Specific Filtering Algorithms for Over-Constrained Problems

Thierry Petit[1/2], Jean-Charles Régin[1] and Christian Bessière[2]
{regin, tpetit}@ilog.fr, {bessiere, tpetit}@lirmm.fr

[1]ILOG, 1681, route des Dolines, 06560 Valbonne, FRANCE
[2]LIRMM (UMR 5506 CNRS), 161, rue Ada, 34392 Montpellier Cedex 5, FRANCE

Abstract. In recent years, many constraint-specific filtering algorithms have been introduced. Such algorithms use the semantics of the constraint to perform filtering more efficiently than a generic algorithm. The usefulness of such methods has been widely proven for solving constraint satisfaction problems. In this paper, we extend this concept to over-constrained problems by associating specific filtering algorithms with constraints that may be violated. We present a paradigm that places no restrictions on the constraint filtering algorithms used. We illustrate our method with a complete study of the All-different constraint.

1 Introduction

A problem is over-constrained when no assignment of values to variables satisfies all constraints. In this situation, the goal is to find a compromise. Violations are allowed in solutions, providing that such solutions retain a practical interest. Therefore, it is mandatory to respect some rules and acceptance criteria defined by the user. A *cost* is generally associated with each constraint in order to quantify its violation [3]. Then, costs can be bounded from above. For instance, consider a cost associated with the violation of a temporal constraint imposing that a person should stop working before a given date: this cost should be proportional to the additional amount of working time she performs, and this amount should not be excessive.

A global objective related to the whole set of costs is usually defined. For instance, the goal can be to minimize the total sum of costs. In some applications it is necessary to express more complex rules on violations, which involve several costs independently from the objective. Such rules can be defined through meta-constraints [9]. In this paper, we are interested in solving such problems.

Existing algorithms dedicated to over-constrained problems [6, 15, 8, 14, 13] are generic. However, the use of constraint-specific filtering algorithms is generally required to solve real-world applications (e.g., [10, 2, 12]), as their efficiency can be much higher.

Regarding over-constrained problems, existing constraint-specific filtering algorithms can be used only in the particular case where the constraint must be satisfied. Indeed, they removes values which are not consistent with the constraint: the deletion condition is linked to the fact that it is mandatory to satisfy the constraint. This condition is not applicable when the violation is allowed.

However, domains can be reduced from the objective and from the costs associated with violations of constraints. The main idea of this paper is to perform this kind of filtering specifically, that is, to take advantage from the semantics of a constraint and from the semantics of its violation to efficiently reduce the domains of the variables it constrains.

The deletion condition will be linked to the necessity of having an acceptable cost, instead of being related to the satisfaction requirement. For instance, consider the constraint $C : x \leq y$, such that $D(x) = [0, 3]$ and $D(y) = [0, 1]$. Assume that, when C is violated, the cost is defined as the difference between x and y. If the cost of C has to be less than or equal to 1, then the value 3 can be removed from $D(x)$. Note that the domain of allowed values of the cost of a given constraint can be reduced during the search, by propagation of the objective and of the other costs.

We present a model such that the costs are expressed by variables directly integrated into the problem. Roughly, the principle is to turn an over-constrained problem into a classical optimization problem. Then, as any other variable, a cost can be constrained. In this way, the violation of a constraint C can be controled through a constraint $\bar{C}$ linking C and the cost. In terms of efficiency, the main interest of this approach is the possibility to associate a specific filtering algorithm with $\bar{C}$, which exploits the semantics.

We discuss how a violation can be quantified. For a given constraint, several possible definitions of the cost can be considered. They correspond to different filtering algorithms. We propose two general definitions for non-binary constraints.

We provide a complete study of the All-different Constraint [10]: we present two algorithms based on flow theory, related to the two definitions of the cost.

2 Background

CSP A *constraint network* $\mathcal{N}$ is defined as a set of n *variables* $X = \{x_1, \ldots, x_n\}$, a set of *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. A *constraint* C on the ordered set of variables $var(C) = (x_{i_1}, \ldots, x_{i_r})$ (also denoted by $C(x_{i_1}, \ldots, x_{i_r})$) is a subset $rel(C)$ of the Cartesian product $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ that specifies the *allowed* combinations of values for the variables $x_{i_1}, \ldots, x_{i_r}$. $D(var(C)) = \cup_{x \in var(C)} D(x)$. An element of $D(x_{i_1}) \times \cdots \times D(x_{i_r})$ is called a *tuple on* $var(C)$. $|var(C)|$ is the *arity* of C. C is *binary* iff $|var(C)| = 2$. A value a for a variable x is denoted by (x, a). A tuple τ on $var(C)$ is *valid* if $\forall (x, a) \in \tau, a \in D(x)$. C is *consistent* iff there exists a tuple τ of $rel(C)$ which is valid. A value $a \in D(x)$ is *consistent with* C iff $x \notin var(C)$ or there exists a valid tuple τ of $rel(C)$ in which a is the value assigned to x. Given $Y \subseteq X$, an *instantiation* I of Y is an assignment of values to variables Y such that $\forall x \in Y$, the value a assigned to x belongs to $D(x)$. Given $Y \subseteq X$ and $C \in \mathcal{C}$ such that

$var(C) \subseteq Y$, an instantiation I of Y *satisfies* a constraint C iff the projection of I on $var(C)$ belongs to $rel(C)$. If I does not satisfy C, then I *violates* C. The *Constraint Satisfaction Problem* (CSP) consists of finding an instantiation I of X such that $\forall C \in \mathcal{C}$, I satisfies C.

Over-Constrained Problem When a CSP has no solution, we say that the problem is *over-constrained.* $\mathcal{C}_h \subseteq \mathcal{C}_s$ is the set of *hard* constraints, that is, the constraints that must necessarily be satisfied. $\mathcal{C}_s = \mathcal{C} \setminus \mathcal{C}_h$ is the set of *soft* constraints. Let I be an instantiation of X. If I is a solution of an over-constrained problem then $\forall C \in \mathcal{C}_h$, I satisfies C. The Maximal Constraint Satisfaction Problem (Max-CSP) is the problem where all the constraints are soft; the goal is to minimize the number of violated constraints.

All-different Constraint The All-different constraint, called *AllDiff*, is the constraint C stating that the variables in $var(C) = \{x_{i_1}, \ldots, x_{i_k}\}$ must take values different from each other: it is defined by the set of tuples $rel(C) = \{(d_1, \ldots, d_k) \in D(x_{i_1}), \ldots, D(x_{i_k})$ s.t. $\forall u, v$: $d_u \neq d_v\}$.

Graphs A *graph* $G = (X, E)$ consists of a set X of vertices and a set of edges E, where every edge is a pair of distinct vertices. $G' = (X', E')$ is a *subgraph* of G iff $X' \subseteq X$ and $E' \subseteq E$. A *clique* of G is a subgraph $G' = (X', E')$ of G such that $\forall u \in X', \forall v \in X'$, $(u, v) \in E'$. A *directed graph* $G = (X, U)$ consists of a vertex set X and a set of arcs U, where every arc (u, v) is a directed pair of distinct vertices. An arc (u, v) *leaves* u and *enters* v. $\Gamma^-(v)$ is the set of edges entering a vertex v. $\Gamma^+(v)$ is the set of edges leaving v. $\Gamma(v) = \Gamma^- \cup \Gamma^+$.

– **Matchings:** Given a graph $G = (X, E)$, a *matching* $M \subseteq E$ is a subset of edges such that no two edges have a vertex in common. The *cardinality* of a matching is the number of edges it contains. A matching of maximum cardinality is called a *maximum matching.* Given a matching M, every edge of E which does not belong to M is *free.* Every vertex v in X which is not an endpoint of an edge of M is *free*; for convenience, if v is not free then we say that $v \in M$.

– **Flows:** Flow theory was originally introduced by Ford and Flukerson [5]. Let $G = (X, U)$ be a directed graph such that each arc (u, v) is associated with two positive integers $lb(u, v)$ and $ub(u, v)$. $ub(u, v)$ is called the *upper bound capacity* of (u, v) and $lb(u, v)$ the *lower bound capacity.* A *flow* in G is a function f satisfying the following two conditions:

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

The *feasible flow problem* is the problem of the existence of a flow in G which satisfies the *capacity constraint*, that is: $\forall (u, v) \in U : lb(u, v) \leq f(u, v) \leq ub(u, v)$.

3 Propagating Costs

3.1 Preliminary Example

Let C be the constraint $x \leq y$. In order to quantify its violation, a *cost* is associated with C. The semantics are the following:

– if C is satisfied then $cost = 0$.

– if C is violated then $cost > 0$ and its value is proportional to the gap between x and y, that is, $cost = x - y$.

Assume that $D(x) = [90001, 100000]$ and $D(y) = [0, 200000]$, and that the cost is contrained to be less than 5. We deduce[1] that $x - y \leq 5$, and, by propagation, $D(y) = [89996, 200000]$.

Such a deduction is made directly by propagating *bounds* of the variables x, y and $cost$. Inequality constraints admit such propagation on bounds without consideration of the domain values which lie between. Such propagation, which depends on the semantics of the inequality constraint, is fundamentally more efficient than the consideration for deletion of each domain value in turn. If we ignore the semantics in the example, the only way to filter a value is to study the cost of each tuple in which this value is involved. Performing the reduction of $D(y)$ in the example above is costly: at least $|D(x)| * 89996 = 899960000$ checks are necessary. This demonstrates the gain to achieved by directly integrating constraints on costs into the problem and employing constraint-specific filtering algorithms.

Following this idea, our goal is to allow the *same* modelling flexibility with respect to violation costs as with any other constrained variable. The most natural way to achieve this is to include these violation costs as variables in a new constraint network[2].

3.2 New Model

The model we present in this section is well suited to use specific algorithms, for any kind of constraints. For sake of clarity, we consider that the values of the cost associated with a constraint C are positive integers. 0 expresses the fact that C is satisfied, and strict positive values are proportional to the importance of a violation. This assumption is not a strong restriction; it just implies that values of cost belong to a totally ordered set.

We suggest solving a new optimization problem derived from the initial problem. It involves the same set of hard constraints $\mathcal{C}_h$, but a set of disjunctive constraints replaces $\mathcal{C}_s$. This set of disjunctive constraints is denoted by $\mathcal{C}_{disj}$ and a one-to-one correspondance is defined between $\mathcal{C}_s$ and $\mathcal{C}_{disj}$. Each disjunction involves a new variable $cost \in X_{costs}$, which is used to express the cost

[1] Either C is satisfied: $x - y \leq 0$, or C is violated: $x - y = cost$ and $cost \leq 5$, which implies $x - y \leq 5$.

[2] In existing frameworks [3] the variable set is not extended. The costs are not included into the problem through variables.

of $C \in \mathcal{C}_s$. A one-to-one correspondance is also defined between $\mathcal{C}_s$ and X_{costs}. Given $C \in \mathcal{C}_s$, the disjunction is the following:

$$[C \wedge [cost = 0]] \vee [\bar{C} \wedge [cost > 0]]$$

$\bar{C}$ is the constraint including the variable $cost$ that expresses the violation[3]. A specific filtering algorithm can be associated with it. Regarding the preliminary example, the constraints C and $\bar{C}$ are respectively $x \leq y$ and $cost = x - y$:

$$[[x \leq y] \wedge [cost = 0]] \vee [[cost = x - y] \wedge [cost > 0]]$$

The new defined problem is not over-constrained: it consists of satisfying the constraints $\mathcal{C}_h \cup \mathcal{C}_{disj}$, while optimizing an objective defined over all the variables X_{costs} (we deal with a classical optimization problem); constraints on a variable $cost$ can be propagated.

Such a model can be used for encoding directly over-constrained problems with exisiting solvers [13]. Moreover, additional constraints on cost variables can be defined in order to select solutions which are acceptable for the user [9].

3.3 Consistency and Domain Reduction

In this section we formalize our motivations. Assume that the objective is to minimize the sum obj of m costs $\{cost_1, ..., cost_m\}$, corresponding to constraints $\mathcal{C}_s = \{C_1, ..., C_m\}$. Let C_i be a constraint in $\mathcal{C}_s$, and $cost_i$ be its cost variable. We are interested in computing a lower bound lb_i of the minimal value of $cost_i$ consistent with $\bar{C}_i$. This lower bound can be used to check the consistency of $\bar{C}_i$. If $lb_i > max(D(cost_i))$, or if $lb_i + \sum_{k \in \{1,...,m\}} min(D(cost_k)) - min(D(cost_i)) > max(obj)$, then $\bar{C}_i$ is not consistent.

Moreover, the same kind of lower bound can be computed in order to reduce domains of variables constrained by C_i. Let x be a variable of C_i, and a be a value of $D(x)$. Let $lb_{i(x,a)}$ be the minimal value of $cost_i$ consistent with $\bar{C}_i \wedge (x = a)$. If $lb_{i(x,a)} > max(D(cost_i))$, or if $lb_{i(x,a)} + \sum_{k \in \{1,...,m\}} min(D(cost_k)) - min(D(cost_i)) > max(obj)$, then a can be removed from $D(x)$.

Note that, when studying the consistency with respect to $max(obj)$, some algorithms have been proposed to compute a global lower bound greater than the sum of minima of costs in the left part of the equations above. Even if it is not the main topic of this paper, we point out that our model does not impose any restriction about using such algorithms. For instance, the best existing algorithm for binary Max-CSPs (e.g., PFC-MRDAC [8]) has been extended to the non-binary case and adapted to this model [13].

Similar reasoning can be performed with objectives different from a minimization of the sum of costs, providing that there is a way of infering how a change to local costs affects the objective. For some problems, the costs should be normalized in order to make them comparable and to integrate them into the global objective. They can also be weighted in order to favor the satisfaction of some constraints rather than others.

[3] It is possible to define any constraint $\bar{C}$, and even a constraint that does not constrain $cost$.

3.4 General Definitions of Cost

When natural semantics can be associated with the violation of a constraint, we use them (for instance, the constraint of the preliminary example $C : x \leq y$). However, it is not necessarily the case. When there are no natural semantics associated with the violation of a constraint, different definitions of the cost can be considered, depending on the problem.

The Alldiff example Let C be an Alldiff defined on variables $var(C) = \{x_1, x_2, x_3, x_4\}$, such that $\forall i \in [1,4], D(x_i) = \{a, b, c, d\}$. If we ignore the symmetric cases by considering that no value (resp. no variable) has more importance than another, we have the following possible assignments: (1) a, b, c, d, (2) a, a, c, d, (3) a, a, c, c, (4) a, a, a, c, (5) a, a, a, a.

Intuitively, it is straightforward that the violation of case (5) is more serious than the one of case (2). This fact has to be expressed through the cost.

We propose two general definitions of the cost associated with the violation of a non-binary constraint.

Definition 1 : Variable Based Violation Cost *Let C be a constraint. The cost of its violation can be defined as the number of assigned values that should change in order to make C satisfied.*

The advantage of this definition is that it can be applied to any (non-binary) constraint. However, depending on the application, it can be inconvenient. In the Alldiff example above, a possible problem is that assignments (3) and (4) have the same cost according to definition 1. For an Alldiff involving more than four variables, a lot of different assignments have the same cost.

Therefore, we propose another definition of the cost, which is well suited to constraints that are representable through a primal graph [4, 7]:

Definition 2 *The* **primal graph** $Primal(C) = (var(C), E_p)$ *of a constraint C is the graph such that each edge represents a binary constraint, and the set of solutions of the CSP defined by $\mathcal{N} = (var(C), D(var(C)), E_p)$ is the set of allowed tuples of C.*

For an Alldiff C, $Primal(C)$ is a complete graph where each edge represents a binary inequality.

Definition 3 : Primal Graph Based Violation Cost *Let C be a constraint representable by a primal graph. The cost of its violation can be defined as the number of binary constraints violated in the CSP defined by $Primal(C)$.*

In the Alldiff case, the user may aim at controlling the number of binary inequalities implicitely violated. The advantage of this definition is that the granularity of the quantification is more accurate (in the example, costs of assignments (3) and (4) are different). Unfortunately, some constraints are not representable through a primal graph (for instance the constraint $C : p = q + r$).

4 Algorithms for the AllDiff

Let $C \in \mathcal{C}_s$ be an Alldiff, and *cost* its cost variable. In the following, we use two bipartite graphs. A bipartite graph is a graph $G = ((X_1, X_2), E)$ such that the vertex set $X_1 \cup X_2$ is partitioned in two disjoint sets X_1 and X_2, and such that there is no edge between any two vertices of the same set.

Definition 4 : Value Graph *Let C be an AllDiff.*
$VG(C) = (var(C), D(var(C)), E)$ is the graph such that $(x, a) \in E$ iff $a \in D(x)$.

Note that an instantiation of $var(C)$ corresponds to a graph where for each variable $x \in var(C)$ there is exactly one edge leaving x. If such a graph is a matching, C is satisfied.

Definition 5 : Value Graph with x=a *Let C be an AllDiff. Let $x \in var(C)$ and $a \in D(x)$. $VG_{(x,a)}(C) = (var(C), D(var(C)), F \cup \{(x, a)\})$ is the graph such that $(y, b) \in F$ iff $y \neq x$ and $b \in D(y)$.*

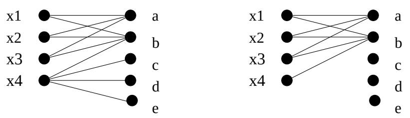

Figure 1: $D(x_1) = \{a, b\}, D(x_2) = \{a, b\}, D(x_3) = \{a, b\}, D(x_4) = \{b, c, d, e\}$.
The left graph is $VG(C)$, the right graph is $VG_{(x4,b)}(C)$.

Notation 1 *$\mu(G)$ is the cardinality of a maximum matching of a graph G.*

4.1 Variable Based Violation Cost

$\bar{C}$ is a constraint of arity $|var(C)| + 1$, where the additional variable is *cost*. According to definition 1, the *cost* is equal to the number of variables that should change their value in order to satisfy the property of having no value of $D(var(C))$ assigned to more than one variable in $var(C)$.

Firstly, we aim at computing a lower bound of *cost* in order to check the consistency of $\bar{C}$:

Property 1 *Let C be an AllDiff, $cost \in X_{costs}$ be the cost associated with C. $lb = |var(C)| - \mu(VG(C))$ is a lower bound of cost. (if $lb > max(D(cost))$ then $\bar{C}$ is not consistent).*

Proof: The existence of an instantiation of $var(C)$ such that C can become satisfied by changing the value of p variables implies the existence of a matching of size $|var(C)| - p$. By definition of $\mu(VG(C))$, $|var(C)| - p \leq \mu(VG(C))$, and $p \geq |var(C)| - \mu(VG(C))$.

The maximum size of a matching in a bipartite graph can be polynomially computed [1] in $O(\sqrt{|var(C)|} * K)$, where $K = \sum |D(x)|, x \in var(C)$.

Secondly, the same principle can be applied to reduce domains of variables in $var(C)$. Let $x \in var(C)$ and $a \in D(x)$. From Property 1 we have:

Theorem 1 *Let $x \in var(C)$ and $a \in D(x)$. Let $lb_{(x,a)} = |var(C)| - \mu(VG_{(x,a)}(C))$. If $lb_{(x,a)} > max(D(cost))$ then a can be removed from $D(x)$.*

The complexity of this filtering algorithm can be improved:

Property 2 *Let $lb = |var(C)| - \mu(VG(C))$. If $lb < max(D(cost))$ then all the values of domains of variables in $var(C)$ are consistent with $\bar{C}$.*

Proof: See Régin's Ph.D. Dissertation [11]

Therefore, the only case to study is $lb = max(D(cost))$:

Property 3 *Given $\bar{C}$, values of the domains of $var(C)$ which are not arc-consistent can be removed in $O(K)$ where $K = \sum |D(x)|, x \in var(C)$.*

Proof: e.g. Régin [10, 11].

4.2 Primal Graph Based Violation Cost

$\bar{C}$ is a constraint of arity $|var(C)| + 1$, where the additional variable is *cost*. According to definition 3, the *cost* is equal to the number of binary inequalities violated in the CSP corresponding to the primal graph of C.

Necessary Condition of Consistency of $\bar{C}$

Notation 2 *Let C be an Alldiff, and I an instantiation of $var(C)$.*
- *$\#(a, I)$ is the number of time the value a is assigned to variables in I.*
- *$max\#(I) = max(\#(a, I), a \in D(var(C)))$ is the maximum value of $\#(a, I)$ among all the values.*

Vertices of the primal graph $Primal(C)$ can be colored w.r.t. I, and *cost* is the sum of number of edges of cliques of each color.

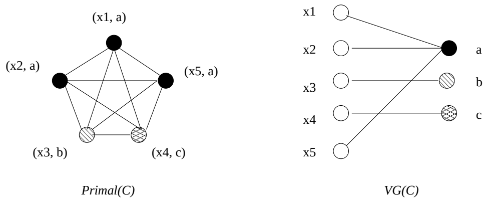

Figure 2: *cliques* $= \{\{x_1, x_2, x_5\}, \{x_3\}, \{x_4\}\}$. *cost* $= (3 * 2)/2 + 0 + 0 = 3$.

The problem we have to solve is: "Among all the possible instantiations of $var(C)$, which is the one which has the lowest cost?"

Consider I any instantiation of $var(C)$. If the number of different values involved in I is equal to $|var(C)| - 1$, then we know that we will have one value assigned to two variables (i.e., $max\#(I) = 2$). Thus, we know that $cost = 1$; if the number of different values involved in I is equal to $|var(C)| - 2$, then we will have two possibilities: either there are two values a and b such that $\#(a, I) = \#(b, I) = 2$ (corresponding to a cost equal to 2), or one value c with $\#(c, I) = 3$ (corresponding to a cost of 3). This means that the cost depends on the number of different values involved in I and also on the maximum number of times a value is taken.

In the following table, all cases up to symmetry are presented for an AllDiff C defined on $var(C) = \{x_1, ..., x_8\}$. "$p$ of q" means p cliques of q variables in the primal graph, and *cost* represents the cost of violation. For instance, the cost value for 5 values and $max\#(I) = 4$ is equal to 6 which is greater than 4, the cost value for 4 values and $max\#(I) = 2$, but which is lower than 7 the cost value for 4 values and $max\#(I) = 4$.

Number of values	$max\#(I)$	Cliques in $Primal(C)$	Cost	Example
1	8	1 of 8	28	(a,a,a,a,a,a,a,a)
2	4	2 of 4	12	(a,a,a,a,b,b,b,b)
	5	1 of 5 + 1 of 3	13	(a,a,a,a,a,b,b,b)
	6	1 of 6 + 1 of 2	16	(a,a,a,a,a,a,b,b)
	7	1 of 7 + 1 of 1	21	(a,a,a,a,a,a,a,b)
3	3	2 of 3 + 1 of 2	7	(a,a,a,b,b,b,c,c)
	4	1 of 4 + 2 of 2	8	(a,a,a,a,b,b,c,c)
	4	1 of 4 + 1 of 3 + 1 of 1	9	(a,a,a,a,b,b,b,c)
	5	1 of 5 + 1 of 2 + 1 of 1	11	(a,a,a,a,a,b,b,c)
	6	1 of 6 + 2 of 1	15	(a,a,a,a,a,a,b,c)
4	2	4 of 2	4	(a,a,b,b,c,c,d,d)
	3	1 of 3 + 2 of 2 + 1 of 1	5	(a,a,a,b,b,c,c,d)
	3	2 of 3 + 2 of 1	6	(a,a,a,b,b,b,c,d)
	4	1 of 4 + 1 of 2 + 2 of 1	7	(a,a,a,a,b,b,c,d)
	5	1 of 5 + 3 of 1	10	(a,a,a,a,a,b,c,d)
5	2	3 of 2 + 2 of 1	3	(a,a,b,b,c,c,d,e)
	3	1 of 3 + 1 of 2 + 3 of 1	4	(a,a,a,b,b,c,d,e)
	4	1 of 4 + 4 of 1	6	(a,a,a,a,b,c,d,e)
6	2	2 of 2 + 4 of 1	2	(a,a,b,b,c,d,e,f)
	3	1 of 3 + 5 of 1	3	(a,a,a,b,c,d,e,f)
7	2	1 of 2 + 6 of 1	1	(a,a,b,c,d,e,f,g)
8	1	8 of 1	0	(a,b,c,d,e,f,g,h)

We aim at proving that:
- There is an instantiation I with a minimum cost which involves the highest number of values and which has the minimum value for $\#max(I)$.
- We can identify I' an instantiation with a minimum value of $max\#(I)$.

Notation 3 *Let I be a instantiation of $var(C)$, V be the set of values involved in I, and $VGI(C) = (var(C) \cup V, E)$ the graph of the instantiation I defined by $E = \{(x, a)$ s.t. a is assigned to $x\}$.*
− matching(I) denotes a matching of $VGI(C)$ such that each value of I is an

extremity of an edge of this matching.
– $cost(I)$ is the sum of edges of cliques in the graph $Primal(C)$ colored w.r.t I.

Note that if a value is involved in I then it belongs to $matching(I)$. By construction this matching is necessarily of matching of maximum size in $VGI(C)$ because it covers V. The following theorem greatly helps us to compute a lower bound of *cost*:

Theorem 2 *For each instantiation I of $var(C)$ there exists an intantiation I' of $var(C)$ such that $max\#(I') \leq max\#(I)$, $|matching(I')| = \mu(VG(C))$ and $cost(I') \leq cost(I)$.*

This theorem is proved by successively applying the following lemma:

Lemma 1 *For each instantiation I of $var(C)$ such that $|matching(I)| = s-1 < \mu(VG(C))$ there exists an instantiation I' of $var(C)$ such that $|matching(I')| = s$, $max\#(I') \leq max\#(I)$ and $cost(I') \leq cost(I)$.*

Proof: Let $M = matching(I)$. Then $|M| = s-1 < \mu(VG(C))$, which means that M is not maximum; thus, from the matching theory, there is an alternating path P (i.e. a path those arcs are alternately in M and not in M) from $x \notin M$ to $v \notin M$. Consider M' be the matching obtained by augmenting M along P. Let I' be the instantiation of $var(C)$ defined as follows: a is assigned to y iff $(y, a) \in M'$ or ($y \notin M'$ and a is assigned to y in I). Value v is taken by one variable in I' whereas it is not involved in I. Let z be the variable which immediately preceeds v in P, and w the value assigned to z in I'. Then, $\#(w, I') = \#(w, I) - 1$. And $\forall u \in (D(var(C)) \setminus \{v, w\})$ $\#(u, I') = \#(u, I)$. Therefore $max\#(I') \leq max\#(I)$. Moreover, the constraints violated by the variables which take their values in $D(var(C)) \setminus \{v, w\}$ are the same for I and I'; v is taken by only one variable in I' thus no violation is induced; there are less variables in I' which take the value w than in I. Hence, $cost(I') \leq cost(I)$.

The algorithm we propose for computing the minimum possible value of $max\#(I)$ for an instantiation I of $var(C)$, denoted by $minMax\#(C)$, is based on the search of feasible flows on the following graph:

Definition 6 : Value Assignment Graph
Let C be an AllDiff and $VG(C)$ its value graph. The **Value Assignment Graph** *of C is the directed graph $VAG(C) = (var(C) \cup D(var(C)) \cup \{s, t\}, E')$ such that E' is defined by:*
– s is a vertex such that $\forall x \in var(C)$, $(s, x) \in E'$.
– t is a vertex such that $\forall v \in D(var(C))$, $(v, t) \in E'$.
– $(t, s) \in E'$.
– $\forall (x, v) \in E$, $(x, v) \in E'$.

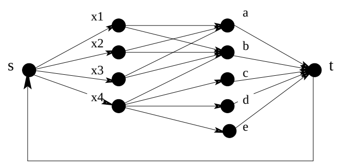

Figure 3: *Example of a Value Assignment Graph.*
$D(x_1) = \{a, b\}, D(x_2) = \{a, b\}, D(x_3) = \{ab\}, D(x_4) = \{b, c, d, e\}.$

The lower-bounds and upper-bound capacities in E' are:
– $\forall(s, x)$, $x \in var(C)$, $lb(s, x) = ub(s, x) = 1$.
– $\forall(x, v)$, $x \in var(C)$, $v \in D(var(C))$, $lb(x, v) = 0$ and $ub(x, v) = 1$.
– $\forall(v, t)$, $v \in D(var(C))$, $lb(v, t) = 0$ and $ub(v, t) = max\#$ where $max\#$ is an integer such that $1 \le max\# \le |var(C)|$.
– $lb(t, s) = ub(t, s) = |var(C)|$.

The link between $VAG(C)$ and an instantiation is a flow satisfying the following condition: if a value v is assigned to a variable x then $f(x, v) = 1$, otherwise $f(x, v) = 0$. The idea is to search for a first feasible flow problem on $VAG(C)$ with $max\#$ equal to 1. If there is none, a new feasible problem is run with $max\#$ equal to 2, and so on, until a feasible flow is found in $VAG(C)$.

Given $minMax(C)$, $|var(C)|$ and a matching size s, the computeMinCost function returns the corresponding minimum *cost*. This function can be run before search in order to compute all the possible costs.

isTooLow($size$, $\#vars$, $\#vals$)
    return $size * \#vals < \#vars$;

isTooHigh($size$, $\#vars$, $\#vals$)
    return $\#vars - size < \#vals - 1$;

existsSize($size$, $\#vars$, $\#vals$)
    return ¬ isTooLow($size$, $\#vars$, $\#vals$)
            ∧ ¬ isTooHigh($size$, $\#vars$, $\#vals$);

computeMinCost($minMax(C)$, $|var(C)|$, s)
    if ¬ existsSize($minMax(C)$, $|var(C)|$, s) **then return -1;**
    if $minMax(C) \neq 1$ **then** $cost \leftarrow minMax(C) * (minMax(C) - 1)/2$;
    $\#vars \leftarrow |var(C)| - minMax(C)$;
    $\#takenVals \leftarrow 1$;
    while $\#vars > 0$ **do**
        while ¬ isTooLow ($minMax(C) - 1$, $\#vars$, $s - \#takenVals$) **do**
            $minMax(C) - -$;
        if $minMax(C) \neq 1$ **then** $cost \leftarrow cost + minMax(C) * (minMax(C) - 1)/2$;
        $\#takenVals + +$;
        $\#vars \leftarrow \#vars - minMax(C)$;
    return $cost$;

Now, we can compute the lower bound. It is equal to the smallest possible cost according to the values of $minMax\#(C)$ and $\mu(VG(C)))$.

It is not optimal because for a given cardinality of a maximum matching $\mu(VG(C))$ and a given value of $minMax\#(C)$, more than one value of cost may be possible.

For instance, if $|var(C)| = 8$, $\mu(VG(C)) = 4$ and $minMax\# = 3$ then the *cost* can be 5 or 6; but since many matchings of maximum size exist, improving more this lower bound would probably be too costly.

The following algorithm returns true if $\bar{C}$ is consistent according to the domain of the variable *cost*:

isConsistent($\bar{C}$, *cost*)
  $\mu(VG(C)) \leftarrow$ **size of a maximum matching of** $VG(C)$;
  $max\# \leftarrow 2$;
  while $max\# \leq |var(C)|$
    if there exists a feasible flow on $(VAG(C), max\#)$ **then**
      return computeMinCost($max\#$, $var(C)$, $\mu(VG(C))) \leq max(D(cost))$;
    else $max\# \leftarrow max\# + 1$;

The complexity of this algorithm is $O(|var(C)| * \sqrt{|var(C)|} * K)$, where $K = \sum |D(x)|, x \in var(C)$.

Based on the same principle, we propose the follwing filtering algorithm:

Let $x \in var(C)$ and $a \in D(x)$. Let $VAG_{(x,a)}(C) = (var(C) \cup D(var(C)) \cup \{s,t\}, E'')$ be the subgraph of $VAG(C) = (var(C) \cup D(var(C)) \cup \{s,t\}, E')$ such that $E'' = E' \setminus \Gamma^{+}(x) \cup \{(x,a)\}$. In the algorithm, we use the graphs $VG_{(x,a)}(C)$ and $VAG_{(x,a)}(C)$:

Filter($\bar{C}$, *cost*)
  for all $x \in var(C)$
    for all $a \in D(x)$
      $\mu(VG_{(x,a)}(C)) \leftarrow$ **size of a maximum matching of** $VG_{(x,a)}(C)$;
      $max\# \leftarrow 2$;
      while $max\# \leq |var(C)|$
        if there exists a feasible flow on $(VAG_{(x,a)}(C), max\#)$ **then**
          if computeMinCost($max\#$, $var(C)$, $\mu(VG_{(x,a)}(C)))$
          $> max(D(cost))$ **then**
            $D(x) \leftarrow D(x) \setminus \{a\}$;
          $max\# = |var(C)| + 1$;
        else $max\# \leftarrow max\# + 1$;

The complexity of this algorithm is $O(|var(C)|^2 * \sqrt{|var(C)|} * K * d)$, where $K = \sum |D(x)|, x \in var(C)$ and $d = max(|D(x)|), x \in var(C)$.

5 Conclusion

This paper introduces a new approach for solving over-constrained problems as classical constraint problems, such that filtering algorithms specific to violated constraints can be used. Two definitions of the cost associated with the violation of a constraint are proposed. A complete study of the All-different constraint is provided: two specific filtering algorithms based on flow theory are presented, related to the two definitions of the cost.

6 Acknowledgements

We thank Paul Shaw for the helpful comments he provided. This work also profited from discussions with Olivier Lhomme and Jean-Francois Puget. It was partially supported by the IST Programme of the European Commission through the ECSPLAIN project (project IST-1999-11969).

References

1. R. Ahuja, T. Magnanti, and J. Orlin. Networks flows: theory, algorithms, and applications. *Prentice Hall, Inc.*, 1993.
2. N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
3. S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs: Frameworks, properties, and comparison. *Constraints*, 4:199–240, 1999.
4. R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.
5. L. Ford and D. Flukerson. Flows in networks. *Princeton University Press*, 1962.
6. E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.
7. I. Gent, K. Stergiou, and T. Walsh. Decomposable constraints. *Artificial Intelligence*, 123:133–156, 2000.
8. J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for MAX-CSP. *Artificial Intelligence*, 107:149–163, 1999.
9. T. Petit, J.-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proceedings ICTAI*, pages 358–365, 2000.
10. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. *Proceedings AAAI*, pages 362–367, 1994.
11. J.-C. Régin. Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la Chimie Organique. *Ph.D. Dissertation, Université Montpellier II*, 1995.
12. J.-C. Régin. Generalized arc consistency for global cardinality constraint. *Proceedings AAAI*, pages 209–215, 1996.
13. J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An original constraint based approach for solving over constrained prolems. *Proceedings CP*, pages 543–548, 2000.
14. T. Schiex. Arc consistency for soft constraints. *Proceedings CP*, pages 411–424, 2000.

15. G. Verfaillie, M. Lemaître, and T. Schiex. Russian doll search for solving constraint optimisation problems. *Proceedings AAAI*, pages 181–187, 1996.