The Weighted Average Constraint

Alessio Bonfietti and Michele Lombardi

DEIS, University of Bologna
{alessio.bonfietti,michele.lombardi2}@unibo.it

Abstract. Weighted average expressions frequently appear in the context of allocation problems with balancing based constraints. In combinatorial optimization they are typically avoided by exploiting problems specificities or by operating on the search process. This approach fails to apply when the weights are decision variables and when the average value is part of a more complex expression. In this paper, we introduce a novel `average` constraint to provide a convenient model and efficient propagation for weighted average expressions appearing in a combinatorial model. This result is especially useful for Empirical Models extracted via Machine Learning (see [2]), which frequently count average expressions among their inputs. We provide basic and incremental filtering algorithms. The approach is tested on classical benchmarks from the OR literature and on a workload dispatching problem featuring an Empirical Model. In our experimentation the novel constraint, in particular with incremental filtering, proved to be even more efficient than traditional techniques to tackle weighted average expressions.

1 Introduction

A weighted average expression is a function in the form:

$$avg(\overline{v}, \overline{w}) = \frac{\sum_{i=0}^{n-1} v_i \cdot w_i}{\sum_{i=0}^{n-1} w_i} \tag{1}$$

where v_i and w_i respectively are the *value* and the *weight* of each term in the average and the notation $\overline{v}$, $\overline{w}$ refers to the whole vector of values/weights and n is the number of terms. We assume all weights are non-negative. Moreover, we assume $avg(\overline{v}, \overline{w}) = 0$ if all weights are 0. When appearing in a combinatorial model, both the term values and weights of the expression may be variables, say `V_i` and `W_i`. Weighted average expressions are found in a number of application settings, such as balancing the workload of assembly lines, fair scheduling of work shifts, uniform distribution of travel distances in routing problems.

Average expressions also frequently appear as inputs of Empirical Models. Those are Machine Learning models (such as Neural Networks, Support Vector Machines or Decision Trees), obtained via training (over a real world system or a simulator) and encoded using some combinatorial optimization technology. Empirical Models have been recently proposed as a mean to enable decision making over complex systems [2]. The approach has in principle broad applicability, but

the extracted models often feature average values as part of their input (e.g. [1]). Hence the ability to deal with weighted average expressions is essential for the applicability of such methods.

Research efforts so far have been focused on balancing issues (see the works on the `spread` [7, 9] and `deviation` constraint [10, 11]). Balancing involves keeping the average close to a target value and avoiding outliers. A general formulation for a `balancing` constraint has been given in terms of p-norm in [10] and [8], for the case where all weights are constant and equal to 1. In this situation the average expression has fixed denominator and is therefore linear and easy to handle. Hence, the `spread` and `deviation` constraint focus on providing more powerful filtering by respectively taking into account restrictions on the variance and on the total deviation.

There are however cases where non-constant weights are definitely handy. Assume we want to balance the average per-depot travel time in a Vehicle Routing Problem with n vehicles and multiple depots. The assignment of a vehicle i to depot j can be conveniently modeled as a $\{0, 1\}$ variable $\mathtt{W_{i,j}}$, leading to:

$$\mathtt{Y_j} = \frac{\sum_{i=0}^{n-1} \mathtt{V_i} \cdot \mathtt{W_i}}{\sum_{i=0}^{n-1} \mathtt{W_i}} \qquad \text{or} \qquad \sum_{i=0}^{n-1} \mathtt{V_i} \cdot \mathtt{W_i} = \mathtt{Y_j} \cdot \sum_{i=0}^{n-1} \mathtt{W_i} \tag{2}$$

where $\mathtt{V_i}$ is the travel distance for vehicle i and $\mathtt{Y_j}$ is the average travel distance for depot j. The left-most formulation requires the denominator to be strictly positive. Unless the number of vehicles per depot is a-priori known, the sum of $\mathtt{W_{i,j}}$ variables is non-fixed, the expression non linear and the resulting propagation weak, thus making non-uniform weights very difficult to deal with. Many allocation problems with balancing restrictions raise similar issues.

In this paper, we introduce a novel `average` global constraint to provide a convenient model and effective filtering for average expressions with non-uniform weights. Unlike `spread` or `deviation`, we do not explicitly try to limit outliers, since filtering for the average value is sufficiently challenging by itself. The paper is structured as follows: in Section 2 we introduce the `average` constraint and provide filtering algorithms with the assumption that all term *values* are fixed. Several improvements to the basic algorithms are discussed in Section 3. In Section 4 the efficiency and effectiveness of our filtering is investigated; moreover, our approach is compared to an alternative method requiring no ad-hoc constraint and to the decomposed constraint from Equation (2). Finally, we show how to deal with variable term values in Section 5 and we provide concluding remarks in Section 6.

2 The Average Constraint

The `average` global constraint provides a convenient model and effective propagation for weighted average expressions. We will initially assume that the term *weights* are decision variables, while the *values* are constant: this setting captures

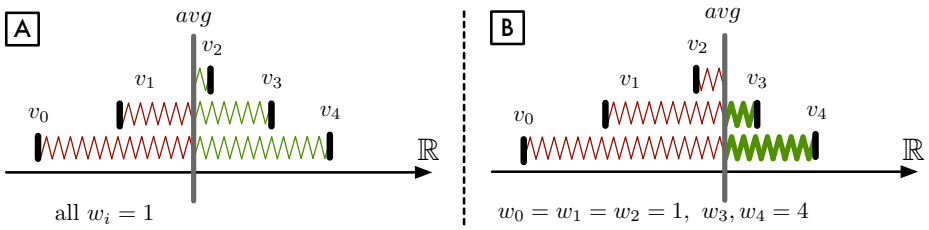

Fig. 1. A: Initial weight configuration ($w_i \in \{1..4\} \forall i$); B: Upper bound configuration

most of the difficulties found by CP methods with weighted average expressions. In the considered case, the constraint signature is:

$$\texttt{average}(\overline{v}_i, \overline{\texttt{W}}, \texttt{Y}) \tag{3}$$

where $\overline{v}$ is a vector containing the integer value of each term and $\overline{\texttt{W}}$ is a vector of integer variables representing the weights. Weights must be non-negative, i.e. for each variable we have $\min(\texttt{W}_\texttt{i}) \geq 0$ (where $\min(\texttt{W}_\texttt{i})$ is the minimum value in the domain of $\texttt{W}_\texttt{i}$). The integer variable $\texttt{Y}$ stores the weighted average value. The constraint maintains bound consistency on the expression:

$$\texttt{Y} = round\left(\frac{\sum_{i=0}^{n-1} v_i \cdot \texttt{W}_\texttt{i}}{\sum_{i=0}^{n-1} \texttt{W}_\texttt{i}}\right) \tag{4}$$

where n is the number of terms and $round(\cdot)$ is the usual rounding operator. The use of integer values and weights is a design choice, motivated by the lack of support for real-valued variables in many current CP solvers. In the following, we provide filtering algorithms for the constraint as from signature (3), while the case of variable term values is considered in Section 5.

2.1 Computing Bounds on Variable Y

We focus on computing an upper bound for $\texttt{Y}$ (a lower bound can be obtained analogously). On this purpose, observe that all value necessarily appear in the expression at least with minimum weight. Hence, the formula:

$$\frac{\sum_{i=0}^{n-1} v_i \cdot \min(\texttt{W}_\texttt{i})}{\sum_{i=0}^{n-1} \min(\texttt{W}_\texttt{i})} \tag{5}$$

provides a first estimate of the weighted average, which can be increased or lowered by including some terms with non-minimal weight. It is convenient to think of the expression as a system of physical springs with fixed anchored points, all pulling a single bar (see Figure 1). Anchor points represents the term values and the spring strengths are the weights. The initial estimate corresponds to a configuration where each spring has minimum strength (see Figure 1A). Intuitively, increasing the spring strength of one of the right-most anchor points will increase the average value. Formally, we have:

Statement 1 *Let $\overline{w}^*$ be a fixed weight assignment. Then, increasing a weight w_j^* by an amount $\delta > 0$ leads to an increased weighted average value iff:*

$$v_j > \frac{\sum_{i=0}^{n-1} v_i \cdot w_i^*}{\sum_{i=0}^{n-1} w_i^*} \tag{6}$$

where we assume the right most part of the inequality is 0 if all weights are 0.

The statement can be proved by algebraical manipulation of Equation (1). Note that the effect of increasing a weight does not depend on δ: if increasing w_i makes the current estimate larger, then the maximum increment leads to the largest increase. Hence we can improve the initial estimate by repeatedly choosing a v_i that satisfies the condition from Statement 1 and then maximizing w_i.

The process may however stop at a local maximum if the v_i are not properly chosen. For example, in Figure 1A we may decide to maximize v_2, since it is higher than the current average estimate. However, this decision makes it impossible to reach the configuration from Figure 1B, corresponding to the actual maximum of the average expression. Luckily, there exists a sequence of weight increments guaranteed to stop at a *global* maximum.

Statement 2 *The maximum value of the weighted average can be obtained starting from Expression* (5)*, by repeatedly maximizing the weight of the term j with maximum v_j, as long as the condition from Theorem 1 is satisfied.*

Assume we have $v_j \geq v_i$: then one can check that maximizing both w_i and w_j cannot lead to a lower average than maximizing w_j alone. Therefore, considering w_j before w_i is safer than the opposite order. By induction we obtain that the term with highest value should alway be considered first. In the example from Figure 1, this kind of reasoning leads to the configuration in Figure 1B, corresponding to the actual maximum.

A filtering rule to propagate changes of $\mathtt{W_i}$ domains to $\mathtt{Y}$ is given by the pseudo-code in Algorithm 1. The algorithm computes the numerator and denominator of Expression 5 (line 2), which are then updated by repeatedly maximizing weights (lines 4-6) as long as the condition from Theorem 1 holds (line 3). The rounding operation at line 6 is needed since $\mathtt{Y}$ is an integer variable. The worst case complexity at search time is $O(n)$, since the initial sorting operation (line 1) can be done once for all at model loading time.

Algorithm 1 Filtering on max($\mathtt{Y}$)

Require: a change in the bound of some $\overline{\mathtt{W_i}}$ domain
1: sort terms by decreasing v_i
2: compute $v_{ub} = \sum_{i=0}^{n-1} v_i \cdot \min(\mathtt{W_i})$ and $w_{ub} = \sum_{i=0}^{n-1} \min(\mathtt{W_i})$
3: **for** $i = 0$ to $n-1$, as long as $v_i > {}^{v_{ub}}/_{w_{ub}}$ **do**
4: $\quad \delta = \max(\overline{\mathtt{W_i}}) - \min(\overline{\mathtt{W_i}})$
5: $\quad v_{ub} = v_{ub} + v_i \cdot \delta$
6: $\quad w_{ub} = w_{ub} + \delta$
7: $\max(\mathtt{Y}) \leftarrow round({}^{v_{ub}}/_{w_{ub}})$

2.2 Computing Bounds on Variables $\mathtt{W_i}$

Lower and upper bounds on the $\mathtt{W_i}$ variables can be deduced based on the domain of $\mathtt{Y}$. In this section we focus on the deductions performed from the domain max, i.e. $\max(\mathtt{Y})$ (filtering with the minimum is analogous). In first place, note that if $\max(\mathtt{Y})$ is equal to its maximum computed as from Section 2.1, no pruning on the $\mathtt{W_i}$ domains is possible. If this is not the case, we may be able to do some filtering. In particular we have:

$$\frac{\sum_{i=0}^{n-1} v_i \cdot \mathtt{W_i}}{\sum_{i=0}^{n-1} \mathtt{W_i}} \leq \max(\mathtt{Y}) \tag{7}$$

Let us assume initially that there is at least a non-zero weight. Then we get:

$$\sum_{i=0}^{n-1} \mathtt{W_i} \cdot rv_i(\mathtt{Y}) \leq 0 \qquad \text{with} \quad rv_i(\mathtt{Y}) = v_i - \max(\mathtt{Y}) \tag{8}$$

where we refer to $rv_i(\mathtt{Y})$ as *reduced value* or term i. The *maximum slack* is obtained by minimizing $\mathtt{W_i}$ if the reduced value is positive and maximizing the variable if the reduced value is negative. Formally:

$$ms(\overline{\mathtt{W}}, \mathtt{Y}) = \sum_{i=0}^{n-1} rv_i(\mathtt{Y}) \cdot \begin{cases} \min(\mathtt{W_i}) & \text{if } rv_i(\mathtt{Y}) > 0 \\ \max(\mathtt{W_i}) & \text{otherwise} \end{cases} \tag{9}$$

By keeping a single variable free, we get:

$$rv_j(\mathtt{Y}) \cdot \mathtt{W_j} \leq -\sum_{i=0, i\neq j}^{n-1} rv_i(\mathtt{Y}) \cdot \begin{cases} \min(\mathtt{W_i}) & \text{if } rv_i(\mathtt{Y}) > 0 \\ \max(\mathtt{W_i}) & \text{otherwise} \end{cases} \tag{10}$$

The right-most member of the Equation (10) is constant and referred to as θ_j in the followings. Then, depending on the sign of the reduced value we get either a lower or an upper bound:

$$\mathtt{W_j} \leq {}^{\theta_j}/_{rv_j(\mathtt{Y})} \text{ if } rv_j(\mathtt{Y}) > 0, \qquad \mathtt{W_j} \geq {}^{\theta_j}/_{rv_j(\mathtt{Y})} \text{ if } rv_j(\mathtt{Y}) < 0 \tag{11}$$

and no propagation in case $rv_j(\mathtt{Y}) = 0$. If all weights in θ_j are 0, then it may be the case that all weights are zero and Equation (7) turns into:

$$\frac{v_j \cdot \mathtt{W_j}}{\mathtt{W_j}} \leq \max(\mathtt{Y}) \tag{12}$$

From which we can deduce that: 1) If $\max(\mathtt{Y}) < 0$, then $\mathtt{W_i}$ must be non-zero since a weighted average is 0 if all weights are 0. 2) If $v_j > \max(\mathtt{Y})$, then $\mathtt{W_i}$ cannot be higher than 0. Note that if both the rules are triggered we have a fail.

The reasoning leads to Algorithm 2. Line 1 performs a de-round operation (needed since all variables are integer). As a side effect, the expression $v_j - y_{max}$

Algorithm 2 Filtering from max(Y) to $\mathtt{W_i}$ variables

Require: a change in max(Y), or in the domain bounds of some $\mathtt{W_i}$

1: Let y_{max} be the largest value v such that $round(v) = \max(\mathtt{Y})$. If y_{max} is greater or equal to the maximum from Section 2.1, then immediately return.
2: Compute the maximum slack $ms(\overline{\mathtt{W}}, \mathtt{Y})$ according to Equation (10), using y_{max} instead of max(Y). Additionally, let w_{ms} be the corresponding sum of weights.
3: **for** $j = 0$ to $n - 1$ **do**
4: **if** $v_j - y_{max} > 0$ **then**
5: $\theta_j = -(ms(\overline{\mathtt{W}}, \mathtt{Y}) - (v_j - y_{max}) \cdot \min(\mathtt{W_j}))$
6: $w_{\theta_j} = w_{ms} - \min(\mathtt{W_j})$
7: **else**
8: $\theta_j = -(ms(\overline{\mathtt{W}}, \mathtt{Y}) - (v_j - y_{max}) \cdot \max(\mathtt{W_j}))$
9: $w_{\theta_j} = w_{ms} - \max(\mathtt{W_j})$
10: **if** $w_{\theta_i} = 0$ **then**
11: **if** $y_{max} < 0$ **then** $\min(\mathtt{W_j}) \leftarrow 1$
12: **if** $v_j > y_{max}$ **then** $\max(\mathtt{W_j}) \leftarrow 0$
13: **else**
14: **if** $v_j - y_{max} > 0$ **then** $\max(\mathtt{W_j}) \leftarrow \lfloor \theta_j / (v_j - y_{max}) \rfloor$
15: **if** $v_j - y_{max} < 0$ **then** $\min(\mathtt{W_j}) \leftarrow \lceil \theta_j / (v_j - y_{max}) \rceil$

is used in the algorithm in place of the reduced value. The maximum slack is computed at line 2. Then, for each term we compute the θ_i value at lines 5 and 8, by subtracting the contribution of term j from $ms(\overline{\mathtt{W}}, \mathtt{Y})$; we also compute the sum w_{θ_i} of weights in θ_i (lines 6 and 9). If $w_{\theta_i} = 0$ we apply the filtering rules from Equation (12) at lines 11-12, otherwise, we use the regular filtering rules from Equation (11) at lines 14-15. The floor/ceil operations at lines 14-15 are needed since all variables are integer. The overall time complexity is $O(n)$.

3 Optimizations

Incremental Upper Bound and Maximum Slack Computation: It is possible to improve the efficiency of the filtering algorithm via incremental computation of the upper bound from Section 2.1 (let this be $ub(\overline{\mathtt{W}})$) and the maximum slack $ms(\overline{\mathtt{W}}, \mathtt{Y})$ from Section 2.2. The approach requires all terms to be sorted by decreasing v_j (this can be done once for all at model loading time).

Specifically, let v_{ub}, w_{ub} be the current numerator and denominator of $ub(\overline{\mathtt{W}})$. Moreover, let j_{ub} be the index of the last term whose weight was maximized by Algorithm 1. In case the domain minimum/maximum of some $\mathtt{W_j}$ variable changes, new values for v_{ub}, w_{ub} and j_{ub} can be computed in two steps. In first place, we have to perform the updates:

$$v_{ub} = v_{ub} + v_j \cdot \begin{cases} \max(\mathtt{W_j})_{new} - \max(\mathtt{W_j})_{old} & \text{if } j \leq j_{ub} \\ \min(\mathtt{W_j})_{new} - \min(\mathtt{W_j})_{old} & \text{otherwise} \end{cases} \tag{13}$$

Algorithm 3 Update j_{ub}

while $v_{j_{ub}} >^{v_{ub}} /_{w_{ub}}$ **do**
  $\delta = \max(\overline{\mathtt{W}}_{\mathtt{j_{ub}}}) - \min(\overline{\mathtt{W}}_{\mathtt{j_{ub}}})$
  $v_{ub} = v_{ub} + v_{j_{ub}} \cdot \delta$
  $w_{ub} = w_{ub} + \delta$
  $j_{ub} = j_{ub} + 1$

Algorithm 4 Update j_{ms}

while $v_{j_{ms}} > y_{max}$ **do**
  $\delta = \min(\overline{\mathtt{W}}_{\mathtt{j_{ms}}}) - \max(\overline{\mathtt{W}}_{\mathtt{j_{ms}}})$
  $v_{ms} = v_{ms} + v_{j_{ms}} \cdot \delta$
  $w_{ms} = w_{ms} + \delta$
  $j_{ms} = j_{ms} + 1$

$$w_{ub} = w_{ub} + \begin{cases} \max(\mathtt{W_j})_{new} - \max(\mathtt{W_j})_{old} & \text{if } j \leq j_{ub} \\ \min(\mathtt{W_j})_{new} - \min(\mathtt{W_j})_{old} & \text{otherwise} \end{cases} \tag{14}$$

Due to the resulting changes of $ub(\overline{\mathtt{W}})$, it may become necessary to maximize *one or more* terms previously not satisfying the condition from Statement 1. This can be done by means of Algorithm 3, which is analogous to lines 3-6 of Algorithm 1, but makes use of the stored j_{ub}. Note that in case the domain of more than a single $\mathtt{W_i}$ variable changes, then *all* the updates from Equation (13) and (14) should be performed before running Algorithm 3. This behavior is easy to implement if the solver provides a delayed constraint queue.

The incremental computation has constant space complexity, since only three values need to be stored (v_{ub}, w_{ub} and j_{ub}). Updating v_{ub} and w_{ub} using Equation (13) and (14) has constant time complexity and the number of possible updates from the root to a leaf of the search tree is $O(n \times \text{max domain size})$. In the important case of $\{0, 1\}$ weights, no more than $2 \times n$ updates can occur. Since the j_{ub} index cannot advance more than n times, the overall complexity is $O(n)$ along a full branch of the search tree.

Incremental computation for the maximum slack from Section 2.2 can be achieved in a similar fashion. In first place, observe that Equation (9) can be decomposed as $ms(\overline{\mathtt{W}}, \mathtt{Y}) = v_{ms} - \max(\mathtt{Y}) \cdot w_{ms}$, where the dependency on $\overline{\mathtt{W}}$ is omitted for sake of simplicity and:

$$v_{ms} = \sum_{i=0}^{n-1} v_i \cdot \begin{cases} \min(\mathtt{W_i}) & \text{if if } rv_i(\mathtt{Y}) > 0 \\ \max(\mathtt{W_i}) & \text{otherwise} \end{cases} \tag{15}$$

$$w_{ms} = \sum_{i=0}^{n-1} \begin{cases} \min(\mathtt{W_i}) & \text{if if } rv_i(\mathtt{Y}) > 0 \\ \max(\mathtt{W_i}) & \text{otherwise} \end{cases} \tag{16}$$

The values v_{ms} and w_{ms} can be updated incrementally similarly to v_{ub} and w_{ub}. We then need to update the index j_{ms} of the last minimized element in the maximum slack configuration, which can be done by means of Algorithm 4.

Reducing the Complexity of Term Weight Filtering: The techniques described in the previous section make the computation of $ub(\overline{\mathtt{W}})$ and $ms(\overline{\mathtt{W}}, \mathtt{Y})$ very efficient. However, pruning the $\mathtt{W_i}$ variables as described in Algorithm 2 still

takes linear time and acts a bottleneck for the propagator complexity. It may be possible to achieve higher efficiency by exploiting ideas in [4]: here however we apply a simpler technique, described in the case of positive reduced values (i.e. terms j with $j < j_{ms}$ from Section 3). The corresponding θ_j value is:

$$\theta_j = -ms(\overline{\mathtt{W}}, \mathtt{Y}) + rv_j(\mathtt{Y}) \cdot \min(\mathtt{W_j}) \tag{17}$$

with $ms(\overline{\mathtt{W}}, \mathtt{Y}) \leq 0$. The upper bound computed by Algorithm 2 on $\mathtt{W_j}$ is ${}^{\theta_j}/_{rv_j(\mathtt{Y})}$ and can be decomposed as:

$$\frac{\theta_j}{rv_j(\mathtt{Y})} = -\frac{ms(\overline{\mathtt{W}}, \mathtt{Y})}{rv_j(\mathtt{Y})} + \min(\mathtt{W_j}) \tag{18}$$

We are interested in finding a sufficient condition to *avoid the bound computation for some* $\mathtt{W_j}$ *variables*. Now, observe that for every $k > j$, we have:

$$-\frac{ms(\overline{\mathtt{W}}, \mathtt{Y})}{rv_j(\mathtt{Y})} + \min_i\{\min(\mathtt{W_i})\} \leq -\frac{ms(\overline{\mathtt{W}}, \mathtt{Y})}{rv_k(\mathtt{Y})} - \min(\mathtt{W_k}) \tag{19}$$

since $v_k \leq v_j$ (due to the initial sorting step). In other words, the left-most quantity in Equation (19) bounds the upper-bound value of all terms k with $k > j$. Therefore, if at line 14 of Algorithm 1 we realize that:

$$-\frac{ms(\overline{\mathtt{W}}, \mathtt{Y})}{rv_j(\mathtt{Y})} + \min_i\{\min(\mathtt{W_i})\} \geq \max_i\{\max(\mathtt{W_i})\} \tag{20}$$

we can immediately stop filtering all $j < j_{ms}$. In order to simplify the computation, we statically compute the minimum and maximum among all domains (i.e. $\min_i\{\min(\mathtt{W_i})\}$ and $\max_i\{\max(\mathtt{W_i})\}$) at model loading time. As a result, checking this stop condition has constant time complexity.

Avoiding Useless Activations of Term Weight Filtering: Finally, we can rely on the stored j_{ms} value to reduce the number of calls to Algorithm 2. On this purpose, observe that the bound computation at lines 14-15 is based on the maximum slack. Note that in the $ms(\overline{\mathtt{W}}, \mathtt{Y})$ expression (see Equation (9)) the terms with positive $rv_($Y$)$ appear with weight minimum, while the terms with non-positive reduced value appear with maximum weight.

As a consequence, changes in $\max(\mathtt{W_i})$ with $i > j_{ms}$ or changes in $\min(\mathtt{W_i})$ with $i \leq j_{ms}$ do not change the value of the maximum slack and do no require to run Algorithm 1. Hence, immediately after the execution of Algorithm 4 we can check which $\mathtt{W_i}$ boundaries have changed after the last update, and avoid performing additional filtering if this is not needed. The same does not hold for changes of $\max(\mathtt{Y})$, that always require to re-execute Algorithm 2.

4 Experimental Results

The described approach was implemented in Google OR-tools [6] and tested on two different benchmarks. In first place, we tackled a variant of the Single

Source Capacitated Facility Location Problem (SSCFLP) [12] with a balancing objective. The problem structure was suitable for different solution approaches, respectively: a model with our average constraint, a model using the decomposed formulation from Equation 2 and a method that employs an alternative search strategy to avoid the use of an explicit average constraint. We performed a comparison to show that our method has is competitive with alternative approaches.

In a second experimentation we use the average constraint to provide input to an Empirical Model, namely a Neural Network capturing the thermal behavior of a multi-core CPU. The network is employed within a large-size workload dispatching problem with a maximal efficiency objective. On this problem, we compare only our constraint and the decomposed formulations, since the the third method cannot be applied. Moreover, we take advantage of the large scale of the instances to compare the performance of our incremental filtering algorithm w.r.t its non-incremental version.

4.1 Balancing the Travel Cost in the SSCFLP

Problem Formulation The Capacitated Facility Location Problems represent a major class of location problems, well known in the OR-literature [12]. The SSCFLP deals with opening facilities j at a set of candidate locations, to serve at minimum cost a set of customers i distributed in a discrete space. Different facility sites have distinct (fixed) opening costs f_j. Each customer must be supplied from a single facility and has a fixed demand d_i. The sum of the demands for a single facility should not exceed its capacity s_j. Assigning a customer to a facility incurs a transportation charge $c_{i,j}$. The usual goal of the problem is to decide which facilities to open and to assign customers so as to minimize the overall cost. Here, we consider instead a variant where the objective is to minimize the worst-case average transportation cost per facility. Additionally, we require that the total opening cost does not exceed the given threshold. We adopt the following model (with n customers and m facilities):

$$\min \mathtt{Z} = \max_{j=0..m-1} \mathtt{A_j} \tag{21}$$

$$\text{s.t.: } \mathtt{X_{i,j}} = 1 \Leftrightarrow (\mathtt{W_i} = j) \quad \forall j = 0..m-1,\ i = 0..n-1 \tag{22}$$

$$0 \leq \mathtt{X_{i,j}} \leq \mathtt{Y_j} \leq 1 \quad \forall j = 0..m-1,\ i = 0..n-1 \tag{23}$$

$$\sum_{i=0}^{n-1} d_i \cdot \mathtt{X_{i,j}} \leq s_j \cdot \mathtt{Y_j} \quad \forall j = 0..m-1 \tag{24}$$

$$\sum_{j=0}^{m-1} f_j \cdot \mathtt{Y_j} \leq F_{max} \tag{25}$$

$$\mathtt{average}(\overline{c}_j, \overline{\mathtt{X}}_\mathtt{j}, \mathtt{A_j}) \quad \forall j = 0..m-1 \tag{26}$$

The decision variable are:

$\mathtt{W_i} \in \{0..m-1\}$ $\quad$ $\mathtt{W_i} = j$ iff customer i is assigned to facility j

$\mathtt{X_{i,j}} \in \{0, 1\}$ $\quad$ $\mathtt{X_{i,j}} = 1$ iff customer i is assigned to facility j

$\mathtt{Y_j} \in \{0, 1\}$	$\mathtt{Y_j} = 1$ iff facility j is open
$\mathtt{A_j} \in \{0..\inf\}$	average transportation cost for each facility j

where $\overline{\mathtt{X}}_\mathtt{j}$ and $\overline{c}_j$ are the subsets of $\mathtt{X_{i,j}}$ and $c_{i,j}$ that refer to facility j. F_{max} is the opening cost limit which was obtained in a preliminary experimentation.

Solution Methods We solved the problem with three different approaches:

1. The model reported in the previous paragraph (referred to as AVG), using the incremental filtering for the average constraint.
2. A model (referred to as DEC) where the `average` constraint is replaced by its decomposed formulation from Equation 2.
3. A method (referred to as SWEEP) where the optimization problem is slit into a sequence of satisfaction problems. Each subproblem is obtained by 1) removing the $\mathtt{A_j}$ variables; 2) removing the cost function; 3) replacing the average constraints with the following linear inequalities:

$$\sum_{i=0}^{n-1} c_{i,j} \cdot \mathtt{X_{i,j}} < z_k^* \cdot \sum_{i=0}^{n-1} \mathtt{X}_{i,j} \qquad \forall j = 0..m-1 \tag{27}$$

   where z_k^* for the first subproblem is a safe upper bound and z_k^* for the k-th problem in the sequence is equal to the cost of the solution the $k-1$-problem.

The AVG and SWEEP approaches are equivalent in term of propagation, but SWEEP has to explore a larger space due to the decoupling. The DEC approach results instead in weaker (but more efficient) propagation. In all cases, we use tree search with random variable and value selection, performing restarts when a fixed fail limit is reached (20000). This approach was chosen since the considered instance size makes a heuristic approach more viable and because restarts make the performance of all methods less erratic. We run the experiments on a subset of the benchmarks by Beasley in the OR-lib [3], in particular the *cap*6X, *cap*7X, *cap*9X, *cap*10X, *cap*12X, and *cap*13X groups, having different configuration number of facilities, customers and costs. All the experiments are run with a time limit of 60 seconds on a 2.8 GHz Intel i7 and repeated 10 times per instance.

Results Table 1 shows the average solution gaps over time (%) between the AVG, the DEC and the SWEEP approach[1]. In particular, we measured the gap between the best solution found by both approaches at some pre-defined sampling points (one per column in the table).

Note that the performance of the AVG approach is considerably better than DEC, in spite of the fact that the heavy use of restarts favors approaches based on weaker (and faster) propagation. This is likely due to the fact that for this

[1] The gap for a single instance is $\frac{Z_{\text{SWEEP}}}{Z_{\text{AVG}}} - 1$, resp. $\frac{Z_{\text{DEC}}}{Z_{\text{AVG}}} - 1$. The reported value is the average gap for all instances in the group.

		Solution Gap (%)					
Instance set	vs.	1s	3s	5s	10s	30s	60s
cap6X	SWEEP	18.9 %	16.5 %	18.55 %	19.7 %	11.45 %	11.6 %
	DEC	37.2 %	47.5 %	49.225 %	17.8 %	6.725 %	7.9 %
cap7X	SWEEP	7 %	7.7 %	7.7 %	7.8 %	10.3 %	9.7 %
	DEC.	22.275 %	14.1 %	11.05 %	8.6 %	10.7 %	11.075 %
cap9X	SWEEP	25.8 %	31.025 %	30.775 %	28.525 %	23.95 %	22.075 %
	DEC.	62.1 %	66.125 %	65.025 %	36.75 %	12 %	12.75 %
cap10X	SWEEP	4.9 %	7.3 %	5.8 %	5.45 %	5.175 %	4.6%
	DEC.	8.75 %	8.6 %	6.7 %	5.775 %	6 %	5.9 %
cap12X	SWEEP	29 %	29 %	35.55 %	28.575 %	23.2 %	20 %
	DEC.	15.1 %	23.275 %	11.625 %	12.225 %	14.9 %	12.5 %
cap13X	SWEEP	4.95 %	4.475 %	4.075 %	4.95 %	4.4 %	3.625 %
	DEC.	10.9 %	6.95 %	7.15 %	6.7 %	5.7 %	4.6 %

Table 1. Average Solution Gap (%) for each benchmarks subset.

problem the weighted average expressions appear straight in the cost function. More surprisingly, AVG works better than SWEEP, which is equivalent in term of propagation effectiveness and faster in terms of propagation efficiency: apparently, the overhead for the repeated replacement of constraints (27) has a negative impact on the solver performance, which overcomes that of the more complex filtering of the average constraint.

4.2 Empirical Models and Incremental Filtering

Problem Formulation We consider a workload dispatching problem for a 48-core CPU by Intel [5]. The platform features some temperature control policies, which may slow down the execution to avoid overheating. We want to map a set of n tasks i on the platform cores j so as to incur the smallest possible slowdown due to the thermal controllers. As in many realistic settings, we assume tasks are preemptive and have unknown duration. Each task must be assigned to a platform core and is characterized by a parameter named CPI (Clock Per Instruction): the smaller the CPI_i value, the more computation intensive the task and the higher the generated heat.

The thermal behavior of a core is also affected by the workload of neighboring cores. The resulting correlation between workload and platform efficiency is strongly non-linear (due to the thermal controllers and thermal effects) and very tough to model via conventional means. In [1] we proposed an approach were such a complex function is captured via a Neural Network, trained over a system simulator. Here, we consider a slightly simplified version of the problem, corresponding to the following model (with n tasks and $m = 48$ cores):

$$\max \mathtt{Z} = \min_{j=0..m-1} \mathtt{E_j} \tag{28}$$

$$\text{s.t.: } \mathtt{E_j} = \mathtt{ANN_j}(temp, \{\mathtt{A_k} \mid \text{core } k \text{ neighbor of } j\}) \quad \forall j = 0..m-1 \tag{29}$$

$$\mathtt{average}(\overline{CPI}, \overline{\mathtt{X}}_\mathtt{j}, \mathtt{A_j}) \qquad \forall j = 0..m-1 \tag{30}$$

$$\sum_{i=0}^{n-1} \mathtt{X_{i,j}} \geq 1 \qquad \forall j = 0..m-1 \tag{31}$$

$$\mathtt{X_{i,j}} = 1 \Leftrightarrow (\mathtt{W_i} = j) \quad \forall i = 0..n-1, j = 0..m-1 \tag{32}$$

The decision variable are:

$\mathtt{W_i} \in \{0..m-1\}$	$\mathtt{W_i} = j$ iff task i is assigned to core j
$\mathtt{X_{i,j}} \in \{0,1\}$	$\mathtt{X_{i,j}} = 1$ iff task i is assigned to core j
$\mathtt{A_j} \in \{0..\infty\}$	average CPI of tasks assigned to core j

The $\mathtt{E_j}$ variables represent the efficiency of core j. The notation $\mathtt{ANN_j}(\cdot)$ stands for the set of Neuron Constraints [2] corresponding the Neural Network we trained for core j. In principle $\mathtt{E_j}$ should take real values in $]0,1]$, in practice we use a fixed precision integer representation.

Solution Method We solved the problem using tree search with random variable and value selection, performing restarts when a fixed fail limit is reached (6000 fails). We compared our approach with a model making use of the decomposed average formulation, referred to as DEC. Moreover, we used this problem to investigate the speed-up of the incremental `average` filtering (referred to as INC) over the non incremental one (referred to as NO INC).

For the experimentation we used the benchmarks from [1], consisting of 3 groups, each counting 2 worksets of 10 instances[2]. Groups differ by the number of tasks (120, 240, or 480). Worksets within each group correspond to different task compositions: all instances in Workset I contain 75% low-CPI and 25% high-CPI tasks, while the mix is instead 85%/15% for Workset II. Each instance was solved 10 times with each method, on an 2.8 GHz Intel i7.

Results Table 2 shows the results of the experimentation. The benchmarks are divided according to the size of the instances (120, 240, and 480 tasks). For each workset (I or II), each column in the table refers to a different sampling time and shows the solution quality gap between AVG and NO INC, and between AVG and DEC (the gap is computed analogously to the previous experimentation). The table also reports the search speed, measured in terms of average number of branches per second. As expected, the incremental approach is considerably better then the non incremental one. Interestingly, AVG has approximatively the same branching speed on the 240 and 480 task instances, suggesting that the propagator is processing a roughly constant number of weight variables (thanks to the stopping condition from Section 3). Interestingly, AVG often works better than DEC: the presence of a the network of Neuron Constraints makes in principle

[2] The benchmarks are available for download at `http://ai.unibo.it/data/TAWD`

			Average Solution Efficiency (%)					
Inst. size	Workset	Algorithm	1 sec	3 sec	5 sec	10 sec	30 sec	60 sec
120	Set I	NO INC	28.66 %	9.03 %	12.68 %	8.08 %	4.62 %	4.21 %
		DEC	0.74 %	-5.71 %	-5.64 %	-4.67 %	-3.86 %	-2.25 %
	Set II	NO INC	18.14 %	11.06 %	11.01 %	4.41 %	6.07 %	3.83 %
		DEC	1.39 %	-1.07 %	2.78 %	3.07 %	4.11 %	4.82 %
Average speed: (branch/secs)			AVG	4043.33	NO INC	2571.88	DEC	9332.96
240	Set I	NO INC	7.58 %	12.14 %	9.19 %	5.85 %	4.23 %	3.97 %
		DEC	0.02 %	1.98 %	4.47 %	2.56 %	1.65 %	2.27 %
	Set II	NO INC	5.79 %	10.14 %	8.62 %	6.11 %	4.18 %	3.14 %
		DEC	3.98 %	1.47 %	2.49 %	2.78 %	3.36 %	2.35 %
Average speed: (branch/secs)			AVG	1845.00	NO INC	949.31	DEC	2904.68
480	Set I	NO INC	*inf*	24.01 %	25.38 %	12.65 %	4.06 %	2.51 %
		DEC	7.82 %	1.41 %	2.73 %	1.72 %	0.86 %	0.63 %
	Set II	NO INC	*inf*	22.79 %	26.10 %	16.24 %	8.65 %	5.71 %
		DEC	6.58 %	2.60 %	3.65 %	5.13 %	4.26 %	3.61 %
Average speed: (branch/secs)			AVG	2119.86	NO INC	635.75	DEC	2797.21

Table 2. Incremental Speed-up: solution efficiency and gaps

the propagation on the average far less critical, and yet employing the global `average` constraint seems to definitely pay off.

5 Dealing with Variable Term Values

It is possible to extend the average constraint to deal with variable term values. The resulting signature is:

$$\texttt{average}(\overline{\texttt{V}}, \overline{\texttt{W}}, \texttt{Y}) \tag{33}$$

where $\overline{\texttt{V}}$ is a vector of integer variables. Variable term values allow to tackle a broader class of problems, such as the routing example mentioned in Section 1. Filtering for the new constraint signature requires a few modifications to the procedures described in Section 2.1 and 2.2.

Computing Bounds on Variable `Y`: Obtaining an upper bound for `Y` is relatively easy, due to the following theorem:

Theorem 1. *Let $\overline{v}^*$, $\overline{w}^*$ be fixed assignments for $\overline{\texttt{V}}$ and $\overline{\texttt{W}}$. Then, increasing a value v_j^* by an amount $\delta > 0$ always leads to an increased weighted average value.*

Proof. Directly follows from:

$$\frac{\sum_{i=0}^{n-1} v_i^* \cdot w_i^* + \delta \cdot w_j^*}{\sum_{i=0}^{n-1} w_i^*} \geq \frac{\sum_{i=0}^{n-1} v_i^* \cdot w_i^*}{\sum_{i=0}^{n-1} w_i^*} \tag{34}$$

since $\delta > 0$ and $w_j^* \geq 0$. □

In other words, higher term values are always to be preferred. Hence, an upper bound on $\mathtt{Y}$ can be computed by means of Algorithm 1 by replacing v_i with $\max(\mathtt{V_i})$. Unlike for the fixed value case, however, the sorting operation at line 1 cannot be performed once for all at model loading time, so that the resulting complexity is $O(n \log n)$.

Computing Bounds on Variables $\mathtt{W_i}$: Bounds for the weight variables in the fixed value case are based on Equation (8), from which the maximum slack value is obtained. The equivalent formula when term values are non constant is:

$$\sum_{i=0}^{n-1} \mathtt{W_i} \cdot (\mathtt{V_i} - \max(\mathtt{Y})) \leq 0 \tag{35}$$

The inequality is valid if $\sum_{i=0}^{n-1} \mathtt{W_i}$ can be > 0. The maximum slack corresponds to an assignment of $\mathtt{V_i}$ variables, besides $\mathtt{W_i}$. The following theorem holds:

Theorem 2. *Let $\overline{v}^*$, $\overline{w}^*$ be fixed assignment for $\overline{\mathtt{V}}$ and $\overline{\mathtt{W}}$. Then, decreasing a value v_j^* by an amount $\delta > 0$ always leads to an increased slack.*

The proof is analogous to that of Theorem 5. As a consequence, choosing the minimum possible value for each $\mathtt{V_i}$ leads to maximum slack, i.e.:

$$ms = \sum_{i=0}^{n-1} (\min(\mathtt{V_i}) - \max(\mathtt{Y})) \cdot \begin{cases} \min(\mathtt{W_i}) & \text{if } \min(\mathtt{V_i}) - \max(\mathtt{Y}) > 0 \\ \max(\mathtt{W_i}) & \text{otherwise} \end{cases} \tag{36}$$

The full notation for the slack expression is $ms(\overline{\mathtt{V}}, \overline{\mathtt{W}}, \mathtt{Y})$. By keeping a single term free and the remaining ones in the maximum slack configuration, we obtain:

$$(\mathtt{V_j} - \max(\mathtt{Y})) \cdot \mathtt{W_j} \leq \theta_i \tag{37}$$

which corresponds to Equation (10) in Section 2.2. The θ_i value is obtained analogously to the case of fixed values. Moreover, we also use w_{θ_i} to denote the sum of the weights employed in the computation of θ. Equation (37) can be used to obtain bounds for $\mathtt{W_i}$ and $\mathtt{V_i}$.

In particular, the domain of variable $\mathtt{V_i}$ can be pruned as described in Algorithm 5 (that should be integrated in Algorithm 2). Lines 2 and 5 are needed to handle the $\mathtt{W_i} = 0$ case. Lines 3 and 6 update the domain max of $\mathtt{V_i}$ according to Equation (37). Note that the $\mathtt{W_i}$ assignment leading to the correct bound depends on the sign of θ_i. The rounding operations at lines 3 and 6 are necessary since $\mathtt{V_i}$ is an integer variable. Finally, y_{max} is defined as from Algorithm 2.

Bounds on the weight variable $\mathtt{W_i}$ can be obtained via Algorithm 6, which should replace lines 14-15 in Algorithm 2. The maximum slack is obtained in this case by assigning $\mathtt{V_i}$ to the minimum possible value. Then, depending on the sign of the resulting reduced value and on the sign of θ_i we obtain either a lower bound or an upper bound on $\mathtt{W_i}$. As a particular case, line 3 corresponds

Algorithm 5 Filtering on $\mathtt{V_i}$

1: **if** $\theta_i < 0$ **then**
2: **if** $\max(\mathtt{W_i}) = 0$ **then** fail
3: **else** $\max(\mathtt{V_i}) \leftarrow \left\lfloor \frac{\theta_i}{\min(\mathtt{W_i})} + y_{max} \right\rfloor$
4: **else**
5: **if** $\max(\mathtt{W_i}) = 0$ **then** do nothing
6: **else** $\max(\mathtt{V_i}) \leftarrow \left\lfloor \frac{\theta_i}{\max(\mathtt{W_i})} + y_{max} \right\rfloor$

Algorithm 6 Filtering on $\mathtt{W_i}$

1: **if** $\theta_i < 0$ **then**
2: **if** $\min(\mathtt{V_i}) - y_{max} < 0$ **then**
3: $\min(\mathtt{W_i}) \leftarrow \left\lceil \frac{\theta}{\min(\mathtt{V_i}) - y_{max}} \right\rceil$
4: **else** fail
5: **else**
6: **if** $\min(\mathtt{V_i}) - y_{max} \leq 0$ **then**
7: do nothing
8: **else** $\max(\mathtt{W_i}) \leftarrow \left\lfloor \frac{\theta}{\min(\mathtt{V_i}) - y_{max}} \right\rfloor$

to a negative upper bound (hence to a fail) and line 5 to a negative lower bound (hence to no pruning).

Similarly to Section 2.2, the case of $w_\theta = 0$ should be handle separately, since it may compromise the assumption that $\sum_{i=0}^{n-1} \mathtt{W_i} > 0$. In this case, we have:

$$\frac{\mathtt{V_i} \cdot \mathtt{W_i}}{\mathtt{W_i}} \leq \max(\mathtt{Y}) \tag{38}$$

If $\max(\mathtt{Y}) < 0$, then we have $\mathtt{V_i} \leq \max(\mathtt{Y})$ and the minimum allowed value for $\mathtt{W_i}$ is 1. Conversely, if $\max(\mathtt{Y}) \geq 0$ then we have $\mathtt{V_i} \leq \max(\mathtt{Y})$ if $\min(\mathtt{W_i}) > 0$, and no propagation if $\min(\mathtt{W_i}) = 0$.

6 Concluding Remarks

We have introduced a novel global constraint to provide a convenient model and effective filtering for weighted average expressions. The `average` constraint can be fruitfully applied to allocation problems with balancing constraints or objectives. Furthermore, `average` is a key enabler for optimization approaches making use of a complex system models, extracted via Machine Learning. We provided an efficient incremental filtering algorithm for the case with fixed term values. This was used to successfully tackle a large scale combinatorial problem featuring a Neural Network model learned from a system simulator. Finally, we discussed how to perform non-incremental filtering when both term values and weights are variable. Further developments of the constraint are strongly connected to ongoing research on the hybridization of CP, Machine Learning and Simulation.

Acknowledgement: This project if funded by a *Google Focused Grant Program on Mathematical Optimization and Combinatorial Optimization in Europe*, with title: "Model Learning in Combinatorial Optimization: a case study on thermal aware dispatching".

References

1. A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Optimization and Controlled Systems: A Case Study on Thermal Aware Workload Dispatching. *Accepted for publication at AAAI 2012*.
2. A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Neuron Constraints to Model Complex Real-World Problems. *Proc. of CP*, pages 115–129, 2011.
3. JE Beasley. OR-Library : Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
4. W. Harvey and J. Schimpf. Bounds consistency techniques for long linear constraints. In *In Proc. of the TRICS workshop, at CP*, 2002.
5. J. Howard and S. Dighe et al. A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS. *Proc. of ISSCC*, pages 108–109, February 2010.
6. L. Perron. Operations Research and Constraint Programming at Google. In *Proc. of CP*, page 2, 2011.
7. G Pesant and J C Régin. Spread: A balancing constraint based on statistics. *Proc. of CP*, pages 460–474, 2005.
8. JC Régin. Global Constraints: a survey. In *Hybrid Optimization*, pages 63–134. Springer New York, 2011.
9. P Schaus, Y Deville, P. Dupont, and J.C. Régin. Simplification and extension of the spread constraint. In *Third workshop on Constraint Propagation and Implementation, in CP06*, pages 72–92, 2006.
10. P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. The deviation constraint. In *Proc. of CPAIOR*, pages 260–274. Springer, 2007.
11. Pierre Schaus, Yves Deville, and Pierre Dupont. Bound-Consistent Deviation Constraint. In *Proc. of CP*, pages 620–634, 2007.
12. R. Sridharan. The capacitated plant location problem. *European Journal of Operational Research*, 87(2):203–213, December 1995.