Submission to CONSTRAINTS, , 1–18 ()
© Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

# Cost based Arc Consistency for Global Cardinality Constraints

JEAN-CHARLES RÉGIN regin@ilog.fr
*ILOG, Les Taissounières HB2, 1661, route des Dolines, Sophia Antipolis, 06560 Valbonne, FRANCE*

**Editor:**

**Abstract.** A global cardinality constraint (gcc) is specified in terms of a set of variables $X = \{x_1, ..., x_p\}$ which take their values in a subset of $V = \{v_1, ..., v_d\}$. It constrains the number of times each value $v_i \in V$ is assigned to a variable in $X$ to be in an interval $[l_i, u_i]$. A gcc with costs (costgcc) is a generalization of a gcc in which a cost is associated with each value of each variable. Then, each solution of the underlying gcc is associated with a global cost equal to the sum of the costs associated with the assigned values of the solution. A costgcc constrains the global cost to be less than a given value. Cardinality constraints with costs have proved very useful in many real-life problems, such as traveling salesman problems, scheduling, rostering, or resource allocation. For instance, they are useful for expressing preferences or for defining constraints such as a constraint on the sum of all different variables. In this paper, we present an efficient way of implementing arc consistency for a costgcc. We also study the incremental behavior of the proposed algorithm.

**Keywords:** Constraint satisfaction problem, global constraint, filtering algorithm, cardinality constraints, cardinality constraints with costs, arc consistency

## 1. Introduction

Constraint satisfaction problems (CSPs) form a simple formal frame to represent and solve certain problems in artificial intelligence. They involve finding values for problem variables subject to constraints on which combinations are acceptable. The problem of the existence of solutions to the CSP is NP-complete. Therefore, methods have been developed to simplify the CSP before or during the search for solutions. The use of filtering algorithms associated with constraints is one of the most promising methods. A filtering algorithm associated with one constraint aims at removing values that are not consistent with the constraint. When all the values that are inconsistent with the constraint are deleted by the filtering algorithm we say that it achieves the arc consistency.

The design of specific filtering algorithms for some constraints is necessary to solve some CSPs. Furthermore, it is also necessary to deal with global constraints. This has been clearly shown by the great interest in solving some real-world problems using the constraints: diff-n, alldiff [10], cumulative [2], global cardinality constraint [11].

A global constraint can be seen as the conjunction of a set of constraints. For instance an alldiff constraint involving the variables $x$, $y$ and $z$, gathers together all the binary $\neq$ constraints between variables $x$, $y$ and $z$. The advantage of dealing

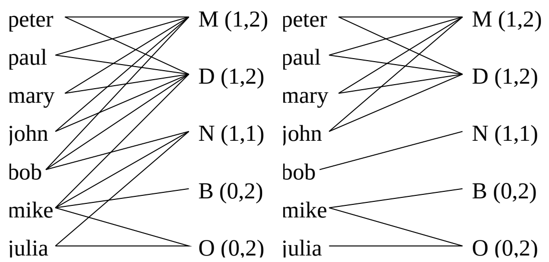

*Figure 1.* An example of global constraint of cardinality.

with global constraints is that the globality of the constraint can be taken into account. This means that, generally, the filtering algorithm associated with a global constraint is more powerful than the conjunction of the filtering algorithms associated with each underlying constraint if we take them separately. A gcc is specified in terms of a set of variables $X = \{x_1, ..., x_p\}$ which take their values in a subset of $V = \{v_1, ..., v_d\}$. It constrains the number of times a value $v_i \in V$ is assigned variables in $X$ to be in an interval $[l_i, u_i]$. Gccs arise in many real-life problems. For instance, consider the example derived from a real problem and given in [4]. The task is to schedule managers for a directory-assistance center, with 5 activities, 7 persons over 7 days. Let us study only one part of this problem: Each day, a person has to perform an activity and we may have a minimum and maximum number of times that this activity can be performed. This constraint can be exactly formulated as gcc. It can be represented by a bipartite graph called a value graph (left graph in Figure 1). The left set corresponds to the person set, the right set to the activity set. There exists an edge between a person and an activity when the person can perform the activity. For each activity, the numbers in parenthesis express the minimum and the maximum number of times the activity has to be assigned. A gcc can be efficiently handled by designing specific algorithms for this constraint [11]. For the example we consider, the achievement of arc consistency for the gcc leads to the right graph in Figure 1. Such a result can be obtained only by taking into account the globality of the constraint. In this paper we propose to add costs to a gcc.

The addition of costs to an alldiff constraint, which is a particular case of gcc, has been studied by Caseau and Laburthe [5]. However, they do not propose any filtering algorithm. They only show the interest of computing the consistency of such a constraint.

Consider again the previous example and suppose that each person expresses some preferences on the activities that they can perform. These preferences can be represented by an integer. A small number indicates that the activity is prefered, while a large number corresponds to a penalty. Suppose also that there is for each day a constraint stating that the sum of the preferences must be less than a

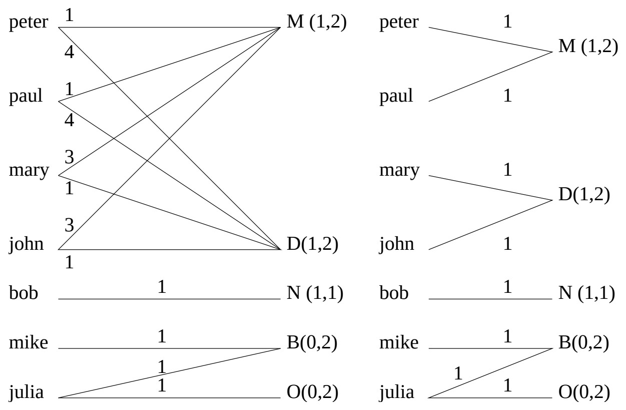

*Figure 2.* An example of global cardinality with costs. The sum of the assignments must be strictly less than 12.

given number. For instance, consider Peter, Paul, Mary, and John. Peter has a preference 1 for the activity $M$ and 4 for the activity $D$; Paul has a preference 1 for the activity $M$ and 4 for the activity $D$; Mary and John a preference of 1 for the activity $D$ and 3 for the activity $M$. For all the other persons preferences are 1 for all the activities. Preferences can be represented by costs in the value graph. (See Figure 2.) A preference $p$ between a person and an activity corresponds to the cost $p$ on the edge between the person and the activity. Thus, we can associate with each person a variable whose domain is defined by the preferences of the person. When an activity is assigned to a person, the variable associated with this person is instantiated to the preference of the person for this activity. We will denote by CPeter, CPaul, CMary, CJohn, CBob, CMike, CJulia these variables. CPeter can take the values 1 or 4; CPaul 1 or 4; CMary 1 or 3; CJohn 1 or 3; CBob, CMike, CJulia are instantiated to 1. The minimal solution has a cost 7 (each person is assigned to his preferred activity). Suppose that a solution with a global cost greater than 11 is not acceptable. This constraint on preferences corresponds to a sum constraint involving the variables associated with every person. More precisely, it is defined by CPeter + CPaul + CMary + CJohn + CBob + CMike + CJulia <= 11. If we consider this sum constraint independently from the gcc, nothing will be deduced because the greatest value of cost variables is consistent with the sum constraint: 4+1+1+1+1+1+1 = 10 <= 11, so no value is removed. However, we can prove, for instance, that it is impossible to have the value 4 for CPeter. This means that the assignment (Peter,D) is incompatible w.r.t. the gcc in conjunction with the sum constraint. Suppose that Peter is assigned to D, then D can also be assigned to Mary or John for a cost 1, but together they cannot be assigned to D, because D can be assigned at the most twice. Thus Mary or John must be assigned to M for a cost 3. The minimum cost if Peter is assigned to D is: $4 + 1 + 1 + 3 = 9$. For the other persons their contribution is 3. The minimum global cost is 12 and so violates the acceptability constraint. The right graph of Figure 2 shows the result of the achievement of arc consistency for the costgcc.

In this paper we present an efficient way of implementing generalized arc consistency for the costgcc. The filtering algorithm is based on the minimum cost flow algorithm.

First, we give some preliminaries on constraint satisfaction problems. And, we present the flow theory because our results are based on it. Then, we present an algorithm checking the consistency of a costgcc. Afterwards, we propose a simple algorithm for achieving generalized arc consistency, which is based on a new proposition in the flow theory. Finally, we conclude.

## 2. CSP

A finite **constraint network** $\mathcal{N}$ is defined as a set of $n$ **variables** $X = \{x_1, \ldots, x_n\}$, a set of current **domains** $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible **values** for variable $x_i$, and a set $\mathcal{C}$ of **constraints** between variables. We introduce the particular notation $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ to represent the set of initial domains of $\mathcal{N}$. Indeed, we consider that any constraint network $\mathcal{N}$ can be associated with an initial domain $\mathcal{D}_0$ (containing $\mathcal{D}$), on which constraint definitions were stated.

A **constraint** $C$ on the ordered set of variables $X(C) = (x_{i_1}, \ldots, x_{i_r})$ is a subset $T(C)$ of the Cartesian product $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ that specifies the **allowed** combinations of values for the variables $x_{i_1}, \ldots, x_{i_r}$. An element of $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ is called a **tuple on** $X(C)$. $|X(C)|$ is the **arity** of $C$.

A value $a$ for a variable $x$ is often denoted by $(x, a)$. $\texttt{var}(C, i)$ represents the i$^{th}$ variable of $X(C)$, while $\texttt{index}(C, x)$ is the position of variable $x$ in $X(C)$. $\tau[k]$ denotes the $k^{th}$ value of the tuple $\tau$. $D(X)$ denotes the union of domains of variables of $X$ (i.e. $D(X) = \cup_{x_i \in X} D(x_i)$). $\#(a, \tau)$ is the number of occurrences of the value $a$ in the tuple $\tau$.

Let $C$ be a constraint. A tuple $\tau$ on $X(C)$ is **valid** if $\forall (x, a) \in \tau, a \in D(x)$. $C$ is **consistent** iff there exists a tuple $\tau$ of $T(C)$ which is valid. A value $a \in D(x)$ is **consistent with** $C$ iff $x \notin X(C)$ or there exists a valid tuple $\tau$ of $T(C)$ with $a = \tau[\texttt{index}(C, x)]$. A constraint is **arc consistent** iff $\forall x_i \in X(C), D(x_i) \neq \varnothing$ and $\forall a \in D(x_i)$, $a$ is consistent with $C$.

The **value graph** [7] of an non-binary constraint $C$ is the bipartite graph $GV(C) = (X(C), D(X(C)), E)$ where $(x, a) \in E$ iff $a \in D_x$.

### 2.1. *Global Cardinality Constraints with costs*

Throughout this paper, we are interested in global cardinality constraints with costs (costgcc). They introduce cost in global cardinality constraints that are defined by the minimal and the maximal number of times each value of $D(X(C))$ must appear in each tuple of the constraints. The minimal and the maximal number of occurrences of each value can be different from the others. More formally we have:

**Definition 1** *A* **global cardinality constraint** *is a constraint $C$ in which each value $a_i \in D(X(C))$ is associated with two positive integers $l_i$ and $u_i$ and*
$T(C) = \{ \tau$ *such that* $\tau$ *is a tuple on* $X(C)$
*and* $\forall a_i \in D(X(C)) : l_i \leq \#(a_i, \tau) \leq u_i\}$
*It is denoted by $gcc(X, l, u)$.*

**Definition 2** *A* **cost function on a variable set** $X$ *is a function which associates with each value $(x, a)$, $x \in X$ and $a \in D(x)$ an integer denoted by $cost(x, a)$.*

A costgcc is the conjunction of a gcc constraint and a sum constraint:

**Definition 3** *A* **global cardinality constraint with costs** *is a constraint $C$ associated with* `cost` *a cost function on $X(C)$, an integer $H$ and in which each value $a_i \in D(X(C))$ is associated with two positive integers $l_i$ and $u_i$*
$T(C) = \{ \tau$ *such that* $\tau$ *is a tuple on* $X(C)$
*and* $\forall a_i \in D(X(C)) : l_i \leq \#(a_i, \tau) \leq u_i$
*and* $\Sigma_{i=1}^{|X(C)|} cost(\mathtt{var}(C, i), \tau[i]) \leq H \ \}$
*It is denoted by $costgcc(X, l, u, cost, H)$.*

There is no assumption made on the sign of costs.

## 3. Flows

### *3.1. Preliminaries*

The definitions about graph theory are due to [12]. The definitions, theorems and algorithms about flow are based on books of [3, 8, 12, 1].

A **directed graph** or **digraph** $G = (X, U)$ consists of a **vertex set** $X$ and an **arc set** $U$, where every arc $(u, v)$ is an ordered pair of distinct vertices. We will denote by $X(G)$ the vertex set of $G$ and by $U(G)$ the arc set of $G$. The **cost** of an arc is a value associated with the arc.

A **path** from node $v_1$ to node $v_k$ in $G$ is a list of nodes $[v_1, ..., v_k]$ such that $(v_i, v_{i+1})$ is an arc for $i \in [1..k-1]$. The path **contains** node $v_i$ for $i \in [1..k]$ and arc $(v_i, v_{i+1})$ for $i \in [1..k-1]$. The path is **simple** if all its nodes are distinct. The path is a **cycle** if $k > 1$ and $v_1 = v_k$. The **length** of a path $p$, denoted by $length(p)$, is the sum of the costs of the arcs contained in $p$. A **shortest path** from a node $s$ to a node $t$ is a path from $s$ to $t$ whose length is minimum. A cycle of negative length is called a **negative cycle**. Let $s$ and $t$ be nodes, there is a shortest path from $s$ to $t$ if and only if there exists a path from $s$ to $t$ and no path from $s$ to $t$ contains a negative cycle. If there is a shortest path from $s$ to $t$, there is one that is simple.

The complexity of the search for shortest paths from a node to every node in a graph with $m$ arcs and $n$ nodes depends on the maximal cost $\gamma$ and on the sign of the costs. Therefore, we will denoted this complexity by $S(m, n, \gamma)$ if all the costs are nonnegative; and $S_{neg}(m, n, \gamma)$ otherwise.

Let $G$ be a graph for which each arc $(i, j)$ is associated with three integers $l_{ij}$, $u_{ij}$, and $c_{ij}$, respectively called the **lower bound capacity**, the **upper bound**

**capacity** and the **cost** of the arc.

A **flow** in $G$ is a function $f$ satisfying the following two conditions:
• For any arc $(i,j)$, $f_{ij}$ represents the amount of some commodity that can "flow" through the arc. Such a flow is permitted only in the indicated direction of the arc, i.e., from $i$ to $j$. For convenience, we assume $f_{ij} = 0$ if $(i,j) \notin U(G)$.
• A **conservation law** is observed at each node: $\forall j \in X(G) : \sum_i f_{ij} = \sum_k f_{jk}$.
The **cost** of a flow $f$ is $cost(f) = \sum_{(i,j)\in U(G)} f_{ij}c_{ij}$.

We will consider three problems of flow theory:
• **the feasible flow problem**: Does there exist a flow in $G$ that satisfies the **capacity constraint**? That is find $f$ such that $\forall(i,j) \in U(G)\ l_{ij} \leq f_{ij} \leq u_{ij}$.
• **the problem of the maximum flow for an arc** $(i,j)$: Find a feasible flow in $G$ for which the value of $f_{ij}$ is maximum.
• **the minimum cost flow problem**: If there exists a feasible flow, find a feasible flow $f$ such that $cost(f)$ is minimum.

Without loss of generality (see p.45 and p.297 in [1]) , and to overcome notational difficulties, we will consider that:
• if $(i,j)$ is an arc of $G$ then $(j,i)$ is not an arc of $G$.
• all boundaries of capacities are nonnegative integers.
In fact, if all the upper bounds and all the lower bounds are integers and if there exists a feasible flow, then for any arc $(i,j)$ there exists a maximum flow from $j$ to $i$ which is integral on every arc in $G$ (See [8] p113.)

**Definition 4** *Let $f$ be a flow in $G$. For each arc $(i,j)$ of $G$, the* **infeasibility number** *$k_{ij}$ w.r.t. $f$ is defined by:*
• *$k_{ij} = 0$ if $l_{ij} \leq f_{ij} \leq u_{ij}$;*
• *$k_{ij} = l_{ij} - f_{ij}$ if $f_{ij} < l_{ij}$; ($(i,j)$ is lower-infeasible)*
• *$k_{ij} = f_{ij} - u_{ij}$ if $f_{ij} > u_{ij}$. ($(i,j)$ is upper-infeasible)*

## *3.2. Flow algorithms*

Consider, for instance, that all the lower bounds are equal to zero and suppose that you want to increase the flow value for an arc $(i,j)$. In this case, the flow of zero on all arcs, called the **zero flow**, is a feasible flow. Let $P$ be a path from $j$ to $i$ different from $[j,i]$, and $val = min(\{u_{ij}\} \cup \{u_{pq} \text{ s.t. } (p,q) \in P\})$. Then we can define the function $f$ on the arcs of $G$ such that $f_{pq} = val$ if $P$ contains $(p,q)$ or $(p,q) = (i,j)$ and $f_{pq} = 0$ otherwise. This function is a flow in $G$. (The conservation law is obviously satisfied because $(i,j)$ and $P$ form a cycle.) We have $f_{ij} > 0$, hence it is easy to improve the flow of an arc when all the lower bounds are zero and when we start from the zero flow. It is, indeed, sufficient to find a path satisfying the capacity constraint.

The main idea of the basic algorithms of flow theory, is to proceed by successive modifications of flows, that are computed in a graph in which all the lower bounds are zero and the current flow is the zero flow. This particular graph can be obtained from any flow and is called the residual graph:

**Definition 5** *The* **residual graph** *for a given flow $f$, denoted by $R(f)$, is the digraph with the same node set as in $G$. The arc set of $R(f)$ is defined as follows:* $\forall (i,j) \in U(G)$:
• $f_{ij} < u_{ij} \Leftrightarrow (i,j) \in U(R(f))$ *and has cost* $rc_{ij} = c_{ij}$ *and upper bound capacity* $r_{ij} = u_{ij} - f_{ij}$.
• $f_{ij} > l_{ij} \Leftrightarrow (j,i) \in U(R(f))$ *and has cost* $rc_{ji} = -c_{ij}$ *and upper bound capacity* $r_{ji} = f_{ij} - l_{ij}$.
*All the lower bound capacities are equal to* $0$.

Instead of working with the original graph $G$, we can work with the residual graph $R(f^o)$ for some $f^o$. From $f'$ a flow in $R(f^o)$, we can obtain $f$ another flow in $G$ defined by: $\forall (i,j) \in U(G) : f_{ij} = f^o_{ij} + f'_{ij} - f'_{ji}$. And from a path in $R(f^o)$ we can define a flow $f'$ in $R(f^o)$ and so a flow in $G$:

**Definition 6** *We will say that $f$* **is obtained from $f^o$ by sending $k$ units of flow along a path $P$ from $j$ to** *$i$ if:*
• *$P$ is a path in* $R(f^o) - \{(j,i)\}$
• $k \leq min(\{r_{ij}\} \cup \{r_{uv} s.t. (u,v) \in P\})$
• *$f$ corresponds in $R(f^o)$ to the flow $f'$ defined by:*
  • $f'_{pq} = k$ *for each arc* $(p,q) \in P \cup \{(i,j)\}$
  • $f'_{pq} = 0$ *for all other arcs.*
*If $k$ is not mentioned it will be assumed that* $k = min(\{r_{ij}\} \cup \{r_{uv} s.t. (u,v) \in P\})$

In the previous definition the path must be different from $[j,i]$, otherwise $f'$ will be the zero flow.

The following proposition shows that the existence of a path in the residual graph is a necessary and sufficient condition:

**Theorem 1** *Let $f^o$ be any feasible flow in $G$, and $(i,j)$ be an arc of $G$.*
• *There is a feasible flow $f$ in $G$ with $f_{ij} > f^o_{ij}$ if and only if there exists a path from $j$ to $i$ in* $R(f^o) - \{(j,i)\}$.
• *There is a feasible flow $f$ in $G$ with $f_{ij} < f^o_{ij}$ if and only if there exists a path from $i$ to $j$ in* $R(f^o) - \{(i,j)\}$.

**proof:** see [8] p112. ⊙

*3.2.1. Maximum flow algorithm* Theorem 1 gives a way to construct a maximum flow in an arc $(i,j)$ by iterative improvement, due to Ford and Fulkerson:
Begin with any feasible flow $f^0$ and look for a path from $j$ to $i$ in $R(f^0) - \{(j,i)\}$. If there is none, $f^0$ is maximum. If, on the other hand, we find such a path $P$, then define $f^1$ obtained from $f^0$ by sending flow along $P$. Now look for a path from $j$ to $i$ in $R(f^1) - \{(j,i)\}$ and repeat this process. When there is no such path for $f^k$, then $f^k$ is a maximum flow.

A path can be found in $O(m)$, thus we have:

**Property 1** *A maximum flow of value $v$ in an arc $(i,j)$ can be found from a feasible flow in $O(mv)$.*

*3.2.2. Feasible flow algorithm* For establishing a feasible flow, several methods exist. For instance, it is possible to transform this problem into one in which all the lower bounds capacities are equal to zero and searching for a particular maximum flow value for one arc. (See [1] p 169.) However, there is a simple method which repeatedly searchs for maximum flows in some arcs:
Start with the zero flow $f^o$. This flow satisfies the upper bounds. Set $f = f^o$, and apply the following process while the flow is not feasible:
1) pick an arc $(i,j)$ such that $f_{ij}$ violates the lower bound capacity in $G$ (i.e. $f_{ij} < l_{ij}$).
2) Find $P$ a path from $j$ to $i$ in $R(f) - \{(j,i)\}$.
3) Obtain $f'$ from $f$ by sending flow along $P$; set $f = f'$ and goto 1)
If, at some point, there is no path for the current flow, then a feasible flow does not exist. Otherwise, the obtained flow is feasible.

**Property 2** *Let $k_{ij}$ be the infeasibility number w.r.t. the zero flow of each arc $(i,j)$ in $G$. We can find a feasible flow in $G$ or prove there is none in $O(m\sum_{(i,j)\in U(G)} k_{ij})$.*

*3.2.3. Minimum cost flow problem* The search for a feasible flow with a minimum cost implies only few modifications in the previous algorithm to ensure that the cost of the feasible flow will be minimum. In fact, only one aspect of the method is modified, the flow will be obtained by sending flow along special paths: the shortest ones. That is, the shortest paths are computed in the residual network by using the residual cost as cost. This algorithm is called the successive shortest path algorithm:
Start with the zero flow $f^o$. This flow satisfies the upper bounds. Set $f = f^o$, and apply the following process while the flow is not feasible:
1) pick an arc $(i,j)$ such that $f_{ij}$ violates the lower bound capacity in $G$ (i.e. $f_{ij} < l_{ij}$).
2) Find $P$ a shortest path from $j$ to $i$ in $R(f) - \{(j,i)\}$.
3) Obtain $f'$ from $f$ by sending flow along $P$; set $f = f'$ and goto 1)
If, at some point, there is no path for the current flow, then a feasible flow does not exist. Otherwise, the obtained flow is feasible and is a minimum cost flow.

The correctness of this algorithm is based on the following proposition:

**Notation 1** *We will denote by:*
- *$G_{l_{ij}\leftarrow a}$ the graph $G$ in which $l_{ij}$ has been replaced by $a$.*
- *$d_{R(f)-\{(j,i)\}}(j,i)$ the length of a shortest path from $j$ to $i$ is $R(f) - \{(j,i)\}$*

**Lemma 1** *Let $f$ be a minimum cost flow in $G$ and $(i,j)$ be an arc of $G$. If there is a shortest path $P$ from node $j$ to node $i$ in $R(f) - \{(j,i)\}$, then the flow $f'$ obtained from $f$ by sending $k$ units of flow along $P$ is a minimum cost flow in $G_{l_{ij}\leftarrow f_{ij}+k}$*

*and* $cost(f') = cost(f) + k(rc_{ij} + d_{R(f)-\{(j,i)\}}(j,i))$*. If such a path* $P$ *does not exist then there is no feasible flow in* $G_{l_{ij} \leftarrow f_{ij}+1}$.

**proof:** see in [8] p130. ⊙

The residual graph contains some negative costs, thus we have:

**Property 3** *Let* $k_{ij}$ *be the infeasibility number w.r.t. the zero flow of each arc* $(i,j)$ *in* $G$*. We can find a minimum cost flow flow in* $G$ *or prove there is none in* $O(S_{neg}(n,m,\gamma) \sum_{(i,j) \in U(G)} k_{ij})$.

## 3.3. Incrementality

Suppose $f^o$ is a minimum cost flow in $G^o$, and $G$ is the same graph as $G^o$ except that some capacity boundaries have been tighten (i.e. some lowers bounds have been increased and some upper bounds have been decreased). $f^o$ is not necessarily feasible in $G$.

By lemma 1, we know how to augment flow for an arc. The reduction of the flow value for an arc is based on a similar lemma:

**Lemma 2** *Let* $f$ *be a minimum cost flow in* $G$ *and* $(i,j)$ *be an arc of* $G$*. If there is a shortest path* $P$ *from node* $i$ *to node* $j$ *in* $R(f) - \{(i,j)\}$*, then the flow* $f'$ *obtained from* $f$ *by sending* $k$ *units of flow along* $P$ *is a minimum cost flow in* $G_{u_{ij} \leftarrow f_{ij}-k}$ *and* $cost(f') = cost(f) + k(rc_{ji} + d_{R(f)-\{(i,j)\}}(i,j))$*. If such a path* $P$ *does not exist then there is no feasible flow in* $G_{u_{ij} \leftarrow f_{ij}-1}$.

Therefore, we can obtain a feasible flow in $G$ which is also a minimum cost flow or prove there is none by applying the following algorithm:
Start with $f = f^o$ and apply the following process while $f$ is infeasible in $G$:
Pick an arc $(i,j)$ such that $f_{ij}$ violates a bound capacity in $G$. If $f_{ij} < l_{ij}$, then find $P$ a shortest path from $j$ to $i$ in $R(f) - \{(j,i)\}$. If $f_{ij} > u_{ij}$, then find $P$ a shortest path from $i$ to $j$ in $R(f) - \{(i,j)\}$. Obtain $f'$ from $f$ by sending flow along $P$; set $f = f'$
If, at some point, there is no path for the current flow, then a feasible flow does not exist. Otherwise, the obtained flow is minimum cost flow.

## 3.4. Working with nonnegative costs

The complexity of the previous algorithms depends on the sign of the costs, because the most powerful algorithms for searching for shortest paths deals only with nonnegative costs. However, the residual graph contains some negative costs. In this section, we present a well known method for modifying the residual graph in a way such that all the costs are nonnegative.

Let $f$ be a current flow, and assume that the shortest path distances from a node $s$ to every node in the residual graph have been computed. In this case, we have:

$$\forall i,j \in X(R(f)) : d_{R(f)}(s,j) \le d_{R(f)}(s,i) + rc_{ij}.$$

This property holds because it means that either there is a shortest path from $s$ to $j$ which contains the arc $(i, j)$ and the equality is reached, or there is no shortest path from $s$ to $j$ which contains the arc $(i, j)$ and the inequality is strict.

Therefore, $\forall i, j \in X(R(f)) : d_{R(f)}(s, i) + rc_{ij} - d_{R(f)}(s, j) \geq 0$. We can replace each cost of the residual graph by $\overline{c}^s_{ij} = d_{R(f)}(s, i) + rc_{ij} - d_{R(f)}(s, j)$. These costs are often called **reduced costs**. We will denote by $\overline{R}^s(f)$ this new graph. All the costs of $\overline{R}^s(f)$ are nonnegative and the best algorithms for computing shortest paths can be used.

The link between the shortest paths in $R(f)$ and $\overline{R}^s(f)$ is given by the following property:

**Property 4** *Let $f$ be a flow, and $\overline{R}^s(f)$ be the residual graph of $f$ in which costs are defined by $\overline{c}^s_{ij} = d_{R(f)}(s, i) + rc_{ij} - d_{R(f)}(s, j)$.*
*Then, $\forall (i, j) \in U(\overline{R}^s(f))$ $\overline{c}^s_{ij} \geq 0$,*
*and $d_{R(f)}(u, v) = d_{\overline{R}^s(f)}(u, v) - d_{R(f)}(s, u) + d_{R(f)}(s, v)$.*

**proof:** Let $P = [ua_1, a_1a_2, ..., a_pv]$ be a shortest path from $u$ to $v$. The length of $P$ is equal to $\overline{c}^s_{ua_1} + \overline{c}^s_{a_1a_2} + ... + \overline{c}^s_{a_pv}$, that is $d_{R(f)}(s, u) + rc_{ua_1} - d_{R(f)}(s, a_1) + d_{R(f)}(s, a_1) + rc_{a_1a_2} - d_{R(f)}(s, a_2) + ... + d_{R(f)}(s, a_p) + rc_{a_pv} - d_{R(f)}(s, v) = d_{R(f)}(s, u) + rc_{ua_1} + rc_{a_1a_2} + ... rc_{a_pv} - d_{R(f)}(s, v) = d_{R(f)}(s, u) + d_{R(f)}(u, v) - d_{R(f)}(s, v)$. $\odot$

When establishing a feasible flow from $f$, the zero flow or a flow which is no longer feasible because some lower bounds have been increased or some upper bounds have been decreased, the search for a shortest path from $i$ to $j$ is made in $R(f) - \{(i, j)\}$. In this case, the arc $(i, j)$ does not belong to the residual graph. Thus, in the context that we consider, the search for a path is made in the residual graph. Hence, the previous transformation can be applied during each computation (i.e. each augmentation or reduction) and it is possible to deal only with nonnegative costs.

## 4. Consistency for a costgcc

A gcc $C$ is consistent iff there is a special flow in an directed graph $N(C)$ called the value network of $C$ [11]:

**Definition 7** *Given $C = gcc(X, l, u)$ be a gcc; the* **value network** *of $C$ is the directed graph $N(C)$ with lower bound capacity and upper bound capacity on each arc. $N(C)$ is obtained from the value graph $GV(C)$, by:*

- *orienting each edge of $GV(C)$ from values to variables. For such an arc $(u, v)$: $l_{uv} = 0$ and $u_{uv} = 1$.*
- *adding a vertex $s$ and an arc from $s$ to each value. For such an arc $(s, a_i)$: $l_{sa_i} = l_i$, $u_{sa_i} = u_i$.*
- *adding a vertex $t$ and an arc from each variable to $t$. For such an arc $(x, t)$: $l_{xt} = 1$, $u_{xt} = 1$.*
- *adding an arc $(t, s)$ with $l_{ts} = u_{ts} = |X(C)|$.*

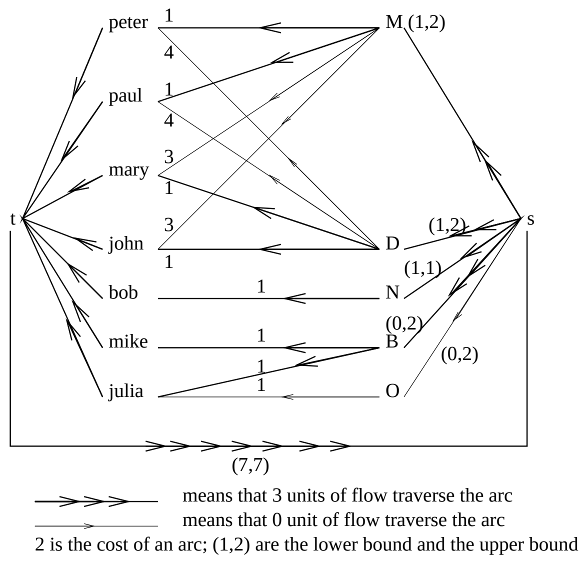
peter
paul
mary
john
bob
mike
julia
t
s
M (1,2)
D
N
B
O
(1,2)
(1,1)
(0,2)
(0,2)
(7,7)
1
4
1
4
3
1
3
1
1
1
1
1
means that 3 units of flow traverse the arc
means that 0 unit of flow traverse the arc
2 is the cost of an arc; (1,2) are the lower bound and the upper bound

*Figure 3.* An example of the value network associated with a costgcc. An optimal solution is indicated by the bold edges.

**Proposition 1** *Let $C$ be a gcc and $N(C)$ be the value network of $C$; the following two properties are equivalent:*
- *$C$ is consistent;*
- *there is a feasible flow in $N(C)$.*

**sketch of proof:** We can easily check that each tuple of $T(C)$ corresponds to a flow in $N(C)$ and conversely. $\odot$

Similarly, we can define the value network associated with a costgcc. (See Figure 3.)

**Definition 8** *Given $C = costgcc(X, l, u, cost, H)$; the* **value network** *of $C$ is the value network $N(C')$ of the underlying gcc $C' = gcc(X, l, u)$ of $C$, in which each arc has a cost defined as follows:*
- $\forall a \in D(X(C)) : c_{sa} = 0$
- $\forall x \in X(C) : c_{xt} = 0$
- $c_{ts} = 0$
- $\forall x \in X(C) : \forall a \in D(x) : c_{ax} = cost(x, a)$.

Note that this network is independent of $H$.

For convenience, let $m = |U(N(C))|$ (i.e. the number of arcs in $N(C)$), $n = |X(C)|$ (i.e. the number of variables involved in $N(C)$) and $d = |D(X(C))|$ (i.e. the number of values involved in $N(C)$) and $\gamma$ be the greatest cost involved in $N(C)$.

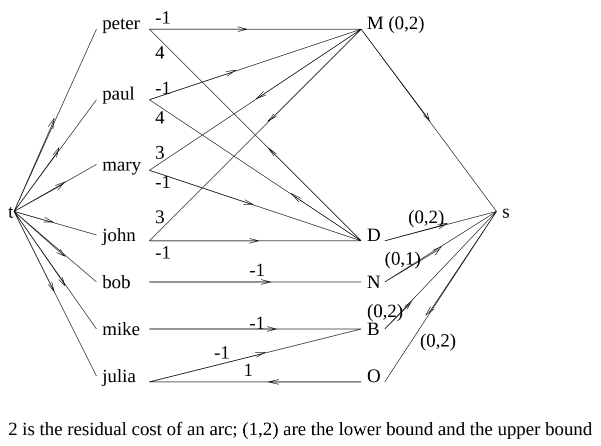

*Figure 4.* The residual graph of the optimal solution given in Figure 3.

We can present the original proposition:

**Proposition 2** *Given $C = costgcc(X, l, u, cost, H)$ and $N(C)$ the value network of $C$; the following two properties are equivalent:*
- *$C$ is consistent;*
- *there is a minimum cost flow in $N(C)$ with a cost less than or equal to $H$.*

**proof:** Let $C'$ be the gcc invoked in $C$. By proposition 1, $C'$ is consistent if and only if there is a feasible flow in $N(C)$. Thus, from each feasible flow in $N(C)$ a tuple of $T(C')$ can be built and from each element of $T(C')$ a feasible flow in $N(C)$ can be defined. Every feasible flow has a cost. If this cost is less than or equal to $H$ then a tuple of $T(C')$ is a tuple of $T(C)$. Moreover, a tuple of $T(C)$ is a tuple of $T(C')$ which corresponds to a feasible flow of cost less than $H$. $\odot$

The successive shortest path algorithm achieves the consistency of a costgcc in $O(nS(m, n + d, \gamma))$ (See properties 3 and 4.) Moreover, often in constraint programming, the consistency of the constraints is systematically checked during the search for solutions. In this case, we can use the incremental minimum cost flow algorithm presented in Section 3. This means that if $k$ values are deleted from the previous call of the consistency algorithm, then the consistency can be computed in $O(kS(m, n + d, \gamma))$.

## 5. Arc consistency for a costgcc

We have proved the following proposition [11].

**Proposition 3** *Let $C$ be a consistent gcc and $f$ be a feasible flow in $N(C)$. A value $a$ of a variable $x$ is not consistent with $C$ if and only if $f_{ax} = 0$ and $a$ and $x$ do not belong to the same strongly connected component in $R(f)$.*

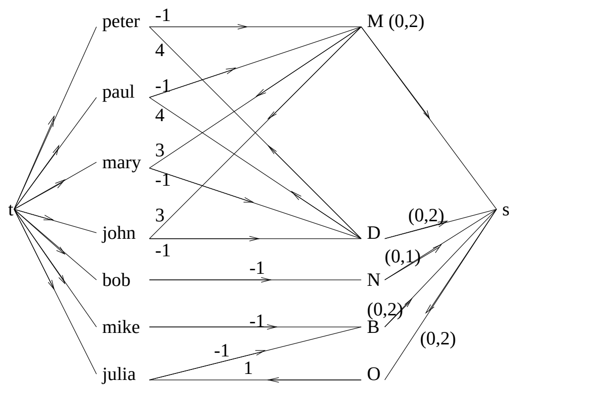

*Figure 5.* The path [julia,B,s,O] has a cost $-1$ and $rc_{Ojulia} = 1$, thus $(julia, O)$ is consistent with the constraint. The shortest path from john to M has a cost 2 and $rc_{Mjohn} = 3$, thus $7 + 3 + 2 > 11$ and $(john, M)$ is not consistent with the constraint.

**proof:** By theorem 1 the flow value for an arc $(a, x)$ is constant if there is no path from $a$ to $x$ in $R(f) - \{(a, x)\}$ and no path from $x$ to $a$ in $R(f) - \{(x, a)\}$. Moreover, $u_{ax} = 1$ thus $(a, x)$ and $(x, a)$ cannot belong simultaneously to $R(f)$, hence $f_{ax}$ is constant iff there is no cycle containing $(x, a)$ or $(a, x)$ in $R(f)$. That is, if $x$ and $a$ belong to different strongly connected components. $\odot$

The advantage of this proposition is that all the values not consistent with the gcc can be determined by only one identification of the strongly connected components in $R(f)$. The search for strongly connected components can be done in $O(m + n + d)$ [12], thus a remarkable complexity for computing arc consistency for a gcc is obtained.

**Corollary 1** *Let $C$ be a consistent gcc and $f$ be a feasible flow in $N(C)$. Arc consistency for $C$ can be achieved in $O(m + n + d)$.*

In order to avoid any problem of the existence of a path from a node to another node, we will consider that the arc consistency algorithm of the underlying gcc has been applied, and that we consider successively each strongly connected component.

Nevertheless, with a costgcc the problem is more complex than with a gcc, because with a gcc we need only to know whether there is a cycle containing a given arc or not. With a costgcc we need to identify whether there is a particular cycle containing a given arc, that is, a cycle with a length greater than a given value, because there are costs on arcs and the global bound must be satisfied.

For convenience we will consider that:

- $C = costgcc(X, l, u, cost, H)$ is a consistent costgcc;
- $f^o$ is a minimum cost flow problem in $N(C)$;

We can define an original correspondence between the consistency of a value with a costgcc and a particular path in the value network. (See figure 5.)

**Proposition 4** *A value $a$ of a variable $y$ is not consistent with $C$ if and only if the two following properties hold:*
*(i)* $f^o_{ay} = 0$
*(ii)* $d_{R(f^o)-\{(y,a)\}}(y,a) > H - cost(f^o) - rc_{ay}$

**proof:** (i) is obvious. Assume that $f^o_{ay} = 0$. From lemma 1, the cost of any flow $f'$ with $f'_{ay} = 1$ is $cost(f') = cost(f^o) + rc_{ay} + d_{R(f)-\{(y,a)\}}(y,a)$. Thus, $cost(f') > H$ if and only if $cost(f^o) + rc_{ay} + d_{R(f)-\{(y,a)\}}(y,a) > H$ and we have the proposition. ⊙

From this proposition we can define a simple algorithm for computing the arc consistency of a consistent costgcc:

for every arc $\{a, y\}$: if the proposition does not hold then remove the arc from $N(C)$ and $a$ from $D(y)$.

From property 4 we can work only with nonnegative costs. Thus, the complexity of this algorithm is $O(mS(m, n + d, \gamma))$.

At first glance, it does not seem easy to improve this algorithm because for each value $a$ of $y$ the distance are computed in $R(f^o) - \{(y, a)\}$. However, in our case, when we search for a path from $y$ to $a$, the arc $(y, a)$ does not belong to $R(f^0)$ since $f^o_{ay} = 0 = l_{ay}$. Thus $R(f^o) - \{(y, a)\} = R(f^o)$.

**Corollary 2** *The value $a$ of $y$ is not consistent with $C$ if and only if*

$$f^o_{ay} = 0 \text{ and } d_{R(f^o)}(y,a) > H - cost(f^o) - rc_{ay}.$$

Thus, if for each variable $y$ we compute the shortest path distance from $y$ to every node in $R(f^o)$, we will be able to know which values of $y$ are not consistent with the constraint. Therefore, since there are $n$ variables, we can achieve arc consistency for a costgcc in $O(nS(m, n + d, \gamma))$ which improves the previous complexity.

We can further use the particular structure of $R(f^0)$. In a solution each variable is assigned to one value. This means that for each variable there is only one arc outgoing to this variable in $R(f^o)$. Thus, all the shortest paths traversing a variable will use this arc. In other words if a variable $y$ is assigned to a value $b$ then all the shortest paths in $R(f^o)$ from $y$ to a value $a$ will use the arc $(y, b)$, and $d_{R(f^o)}(y, a) = rc_yb + d_{R(f^o)}(b, a)$. Hence, we can determine the values of $y$ that are not consistent with $C$ by searching for shortest paths in $R(f^o)$ from the value $b$, that is the value assigned to $y$:

**Corollary 3** *Let $y$ be any variable such that $f^o_{by} = 1$. Then, the value $a$ of $y$ is not consistent with $C$ if and only if*

$$f^o_{ay} = 0 \text{ and } d_{R(f^o)}(b,a) > H - cost(f^o) - rc_{ay} - rc_{yb}.$$

**proof:** By definition of the value network and since $C$ is consistent, $(y, b)$ is the only one arc outgoing $y$ in $R(f^o)$. Thus, $d_{R(f^o)}(y, b) = rc_{yb}$ and $d_{R(f^o)}(y, a) = d_{R(f^o)}(y, b) + d_{R(f^o)}(b, a) = rc_{yb} + d_{R(f^o)}(y, b)$, Hence by corollary 2 we have the corollary. ⊙

The advantage of this method (i.e. computing shortest paths from values instead of variables) is that we can gather some computations, because several variables can be assigned to the very same value.

Let $\Delta$ be the set of values $b$ such that $f^o_{sb} > 0$. Consider such a value $b$, we will denote by $\delta(b)$ be the set of values defined by $\delta(b) = \{a \in D(X(C))$ s.t. $a \neq b$ and $a \in D(y)$ and $f^o_{by} = 1\}$. Thus, arc consistency can be achieved by searching for each value $b$ of $\Delta$ the shortest path distance from $b$ to every node in $\delta(b)$.

```
ARCCONSISTENCY($C, N(C), f^o$)
// $f^o$ is a minimum cost flow in $N(C)$
$\Delta \leftarrow \varnothing$
Compute $d_{R(f^o)}(t,i)$ for every node $i$ in $R(f^o)$
for each value $a$ do if $f^o_{sa} > 0$ then $\Delta \leftarrow \Delta \cup \{a\}$
for each value $b \in \Delta$ do
    $\delta(b) \leftarrow \varnothing$
    for each arc $(b,y) \in N(C)$ do
        if $f^o_{by} = 1$ then
            for each $a \in D(y)$ do $\delta(b) \leftarrow \delta(b) \cup \{a\}$
    remove $b$ from $\delta(b)$
    COMPUTESHORTESTPATHS($b, \delta(b), \overline{R}^t(f^o)$)
    for each arc $(b,y) \in N(C)$ do
        if $f^o_{by} = 1$ then
            for each $a \in D(y)$ do
                if $d_{\overline{R}^t(f^o)}(b,a) > H - cost(f^o) - \overline{c}^t_{ay} - \overline{c}^t_{yb}$ then remove $a$ from $D(y)$
```

*Algorithm 1.* Arc consistency algorithm for a costgcc.

We can deal only with nonnegative costs by using Property 4. Then, we have:

**Corollary 4** *Let $y$ be any variable such that $f^o_{by} = 1$. Then, the value $a$ of $y$ is not consistent with $C$ if and only if*

$$f^o_{ay} = 0 \text{ and } d_{\overline{R}^t(f^o)}(b,a) > H - cost(f^o) - \overline{c}^t_{ay} - \overline{c}^t_{yb}.$$

**proof:** By Corollary 3: a value $a$ of $y$ is not consistent with $C$ iff $f^o_{ay} = 0$ and $d_{R(f^o)}(b,a) > H - cost(f^o) - rc_{ay} - rc_{yb}$. From Property 4 it can be rewritten as $d_{\overline{R}^t(f^o)}(b,a) - d_{R(f^o)}(t,b) + d_{R(f^o)}(t,a) > H - cost(f^o) - [\overline{c}^t_{ay} - d_{R(f^o)}(t,a) + d_{R(f^o)}(t,y)] - [\overline{c}^t_{yb} - d_{R(f^o)}(t,y) + d_{R(f^o)}(t,b)]$ which is equivalent to $d_{\overline{R}^t(f^o)}(b,a) > H - cost(f^o) - \overline{c}^t_{ay} - \overline{c}^t_{yb}$ ⊙

Algorithm 1 is a possible implementation of the achievement of arc consistency for a costgcc. Function COMPUTESHORTESTPATHS($b, \delta(b), \overline{R}^t(f^o)$) computes the shortest path in $\overline{R}^t(f^o)$ from a node $b$ to every node in $\delta(b)$.

Since we have $|\Delta| \leq min(n,d)$, the previous complexity is improved.

**Property 5** *Let $C$ be a consistent costgcc, $f^o$ be a minimum cost flow in $N(C)$. Arc consistency for $C$ can be achieved in $O(|\Delta| S(m, n+d, \gamma))$.*

**Practical improvements**

We can propose some heuristics for improving the behavior of the arc consistency algorithm in practice .

Let $(a, y)$ be any arc with $f^o_{ay} = 0$ and $b$ be the value with $f^o_{by} = 1$. First, if $\overline{c}^t_{ay} > H - cost(f^o) - \overline{c}^t_{yb}$ then we can immediately remove the value $a$ from $D(y)$, since any shortest path distance is nonnegative. Using for each arc $(b, y)$ with $f^o_{by} = 1$, $m(b) = \min(\{\overline{c}^t_{zb} + \overline{c}^t_{zc}$ with $z \neq y$ and $c \in D(z)\})$ can refine this idea. Let $M = H - cost(f^o)$, then, $(y, a)$ is not consistent with $C$ if one of the following conditions holds:

- $f_{sa} < u_{sa}$ and $f_{sb} > l_{sb}$ and $\overline{c}^t_{ay} > M - \overline{c}^t_{yb}$
- $f_{sa} < u_{sa}$ and $f_{sb} = l_{sb}$ and $\overline{c}^t_{ay} + m(b) > M$
- $f_{sa} = u_{sa}$ and $f_{sb} > l_{sb}$ and $\overline{c}^t_{ay} + m(a) > M - \overline{c}^t_{yb}$
- $f_{sa} = u_{sa}$ and $f_{sb} = l_{sb}$ and $\overline{c}^t_{ay} + m(a) + m(b) > M$

Furthermore, it is not necessary to search for the shortest path distances from a value $b$ in $\Delta$ to every node in $\delta(b)$. When all the current distances from $b$ to nodes in $\delta(b)$ do not satisfy the inequality of corollary 4 we can stop the algorithm, because all the values of $\delta$ are consistent with the constraint. Moreover, if we use Disjkstra's algorithm for searching for shortest paths then if the current scanned node of the Disjkstra's algorithm is greater than $H - cost(f^o)$ then we can also stop the algorithm, because the shortest path distances of the nodes that have not been scanned will not be less than the $H - cost(f^o)$. And since all the costs are nonnegative, all the values that have not been scanned will satisfy corollary 4.

There are several possible ways for implementing the priority queue needed by the Dijkstra's algorithm [6]. We can implement the Dijkstra's algorithm in $O(m + H)$ with a simple bucket data structure or in $O(m + n \log \gamma)$ with a radix heap data structure. We can also use a Fibonacci heap in order to obtain a strongly polynomial algorithm $O(m + n \log(n))$.

## 6. Discussion

### *6.1. Removal of negative costs*

Consider $C = costgcc(X, l, u, cost, H)$ and suppose that some costs are negative. This constraint can be transformed into an equivalent one in which there are only nonnegative costs [9]. Let $K$ be the opposite of the minimum value of costs. If this value is added to all costs then all the obtained costs are nonnegative. In general this method does not work, because the problem is transformed. Indeed, two paths with a different number of arcs and the same length in the initial problem will have different length in the new problem. However, in our case, each variable will be instantiated to exactly one value, thus by adding $K$ to each cost, the cost of the minimum cost flow will be increased by $|X|K$. This means that the constraint $costgcc(X, l, u, cost{+}K, H{+}|X|K)$, where $(cost{+}K)(e) = cost(e){+}K$, is equivalent to the constraint $costgcc(X, l, u, cost, H)$, and contains only nonnegative costs.

## *6.2. Constraint on the sum of all different variables*

An interesting example of costgcc is the constraint on the sum of all different variables. Some real-world problems involve this constraint. More precisely, for a given set of variable $X$, this constraint is the conjunction of the constraint $\sum_{x_i \in X} x_i \leq H$ and alldiff$(X)$.

Let us define the boundaries and cost function as follows:

- For each value $a_i \in D(X)$ we define $l_i = 0$ and $u_i = 1$
- For each variable $x \in X$ and for each value $a \in D(x)$, $cost(x, a) = a$

Then, it is easy to prove that the costgcc constraint $costgcc(X, l, u, cost, H)$ represents the conjunction of the constraint $\sum_{x_i \in X} x_i \leq H$ and alldiff$(X)$.

Thus, with the algorithm we propose, we are able for the first time to achieve arc consistency for this constraint.

Moreover, note that the constraint which is the conjunction of the constraint $\sum_{x_i \in X} \alpha_i x_i \leq H$ and alldiff$(X)$, can also be represented by a costgcc, by defining in the previous model $cost(x_i, a) = \alpha_i a$.

## *6.3. Constraint on the minimum value of the sum*

Suppose that instead of constraining the maximum value of the sum of an instantiation of any solution of a gcc, we would like to constraint the minimum value of this sum. More precisely for each tuple $\tau$ of a given gcc $C = gcc(X, l, u)$, we impose that $\Sigma_{i=1}^{|X(C)|} cost(\texttt{var}(C, i), \tau[i]) \geq L$.

In this case we have to solve a maximum cost flow problem instead of a minimum cost flow problem. This problem can be solved by replacing all costs by their opposite value. Therefore this constraint can be represented by the costgcc defined by $costgcc(X, l, u, -cost, -L)$.

However, it is not easy to take into account at the same time a constraint on the minimum value of the sum and on the maximum value. We can prove that there is a tuple which satisfies the lower bound condition on the sum, we can also prove that there is a tuple which satisfies the upper bound condition on the sum, but, unfortunately, we have absolutely no guarantee on the existence of a tuple that satisfies both these two conditions. For instance, consider the problem involving three variables $x_1$, $x_2$ and $x_3$ with $D(x_1) = \{a, b\}$, $D(x_2) = \{b, c\}$ and $D(x_3) = \{a, c\}$. Each value has to be taken at most 1. A cost function is defined as follows: $cost(x_1, a) = cost(x_2, b) = cost(x_3, c) = 1$ and $cost(x_1, b) = cost(x_2, c) = cost(x_3, a) = 3$. The sum of any instantiation of all the variables must be greater than 4 and less than 8. Clearly, there are only two possible ways for satisfying the *gcc*: $((x_1, a), (x_2, b), (x_3, c))$ and $((x_1, b), (x_2, c), (x_3, a))$. The cost of the first solution is 3 and satisfies $3 \leq 8$, the cost of the second one is 9 and satisfies $9 \geq 4$. However, this problem has no solution. We do not know any general algorithm for obtaining such a result.

### *6.4. Interest of shortest path distances as heuristic*

We can use the information given by the shortest path distance for guiding the search for solutions, and in particular, for choosing the next variable to instantiate.

In many optimization problems, the **max-regret** heuristic is considered as one of the best heuristics. For each variable the regret can be defined as the difference between the cost of the best assignment and the cost of the second best. Then, the variable with the regret of maximal value is chosen. Intuitively, the idea is that if we do not choose this variable and if this variable is instantiated with a value different from the one leading to the best assignment, we will have to pay at least the value of the regret.

Usually, the regret is not exactly computed and an approximation of the regret is considered. For instance, the regret is often defined for every variable as the difference between the minimum cost involving this variable and the second minimum. With our approach it becomes possible to exactly compute the value of the regret, and so to improve the search for solution.

## 7. Conclusion

In this paper we have proposed an efficient way of implementing generalized arc-consistency for the global cardinality constraint with costs. We have shown that costgcc constraints are powerful constraints for modelling several conjunctions of constraints often arising in practice. We have also explained how the algorithms we propose can help in the definition of an interesting heuristic.

## References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, 1993.
2. N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
3. C. Berge. *Graphe et Hypergraphes.* Dunod, Paris, 1970.
4. Y. Caseau, P-Y. Guillo, and E. Levenez. A deductive and object-oriented approach to a complex scheduling problem. In *Proceedings of DOOD'93*, 1993.
5. Y. Caseau and F. Laburthe. Solving various weighted matching problems with constraints. In *Proceedings CP97*, pages 17–31, Austria, 1997.
6. B.V. Cherkassky, A.V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.
7. J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.
8. E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.
9. J-F. Puget. Personal communication, 1999.
10. J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI-94*, pages 362–367, Seattle, Washington, 1994.
11. J-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proceedings AAAI-96*, pages 209–215, Portland, Oregon, 1996.
12. R.E. Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.