Generalized Arc Consistency for Global Cardinality Constraint

Jean-Charles RÉGIN
ILOG S.A.
9, rue de Verdun BP 85
94253 Gentilly Cedex – FRANCE
e-mail : regin@ilog.fr

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 a value $v_i \in V$ is assigned to a variable in X to be in an interval $[l_i, c_i]$. Cardinality constraints have proved very useful in many real-life problems, such as scheduling, timetabling, or resource allocation. A gcc is more general than a constraint of difference, which requires each interval to be $[0, 1]$. In this paper, we present an efficient way of implementing generalized arc consistency for a gcc. The algorithm we propose is based on a new theorem of flow theory. Its space complexity is $O(|X| \times |V|)$ and its time complexity is $O(|X|^2 \times |V|)$. We also show how this algorithm can efficiently be combined with other filtering techniques.

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. Arc consistency is one of the most basic and useful such method. Several algorithms achieving arc consistency have been proposed for binary CSPs (Mackworth 1977; Mohr & Henderson 1986; Bessière 1994; Bessière, Freuder, & Régin 1995) and for n-ary CSPs (Mohr & Masini 1988a). Only limited work has been carried out on the semantics of constraints. (Mohr & Masini 1988b) have described an improvement of the algorithm AC-4 for special constraints introduced by a vision problem; (Van Hentenryck, Deville, & Teng 1992) have studied monotonic and functional binary constraints; (Nuijten 1994) has proposed several filtering algorithms for constraints found in scheduling problems; and (Régin 1994) has given an efficient way of implementing generalized arc consistency for constraint of difference. In this work, we are interested in a special case of n-ary constraints: global cardinality constraints (gcc), for which we propose a filtering algorithm.

	Mo	Tu	We	Th	...
peter	D	N	O	M	
paul	D	B	M	N	
mary	N	O	D	D	
...					

A = {M,D,N,B,O}, P = {peter, paul, mary, ...}
W = {Mo, Tu, We, Th, ...}
M: morning, D: day, N: night B: backup, O: day-off

Figure 1: An Assignment Timetable.

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 to a variable in X to be in an interval $[l_i, c_i]$. Gccs arise in many real-life problems. For instance, consider the example derived from a real problem and given in (Caseau, Guillo, & Levenez 1993) (cf. Figure 1). The task is to schedule managers for a directory-assistance center, with 5 activities (set A), 7 persons (set P) over 7 days (set W). Each day, a person can perform an activity from the set A. The goal is to produce an assignment matrix that satisfies the following global and local constraints:

- **global constraints** restrict the assignments. First, for each day we may have a minimum and maximum number for each activity. Second, for each week, a person may have a minimum and maximum number for each activity. Thus, for each row and each column of the assignment matrix, there is a global cardinality constraint.
- **local constraints** mainly indicate incompatibilities between two consecutive days. For instance, a morn-

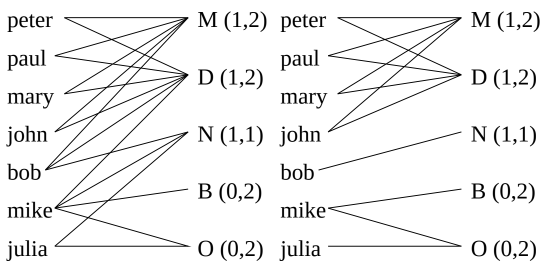


Figure 2: An example of global constraint of cardinality.

ing schedule cannot be assigned after a night schedule.

There are two ways for trying to handle the global constraints:

- By representing each global constraint by an n-ary constraint and by using the generalized arc consistency algorithm GAC-4 (Mohr & Masini 1988a) to filter them. This filtering efficiently reduces the domains, but its complexity depends on the length and the number of all admissible tuples. That number can be exponential because these global constraints are more general than difference constraints; therefore, GAC-4 cannot be used in pratice even for small problems.

- By representing each global constraint by as many min/max constraints as the number of involved activities. Now, these min/max constraints can be easily handled with, for instance, the *atmost/atleast* operators proposed in (Van Hentenryck & Deville 1991). Such operators are implemented using local propagation. But as it is noted in (Caseau, Guillo, & Levenez 1993): "The problem is that efficient resolution of a timetable problem requires a global computation on the set of min/max constraints, and not the efficient implementation of each of them separately." Hence, this way is not satisfactory.

In order to show the difference in global and local filtering, consider a gcc associated with a day (cf figure 2). The constraint can be represented by a bipartite graph called a value graph (left graph in Figure 2). 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 between parentheses express the minimum and the maximum number of times the activity has to be assigned. For instance, John can work the morning or the day but not the night; one manager is required to work the morning, and at most two managers work the morning. We recall that each person has to be associated with exactly one activity.

Local filtering deletes no values. Now, we can carefully study this constraint. Peter, Paul, Mary, and John can work only in the morning and during the day. Moreover, morning and day can be assigned together to at most 4 persons. Thus, no other persons (i.e. Bob, Mike, nor Julia) can perform activities M and D. So we can delete the edges between Bob, Mike, Julia and D, M. Now only one possibility remains for Bob: N, which can be assigned at most once. Therefore, we can delete the edges {mike,N} and {julia,N}. This reasoning leads to the right graph in Figure 2. It corresponds to the achievement of generalized arc consistency for the constraint.

In this paper we present an efficient way of implementing generalized arc consistency for the gcc in order to benefit from its pruning performances.

First, we give some preliminaries on graphs, constraint satisfaction problems and flows. Then, we present an algorithm checking the consistency of a gcc. We introduce a new theorem in flow theory. From it, we propose a very simple algorithm for achieving generalized arc consistency. We show how this new filtering algorithm can be easily and efficiently combined with other filtering techniques. Finally, we recapitulate our conclusions.

Preliminaries

Graph

The following definitions are due to (Tarjan 1983).

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. An *oriented graph* is a digraph having no symmetric pairs of arcs. We will denote by $X(G)$ the vertex set of G and by $U(G)$ the arc set of G.

An arc (u, v) *leaves* u and *enters* v. A *path* in a graph *from* v_1 *to* v_k is a sequence of vertices $[v_1, v_2, ..., v_k]$ such that (v_i, v_{i+1}) is an arc for $i \in [1, ..., k-1]$. The path is *simple* if all its vertices are distinct. A path is a *cycle* if $k > 1$ and $v_1 = v_k$.

CSP

A finite *constraint satisfaction problem* (CSP) $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is defined as a set of n *variables* $X = \{x_1, ..., x_n\}$, a set of finite *domains* $\mathcal{D} = \{D_1, ..., D_n\}$ where D_i is the set of possible *values* for variable x_i, and a set $\mathcal{C}$ of *constraints* between variables.

Let $X_{i,...,k}$ be a subset of X. A tuple of values from $D_i, ..., D_k$ is called a *tuple of* $X_{i,...,k}$. For instance, if $a \in D_i$, $b \in D_j$ and $c \in D_k$, then (a, b, c) is a tuple of

X_{ijk}. A constraint C is defined on a set of variables $X(C) = \{x_i, ..., x_k\}$ by a subset of the Cartesian product $D_i \times ... \times D_k$ (i.e. a set of tuple, denoted by $T(C)$).

With C a constraint:
$D(C)$ denotes the union of domains of variables of $X(C)$ (i.e. $D(C) = \cup_{i \in X(C)} D_i$).
$k = |X(C)|$ denotes the arity of C.
d denotes the cardinality of $D(C)$.
Moreover, we will denote by $\#(a, P)$ the number of occurences of the value a in the tuple P.

For convenience, we introduce the concepts of consistency and arc consistency for one constraint.

Let $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ be a CSP, $C \in \mathcal{C}$ be a constraint, x be the i^{th} variable of $X(C)$, and a be a value of x:

• C is *consistent* iff $T(C) \neq \varnothing$.

• a is *consistent with* C iff $\exists P \in T(C)$ such that a appears at the i^{th} position in P.

• C is *arc consistent* iff $\forall x_i \in X(C), \forall a \in D_i, a$ is consistent with C.

• $\mathcal{P}$ is *arc consistent* iff $\forall C \in \mathcal{C}$, C is arc consistent.

The arc consistency for n-ary constraints is usually called *generalized arc consistency.*

The *value graph* (Laurière 1978) of an n-ary constraint C is the bipartite graph $GV(C) = (X(C), D(C), E)$ where $(x_i, a) \in E$ iff $a \in D_i$. This graph establishes an obvious correspondence between any tuple of any n-ary constraint and a special set of edges in the value graph.

Proposition 1 *Let C be a constraint. Every tuple of $T(C)$ corresponds to a set S of edges in $GV(C)$ such that every vertex $x_i \in X_C$ is an end*[1] *of exactly one edge in S.*

Throughout this paper, we are interested in global cardinality constraints (gcc). They are defined by the minimal and the maximal number of times the values of $D(C)$ must appear in each tuple of the constraints. The minimal and the maximal number of occurences 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 $a_i \in D(C)$ is associated with two positive integers l_i and c_i and*

$$T(C) = \{ \quad P \text{ such that } P \text{ is a tuple of } X(C) \text{ and } \forall a_i \in D(C) : l_i \leq \#(a_i, P) \leq c_i\}$$

In previous work (Régin 1994), we have proposed an efficient implementation to achieve generalized arc consistency for the difference constraints. These constraints can be defined by prescribing, in the previous definition, each lower bound to be 0 and each upper bound to be 1. So, a gcc is more general than a difference constraint. Moreover, the filtering we have proposed is based on matching theory in a bipartite graphs (which can be viewed as a specialization of flow theory). Hence, in order to be more general, we introduce flow theory and show how it can help to efficiently implement generalized arc consistency for a gcc.

[1] u and v are the ends of the edge $\{u, v\}$.

Flows

Flow theory was originally developed by Ford and Fulkerson (Ford & Fulkerson 1962).

Let G be an oriented graph for which each arc (u, v) is associated with two positive integers $l(u, v)$ and $c(u, v)$. The number $c(u, v)$ is called the *capacity* of the arc (u, v) and $l(u, v)$ the *lower bound.*

A *flow* in G is a function f satisfying the following two conditions:

• For any arc (u, v), $f(u, v)$ 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 u to v.

• A *conservation law* is observed at each of the vertices[2]:

$$\forall v \in X(G) : \textstyle\sum_u f(u, v) = \sum_w f(v, w).$$

We will consider two problems of flow theory:

• *the feasible flow problem*: Does there exist a flow in G that satisfies the *capacity constraint*, that is, $\forall (u, v) \in U(G) : l(u, v) \leq f(u, v) \leq c(u, v)$?

• *the problem of the maximum flow for an arc* (u, v): Find a feasible flow in G for which the value of $f(u, v)$ is maximum. This problem is also called *maximum flow from v to u*, and $f(u, v)$ is called the *value* of such a flow.

If lower bounds are equal to zero, note that a maximum flow for any arc always exists; otherwise there is not necessarily a feasible flow.

The following theorem shows the interest of integral capacities and integral lower bounds:

Theorem 1 (Flow Integrality Theorem) *If all the capacities and all the lower bounds are integers and if there exists a feasible flow, then for any arc (u, v) there exists a maximum flow from v to u which is integral on every arc in G.*

Consistency of a gcc

From a gcc C we propose to build a particular oriented graph, denoted by $N(C)$. Then, we show that a gcc is consistent iff there is a special flow in this graph.

[2] For convenience, we assume $f(u, v) = 0$ if $(u, v) \notin U(G)$.

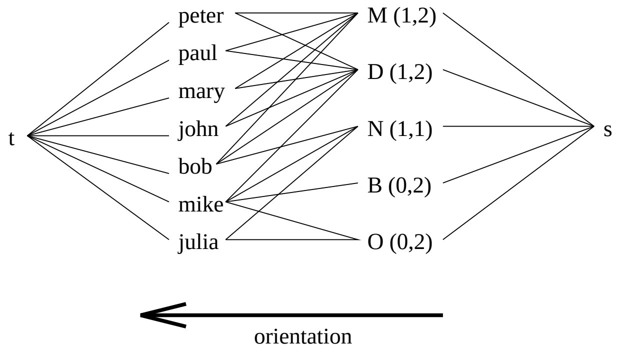


Figure 3: The value network of the gcc given in Figure 2.

Definition 2 *Let C be a gcc; the* **value network** *of C is the oriented graph $N(C)$ with capacity and lower bound 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): $l(u, v) = 0$ and $c(u, v) = 1$.*
- *adding a vertex s and an arc from s to each value. For such an arc (s, a_i): $l(s, a_i) = l_i$ and $c(s, a_i) = c_i$.*
- *adding a vertex t and an arc from each variable to t. For such an arc (x_i, t): $l(x_i, t) = 0$ and $c(x_i, t) = 1$.*
- *adding an arc (t, s) with $l(t, s)$=0 and $c(t, s)$=∞.*

We will denote by δ the number of arcs in $U(N(C))$.

Figure 3 gives an example of a value network. All arcs are oriented from s to t. For clarity, the arc (t, s) is omitted.

Proposition 2 *Let C be a gcc of arity k and $N(C)$ be the value network of C; the following two properties are equivalent:*

- *C is consistent;*
- *there is a maximum flow from s to t in $N(C)$ of value k.*

proof: First, note that no flow from s to t in $N(C)$ can have a value greater than k because this value can be obtained only if all arcs entering t are saturated.

Suppose C is consistent. So, $T(C) \neq \varnothing$. Let us consider $P \in T(C)$. We can build a function f in $N(C)$ as follows:

(a) $\forall x_i \in X(C), f(x_i, t) = 1$;

(b) $\forall x_i \in X(C), f(a, x_i) = 1$ if a appears at the i^{th} position in P; otherwise, $f(a, x_i) = 0$;

(c) $\forall a_j \in D(C), f(s, a_j) = \#(a_j, P)$.

It is easy to check that this function satisfies the conservation law and the capacity constraint. Furthermore, $\sum_{a_i} f(s, a_i) = k$.

On the other hand, suppose there is a feasible flow f from s to t of value k integral on every arc in $N(C)$ (by Theorem 1, a maximum flow integral on every arc always exists when a feasible flow exists), so $\sum_x f(x, t) = k$ and $\forall x_i \in X(C) : f(x_i, t) = 1$. By the conservation law, $\sum_a f(a, x_i) = \sum_w f(x_i, w) = f(x_i, t) = 1$. Since f is integral on every arc, there is exactly one a_j for each $x_i \in X(C)$ such that $f(a_j, x_i) = 1$. Thus, the set of arcs such that $f(a_j, x_i) = 1$ corresponds to a set of edges in the value graph, and by Proposition 1, this corresponds to a tuple of $X(C)$. By the capacity constraint: $\forall a_i \in D(C) : l_i \leq f(s, a_i) \leq c_i$, and by the conservation law $f(s, a_i) = \sum_x f(a_i, x) \leq c_i$, therefore $l_i \leq \#(a_i, P) \leq c_i$, and C is consistent.□

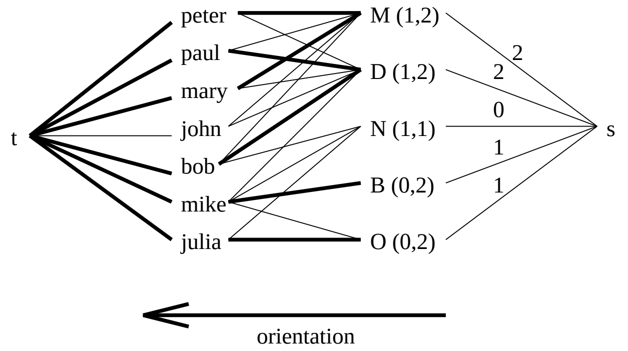


Figure 4: An infeasible flow from s to t.

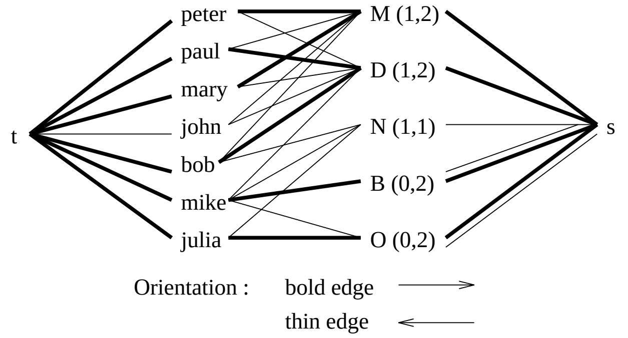


Figure 5: The residual graph for the flow given in Figure 4.

This proposition gives us a way to compute the consistency of a gcc. Now the problem is the search for a maximum flow from s to t in $N(C)$.

Computation of maximum flow

This presentation is inspired by (Lawler 1976; Berge 1970; Tarjan 1983; Ford & Fulkerson 1962).

Let G be an oriented graph for which each arc is associated with a lower bound and with a capacity, (t, s) be an arc in G, and f be a flow from s to t in G.

The *residual graph* for f, denoted by $R(f)$, is the digraph with the same vertex set as in G. The arc set

of $R(f)$ is defined as follows:
$\forall(u,v) \in U(G)$:

• If $f(u,v) < c(u,v)$, then $(u,v) \in U(R(f))$ and $res(u,v) = c(u,v) - f(u,v)$. Such an arc (u,v) will be denoted by a^+.

• If $f(u,v) > l(u,v)$ then $(v,u) \in U(R(f))$ and $res(v,u) = f(u,v) - l(u,v)$. Such an arc (v,u) will be denoted by a^-.

$res(u,v)$ is called the *residual capacity* of (u,v) in $R(f)$. The *residual capacity* of p a path in $R(f)$, denoted by $res(p)$, is the minimum value of $res(u,v)$ for each arc (u,v) of p. $A^+(p)$ (resp. $A^-(p)$) is the set of arcs a^+ (resp. a^-) of p.

Intuitively, an arc $a^+ = (u,v)$ can receive $res(u,v)$ units of commodity, and an arc a^- can give $res(u,v)$ units of commodity. Thus, if we find p a simple path in $R(f)$ from x to y which does not contain (x,y) and if $f(y,x) < c(y,x)$, then we can augment f along p to get a flow f' from x to y in G with $f'(y,x) = f(y,x) + val$, where $val = min(res(p), res(y,x))$. To ensure f' is a flow, it is defined as follows:

$\forall(u,v) \in A^+(p) : f'(u,v) = f(u,v) + val;$
$\forall(v,u) \in A^-(p) : f'(u,v) = f(u,v) - val;$
$f'(y,x) = f(y,x) + val$

For all other arcs (u,v) in $U(G)$: $f'(u,v) = f(u,v)$.
Moreover, if f is feasible, then so is f'.

We will say that there exists *an augmenting path p from s to t for f* iff $f(t,s) < c(t,s)$ and p is a simple path from s to t in $R(f)$ which does not contain (s,t).

Figures 4 shows a flow from s to t, and Figure 5 shows its residual graph. The number over an arc (s,a) indicates the value of $f(s,a)$. In absence of such number, a bold arc means the value 1 for the flow, while a thin arc represents the value 0. $(s, N, bob, D, john, t)$ is an augmenting path from s to t for f. (For clarity the arcs (t,s) and (s,t) are omitted.)

In the next section, we will assume a feasible flow from s to t is known. In a latter section, we will show how such a feasible flow can be computed.

Computation of a flow from s to t from a feasible flow

Theorem 2 *A flow f from s to t is maximum if and only if there is no augmenting path from s to t for f.*

proof: See page 99 in (Tarjan 1983), for instance.□

This Theorem gives a way to construct a maximum flow from s to t by an iterative improvement, the *augmenting path method* of Ford and Fulkerson: begin with any feasible flow f_0 from s to t and look for an augmenting path from s to t for f_0. If there is none, f_0 is maximum. If, on the other hand, we find such a path p, then augment f_0 along p to get a flow f_1 from s to t. Now look for an augmenting path from s to t for f_1 and repeat this process. Terminate if f_k is a maximum flow.

The question that still remains is that of determining a feasible flow.

Computation of a feasible flow The method is similar to the previous one (see page 139 in (Lawler 1976)). Suppose f is an infeasible flow. Pick an arc (y,x) such that $f(y,x) < l(y,x)$. Find p an augmenting path from x to y for f. Augment f along p to get a flow f'; set $f = f'$ and repeat until $f(y,x) \geq l(y,x)$. Then repeat for another arc for which the flow is infeasible. If, at some point, there is no augmenting path for the current flow, then a feasible flow does not exist. Otherwise, the obtained flow is feasible.

The zero flow ($\forall(u,v) \in U(G) : f(u,v) = 0$) can be used to start the computation in the absence of another initial flow.

Complexity

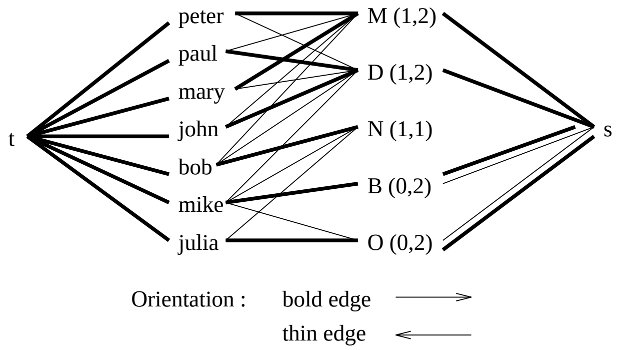
Corollary 1 *Let C be a gcc. The augmenting path method of Ford and Fulkerson finds a maximum flow f^* from s to t in $N(C)$ or proves there is none in $O(k\delta)$ which is less than or equal to $O(k^2 d)$.*[3]

proof: The search for p an augmenting path, called an augmenting step, can be computed in $O(\delta + k + d)$, for instance, by a breadth-first search. All lower bounds and all capacities are integers, so $res(p)$ is an integer strictly greater than 0. Hence, each augmentation increases the value of the flow for the considered arc by at least 1. For the computation of a maximum flow from a feasible flow, the number of augmenting steps is clearly bound by $O(f^*(t,s)) \leq O(k)$. For the computation of a feasible flow, $\sum_a l(s,a) \leq k$ (else there is no solution); so at most $\sum_a l(s,a)$ augmenting steps are needed. Therefore, the consistency of a gcc C can be checked in $O(\delta k) \leq O(k^2 d)$.□

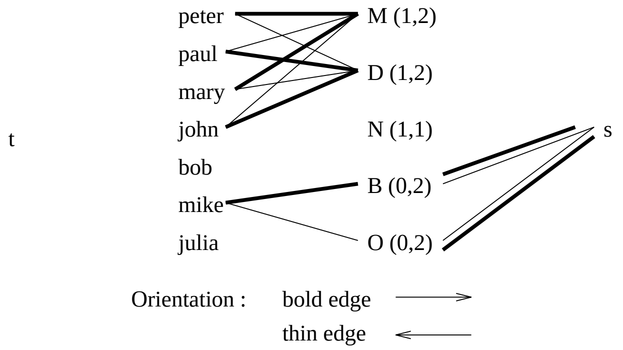
This complexity is good in regard to the space complexity ($O(kd)$) and to the complexity obtained to check the consistency of one difference constraint ($O(kd\sqrt{k})$).

In general, Ford and Fulkerson's method is not efficient, because if the capacities are large integers, the value of a maximum flow may be large, and the augmenting path method makes many augmentations. Furthermore, if the capacities are irrational, the method may not halt. In our case, since the value of a maximum flow from s to t is less than the number of vertices in $N(C)$, and because most of lower bounds are 0, the complexity of Ford and Fulkerson's algorithm

[3] We recall that k=$|X_C|$, d=$|D(C)|$ and δ=$|U(N(C))|$.



The residual graph for a maximum flow from s to t.



Arcs joining two vertices that belong to the same strongly connected components in the residual graph.

Figure 6:

is acceptable. For our problem, indeed, no other algorithm leads to a complexity lower than $O(\delta k)$. However, Dinic's algorithm (Dinic 1970) should give the best results in pratice.

Generalized arc consistency for gcc

Let f be a flow from x to y in a graph G. If we find p, an augmenting path from x to y for f, we know that the value of $f(y,x)$ can be augmented by $min(res(p), res(y,x))$. Similarly, if we find p, a simple path from y to x in $R(f)$ which does not contain (y,x), and if $f(y,x) > l(y,x)$, then f can be reduced along p to get a flow f' with $f'(y,x) = f(y,x) - min(res(p), res(y,x))$. Furthermore, if f is feasible, then so is f'. We will say that there exists *a reducing path p from t to s for f* iff $f(t,s) > l(t,s)$ and p is a simple path from t to s in $R(f)$ which does not contain (t,s). The following theorem is similar to Theorem 2:

Theorem 3 *A flow f from s to t is minimum if and only if there is no reducing path from t to s for f.*

Let $R(f) - \{(s,t)\}$ denote the residual graph for f minus the arc (s,t). We introduce a new Theorem in flow theory:

Theorem 4 *Let G be an oriented graph for which each arc is associated with two positive integers; f be an arbitrary maximum flow in G from s to t; and (u,v) an arc in G. For all maximum flows f' in G from s to t, $f'(u,v) = f(u,v)$ if and only if (u,v) and (v,u) are not contained in a simple cycle in $R(f) - \{(s,t)\}$ involving at least 3 vertices.*

proof: (v,u) is not contained in a cycle in $R(f) - \{(s,t)\}$ involving at least 3 vertices means that there is no augmenting path from u to v for f. So, by Theorem 2, f is also a maximum flow from u to v.

(u,v) is not contained in a cycle in $R(f) - \{(s,t)\}$ involving at least 3 vertices means that there is no reducing path from v to u for f. So, by Theorem 3, f is also a minimum flow from u to v. $\square$

Consider C, a gcc, f a maximum flow from s to t in $N(C)$, and an arc (a,x) which leaves a value and enters a variable such that $f(a,x) = 0$. Then, by construction of $R(f)$, (x,a) does not belong to $R(f)$. So, by the previous theorem, if (a,x) is not contained in a cycle in $R(f)$, then for all maximum flows f^* from s to t in $N(C)$, $f^*(a,x) = 0$. Thus, a is not consistent with C. Moreover, an arc is contained in a cycle iff it leaves and enters two vertices belonging to the same strongly connected components (cf. Figure 6). Hence, we have the following corollary:

Corollary 2 *Let C be a consistent gcc and f be a maximum flow in $N(C)$ from s to t. A value a of a variable x is not consistent with C if and only if $f(a,x) = 0$ and a and x do not belong to the same strongly connected component in $R(f) - \{(s,t)\}$.*

The search for strongly connected components of a graph can be done in $O(m+n)$ for a graph with m edges and n vertices (Tarjan 1972). Consequently:

Corollary 3 *For one consistent gcc, generalized arc consistency can be achieved in $O(\delta + k + d)$.*

Combination with other filtering

The deletion of values for one gcc can involve modifications for the other constraints. Consider again the example given in Figures 1 and 2, and R another gcc defined by a row. Suppose the previous filtering applied to R leads to the removal of value M from the domains of the variables peter and paul. The constraint C also is modified, because edges {peter, M} and {paul, M} have been deleted from its value graph. Then, it is interesting to achieve again generalized arc consistency for C. But, we can do better than entirely repeat the previous algorithm because C was consistent before the deletions. Thus, a maximum flow f from s to t in $N(C)$ is known. This flow can be used to search for a maximal flow from s to t in $N'(C)$, the network obtained after the deletions.

Corollary 4 *Let C be a consistent gcc; f be a maximum flow from s to t in $N(C)$; $N'(C)$ be the new value network of C obtained by removing V values from domains of variables. A maximum flow from s to t in $N'(C)$ can be computed in $O(V|U(N'(C))|)$ and generalized arc consistency for C can be achieved in $O(V|U(N'(C))| + |U(N'(C))| + k)$.*

proof: Let $D'(C)$ be the value set of $N'(C)$. Consider the flow f' defined in $N'(C)$ by:

$$\forall a_i \in D'(C) : f'(a_i, x_j) = f(a_i, x_j)$$
$$\forall a_i \in D'(C) : f'(s, a_i) = \sum_{(a_i,x_j)\in U(N(C'))} f(a_i, x_j)$$
$$\forall x_j \in X(C) : f'(x_j, t) = \sum_{(a_i,x_j)\in U(N(C'))} f(a_i, x_j)$$
$$f'(t, s) = \sum_{(s,a_i)\in U(N(C'))} f(s, a_i).$$

Let $\nu = \sum_{a_i \in D'(C) s.t. f'(s,a_i) < l(s,a_i)} (l(s, a_i) - f'(s, a_i))$. The computation of f'' (a feasible flow from s to t from f') needs at most ν augmenting steps. Furthermore $f''(t,s) \geq f'(t,s) - \nu$, so the computation of f^* (a maximum flow from s to t in $N'(C)$) requieres at most $f(t,s) - f'(t,s) + \nu$ steps. Since $f'(t,s) \geq f(t,s) - V$ and $\nu \leq V$, f^* can be computed in $O(V + f(t,s) - f(t,s) + V + V)$ steps. Hence the consistency of C can be checked in $O(V|U(N'(C))|)$.□

Therefore, if all constraints are combined, the total complexity of several stages of generalized arc consistency for one global cardinality constraint will never exceed $O(k^2 d^2)$.

Conclusion

In this paper we have presented a filtering algorithm for global cardinality constraints in CSPs. This algorithm can be viewed as an efficient way of implementing generalized arc-consistency for this kind of constraint. It exploits the pruning performance of the previous condition with a low complexity. In fact, for one gcc C, its space complexity is $O(|X(C)||D(C)|)$ and its time complexity is $O(|X(C)|^2|D(C)|)$. Moreover, we have proved a new Theorem in flow theory. This Theorem is general and should be used for global filtering of others constraints.

Acknowledgments

We thank Pascal van Hentenryck and Jean-François Puget for helpful discussions.

References

Berge, C. 1970. *Graphe et Hypergraphes.* Paris: Dunod.

Bessière, C.; Freuder, E.; and Régin, J.-C. 1995. Using inference to reduce arc consistency computation. In *Proceedings of IJCAI'95*, 592–598.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65(1):179–190.

Caseau, Y.; Guillo, P.-Y.; and Levenez, E. 1993. A deductive and object-oriented approach to a complex scheduling problem. In *Proceedings of DOOD'93.*

Dinic, E. 1970. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Dokl.* 11:1277–1280.

Ford, L., and Fulkerson, D. 1962. *Flows in Networks.* Princeton: Princeton University Press.

Laurière, J.-L. 1978. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence* 10:29–127.

Lawler, E. 1976. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston.

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99–118.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28:225–233.

Mohr, R., and Masini, G. 1988a. Good old discrete relaxation. In *Proceedings of ECAI-88*, 651–656.

Mohr, R., and Masini, G. 1988b. Running efficiently arc consistency. *Syntactic and Structural Pattern Recognition* F45:217–231.

Nuijten, W. 1994. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach.* Ph.D. Dissertation, Eindhoven University of Technology.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI-94*, 362–367.

Tarjan, R. 1972. Depth-first search and linear graph algorithms. *SIAM Journal of Computing* 1:146–160.

Tarjan, R. 1983. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics. 97–113.

Van Hentenryck, P., and Deville, Y. 1991. The cardinality operator: A new logical connective for constraint logic programming. In *Proceedings of ICLP-91*, 745–759.

Van Hentenryck, P.; Deville, Y.; and Teng, C. 1992. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence* 57:291–321.