The Ordered Distribute Constraint

Thierry Petit
Mines-Nantes, LINA UMR CNRS 6241
4, rue Alfred Kastler,
44307 Nantes, France.
Email: Thierry.Petit@mines-nantes.fr

Jean-Charles Régin
Université de Nice-Sophia Antipolis, I3S, CNRS
930, route des colles, BP 145
06903 Sophia Antipolis cedex, France.
Email: Jean-Charles.Regin@unice.fr

***Abstract*—In this paper we introduce a new cardinality constraint: OrderedDistribute. Given a set of variables, this constraint limits for each value v the number of times v or any value greater than v is taken. It extends the global cardinality constraint, that constrains only the number of times a value v is taken by a set of variables and does not consider at the same time the occurrences of all the values greater than v. We design an algorithm for achieving generalized arc-consistency on OrderedDistribute, with a time complexity linear in the sum of the number of variables and the number of values in the union of their domains. In addition, we give some experiments showing the advantage of this new constraint for problems where values represent levels whose overrunning has to be under control.**

I. Introduction

Recent works address the issue of characterizing solutions that are acceptable in practice thanks to global constraints. The Spread and Deviation constraints [4], [11] enforce some balancing of values within a set of variables. Balance is often important in assignment problems, for instance the daily assignment of newborn infant patient to nurses [3], [12].

In some problems, it is required to define for some subsets of variables several levels of values, and to limit for each level the maximum number of values over this level. Existing balancing constraints such as Spread or Deviation cannot be used because they globally limit the sum of the taken values. Furthermore, since often we manipulate values representing costs, a high value v is generally at least as undesirable as any of the values which are less than v. In this case, classical cardinality constraints such as Gcc [9] are not well-suited because they limit the number of occurrences of each value taken separately.

In order to solve this issue, we introduce OrderedDistribute, a new constraint that fills in this gap by limiting, for each value v, the *number of occurrences of v and all the values greater than v* within a set of variables. Then, we describe an efficient filtering algorithm establishing arc consistency associated with it.

The paper is organized as follows. Section II gives the background useful to understand our contribution. Some motivations of our work are presented in Section III. Section IV discusses the reformulation of OrderedDistribute with existing arithmetic and cardinality constraints. In section V, we propose a filtering algorithm establishing generalized arc-consistency in a linear time complexity. We illustrate in Section VI the practical interest of our approach by some experiments. At last, we discuss the extension of OrderedDistribute with range variables representing the cardinalities and we conclude.

II. Background

A *constraint network* $\mathcal{N}$ is defined as a set of n *variables* $X = \{x_1, \dots, x_n\}$, a set of current *domains* $\mathcal{D} = \{D(x_1), \dots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible *values* for variable x_i, and a set $\mathcal{C}$ of *constraints* between variables. An assignment of values to variables in X is denoted by $A(X)$, and for each $x \in X$, $A(X, x)$ is the value of x in $A(X)$. $A(X)$ is *valid* iff $\forall x_i \in X$, $A(X, x_i) \in D(x_i)$. A constraint $C(X)$ specifies the allowed combinations of values for a set of variables X, that is, it defines a subset $\mathcal{R}_C(\mathcal{D})$ of the Cartesian product $\Pi_{x_i \in X} D(x_i)$ of the domains of variables in X. A *feasible assignment* of $C(X)$ is an assignment which is in $\mathcal{R}_C(\mathcal{D})$. If $A(X)$ is a feasible assignment of $C(X)$ then we say that $A(X)$ *satisfies* $C(X)$.
For convenience, given a value v and an assignment $A(X)$, we denote by $\#(v, A(X))$ the number of time v appears in $A(X)$ and by $\#(\geq v, A(X))$ the number of values $w \geq v$ that appear in $A(X)$.

Let C be a constraint over the variables X. A *support* on C is an assignment which satisfies C. A domain $D(x)$ of $x \in X$ is *arc-consistent* w.r.t. C iff $\forall v \in D(x)$, v belongs to a valid support on C. C is *(generalized) arc-consistent* (GAC) iff $\forall x_i \in X$, $D(x_i)$ is arc-consistent w.r.t. C.

III. Motivations and Definition

To illustrate the need of the OrderedDistribute constraint, we present an example of cumulative scheduling where costs (*i.e.*, values) are used to express over-loads of capacity and where constraints related to different levels of over-loads are defined.

Scheduling problems consist in ordering some activities. In *cumulative scheduling*, each activity requires for its execution the availability of a certain amount of renewable resource. In CP, activities are represented by variables, and cumulative problems can be encoded thanks to a dedicated constraint, Cumulative [1].

Let A be a set of n non-preemptive activities (i.e. activities that cannot be interrupted). For each $a \in A$, $start[a]$ is the variable representing its starting point in time. $dur[a]$ is the variable representing its duration. $res[a]$ (height of a) is the variable representing the discrete amount of resource

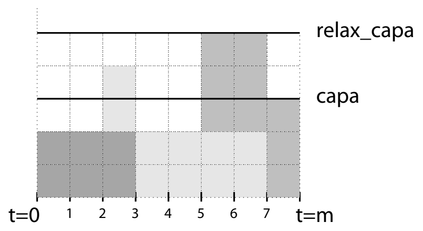

Fig. 1. Example of SOFTCUMULATIVESUM with a ground schedule with 3 fixed activities: a_1 starts at 0 and ends at 3, and $res[a_1] = 2$. a_2 starts at 2 and ends at 7, and $res[a_2] = 2$. a_3 starts at 5 and ends at 8, and $res[a_3] = 3$. There is 3 over-loads: $cost[2] = 1$, $cost[5] = 2$, $cost[6] = 2$.

consumed by activity a. In this paper, we will consider cumulative problems where each activity consumes a resource.

*Definition 1 (*CUMULATIVE*):* Given one resource with a capacity limited by $capa$ and a set A of n activities, at each point in time t the cumulated height h_t of the activities overlapping t is $h_t = \sum_{a \in A, start[a] \leq t < end[a]} res[a]$. The CUMULATIVE$(A, capa)$ constraint [1] enforces that:

C1 : For each activity $a \in A, start[a] + dur[a] = end[a]$.
C2 : At each point in time t, $h_t \leq capa$.

When the *time horizon* (maximum latest completion time among all activities) is fixed, the problem may have no solution if no over-loads on the resource capacity are tolerated. In [5], authors introduced a relaxed version of CUMULATIVE devoted to this class of problems.

*Definition 2 (*SOFTCUMULATIVESUM*):* The constraint SOFTCUMULATIVESUM relaxes CUMULATIVE by introducing:

- A time horizon th.
- A *relaxed capacity* of the resource denoted by $relax\_capa$ with $capa \leq relax\_capa$.
- An integer variable $cost[t]$ for each point in time[1] $t < th$.
- An integer variable obj.

It enforces the following constraints:

C1 : For each activity $a \in A, start[a] + dur[a] = end[a]$.
C2 : At each point in time t, $h_t \leq relax\_capa$.
C3 : For each point in time t, $cost[t] = max(0, h_t - capa)$
C4 : $obj \ = \sum_{t \in \{0,\dots,m-1\}} cost[t]$

Over-loads are accepted in the SOFTCUMULATIVESUM constraint and the objective variable (obj) represents the sum of the overloads. Figure 1 contains an example of this constraint.

However, for some problems, constraining the sum of the over-loads is not enough. Some additionnal constraints w.r.t. these over-loads should be satisfied. A frequent requirement is to distribute fairly over-loads within a long time period, while limiting the number of big over-loads for each short period of time.

For instance, assume that the resource is the number of employees in a team. Each day (8 hours), each person contributes to some activities. An over-load will entail, in practice, either that extra-employees are hired, or that some activities are performed by a number of employees less than the number which was initially planed (for instance, some employees may accept to do homework). Typically, small over-loads ($cost[t] = 1$) correspond to the second case, while big ones (e.g., $cost[t] > 3$) require to hire extra-employees. Then, the number of each type of over-loads has to be limited. This means that the number of times a variable $cost$ is greater than a given value has to be limited. Thus, we need to express that *at most k variables can take a value v or greater.*

This is the purpose of our new constraint ORDEREDDISTRIBUTE. We define it formally:

*Definition 3 (*ORDEREDDISTRIBUTE*):* Let

- X be a set of variables
- T be an array of increasing values that can be assigned to variables in X, with $|T| \geq 2$.
- I_{max} be an array of maximum possible number of occurrences of values in T, where $I_{max}[i]$ corresponding to the value $T[i]$, and such that $\forall i \in \{1, \dots, |T| - 1\}$, $I_{max}[i-1] \geq I_{max}[i]$.

An assignment $A(X)$ satisfies the constraint ORDEREDDISTRIBUTE*(X, T, I_{max})* iff

1) For each $i \in \{0, \dots, |T| - 1\}$, the number of values v in $A(X)$ s.t. $v \geq T[i]$ is at most equal to $I_{max}[i]$.
2) The number of times value $T[0]$ appears in $A(X)$ is at least equals to $|X| - I_{max}[1]$.

Observe that ORDEREDDISTRIBUTE also implicitly affects minimum occurrences of values: given an index $i < |T| - 1$, the number of times a value $v \leq T[i]$ appears in $A(x)$ is at least equal to $|X| - I_{max}[i + 1]$.

Example 1: We consider a human resource cumulative problem with 5 days of 8 hours, and a team of 8 employees. The following constraints are imposed w.r.t. over-loads.

1) At most 5 over-loaded hours by day.
2) At most 3 over-loads greater than or equal to 2.
3) At most 1 over-load equal to 4. (no over-load should exceed 4 employees).

Here is a fully detailed instance. We consider 40 activities with the following durations: [1, 1, 3, 1, 4, 2, 2, 1, 4, 4, 2, 4, 2, 3, 1, 2, 1, 1, 1, 2, 1, 3, 3, 2, 4, 3, 3, 1, 4, 2, 4, 2, 3, 4, 3, 4, 2, 3, 3, 2, 2, 2, 4, 4, 2, 4, 3, 4, 3, 4, 3, 2, 4, 4, 4]. Heights of activities are the following: [4, 2, 1, 1, 2, 3, 4, 1, 3, 3, 2, 1, 4, 4, 2, 3, 4, 1, 1, 1, 2, 4, 3, 3, 3, 1, 2, 4, 3, 2, 1, 4, 2, 3, 4, 1, 4, 4, 1, 2, 1, 1, 4, 3, 2, 2, 1, 3, 4, 4, 3, 2, 4, 1, 1].

Observe that an over-load of 4 is at least as undesirable as an over-load of 3, an over-load of 3 is at least as undesirable as an over-load of 2, and so on.

Figure 2 shows the over-loads in each day, in the cumulative profile corresponding to that instance. The sum of over-loads over the whole month is 48.

We can easily encode this problem by using the ORDEREDDISTRIBUTE constraint defined with $T = [0, 1, 2, 3, 4]$ and $I_{max} = [8, 5, 3, 3, 1]$ where T measures the over-loads

[1]In [5], each variable $cost[t]$ can correspond to an interval of consecutive points in time, not only one point in time. Without loss of generality and for sake of simplicity, we use a definition where the length of intervals is 1.

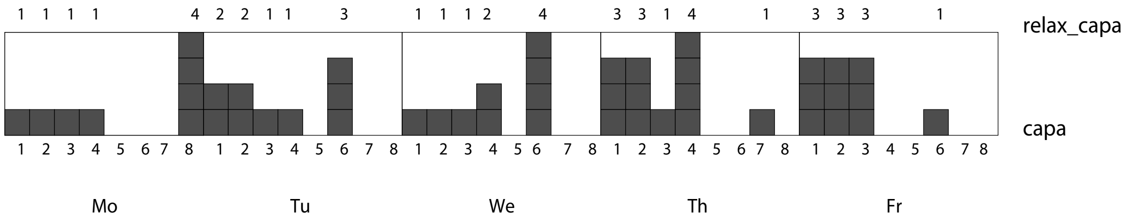

Fig. 2. Over-loads in the solution of Example 1.

and I_{max} limits the number of occurrences for each day. For instance, $T[2] = 2$ and $I_{max}[2] = 3$ means that we can have at most 3 times an over-load of size 2 or more for each day. The following model, written in pseudo-code, represents this problem.

```
// ds are the durations, hs the heights
// capa and relax_capa the capacities
IntDomainVar[] start, costVar;
IntDomainVar obj;
SOFTCUMULATIVESUM(start, costVar, obj, ds,
                  hs, capa, relax_capa);
// additionnal constraints
$T = [0,1,2,3,4]$;  $I_{max} = [8,5,3,3,1]$;
for each day in a partition of costVar[]:
      ORDEREDDISTRIBUTE(day, $T$, $I_{max}$)
// objective
minimize(obj);
```

Here is a solution of the problem (each range $[d_i, e_i[$ indicates the points in time where activity a_i starts and ends in the schedule): $[30,31[$ $[37,38[$ $[32,35[$ $[12,13[$ $[0,4[$ $[24,26[$ $[21,23[$ $[15,16[$ $[0,4[$ $[8,12[$ $[35,37[$ $[27,31[$ $[26,28[$ $[7,10[$ $[38,39[$ $[26,28[$ $[31,32[$ $[39,40[$ $[39,40[$ $[38,40[$ $[39,40[$ $[12,15[$ $[23,26[$ $[28,30[$ $[8,12[$ $[32,35[$ $[32,35[$ $[31,32[$ $[10,14[$ $[35,37[$ $[32,36[$ $[27,29[$ $[32,35[$ $[15,19[$ $[13,16[$ $[32,36[$ $[29,31[$ $[19,22[$ $[32,35[$ $[36,38[$ $[38,40[$ $[38,40[$ $[0,4[$ $[16,20[$ $[36,38[$ $[23,27[$ $[37,40[$ $[16,20[$ $[20,23[$ $[4,8[$ $[23,26[$ $[37,39[$ $[4,8[$ $[32,36[$ $[32,36[$

More generally, OrderedDistribute may be useful in practice in several classes of problems:

- In bin-packing problems when objects have to be brought in containers. For instance, a fair distribution based on some degrees indicating frailty of the objects allows to limits the negative consequences (in terms of financial costs) of damaged containers.
- In assignment problems when teams of have to be balanced with respect to the hierarchical skills of the members.
- In over-constrained problems, in which costs represent degrees of violation of constraints.[2] These costs are often strongly ordered. (Example 1 belongs to this class.)

[2]A problem is over-constrained when it has no solution. To find compromising solutions that will be concretely applied in practice, these problems can be view as optimization problems in which some constraints may be violated.

IV. Reformulation of OrderedDistribute

Reformulating OrderedDistribute with existing cardinality constraints is not straightforward because limiting the maximum number of occurrences of a value v does not constrain the occurrences of all the values which are greater than v. It is necessary to augment such existing cardinality constraints with addional constraint like arithmetic ones.

The most famous cardinality constraint is the global cardinality constraint or Gcc. It is defined as follows:

*Definition 4 (*Gcc*):* Let X be a set of variables, T be an array of values, and I be the array of allowed integer ranges for each value of T.
An assignment $A(X)$ satisfies the constraint Gcc(X, T, I) iff any value $v = T[i]$ appears in $A(X)$ a number of times which belongs to $I[i]$, i.e. $\#(T[i], A(X)) \in I[i]$, $i = 0, \ldots, |X| - 1$.

GAC can be established efficiently on Gcc [9], [8].

When I are defined by boundaries of range variables (the *card variables*), the Gcc constraint becomes the cardVar-Gcc constraint [10]. A range variable is a variable which is represented by the minimum and the maximum values in its domain. Thus, the parameter I is replaced by the set of range variables $Card$, so as the signature is cardVar-Gcc(X, T, $Card$). Filtering algorithms for cardVar-Gcc can be found in [10], [8]. Unfortunately, their time complexity is cubic.

Now, we can reformulate the OrderedDistribute constraint with a cardVar-Gcc constraint and some arithmetic constraints.

Consider C =OrderedDistribute(X,T,I_{max}). Let $t = |T| - 1$ and $n = |X|$. We define $CN(C)$ the constraint network corresponding to the OrderedDistribute constraint as follows:

- The variable set is $X \cup Card$, where $Card = \{Card[0], ..., Card[t]\}$ is a set of non negative integer variables.
- The constraints set if defined by the constraints:
  - $\forall k = 0...t$, $\sum_{i=k}^{t} Card[i] \leq I_{max}[k]$
  - $\forall k = 1...t$, $\sum_{i=0}^{k-1} Card[i] \geq n - I_{max}[k]$
  - cardVar-Gcc(X, T, $Card$)

We explicitly add the second type of constraints because as far as we know solvers are not able to deduce from two sum constraints $\sum_{i=p+1}^{n} y_i \leq a$ and $\sum_{i=1}^{n} y_i = n$ that we have $\sum_{i=1}^{p} y_i \geq n - a$.

It is easy to check that this constraint network reformulates the constraint C (*i.e.*, they have the same set of solutions). Unfortunately, this reformulation is weak even if the strongest filtering algorithms are used for each constraint, as shown by the following example:

Example 2: Let $X = [x_1, x_2, x_3, x_4, x_5]$ be 5 variables such that $D(x_1) = D(x_2) = \{0, 1\}$, $D(x_3) = \{0, 1, 2\}$, and $D(x_4) = D(x_5) = \{2, 3\}$. We consider the following constraints :

1) At most 3 x_i greater than or equal to 1.
2) At most 2 x_i greater than or equal to 2.
3) At most 2 x_i equal to 4.

This example can be modeled by the constraint C =ORDEREDDISTRIBUTE(X, T, I_{max}), with $T = [0, 1, 2, 3]$ and $I_{max} = [5, 3, 2, 2]$. The constraint network associated with C involves the constraints:

(1) $Card[3] \leq I_{max}[3] = 2$
(2) $Card[2] + Card[3] \leq I_{max}[2] = 2$
(3) $Card[1] + Card[2] + Card[3] \leq I_{max}[1] = 3$
(4) $Card[0] + Card[1] + Card[2] + Card[3] \leq I_{max}[0] = 5$
(5) $Card[0] + Card[1] + Card[2] \geq |X| - I_{max}[3] = (5 - 2) = 3$
(6) $Card[0] + Card[1] \geq |X| - I_{max}[2] = (5 - 2) = 3$
(7) $Card[0] \geq |X| - I_{max}[1] = (5 - 3) = 2$
(8) CARDVAR-GCC(X, T, $Card$)

If the strongest filtering algorithms are used for $CN(C)$ then the domains of the $Card$ variables become $D(Card[3]) = \{0, 1, 2\}$; $D(Card[2]) = \{0, 1, 2\}$; $D(Card[1]) = \{0, 1, 2, 3\}$; $D(Card[0]) = \{2, 3, 4, 5\}$.

No value is removed from the domain of variables of X, whereas it should. Note that variables x_4 and x_5 can only take a value in $\{2, 3\}$ and at the same time the number of times any value greater than or equal to 2 can be taken is 2, because of the constraint number 2) in Example 2. Therefore, no other variable can take a value in $\{2, 3\}$ and so value 2 can be safely removed from $D(x_3)$. CARDVAR-GCC does not remove such a value, because it considers, for instance, that x_4 and x_5 can take value 3 and then x_3 value 2. In addition, note that the filtering algorithms associated with some constraints used in this reformulation are costly in practice. Thus, we have two reasons for designing an efficient filtering algorithm for the ORDEREDDISTRIBUTE constraint.

V. FILTERING ALGORITHMS

A. Filtering Algorithm Based on Flow

First, we can note that it is possible to design a filtering algorithm based on flow as for the GCC constraint. An algorithm in $O(n^2k)$ for checking consistency and establishing arc consistency is presented in [6], where $n = |X|$ and $k = |T|$. We will not detail the algorithm here, but we will give the main idea because it solves the issue w.r.t. the lack of filtering of the reformulation illustrated by Example 2. Moreover, it shows how some arithmetic constraints between cardinality variables can be integrated into a classical filtering algorithm for GCC. Further, this technique might also be useful to deal with generalizations of the ORDEREDDISTRIBUTE constraint.

This filtering algorithm is based on the search for a flow of value $n = |X|$ in a particular digraph. For convenience, we will denote by lb the lower bound capacity of an arc and by ub its upper bound capacity. Figure 3 is an example of such a digraph associated with an ORDEREDDISTRIBUTE constraint.

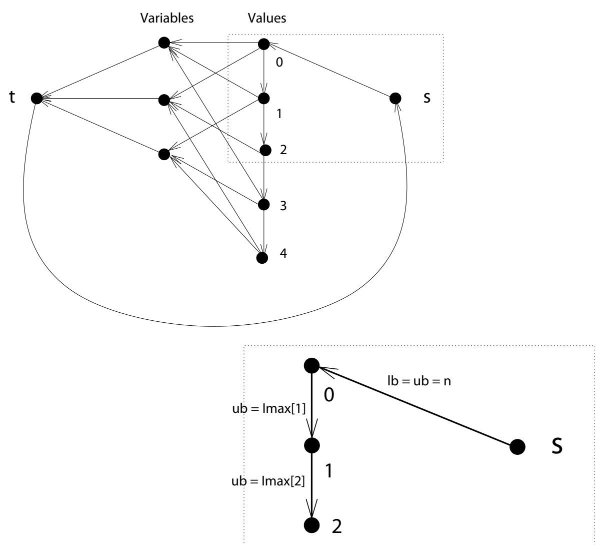

Fig. 3. Example of a digraph representing an ORDEREDDISTRIBUTE(X, T, I_{max}). Arcs between values and variables have a lower bound equal to 0 and an upper bound equal to 1.

*Definition 5 (*ORDEREDDISTRIBUTE *variable-value digraph):* Let C =ORDEREDDISTRIBUTE(X, T, I_{max}), we define the digraph $G(C) = (X_G, U_G)$ as follows:

- $X_G = \{s, t\} \cup X \cup T$.
- U_G contains:
  - An arc $(T[i], x)$ for each $T[i] \in T$ and $x \in X$ s.t. $T[i] \in D(x)$. $ub(T[i], x) = 1$.
  - An arc $(T[i], T[i+1])$ for each $i \in \{0, \dots, |T| - 2\}$ with $ub(T[i], T[i+1]) = I_{max}[i+1]$.
  - An arc (x, t) for each $x \in X$ with $ub(x, t) = 1$.
  - An arc $(s, T[0])$ with $ub(s, T[0]) = |X|$.
  - An arc (t, s) with $lb(t, s) = |X|$ and $ub(t, s) = |X|$.

The main idea is to link the values together. In this way, the flow value which reaches a value v of T must pass by all the values less than v and so it is possible to count for these values the quantity of flow corresponding to the assignments of variables to values greater than them.

Proposition 1: Given C = ORDEREDDISTRIBUTE(X, T, I_{max}) and its corresponding digraph $G(C)$, the two following properties are equivalent:

- ORDEREDDISTRIBUTE(X, T, I_{max}) has a solution.
- There exists a feasible flow from t to s in $G(C)$.

Once a feasible flow of value n has been computed in $G(C)$, arc consistency can be established with the same algorithm as for GCC, so with a linear complexity.

Such a filtering technique requires to work with an additionnal data structure (the digraph associated with the constraint) and to compute and maintain a flow. In the next section we propose a simpler algorithm which is also more efficient in practice.

B. Linear Filtering Algorithm

We first come up a consistency check for OrderedDistribute, in a time complexity linear over the number of variables. Then, we come up with the linear GAC filtering algorithm.

By Definition 3, we have the following lemma.

Lemma 1: If OrderedDistribute(X, T, I_{max}) has a solution then $I_{max}[0] \geq |X|$.

We define the assignment with all minimum values of domains.

Definition 6 (Min-domain assignment): The min-domain assignment of a set of variables X is the unique assignment $\underline{A}(X)$ of values to variables in X s.t. $\forall x \in X$, $\underline{A}(X, x)$ is equal to the minimum value in $D(x)$.

Obviously, it is always possible to build a min-domain assignment if there is no empty domain, but this assignment does not necessarily satisfy the constraint.

Proposition 2: Let $\underline{A}(X)$ be the min-domain assignment of an instance C of OrderedDistribute. The two propositions are equivalent:

(α) $\#(T[0], \underline{A}(X)) \geq |X| - I_{max}[1]$
and $\forall i = 0, ..., |T| - 1$: $\#(\geq T[i], \underline{A}(X)) \leq I_{max}[i]$.
(β) C has a solution.

Proof: ($\Rightarrow$) Suppose that (α) is satisfied. By definition 3, $\underline{A}(X)$ is a solution of C. ($\Leftarrow$) Suppose that C has a solution. By contradiction: assume that the min-domain assignment $\underline{A}(X)$ of C does not satisfy (α). Two cases are (mutually) possible: (i) The number of times $T[0]$ is assigned to a variable in $\underline{A}(X)$ is strictly less than $|X| - I_{max}[1]$. By Definition 6, any variable x s.t. $T[0] \in D(x)$ is assigned to $T[0]$ in $\underline{A}(X)$. Therefore, no other assignment can have a greater number of occurrences of $T[0]$ and thus satisfies C, a contradiction. (ii) Assume that a value $T[i]$ is s.t. $\#(\geq T[i], \underline{A}(X)) > I_{max}[i]$. By definition 6, if a value greater than $T[0]$ is assigned to a variable x in $\underline{A}(X)$, this value is the minimum of $D(x)$. Variables x in $\underline{A}(X)$ that take value $T[i]$ cannot take a value strictly less than $T[i]$. No assignment exists with a lower value for $\#(\geq T[i], \underline{A}(X))$. C has no solution, a contradiction. ■

From Proposition 2 and Definition 6, the feasibility of an OrderedDistribute can be checked in $O(n)$, where $n = |X|$.

Algorithm 1: isSatisfiable(OrderedDistribute(X, T, I_{max})): boolean

```
1 if $\#(T[0], \underline{A}(X)) < |X| - I_{max}[1]$ then
2     return false;
3 foreach $T[i] \in T$ do
4     if $\#(\geq T[i], \underline{A}(X)) > I_{max}[i]$ then return false;
5 return true;
```

Thanks to this procedure, time complexity of a flow-based algorithm (see Section V-A) can be decreased to $O(nk)$, where $k = |T|$. This time complexity can be improved again, by using an algorithm which does not require to work with an additionnal data structure.

We present now this dedicated filtering algorithm for OrderedDistribute.

Next Corollary gives a sufficient condition for having all the values consistent with OrderedDistribute.

Corollary 1: Let C =OrderedDistribute(X, T, I_{max}), if $\forall i \in \{0, \ldots, |T| - 1\}$, $\#(\geq T[i], \underline{A}(X)) < I_{max}[i]$ then $\forall x \in X$, $\forall v \in D(x)$, (x, v) is consistent with C.

Proof: The assignment $A'(X)$ obtained by replacing $\underline{A}(X, x)$ by v in $\underline{A}(X)$ is s.t. $\forall T[i] \in T, \#(\geq T[i], A'(X)) \leq I_{max}[i]$. By Definition 3, $A'(X)$ is a solution of C. ■

Now, we can establish the corollary defining precisely the consistent values. Intuitively, there are two reasons for a value $v \in D(x)$ not to be consistent. The first one is that its variable x must be assigned to $T[0]$ to satisfy the minimum requirement and $v > T[0]$. The second one is that, a maximum of occurrences is reached for a value $w \leq v$ when all variables take their minimum value and x is assigned to a value $u < w$. Thus x cannot be assigned to v because in this case the number of value assigned to a value equal or greater than w will be strictly greater than $I_{max}[k]$ with $w = T[k]$. We denote by $X_\perp$ the set of variables whose domain contains $T[0]$: $X_\perp = \{x \in X \text{ s.t. } T[0] \in D(x)\}$.

Corollary 2: Let C be a feasible OrderedDistribute(X, T, I_{max}) and $\underline{A}(X)$ be the min-domain assignment. A value $v \in D(x)$ is *not* consistent with C if and only if one of the two following property is satisfied:

(α) $x \in X_\perp$, $v > T[0]$ and $|X_\perp| = |X| - I_{max}[1]$.
(β) $\exists T[i] \in T, \#(\geq T[i], \underline{A}(X)) = I_{max}[i]$ and $\underline{A}(X, x) < I_{max}[i]$ and $v \geq T[i]$.

Proof: (α) is immediate. (β) By definition, if in $\underline{A}(X)$ a value is assigned to a variable x, this value is the minimum of $D(x)$. Therefore, if $T[i]$ satisfies $\#(\geq T[i], \underline{A}(X)) = I_{max}[i]$, then given $x \in X$ with $\underline{A}(X, x) < I_{max}[i]$, there exists no assignment $A'(X)$ s.t. $A'(X, x) \geq I_{max}[i]$ and $\#(\geq T[i], A'(X)) \leq I_{max}[i]$. ■

From Corollary 2 we obtain Algorithm 2. Values removed from a domain $D(x)$ are necessarily strictly greater than $\underline{A}(X, x)$. It is necessary to evaluate each of these values (line 10) because some new variables can be reached when evaluating higher values in $T_=$ (defined in line 8), thanks to the condition of line 12.

This algorithm applies Corollary 2 and it enforces GAC. Indeed, assume that a value $T[i] \in D(x)$ is not consistent with the constraint after the run of the algorithm. This means that assigning $T[i]$ to x entails in any complete assignment of X the existence of a value $T[j]$ such that $\#(\geq T[j], \underline{A}(X)) > I_{max}[j]$ and $j \leq i$; especially in the min-domain assignment $\underline{A}(X)$. By construction of $T_=$ and from line 13 of the algorithm, when $T[j] \in T_=$ was treated, $T[i]$ was removed from $D(x)$ by the algorithm, a contradiction.

Proposition 3: The time complexity of this algorithm is $O(n + k)$, where $n = |X|$ and $k = |T|$, provided that given $v \in D(x)$, we can remove all values greater than v in $O(1)$.

Algorithm 2: Filtering algorithm for OrderedDistribute(X, T, I_{max})

```
1  if ¬ isSatisfiable(OrderedDistribute($X$, $T$, $I_{max}$)) then
2      return;
3  $\underline{A}(X) \leftarrow$ min-domain assignment of $X$ ;
4  $X_{\perp} \leftarrow \{x \in X$ s.t. $T[0] \in D(x)\}$ ;
5  if $|X_{\perp}| = |X| - I_{max}[1]$ then
6      foreach $x \in X_{\perp}$ do
7          $D(x) \leftarrow \{T[0]\}$ ;
8  $T_{=} \leftarrow \{T[i] \in T$ s.t. $\#(\geq T[i], \underline{A}(X)) = I_{max}[i]\}$ ;
9  $X' \leftarrow X$ ;
10 while $T_{=} \neq \emptyset \wedge X' \neq \emptyset$ do
11     Pick and remove the minimum value $T[i]$ in $T_{=}$ ;
12     foreach $x \in X'$ s.t. $\underline{A}(X, x) < I_{max}[i]$ do
13         Remove from $D(x)$ the set $\{v \in D(x), v \geq T[i]\}$ ;
14         $X' \leftarrow X' \setminus \{x\}$ ;
```

Proof: (Sketch) Values in $T_{=}$ (line 11) and in $\underline{A}(X)$ (line 12) can be sorted in increasing order in linear time by a counting sort since T is bounded. The total number of times lines $12 - 14$ of the algorithm are executed is upper-bounded by $|X|$: if a variable is reached then it is removed from X' by line 14. Thus the time complexity is in $O(|T| + |X|)$ ■

Proposition 3 states that GAC can be enforced on OrderedDistribute with a time complexity linear in the sum of the number of variables and the number of values in the union of their domains. Regarding the literature, we can note that some other simple generalizations of Gcc are NP-Hard [7].

We can adapt Algorithm 2 to make it incremental, by maintaining at least the following data:

- The min-domain assignment $\underline{A}(X)$.
- An array $T_{\#_{\geq}}$ counting, for each value $T[i]$, the number of values $w \geq T[i]$ that appear in $\underline{A}(X)$.

However, even in this case, each time the counter of a value in $T_{\#_{\geq}}$ becomes such that $T_{\#_{\geq}}[i] = I_{max}[i]$ it will be necessary to scan the variables in order to remove all the values greater than $T_{\#_{\geq}}[i]$ for all variables having a value in $\underline{A}(X)$ strictly less than $I_{max}[i]$. Thus, time complexity remains in $O(|X| + |T|)$, and it can be amortized on a given branch of the search tree only w.r.t. $|T|$, which is not very relevant : Algorithm 2 has a linear time complexity and there is generally a few number of cost values. The practical time cost for updating the stored data $\underline{A}(X)$ and $T_{\#_{\geq}}$ and the trail seems to be prohibitive. For the experiments we implemented Algorithm 2 with the version presented in this section.

VI. Experiments

We experimented our global constraint on instances of the problem described in Example 1, which is derived from [5], using the Java-based constraint programming engine Choco[3].

We compared two representations of OrderedDistribute.

[3]http://www.emn.fr/z-info/choco-solver/index.html

	CN-Model with $dom/wdeg$	GlobalCt-Model with $StartsMinDomain$
Obj	#Bk / Time / Proof	#Bk / Time / Proof
47	- / > 10 min / -	**963 / 4 s / yes**
13	**122 / 0 s / yes**	50027 / 59 s / yes
31	18550 / 2 min 52 s / yes	**370 / 5s / yes**
16	**44 / 0 s / yes**	2562 / 11 s / yes
9	- / > 10 min / -	**448192 / 10 min / no**
56	- / > 10 min / -	**529 / 1 s / yes**
19	**26 / 0 s / yes**	1361 / 8 s / yes
2	1819 / 5 s / yes	965 / 7 s / yes
5	12251 / 14 s / yes	**665 / 10 s / yes**
42	- / > 10 min / -	**5406 / 9s / yes**
13	33484 / 38 s / yes	**4274 / 12 s / yes**
unsat.	- / > 10 min / -	- / > 10 min / -
17	- / > 10 min / -	**533526 / 10 min / no**
48	- / > 10 min / -	**772 / 3 s / yes**
33	- / > 10 min / -	**860 / 3 s / yes**
56	- / > 10 min / -	**246 / 1 s / yes**
46	- / > 10 min / -	**546 / 3 s / yes**
14	54946 / 62 s / yes	**1116 / 8 s / yes**
43	- / > 10 min / -	**615 / 3 s / yes**
unsat.	- / > 10 min / -	- / > 10 min / -

TABLE I
$n = 55$ activities, $m = 40$ hours.

- In the first one, we use the new global constraint we have defined. We name this model GlocalCt-Model.
- In the second one the constraint is replaced by a constraint network equivalent to the one proposed in Section IV. We name this model CN-model.

We tried for each model several search strategies. With the first model (GlobalCt-Model), the best results are obtained if, first, we first assign the minimum value to the start variable having the minimum-sized domain and then the same for $cost$ variables. With the second model (CN-Model), the search strategy $dom/wdeg$ [2] was the most efficient.

Instances involve $n = 55$ activities, $m = 40$ hours, durations between 1 and 4, resource consumption between 1 and 4, $capa = 8$, $relax\_capa = 12$. Costs at each point in time are from 0 to 4, the imposed distribution is for each day of 8 hours : At most 5 costs ≥ 1, at most 3 costs ≥ 2, at most 1 cost equal to 4.

Table I gives the results obtained for instances such that no solution exists without over-loads (if there is no over-load then OrderedDistribute is not useful). Time limit is 10 minutes. The two unsolved problems have no solution satisfying the ordered cardinality constraints, but proving this unsatisfiability requires more than 10 minutes.

These results show that the use of a generic heuristic $dom/wdeg$ which is well-suited does not compensate the lack of filtering of the model CN-Model, except for a few number of instances which are easy to solve.

With OrderedDistribute (GlobatCt-Model), 16 of the 20 instances are solved and proved to be optimal in less than one minute (15 of them in less than 12 seconds), while the other model proves optimality only for 8 of the 20 instances. This latter model is not able to find a solution in a time less than 10 minutes for the 12 remaining instances. A few instances remain hard for the two models, since the optimum value cannot be found in less than 10 minutes. This is not surprising since searching for the minimum sum of over-loads in a soft cumulative problem (and proving optimality) is known to be a difficult problem.

VII. Extensions of OrderedDistribute

A natural extension of OrderedDistribute is to consider that maximum number of occurrences of values are not scalar integers but a set of range variables. We distinguish two cases. The first one is the definition directly obtained from in Definition 3 by replacing I_{max} by a set of range variables. The second one is a more useful extension of OrderedDistribute, where the number of values greater than or equal to a given value v should be *equal* to the value of its range variable.

A. OrderedDistributeRangeLeq

Definition 7 (OrderedDistributeRangeLeq): In a OrderedDistributeRangeLeq*(X, T, R)*, we use the parameters described in Definition 3 except that R is a set of range variables. Given an assignment $A(X)$, OrderedDistribute*(X, T, R)* is satisfied iff the two following constraints are satisfied.

1) $\#(T[0], A(X)) \geq |X| - R[1]$.
2) $\forall i = 0, \ldots, |T|-1$, $\#(\geq T[i], A(X)) \leq R[i]$

The consistency check remains the same than the one of OrderedDistribute except that we replace I_{max} by the sequence of maximum values in domains of variables in R. The filtering algorithm remains the same concerning variables in X. It differs from OrderedDistribute w.r.t. lower bounds of domains variables in R: They may become not consistent with OrderedDistributeRangeLeq according to the current domains of variables in X. For instance, for a value $T[i]$ it may not remain enough values less than $T[i]$ in domains of variables X to impose $\#(\geq T[i], \underline{A}(X)) \leq 0$, so as 0 can be removed from $R[i]$. Minimum values for variables in R can be updated directly by counting, for each $T[i]$, the number of variables in $\underline{A}(X)$ greater than or equal to $T[i]$. This can be done incrementaly while $\underline{A}(X)$ is built (note that $\underline{A}(X)$ remains the same after the filtering of variables in X). At last, increasing explicitly a lower-bound of a variable $R[i]$ has no consequence except a fail if $min(D(R[i])) > max(D(R[i]))$. Thus, GAC can be achieved on OrderedDistributeRangeLeq in $O(n+k)$ time, with $n = |X|$ and $k = |T|$.

B. OrderedDistributeRangeEq

In this section, we study the case where the number of values greater than or equal to a given value v should be *equal* to the value of its corresponding range variable.

Definition 8 (OrderedDistributeRangeEq): In a OrderedDistributeRangeEq*(X, T, R)*, we use the parameters described in Definition 3 except that R is a set of range variables. Given an assignment $A(X)$, OrderedDistribute*(X, T, R)* is satisfied iff the two following constraints are satisfied.

1) $\#(T[0], A(X)) \geq |X| - R[1]$.
2) $\forall i = 0, \ldots, |T|-1$, $\#(\geq T[i], A(X)) = R[i]$

The filtering differs from OrderedDistributeRangeLeq. We need to compute the exact upper bounds of domains of variables in R (lower bounds are given by $\underline{A}(X)$ similarly to OrderedDistributeRangeLeq). Next example shows that such a computation does *not* simply consists in sorting the variables $x_i \in X$ by non decreasing $max(D(x_i))$, then remove the first $|X| - R[1]$ ones, and then count for each $v \in T$ the number of values greater than v in domains of remaining x_i's.

Example 3: Let $X = \{x_1, \ldots, x_5\}$ such that $D(x_1) = D(x_2) = \{0, 4\}$, $D(x_3) = \{0, 3, 4\}$, $D(x_4) = D(x_5) = \{1, 2, 3\}$. $T = [0, 1, 2, 3, 4]$. Assume that $D(R[4]) = [0, 1]$ and $D(R[1]) = [0, |X|]$, and that we wish to prune $D(R[3])$, which is currently $[0, 5]$. The maximum possible value for $R[3]$ is 4 because x_1 and x_2 cannot take both value 4.

Consider C = OrderedDistributeRangeEq(X, T, R). We search for each value $v = T[i]$ the assignment satisfying C and having the maximum number of values greater than or equal to v. Thus, we consider a value v. First, we propose to simplify the problem by only considering two values per domain of each variable.

Notation 1: Let v be a value in T. For any variable in X let $min(x)$ be the minimum value of the domain and $w(v, x)$ be the value of $D(x)$ which is the nearest of v by excess (i.e. $w(v, x) \geq v$ and $\nexists u \in D(x)$ with $w(v, x) \geq u \geq v$).

Property 1: Let v be a value in T and $A(X)$ be any assignment satifying C with $\#(\geq v, A(X)) = j$. Then, the assignment $A'(X)$ defined from $A(X)$ as follows:

- $A'(X, x) = min(x)$ iff $A(X, x) < v$
- $A'(X, x) = w(v, x)$ iff $A(X, x) \geq v$

satisfies the constraint C and satisfies $\#(\geq v, A'(X)) = j$.

Proof: Clearly we have $\#(\geq v, A'(X)) = j$. In addition, since each value of $A(X)$ is replaced by a smaller value it is clear that if $A(X)$ satisfies C then $A'(X)$ also. ∎

We propose the following greedy algorithm:

1) We order in L the variables by their non increasing min value. We break tie by non decreasing w value.
2) We repeat the following process until L is empty: We take the first variable x of L and remove it from L; and we assign $w(v, x)$ to x if it does not violate C; otherwise we assign $min(x)$ to x.
3) The obtained assignment maximizes the number of values greater than v.

To prove correctness of this algorithm we introduce two lemmas. Let $A(X)$ be any assignment satisfying C and two variables x_1 and x_2:

Lemma 2: Assume $w(v, x_1) = A(X, x_1)$ and $min(x_2) = A(X, x_2)$. If $min(x_1) \leq min(x_2)$ and $w(v, x_1) \geq w(v, x_2)$ then the assignment $A'(X)$ with $min(x_1) = A'(X, x_1)$ and $w(v, x_2) = A'(X, x_2)$ and $\forall y \in X - \{x_1, x_2\} : A'(X, y) =$

$A(X, y)$ satisfies C and has the same number of values greater than or equal to v as $A(X)$.

Proof: $A'(X)$ has obviously the same number of values greater than or equal to v as $A(X)$. $A(X)$ contains the set of value V and the values $w(v, x_1)$ and $min(x_2)$; and $A'(X)$ contains the set of value V and the values $min(x_1)$ and $w(v, x_2)$. Since $min(x_1) \leq min(x_2)$ and $w(v, x_2) \leq w(v, x_1)$ then if $A(X)$ satisfies C then $A'(X)$ also. ■

Lemma 3: Assume $w(v, x_1) = A(X, x_1)$, $min(x_2) = A(X, x_2)$, x_1 is the variable s.t. $w(v, x_1) < w(v, x_2)$ and $\nexists y \in X - \{x_1, x_2\}$ with $w(v, x_1) < w(v, y) \leq w(v, x_2)$ and $\#(\geq w(v, x_2), A(X)) < R[k]$ and $T[k] = w(v, x_2)$. If $min(x_1) \leq min(x_2)$ then the assignment $A'(X)$ with $min(x_1) = A'(X, x_1)$ and $w(v, x_2) = A'(X, x_2)$ and $\forall y \in X - \{x_1, x_2\} : A'(X, y) = A(X, y)$ satisfies C and has the same number of values greater than or equal to v as $A(X)$.

Proof: $A'(X)$ has obviously the same number of values greater than or equal to v as $A(X)$. $A(X)$ contains the set of value V and the values $w(v, x_1)$ and $min(x_2)$; and $A'(X)$ contains the set of value V and the values $min(x_1)$ and $w(v, x_2)$. We have $w(v, x_1) < w(v, x_2)$ and $\#(\geq w(v, x_2), A(X)) < R[k]$ and $T[k] = w(v, x_2)$ hence by exchanging $w(v, x_2)$ and $w(v, x_1)$ the assignment remains a solution. ■

We prove by induction that it is enough to assign $w(v, x)$ to x when x has to be assigned and $D(x) = \{min(x), w(v, x)\}$. It is obviously true when there is only one variable (if the min value is taken then the obtained assignment has less value greater than v than when w is taken).

Thus, consider that the current variable that has to be assigned within the greedy algorithm is x, with $D(x) = \{min(x), w(v, x)\}$. Suppose that we assign $min(x)$ to x. We will show that we can obtain an equivalent or "better" solution (which maximizes the number of values greater than v) by taking $w(v, x)$. After assigning $min(x)$ to x, the greedy algorithm is continued and an assignment $A(X)$ is computed. This assignment satisfies the OrderedDistribute constraint. Now, if we impose $A(X, x) = w(v, x)$ and if the assignment remains a solution then the current solution is improved and it is better to assign x with $w(v, x)$. Therefore, we consider that this swap between $min(x)$ and $w(v, x)$ is not possible. Consider Y the set of variables assigned after x. Note that, by definition of the greedy algorithm any variable $y \in Y$ satisfies $min(y) \leq min(x)$. Then, we have two possible cases:

- There exists a variable $y \in Y$ with $w(v, y) \geq w(v, x)$. Then, Lemma 2 can be applied. This means that we can safely assign $w(v, x)$ to x.
- Each variable $y \in Y$ satisfies $w(v, y) < w(v, x)$. We consider the one with the w value which is the closest to $w(v, x)$ (that is, we define $z \in Y$ satisfying $\nexists y \in Y - \{z\}$ with $w(v, z) < w(v, y) < w(v, x)$). At the moment where x has been assigned, $min(x)$ and $w(v, x)$ were in its domain, therefore we had $\#(\geq w(v, x), Ap(X)) < R[k]$ and $T[k] = w(v, x)$, where $Ap(X)$ was a partial assignment. By the absence of variable of Y with $w(v, y) = w(v, x)$ and by the definition of z it means that this property still holds for $A(X)$. Therefore, Lemma 3 can be applied and we can safely assign x to $w(v, x)$.

Thus, in all the cases it is safe to assign x to $w(v, x)$ and this leads to an equivalent or better solution.

Property 2: The greedy algorithm can be applied for each value v of T with an overall time complexity in $O(nk + k^2)$.

Proof: (Sketch) The complexity for one value v depends on the computation of $w(v, x)$ for each variable and the double sorts (check of consistency can be done in constant time each time a variable is fixed). Consider $n = |X|$ and $k = |T|$. Each sort can be performed in $O(k)$ by a counting sort. The computation of $w(v, x)$ can be done for each variable in $O(\log(k))$. So for one value v we obtain a complexity in $O(n \log(k) + k)$. However, when running the greedy algorithm for each value v we can amortize some computations: All the w values for a variable x can be computed in $O(k)$ by traversing the domain while v is increased. Since there are k values to consider, the overall complexity is $O(nk + k^2)$. ■

VIII. Conclusion

In this paper we presented a new global constraint, OrderedDistribute, which solves a practical modelling issue with respect to problems involving cost variables with strongly ordered domains. This constraint is complementary to global constraints based on statistics, such as Spread or Deviation. OrderedDistribute addresses those problems where variables can be carved in disjoint subsets, to control in a very precise way the number of occurrences of cost values within each subset. We provided a linear GAC filtering algorithm for OrderedDistribute. We experimented successfully our global constraint on a cumulative problem with over-loads.

References

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.

[2] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. *Proc. ECAI*, pages 146–150, 2004.

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

[4] G. Pesant and J.-C. Régin. Spread: A balancing constraint based on statistics. *Proc. CP*, pages 460–474, 2005.

[5] T. Petit and E. Poder. Global propagation of side constraints for solving over-constrained problems. *To appear in the Annals of 0perations Research*, 2010.

[6] T. Petit and J-C. Régin. The ordered global cardinality constraint. Research report 09-07-INFO, École des Mines de Nantes, 2009.

[7] C.-G. Quimper. Enforcing domain consistency on the extended global cardinality constraint is np-hard. Technical Report CS-2003-39, School of Computer Science, University of Waterloo, 2003.

[8] C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the *global cardinality* constraint. *Proc. CP*, 2004.

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

[10] J.-C. Régin and C. Gomes. The cardinality matrix constraint. *Proc. CP*, pages 572–587, 2004.

[11] P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. The deviation constraint. *Proc. CPAIOR*, 4510:260–274, 2007.

[12] P. Schaus, P. Van Hentenryck, and J-C. Régin. Scalable load balancing in nurse to patient assignment problems. *Proc. CPAIOR*, 5547:248–262, 2009.