The Ordered Global Cardinality Constraint

TR Mines Nantes 09/7/INFO

Thierry Petit[1] and Jean-Charles Régin[2]

[1]École des Mines de Nantes, LINA UMR CNRS 6241,
4, rue Alfred Kastler, FR-44307 Nantes, France.
[2] Université de Nice-Sophia Antipolis,
930, route des colles, BP 145 06903 Sophia Antipolis cedex, France.
thierry.petit@emn.fr, regin@polytech.unice.fr

Abstract. Global constraints involving cost variables provide an expressive modelling tool for optimization problems. These constraints are used to characterize solutions which have a practical interest. When costs represent undesirable quantities, *e.g.*, over-loads of resource in a scheduling cumulative problem, their values are generally totally ordered. A great value is at least as undesirable as a lower value. This has an impact on the characterization of solutions. Some constraints related to the occurrences of values within a set of cost variables cannot be formulated efficiently using global constraints of the literature. To solve this issue, this report presents a new global constraint which takes account of this ordering *semantically*. We motivate our work by a concrete example. We design an efficient technique for enforcing arc-consistency on our global constraint. We present an extension of this constraint and its pruning procedure.

1 Introduction

This paper presents a new global constraint that solves an important modelling issue with respect to problems which involve cost variables. We provide a filtering algorithm, which achieves arc-constency in $O(|X|+|T|)$, where $|X|$ is the number of variables and $|T|$ the total number of values present in domains.

Encoding problems with Constraint Programming (CP) often requires to define global constraints that relate to a set of cost variables. For instance, the objective criterion of an optimization problem may be a sum of costs. Even if this sum constraint is internally implemented so as to be effective, it is generally mandatory to define explicitly variables for the costs. Costs represent local quantities that may be involved in many other constraints. Industrial problems often require of their solutions that they satisfy some side constraints, which are independent from the objective criteria. Recent works addressed the issue of characterizing solutions that are acceptable in practice thanks to global constraints. The `Spread` and `Deviation` constraints [5, 11] enforce some balancing of values within a set of cost variables. In [7], authors present families of constraints

that are essential for modelling over-constrained problems.[1] The importance of an effective propagation of global constraints that express such side constraints is experimentally shown in [6].

This paper focuses on the distribution of values within a set of cost variables. Unlike the `Spread` constraint, we address those problems where the user needs to control in a very precise way the number of occurrences of cost values. Classical cardinality constraints may be used in some cases, *e.g.*, `Gcc` [10]. However, when costs represent undesirable quantities, their values are generally totally ordered. A large value is at least as undesirable as a lower value. Semantically, this ordering has a strong impact on the characterization of solutions, especially concerning the distribution of costs values. Classical cardinality constraints are not satisfying to characterize solutions that are acceptable in practice.

To solve this modelling issue, we present `OrdGcc`, a new global cardinality constraint which takes account of this ordering *semantically*. Section 2 presents our motivations through a concrete example, and defines formally the `OrdGcc` constraint. In section 4, we present the pruning algorithm of this global constraint, which achieves a complete pruning in $O(|X| + |T|)$. In section 5, we discuss an extension of our constraint. We provide for this extension a complete pruning algorithm in $O(|X| + |T|)$. At last, we conclude.

2 The `OrdGcc` Global Constraint

Constraints related to occurrences of values within a set of variables form a very useful modelling toolkit in CP. To constrain the number of occurrences of a given value, one may use the `AtLeast` and `AtMost` constraints [4].

Definition 1 (AtMost, AtLeast). *Let N be a variable, v a value and X a set of variables. Given an assignment $\mathscr{A}$ of values to all variables in X, and n the value assigned to N:*

- `AtLeast`(X, N, v) *is satisfied iff v satisfies the property to appear in $\mathscr{A}$ a number of times greater than or equal to n.*
- `AtMost`(X, N, v) *is satisfied iff v satisfies the property to appear in $\mathscr{A}$ a number of times less than than or equal to n.*

Many solvers propose simplified versions of `AtLeast` and `AtMost`, defined with a range or even a simple value, instead of the variable N in Definition 1. In [10], Régin proposed an efficient GAC algorithm (*i.e.*, enforcing a complete pruning) for the global cardinality constraint, which represent a conjunction of `AtLeast` and `AtMost` constraints.

Definition 2 (Gcc). *In a* `Gcc`(X, T, I)*, X is a set of variables. T is an array containing values that can be assigned to variables in X, and I is the array*

[1] 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.

of allowed integer ranges for the number of occurrences of each value in an assignment e of X. I and T are one-to-one mapped. We use the index $i \in \{0, \ldots, |T| - 1\}$ to obtain the range $I[i]$ of value $T[i]$. Given an assignment $\mathscr{A}$ of values to all variables in X, the constraint is satisfied iff any value $v = T[i]$ satisfies the property to appear in $\mathscr{A}$ a number of times which belongs to $I[i]$.

When X is a set of cost variables, requiring that any solution should contain "at most k times value v" implies often implicitly that "at most k times value v *or any other value greater than v*" are accepted (because cost values represent undesirable quantities). An efficient CP model with global constraints on the occurrences of values should take into account this total order. Using `Gcc`'s is not sufficient. To emphasize this issue, we consider the following time tabling example. This is a cumulative problem where some over-loads of resource are tolerated, in order to produce a solution applicable in practice for instances where the total limit in time is fixed, even if this limit is too short.

Example 1. Consider one resource (for instance a team of employees) and a set of activities, each with a fixed duration (w.l.o.g.) and consuming a given amount of this resource. For each point in time i, h_i is the cumulated integer height of activities overlapping i. If this quantity is greater than a given unique fixed capacity max_capa, then there is an over-load of resource. This is expressed at a given point i by a variable $cost_i$ and a constraint: $cost_i = max(0, h_i - max\_capa)$.

Assume that we fix the makespan to 90 points in time (corresponding for instance to one week of work). This may lead quite quickly to a problem where any solution entails some over-loads. Suppose that realistic solutions are characterized by splitting the total duration in 6 sets of 15 points in time (for instance, 6 days, each point in time corresponding to one half-hour): 0 to 14, 15 to 29, etc. The user defines a solution of the problem as an ordering of activities which satisfies the following side constraints, for every range of 15 points in time.

1. At most 10 over-loads per range.
2. At most 6 over-loads greater than or equal to 2.
3. At most 3 over-loads greater than or equal to 3.
4. At most 1 over-load equal to 4.
5. No over-load should exceed value 4.

To solve the problem of example 1, scheduling the activities and satisfying the constraint 5. can be done thanks to one occurrence of the `Cumulative` constraint [2] (on the whole problem). Minimizing effectively over-loads can be achieved with one occurrence of `SoftCumulativeSum` constraint [6].

Side constraints 1. to 4. require to handle globally cardinalities of cost values (the over-loads) within each range of 15 points in time. These rules characterize precisely the distribution of over-loads in the schedule. One may try to encode these rules thanks to some `Gcc`'s, one per range of 15 points in time.

In this example, in each range a cost variable is associated with each point in time.[2] Thus, the `Gcc` involves 15 cost variables. Concerning rules 1. to 4., ranges

[2] In real problems costs may be rather attached to intervals of points in time, but w.l.o.g. we present an example which is as simple as possible. Investigating how the costs are linked to a given problem is out of the scope of this report.

in I are $[5, 15]$ for value 0, $[0, 10]$ for value 1, $[0, 6]$ for value 2, $[0, 3]$ for value 3, $[0, 1]$ for value 4. This is not a convenient solution.

Definition 3 (Cumulated profile of a ground solution). *In a cumulative problem, the* Cumulated Profile *$CumP$ of a solved schedule is the cumulated resource consumption, over time, of all the activities. We denote by A the set of activities, $start_a$ the value assigned for the start of activity a, d_a its duration and res_a the fixed amount of resource consumed by a. For a given point in time i, the height of $CumP$ at time i is equal to*

$$\sum_{a \in A / i \in [start_a, start_a + d_a[} res_a$$

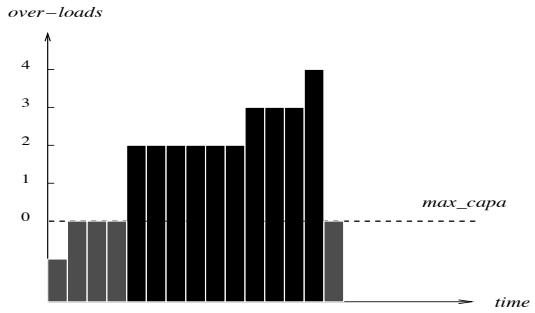

Fig. 1. The solution with values $(0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, 0)$ w.r.t. cost variables X (the over-loads) strongly violates rules 2 and 3 although it satisfies the constraint $\texttt{Gcc}(X, T, I)$ with $T = (0, 1, 2, 3, 4)$ and $I = ([5, 15], [0, 10], [0, 6], [0, 3], [0, 1])$.

As depicted by the draw of the cumulated consumption profile of all activities in figure 2, the solution $(0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, 0)$ satisfies the `Gcc` constraint while it violates strongly rules 2 and 3. To express such rules, ranges in I should be also constrained so as to express dependencies between values. Unfortunately, propagating a `Gcc` where ranges in I are represented by variables is NP-Hard [8]. As far as we know the only possible alternative might be to use a set of `AtMost` constraints (one per cost value), each defined with a variable for the range of cardinality, and to add some arithmetical constraints on these variables representing ranges, in order to express the rules (as it is shown in example 2).

Example 2. Consider the example 1, and, for instance, the first range of 15 points in time: variables $X = \{x_0, \ldots, x_{14}\}$ are points in time 0 to 14. Rules 1. to 4. can be fully expressed by the following conjunction of constraints:

- $\texttt{AtLeast}(X, N_0, 0)$ with the domain $\{5, \ldots, 15\}$ for N_0
- $\texttt{AtMost}(X, N_1, 1)$ with the domain $\{0, \ldots, 10\}$ for N_1

- $\texttt{AtMost}(X, N_2, 2)$ with the domain $\{0, \ldots, 6\}$ for N_2
- $\texttt{AtMost}(X, N_3, 3)$ with the domain $\{0, \ldots, 3\}$ for N_3
- $\texttt{AtMost}(X, N_4, 4)$ with the domain $\{0, \ldots, 1\}$ for N_4
- $[N_3 + N_4 \leq 3] \wedge [N_2 + N_3 + N_4 \leq 6] \wedge [N_1 + N_2 + N_3 + N_4 \leq 10]$

With respect to the efficiency of the solving process, this is a poor model. The lack of a global cardinality constraint which takes into account the total order on values is penalizing. This issue occurs in many other problems where one have to deal with cardinalities of cost values that are totally ordered. Therefore, we present `OrdGcc`, a new global cardinality constraint that takes into account *semantically* a total order on values. We focus on the definition of the global constraint which is useful in practice: low costs are prefered to higher ones and the only desirable cost is the lowest one (*e.g.,* value 0 in the example, which means that there is no over-load at this point in time).

Definition 4 (OrdGcc). *In a* $\texttt{OrdGcc}(X, T, I_{max}, min_{\perp})$*,*

- *X is a set of variables and T is an array containing values that can be assigned to variables in X. Values in T are totally ordered and sorted from the lowest to the highest.*
- *I_{max} is an array of maximum possible number of occurrences of values, one to one mapped with T. We use the index $i \in \{0, \ldots, |T| - 1\}$ to obtain the number $I_{max}[i]$ corresponding to value $T[i]$.*
- *$min_{\perp}$ is a value, corresponding to the possible minimum number of occurrences of $T[0]$, the minimum value in T.*

Given an assignment $\mathscr{A}$ of values to all variables in X, the $\texttt{OrdGcc}(X, T, I_{max}, min_{\perp})$ *is satisfied iff the two following constraints are satisfied.*

1. *For each $i \in \{0, \ldots, |T| - 1\}$, the number of values v in $\mathscr{A}$ s.t. $v \geq T[i]$ is at most equal to $I_{max}[i]$.*
2. *The number times value $T[0]$ appears in $\mathscr{A}$ is at least equal to $min_{\perp}$.*

Example 3. Consider the example 1, and, for instance, the first range of 15 points in time: variables $X = \{x_0, \ldots, x_4\}$ represent points in time 0 to 14. The rules 1. to 4. can be fully expressed by one $\texttt{OrdGcc}(X, T, I_{max}, min_{\perp})$, with $T = [0, 1, 2, 3, 4]$, $min_{\perp} = 5$ and $I_{max} = [15, 10, 6, 3, 1]$. The solution $(0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, 0)$ depicted by Figure 2 violates the constraint, since more than 6 values are greater than or equal to 2.

3 Feasibility

This section presents a $O(|X|)$ consistency check for the `OrdGcc`.

Notation 1 *$D(x)$ is the domain of variable $x \in X$. We call $|X|$ the number of variables in X and $|T|$ the number of values in T. Given an assignment $\mathscr{A}$ of values to variables in X, $\mathscr{A}[x]$ denotes the value of variable x in $\mathscr{A}$.*

By Definition 4 we have the two following lemmas.

Lemma 1. *If* $\texttt{OrdGcc}(X, T, I_{max}, min_{\perp})$ *has a solution then* $I_{max}[0] \geq |X|$.

Lemma 2. *If* $\texttt{OrdGcc}(X, T, I_{max}, min_{\perp})$ *has a solution then: given* $k \geq 0, I_{max}[i] \geq I_{max}[i+k]$.

We define the assignment with all minimum values of domains.

Definition 5 (Min-covering assignment). *The min-covering assignment of a set of variables* X *is the unique assignment* $\mathscr{A}$ *of values to variables in* X *s.t.* $\forall x \in X$, $\mathscr{A}[x]$ *is equal to the minimum value in* $D(x)$.

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

Notation 2 *We denote by* $\#(\geq v, \mathscr{A})$ *the number of values* $w \geq v$ *that appear in* $\mathscr{A}$.

Proposition 1. *Let* $\mathscr{A}$ *be the min-covering assignment of an* $\texttt{OrdGcc}$ C. *The two propositions are equivalent:*

1. *The number of variables* $x \in X$ *s.t.* $\mathscr{A}[x] = \perp$ *is greater than or equal to* $min_{\perp}$ *and* $\forall T[i] \in T$, $\#(\geq T[i], \mathscr{A}) \leq I_{max}[i]$.
2. C *has a solution.*

Proof. ($\Rightarrow$) Suppose that 1. is satisfied. By definition 4, $\mathscr{A}$ is a solution of C. ($\Leftarrow$) Suppose that C has a solution. By absurd, assume that the min-covering assignment $\mathscr{A}$ of C does not satisfy proposition 1. Two cases are (mutually) possible: (1) The number of times $\perp$ is assigned to a variable in $\mathscr{A}$ is strictly less than $min_{\perp}$. By Definition 5, any variable x s.t. $\perp \in D(x)$ is assigned with $\perp$ in $\mathscr{A}$. Therefore, no other assignment can have a greater number of occurrences of $\perp$ and thus satisfy C, a contradiction. (2) Assume that a value $T[i]$ is s.t. $\#(\geq T[i], \mathscr{A}) > I_{max}[i]$. By definition 5, if a value greater than $\perp$ is assigned to a variable x in $\mathscr{A}$, this value is the minimum of $D(x)$. Variables x in $\mathscr{A}$ that take value $T[i]$ cannot take a value strictly less than $T[i]$. No assignment exists with a lower value for $\#(\geq T[i], \mathscr{A})$. C has no solution, a contradiction. □

From Proposition 1 and Definition 5, the feasibility of an $\texttt{OrdGcc}$ can be checked in $O(|X|)$.

4 Filtering algorithm

This section presents two filtering algorithms for $\texttt{OrdGcc}$, which achieves GAC in $O(|X| + |T|)$.

Notation 3 *Given a set of variables* X, $X_{\perp} = \{x \in X \text{ s.t. } \perp \in D(x)\}$.

The following corollary underlines the main consistency condition of proposition 1.

Corollary 1. *If* $\exists i \in \{0, \ldots, |T|-1\}$, $\#(\geq T[i], \mathcal{A}) > I_{max}[i]$ *then* `OrdGcc`$(X,$ $T,$ $I_{max},$ $min_{\perp})$ *has no solution.*

Proof. Obvious from proposition 1. □

Corollary 2. *If* $\forall i \in \{0, \ldots, |T|-1\}$, $\#(\geq T[i], \mathcal{A}) < I_{max}[i]$ *then* $\forall x \in X$, $\forall v \in D(x)$, (x, v) *is consistent*[3] *with* `OrdGcc`$(X,$ $T,$ $I_{max},$ $min_{\perp})$.

Proof. (sketch) The assignment $\mathscr{A}'$ obtained by replacing $\mathscr{A}[x]$ by $T[i]$ in $\mathscr{A}$ is s.t. $\forall T[i] \in T, \#(\geq T[i], \mathscr{A}) \leq I_{max}[i]$. By definition 4, $\mathscr{A}'$ is a solution of C. □

To prune values it is necessary that, in the min-covering assignment $\mathscr{A}$ of an `OrdGcc`, at least one value $T[i] \in T$ occurs $I_{max}[i]$ times.

Corollary 3. *Given a feasible* `OrdGcc`$(X,$ $T,$ $I_{max},$ $min_{\perp})$, *if* $|X_{\perp}| = min_{\perp}$ *then for all* $x \in X_{\perp}$ *all the values but* $\perp$ *can be removed from* $D(x)$.

Proof. By definition 4. □

Corollary 4 provides the pruning rule that can be used when there exists $T[i] \in T$ such that $\#(\geq T[i], \mathscr{A}) = I_{max}[i]$.

Corollary 4. *Given a feasible* $C =$ `OrdGcc`$(X, T, I_{max}, min_{\perp})$ *s.t. either* $|X_{\perp}| > min_{\perp}$ *or* $\forall x \in X_{\perp}, D(x) = \{\perp\}$, *and the min-covering assignment* $\mathscr{A}$, *if* $\exists T[i] \in T, \#(\geq T[i], \mathscr{A}) = I_{max}[i]$ *then* $\forall x \in X$ *s.t.* $\mathscr{A}[x] < I_{max}[i]$, *the set of values* $v \in D(x)$ *s.t.* $v \geq T[i]$ *can be removed from* $D(x)$.

Proof. By definition 5, if in $\mathscr{A}$ a value is assigned to a variable x, this value is the minimum of $D(x)$. Therefore, if $T[i] \in T$ is s.t. $\#(\geq T[i], \mathscr{A}) = I_{max}[i]$, then given $x \in X$, $\mathscr{A}[x] < I_{max}[i]$, there exists no assignment $\mathscr{A}'$ s.t. $\mathscr{A}'[x] \geq I_{max}[i]$ and $\#(\geq T[i], \mathscr{A}') \leq I_{max}[i]$. □

Algorithm 1: Filtering Algorithm for a feasible `OrdGcc`$(X,$ $T,$ $I_{max},$ $min_{\perp})$

```
1  A ← min-covering assignment of X ;
2  X⊥ ← {x ∈ X s.t. ⊥ ∈ D(x)} ;
3  if |X⊥| = min⊥ then foreach x ∈ X⊥| do D(x) ← {⊥} ;
4  T= ← {T[i] ∈ T s.t. #(≥ T[i], A) = Imax[i]} ;
5  X' ← X ;
6  while T= ≠ ∅ ∧ X' ≠ ∅ do
7      Pick and remove the minimum value T[i] in T= ;
8      foreach x ∈ X' s.t. A[x] < Imax[i] do
9          Remove from D(x) the set {v ∈ D(x), v ≥ T[i]} ;
10         X' ← X' \ {x} ;
```

[3] That is, C has a solution with v assigned to x.

From Corollaries 3 and 4 we obtain Algorithm 1. Values in $T_=$ can be sorted in increasing order in linear time since T is ordered. Values removed from a domain $D(x)$ are necessarily strictly greater than $\mathscr{A}[x]$. It is necessary to evaluate each of these values (line 6) because some new variables can be reached when evaluating higher values in $T_=$, thanks to the condition of line 8.

This algorithm enforces GAC. Indeed, assume that a value $T[i] \in D(x)$ is not consistent with the constraint after the run of the algorithm. Since Corollary 3 has been applied, this means that assigning $T[i]$ to x entails in any complete assignment of X the existence of a value $T[j]$ such that $\#(\geq T[j], \mathscr{A}) > I_{max}[j]$ and $j \leq i$; especially in the minimum-covering assignment $\mathscr{A}$. By construction of $T_=$ and from line 9 of the algorithm, when $T[j] \in T_=$ was treated, $T[i]$ was removed from $D(x)$ by the algorithm, a contradiction.

The time complexity of this algorithm is $O(|X| + |T|)$. Indeed, the total number of times lines $8 - 10$ of the algorithm are executed is upper-bounded by $|X|$: if a variable is reached then it is removed from X' by line 10. Given $v \in D(x)$, we can remove all values greater that v in $O(1)$.

5 An Extended Definition: `Cost-OrdGcc`

5.1 Motivations and definition

In some cases, the negative impact of a given cost value v is more important when this value is assigned to some variables rather than some other variables.

For instance, in the time tabling problem of introduction (example 1), the user may say that all over-loads are twice costly on Monday and that over-loads greater than or equal to 2 are twice costly on Friday. This makes sense when the first and the last day of a week are more critical than the other days of the week. In other terms, some penalties are assigned with some values depending on the variable which takes this value.

To express this need, we define a set of penalties. Each penalty is associated with each value in each domain. Within a given domain, the penalty of a high value should be greater than or equal the one of a lower value.

Notation 4 *P is a set of penalties, of size $\sum |D(x)|, x \in X$, such that:*

- *An integer positive penalty $p(x, v)$ is associated with each value v in the domain $D(x)$ of each variable x.*
- *Given $u \in D(x)$, $v \in D(x)$ such that $u < v$, we have $p(x, u) \leq p(x, v)$.*

The interest of those penalties is to aggregate them so as to compute the global penalty of a complete asssignment. The following extension of `OrdGcc` deals with the case where an objective variable represents the *sum* of all penalties.

Definition 6 (Cost-OrdGcc).

We use notations of definition 4 and the notation 4. Moreover, obj is an objective integer variable such that $min(D(obj)) = 0$.

Given an assignment $\mathscr{A}$ *of values to all variables in* X*, the constraint* `Cost-OrdGcc`$(X, T, I_{max}, min_{\perp}, P, obj)$ *is satisfied iff the two following constraints are satisfied.*

1. `OrdGcc`$(X, T, I_{max}, min_{\perp})$ *is satisfied.*
2. $obj = \sum p(x, v), (x, v) \in \mathscr{A}$.

5.2 Feasibility

Proposition 2. *Let* $\mathscr{A}$ *be the min-covering assignment of a* `Cost-OrdGcc` C*. The two propositions are equivalent:*

1. *The three following properties are satisfied:*
   - *The number of variables* $x \in X$ *s.t.* $\mathscr{A}[x] = \perp$ *is* $k \geq min_{\perp}$.
   - $\forall T[i] \in T$, $\#(\geq T[i], \mathscr{A}) \leq I_{max}[i]$.
   - $\sum_{x \in X}(p(x, \mathscr{A}[x])) \leq max(D(obj))$.
2. *C has a solution.*

Proof. ($\Rightarrow$) Suppose that 1. is satisfied. By definition 6, $\mathscr{A}$ is a solution of C. ($\Leftarrow$) Suppose that C has a solution. By construction, $\sum_{x \in X}(p(x, v))$ is a lower bound of obj. The remaining of the proof is similar to the proof of proposition 1. □

The feasibility of a `Cost-OrdGcc` can be checked in $O(|X|)$.

5.3 Filtering algorithm

Recall that when `Cost-OrdGcc` has been checked and its feasibility is proven, $min(D(obj))$ can be updated to $\sum_{x \in X}(p(x, \mathscr{A}[x]))$.

Algorithm 2: Filtering of a feasible `Cost-OrdGcc`(X, T, I_{max}, $min_{\perp}$, obj)

```
1  𝒜 ← min-covering assignment of X ;
2  min(D(obj)) ← Σ_{(x,𝒜[x])∈𝒜} (p(x, 𝒜[x])) ;
3  foreach x ∈ X do
4  |   foreach v ∈ D(x) s.t. min(D(obj)) − 𝒜[x] + v > max(D(obj)) do
5  |   |   D(x) ← D(x) \ {v} ;
6  X_⊥ ← {x ∈ X s.t. ⊥ ∈ D(x)} ;
7  if |X_⊥| = min_⊥ then  foreach x ∈ X_⊥ do  D(x) ← {⊥} ;
8  T_= ← {T[i] ∈ T s.t. #(≥ T[i], 𝒜) = I_max[i]} ;
9  X' ← X ;
10 while T_= ≠ ∅ ∧ X' ≠ ∅ do
11 |   Pick and remove the minimum value T[i] in T_= ;
12 |   foreach x ∈ X' s.t. 𝒜[x] < I_max[i] do
13 |   |   Remove from D(x) the set {v ∈ D(x), v ≥ T[i]} ;
14 |   |   X' ← X' \ {x} ;
```

The time complexity of this algorithm is $O(|X| + |T|)$. The total number of times lines $4-5$ and 13 of the algorithm are executed is upper-bounded by $|X|$. Given $v \in D(x)$, we can remove all values greater that v in $O(1)$.

6 Conclusion

This paper presented a new global constraint, `OrdGcc` that solves an important modelling issue with respect to problems which involve cost variables. This global constraint is especially useful to encode over-constrained problems with side constraints. We provided an efficient flow-based filtering algorithm, which achieves arc-constency. We discussed an extension of this constraint where costs are associated with values in each domain. We provided the related filtering algorithm.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. Networks flows: theory, algorithms, and applications. *Prentice Hall, Inc.*, 1993.
2. N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. *Proc. CP*, pages 63–79, 2002.
3. L. Ford and D. Flukerson. Flows in networks. *Princeton University Press*, 1962.
4. Pascal Van Hentenryck and Yves Deville. The cardinality operator: A new logical connective for constraint logic programming. *WCLP*, pages 283–403, 1991.
5. G. Pesant and J.-C. Régin. Spread: A balancing constraint based on statistics. *Proc. CP*, pages 460–474, 2005.
6. T. Petit and E. Poder. Global propagation of practicability constraints. *Proc. CPAIOR*, pages 361–366, 2008.

7. T. Petit, J-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proc. IEEE-ICTAI*, pages 358–365, 2000.
8. 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.
9. C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the *global cardinality* constraint. *Proc. CP*, 2004.
10. J-C. Régin. Generalized arc consistency for global cardinality constraint. *Proc. AAAI*, pages 209–215, 1996.
11. Pierre Schaus, Yves Deville, Pierre Dupont, and Jean-Charles Régin. The deviation constraint. *Proc. CPAIOR*, pages 260–274, 2007.

Appendix

This appendix present a flow-based algorithm for `OrdGcc` derived from the filtering algorithm of `Gcc` [10]. Even if its complexity is higher that the algorithm we presented in the report, this algorithm is of interest because it can be extended to the generalization of `OrdGcc` to the case where several cost values are desirable (and not only one, as $\perp$ in `OrdGcc`). Section 8 presents the generalization of `OrdGcc` and the principle of its pruning algorithm.

7 Flow-based algorithm for `OrdGcc`

7.1 Related work

An algorithm in $O(|X|^2|T|)$ for checking the consistency and removing inconsistent values in a `Gcc` is presented in [10], based on a flow on a particular variable-value graph.

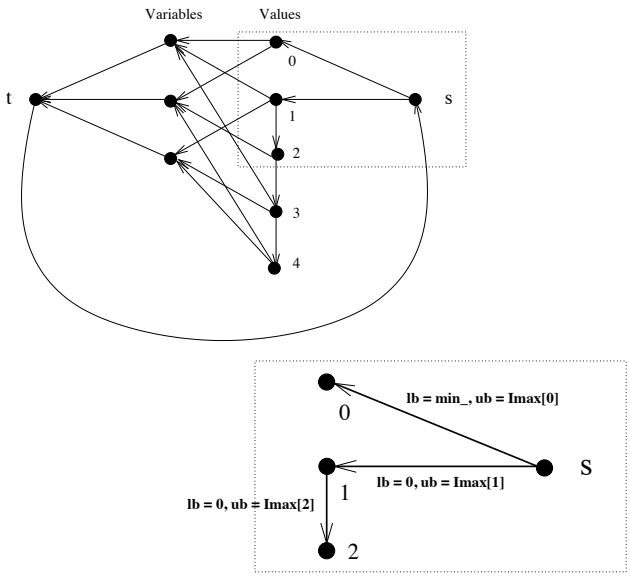

Fig. 2. Example of a digraph representing an `OrdGcc`(X, T, I_{max}, $min_{\perp}$). Given an arc, lb and ub are the lower bound and upper bound capacities [3]. In this graph lower and upper bound capacities of arc (t, s) are equal to the number of variables, and arcs between values and variables are s.t. $lb = 0$ and $ub = 1$.

It is possible to use a similar algorithm[4] for the `OrdGcc` (see Figure 2). Compared with the `Gcc`, using the linear feasibility check of section 3 allows to make the global constraint GAC in $O(|X||T|)$. Next paragraphs present the details of this approach.

7.2 Background

A *digraph* $\mathscr{G} = (\mathscr{X}, \mathscr{U})$ consists of a vertex set $\mathscr{X}$ and a set of arcs $\mathscr{U}$, where every arc (u, v) is a directed pair of distinct vertices. An arc (u, v) *leaves* u and *enters* v. $\Gamma^-(v)$ is the set of edges entering a vertex v. $\Gamma^+(v)$ is the set of edges leaving v. $\Gamma(v) = \Gamma^- \cup \Gamma^+$.

Let $\mathscr{G} = (\mathscr{X}, \mathscr{U})$ be a digraph s.t. each arc (u, v) is associated with two positive integers $lb(u, v)$ and $ub(u, v)$. $ub(u, v)$ is called the *upper bound capacity* of (u, v) and $lb(u, v)$ the *lower bound capacity*. A *flow* [3] in $\mathscr{G}$ is a function f satisfying the following two conditions:

1. For any arc (u, v), $f(u, v)$ represents the amount of commodity which flows along the arc. Such a flow is allowed only in the direction of the arc (u, v), that is, from u to v.
2. A *conservation law* is observed at each of the vertices. It enforces *the capacity constraint*: $\forall v \in X, \sum_{u \in \Gamma^-(v)} f(u, v) = \sum_{w \in \Gamma^+(v)} f(v, w)$.

The *feasible flow problem* is the problem of the existence of a flow in $\mathscr{G}$ which satisfies:

$\forall (u, v) \in \mathscr{U} : lb(u, v) \leq f(u, v) \leq ub(u, v)$.

The *maximum flow problem* for an arc (u, v) is the problem of existence of a feasible flow in $\mathscr{G}$ for which $f(u, v)$ is maximum. $f(u, v)$ is then called the value of the maximum flow from v to u.

The *flow integrality theorem* [3] emphasizes that if all upper and lower bound capacities are integers and if there exist a feasible flow, then for any arc (u, v) there exists a maximum flow from u to v which is integral on every arc in G. This means that each amount of commodity which flows along a given arc is integer. We consider here graphs where all capacities are integers.

7.3 Graph based representation and algorithms

Definition 7 (Digraph $\mathscr{G}_{XT}$). *Given a constraint* `OrdGcc`$(X, T, I_{max}, min_{\perp})$, *we define the digraph* $\mathscr{G}_{XT} = (X_{\mathscr{G}_{XT}}, \mathscr{U}_{\mathscr{G}_{XT}})$ *as follows.*

- $X_{\mathscr{G}_{XT}} = \{s, t\} \cup X \cup T$.
- $\mathscr{U}_{\mathscr{G}_{XT}}$ *contains:*
  - *For each* $T[i] \in T$ *and* $x \in X$, *an arc* $(T[i], x)$ *if and only if* $T[i] \in D(x)$. $lb(T[i], x) = 0$ *and* $ub(T[i], x) = 1$.

[4] Note that, conversely, the generalization of the Hopcroft and Karp algorithm for the `Gcc` presented in [9] should not be transposable to the `OrdGcc` because in the variable-value graph some arcs are defined between value nodes

- *For each* $i \in \{1, \ldots, |T|-2\}$, *an arc* $(T[i], T[i+1])$. $lb(T[i], T[i+1]) = 0$ *and* $ub(T[i], T[i+1]) = I_{max}[i+1]$.
- *An arc* (x, t) *for every variable* $x \in X$. *For such an arc,* $lb(x, t) = 0$ *and* $ub(x, t) = 1$.
- *An arc* $(s, T[0])$ *with* $lb(s, T[0]) = min_\perp$ *and* $ub(s, T[0]) = I_{max}[0]$.
- *An arc* $(s, T[1])$ *with* $lb(s, T[1]) = 0$ *and* $ub(s, T[1]) = I_{max}[1]$.
- *An arc* (t, s) *with* $lb(t, s) = |X|$ *and* $ub(t, s) = |X|$.

Proposition 3. *Given* $C = \texttt{OrdGcc}(X, T, I_{max}, min_\perp)$ *and its corresponding digraph* $\mathscr{G}_{XT}$, *the two following properties are equivalent:*

- $\texttt{OrdGcc}(X, T, I_{max}, min_\perp)$ *has a solution.*
- *There exists a feasible flow from* t *to* s *in* $\mathscr{G}_{XT}$.

Proof. ($\Rightarrow$) Suppose C is consistent. There exist at least one assignment of values to all variables in X satisfying C. Let $\mathscr{A}$ be such an assignment. Given a value v, $\#(v, \mathscr{A})$ is the number of occurrences of v in $\mathscr{A}$. We denote by $\mathscr{A}[x]$ the value of variable x in $\mathscr{A}$. From $\mathscr{A}$, we can build in $\mathscr{G}_{XT}$ a function f such that:
(1) $\forall x \in X, f(x, t) = 1$ (any variable has a value).
(2) $\forall v \in T, \forall x \in X, f(v, x) = 1$ if $\mathscr{A}[x] = v$, otherwise $f(v, x) = 0$.
(3) $f(s, T[0]) = \#(0, \mathscr{A})$ and $f(s, T[1]) = \#(1, \mathscr{A})$.
(4) $\forall i \in \{1, \ldots, |T|-2\}$, $f(T[i], T[i+1]) = \sum_{j \in \{i+1, \ldots, |T|\}} \#(T[j], \mathscr{A})$.
f satisfies the capacity constraint, otherwise by definition 7 the assignment $\mathscr{A}$ violates C, a contradiction. By construction, f satisfies the conservation law and $f(t, s) = |X|$. The flow saturates all arcs entering t.

($\Leftarrow$) Suppose there exists a feasible flow from t to s in $\mathscr{G}_{XT}$. Since $lb(t, s) = |X|$ and $ub(t, s) = |X|$, this flow is maximum for (t, s). By the flow integrality theorem there exists a maximum flow f from t to s which is integral on any arc in $\mathscr{G}_{XT}$. We have: (A) Since the value of f is $|X|$, $f(x, t) = 1$ for every $x \in X$. (B) By the conservation law and since f is integral on any arc, for each $x \in X$ there is exactly one value v in T s.t. $f(v, x) = 1$. The other arcs entering x have a flow equal to 0. From (A) and (B), the set $\mathscr{A} = \{(x, v), x \in X, v \in T, f(v, x) = 1\}$ forms a complete assignment of values to variables in X. By the conservation law $f(s, T[0]) = \sum_{x \in X}(f(x, T[0]))$ since no arc but $(s, T[0])$ is entering $T[0]$. The same reasoning leads to $f(s, T[1]) = \sum_{x \in X}(f(x, T[1]))$. For $T[0]$ and $T[1]$, by the capacity constraints, $min_\perp \leq f(s, T[0]) \leq I_{max}[T[0]]$ and $0 \leq f(s, T[1]) \leq I_{max}[T[1]]$. Moreover, given an arc $(T[i], T[i+1])$, $i \in \{1, \ldots, |T|-2\}$, by the capacity constraint the flow does not exceeds $I_{max}[T[i+1]$. Therefore, by construction the number of arcs outgoing a vertex $T[i+1]$ is less than $I_{max}[T[i+1]]$. This number of arcs is the number of times $\mathscr{A}$ contains value $T[i+1]$. The constraint is satisfied. □

From Propositions 1 and 3, if the constraint has a solution then the min-covering assignment, which is s.t. the number of variables $x \in X$ s.t. $\mathscr{A}[x] = \perp$ is greater than or equal to $min_\perp$ and, $\forall T[i] \in T$, $\#(\geq T[i], \mathscr{A}) \leq I_{max}[i]$, can be directly translated into a feasible flow in $\mathscr{G}_{XT}$. This feasible flow can thus be obtained in $O(|X|)$. Given such a feasible flow, a pruning algorithm can be written using the residual graph of this flow, as for the `Gcc` [10].

Definition 8 (Residual graph [1]). *The residual graph $R(f)$ of a flow f in $\mathscr{G}_{XT}$ is the digraph with the same vertex set as $\mathscr{G}_{XT}$ and the following arcs stem from arcs in $\mathscr{G}_{XT}$. $\forall (i,j) \in \mathscr{U}_{\mathscr{G}_{XT}}$:*

- *$f(i,j) < ub(i,j) \Leftrightarrow (i,j) \in \mathscr{U}_{R(f)}$ with a new upper bound capacity equal to $ub(i,j) - f(i,j)$, and a lower bound capacity equal to 0.*
- *$f(i,j) > lb(i,j) \Leftrightarrow (j,i) \in \mathscr{U}_{R(f)}$ with a new lower bound capacity equal to $f(i,j) - lb(i,j)$, and a lower bound capacity equal to 0.*

Corollary 5. *Given an arbitrary feasible flow f in the digraph $\mathscr{G}_{XT}$ representing a* `OrdGcc`*, if the two following conditions are satisfied for a value $T[i] \in T$ and a variable x:*

- *$f(T[i], x) = 0$)*
- *x and $T[i]$ do not belong to the same strongly connected component in $\mathscr{G}_{XT}$,*

then $T[i]$ can be removed from $D(x)$.

Proof. By proposition 3, if f exists then the constraint has a solution. By definition 7, any feasible flow in $\mathscr{G}_{XT}$ is maximum for (t,s). Consider a feasible flow f and an arc $(T[i],x)$, $T[i] \in T$, $x \in X$, s.t. $f(T[i],x) = 0$. By construction of $R(f)$, $(T[i],x)$ is not in the set of arcs of $R(f)$. From [10] (Theorem 4.), if $T[i]$ and x are not in the same strongly connected component then there is no flow f' with $f'(T[i],x) = 0$. By proposition 3, $T[i]$ is not consistent with `OrdGcc`. □

This leads to a polynomial GAC algorithm. A feasible flow can be built in $O(|X|)$, and the search for strongly connected components can be done in $O(|X||T|)$.

8 Generalized `OrdGcc`

8.1 Motivations and definition

A natural generalization of Definition 4 consists of defining a set of minimum cardinalities I_{min} for all values, instead of considering a single desirable cost value $min_\perp$.

Even is its practical usefulness is rare compared with the `OrdGcc` global constraint, this generalization can be helpful in problems where several cost values are desirable. The `OrdGcc` deals with problems where only the bottom value $min_\perp$ is desirable. For instance, a violation cost $min_\perp = 0$ in an over-constrained problem. In some other problems, several cost values may be desirable. Semantically, between two desirable values v and w, the lower one, for instance v, is the *prefered* one and should be favoured. Nevertheless w is not undesirable. Defining a minimum number of occurrences for this second value (or any value lower than w) makes sense.

Definition 9 (GenOrdGcc).

We use notations of definition 4 plus an array I_{min} of minimum required number of occurrences of values, one to one mapped with T. We use the index $i \in \{0, \ldots, |T|-1\}$ to obtain the number $I_{min}[i]$ corresponding to value $T[i]$.

Given an assignment $\mathscr{A}$ of values to all variables in X, the constraint `GenOrdGcc`(X, T, I_{max}, I_{min}) *is satisfied iff the two following constraints are satisfied.*

1. *For each $i \in \{0, \ldots, |T|-1\}$, the number of values v in $\mathscr{A}$ s.t. $v \geq T[i]$ is at most equal to $I_{max}[i]$.*
2. *For each $i \in \{0, \ldots, |T|-1\}$, the number of values v in $\mathscr{A}$ s.t. $v \leq T[i]$ is at least equal to $I_{min}[i]$.*

By construction, values in I_{min} are increasing. Given $i \in \{0, \ldots, |T|-1\}$, $I_{min}[i]$ can be greater than $I_{max}[i]$.

8.2 Discussion : Feasibility and filtering algorithm

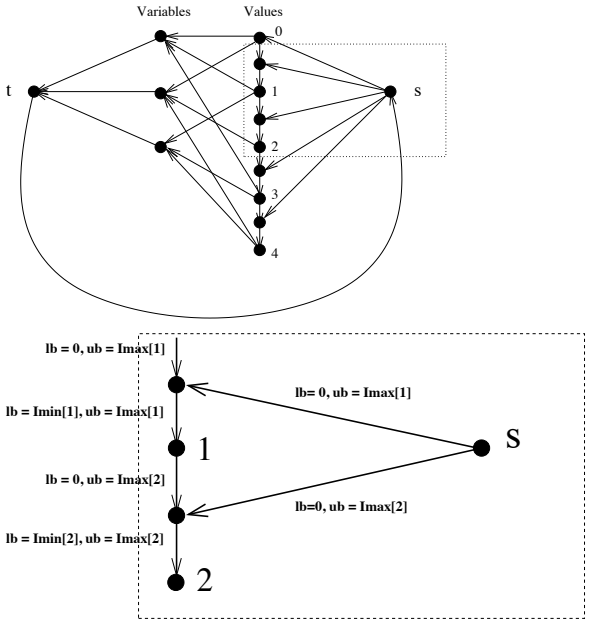

Fig. 3. Example of digraph representing a `GenOrdGcc`(X, T, I_{max}, I_{min}).

A filtering algorithm achieving arc-consistency can be defined using the digraph depicted by Figure 3.

Definition 10 (Digraph $\mathscr{G}_{XTT'}$). *Given a constraint* `GenOrdGcc`(X, T, I_{max}, I_{min})*, we define the digraph $\mathscr{G}_{XTT'} = (X_{\mathscr{G}_{XTT'}}, \mathscr{U}_{\mathscr{G}_{XTT'}})$ as follows.*

- $X_{\mathscr{G}_{XTT'}} = \{s,t\} \cup X \cup T \cup T'$, *where* T' *is an new array of vertices s.t.* $T' = |T| - 1$.
- $\mathscr{U}_{\mathscr{G}_{XTT'}}$ *contains:*
  - *For each* $T[i] \in T$ *and* $x \in X$, *an arc* $(T[i], x)$ *if and only if* $T[i] \in D(x)$. $lb(T[i], x) = 0$ *and* $ub(T[i], x) = 1$.
  - *For each* $i \in \{0, \ldots, |T| - 2\}$, *an arc* $(T[i], T'[i])$. $lb(T[i], T'[i]) = 0$ *and* $ub(T[i], T'[i]) = I_{max}[i+1]$.
  - *For each* $i \in \{1, \ldots, |T| - 1\}$, *an arc* $(T'[i-1], T[i])$. $lb(T'[i-1], T[i]) = I_{min}[i]$ *and* $ub(T'[i-1], T[i]) = I_{max}[i]$.
  - *For each* $i \in \{0, \ldots, |T'| - 1\}$, *an arc* $(s, T'[i])$. $lb(s, T'[i]) = 0$ *and* $ub(s, T'[i]) = I_{max}[i+1]$.
  - *An arc* (x, t) *for every variable* $x \in X$. *For such an arc,* $lb(x, t) = 0$ *and* $ub(x, t) = 1$.
  - *An arc* $(s, T[0])$ *with* $lb(s, T[0]) = I_{min}[0]$ *and* $ub(s, T[0]) = I_{max}[0]$.
  - *An arc* (t, s) *with* $lb(t, s) = |X|$ *and* $ub(t, s) = |X|$.

The principle is similar to the flow-based algorithm of the `OrdGcc` global constraint, leading to an incremental algorithm in $O(|X||T|)$ for enforcing GAC.