The Complexity of Global Constraints

Christian Bessiere
LIRMM-CNRS
Montpellier, France
email: bessiere@lirmm.fr

Emmanuel Hebrard and **Brahim Hnich** and **Toby Walsh***
Cork Constraint Computation Centre
University College Cork, Ireland.
email: {e.hebrard,b.hnich,tw}@4c.ucc.ie

Abstract

We study the computational complexity of reasoning with global constraints. We show that reasoning with such constraints is intractable in general. We then demonstrate how the same tools of computational complexity can be used in the design and analysis of specific global constraints. In particular, we illustrate how computational complexity can be used to determine when a lesser level of local consistency should be enforced, when decomposing constraints will lose pruning, and when combining constraints is tractable. We also show how the same tools can be used to study symmetry breaking, meta-constraints like the cardinality constraint, and learning nogoods.

Introduction

Global constraints are one of the factors central to the success of constraint programming. See, for example, (Régin 1994; 1996; Bessiere & Régin 1997; Régin & Rueher 2000; Beldiceanu & Contegean 1994; Frisch *et al.* 2002). Global constraints specify patterns that occur in many problems, and exploit efficient and effective constraint propagation algorithms for pruning the search space. For instance, we often have sets of variables which must take different values (e.g. activities in a scheduling problem requiring the same resource must all be assigned different times). Most constraint solvers therefore provide a global all-different constraint which is propagated efficiently and effectively (Knuth & Raghunathan 1992; Régin 1994).

What are the limits of reasoning with global constraints? In this paper, we show how the basic tools of computational complexity can be used to uncover many of the basic limits. We will show that reasoning with global constraints is intractable in general. We therefore need to focus on specific constraints like the all-different constraint which are tractable. We then show how these same tools of computational complexity can be used to analyse specific global constraints proposed in the past like the number of values constraint (Pachet & Roy 1999), as well as to help design new global constraints.

*All supported by Science Foundation Ireland and an ILOG grant.

Formal background

A *constraint satisfaction problem* (CSP) is a set of variables, each with a domain of values, and a set of constraints that specify allowed combinations of values for subsets of variables. We assume a constraint C is given intensionally by a function of the form $f_C : D_1 \times \ldots \times D_n \mapsto \{True, False\}$ where D_i are the domains of the variables in the scope of the constraint C. We cannot permit any sort of function. For example, suppose domains are integers of size m and $f_C(X_1, X_2, X_3, \ldots)$ is the function that halts iff $X_1 + X_2 * m + X_3 * m^2 + \ldots$ is the Gödel number of a halting Turing machine. Testing if an assignment satisfies the constraint C is then undecidable. We therefore insist that f_C is computable in polynomial time.

A solution to a CSP is an assignment of values to the variables satisfying the constraints. To find such solutions, we often use tree search algorithms that construct partial assignments and enforce a local consistency like generalized arc consistency to prune the search space. A constraint C is *generalized arc consistent* (GAC) iff, when a variable in the scope of C is assigned any value, there exists an assignment of the other variables in C such that C is satisfied (Mohr & Masini 1988). This satisfying assignment is called *support* for the value. An algorithm like GAC-Schema (Bessiere & Régin 1997) removes values from the initial domains of variables till we have the *maximal* generalized arc consistent subdomains. That is, the set of subdomains that are GAC and any larger set of subdomains are not GAC.

We have derived very similar results to those presented in this paper for other local consistencies. For example, many of these results map over to bounds consistency on finite domain, set or multiset variables. For reasons of space, however, we are only able to present the results here for generalized arc consistency.

Complexity of global constraints

Reasoning with global constraints is intractable in general. We consider two decision problems at the heart of reasoning with global constraints. The first is GACSUPPORT, the problem of deciding if a value for a variable has support on a constraint. In general, this is NP-complete to decide.

Theorem 1 GACSUPPORT *is NP-complete.*

Proof: Clearly it is in NP as a support is a polynomial witness which can be checked (by definition) in polynomial time. To show completeness, we transform the satisfiability of the Boolean formula φ into the problem of determining if a particular value has support. We simply construct the global constraint C involving the variables of φ plus an additional variable X, and defined by $f_C = (X \rightarrow \varphi)$. If $X = True$ has support then φ is satisfiable. $\heartsuit$

The second decision problem we consider is MAXGAC. Given a constraint and initial domains for its variables, this is the problem of deciding if a given set of subdomains is the maximal GAC set of subdomains. This is in the complexity class D^P. This contains problems which are the conjunction of a problem in NP and one in coNP (Papadimitriou & Yannakakis 1984). The class D^P is also known as the second level of the Boolean hierarchy, BH_2. A typical example of a D^P-complete decision problem is the EXACT TRAVELING SALESPERSON PROBLEM. Given a distance matrix and an integer B, EXACT TSP decides if the shortest tour is *equal* to B. By comparison, TSP decides if there is a tour of length B *or less*, which is NP-complete. We show that MAXGAC is D^P-complete.

Theorem 2 MAXGAC *is D^P-complete.*

Proof: A problem Q is in D^P if there exist a NP problem Q_1 and a coNP one Q_2 such that Q answers "yes" iff Q_1 and Q_2 answer "yes". If Q_1 is NP-complete and Q_2 is coNP-complete, then Q is D^P-complete. We use 3COL and UN3COL as Q_1 and Q_2. We suppose without loss of generality that Q_1 and Q_2 both involve the same set X of vertices. E_i is the set of edges in Q_i.

We introduce a variable for each vertex with domain $\{R_1, G_1, B_1, R_2, G_2, B_2\}$. We then define a global constraint as follows. For each pair (x_i, x_j) of vertices with an edge between in both Q_1 and Q_2, we permit pairs of values that are different but have the same subscript (i.e., the set $\{(R_1, G_1), (R_1, B_1), (G_1, R_1), (G_1, B_1), (B_1, R_1), (B_1, G_1), (R_2, G_2), (R_2, B_2), (G_2, R_2), (G_2, B_2), (B_2, R_2), (B_2, G_2)\}$). For each pair (x_i, x_j) of vertices with an edge between in Q_1 and not in Q_2, we permit pairs of values that are different for the subscript 1, and any combination for subscript 2 (i.e., the set $\{(R_1, G_1), (R_1, B_1), (G_1, R_1), (G_1, B_1), (B_1, R_1), (B_1, G_1)\} \cup \{R_2, G_2, B_2\}^2$). Similarly, for each pair (x_i, x_j) of vertices with an edge between in Q_2 and not in Q_1, we permit pairs of values that are different for the subscript 2, and any combination for subscript 1. Finally, for each pair (x_i, x_j) of vertices with no edge between in Q_1 or in Q_2, we permit any pairs of values with the same subscript (i.e., the set $\{R_1, G_1, B_1\}^2 \cup \{R_2, G_2, B_2\}^2$). By construction, R_i, G_i and B_i are GAC iff (X, E_i) is 3-colorable. Hence, $\{R_1, G_1, B_1\}$ is the maximal GAC subdomain for each variable iff (X, E_1) is 3-colorable, and (X, E_2) is not 3-colorable. $\heartsuit$

Corollary 1 *Enforcing GAC is NP-hard.*

Proof: Enforcing GAC directly answers both the GACSUPPORT and MAXGAC problems. $\heartsuit$

Reasoning with global constraints is thus not tractable in general. Global constraints which are used in practice are therefore usually part of that special subset for which constraint propagation is polynomial. For example, GAC on an n-ary `AllDifferent` constraint can be enforced in $O(n^{\frac{3}{2}}d)$ time (Régin 1994). In the rest of this paper, we show how we can further use the tools of computational complexity in the design and analysis of *specific* global constraints.

Local consistency

Computational complexity results can indicate what level of local consistency to enforce on a constraint. If achieving a given local consistency on a constraint is NP-hard, then enforcing a lower level of consistency is usually advisable. For example, the number of values constraint, $\texttt{NValue}(X_1, \ldots, X_n, N)$ (Pachet & Roy 1999; Beldiceanu 2001) ensures that N values are used by the n finite domain variables X_i. Note that N can itself be an integer variable. The `AllDifferent` constraint is a special case of the `NValue` constraint in which $N = n$. The `NValue` constraint is useful for reasoning about resources. For example, if the values are workers assigned to a particular shift, we may have a `NValue` constraint on the number of shifts that someone can work. Enforcing GAC on the `NValue` constraint is intractable in general. We therefore look to enforce a lower level of consistency.

Theorem 3 *Enforcing GAC on a* $\texttt{NValue}(X_1, \ldots, X_n, N)$ *constraint is NP-hard, and remains so even if N is ground and different to n.*

Proof: (Sketch) Reduction from 3SAT. Given a Boolean formula in k variables (labelled from 1 to k) and m clauses, we construct the $\texttt{NValue}(X_1, \ldots, X_{k+m}, N)$ constraint in which $X_i = \{i, -i\}$ for all $i \in [1, k]$, and each X_i for $i > k$ represents one of the m clauses. If the jth clause is $x \vee \neg y \vee z$ then $X_{k+j} = \{x, -y, z\}$. If $N = k$, the constructed `NValue` constraint has a solution iff the original 3SAT problem has a satisfying assignment. Hence testing a value for support is NP-complete, and enforcing GAC is NP-hard. $\heartsuit$

Decomposing constraints

Computational complexity results can tell us more than just what level of local consistency to enforce. It can also indicate properties that any possible decomposition of a constraint must possess. We say that a decomposition of a global constraint is *GAC-poly-time* if we can enforce GAC on the decomposition in polynomial time. A decomposition is GAC-poly-time when, for example, the number of decomposed parts is polynomial, and each decomposed part either is a specific constraint that has a polynomial GAC algorithm (like `AllDifferent`) or has a bounded arity. The following result tells us when such decomposition hinders constraint propagation.

Theorem 4 *If enforcing GAC on a constraint C is NP-hard, then there does not exist any GAC-poly-time decomposition of C that achieves GAC on C (assuming P $\neq$ NP).*

Proof: By definition, enforcing GAC on a GAC-poly-time decomposition is polynomial. Hence, if GAC on the decomposition was equivalent to GAC on the original constraint, then P would equal NP. $\heartsuit$

For example, (Sadler & Gervet 2001) introduce the `Atmost1` constraint. This ensures that n set variables of a fixed cardinality intersect in at most one value. To fit this within the theoretical framework presented in this paper, we consider the characteristic function representation for each set variable (i.e. a vector of 0/1 decision variables). Enforcing GAC on such a representation is equivalent to enforcing bounds consistency on the upper and lower bounds of the set variables (Walsh 2003a). The `Atmost1` constraint can be decomposed into pairwise intersection and cardinality constraints. That is, it can be decomposed into $|X_i \cap X_j| \leq 1$ for $i < j$ and $|X_i| = c$ for all i. On the characteristic function representation, this is $\sum_k X_{ik} \cdot X_{jk} \leq 1$ and $\sum_k X_{ik} = c$, which are both GAC-poly-time. Such decomposition hinders constraint propagation.

Theorem 5 *GAC on any GAC-poly-time decomposition of the* `Atmost1` *constraint is strictly weaker than GAC on the undecomposed constraint (assuming P $\neq$ NP).*

Proof: We show that enforcing GAC on an `Atmost1` constraint is NP-hard, and apply theorem 4.

To show that enforcing GAC on the `Atmost1` constraint is NP-hard, we consider the case when the cardinality $c = 2$. For $c > 2$, we can use a similar construction as in the $c = 2$ reduction but add $c - 2$ distinct values to each set. The proof uses a reduction from 3SAT. For each clause σ, we introduce a set variable, X_σ. Suppose $\sigma = x_i \vee \neg x_j \vee x_k$, then X_σ has the domain $\{m_\sigma\} \subseteq X_\sigma \subseteq \{m_\sigma, i_\sigma, \neg j_\sigma, k_\sigma\}$. If the intersection and cardinality constraint is satisfied, X_σ takes the value $\{m_\sigma, i_\sigma\}$, $\{m_\sigma, \neg j_\sigma\}$, or $\{m_\sigma, k_\sigma\}$. The first case corresponds to x_i being $True$ (which satisfies σ), the second to $\neg x_j$ being $True$, and the third to x_k being $True$.

We use additional (at most quadratic) set variables to ensure that contradictory assignments are not made to satisfy other clauses. Suppose we satisfy σ by assigning x_i to $True$. That is, $X_\sigma = \{m_\sigma, i_\sigma\}$. Consider any other clause, τ which contains $\neg x_i$. We construct two set variables, $Y_{\sigma\tau i}$ and $Z_{\sigma\tau i}$ with domains $\{m_\sigma\} \subseteq Y_{\sigma\tau i} \subseteq \{m_\sigma, i_\sigma, \neg i_\tau\}$ and $\{\neg i_\tau\} \subseteq Z_{\sigma\tau i} \subseteq \{m_\sigma, m_\tau, \neg i_\tau\}$. Since $X_\sigma = \{m_\sigma, i_\sigma\}$, then $Y_{\sigma\tau i} = \{m_\sigma, \neg i_\tau\}$ and $Z_{\sigma\tau i} = \{m_\tau, \neg i_\tau\}$. Hence, $X_\tau \neq \{m_\tau, \neg i_\tau\}$. That is, τ cannot be satisfied by $\neg x_i$ being assigned $true$. Some other literal in τ has to satisfy the clause.

The constructed set variables thus have a solution which satisfies the intersection and cardinality constraints iff the original 3SAT problem is satisfiable. Hence testing a value for support is NP-complete, and enforcing GAC is NP-hard. $\heartsuit$

A similar result can be given for the `Distinct` constraint introduced in (Sadler & Gervet 2001). This constraint ensures that n set variables of a fixed cardinality intersect in at least one value. Again, GAC-poly-time decomposition of such a constraint hinders constraint propagation (assuming P $\neq$ NP).

Combining constraints

Global constraints specify patterns that reoccur in many problems. There are, however, only a limited number of common constraints which repeatedly occur in problems. One strategy for developing new global constraints is to identify conjunctions of constraints that often occur together, and developing constraint propagation algorithm for their combination. For example, (Régin & Rueher 2000) propose a propagation algorithm for a constraint which combines together sum and difference constraints. As a second example, (Carlsson & Beldiceanu 2002) combine together a chain of lexicographic ordering constraints. As a third example, (Hnich, Kiziltan, & Walsh 2004) combine together a lexicographic ordering and two sum constraints.

We can use results from computational complexity to determine when we should combine together constraints. For example, scalar product constraints occur in many matrix models like the balanced incomplete block design, template design and social golfers problems (Walsh 2003b). Often such problems have scalar product constraints between all pairs of rows in a 0/1 matrix. We can enforce GAC on a scalar product constraint between two rows in linear time. Should we consider combining together all the row scalar product constraints into one large global constraint? Such a matrix `ScalarProduct` constraint would ensure that $\forall i < j \ \sum_k X_{ik} \cdot X_{jk} = m$. The following result shows that enforcing GAC on such a composition of constraints is intractable.

Theorem 6 *Enforcing GAC on a matrix* `ScalarProduct` *constraint is NP-hard, even when restricted to 0/1 variables.*

Proof: (Sketch) We consider the case when the scalar product $m = 1$. For $m > 1$, we use a reduction that adds $m - 1$ additional columns to the matrix, each column containing variables that must take the value 1.

We reduce 1IN3-3SAT on positive formulae (which is NP-complete (Garey & Johnson 1979)) to testing GAC on a matrix `ScalarProduct` constraint over 0/1 variables. The first row of the matrix represents the model which satisfies the 1IN3-3SAT problem. There is a column for each occurrence of a literal in a clause. This is assigned 1 iff the corresponding literal is *True*. There is also a column for the negation of each literal. This is assigned 1 iff the corresponding literal is *False*. There are also "dummy" columns to ensure each pair of rows has the required scalar product.

The remaining rows are divided into two types. First, there is a row for each clause. The columns corresponding to literals in the clause have 0/1 values. Columns corresponding to literals not in the clause only have the value 0. The scalar product constraint between a row representing a clause and the row representing the model ensures that only one of the literals in the clause is *True*. Second, there are rows for each occurrence of a positive literal to ensure that the row representing the model does not assign both a literal and its negation to *True*.

The 1IN3-3SAT problem has a model iff the constructed matrix has a solution. Hence testing a value for support is NP-complete, and enforcing GAC is NP-hard. $\heartsuit$

Special cases of the matrix `ScalarProduct` constraint are tractable. For instance, if the scalar product is zero and variables are 0/1 then the constraint is equivalent to the pairwise `Disjoint` constraint on set variables, which is tractable (Walsh 2003a).

Symmetry breaking

Like global constraints, symmetry is an important aspect of constraint programming. Computational complexity results can indicate how difficult it is to deal with particular symmetries. For example, symmetric rows and columns occur in many matrix models (Flener *et al.* 2002). Lexicographical ordering constraints can be posted between the rows to break all such row symmetry, and between all rows and columns to break some (but not all) such row and column symmetry (Frisch *et al.* 2002). Why don't we identify additional constraints to post that will break the remaining symmetry?

Some simple complexity analysis suggests that it will be intractable to do so. We define a symmetry breaking constraint, `MatrixSymmetry` which ensures that all permutations of the rows or columns give "smaller" matrices when the matrices of decision variables are ordered by conjoining their rows and comparing the resulting one-dimensional vectors lexicographically. This constraint eliminates all row and column symmetry from the matrix. Note that the `MatrixSymmetry` constraint implies that the rows and columns are lexicographically ordered (but not vice versa).

To demonstrate the intractability of computing symmetry breaking constraints in general, (Crawford *et al.* 1996) prove that the MAXIMUM INCIDENCE MATRIX problem is NP-complete. This result can be used immediately to show that the `MatrixSymmetry` constraint is intractable. For a graph (V, E), an incidence matrix can be constructed by taking any ordering on V and E and constructing the matrix A in which $A_{ij} = 1$ iff the ith vertex in V is connected by the jth edge in E. Given a graph (V, E) and an incidence matrix A for (V, E), the MAXIMUM INCIDENCE MATRIX problem is to determine if there is another incidence matrix B for (V, E) such that $B >_{\text{lex}} A$. It follows immediately that determining if an assignment satisfies the `MatrixSymmetry` constraint is coNP-complete. This is beyond our assumption that checking if an assignment satisfies a constraint is polynomial. Therefore, on such a "super" constraint, GACSUPPORT and MAXGAC would definitely be NP-hard. In addition, assuming P≠NP, there cannot exist any GAC-poly-time decomposition of the `MatrixSymmetry` constraint since on such a decomposition, satisfiability testing of an assignment would obviously be polynomial. As a result, whilst we may post symmetry breaking constraints in addition to the lexicographical ordering constraints, we cannot break all row and column symmetry in polynomial time.

Meta-constraints

Computational complexity can also be used to study "meta-constraints" that combine together other constraints. For example, the `Cardinality` constraint (Van Hentenryck & Deville 1991) is provided by many constraint toolkits. It ensures that N constraints from a given set are satisfied, where N is an integer decision variable. The cardinality constraint can be used to implement conjunction, disjunction, negation, as well as a host of other useful constraints. Only a limited form of consistency is enforced on a `Cardinality` constraint. It is easy to show why this is necessary in general.

Theorem 7 *Enforcing GAC on the* `Cardinality` *constraint is NP-hard, and remains so even if all the constraints are identical and binary and no variable is repeated more than three times.*

Proof: (Sketch) We use a reduction from the special case of 3SAT in which at most three clauses contain a variable or its negation. Each Boolean variable x is represented by a CSP variable X with domain $\{0, 1\}$. Each clause σ is represented by three CSP variables, U_σ, V_σ and W_σ, and five binary constraints posted on these variables. The domain of U_σ is a strict subset of $\{8, \ldots, 15\}$, of V_σ is a strict subset of $\{16, \ldots, 23\}$ and of W_σ is a strict subset of $\{24, \ldots, 31\}$. The domain values serve two purposes. First, the bottom three bits indicate the truth values taken by the variables that satisfy the clause. We therefore have to delete one value from each domain. This is the assignment of truth values which does not satisfy the clause. For example, if σ is $x \vee \neg y \vee z$ then the only assignment to X, Y and Z, which does not satisfy the clause is 0, 1, 0. We therefore delete the value 26 from W_σ as $26 \bmod 8$ is 2 (or 010 in binary). Similarly, we delete the value 18 from V_σ as $18 \bmod 8$ is 2, and 10 from U_σ. Second, the top two bits of the values of U_σ, V_σ and W_σ point to one of the three positions in the clause. We add three binary constraints to the cardinality constraint: $C(U_\sigma, X)$, $C(V_\sigma, Y)$ and $C(W_\sigma, Z)$.

We also need to ensure that U_σ, V_σ and W_σ take consistent values. We therefore add two binary constraints: $C(U_\sigma, V_\sigma)$, and $C(V_\sigma, W_\sigma)$. Finally, we define $C(X, Y)$ as follows. If $Y \in \{0, 1\}$, there are three cases. If $8 \leq X \leq 15$ then C is satisfied iff $(X \bmod 8) \operatorname{div} 4 = Y$ (i.e., the third bit of X agrees with Y). If $16 \leq X \leq 23$ then C is satisfied iff $(X \bmod 4) \operatorname{div} 2 = Y$ (i.e., the second bit of X agrees with Y). If $24 \leq X \leq 31$ then C is satisfied iff $X \bmod 2 = Y$ (i.e., the first bit of X agrees with Y). Otherwise $Y \geq 8$ and C is satisfied iff $X \bmod 8 = Y \bmod 8$.

The constructed cardinality constraint has a solution iff there is an assignment to the Boolean variables that satisfies all of the clauses. Hence enforcing GAC is NP-hard. ♡

A more restricted, but nevertheless very useful form of the cardinality constraint is the cardinality path constraint (Beldiceanu & Carlsson 2001). This "slides" a constraint of fixed arity down a sequence of variables and ensures that it holds N times, where N is itself an integer decision variable. This constraint can be used to implement the change, smooth, number of rests, and sliding sum constraints. In (Beldiceanu & Carlsson 2001), a greedy algorithm is given for partially propagating the cardinality path constraint. However, even for binary constraints, the algorithm fails to prune all possible values. Again, it is not hard to show that this constraint is intractable in general to propagate.

Learning nogoods

When we enforce GAC, we are essentially learning unary nogoods. A *nogood* is a partial assignment that cannot be extended to a complete solution. We can also apply the tools of computational complexity to study the learning of larger arity nogoods. We consider here two decision problems at the heart of learning. The first is NOGOOD testing, the problem of deciding if a partial assignment is nogood. In general, this is intractable to decide.

Theorem 8 NOGOOD *testing is coNP-complete, even for nogoods of bounded size.*

Proof: We consider the complement problem of GOOD testing. This is determining if an assignment can be extended to a solution. A polynomial witness to this is a solution that extends this assignment. To show completeness, we reduce SAT to GOOD testing. We construct a CSP with two disjoint parts. The first part accepts the partial assignment whatever. The second part uses disjoint variable names and encodes the SAT problem using a reduction of SAT to CSPs. Note that, even if the nogoods are of bounded size, the second part of the construction is large enough to perform the reduction of SAT to CSPs. ♡

The second problem we consider is MINIMALNOGOOD testing, the problem of determining if a (partial) assignment is nogood, whilst all strict subsets of the assignment are not. This is again D^P-complete.

Theorem 9 MINIMALNOGOOD *testing is D^P-complete, even for nogoods of bounded size.*

Proof: Recall that a problem Q is in D^P if there exist a NP problem Q_1 and a coNP one Q_2 such that Q answers "yes" iff Q_1 and Q_2 answer "yes". If Q_1 is NP-complete and Q_2 coNP-complete, then Q is D^P-complete. We use SAT and UNSAT as Q_1 and Q_2. Let φ_i be the formulae in Q_i being tested for (un)satisfiability. We suppose without loss of generality that φ_1 and φ_2 have disjoint sets of Boolean variables. Let x and y be two new Boolean variables. We construct the formula $(x \rightarrow \varphi_1) \wedge ((x \wedge y) \rightarrow \varphi_2)$. Then $x \wedge y$ is a minimal nogood iff φ_1 is SAT and φ_2 is UNSAT. ♡

Connections exist between the complexity of NOGOOD and MINIMALNOGOOD testing. For example, if NOGOOD testing is polynomial then MINIMALNOGOOD testing must also be polynomial. Connections also exist between the complexity of constraint propagation and that of testing nogoods. For example, if NOGOOD testing is polynomial then enforcing GAC is also. The reverse does not hold.

Theorem 10 *There exist a class of global constraints on which GAC is polynomial but* NOGOOD *testing is coNP-complete and* MINIMALNOGOOD *testing is D^P-complete.*

Proof: We construct a class of global constraints which encode the 3COL problem but on which GAC is polynomial. Each of the n nodes in the graph is represented by a variable. The domain of each variable contains the three colors and one don't care value. The global constraint is satisfied iff variables representing adjacent nodes in the graph take different colors, or $n-1$ of the variables have the don't care value. This encoding is always GAC (even if the graph is not 3-colorable) since no single value can be removed. However, testing whether an assignment of two different colors to two variables representing adjacent nodes is good is equivalent to determining if the graph can be 3-colored, which is NP-complete. Thus, NOGOOD testing is coNP-complete, and MINIMALNOGOOD testing is D^P-complete. ♡

Space prevents us from listing in detail some of the other applications we are exploring related to nogood learning. For example, we have identified a number of special cases where nogood learning is tractable. As a second example, we have studied the computational complexity of size and relevance bounded nogood learning. A related, yet different problem, the number of minimal nogoods, is addressed in (Dechter 1986).

Related work

For constraints of bounded arity, asymptotic analysis has been extensively used to study the complexity of constraint propagation both in general and for specific constraints. For example, the GAC-Schema algorithm of (Bessiere & Régin 1997) has an $O(d^n)$ time complexity on constraints of arity n and domains of size d, whilst the GAC algorithm of (Régin 1994) for the n-ary `AllDifferent` constraint has $O(n^{\frac{3}{2}}d)$ time complexity. By comparison, we have considered here what happens when we let the arity of global constraints grow. This happens in many real world problems. For instance, in the balanced incomplete block design problem (prob028 in CSPLib), the arity of the constraints grows with the problem size.

For global constraints like the `Cummulative` and `Cycle` constraints, there are very immediate reductions from the bin packing and Hamiltonian circuit which demonstrate that reasoning with these constraints is intractable in general. It is therefore perhaps not surprising that there has been little comment in the past about their intractability. However, as we show here, there are many other global constraints proposed in the past like `NValue` and `Atmost1` where a reduction is less immediate, but the constraint is intractable nevertheless.

In many constraint problems, the goal is not only to satisfy all the constraints, but also to minimize (or maximize) an objective function. Constraint propagation can be enhanced in these problems by cost-based filtering where we also remove values that are proven sub-optimal. *Optimisation constraints*, that combine a regular constraint of the problem with a constraint on the maximal value the objective function can take have been advocated in (Caseau & Laburthe 1997). GAC on such a combined constraint will not only prune the values having no support on the regular constraint, but also the values that do not extend to any satisfying assignment of the constraint improving the given bound. However, as in the case of combining constraints (see Section on combining constraints), such compositions have to be handled with care. The optimisation version of a constraint for which enforcing GAC is intractable obviously remains intractable (e.g., (Sellmann 2003a)). However, the optimisation version of a constraint for which GAC is polynomial either remains tractable (e.g., (Focacci, Lodi, & Milano 2002; Régin 2002)) or may become intractable. An example of the latter situation is the shorter path constraint, which is the optimization version of the path constraint (Sellmann 2003b).

Beldiceanu has proposed a general framework for describing many global constraints in terms of graph properties on structured networks of simple elementary constraints

(Beldiceanu 2000). It is an interesting open question if we can identify properties or elementary constraints within this framework which guarantee that a global constraint is computationally (in)tractable. Finally, computational complexity can help us classify constraints wrt the notions of globality proposed in (Bessiere & Van Hentenryck 2003). Indeed, NP-hardness of enforcing GAC is a sufficient condition for a constraint to be *operationally GAC-global* wrt GAC-polytime decompositions.

Conclusions

We have studied the computational complexity of reasoning with global constraints. We have shown that it is NP-complete in general to determine if a value has support, and D^P-complete to decide if a subdomain is the maximal generalized arc consistent subdomain. We have then demonstrated how the same tools of computational complexity can be used in the design and analysis of specific global constraints like the `NValue` and `Atmost1` constraints. In particular, we have illustrated how computational complexity can be used to determine when a lesser level of local consistency should be enforced, when decomposing constraints will reduce propagation and when constraints can be combined tractably. We have also shown how the same tools can be used to study symmetry breaking, meta-constraints like the `Cardinality` constraint, and learning nogoods. In the future, we plan an extensive study of the computational complexity of global constraints on set and multiset variables. Computational intractability is very common here as there are, in the worst case, an exponential number of sets or multisets between the upper and lower bounds on a variable.

References

Beldiceanu, N., and Carlsson, M. 2001. Revisiting the cardinality operator and introducing cardinality-path constraint family. In *Proceedings ICLP'01*, 59–73.

Beldiceanu, N., and Contegean, E. 1994. Introducing global constraints in CHIP. *Mathematical Computer Modelling* 20(12):97–123.

Beldiceanu, N. 2000. Global constraints as graph properties on a structured network of elementary constraints of the same type. In *Proceedings CP'00*, 52–66.

Beldiceanu, N. 2001. Pruning for the minimum constraint family and for the number of distinct values constraint family. In *Proceedings CP'01*, 211–224.

Bessiere, C., and Régin, J. 1997. Arc consistency for general constraint networks: preliminary results. In *Proceedings IJCAI'97*, 398–404.

Bessiere, C., and Van Hentenryck, P. 2003. To be or not to be ... a global constraint. In *Proceedings CP'03*, 789–794. Short paper.

Carlsson, M., and Beldiceanu, N. 2002. Arc-consistency for a chain of lexicographic ordering constraints. Technical report T2002-18, Swedish Institute of Computer Science. ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T–2002-18–SE.ps.Z.

Caseau, Y., and Laburthe, F. 1997. Solving various weighted matching problems with constraints. In *Proceedings CP'97*, 17–31.

Crawford, J.; Luks, G.; Ginsberg, M.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proceedings KR '96*, 148–159.

Dechter, R. 1986. Learning while searching in constraint-satisfaction-problems. In *Proceedings AAAI'86*, 178–183.

Flener, P.; Frisch, A.; Hnich, B.; Kiziltan, Z.; Miguel, I.; Pearson, J.; and Walsh, T. 2002. Breaking row and column symmetries in matrix models. In *Proceedings CP'02*, 462–476.

Focacci, F.; Lodi, A.; and Milano, M. 2002. Optimization-oriented global constraints. *Constraints* 7:351–365.

Frisch, A.; Hnich, B.; Kiziltan, Z.; Miguel, I.; and Walsh, T. 2002. Global constraints for lexicographic orderings. In *Proceedings CP'02*.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to NP-Completeness*. San Francisco CA: Freeman.

Hnich, B.; Kiziltan, Z.; and Walsh, T. 2004. Combining symmetry breaking with other constraints: lexicographic ordering with sums. In *Proceedings of the 8th International Symposium on the Artificial Intelligence and Mathematics*.

Knuth, D., and Raghunathan, A. 1992. The problem of compatible representatives. *SIAM Journal of Discrete Mathematics* 5(3):422–427.

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

Pachet, F., and Roy, P. 1999. Automatic generation of music programs. In *Proceedings CP'99*, 331–345.

Papadimitriou, C., and Yannakakis, M. 1984. The complexity of facets (and some facets of complexity). *J. Comput. System Sci.* 28:244–259.

Régin, J., and Rueher, M. 2000. A global constraint combining a sum constraint and difference constraint. In *Proceedings CP'00*, 384–395.

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

Régin, J. 1996. Generalized arc consistency for global cardinality constraint. In *Proceedings AAAI'96*, 209–215.

Régin, J. 2002. Cost-based arc consistency for global cardinality constraints. *Constraints* 7:387–405.

Sadler, A., and Gervet, C. 2001. Global reasoning on sets. In *Proceedings of Workshop on Modelling and Problem Formulation (FORMUL'01)*. held alongside CP-01.

Sellmann, M. 2003a. Approximated consistency for knapsack constraints. In *Proceedings CP'03*, 679–693.

Sellmann, M. 2003b. Cost-based filtering for shorter path constraints. In *Proceedings CP'03*, 694–708.

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

Walsh, T. 2003a. Consistency and propagation with multiset constraints: a formal viewpoint. In *Proceedings CP'03*, 724–738.

Walsh, T. 2003b. Constraint patterns. In *Proceedings CP'03*.