}^{m} L_i$ allowing a better communication between the loads and hence a stronger filtering [52]. We recall some useful definitions from [52].

- The possible set P_i for a bin i is the set of items that can be packed into bin i that is $P_i = \{j \in [1..n] \mid i \in Dom(B_j)\}$.
- The required set R_i of a bin i is the set of items packed into bin i that is $R_i = \{j \in [1..n] \mid Dom(B_j) = \{i\}\}$.

- The candidate set is $C_i = P_i \setminus R_i$.
- The set of unpacked items is $U = \bigcup_{i=1}^m C_i$.
- For a set of items indices S, $\mathrm{sum}(S) = \sum_{i \in S} s_i$.

In addition to the bin-packing model described above, the filtering of Shaw is composed of a binary-knapsack reasoning on each bin separately and of an inconsistency test based on the computation of a bin-packing lower bound.

Knapsack Filtering

Achieving domain consistency for `Pack` is NP-Complete. Shaw consider the relaxation of dividing the problem into packing each bin i separately with its candidate set inside its allowed load range $[L_i^{\min}, L_i^{\max}]$. This is a relaxation since a same item might be used to pack two different bins.

The set of sets of additional items which can be packed while respecting the constraints on load is

$$M_i = \{A \mid A \subseteq C_i \wedge L_i^{\min} \leq \mathrm{sum}(R_i \cup A) \leq L_i^{\max}\}.$$

Three filtering rules can occur when examining M_i

- If M_i is empty then the bin i is not *packable* and the constraint fails.
- If an item is present in every sets of M_i then this item is assigned to the bin i.
- If an item is present in none of the sets composing M_i then bin i can be removed from the domain of this item.

Is also possible to filter the load variable L_i by deducing illegal bin loads. This filtering can be achieved with with a pseudo polynomial dynamic programming algorithm in $O(|C_j|(L_j^{\max})^2)$ [55]. Since this complexity becomes prohibitive when the load capacity are large, Shaw relaxes the problem with a very efficient algorithm capable to detect non-packable bins. This algorithm is a relaxation since all non packable bins may not be detected. Shaw introduces the concept of *neighboring subsets* of C_i.

Definition 12 ([52])**.** Two subsets C_i^1, C_i^2 are said to be neighboring if there is no other subset of C_i whose items sum a value strictly between $\mathrm{sum}(C_i^1)$ and $\mathrm{sum}(C_i^2)$.

Hence if two neighboring subsets C_i^1, C_i^2 are detected with $\mathrm{sum}(C_i^1) < L_i^{\min} \leq L_i^{\max} < \mathrm{sum}(C_i^2)$ then the constraint fails because by definition of neighboring subsets, the bin i is not packable. Shaw gives a procedure `NoSum`(X, α, β) which returns *true* when it detects two neighboring subsets X^1, X^2 on X with $\mathrm{sum}(X^1) \leq \alpha \leq \beta \leq \mathrm{sum}(X^2)$. This procedure executes in $O(|X|)$. Unfortunately `NoSum` will sometimes return *false* when actually there exists two neighboring subsets satisfying the condition. This is because all neighboring subsets are not generated by the procedure.

The three pruning rules described above can be reformulated with `NoSum` as:

- If `NoSum`$(C_i, L_i^{\min} - \mathrm{sum}(R_i), L_i^{\max} - \mathrm{sum}(R_i))$ then fail.
- If for a candidate item $j \in C_i$, `NoSum`$(C_i \setminus j, L_i^{\min} - \mathrm{sum}(R_i), L_i^{\max} - \mathrm{sum}(R_i))$ then $B_j \leftarrow i$.
- If for a candidate item $j \in C_i$, `NoSum`$(C_i \setminus j, L_i^{\min} - \mathrm{sum}(R_i) - s_j, L_i^{\max} - \mathrm{sum}(R_i) - s_j)$ then $Dom(B_j) \leftarrow Dom(B_j) \setminus i$.

When `NoSum` answers in the affirmative is also returns the total size of the neighboring subsets α' and β': $\alpha' = \mathrm{sum}(X^1) \leq \alpha \leq \beta \leq \mathrm{sum}(X^2) = \beta'$. This values can also be used to tighten the bounds of L_i:

- If `NoSum`$(C_i, L_i^{\min} - \mathrm{sum}(R_i), L_i^{\min} - \mathrm{sum}(R_i))$ then
$$L_i^{\min} \leftarrow \max\{L_i^{\min}, \mathrm{sum}(R_i) + \beta'\}.$$
- If `NoSum`$(C_i, L_i^{\max} - \mathrm{sum}(R_i), L_i^{\max} - \mathrm{sum}(R_i))$ then
$$L_i^{\max} \leftarrow \min\{L_i^{\max}, \mathrm{sum}(R_i) + \alpha'\}.$$

The complexity of filtering one bin can be up to $O(|C_i|^2)$ and $O(m \cdot |C_i|^2)$ for all the bins but it is much more efficient in practice [52].

In Ilog Solver, the `Pack` constraint is called `IloPack` and in Comet it is called `multiknapsack`. Unfortunately, Paul Shaw does not explain for which load values it becomes more advantageous to use the filtering based on `NoSum` rather than the stronger filtering based on dynamic programming. Hence is is very difficult to predict the filtering that will be achieved with `IloPack`.

Inconsistency detection with Bin-Packing Lower Bound

The inconsistency detection of the `Pack` constraint can be reduced to classical bin-packing[1] problem with a fixed capacity. Hence bin-packing lower bounds can be very useful to detect quickly inconsistencies by comparing the lower bound to the number of available bins m. If the lower bound is larger than m in the reduced problem, then the constraint fails. The reduction to a bin packing problem proposed by Shaw is the following:

- Take the maximum load as bin capacity $c = \max_{i \in [1..m]} L_i^{\max}$.
- The set of items to be packed is $U \cup \{a_1, \dots, a_m\}$ where
  - U is the set of unassigned items.
  - a_i is a pseudo item created to take into account the already packed items into bin i and the fact that the maximum load of $L_i^{\max}$ might be smaller than c. The size of item a_i is $\mathrm{sum}(R_i) + (c - L_j^{\max})$.

Shaw then uses the Martello and Toth lower bound denoted $\mathcal{L}_2$ [28] to compute a lower bound on the number of bins required. `Pack` fails if this number is larger than m. The bound $\mathcal{L}_2$ can be computed in linear time when the items a sorted decreasingly. Items in U can be assumed to be sorted decreasingly without loss of generality but the items $\{a_1, \dots, a_m\}$ need to be sorted and then merged with items from U. Hence the complexity of the inconsistency detection is $O(m \cdot log(m) + n)$.

4.1.3 Additional Filtering for the Bin Packing Global Constraint

We try to improve the filtering of the bin-packing constraint in the following way:

- We show that it is in some cases possible to filter additionally the domains or detect earlier failure situations with another relaxation which is the preemption of the items. The difference with Shaw relaxation is that items can be split into several bins but all the bins are considered at once and not separately. This filtering is not

[1]given a bin capacity and sized items, what is the minimum number of bins necessary to pack all the items.

better but complementary and is shown to be useful on practical problems in our experiments.

- We improve the failure detection test based on fast bin-packing lower bound algorithm by proposing a different reduction and using a stronger bin-packing lower bound that the one used in [52].
- We also introduce a global cardinality redundant constraint for the bin-packing constraint with propagator filtering dynamically the cardinality variables.

Inconsistency detection based on the Preemption Relaxation

Previous section recalls the filtering introduced by Shaw for the `Pack` constraint [52]. The relaxation of Shaw is to work on each bin separately with knapsack reasoning to filter. We suggest another relaxation complementary which is the preemption of items to detect inconsistencies not detected by Shaw's propagators.

The preemption relaxation is that items can be split among several bins.

Given a partial[2] solution for `Pack`$([B_1, ..., B_n], [s_1, ..., s_n], [L_1, ..., L_m])$ the consistency test succeeds if it is possible to find a preemptive schedule of the unassigned items such that:

- each (part of) item $j \in U$ is scheduled in a bin allowed by its domain ($\in Dom(B_j)$) and
- the bin capacities are not exceeded.

Example 18. Consider the following bins and items:

- 5 bins with capacity 5: $\forall i \in [1..5],\ Dom(L_i) = [1..5]$.
- 11 items with sizes $\forall i \in [1..10], s_i = 1$ and $s_{11} = 2$.
- $Dom(B_1) = [1..5]$ and $\forall i \in [2..11], Dom(B_i) = \{4, 5\}$.

We immediately see that the total size of the items placed in the last two bins is 11 (9 items of size 1 and one item of size 2). But the total space available in the last two bins is only 10 (two times the capacity 5). Hence it is impossible to place preemptively the last 10th items into the last two bins. The constraint `Pack` should fail in such a case but unfortunately it does not detect the failure.

[2]some items may already be assigned

The preemption based inconsistency test can be achieved in polynomial time by solving a maximum flow problem in a bipartite graph similarly to the domain consistency filtering of the Global Cardinality Constraint [34]. For the sake of completeness we recall some important preliminaries about network flows inspired from [6] and [34].

A flow network $G = (V, A)$ is a directed graph in which each arc $(u, v) \in A$ has non negative integer capacity $c(u, v) \geq 0$. If $(u, v) \notin A$, we assume that $c(u, v) = 0$. Two vertices has a special status in a network flow: the *source* s and the *sink* t. A *path* in a graph from v_1 to v_k is a sequence of vertices $[v_1, v_2, \ldots, v_k]$ such that (v_i, v_{i+1}) is an arc for $i \in [1, \ldots, k-1]$. We assume the network with no symmetric pairs of arcs and connected such that for every vertex $v \in V$ there is a path $s \rightsquigarrow v \rightsquigarrow t$. A *flow* in G is real-valued function $f : V \times V \rightarrow \Re$ that satisfies the following three properties:

- Capacity constraint: $\forall u, v \in V : f(u, v) \leq c(u, v)$.
- Skew symmetry: $\forall u, v \in V : f(u, v) = -f(v, u)$.
- Flow conservation: $\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(u, v) = 0$

The quantity $f(u, v)$, which can be positive, zero, or negative is called the *flow* from vertex u to vertex v. The *value* of a flow f is defined as $|f| = \sum_{v \in V} f(s, v)$. We are interested in the *maximum-flow problem* that is finding a flow of maximum value in a flow network. An important property is that there exists an maximum flow from s to t which is integral on every arc in G.

The *variable-value graph* [26] of the variables $[X_1, \ldots, X_n]$ is a directed bipartite graph from the variables to the values such that there is an arc from a variable vertex X to a value vertex v if $v \in Dom(X)$.

For a $\texttt{Pack}([B_1, ..., B_n], [s_1, ..., s_n], [L_1, ..., L_m])$ constraint we build a flow network that we denote $N(\text{Pack})$. The consistency of `Pack` with the preemption relaxation is tested by solving a maximum flow problem in $N(\text{Pack})$.

Definition 13 (N(Pack))**.** The flow network $N(\text{Pack})$ is composed of

- The *variable-value* graph of the variables $[B_1, \ldots, B_n]$ with an infinite capacity for these arcs.
- A source vertex s is linked to every item vertex j with a capacity equal to the size of the item s_j.

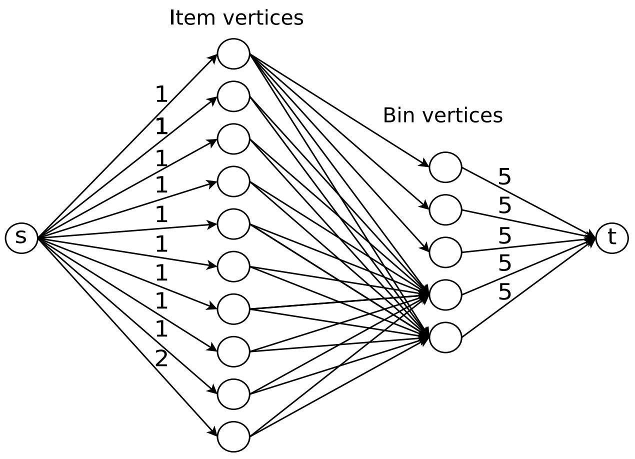


Figure 4.1: Flow network $N(\text{Pack})$ of Example 18.

- Every bin vertex i is linked to a sink vertex t with a capacity equal to the capacity of the bin $L_i^{\max}$.

Example 19. Figure 4.1 gives the flow network $N(\text{Pack})$ of Example 18

Note that to every maximum flow in $N(\text{Pack})$ with value $(\sum_{i\in[1..n]} s_i)$ corresponds a preemptive schedule of the items satisfying the bin capacity and such that (part of) items are only placed into bins allowed by their domains. The amount of an item placed into a bin is given by the flow value of the arc from the item vertex to the bin vertex. The contrary is also true, given a preemptive schedule of the items satisfying the bin capacity and such that (part of) items are only placed into bins allowed by their domains, it is immediate to build maximum flow in $N(\text{Pack})$. Because the preemption hypothesis is a relaxation of the real problem, the following lemma gives an easy way to detect inconsistent partial solutions to the `Pack` constraint.

Lemma 7. If the maximum flow in $N(\text{Pack})$ from the source s to the sink t is less than $(\sum_{i\in[1..n]} s_i)$ then `Pack` is inconsistent.

In a flow network $G = (V, A)$ the Edmonds-Karp algorithm finds a maximum flow in $O(|V| \cdot |A|^2)$. For the network of interest $N(\text{Pack})$, $|V| = n + m + 2 \sim O(n)$ since we suppose there are less bins than items. The number of arcs is dominated by the number of arcs in the variable-value graph that is $O(n \cdot m)$ if every item can be placed in every bin. Hence the complexity of finding a maximum flow in $N(\text{Pack})$

is $O(n \cdot (n \cdot m)^2) = O(n^3 m^2)$. In practice, this complexity is much more efficient since it is maintained incrementally during the search as explained in the implementation section.

Implementation details

To implement the network flow $N(\text{Pack})$, we use an implicit representation of the graph. The flow is stored in an $m \times n$ integer matrix such that the flow of an arc can be updated in constant time. Additionally, an array of size n stores the flow values of the arcs from the source to the item vertices and one array of size m stores the flow from the bin vertices to the sink. The maximum flow is maintained incrementally:

- When a bin is removed from the domain of an item and that the corresponding arc has a positive flow value in the current network flow, the flow is not maximum anymore. Indeed the flow is updated to reflect the fact that this arc does not exists anymore. Then a maximum flow is retrieved incrementally with the augmenting path algorithm. A failure occurs if it is not possible to place completely every items.

- Whenever the upper bound $L_j^{\max}$ of a load variable is decreased. If the flow in the arc from L_j to the sink t is larger than $L_j^{\max}$, the flow is decreased to satisfy this new capacity that is $L_j^{\max}$. Then a new maximum flow is computed from the current one. A failure occurs if it is not possible to place completely every items.

Note that the current flow is valid on backtracking since arcs are only added. This prevent us to deal specifically the backtracking and to store and restore state of the network flow.

Experiments

Table 4.1 illustrates the results on Assembly Line Balancing instances with the failure detection based on the preemption relaxation. We can see that this filtering is interesting only for one instance: GUNTHER 8. For the other instances it introduces an overhead in time that is not compensated by the stronger pruning. We think that this idea is not promising for the bin-packing problems because the relaxation is very weak or it should be used in a quick shaving procedure.

inst. caract.		**pack**				**pack+flow**			
name	stations	solved	fails	time	max	solved	fails	time	max
BUXEY	6	1	9	33	55	1	9	183	55
BUXEY	8	1	0	12	41	1	0	61	41
BUXEY	10	1	11	69	34	1	11	347	34
GUNTHER	6	1	6	25	84	1	6	163	84
GUNTHER	8	1	146873	32672	63	1	**15977**	47366	63
GUNTHER	10	1	49430	16503	50	1	38567	136826	50
HAHN	6	1	133	259	240	1	64	640	240
HAHN	8	1	281	675	190	1	103	1766	190
HAHN	10	1	0	33	177	1	0	258	177
LUTZ1	6	1	78	109	2396	1	41	259	2396
LUTZ1	8	1	2016	778	1860	1	1264	2747	1860
LUTZ1	10	1	7713	3965	1526	1	**1558**	6005	1526
LUTZ2	6	1	7821	3088	81	1	4714	16963	81
LUTZ2	8	1	2	78	61	1	2	599	61
LUTZ2	10	0	256803	200002	50	0	33602	200034	50
LUTZ3	6	1	1502	1393	275	1	1490	12652	275
LUTZ3	8	1	29756	22153	207	0	22725	200013	208
LUTZ3	10	1	250192	137974	165	0	24969	200005	169
MITCHELL	6	1	0	6	18	1	0	37	18
MITCHELL	8	1	0	7	14	1	0	50	14
MITCHELL	10	1	0	8	13	1	0	81	13
ROSZIEG	6	1	1	7	21	1	1	41	21
ROSZIEG	8	1	19	23	16	1	19	142	16
ROSZIEG	10	1	3	15	14	1	3	150	14
SAWYER	6	1	6	27	55	1	6	131	55
SAWYER	8	1	9	19	41	1	9	93	41
SAWYER	10	1	13490	6865	34	1	7651	42422	34
TONGE	6	0	559218	200001	586	0	47858	200007	586
TONGE	8	0	418464	200000	442	0	89374	200001	442
TONGE	10	0	392998	200000	352	0	22911	200010	352
WARNECKE	6	1	2129	1940	258	1	741	3800	258
WARNECKE	8	1	64261	31150	194	0	50648	200013	195
WARNECKE	10	0	197549	200001	157	0	30678	200008	157
WEE-MAG	6	1	1	41	250	1	1	314	250
WEE-MAG	8	1	9	59	188	1	9	505	188
WEE-MAG	10	0	531952	200001	152	0	36520	200000	152

Table 4.1: Results of the failure detections with flow on Assembly Line Balancing Instances

Filtering based on the Preemption Relaxation

If a (part) of an item j does not occur in bin i in any preemptive solution of the constraint, bin i can be removed from the domain of variable B_j. The network flow $N(\text{Pack})$ can be used to detect zero flow arcs by computing the strongly connected components (scc's) on the residual graph similarly to the Global Cardinality Constraint [34]. The idea is to detect in $N(\text{Pack})$ the arcs from items to bins that don't participate in any maximum flow. These arcs are identified by computing the scc's on the residual graph of a maximum flow in $N(\text{Pack})$. If an arc $(Item, Bin)$ has a flow value of zero in the maximum flow and if its extremities belong to two different scc's of the residual graph, it can be safely removed (the Bin is removed from the domain of $Item$) since it is a zero flow arc. The complexity of this filtering is dominated by the computation of the scc's that is $O(n \cdot m)$ (linear in the number of arcs and vertices). Although the theoretical complexity is smaller than the one necessary to maintain the maximum flow, in practice this filtering is not incremental and slows down significantly the propagator. We made some preliminary experiments showing that the cost of this filtering is never compensated by the reduced search space.

Inconsistency detection based on Fast Bin Packing Lower Bounds

We have seen in Section 4.1.2 the inconsistency detection proposed by Shaw based on a reduction to a standard bin-packing problem. We suggest a different reduction which is in some case stronger in the sense that for a same lower bound algorithm, our reduction allows to detect inconsistencies that are not detected with Shaw's reduction.

Improvement of the inconsistency detection

- Take the maximum free space as bin capacity

$$c = \max_{i \in [1..m]} (L_i^{\max} - \text{sum}(R_i)).$$

- The set of items to be packed is $U \cup \{a_1, \ldots, a_m\}$ where
  - U is the set of not yet assigned items.
  - a_i is a pseudo item created to take into account the fact that the free space of bin i might be smaller than c. The size of item a_i is $(c - (L_i^{\max} - \text{sum}(R_i)))$.

The following example shows that the reduction proposed above finds an inconsistency that is not detected with Shaw's reduction.

Example 20. Consider a `Pack` constraint with two bins. The domains of the load variables are $Dom(L_1) = Dom(L_2) = [4..8]$. One item of size 4 is packed in each bin: $Dom(B_1) = \{1\}, Dom(B_2) = \{2\}, s_1 = 4, s_2 = 4$. Three other items are unpacked $Dom(B_3) = Dom(B_4) = Dom(B_5) = \{1, 2\}$ with respective sizes $s_3 = 3, s_4 = 3, s_5 = 2$. This partial solution is represented on the left of Figure 4.2. The constraint is inconsistent since the two unpacked items of size 3 must be in two different bins and the remaining space in each bin is consequently not sufficient to accept the last item of size 2. The consistency test with the reduction of Paul Shaw will not detect this inconsistency while our reduction will detect it:

- Shaw reduction: The capacity of the bin is 8 (maximum upper bounds of load variable). Two pseudo items are created with size 4. Hence the sizes of the items to pack are $4, 4, 3, 3, 2$. As shown on the middle of Figure 4.2 this can be done with two bins hence the constraint is considered as consistent with this test.

- Our reduction: The capacity of the bin is 4 (maximum free space available). No pseudo items are created since the free space is the same in both bins. The sizes of the items to pack are $3, 3, 2$. As shown on the right of Figure 4.2 this can be done with at least three bins hence the inconsistency is detected since only two bins are available.

We don't know if the contrary is also true that is if Shaw's reduction allows to detect inconsistencies that we cannot detect (we couldn't find a counter example nor prove that it is impossible). Nevertheless, even if this is the case, a lower bound algorithm is used after that on these reduced problem to detect the inconsistency. Since lower bound algorithms that we use don't have a property of monotonicity and that both reductions are linear, we think that they should both be used inside a propagator to benefit from the advantages of the two.

The inconsistency test proposed by Paul Shaw can also be improved by using a different lower bound for the reduced bin-packing problem. Paul Shaw propose to use the bound $\mathcal{L}_2$ of Martello and Toth [28]. Recently the lower bound $\mathcal{L}_3$ of Labbé [25] has been proved to be always

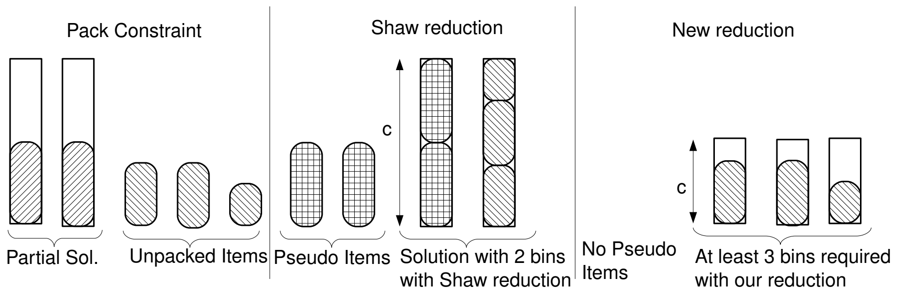


Figure 4.2: Illustration of the inconsistency test based on bin-packing reductions

larger or equal and to benefit from a better worst case performance ratio for a same linear computation time.

Let us denote by $N = \{1, \ldots, n'\}$ the set of items to be packed in the reduced bin packing problem (unpacked items and pseudo items) and their weights $w_1, \ldots, w_{n'}$. We now formulate the lower bounds $\mathcal{L}_2$ and $\mathcal{L}_3$ following the notations from [3] with $W(a,b) = \{i \in N | a < w_i \leq b\}$ and $\overline{W}(a,b) = \{i \in N | a \leq w_i \leq b\}$:

$$\mathcal{L}_2 = \max_{0 \leq \nu \leq c/2} \{\mathcal{L}_2(\nu)\}$$
$$\mathcal{L}_2(\nu) = |W(c/2,c)| + l(\nu)$$
$$l(\nu) = \max\left(0, \left\lceil \frac{\sum_{i \in \overline{W}(\nu, c-\nu)} w_i - |W(c/2, c-\nu)| \cdot c}{c} \right\rceil\right)$$
$$\mathcal{L}_3 = \max_{0 \leq v \leq c/3} \{\mathcal{L}_3(\nu)\}$$
$$\mathcal{L}_3(\nu) = |W(c/2,c)| + \lceil H/2 \rceil + p(\nu)$$
$$p(\nu) = \max\left(0, \left\lceil \frac{\sum_{i \in \overline{W}(\nu, c-\nu)} w_i - (|W(c/2, c-\nu)| + \lceil H/2 \rceil) \cdot c}{c} \right\rceil\right)$$

Where H denote the set of items having their weight in $]c/3, c/2]$ that cannot be matched with items having their weight in $]c/2, 2c/3]$.

We give some intuitions on each terms of these bounds:

- $|W(c/2,c)|$: For both bounds each items from $W(c/2,c)$ must be placed in a different bin since two items from this set cannot fit together in a bin. This is why there is the term $|W(c/2,c)|$ in both bound expressions.

- $l(\nu)$: All the items with size larger or equal to ν cannot be placed with any items from $W(c-\nu,c)$ (items smaller that ν are not considered). With $|W(c/2,c)|$, we already packed each items of $W(c/2,c-\nu)$ in separate bins. The idea is to fill preemptively and completely these bins with the items of $\overline{W}(\nu,c-\nu)$. The eventual left over that is $(\sum_{i\in\overline{W}(\nu,c-\nu)} w_i - |W(c/2,c-\nu)|\cdot c)$ is used to fill (preemptively) new bin(s). Since different values for ν can lead to different $l(\nu)$, the maximum over ν is taken as the contributing term in the bound.

- $\lceil H/2\rceil$: For the bound $\mathcal{L}_3$, we now explain the addition of the term $\lceil H/2\rceil$. A bin cannot contain more than two items from the set $W(c/3,c/2)$ and of course no more than two items from the set $W(c/3,c)$. We try to pair as much items as possible from the set $W(c/3,c/2)$ with the items from $W(c/2,c)$ already placed in separate bins. The items from $W(c/3,c/2)$ that could not be matched are paired together in different bins. This is the origin of the term $\lceil H/2\rceil$ in the bound $\mathcal{L}_3$.

- $p(\nu)$: the explanation of this term is similar to the one of $l(\nu)$.

These bounds can be computed in linear time when the size of the items are sorted. Here follow some details about the computation of these bounds:

- These bounds seem to require a pseudo-polynomial computation time because of the maximization over ν in the last term. Actually, only the values for ν corresponding to a size of the item set need to be tested.

- The set H can be computed efficiently with an adapted first fit procedure: by assigning the smallest remaining size to the remaining bin having the smallest residual capacity. H is simply the set of items from $W(c/3,c/2)$ that could not be assigned through this process to one of the $|W(c/2,c)|$ bins (each of these bins has already one item from $W(c/2,c)$).

- An algorithm to compute $\mathcal{L}_2$ is described in [28]. We implemented a linear linear time algorithm to compute $\mathcal{L}_3$.

It has been shown in [3] that $\mathcal{L}_3 \geq \mathcal{L}_2$. Next example illustrates on a small instance that $\mathcal{L}_3$ might be larger than $\mathcal{L}_2$.

Example 21. Let $c = 10$ and the weight vector be $[4, 4, 4, 4, 4]$. For both bounds $W(c/2, c) = \phi$. For the $\mathcal{L}_2$ bound : $\mathcal{L}_2(0) = \mathcal{L}_2(4) = \lceil (20 - 0)/10 \rceil = 2$. Hence $\mathcal{L}_2 = 0 + 2 = 2$. For bound $\mathcal{L}_3$, $H = \{4, 4, 4, 4, 4\}$. Hence the only contributing term in $\mathcal{L}_3$ is $\lceil H/2 \rceil = 3$.

It as recently been shown in [3] that the asymptotic performance of $\mathcal{L}_3$ are better than for $\mathcal{L}_2$. We first recall the definition of the worst-case asymptotic performance ratio $R^\infty(\mathcal{L})$ for a bound $\mathcal{L}$ then we give the performance of both bounds.

Definition 14. The worst-case asymptotic performance ratio $R^\infty(\mathcal{L})$ for a lower bound $\mathcal{L}$ is defined as

$$R^\infty(\mathcal{L}) = \lim_{s \to \infty} \sup \left\{ \frac{\mathcal{L}(I)}{OPT(I)} \mid \forall I : OPT(I) \geq s \right\}$$

where I is a bin-packing instance and $OPT(I)$ is the optimal number of bins for the instance.

Theorem 10. $R^\infty(\mathcal{L}_2) = 2/3$ [28] and $R^\infty(\mathcal{L}_3) = 3/4$ [3].

Experiments on bin-packing problems

We show how to model a bin-packing problem in Comet using the search procedure described in [52]: *We use a standard search procedure, complete decreasing best fit (CDBF), which packs items in order of non-increasing size, packing each item in the first bin with least free space that will accommodate it. On backtracking, the search states that the chosen item cannot be placed in the selected bin. A symmetry breaking rule is also used which states that on backtracking, all "equivalent" bins are also eliminated as candidates for the current item and indeed all other unpacked items of the same size. An equivalent bin is one which carries the same load as the one being eliminated from consideration. In addition, no choice point is created if all available bins for a particular item are equivalent: the item is packed into the first of them. Finally, we added the additional dominance rule which states that if an item can fill a partially filled bin to capacity, then it is immediately packed in this bin* [52].

We improve this search procedure with the addition of another important dominance rule from [23] which state that: *if for a bin, at most* *one item can be added to it without violating its capacity, then add immediately the largest item fitting into it.* The optimal solution is found by solving a succession of satisfaction problems. Initially, the number of available bins is set with the lower bound $\mathcal{L}_1$ and each time it is proved that no solution exists, the number of bins is incremented until a solution is met. The complete implementation is given on Listing 4.1. There are only two sets of variables:

- `x[i]` indicates the bin where item `i` is placed, and
- `l[j]` is the total size packed into bin `j`.

The integer `nbBins` is the number of available bins initially computed with the $\mathcal{L}_1$ lower bound. These two sets of variables are linked with the `multiknapsack` constraint and we additionally add our new failure detection propagator called `binpackingLB`.

We start by explaining the search and the dynamic symmetry breaking strategy then we explain the dominance rules in Listing 4.2. The items are assumed to be sorted in decreasing weights. The search starts by setting to 0 the capacity of the bins with an index larger than `nbBins` to ensure that at most `nbBins` are available. Then the heaviest available item is selected (line 23) and placed in the first available bin (line 25). The array `currfill` which is the current packed load of each bin (sum of the weight of items placed into each bin) is used in the subsequent lines for the symmetry breaking. Note that at line 25, at most one empty bin is tried since all these empty bins are equivalent. When backtracking, the `onFailure` block is executed (meaning that the placement of item i did not succeed into bin b) and all the equivalent items are prevented to be placed into an equivalent bin in the alternatives. The call `cp.restartOnCompletion()` at line 10 causes the block `onRestart` to be executed if the satisfaction problem has no solution before restarting the search again. We simply increment the number of available bins by 1 in this `onRestart` block.

The Listing 4.2 that must be plugged into line 21 of Listing 4.1 implements the two dominance rules. The dominance rules detection is executed at each node of the tree. The two rules are placed inside a loop (`while (dominance)`) until no more dominance is detected and the branching can continue. The first simpler rule is implemented at lines 4–5 and tries to detect if a bin can be completed up to capacity with one item. The second rule lines 10–28 checks if it is impossible to

Listing 4.1: Bin-Packing Comet Model

```
range Items; range Bins;
int weights [Items]; int capa;
float totWeight = sum(i in Items) weights[i];
Integer nbBins((int)ceil(totWeight/capa));

Solver<CP> cp();
var<CP>{int} x[Items](cp,Bins);
var<CP>{int} l[Bins](cp,0..capa);

cp.restartOnCompletion();
solve<cp> {
  cp.post(multiknapsack(x,weights,l));
  cp.post(binpackingLB(x,weights,l));
} using {
  forall(b in Bins: b>nbBins)
    cp.label(l[b],0);
  while(!bound(x)){
    int currfill[b in Bins] =
      sum(i in Items: x[i].bound() && x[i].getValue()==b) weights[i];

    // dominance rules from Listing 4.2

    selectMin(i in Items:!x[i].bound()) (i){
      int ms = max(0,maxBound(x));
      tryall<cp>(b in 1..ms+1)
        cp.label(x[i],b);
      onFailure{
        forall(j in Items: !x[j].bound() && weights[j]==weights[i])
          forall(b_ in 1..ms+1: currfill[b_]==currfill[b]){
            cp.diff(x[j],b_);
          }
      }
    }
  }
}
onRestart {
  nbBins := nbBins+1;
}
```

pack two items into a bin. If this is impossible, lines 22–27 are executed and the largest item that can be placed is packed into the bin.

The objective of this experiment is to discover if our improved failure detection makes a difference in practice on pure bin-packing problems. We experiment the Comet model of Listing 4.2 on classical bin-packing instances introduced in [47] with the failure detection of [52] and with our new failure detection. Contrarily to [52] that experimented on the instances with 50 items, we work on larger instances with 100 items[3]. The data set if composed of 180 instances and we report only in Table 4.2 the results on the instances for which there is a difference in the search tree. The timeout was set to 300 seconds. A total of 10 instances could not be solved with our new failure detection test and 14 instances could not be solved with Shaw's test. These 4 instances that we can solve with our new test are the most impressive differences. The results on instances N2C1W1B and N2C2W1A are particularly appealing since we obtain a small number of failures and we reach the timeout with Shaw's test. The other results are less significant since we are only able to reduce the number of failures from a few percents up to 50 percents on some instances.

A Dynamic Redundant Global Cardinality Constraint

The conjunction of a multiknapsack constraint and a global cardinality constraint (`gcc`) [34] occurs in the *cardinality constrained bin packing problem* [20, 24]. We show in this section that a bin-packing reasoning allows to filter the cardinalities and consequently the places where the items can be placed.

The idea is to reason on the maximum number of items that can be placed into a bin as illustrated in the following example.

Example 22. Consider a pack constraint with 5 bins of capacity 3 and five items of size 2 $\texttt{Pack}([B_1, ..., B_5], [2,2,2,2,2], [L_1, ..., L_5])$, $\forall i \in [1..5]$, $Dom(L_i) = [0..3]$, $Dom(B_1) = Dom(B_2) = \{1,2\}$, $Dom(B_3) = Dom(B_4) = \{3,4\}$, $Dom(B_5) = \{1, \ldots, 5\}$. Clearly because of the capacity of the bins (3) and the size of the items (2), each bin can accept at most one item. Since the four first bins are occupied by the four first items, the fifth item can only be placed in in the last bin: $Dom(B_5) = \{\cancel{1}, \cancel{2}, \cancel{3}, /$

[3]these instances are very difficult to solve without the additional dominance rule that we use.

Listing 4.2: Dominance rules for Bin-Packing

```
bool dominance=true;
while(dominance){
  dominance=false;
  select(i in Items, b in Bins: !x[i].bound() &&
                    currfill[b]+weights[i]==capa && b<=nbBins){
    cp.label(x[i],b);
    currfill[b] += weights[i];
    dominance=true;
  }
  forall(b in Bins: b<=nbBins){
    set{int} candidates = filter(i in Items)(!x[i].bound() &&
                                    currfill[b]+weights[i]<=capa);
    bool possible = false;
    selectMin(i in candidates)(weights[i]){
      selectMin(j in candidates:j!=i)(weights[j]){
        if(currfill[b]+weights[i]+weights[j]<=capa){
          possible=true;
        }
      }
    }
    if(! possible){
      selectMax(i in candidates)(weights[i]){
        cp.label(x[i],b);
        currfill[b] += weights[i];
        dominance=true;
      }
    }
  }
}
```

Table 4.2: Comparison of the improved failure detection on bin-packing instances

	New failure detection			**Shaw Failure Detection**		
Instance	Bins	Fails	Time(s)	Bins	Fails	Time(s)
N2C1W1A	48	174208	200	-1	243317	300
N2C1W1B	49	1640	26	-1	901785	300
N2C1W1D	50	7684	70	50	15329	75
N2C1W1E	58	1551	50	58	2763	58
N2C1W1F	50	26359	92	50	47481	117
N2C1W1H	52	3318	76	52	3601	77
N2C1W1O	48	2512	82	48	3358	83
N2C1W4A	73	2511	63	73	4523	78
N2C1W4E	73	4902	78	73	4971	78
N2C1W4M	72	2440	59	72	4726	73
N2C2W1A	42	1420	28	-1	700851	300
N2C2W1H	46	346579	142	-1	678265	300
N2C2W2B	56	1485	52	56	2863	67
N2C2W2E	54	1378	51	54	3306	65
N2C2W2F	48	10763	77	48	15173	79
N2C2W2I	49	8997	78	49	15736	80
N2C2W2K	50	3104	57	50	3445	56
N2C2W2M	54	1378	44	54	2506	51
N2C2W4T	57	1540	50	57	2815	74
N2C3W2D	41	114645	134	41	118918	127
N2C3W2S	43	21153	42	43	21606	42

$4, 5\}$. This filtering is not achieved by the `IloPak` constraint and would not be possible neither with the preemption filtering based on flows.

The reasoning used to filter the domain of the B_i's is achieved in two steps:

1. For each bin, an upper bound on the maximum number of items that can be placed into that bin is computed given the capacity of the bin.

2. A global cardinality constraint (`gcc` [34]) can be used to filter the domain of the $B_i's$ given the cardinalities computed in previous step.

A `gcc`$([D_1, \ldots, D_m], [X_1, \ldots, X_n])$ constraints holds if and only if $\forall i \in [1..m] : D_i = |\{X_j | X_j = i \wedge j \in [1..n]\}|$. We can post a cardinality constraint `gcc`$([D_1, \ldots, D_m], [B_1, \ldots, B_n])$ redundant to `Pack`$([B_1, ..., B_n], [s_1, ..., s_n], [L_1, ..., L_m])$ with a propagator responsible to filter the upper bound of the cardinality variable D_i by computing an upper bound on the number of items that can be placed inside the bin i. The filtering applied by the propagator on the cardinality variable D_i of the bin i is

$$D_i^{\max} \leftarrow \min(D_i^{\max}, \max_{T \subseteq C_i}\{|T| \mid \mathrm{sum}(R_i \cup T) \leq L_i^{\max}\})$$

where C_i is the candidate set of bin i (unpacked items that have bin i inside their domain). This propagation rule is computed in linear time if the items are sorted in increasing size.

The constraint `gcc`$([D_1, \ldots, D_m], [B_1, \ldots, B_n])$ is thus a redundant constraint where the domain of the variables $D_1, \ldots, D_m$ are tightened during the search by an external propagator. This redundant constraint is particularly useful when the sizes of the items are nearly equal because in this case the upper bounds computed on the cardinality variables are quite precise. This is the case for example in the triplet data set for the bin-packing problem [9]. Note that to gain efficiency, the `gcc` should not propagate the cardinality variables because it is time consuming and not useful in this case [4].

[4]Unfortunately it is often not possible to deactivate this propagation

4.2 Bin-Packing with Precedence Constraints

We are interested here in bin packing problems with precedence constraints between items (BPPC). A precedence constraint between items a_1 and a_2 is satisfied if item a_1 is placed in a bin B_1, and item a_2 in a bin B_2, with $B_1 \leq B_2$.

BPPC are ubiquitous in assembly line balancing problems in the industry[5]. We are given a set of tasks of various lengths, subject to precedence constraints, and a time constant called cycle time. The problem is to distribute the tasks over workstations along a production (assembly) line, so that no workstation takes longer than the cycle time to complete all the tasks assigned to it (station time), and the precedence constraints are satisfied. The decision problem of optimally partitioning (balancing) the tasks among the stations with respect to some objective is called the assembly line balancing problem (ALBP) [4, 49].

In particular, when the number of stations is fixed, the problem is to distribute the tasks to stations such that the station time is balanced, that is to minimize the cycle time. This type of problems is usually called Simple ALBP-2 (SALBP-2) [4]. It is clear that SALBP-2 and BPPC are equivalent.

BPPC and SALBP-2 can be solved by exact or heuristic methods. We are interested here in *exact* methods using constraint programming. Existing exact methods are usually dedicated branch and bound algorithms such as Salome 2 [22]. These algorithms are very efficient and have been improved since about 50 years. Unfortunately these algorithms are not flexible to new constraints that might appear in real applications such as minimal distance between two tasks or restriction on the cumulated value of a particular task attribute (see the problem classifier available on *www.assembly-line-balancing.de* for more details). When such constraints are added, we obtain so-called Generalized Assembly Line Balancing Problems (GALBP) [1]. The existing exact methods are not flexible enough to handle GALBP efficiently. The Constraint Programming (CP) paradigm is a good candidate to tackle such problems since constraints can be added very easily to the model.

The Balanced Academic Curriculum Problem (BACP) is equivalent to BPPC. The objective is to schedule courses into a given number of periods such that the prerequisites relations between the courses are sat-

[5]See for example the two commercial softwares Proplanner® *www.proplanner.com* and OptiLine® *www.optimaldesign.com*

isfied and such that the workloads among the periods are well balanced. Basic CP models have been proposed in [5, 18].

We propose a CP model for BPPC and SALBP-2, allowing a flexible expression of new constraints, such as in GALBP. More specifically, our contributions are:

- An efficient CP model for the bin packing with precedence constraints (BPPC) with redundant constraints.
- A new global constraint for BPPC and its filtering algorithm exploiting the transitive closure of the precedence graph.
- An experimental validation on standard SALBP-2 benchmarks, showing the feasibility, the efficiency, and the flexibility of this approach.

4.2.1 A CP Model for the Bin Packing with Precedence Constraints

The bin packing with precedence constraint (BPPC) is the conjunction of one

- `Pack`$([B_1, ..., B_n], [s_1, ..., s_n], [L_1, ..., L_m])$ constraint, and
- a set of precedence constraints $B_i \leq B_j$ for each edge $(i, j) \in E$ of a precedence (directed and acyclic) graph $G(\{1, .., n\}, E)$.

The filtering obtained by posting the `Pack` constraint and the precedences constraints separately can be improved with redundant constraints. We introduce one set variable $\mathcal{P}_i$ representing the predecessors of item i. An item j is predecessor of an item i if and only if item j is placed in a bin preceding or equal to the bin of item i:

$$j \in \mathcal{P}_i \leftrightarrow B_j \leq B_i.$$

The lower bound of $\underline{\mathcal{P}_i}$ can be initialized to i plus the set of items having an arc pointing to i in the transitive closure of the precedence graph. The transitive closure is computed with the $O(n^3)$ Floyd Warshall's algorithm (see [6]). The preprocessing time to compute the transitive closure for the instantiation of the lower bounds $\underline{\mathcal{P}_i}$ is negligible for the instances considered in the experimental section (less than 150 tasks). The upper bound $\overline{\mathcal{P}_i}$ is initially all the tasks, that is $\{1, ..., n\}$. A similar reasoning holds for the successor set variable $\mathcal{S}_i$ of item i. The set of predecessors of item i can be used to filter the lower bound of B_i since we know that items in $\mathcal{P}_i$ must be placed before item i. For a set U with elements taken from $\{1, ..., n\}$, we denote by $\text{sum}(U)$ the total size of items in U, that is $\text{sum}(U) = \sum_{j \in U} s_j$.

Theorem 11. A redundant constraint for the `BPPC` is:

$$\text{sum}(\mathcal{P}_i) = \sum_{k \leq B_i} L_k. \tag{4.1}$$

Proof. The left and right members are simply two different ways of counting the cumulated size of the B_i first bins. The right member is the natural way and the left member counts it by summing the sizes of the items lying in a bin smaller or equal to B_i. □

Computation of $\text{sum}(\mathcal{P}_i)$**:** Set variables are not yet implemented in COMET. Therefore, we represent the set of predecessors of item i with a vector of binary variables $P_{i,j}$ with $P_{i,j} = 1 \leftrightarrow B_j \leq B_i$. The computation of $\text{sum}(\mathcal{P}_i)$ is then a binary knapsack constraint: $\text{sum}(\mathcal{P}_i) = \sum_{j \in [1..n]} P_{i,j} \cdot s_j$. The binary knapsack constraint can be expressed as such with a sum constraint however the filtering is very weak. A pseudo polynomial time algorithm was introduced in [55] to achieve arc consistency on a binary knapsack constraints[6]. However in line balancing instances, the size of the items can be arbitrarily large hence we prefer a weaker but strongly polynomial filtering algorithm. We use the `lightBinaryKnapsack` constraint of COMET which uses the filtering of the `multiknapsack` constraint for a single bin.

In Ilog Solver, $\mathcal{P}_i$ is represented with subset bound set variables and $S(\mathcal{P}_i) = \sum_{j \in \mathcal{P}_j} s_j$ is expressed as such in Ilog Solver using a global constraint called `IloEqSum` making a summation over a set variable. A function must be defined to make the mapping between the indices of items and the size of the items: $f : \{1, ..., n\} \mapsto \{s_1, ..., s_n\} : f(j) = s_j$. The global constraint takes three arguments: a set variable, a variable and a function:

$$\texttt{IloEqSum}(\mathcal{P}_i, \text{sum}(\mathcal{P}_i), f) \equiv \sum_{j \in \mathcal{P}_i} f(j) = \text{sum}(\mathcal{P}_i).$$

Unfortunately, we have no idea about the filtering achieved by `IloEqSum`.

[6]This algorithm is implemented as `binaryknapsack` in COMET

Computation of $\sum_{k \leq B_i} L_k$: The formulation of $\sum_{k \leq B_i} L_k$ could be achieved with m binary variables for each item i. A better formulation is possible introducing an array of m variables $\mathbf{CL} = [CL_1, ..., CL_m]$: $CL_i = \sum_{k=1}^{i} L_k$ for $i \in [1, ..., m]$ (CL for Cumulated Load). With this array, $\sum_{k \leq B_i} L_k$ can be written with an element constraint [56] as $\mathbf{CL}_{B_i}$. We have implemented a dedicated bound-consistent element constraint (`sortedElement`) faster than the general implementation of the element constraint by taking into account that the variables $[CL_1, ..., CL_m]$ are increasing.

The Comet model of the redundant constraints (4.1) for the predecessors of item i is:

$$\begin{aligned}\texttt{lightBinaryKnapsack}([P_{i,1}, \dots, P_{i,n}], [s_1, \dots, s_n], \mathrm{sum}(\mathcal{P}_i)) \wedge \\ \texttt{sortedElement}([CL_1, ..., CL_m], B_i, \mathrm{sum}(\mathcal{P}_i)).\end{aligned} \tag{4.2}$$

The Ilog model of the redundant constraints (4.1) for the predecessors of item i is:

$$\texttt{IloEqSum}(\mathcal{P}_i, \mathrm{sum}(\mathcal{P}_i), f) \wedge \mathrm{sum}(\mathcal{P}_i) = \mathbf{CL}_{B_i}. \tag{4.3}$$

Constraint (4.3) filters the domains of $\mathcal{P}_i$, B_i and the L_i's. We define similar constraints for the successor variables $\mathcal{S}_i$.

4.2.2 A Global Constraint for the BPPC

We give an $O(n^2)$ algorithm to filter further the domains of $[B_1, ..., B_n]$ and $[L_1, ..., L_m]$. This filtering does not subsume the filtering obtained with the redundant constraints. Hence it must be added to the filtering obtained with the redundant constraints.

By adding the redundant constraints (4.3) from previous section, we mainly prune the lower bound of the variable B_i. Considering the array of the upper bounds of the bin loads $[L_1^{\max}, ..., L_m^{\max}]$, the redundant constraint enforces that

$$B_i^{\min} \leftarrow \min\{j : \sum_{k=1}^{j} L_k^{\max} \geq \sum_{j \in \mathcal{P}_i} s_j\} \tag{4.4}$$

The filtering rule (4.4) is one of the pruning achieved by (4.3).

Example 23. An item has a size of 4 and has three predecessors of size 4,3,5. The maximum height of all the bins is 5. The item can certainly not be placed before the bin 4 because for the bin 4 we have $\sum_{k=1}^{4} L_k^{\max} = 20 \geq 16$ while for the bin 3 we have $\sum_{k=1}^{3} L_k^{\max} = 15 < 16$.

Rule (4.4) is a relaxation of the largest lower bound that could be found for B_i:

- it assumes a preemption of the items over the bins, and
- it assumes that all the predecessors can potentially start from the first bin.

We propose an algorithm to compute a better lower bound by conserving the preemption relaxation but disallowing a predecessor j to start before its earliest possible bin $B_j^{\min}$.

Our algorithm requires the predecessors j to be sorted increasingly with respect to their earliest possible bin $B_j^{\min}$. This is achieved in $\Theta(|\underline{\mathcal{P}}_i| + m)$ with a counting sort algorithm [6] since the domains of the B_j's range over $[1, ..., m]$. This complexity can be simplified to $O(n)$ since $|\underline{\mathcal{P}}_i| < n$ and typically $m \sim O(n)$ (less bins than items).

Algorithm 10 computes the minimum possible bin for item i by considering that :

- each predecessor j cannot start before its earliest possible bin $B_j^{\min}$ but can end in every other larger bin and
- an item can be split among several bins (the preemption relaxation).

Algorithm 10 first places the predecessors of i that is elements of $\underline{\mathcal{P}}_i$. This is done in the **forall** external loop. Then the item i is placed in the earliest possible bin without preemption for it. We assume that there are $m+1$ bins of capacity $[L_1^{\max}, ..., L_m^{\max}, \sum_{i=1}^{n} s_i]$. The additional fictive $(m+1)^{th}$ bin has a capacity large enough such that every items can be put inside it. This guarantees the termination of the **while** loops. The complexity of the algorithm is $O(|\underline{\mathcal{P}}_i| + m)$ where m is the number of bins. For n items, the complexity becomes $O(n^2)$.

Algorithm 10 returns two values *bin* and *idle*. The value *bin* is used to prune the lower bound of B_i:

$$B_i^{\min} \leftarrow \max(B_i^{\min}, bin).$$

The value *idle* is used to prune $L_{bin}^{\min}$. Indeed if B_i is assigned then $bin = B_i$. It means that $L_{bin}^{\min}$ must be at least larger that $L_{bin}^{\max} - idle$:

$$L_{bin}^{\min} \leftarrow \max(L_{bin}^{\min}, L_{bin}^{\max} - idle).$$

Of course, we use a similar filtering using the set variable $\mathcal{S}_i$ to filter the upper bound of B_i.

4.2.3 Experimental results

We experimented three different models for the BPPC on SALBP2 instances:

A is a simple model obtained with the `Pack` constraint and the precedence constraints.

B is the model A + the redundant constraints.

C is the model B + the global constraint.

The results are given on Table 4.3.

Analysis of results: The model B and C clearly dominate the model A. See for example the instance LUTZ2 (10 ws) which cannot be solved with model A but can easily be solved for model B and C. The global constraint introduced in model C improves a bit the filtering but it does not compensate the overhead in time introduced by the algorithm. See for example the instance GUNTHER (8 ws) where the number of backtrack with B is 3314 and 1635 for C but the time was increased from 4 seconds to 8 seconds. This overhead could be reduced by implementing it in C++ rather than in the Comet language but even so, it does not seem very interesting since the number of backtracks is never reduced by a huge factor.

Discussion and Comparison with state of the art dedicated algorithm: The state of the art algorithm for this problem is Salome 2 [22, 49]. A binary file of the implementation of the algorithm is available on *www.assembly-line-balancing.de*. Salome 2 finds the optimal solutions of almost all the instances within less than one second. As with our solution, it is not able to find and prove the optimum for the instances Wee-mag 10. Salome 2 uses a lot of dominance and reduction rules specific to this problem and objective function. In real life assembly line problems, additional requirements are possible and the dominance rules used in Salome 2 are not valid anymore. Some possible additional constraints are [1]:

- some tasks must be assigned in the same station,
- some tasks can not be assigned in the same station,

- there is a restriction on the cumulated value of particular task attributes,
- some tasks need to be assigned to particular stations,
- some tasks can not be assigned to particular stations,
- some tasks need a special station,
- some tasks need a minimum distance to other tasks,
- some tasks need a maximum distance to other tasks.

All these additional constraints can be very easily added our CP model without changing anything else while dedicated algorithms such as Salome 2 cannot.

Another advantage of the constraint programming approach is the modularity offered.

The propagator for BPPC can be reused in a problem with another objective function. For example, it is often desirable to smooth the workload among a given number of stations [1, 49, 38]. This problem is called Vertical Line Balancing. We have seen in Chapter 3 that it can be efficiently achieved in CP with the global constraints `spread` and `deviation` for the variance [31, 43] and the mean absolute deviation [44].

Another classical problem in line balancing is the U-line Assembly Line Balancing Problem (UALBP). It considers the case of U-shaped (single product) assembly lines, where stations are arranged within a narrow U. In this problem, workers are allowed to work on both side of the U, *i.e.* on early and late tasks in the production process simultaneously. Figure 4.3 illustrate a problem with 4 work-stations (workers). The circuit followed by the product is given by the direction of arrows. The colors of the tasks is to distinguish in the solution the ones achieved in the *left-to-right* or in the *right-to-left* direction of the product. UALBP can be easily modeled in constraint programming and the BPPC can also be reused in this problem. The modelization trick is to divide each station into two stations, one for the left-to-right tasks of the station and one for the right-to-left tasks of the station. Hence if m work-stations are available, the load variables of the workstations are $L_1, \ldots, L_m$ before the division and it becomes $L_1^{\rightarrow}, \ldots, L_m^{\rightarrow}, L_m^{\leftarrow}, \ldots, L_1^{\leftarrow}$ with the load sorted in the direction of the product along the line. The

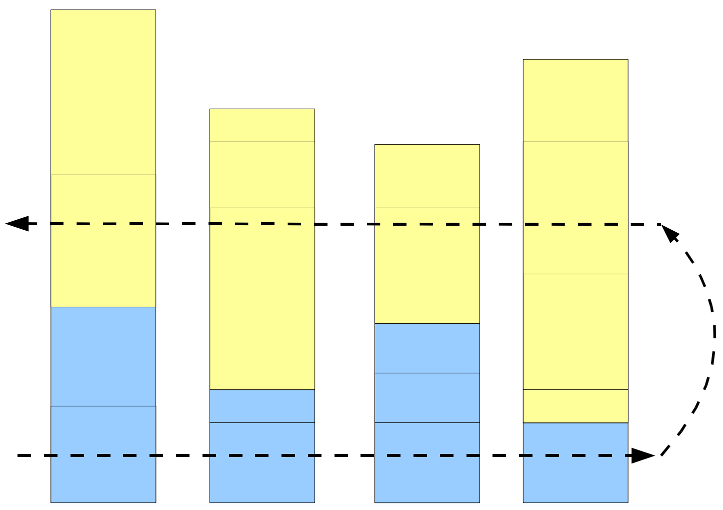


Figure 4.3: Example of the U-line Assembly Line Balancing Problem

precedence constraints of the BPPC are posted on these last load variables $L_1^{\rightarrow}, \ldots, L_m^{\rightarrow}, L_m^{\leftarrow}, \ldots, L_1^{\leftarrow}$ and the link to the real work-stations is simply that $L_j = L_j^{\rightarrow} + L_j^{\leftarrow}$.

Algorithm 10: Considering bins of maximum loads $[L_1^{\max}, ..., L_m^{\max}, \sum_{i=1}^{n} s_i]$, *bin* is the smallest bin index such that every item from the set $\underline{\mathcal{P}}_i$ have been placed in a preemptive way in a bin smaller or equal to *bin* without starting before their earliest possible bin. The value *idle* is the remaining space in this bin.

```
1  $bin \leftarrow 0$
2  $idle \leftarrow L_{bin}^{\max}$
3  forall $j \in \underline{\mathcal{P}}_i \setminus \{i\}$ do
       /* invariant: $bin$ is the smallest bin index such that items
          $\{1, ..., j-1\} \setminus \{i\}$ have all been placed in a preemptive way in a
          bin smaller or equal to $bin$ without starting before their earliest
          possible bin and $idle$ is the remaining place in this bin.          */
4      if $B_j^{\min} > bin$ then
5          $bin \leftarrow B_j^{\min}$
6          $idle \leftarrow L_{bin}^{\max}$
7      $s \leftarrow s_j$
8      while $s > 0$ do
9          if $idle > s$ then
10             $idle \leftarrow idle - s$
11             $s \leftarrow 0$
12         else
13             $s \leftarrow s - idle$
14             $bin \leftarrow bin + 1$
15             $idle \leftarrow L_{bin}^{\max}$
   /* place item $i$ without preemption                                    */
16 if $B_i^{\min} > bin$ then
17     $bin \leftarrow B_i^{\min}$
18     $idle \leftarrow L_{bin}^{\max}$
19 while $idle < s_i$ do
20     $bin \leftarrow bin + 1$
21     $idle \leftarrow L_{bin}^{\max}$
22 $idle \leftarrow idle - s_i$
23 return $bin$, $idle$
```

Table 4.3: Comparison of the BPPC with redundant constraints and global constraints

		A				**B**				**C**			
Instance	WS	Sol	Fails	t(s)	Max	Sol	Fails	t(s)	Max	Sol	Fails	Time(s)	Max
BUXEY	6	1	9	0	55	1	1	0	55	1	1	1	55
BUXEY	8	1	0	0	41	1	0	0	41	1	0	0	41
BUXEY	10	1	11	0	34	1	0	0	34	1	0	1	34
GUNTHER	6	1	6	0	84	1	0	0	84	1	0	1	84
GUNTHER	8	1	156175	30	63	1	3314	4	63	1	1635	8	63
GUNTHER	10	1	50206	18	50	1	6821	11	50	1	5422	21	50
HAHN	6	1	133	0	240	1	3	1	240	1	2	4	240
HAHN	8	1	281	1	190	1	14	1	190	1	10	4	190
HAHN	10	1	0	0	177	1	0	1	177	1	0	3	177
LUTZ1	6	1	78	0	2396	1	7	0	2396	1	6	1	2396
LUTZ1	8	1	1988	1	1860	1	355	1	1860	1	276	2	1860
LUTZ1	10	1	8533	4	1526	1	756	1	1526	1	422	2	1526
LUTZ2	6	1	7821	4	81	1	1	7	81	1	1	22	81
LUTZ2	8	1	2	0	61	1	1	7	61	1	1	21	61
LUTZ2	10	0	300599	200	50	1	81	8	49	1	39	24	49
LUTZ3	6	1	1588	1	275	1	6	9	275	1	4	24	275
LUTZ3	8	1	29966	25	207	1	7	9	207	1	3	24	207
LUTZ3	10	1	253053	139	165	1	264	11	165	1	91	27	165
MITCHELL	6	1	0	0	18	1	0	0	18	1	0	0	18
MITCHELL	8	1	0	0	14	1	0	0	14	1	0	0	14
MITCHELL	10	1	0	0	13	1	0	0	13	1	0	0	13
ROSZIEG	6	1	1	0	21	1	1	0	21	1	1	0	21
ROSZIEG	8	1	19	0	16	1	4	0	16	1	3	0	16
ROSZIEG	10	1	3	0	14	1	0	0	14	1	0	0	14
SAWYER30	6	1	6	0	55	1	0	0	55	1	0	1	55
SAWYER30	8	1	9	0	41	1	4	0	41	1	4	1	41
SAWYER30	10	1	14399	8	34	1	203	1	34	1	157	1	34
TONGE70	6	0	589183	200	586	0	57131	200	586	0	27784	200	586
TONGE70	8	0	448178	200	442	0	77357	200	442	0	31018	200	442
TONGE70	10	0	408923	200	352	1	10463	73	352	1	8542	124	352
WARNECKE	6	1	2227	2	258	1	99	2	258	1	99	6	258
WARNECKE	8	1	67519	36	194	1	23541	73	194	1	20751	139	194
WARNECKE	10	0	182235	200	157	0	59789	200	157	0	24856	200	157
WEE-MAG	6	1	1	0	250	1	1	4	250	1	1	11	250
WEE-MAG	8	1	9	0	188	1	6	4	188	1	6	11	188
WEE-MAG	10	0	517989	200	152	0	55917	200	152	0	26159	200	152

5
Conclusion

In the first part of this thesis we have introduced the bound consistent filtering algorithms of two balancing constraints: `spread` and `deviation`:

- `spread` constraints the variance of a set of variables with a fixed mean, and
- `deviation` constraints the mean absolute deviation of a set of variables with a fixed mean.

The classical bound-consistency definition has been refined into the $\mathbb{Q}$-Bound-Consistency and $\mathbb{Z}$-Bound-Consistency definitions following that the relaxation is on in rational or integer interval domains. The propagators for `spread` and `deviation` have been developed for both definitions.

These constraints have been used to solve exactly instances of two combinatorial problems:

- the vertical assembly line balancing problem, and
- the work load balancing of nurses in a hospital.

We have also shown that balancing constraints are useful to soften in a new way a particular case of the global cardinality constraint in which the ideal value distribution is given.

In the second part of this work we have improved the filtering the bin-packing constraint by introducing a flow-based filtering. The results of this approach are not really encouraging. We also improve a failure detection algorithm for the bin-packing constraint using a new reduction to a fast bin-packing lower-bound algorithm.

Finally we have shown that when bin-packing comes with precedences between items, more of the structure can be captured by expressing redundant constraints based on the transitive closure of the precedence graph. We also introduced a filtering algorithm for a bin-packing constraint with precedences between items. Experiments show that the new filtering helps a lot to solve assembly line balancing instances.

Perspective and Future works

Balancing Constraints: Next table summarizes what has been done for balancing constraints with a fixed mean:

	`spread`		`deviation`	
	$\mathbb{Q}$-BC	$\mathbb{Z}$-BC	$\mathbb{Q}$-BC	$\mathbb{Z}$-BC
Δ	$\surd$ [31]	$\surd$	$\surd$	$\surd$
$\mathbf{X}$	$\surd$	$\surd$	$\surd$	$\surd$

The original specification of `spread` in [31] considered a variable mean. The algorithms become more complicated with a variable mean and some of them still need to be discovered as shown on the next table.

	`spread`		`deviation`	
	$\mathbb{Q}$-BC	$\mathbb{Z}$-BC	$\mathbb{Q}$-BC [1]	$\mathbb{Z}$-BC
Δ	$\surd$ [31]	$\surd$ [2]	$\surd$	
$\mathbf{X}$			$\surd$	
S	$\surd$ [31, 43]		$\surd$	

As pointed out by Yves Crama, resource allocation problems have strong similarities with `spread` and `deviation`. The next step is probably to implement a more general resource allocation global constraint of the form

$$\texttt{resource-allocation}([X_1,\ldots,X_n], s, [f_1,\ldots,f_n], \Delta)$$

[1]Described in a report but never published

[2]Informally described by Jean-Charles and I but never published

constraining that:

$$\sum_{i=1}^{n} f_i(X_i) \leq \Delta$$

$$\text{such that : } \sum_{i=1}^{n} X_i = s$$

where the f_i are convex functions. Finding a $\mathbb{Q}$-bound-consistent and $\mathbb{Z}$-bound-consistent lower bound for Δ can be done respectively with algorithm `BRELAX2` and `CONTINUOUS2` described p29 and p76 of [19]. But we think that pruning the bounds of variables X_i is a more difficult exercise since those algorithms must still be discovered.

As pointed out by Jean-Charles Régin, it would be interesting to be able to optimize the balance with respect to several balancing objective. For instance we could imagine minimizing the `max`, `spread` and `deviation` criteria altogether. Be able to be Pareto optimal with respect to several objective functions in CP is also an interesting topic for future work.

Bin-Packing Constraints: We think that the flow based filtering is a preliminary work but it might be worth to investigate it further. It presents the advantage to consider globally all the bins and items at once. We also think that it could be worth to try scaling algorithms for bin-packing as it was the achieved for `knapsack` filtering. Finally, the bin-packing constraint with precedences can also be improved. As future work, we want to integrate the fast bin-packing lower bound algorithms in this constraint. For instance, a bin-packing lower bound can be used on the predecessors of an item to compute its earliest possible bin.

6

Annexes

Available Propagators

The most successful filtering algorithms introduced in this thesis are available in Comet allowing to someone else from the scientific community to reproduce the results. We review the API of the exposed propagators in Comet.

Balancing Constraints

The $\mathbb{Z}$-bound-consistent propagators for `spread` and `deviation` with a fixed sum/mean are implemented and available in the Comet API. The $\mathbb{Q}$-bound-consistent propagators were also implemented internally but there are not publicly exposed since our experiences have shown that they were less efficient in terms of filtering for a same time complexity. The signature of the function that returns a `spread` constraint is the following:

```
spread(var<CP>{int}[] x,int s,var<CP>{int} nd)
```

The arguments correspond exactly to the ones of `spread`$(\mathbf{X}, s, \Delta)$ used along this text. The signature of the function that returns a `deviation` constraint is:

```
deviation(var<CP>{int}[] x,int s,var<CP>{int} nd)
```

where the arguments also correspond to the ones of `deviation`$(\mathbf{X}, s, \Delta)$ used along this text.

Bin-Packing Constraints

Two bin-packing propagators using bin-packing products of this thesis are available in COMET.

The first is the bin-packing lower bound algorithm with the signature:

```
binpackingLB(var<CP>{int}[] x,int[] size,var<CP>{int}[] load)
```

The arguments are the same as for the `multiknapsack` constraint that is:

- `x` is the vector representing the bin where each item is placed,
- `size` is the vector representing the size of each item, and
- `load` is the vector representing the load variable of each bin (i.e. the sum of the sizes of items placed into that bin).

The bin-packing reductions used are the new one introduced in this thesis as well as the one of Paul Shaw and the bin-packing lower bound algorithm implemented is the one of Labbé.

The second bin-packing constraint available is the bin-packing with precedence constraints. This is a function that doesn't return a COMET constraint (void) like previous functions but it posts all the redundant constraints when you call it.

```
multiknapsackWithPrecedences(var<CP>{int}[] x,int[] size,
             var<CP>{int}[] load,PrecedenceGraph predGraph)
```

There is only one extra argument that is the precedence graph that must be build before calling the function with the following API:

```
PrecedenceGraph::PrecedenceGraph(range nodes)
void addPrecedence(int i1,int i2)
set{pred} getAddedPrecedences()
range getItems()
int getNbPredecessors(int )
int getNbSuccessors(int i)
set{int} getPredecessors(int i)
set{int} getSuccessors(int i)
```

The constructor only asks for a range that represents the nodes of the graph. Then a precedence can be added between two nodes `i1` and `i2` with the method `addPrecedence`. There is no need to express explicitly the transitive closure of the graph since this one is computed behind the scene and the predecessors/successors of a node `i` in the transitive closure can be retrieved with `getPredecessors`/`getSuccessors`. The implementation of this constraint relies only on the redundant constraints introduced in Section 4.2 together with a post of the `multiknapsack` constraint available in COMET implementing the filtering of Shaw. The global constraint introduced in 4.2.2 was not exposed since the stronger filtering did not pay off for most of the instances when added to the redundant constraints.

Problem definitions

BACP (Balanced Academic Curriculum Problem) The goal is to assign a period to courses in a way that the prerequisite relationships are satisfied and the academic load of each period is balanced. Each course is weighted by its number of credits. A prerequisite relation is a strict precedence ($<$).

ALBP (Assembly Line Balancing Problem A given number of workstations are placed along a conveyor belt. The workpieces are consecutively launched down the line from station to station until the end of the line. Some operations are performed on any workpieces in each station. The problem is to assign all the operations to the workstations such that precedences between the operations are satisfied and some objective is optimized.

SALBP2 (Simple ALBP of type 2) A particular version of the general ALBP where the number of stations is fixed and the problem is to distribute the tasks to stations such that cycle time is minimized. This problem is similar to the BACP except that the precedences are not strict ($\leq$).

VSALBP2 (Vertical Simple ALBP of type 2) Same problem as the SALBP2 except that the objective is to balance workload of the stations rather that minimizing the cycle time.

(BPP) Bin-Packing problem Given a bin capacity and sized items, the objective is to minimize the number of bins necessary to place every items (items cannot be split among several bins).

(BPPC) Bin-Packing problems with Precedence Constraints We use this term to denote any problems where sized items must be placed into bins with ($\leq$) precedence constraints between items.

(FANP) Fair Assignment of Nurses to Patients in a hospital This problem is to assign infants to nurses while optimizing the balance of the nurse work loads which is essential for optimal quality of care. Each infant is characterized by the amount of work he requires and the zone where he is located. A nurse can work in only one zone. A nurse cannot be responsible of more than a given upper bound of number of children. The total amount of acuity of a nurse cannot exceed a given value.

Bibliography

[1] Christian Becker and Armin Scholl. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3):694–715, 2006.

[2] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Disjoint, partition and intersection constraints for set and multiset variables. In *Principles and Practice of Constraint Programming CP 2004*, pages 138–152, 2004.

[3] J.-M. Bourjolly and V. Rebetez. An analysis of lower bound procedures for the bin packing problem. *Computer & Operations Research*, 32:395–405, 2005.

[4] Nils Boysen, Malte Fliedner, and Armin Scholl. A classification of assembly line balancing problems. *European Journal of Operational Research*, pages 674–693, 2007.

[5] C. Castro and S. Manzano. Variable and value ordering when solving balanced academic curriculum problem. *Proc. of the ERCIM WG on constraints*, June 2001.

[6] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2001.

[7] Edward G. Coman, Jr. János Csirik, David S. Johnson, and Gerhard J. Woeginger. An introduction to bin packing. *Compiles bibliograhy for a forthcoming book*, 2004.

[8] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI*, pages 290–295, 1988.

[9] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[10] M. R. Garey and David S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[11] C. Gervet. New structures of symbolic constraint objects: sets and graphs, 1993.

[12] Carmen Gervet. Constraints over structured domains. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 17. Elsevier, 2006.

[13] Carmen Gervet and Pascal Van Hentenryck. Length-lex ordering for set csps. In *AAAI.* AAAI Press, 2006.

[14] 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.

[15] Stephen Gorard. Revisiting a 90-year-old debate: The advantages of the mean deviation. *British Journal of Educational Studies*, pages 417–439, 2005.

[16] P. Van Hentenryck and L. Michel. The steel mill slab design problem revisited. *CP'AI'OR-08, Paris, France*, 5015:377–381, May 2008.

[17] Pascal Van Hentenryck, Justin Yip, Carmen Gervet, and Grégoire Dooms. Bound consistency for binary length-lex set constraints. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 375–380. AAAI Press, 2008.

[18] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Modelling a balanced academic curriculum problem. *Proceedings of CP-AI-OR-2002*, 2002.

[19] T. Ibaraki and N. Katoh. *Resource Allocation Problems.* MIT Press, 1988.

[20] H. Kellerer and U. Pferschy. Cardinality constrained bin-packing problems. *Annals of Operations Research*, 92:335348, 1999.

[21] Moutaz Khouja and Conrad Robert. Balancing the assignment of customer groups among employees. *International Journal of Operations and Production Management*, pages 76–85, 1995.

[22] Robert Klein and Armin Scholl. Maximizing the production rate in simple assembly line balancing – a branch and bound procedure. *European Journal of Operational Research*, 91(2):367–385, June 1996.

[23] R. Korf. An improved algorithm for optimal bin packing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1252–1258, 2003.

[24] M. Labbé, G. Laporte, , and S. Martello. Upper bounds and algorithms for the maximum cardinality bin packing problem. *European Journal of Operational Research*, 149:490–498, 2003.

[25] Martine Labbé, Gilbert Laporte, and Hélène Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39:616–622, August 1991.

[26] JL Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 1978.

[27] S. B. Liu, H. L. Ong, and H.C. Huang. Two bi-directional heuristics for the assembly line type 2 problem. *The International Journal of Advanced Manufacturing Technology*, 22:656–661, 2003.

[28] Silvano Martello and Paolo Toth. *Knapsack Problems (chapter 8)*. John Wiley & Sons Inc, 1990.

[29] Alan G. Merten and Mervin E. Muller. Variance minimization in single machine sequencing problems. *Management Science*, 18(9):518–528, 1972.

[30] C Mullinax and M Lawley. Assigning patients to nurses in neonatal intensive care. *Journal of the Operational Research Society*, 53:25–35, 2002.

[31] G. Pesant and J.C. Régin. Spread: A balancing constraint based on statistics. *Lecture Notes in Computer Science*, 3709:460–474, 2005.

[32] Thierry Petit, Jean-Charles Regin, and Christian Bessiere. Meta-constraints on violations for over constrained problems. *IEEE International Conference on Tools with Artificial Intelligence*, pages 358–365, 2000.

[33] Rachamadugu R and Talbot B. Improving the equality of workload assignments in assembly lines. *International Journal of Production Research*, 29:619–633, 1991.

[34] J-C. Régin. Generalized arc consistency for global cardinality constraint. *AAAI-96*, pages 209–215, 1996.

[35] J.C. Régin. Habilitation à diriger des recherches (hdr) : modelization and global constraints in constraint programming. *Université Nice*, 2004.

[36] J.C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An original constraint based approach for solving over constrained problems. *Sixth International Conference on Principles and Practice of Constraint Programming (CP 2000)*, 1894, 2000.

[37] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, pages 362–367, 1994.

[38] B. Rekiek, P. De Lit, F. Pellichero, E. Falkenauer, and A. Delchambre. Applying the equal piles problem to balance assembly lines. *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning*, pages 399–404, 1999.

[39] P. Schaus and Y. Deville. A global constraint for bin-packing with precedences: Application to the assembly line balancing problem. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 369–374, Chicago, Illinois, USA, July 2008. AAAI Press.

[40] P. Schaus, Y. Deville, and P. Dupont. Bound-consistent deviation constraint. In *13th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4741 of *Lecture Notes in Computer Science*, pages 620–634, Providence, RI, USA, September 2007. Springer.

[41] P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. Simplification and extension of the spread constraint. In *Future and Trends of Constraint Programming*, pages 95–99. ISTE, 2007.

[42] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. Simplification and extension of spread. *3th Workshop on Constraint Propagation And Implementation*, 2006.

[43] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. Simplification and extension of the spread constraint. *Third International Workshop on Constraint Propagation And Implementation*, 2006.

[44] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. The deviation constraint. In *4th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 4510 of *Lecture Notes in Computer Science*, pages 269–284, Brussels, Belgium, 2007. Springer.

[45] P. Schaus, P. Van Hentenryck, and J.C. Régin. Scalable load balancing in nurse to patient assignment problems. In *6th International Conference Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, Lecture Notes in Computer Science, Pittsburgh, Pennsylvania, USA, 2009. Springer.

[46] Pierre Schaus, Pascal Van Hentenryck, and Alessandro Zanarini. Corrected algorithm for the soft global cardinality constraint. In *TO APPEAR*, 2010.

[47] A. Scholl, R. Klein, and C. Jurgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 24:627645, 1997.

[48] Armin Scholl. Data of assembly line balancing problems. *Technische Universitat Darmstadt*, 93.

[49] Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, 2006.

[50] Armin Scholl and Stefan Voß. Simple assembly line balancing: Heuristic approaches. *Journal of Heuristics*, 2(3):217–444, 1997.

[51] M. Sellmann. Approximated consistency for knapsack constraints. *9th International Conference on Principles and Practice of Constraint Programming (CP 2003)*, page 679693, 2003.

[52] Paul Shaw. A constraint for bin packing. In *Principles and Practice of Constraint Programming CP 2004*, pages 648–662, 2004.

[53] Helmut Simonis. Models for global constraint applications. *Constraints*, 12:63–92, March 2007.

[54] Barbara M. Smith. Modelling. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 11. Elsevier, 2006.

[55] M. Trick. A dynamic programming approach for consistency and propagation for knapsack constraints, 2001.

[56] Carillon J.-P. Van Hentenryck P. Generality versus specificity: an experience with ai and or techniques. *In Proceedings of AAAI-88*, pages 660–664, August 1988.

[57] Willem Jan van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming: Flow-based soft global constraints. *J. Heuristics*, 12(4-5):347–373, 2006.

[58] Alessandro Zanarini, Michela Milano, and Gilles Pesant. Improved algorithm for the soft global cardinality constraint. In *CPAIOR*, pages 288–299, 2006.