The Tractability of Global Constraints

Christian Bessiere[1], Emmanuel Hebrard[2], Brahim Hnich[2], and Toby Walsh[2]

[1] LIRMM-CNRS, Montpelier, France.
bessiere@lirmm.fr
[2] Cork Constraint Computation Center, University College Cork, Ireland.
{e.hebrard, b.hnich, tw}@4c.ucc.ie

Abstract. Constraint propagation is one of the techniques central to the success of constraint programming. Fast algorithms are used to prune the search space either before or during backtracking search. Propagating global constraints is intractable in general. In this paper, we characterize a number of important questions related to constraint propagation. For example, we consider the two questions: "Is this problem generalized arc-consistent?" and "What are the maximal generalized arc-consistent domains?". We identify dependencies between the tractability and intractability of these questions for finite domain variables. Finally, we prove intractability for a range of global constraints.

1 Introduction

It is well known that constraint propagation on binary (or bounded arity) constraints is polynomial. However, constraint toolkits support an increasing number of non-binary or global constraints. Global constraints permit users to model problems compactly and solvers to prune the search space efficiently and effectively. In many problems, the arity of such global constraints can grow with the problem size. Such global constraints may therefore exhibit complexities far beyond the quadratic propagation cost of binary constraints. Indeed, it is easy to see that reasoning with global constraints is intractable in general. In this paper, we characterize the different reasoning tasks related to constraint propagation. For example, does this value have support? As a second example, what are the maximal generalized arc-consistent domains? We identify dependencies between the tractability and intractability of these different questions. Afterwards, we study a range of existing and new global constraints We show that they are NP-hard to propagate. Thus, we expect that any decomposition will hinder propagation (unless P=NP).

2 Theoretical Background

A *constraint satisfaction problem* (CSP) involves 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 will denote variables with upper case letters and values with lower case. We assume a constraint C is given intensionally by a function of the form $f_C : D(X_1) \times \ldots \times D(X_n) \mapsto \{True, False\}$ where $D(X_i)$ are the domains of the variables in the scope $var(C) = (X_1, \ldots, X_n)$ of the constraint C. (We say that D is a domain on $var(C)$.) We only consider constraints C for which f_C is computable in polynomial time.

Constraint toolkits usually contain a library of predefined *constraint types* with a particular semantics that can be applied to sets of variables with various arities and domains. A constraint is only an instance of a constraint type on given variables and domains. For instance, *alldifferent* is a constraint type. *alldifferent*$(X_1, .., X_3)$ with $D(X_1) = D(X_2) = \{1, 2\}, D(X_3) = \{1, 2, 3\}$ is an instance of constraint of the type *alldifferent*.

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 to prune the search space. One of the oldest and most commonly used local consistencies is generalized arc consistency. 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 to the other variables in C such that C is satisfied [5]. This satisfying assignment is called *support* for the value. In general, applying GAC can remove any value anywhere in the domain of a variable. This is why GAC is usually applied to constraints that involve *finite domain* variables. In the following, we will consider finite domain integer variables in which every value in the domain is given extensionally.

3 Complexity of Generalized Arc Consistency

We characterize five different problems related to genaralized arc consistency reasoning. These problems can be adapted to any other local consistency as long as it rules out values in domains (e.g., bounds consistency, singleton arc consistency, etc.) and not non-unary tuples of values (e.g., path consistency, relational-k-consistency, etc.)

In the following, PROBLEM($\mathcal{C}$) represents the class of problems defined by PROBLEM on constraints of the type $\mathcal{C}$. PROBLEM($\mathcal{C}$) will sometimes be written PROBLEM when no confusion is possible. Note also that we use the notation PROBLEM[data] to refer to the instance of PROBLEM($\mathcal{C}$) with the input 'data'. $\mathcal{U}$ denotes the set of all constraint types.

Table 1 contains the five problems. The first problem we consider is GACSUPPORT. It is at the core of all the generic arc consistency algorithms. The second problem, ISITGAC, is not directly related with operations used in basic propagation algorithms. It is largely introduced for academic purposes. The third question, NOGACWIPEOUT, can be used to decide if we do not need to backtrack at a given node in the search tree. (Note that $D' \subseteq D$ stands for: $\forall X_i \in var(C), D'(X_i) \subseteq D(X_i)$.) An algorithm like GAC-Schema [4] 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. MAXGAC characterizes this "maximality" problem. We finally consider GACDOMAIN, the non-decision problem of returning the domains that a GAC algorithm computes.

In the following, we describe the relationships between the tractability and intractability of the different problems defined above.

Table 1. The five problems related to genaralized arc consistency

Problem	Instance	Question/Output
GACSUPPORT($\mathcal{C}$)	$C \in \mathcal{C}$, D on $var(C)$, $X \in var(C)$, and $v \in D(X)$	Does value v for X have a support on C in D?
ISITGAC($\mathcal{C}$)	$C \in \mathcal{C}$, D on $var(C)$	Does GACSUPPORT$[C, D, X, v]$ answer "yes" for each variable $X \in var(C)$ and each value $v \in D(X)$?
NOGACWIPEOUT($\mathcal{C}$)	$C \in \mathcal{C}$, D on $var(C)$	Is there any non empty $D' \subseteq D$ on which ISITGAC$[C, D']$ answers "yes"?
MAXGAC($\mathcal{C}$)	$C \in \mathcal{C}$, D on $var(C)$, and $D \subseteq D_0$	Is it the case that ISITGAC$[C, D]$ answers "yes" and $\nexists D'$, $D \subset D' \subseteq D_0$, on which ISITGAC$[C, D']$ answers "yes"?
GACDOMAIN($\mathcal{C}$)	$C \in \mathcal{C}$, D_0 on $var(C)$	The domain D such that MAXGAC$[C, D_0, D]$ answers "yes"

3.1 Tractable cases

The five problems defined above are not independent. Knowledge about the tractability of one can give information on the tractability of others. We identify here the dependencies between the tractabilities of the different questions.

Theorem 1. *Given a constraint type $\mathcal{C}$,*

1. GACSUPPORT $\in P$ *iff* NOGACWIPEOUT $\in P$ *iff* GACDOMAIN $\in P$
2. $\{$GACSUPPORT, NOGACWIPEOUT, GACDOMAIN$\} \in P \Rightarrow$ ISITGAC $\in P$
3. $\{$GACSUPPORT, NOGACWIPEOUT, GACDOMAIN$\} \in P \Rightarrow$ MAXGAC $\in P$
4. MAXGAC $\in P \Rightarrow$ ISITGAC $\in P$

Proof. (1) If GACSUPPORT is in P, then we can answer NOGACWIPEOUT just by checking that at least one value in the domain of a variable X in $var(C)$ has support. Indeed, all the values in the support themselves have support.

If NOGACWIPEOUT is in P, we can check that (X, v) has support just by calling NOGACWIPEOUT with only v in the domain of X.

It is trivial that if GACSUPPORT is in P, using a generic GAC algorithm that calls GACSUPPORT a polynomial number of times, we will have the output of GACDOMAIN in polynomial time.

By calling GACDOMAIN with only (X, v) in the domain of X, the obtained domain will be non empty iff (X, v) has a support, then answering GACSUPPORT in polynomial time if GACDOMAIN was in P.

(2) Trivial.

(3) Trivial.

(4) If MAXGAC is in P, it is sufficient to call it with D both as the initial and current domain to answer ISITGAC on D. $\square$

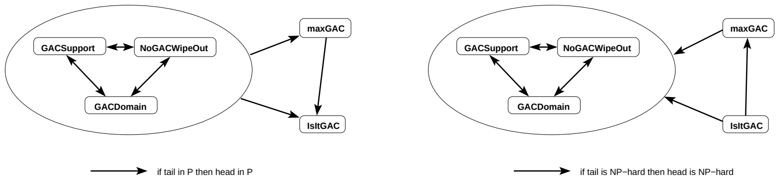

Fig. 1. Summary of the dependencies between problems

3.2 Intractable cases

We can identify similar dependencies between these questions when they are intractable. Interestingly, we have only been able to identify the inverse relationships between MAXGAC, ISITGAC and the other three problems. It is a challenging open question to prove either that one of these results which is just an implication reverses or that it does not reverse in general.

Theorem 2. *Given a constraint type $\mathcal{C}$,*

1. GACSUPPORT *is NP-complete iff* NOGACWIPEOUT *is NP-complete iff* GACDOMAIN *is NP-hard*
2. ISITGAC *is NP-complete* $\Rightarrow$ GACSUPPORT, NOGACWIPEOUT, GACDOMAIN *are NP-hard*
3. MAXGAC *is NP-hard* $\Rightarrow$ GACSUPPORT, NOGACWIPEOUT, GACDOMAIN *are NP-hard*
4. ISITGAC *is NP-complete* $\Rightarrow$ MAXGAC *is NP-hard*

Proof. (1) GACSUPPORT($\mathcal{C}$) can be transformed in NOGACWIPEOUT($\mathcal{C}$): Given $C \in \mathcal{C}$, GACSUPPORT$[C, D, X, v]$ is solved by calling NOGACWIPEOUT$[C, D|_{D(X)=\{v\}}]$.

NOGACWIPEOUT$[C, D]$ can be reduced to GACSUPPORT by calling GACSUPPORT$[C, D, X, v]$ for each value v in $D(X)$ for one of the X in $var(C)$. GAC leads to a wipe out iff none of these values has a support.

GACSUPPORT($\mathcal{C}$) can be reduced to GACDOMAIN($\mathcal{C}$) since GACSUPPORT$[C, D, X, v]$ answers "yes" iff GACDOMAIN$[C, D|_{D(X)=\{v\}}]$ doesn't return empty domain.

GACDOMAIN$[C, D]$ can be reduced to GACSUPPORT by performing a polynomial number of calls to GACSUPPORT$[C, D, X, v]$, one for each $v \in D(X)$, $X \in var(C)$. When the answer is "no" the value v is removed from $D(X)$, otherwise it is kept. The domain obtained at the end of this process represents the output of GACDOMAIN.

(2) ISITGAC$[C, D]$ can be reduced to GACSUPPORT by performing a polynomial number of calls to GACSUPPORT$[C, D, X, v]$, one for each $v \in D(X)$, $X \in var(C)$. If one of them answers "no" ISITGAC answers "no", otherwise it answers "yes".

(3) MAXGAC$[C, D_0, D]$ can be reduced to GACSUPPORT. We perform a polynomial number of calls to GACSUPPORT$[C, D_0, X, v]$, one for each $v \in D_0(X)$, $X \in var(C)$. When the answer is "yes" the value v is added to a (initially empty) set $D'(X)$. MAXGAC answers "yes" if and only if the domain D' obtained at the end of the process is equal to D.

(4) ISITGAC$[C, D]$ can be transformed in MAXGAC$[C, D, D]$. □

Table 2. A list of counting constraints that are intractable to propagate with GAC.

Name	Definition
$\texttt{nvalue}(N, [X_1, \ldots, X_n])$	$N = \{X_i \mid 1 \leq i \leq n\}$
$\texttt{egcc}([X_1, \ldots, X_n], [O_1, \ldots, O_m])$	$\forall j, O_j = \sum_i \lvert X_i = j \rvert$
$\texttt{rgcc}([[X_1, \ldots, X_n]], [o_1, \ldots, o_m])$	$\forall j, o_j = \sum_i \lvert X_i = j \rvert$
$\texttt{common}(N, M, [X_1, \ldots, X_n], [Y_1, \ldots, Y_m])$	$N = \{i \mid X_i = Y_j\}$ and $M = \{j \mid X_i = Y_j\}$
$\texttt{cardpath}(N, [X_1, \ldots, X_m], C)$	$N = \sum_{i=1}^{m-k+1} \lvert C(X_i, \ldots, X_{i+k-1}) \rvert$

4 Examples of intractable constraints

In the long version of this paper [3], we give a number of constraints for which the complexity of GAC was not known. We use the basic tools of computational complexity to show their tractability or intractability. Table 2 gives some of the intractability results we obtained for counting constraints on integer variables. Proofs are in [3].

The `nvalue` constraint was proposed in [6]. The extended global cardinality constraint, `egcc`, allows the O_j to be variables and not just fixed intervals as in [8]. `egcc` has been proved intractable in [7]. The `rgcc` constraint is a simple `gcc` in which repetitions of variables are allowed in the sequence $[[X_1, \ldots, X_n]]$. The `common` constraint was introduced in [1]. The `cardpath` constraint [2], ensures that when we slide C down the sequence $X_1, \ldots, X_m$ it holds N times. `cardpath` is intractable even if enforcing GAC on C is polynomial and the sequence of variables $[X_1, \ldots, X_m]$ does not contain any repetition.

Aknowledgements The last three authors are supported by Science Foundation Ireland and an ILOG software grant. We thank Marie-Christine Lagasquie for some advice on reducibility notions.

References

1. N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. SICS Technical Report T2000/01.
2. N. Beldiceanu and M. Carlsson. Revisiting the cardinality operator and introducing cardinality-path constraint family. In *Proceedings ICLP'01*, pages 59–73, 2001.
3. C. Bessiere, E. Hebrard, , B. Hnich, and T. Walsh. The tractability of global constraints. Technical Report APES-83-2004, APES Research Group, May 2004.
4. C. Bessiere and J.C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proceedings IJCAI'97*, pages 398–404, 1997.
5. R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings ECAI'88*, pages 651–656, 1988.
6. F. Pachet and P. Roy. Automatic generation of music programs. In *Proceedings CP'99*, pages 331–345, 1999.
7. C. Quimper. Enforcing domain consistency on the extended global cardinality constraint is NP-hard. TR CS-2003-39, School of Computer Science, University of Waterloo, 2003.
8. J-C. Régin. Generalized arc consistency for global cardinality constraints. In *Proceedings AAAI'96*, pages 209–215, 1996.