Generalizations of the Global Cardinality Constraint for Hierarchical Resources

Alessandro Zanarini and Gilles Pesant

Département de génie informatique
École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville
Montreal, Canada H3C 3A7
{azanarini,pesant}@crt.umontreal.ca

Abstract. We propose generalizations of the Global Cardinality Constraint (gcc) in which a partition of the variables is given. In the context of resource allocation problems, such constraints allow the expression of requirements, in terms of lower and upper bounds, for resources with different capabilities. Alternate models using gcc's are shown to be weaker. We present filtering algorithms based on flow theory that achieve domain consistency and give experimental evidence of the usefulness of such constraints. We consider an optimization version of the constraints and discuss its relationship with the cost_gcc.

1 Introduction

Resource allocation problems occur in many real-life problems whenever it is necessary to assign resources to tasks that need to be accomplished. It can be thought of as a one to one assignment or, more generally, a many to one relation in which tasks can be assigned one or more resources. Typically, for each task a minimum and maximum number of required resources is defined. Resources may be *homogeneous* in the sense that they have identical capabilities or skills. In Constraint Programming, problems with homogeneous resources can be easily modeled by a Global Cardinality Constraint [6] (gcc) in which each resource is represented by a variable whose domain is the set of tasks and each task defines its resource requirements through the bounds on the number of occurrences in the definition of the constraint. However for some real-world problems this scenario is too simplistic: resources are heterogeneous and tasks require resources with different capabilities or skill levels. We further distinguish three cases: in the first, referred to as *disjoint heterogeneous resources*, the different skill levels are considered independently i.e. a resource with a given skill level can only satisfy requirements defined on this level; in the second, referred to as *nested heterogeneous resources*, resources are organized in a total order, that is, a resource with skill level ℓ is able to satisfy requirements of level ℓ or below; in the third, which we call *hierarchical heterogeneous resources*, the relationship between resources generalizes beyond the linear order of the nested case to a tree-like hierarchy.

Problems with disjoint heterogeneous resources are also easily modeled, this time using a set of `gcc`'s, each of them representing a single skill level as before, and domain consistency can still be achieved. Unfortunately a similar model for the nested and hierarchical cases does not guarantee domain consistency. This paper focuses on the important cases of nested and hierarchical heterogeneous resources for which we propose generalizations of the global cardinality constraint that achieve domain consistency.

Example 1. We need to accomplish two tasks $T1$ and $T2$ that have different requirements of resources of level 1 and 2. Three resources R_1^1, R_2^1, R_3^1 of level 1 and three resources R_1^2, R_2^2, R_3^2 of level 2 are available. Each resource can be assigned to any task. Both tasks $T1$ and $T2$ need between 1 and 2 resources of level 2, and between 2 and 3 resources of level 1. In a disjoint heterogeneous resources setting, resources can only satisfy requirements of their level. Since the tasks need at least 4 resources of level 1 the problem is unsatisfiable. In a nested heterogeneous resources setting, resources can satisfy requirements of their level or below. The minimum requirements of resources of level 2 is equal to 2 (one for each task). Then, one resource of level 2 can be assigned to a task for satisfying the requirements of level 1. Thus, the problem is satisfiable.
Note that in the case of nested heterogeneous resources the problem can be restated as follows: both tasks $T1$ and $T2$ need respectively between 3 and 5 resources of level 1 or higher, and among them 1 or 2 must be of level 2.

The initial motivation for this work came from a real-life manpower planning and scheduling problem proposed in [8] by France Telecom for the 2007 ROADEF Challenge. A subproblem consists of forming teams of technicians that have to accomplish a set of tasks. The technicians have different skill levels and a technician can satisfy task requirements of his level or below. Each task defines the minimum number of technicians required for each skill level. This corresponds exactly to nested heterogeneous resources. Another important application area is nurse rostering. Here a minimum number of nurses on duty is specified for each work shift and sometimes a minimum is also given for senior nurses acting in a supervisory role but who can perform the duties of regular nurses as well. Applications of hierarchical heterogeneous resources are found in the computer software industry or generally in large projects with multiskilled resources.

The paper is organized as follows: Section 2 gives a brief background of Constraint Programming and Network Flows that will be used in the following sections. In Section 3, we formally introduce the `nested_gcc`, its graph representation as well as the theoretical basis for achieving domain consistency. Section 4 is dedicated to a generalization of the `nested_gcc` called `hierarchical_gcc`. In Section 5 we show experimental evidence of the usefulness of the presented constraints. Section 6 considers an optimization version of `nested_gcc` that allows the expression of preferences. Finally, conclusions are drawn in Section 7.

2 Preliminaries

2.1 Constraint Satisfaction Problem

A *Constraint Satisfaction Problem* (CSP) consists of a finite set of variables $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ with finite domains $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ such that $x_i \in D_i$ for all i, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{X}$. A constraint $C \in \mathcal{C}$ is a subset $T(C)$ of the Cartesian product of the domains of the variables that are in C. We write $X(C)$ to denote the set of variables involved in C and we call tuple $\tau \in T(C)$ an allowed combination of values of $X(C)$. Given a set of variables $X' \subseteq X(C)$, $\tau \downarrow_{X'}$ is the projection of the tuple τ over the set X'. The number of occurrences of a value d in a tuple τ is denoted by $\#(d, \tau)$; analogously $\#(d, \tau \downarrow_{X'})$ is the number of occurrences of d in the projection of the tuple τ over the set X'. An assignment $(d_1, \ldots, d_k) \in X(C)$ satisfies a constraint C if it belongs to $T(C)$. A *solution* to a CSP is an assignment of all the variables such that it satisfies all the constraints.

In Constraint Programming (see [2]), the solution process consists of iteratively interleaving search phases and propagation phases. During the search phase all the combinations of values are evaluated. It is generally performed on a tree-like structure and each step consists of instantiating a variable to a value of its domain. In order to avoid the systematic generation of all the combinations and reduce the search space, the propagation phase shrinks the search space: each constraint propagation algorithm removes values that a priori cannot be part of a solution w.r.t. the partial assignment built so far. The propagation can be eventually performed an exponential number of times thus it needs to be efficient and effective. In order to be effective, each constraint filtering algorithm should remove as many variable domain values as possible and possibly achieve *domain consistency* (also referred to as hyper-arc consistency or generalized-arc consistency).

Definition 1 (Domain Consistency). *Given a constraint C defined on the variable set $x_1, \ldots, x_n$ with respective domains $D_1, \ldots, D_n$, the constraint is* domain consistent *iff for each variable x_i and each value $d_i \in D_i$ there exists a value $d_j \in D_j$ for all $j \neq i$ such that $(d_1, \ldots, d_n) \in T(C)$.*

2.2 Network Flows

In this section we recall the main results and definitions that will be used in the following sections (see [1] for further explanations). An oriented graph is defined as $G = (V, A)$ where V is a set of vertices and A is a set of ordered pairs (arcs) from V. We write $\delta^{out}(v)$ to refer to the set of outgoing arcs of v: $\delta^{out}(v) = \{(v, u) \,|\, (v, u) \in A\}$. Similarly, the set of ingoing arcs of v is denoted by $\delta^{in}(v) = \{(u, v) \,|\, (u, v) \in A\}$. An oriented path in a oriented graph $G = (V, A)$ is a sequence of vertices $v_0, v_1, \ldots, v_k$ such that $(v_i, v_{i+1}) \in A$ with $i = 0, \ldots, k-1$. An oriented graph is called *strongly connected* iff for each ordered pair (u, v) of vertices there exists an oriented path from u and v. A *strongly connected*

component of an oriented graph $G = (V, A)$ is a strongly connected subgraph G' of G such that no other strongly connected subgraph of G contains G'.

Let $G = (V, A)$ be an oriented graph, $l(a)$ and $c(a)$ the demand and the capacity of each arc $a \in A$ $(0 \leq l(a) \leq c(a))$. We define an *s-t flow* as a function $f : A \to \mathbb{R}$ such that:

$$\forall v \in V \setminus \{s, t\} : \sum_{a \in \delta^{out}(v)} f(a) = \sum_{a \in \delta^{in}(v)} f(a)$$

where s and t are respectively *source* and *sink* of the flow. The flow is feasible if $\forall a \in A : l(a) \leq f(a) \leq c(a)$. The value of a flow f is defined as $value(f) = \sum_{\{a \in \delta^{out}(s)\}} f(a) - \sum_{\{a \in \delta^{in}(s)\}} f(a)$. A feasible flow f is maximum if there is no other feasible flow f' such that $value(f') > value(f)$.

Theorem 1. *If all arc demands and capacities are integer and there exists a feasible flow then the maximum flow problem has an integer maximum flow.*

Given a flow f on a graph $G = (V, A)$, the residual graph is defined as $G_f = (V, A_f)$ where $A_f = \{(u, v) \in A : f((u, v)) < c((u, v))\} \bigcup \{(v, u) : (u, v) \in A, l((u, v)) < f((v, u))\}$.

3 Nested Global Cardinality Constraint

Let $\{X^k\}_{1 \leq k \leq \ell}$ represent a family of ℓ disjoint sets of variables. Define further $\mathbb{X}^k = \bigcup_{k \leq j \leq \ell} X^j$, with $\mathbb{X} = \mathbb{X}^1$ for short. Observe that this new family of sets is nested: $\mathbb{X}^\ell \subseteq \mathbb{X}^{\ell-1} \subseteq \cdots \subseteq \mathbb{X}^1$. The variables $X^k = \{x_1^k, \ldots, x_{n_k}^k\}$ are defined over the domains $D_1^k, \ldots, D_{n_k}^k$. We write D_{X^k} for the union of the domains of the variables in X^k; analogously $D_{\mathbb{X}}$ stands for the union of all the domains of the variables in $\mathbb{X}$.
We denote by l_d^k and u_d^k the lower and upper bounds on the number of occurrences of value $d \in D_{\mathbb{X}}$ among $\mathbb{X}^k$. It follows that we should have $l_d^{k+1} \leq l_d^k$ and $u_d^{k+1} \leq u_d^k$ for $k = 1, \ldots, \ell - 1$. For example $l_d^1 = 5$, $u_d^1 = 7$, $l_d^2 = 3$, $u_d^2 = 4$ means that value d occurs between 5 and 7 times in $\mathbb{X}^1$, including between 3 and 4 times in $\mathbb{X}^2$. The related vectors of lower and upper bounds are denoted by l^k and u^k for $k = 1, \ldots, \ell$.

Definition 2 (nested gcc). *The nested global cardinality constraint is formally defined as*

$$\texttt{nested\_gcc}(\mathtt{X}^1, \ldots, \mathtt{X}^\ell, (\mathtt{l}^1, \mathtt{u}^1), \ldots, (\mathtt{l}^\ell, \mathtt{u}^\ell)) =$$

$$\{\tau = (d_1^1, \ldots, d_{n_1}^1, d_1^2, \ldots, d_{n_\ell}^\ell) \,|\, d_i^k \in D_i^k, \forall 1 \leq k \leq \ell, \forall d \in D_{\mathbb{X}} : l_d^k \leq \#(d, \tau \downarrow_{\mathbb{X}^k}) \leq u_d^k\}$$

Note that it is possible to model the `nested_gcc` as a set of traditional `gcc`'s: for each set $\mathbb{X}^k$, we define a `gcc` in which we set the corresponding upper and lower bounds. But such a formulation is strictly weaker, as we shall see in Proposition 1.

Going back to our resource allocation problem, the tasks would correspond to the values and the resources to the variables, arranged in disjoint sets according to their level. As defined, the constraint requires that each resource be assigned to a task. In some problems, like rostering, it might be useful to find an assignment that, while satisfying the lower bound constraints, keeps some resources unassigned. This can be easily modeled by adding an extra value (without requirements) representing a fake activity; resources that are assigned to it are in fact unused.

Example 2. A company needs to form some teams in order to accomplish 5 tasks. Only 6 technicians with different skills are available. Three of them have skill level equal to 2 (capable of accomplishing a job requiring skill level 1 or 2) and the remaining three have a skill level equal to 1. Moreover the three technicians with basic skills are not allowed to be assigned to the task 5. We model the problem with 6 variables representing the resources divided in two sets: $X^1 = \{x_1^1, x_2^1, x_3^1\}$ and $X^2 = \{x_1^2, x_2^2, x_3^2\}$. The variable domains are respectively: $D_1^1 = D_2^1 = D_3^1 = \{d_1, d_2, d_3, d_4\}$ and $D_1^2 = D_2^2 = D_3^2 = \{d_1, d_2, d_3, d_4, d_5\}$. The tasks require respectively a minimum of 1, 1, 1, 1 and 2 technicians of skill level at least 1. Tasks 3 and 4 each need at least one technician of level 2. None of the tasks can accommodate more than 3 technicians (independently from the level). We would model this situation as $\texttt{nested\_gcc}(\{\texttt{x}_1^1, \texttt{x}_2^1, \texttt{x}_3^1\}, \{\texttt{x}_1^2, \texttt{x}_2^2, \texttt{x}_3^2\}, ([\texttt{1}, \texttt{1}, \texttt{1}, \texttt{1}, \texttt{2}], [\texttt{3}, \texttt{3}, \texttt{3}, \texttt{3}, \texttt{3}]), ([\texttt{0}, \texttt{0}, \texttt{1}, \texttt{1}, \texttt{0}], [\texttt{3}, \texttt{3}, \texttt{3}, \texttt{3}, \texttt{3}]))$. The alternate model using two `gcc`'s is illustrated at Figure 1.

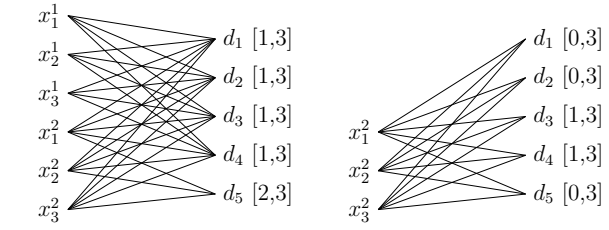

Fig. 1. Traditional GCC modelling for the Nested-GCC

Proposition 1. *Modelling the constraint* `nested_gcc` *as a set of traditional* `gcc` *does not achieve domain consistency.*

Proof. Consider Example 2. Both `gcc` constraints are domain consistent however the instance is unsatisfiable. Two variables in X^2 must take the value d_3 and d_4 (from the level 2 `gcc`), hence there is only one variable left to assign to d_5 that requires a minimum of two variables (from the level 1 `gcc`).

Even though it has been proven ([3]) that finding a consistent solution to a set of overlapping `gcc`'s is an NP-Complete problem, the particular nested structure is such that it is possible to find a consistent assignment in polynomial time as we shall see in the next section.

3.1 Graph Representation

We propose a new graph representation for the `nested_gcc`. Informally, it contains vertices representing the variables from the X^k sets and vertices that denote the values; differently from the traditional `gcc`, the value vertices are duplicated for each set X^k while variable vertices remain singletons; arcs connect successive replications of value vertices. In order to identify duplicate value vertices, for each value $d_i \in D_{\mathbb{X}}$ we add a superscript that refers to its corresponding set X^k. We write $D^k_{\mathbb{X}}$ to denote the set of values in $D_{\mathbb{X}}$ with superscript k; hence $d^k_i \in D^k_{\mathbb{X}}$ and $d^{k'}_i \in D^{k'}_{\mathbb{X}}$ represent the value d_i but two different value vertices for sets X^k and $X^{k'}$. The directed graph $G = (V, A)$ is defined as follows:

$$V = \mathbb{X} \cup \big(\bigcup_{k=1}^{\ell} D^k_{\mathbb{X}}\big) \cup \{s, t\}$$

$$A = A_s \cup \big(\bigcup_{k=1}^{\ell} A_{X^k}\big) \cup \big(\bigcup_{k=1}^{\ell} A^k_{req}\big)$$

where

$$\begin{aligned} A_s &= \{(s, x^k_i) \mid k = 1, \ldots, \ell\,,\ i = 1, \ldots, n_k\} \\ A_{X^k} &= \{(x^k_i, d^k_j) \mid i = 1, \ldots, n_k, d^k_j \in D^k_i\} \\ A^k_{req} &= \begin{cases} \{(d^1_i, t) \mid i = 1, \ldots, |D_{\mathbb{X}}|\} & \text{if } k = 1 \\ \{(d^k_i, d^{k-1}_i) \mid i = 1, \ldots, |D_{\mathbb{X}}|\} & \text{if } 2 \le k \le \ell \end{cases} \end{aligned}$$

The lower bounds and upper bounds of the arcs $a \in A_s$ are unitary; they are respectively null and unitary for the arcs $a \in A_{X^k}$. For each arc (d^k_i, v) the lower bound is equal to l^k_i and the upper bound is u^k_i.
The graphical representation of Example 2 is given at Figure 2.

3.2 Domain consistency and propagation algorithm

A feasible flow on the introduced graph representation reflects a feasible assignment of the `nested_gcc` constraint. A flow going from a variable vertex x^k_i to a value vertex d^k corresponds to the assignment $x^k_i = d$. In addition a value vertex d^k collects all the flow coming from duplicate value vertices $d^j, j \ge k$, which means it receives a flow equal to the number of assignments of d to variables in $\mathbb{X}^k$. For a given value vertex, the bounds on the single outgoing arc constraint the number of occurrences according to the definition of the constraint, by construction.

Theorem 2. *There is a bijection between solutions to the* `nested_gcc` *and feasible flows in the related graph representation G.*

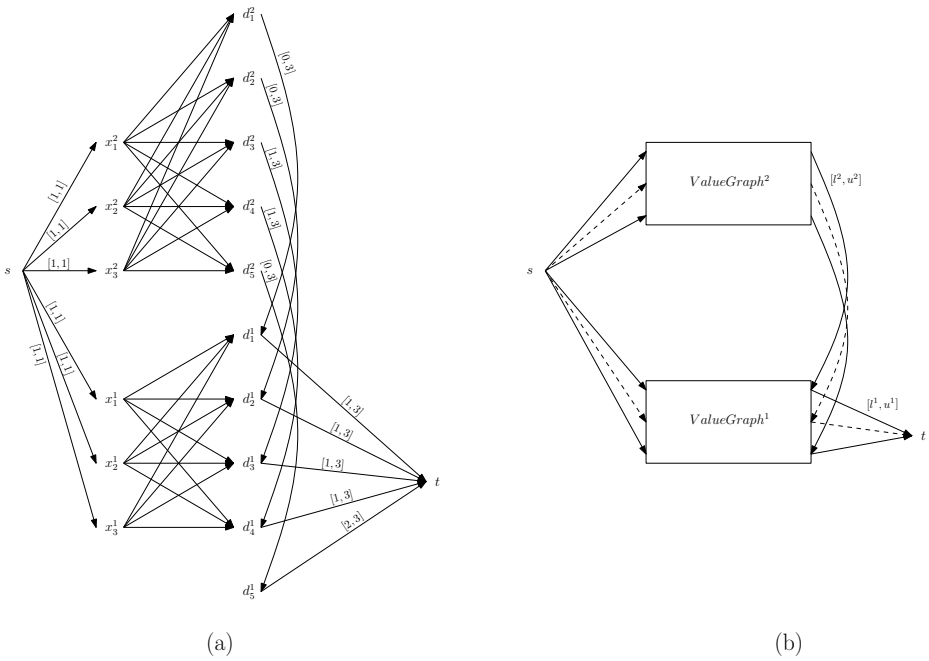

Fig. 2. (a) Nested-GCC Graph Representation for Example 2: if not shown the lower and upper bounds are respectively null and unitary. (b) Schematic graph representation for Example 2.

Proof. $\Rightarrow$ *Given a solution, we can build a feasible flow setting a unitary flow in the arc* (x_i^k, d_j^k) *for each assignment* $x_i^k = d_j$*. The arcs in* A_s *are all saturated. An arc* $(d_j^k, v) \in A_{req}^k$ *has a flow equal to* $\#(d_j, \tau \downarrow_{\mathbb{X}^k})$*. Note that for any given* k *and vertex* d_j^k*, demands and capacities of the arc* $(d_j^k, v) \in A_{req}^k$ *are satisfied since the related flow is equivalent to the sum of the flow coming from level* k *and higher.*
$\Leftarrow$ *Given a feasible (integral) flow, we build an assignment setting* $x_i^k = d_j$ *whenever* $f(x_i^k, d_j^k) = 1$*.*

Consider again Example 2: the constraint is infeasible and there is no feasible flow in the related graph G of Figure 2.

Corollary 1. *Let* G *be the graph representation of a* `nested_gcc` *and* f *a feasible flow on* G*. The constraint is domain consistent iff for each arc* $a \in A_{X^k}$ *there exists a feasible flow such that* $f(a) = 1$*.*

Proof. From Theorem 2, if there exists a feasible flow that has $f(a) = 1$ *with* $a = (x_i^k, d^k)$ *then there exists a solution with* $x_i^k = d$*. Analogously, if there exists a solution with* $x_i^k = d$ *then there exists a flow with* $f(a) = 1$ *where* $a = (x_i^k, d^k)$*.*

Following Régin in [6], we can design a filtering algorithm in which we find a feasible flow in the graph representation in order to check the feasibility of the

constraint. If it does not exist then the constraint is infeasible. Otherwise, we compute the strongly connected component [7] of the residual graph and then every arc that does not belong to any strongly connected component can be removed.

3.3 Complexity

In the following, we use N to indicate the total number of variables and d for $|D_{\mathbb{X}}|$. The propagation of the `nested_gcc` requires $O(nm)$ time to find a feasible flow (Ford-Fulkerson) and $O(n+m)$ to find infeasible values where n is the number of vertices and m is the number of arcs of the `nested_gcc` graph representation. Here, n is in $O(N+d\ell)$ and m is in $O(Nd+d\ell)$. Note that the equivalent set of `gcc`'s representing the `nested_gcc` requires the propagation of ℓ different `gcc`'s.A single `gcc` propagates in $O(\sqrt{n'}m')$ [4] where n' and m' are respectively the number of vertices and the number of arcs of the `gcc` graph representation and $n' \in O(N+d)$ and $m' \in O(Nd)$.

4 Further generalization

So far, skill levels have been considered linearly ordered: a resource of level k is able to satisfy requirements of levels $k, k-1, \ldots, 1$. The main challenge is now how far we can generalize relations between skill levels in order to address more complex problems while still using the flow algorithm.

Different skill level relations are shown in Figure 3: in (a) the skill levels are linearly ordered while in (b) levels are organized in a tree-like fashion. The semantic of 3b is that both resources of type β and γ can accomplish a task with requirements of type α, whereas resources of type β cannot satisfy requirements of type γ (and the other way around). Equivalently, we write $\delta \succ \beta$, $\beta \succ \alpha$, $\epsilon \succ \gamma$, $\zeta \succ \gamma$, $\eta \succ \gamma$ and $\gamma \succ \alpha$, where we consider $\succ$ a reflexive, antisymmetric and non-transitive relation. The transitive closure of $\succ$ is denoted by $\succ^*$ (hence, for instance $\delta \succ^* \alpha$). Note that the relation between resource classes is not a partial order relation: we cannot have $\lambda \succ \mu$ and $\lambda \succ \nu$ or, in other words, lower classes cannot rejoin in a single higher class. The reason of this limitation will be clarified in the next paragraph. Furthermore the relation set is such that there exists only a single root: a definition of multiple roots (a forest) simply gives rise to different constraints.

Example 3. A company is planning to develop two software components c_1 and c_2 for an application. The component c_1 requires between 7 and 10 programmers while c_2 between 8 and 10. Particularly, both components need 1 or 2 expert developers and 3 or 4 testers. A programmer is either a basic developer or an expert developer or a tester, however both expert developers and testers can accomplish duties as a basic developer ("expert developer" $\succ$ "basic developer", "tester" $\succ$ "basic developer"). The company has 4 novices, 8 testers and 3 expert developers. The different relations and component requirements are depicted in

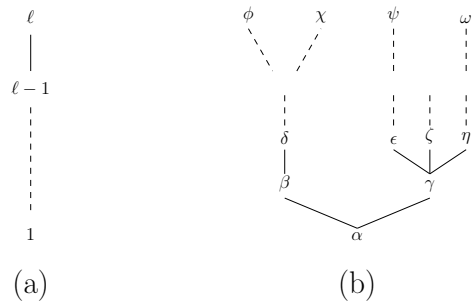

Fig. 3. Skill level relations: (a) linearly ordered skill levels and (b) tree-like ordered skill levels.

Figure 4. A possible solution is to assign 4 testers for each component; one tester for each component should work as a basic developer. Novices are evenly divided between the two tasks, one expert developer will be assigned to the development of component c_1 and finally the remaining two expert developers will work for the component c_2 (one as a basic developer).

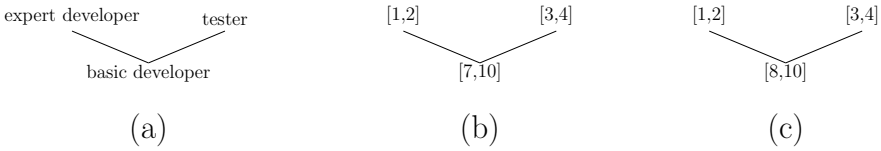

Fig. 4. (a) Programmers skill relations. (b) Requirements for component c_1. (c) Requirements for component c_2.

Note that, more generally, whenever we have a taxonomy or hierarchy of resources, we can easily derive the resource relations. This scenario fits perfectly applications in which resources are represented as classes in a UML class diagram and they are organized in a hierarchy (with single inheritance); a subclass by definition is a specialization of the superclass, it is able to act as the superclass (the subclass "is" a kind of superclass) but it has additional capabilities.

We now formally introduce the constraint that models the described problem substructure. In the following, Σ represents the set of different resource classes where, arbitrarily, α is considered the lower level (i.e. the root class). The variables representing resources of class $\gamma \in \Sigma$ are denoted by X^γ. We write $\mathbb{X}^\lambda = \bigcup\{X^\gamma \,|\gamma \in \Sigma, \gamma \succ^* \lambda\}$ to represents the union of the variables of level λ and higher w.r.t. the relation $\succ$[1]. For short, we write $\mathbb{X} = \mathbb{X}^\alpha$.

[1] of which the linear order relation used for the `nested_gcc` is a special case

Definition 3 (hierarchical gcc). *The hierarchical global cardinality constraint is formally defined as*

$$\texttt{hierarchical\_gcc}(\texttt{X}^{\alpha}, \ldots, \texttt{X}^{\omega}, (\texttt{l}^{\alpha}, \texttt{u}^{\alpha}), \ldots, (\texttt{l}^{\omega}, \texttt{u}^{\omega}), \succ) =$$

$$\{\tau = (d_1^{\alpha}, \ldots, d_{n_\alpha}^{\alpha}, d_1^{\beta}, \ldots, d_{n_\omega}^{\omega}) \,|\, d_i^{\gamma} \in D_i^{\gamma}, \forall \gamma \in \Sigma, \forall d \in D_{\mathbb{X}} : l_d^{\gamma} \leq \#(d, \tau \downarrow_{\mathbb{X}^{\gamma}}) \leq u_d^{\gamma}\}$$

4.1 Graph Representation

The graph representation is similar to the one introduced for the `nested_gcc`; it differs mainly in how the `gcc` subgraphs are connected. We have a `gcc` substructure for each resource class; value vertices are still duplicated and they are connected to the equivalent value vertices following the resource relations. Note again that resources can be only of a given class, hence a resource is represented by exactly one vertex. The total amount of flow coming out from a variable vertex is still unitary.
More formally, the graph $G = (V, A)$ is defined as follows:

$$V = \mathbb{X} \cup \Big(\bigcup_{\gamma \in \Sigma} D_{\mathbb{X}}^{\gamma} \Big) \cup \{s, t\}$$

$$A = A_s \cup \Big(\bigcup_{\gamma \in \Sigma} A_{X^{\gamma}} \Big) \cup \Big(\bigcup_{\gamma \in \Sigma} A_{req}^{\gamma} \Big)$$

where

$$\begin{aligned} A_s &= \{(s, x_i^{\gamma}) \,|\, \gamma \in \Sigma\,,\ i = 1, \ldots, n_{\gamma}\} \\ A_{X^{\gamma}} &= \{(x_i^{\gamma}, d_j^{\gamma}) \,|\, \gamma \in \Sigma, d_j^{\gamma} \in D_i^{\gamma}\} \\ A_{req}^{\gamma} &= \begin{cases} \{(d_i^{\alpha}, t) \,|\, i = 1, \ldots, |D_{\mathbb{X}}|\} & \text{if } \gamma = \alpha \\ \{(d_i^{\gamma}, d_i^{\lambda}) \,|\, i = 1, \ldots, |D_{\mathbb{X}}| : \gamma \succ \lambda\} & \text{if } \gamma \neq \alpha \end{cases} \end{aligned}$$

Arcs in A_s have unitary lower and upper bounds, whereas arcs in $A_{X^{\gamma}}$ have null lower bounds and unitary upper bounds. Each arc $(d_i^{\gamma}, v) \in A_{req}^{\gamma}$ has lower and upper bound respectively equal to $l_{d_i}^{\gamma}$ and $u_{d_i}^{\gamma}$.

An example is given in Figure 5; four classes of resources are defined with the following relations: $\delta \succ \beta$, $\beta \succ \alpha$ and $\gamma \succ \alpha$. The equivalent constraint graph representation is shown in Figure 5b.

One of the reasons why it is not possible to express resource relations as a partially ordered set (poset) is that we might have a situation like: $\lambda \succ \mu$ and $\lambda \succ \nu$. Thus, the outcome of a `gcc` substructure may have to flow in two different `gcc` substructures: $\texttt{gcc}^{\lambda}$ should output both in $\texttt{gcc}^{\mu}$ and $\texttt{gcc}^{\nu}$. Doubling the input flow (and consequently the output flow) of the $\texttt{gcc}^{\lambda}$ will clearly lead to inconsistencies inside the `gcc` substructure.

4.2 Domain consistency and propagation algorithm

Theorem 3. *There is a bijection between solutions to the* `hierarchical_gcc` *and feasible flows in the related graph representation G.*

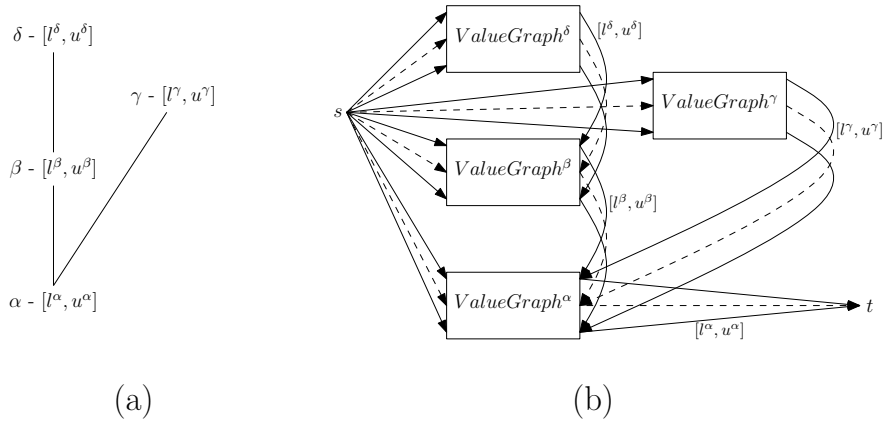

Fig. 5. (a) Resource relation. (b) Constraint graph representation.

Proof. $\Rightarrow$ *Given a solution, we can build a feasible flow setting a unitary flow in the arc* (x_i^γ, d_j^γ) *for each assignment* $x_i^\gamma = d_j$*. The arcs in* A_s *are all saturated. An arc* $(d_j^\gamma, v) \in A_{req}^\gamma$ *has a flow equal to* $\#(d_j, \tau \downarrow_{\mathbb{X}^\gamma})$*.*
$\Leftarrow$ *Given a feasible (integral) flow, we build an assignment setting* $x_i^\gamma = d_j$ *whenever* $f(x_i^\gamma, d_j^\gamma) = 1$*.*

Corollary 2. *Let* G *be the graph representation of a* `hierarchical_gcc` *and* f *a feasible flow on* G*. The constraint is domain consistent iff for each arc* $a \in A_{X^\gamma}$ *there exists a feasible flow such that* $f(a) = 1$*.*

Proof. From Theorem 3, if there exists a feasible flow that has $f(a) = 1$ *with* $a = (x_i^\gamma, d_j^\gamma)$ *then there exists a solution with* $x_i^\gamma = d_j$*. Analogously, if there exists a solution with* $x_i^\gamma = d_j$ *then there exists a flow with* $f(a) = 1$ *where* $a = (x_i^\gamma, d_j^\gamma)$*.*

The propagation algorithm works exactly as in the `nested_gcc` with the only difference given by the graph. The resulting complexity is then equivalent, that is, $O(nm)$ where n and m are respectively the number of vertices and edges of the graph representation. Here, n is in $O(N + d\ell)$ and m is in $O(Nd + d\ell)$.

5 Experimental results

We implemented the `nested_gcc` and `hierarchical_gcc` and we compared them with the equivalent set of `gcc`'s. Due to time constraints, we were able to generate significant instances only for the `nested_gcc` setting. We chose as benchmark some random instances of the ROADEF challenge. We recall briefly that the problem consists of grouping technicians in teams in order to accomplish a set of tasks. A technician has skills in different domains and, particularly, he has associated a skill level for each domain; a technician is able to satisfy requirements

of his skill level and lower. A task requires a specified number of technicians for each pair domain-level in order to be accomplished. The goal is to form teams of technicians such that they are able to perform a given set of tasks.

The problem is modeled as a set of `nested_gcc`'s one for each domain where the variables represent the technicians and the values represent the tasks. As variable selection heuristic, we developed an ad-hoc heuristic that chooses the most skilled technicians first; from preliminary tests this heuristic seemed to narrow the gap between `nested_gcc` and the set of `gcc` representations. The testbed has been generated as follows: each task has associated an optimistic approximation of the technicians needed; then from the set of tasks, we chose randomly a subset such that the sum of the approximations is less than the number of available technicians. Furthermore, we randomly removed values from variable domains according to an input percentage.

The constraint has been implemented with Ilog Solver 6.2 and the tests were performed on a machine with an AMD Dual Opteron 250 (2.4GHz) with 3GB RAM (note however that only one processor has been used). We set a time limit of 600 seconds for each run. We compared two different models: the former exploits the `nested_gcc`, the latter uses traditional `gcc`'s. For a fair comparison, the second model has been solved using both our implementation of the `gcc` and the ILOG's one. The results are shown in Table 1. Instances from data11 have 4 skill domains with 4 skill levels each whereas instances from data12 have 5 domains with 3 skill levels each. Each instance has been tried with different percentages of domain value removals (shown in the second column). The remaininig columns show the running times and the number of backtracks respectively for the `nested_gcc`, `gcc` and the ILOG's one; in all three approaches the same variable and value ordering heuristics have been used. If the running time is not shown, the solver timed out either without finding a solution or without proving the infeasibility of the instance (however in those cases we show the number of backtracks performed within the time limit).

Depending on the instance, the reduction on the number of backtracks using the `nested_gcc` can go from null to two orders of magnitude; whenever the reduction is significant, we obtain better running times. Nonetheless there are instances in which even with our implementation of the `gcc` we get better performances over the `nested_gcc`. Hence, further studies are required in order to better characterize the instances and understand when the use of the `nested_gcc` is likely to lead to better running times. We think that an instance generator with a more fine grained parameterization could help us in this task as well as in generating a broader testbed. In fact, the basic generator produced too many instances either too easy or too hard, the same problem that we encountered also during the generation of a testbed for the `hierarchical_gcc`.

Instance	Perc.	nested_gcc		gcc's		ILOG gcc's	
		Time (secs)	Backtracks	Time (secs)	Backtracks	Time (secs)	Backtracks
data11-a	0.1	0.01	4	0.01	4	0.01	4
data11-a	0.2	0.01	0	0.01	0	0.01	0
data11-a	0.3	13.61	73381	14.12	73381	**11.92**	73381
data11-a	0.4	0.40	2261	0.38	2261	**0.33**	2261
data11-a	0.5	0.02	68	**0.01**	68	**0.01**	68
data11-b	0.1	-	1944273	-	2444379	-	3800974
data11-b	0.2	**35.75**	106324	56.27	202690	51.76	202690
data11-b	0.3	**3.85**	10992	4.65	17454	3.96	17454
data11-b	0.4	3.1	8267	2.75	9219	**2.51**	9219
data11-b	0.5	1.76	4986	**1.36**	5382	1.52	5382
data11-c	0.1	6.43	23531	5.49	23778	**3.52**	23778
data11-c	0.2	3.18	11771	2.71	11979	**1.81**	11979
data11-c	0.3	0.07	247	0.07	247	**0.05**	247
data11-c	0.4	0.01	1	0.01	1	0.01	1
data11-c	0.5	0.01	1	0.01	1	0.01	1
data11-d	0.1	**4.15**	16478	7.04	26766	5.59	26766
data11-d	0.2	**0.01**	11	**0.01**	38	0.02	38
data11-d	0.3	0.01	4	0.01	15	0.01	15
data11-d	0.4	0.01	1	0.01	1	0.01	1
data11-d	0.5	0.01	1	0.01	1	0.01	1
data12-a	0.2	-	2219874	-	2495188	-	2763002
data12-a	0.3	**140.43**	422107	-	2373366	-	2650290
data12-a	0.4	**0.16**	419	135.44	505243	154.01	505243
data12-a	0.5	**0.11**	300	6.47	22099	6.64	22099
data12-a	0.6	**0.01**	1	0.03	75	0.04	75
data12-b	0.2	-	1327519	-	1601952	-	1231394
data12-b	0.3	-	1376981	-	1642615	-	1169144
data12-b	0.4	**20.16**	45634	36.00	109762	53.24	109762
data12-b	0.5	**0.38**	827	0.52	1274	0.74	1274
data12-b	0.6	0.01	1	0.01	1	0.01	1
data12-c	0.2	-	1810092	-	2184929	-	3317824
data12-c	0.3	**13.24**	36336	20.12	75787	14.74	75787
data12-c	0.4	1.12	2835	1.55	5475	**1.02**	5475
data12-c	0.5	**0.12**	266	0.28	1033	0.26	1033
data12-c	0.6	**0.05**	69	0.83	2618	0.74	2618
data12-d	0.2	-	1434098	-	2363013	-	2361900
data12-d	0.3	-	1483539	-	2185816	-	2030271
data12-d	0.4	179.00	353462	**148.62**	445726	172.92	445726
data12-d	0.5	98.02	185798	74.77	203920	**74.04**	203920
data12-d	0.6	0.09	152	0.06	152	**0.04**	152

Table 1. Experimental results.

6 Expressing preferences

For ease of presentation, in this section we take into consideration only linearly ordered resources. However the results can be easily extended to the hierarchical version.

In the constraints presented, there might be cases in which we would like to define some preferences among different consistent assignments. Imagine, for example, that a given task requires one resource of level 1 and one of level 2; however only one resource of level 2 and one of level 3 are available. The constraint does not allow to express a difference between a solution in which the level 3 resource will perform level 1 duties or a solution in which the level 2 resource will carry out level 1 duties and the level 3 resource level 2 duties. Nonetheless, in both solutions we simply have the two resources assigned to the same task without any information about who is going to perform what. We should then enrich the model in order to express this new information.

In this new setting, domains should contain for each task different values to denote different duty levels. So, d_j^k represents the task j with duty level k; assignment $x_i^{k'} = d_j^k$ means that resource i of level k' is going to perform duties of level k of task j. For the sake of clarity, note that in the `nested_gcc` we would have had simply $x_i^{k'} = d_j$; the differentiation of duty levels is only present inside the graph representation but not at the constraint level; for expressing preferences such differentiation needs to be brought up to the constraint level.

In the graph representation of `nested_gcc` with preferences, a variable $x_i^{k'}$ is connected directly to value vertices d_j^k with $k \leq k'$. Lower and upper bounds of value occurrences are expressed directly for each d_j^k. It follows that occurrences of a value d_j^k do not interfere with the ones of value $d_j^{k'}$ with $k \neq k'$. Hence, the graph representation does not contain anymore the arcs A_{req}^k that connect value vertices of different levels but rather value vertices are directly connected to the sink. In order to express preferences, we should also introduce positive costs on the arcs $(x_i^{k'}, d_i^k)$ whenever $k < k'$. Thus, the overall graph representation is similar to a particular case of the `cost_gcc` [5]. Figure 6 shows an example of a `nested_gcc` with preferences in which we have 5 variables and 9 values (5 resources with 3 different skill levels and 3 tasks leading to an overall of 9 different values). With such a representation it is also straightforward to constraint the eventual gap in a given assignment between the resource level and the duty level.

Note that this particular graph representation could not be used for the `nested_gcc`. For instance, take in consideration Figure 6: if this would have been a traditional `nested_gcc`, value vertices represent simply tasks, hence d_2^2 denotes the task 2 as well as d_2^1. Suppose furthermore that task 2 requires at most 1 resource of level 2 or higher. The constraint would be consistent even with assignments $x_1^2 = d_2^1$ and $x_2^2 = d_2^2$ hence leading to contradiction.

7 Conclusions

We proposed generalizations of the `gcc` to address certain resource allocation problems. We showed both theoretically and empirically that such generalizations can outperform an alternate formulation using `gcc`'s. As future work, we plan to do a more extensive empirical analysis in order to better characterize the hardness of the instances. Finally we will consider filtering on occurrence variables.

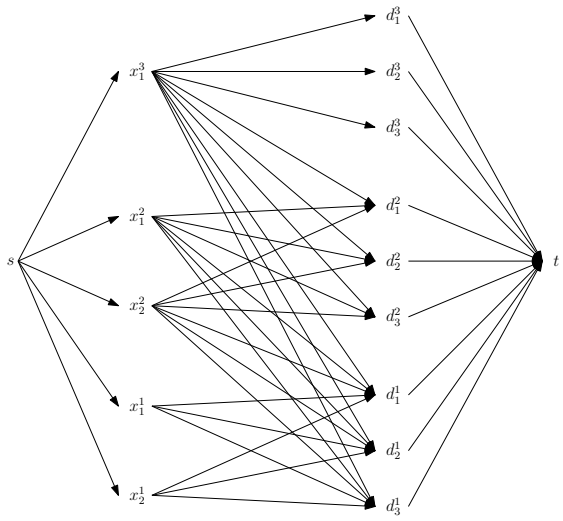

Fig. 6. Graph representation for the nested gcc with preferences.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. Network Flows. Prentice Hall, 1993.
2. K.R. Apt. Principles of Constraint Programming. Cambridge Unievristy Press, 2003.
3. K. Elbassioni, I. Katriel, M. Kutz, and M. Mahajan. Simultaneous Matchings. *Proceedings of the Sixteenth International Symposium on Algorithms and Computation (ISAAC 2005)*, Springer LNCS 3827: 106-115.
4. C-G. Quimper, Alejandro López-Ortiz, P. van Beek and Alexander Golynski. Improved Algorithms for the Global Cardinality Constraint. *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, Springer LNCS 3258: 542-556.
5. J-C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP 1999)*, Springer LNCS 1713: 390-404.
6. J-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, AAAI Press: 209-215.
7. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146-160, 1972.
8. Challenge Roadef 2007, http://gilco.inpg.fr/ChallengeROADEF2007/ , 2007-01-30.