Reasoning About NP-complete Constraints
Emmanuel Hébrard

To cite this version:
Emmanuel Hébrard. Reasoning About NP-complete Constraints. 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), Jul 2018, Stockholm, Sweden. 5p. hal-01873503

HAL Id: hal-01873503
https://laas.hal.science/hal-01873503
Submitted on 13 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Reasoning About NP-complete Constraints

Emmanuel Hebrard
LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
hebrard@laas.fr,

Abstract
The concept of local consistency – making global deductions from local infeasibility – is central to constraint programming. When reasoning about NP-complete constraints, however, since achieving a “complete” form of local consistency is often considered too hard, we need other tools to design and analyze propagation algorithms.

In this paper, we argue that NP-complete constraints are an essential part of constraint programming, that designing dedicated methods has lead to, and will bring, significant breakthroughs, and that we need to carefully investigate methods to deal about a necessarily incomplete inference. In particular, we advocate the use of fixed-parameter tractability and kernelization to this purpose.

1 Introduction
Constraint programming (CP) has been a very successful framework for modeling and solving hard combinatorial problems. Many problems are naturally framed as constraint satisfaction or optimization problems, where a set of discrete valued variables is to be assigned while satisfying a set of constraints. For instance, CP has been successfully applied to managing the aftermath of natural disasters [Hentenryck, 2013, optimizing the delivery of radiotherapy in cancer treatment [Cambazard et al., 2012] or scheduling the exploration of a comet [Simonin et al., 2012].

A first strength of CP often put forward, is its declarative aspect. Another essential feature is the modularity and versatility of the inference mechanisms. The key principle is the notion of local consistency: if we can deduce that some assignments are locally infeasible, for instance with respect to a single constraint, then the same deduction holds globally. In particular, a constraint network is said domain consistent [Waltz, 1975] when there is no further deduction to be made, with respect to any single constraint, about the possible values that its variables can take. It was shown that maintaining this level of consistency during search was often a good choice [Sabin and Freuder, 1994] and most constraint solvers are designed according to this principle.

The notion of global constraint makes this type of reasoning extremely powerful. A constraint is said global when it corresponds to a class of relations on an arbitrary number of variables. For instance, the constraint ALLDIFFERENT ensures that a set of variables take all pairwise distinct values. Given a recurring subproblem for which there exists an efficient algorithm, it is natural to derive the corresponding constraint, that is, design a propagation algorithm responsible for making deduction locally to that constraint. For instance, the propagation algorithm for the ALLDIFFERENT constraint [Régis, 1994] is based on Hopcroft and Karp’s maximum matching algorithm [Hopcroft and Karp, 1973]. Similarly, a number of constraints are propagated using results from flow theory [van Hoeve et al., 2006] or dynamic programming [Pesant, 2004; Hebrard et al., 2009].

Since constraints must be easily checkable, they lie within the complexity class NP. However, many recurring subproblems that might qualify as constraints are NP-complete [Bessiere et al., 2006]. Examples of NP-complete constraints are numerous. For instance for ensuring that a set of Boolean variables represent a clique of a certain size [Fahle, 2002; Régis, 2003]; that integer variables representing vertices of a graph are given a value corresponding to an isomorphic vertex in a larger graph [Régis, 1995]; that the least and highest number of occurrences of any value is within some bounds [Schaus et al., 2007]; that two overlapping sets of variables both have all pairwise distinct values [Kutz et al., 2008]; that there is an upper bound on the cardinality of the set of assigned values [Bessiere et al., 2006]; that a sequence of variables encode the successors of an Hamiltonian circuit of bounded length in a graph [Ducommman et al., 2016]; or that a set of tasks share a disjunctive [Baptiste and Le Pape, 1995] or cumulative resource [Baptiste and Le Pape, 1997].

However, whereas one can enforce domain consistency on polynomial constraints, it is often necessary to find a compromise between inference strength and computational effort on NP-complete constraints. In this paper, we advocate the design of propagation algorithms for NP-complete constraints. Furthermore, we argue that there is a need for a theoretical characterization of the consistency enforced by these algorithms. We show that fixed-parameter tractability [Downey et al., 1999], and in particular the notion of kernel are extremely relevant in this context and that the latter can be extended to fill this role of measure of propagation strength.

Or equivalently arc consistent
2 NP-complete Constraints

A constraint network is a triple \((\mathcal{X}, \mathcal{U}, \mathcal{C})\) where \(\mathcal{X}\) is a totally ordered set of variables, the universe \(\mathcal{U}\) is a finite set of values and \(\mathcal{C}\) is a set of constraints. A constraint \(C\) is a pair \((S_C, R_C)\) where the scope \(S_C\) is a subset of \(\mathcal{X}\) and \(R_C\) is a relation on \(\mathcal{U}\) of arity \(|S_C|\). The Constraint Satisfaction Problem (CSP) asks whether a constraint network \((\mathcal{X}, \mathcal{U}, \mathcal{C})\) has a solution, that is, a \(|\mathcal{X}|\)-tuple \(\sigma \in \mathcal{U}^{|\mathcal{X}|}\) such that for each \(C \in \mathcal{C}\), the projection \(\sigma(S_C)\) of \(\sigma\) onto \(S_C\) is in \(R_C\).

A global constraint is a class of relations, one for each possible value of its parameters. The most common parameter is the number of constrained variables. For instance we can define the NVALUE [Pachet and Roy, 1999] constraint with one relation per integer \(n\) given by the following predicate:

**Definition 1** (NVALUE).

\[
\text{NVALUE}(x_1, \ldots, x_n, y) \iff |\{x_i | 1 \leq i \leq n\}| \leq y
\]

During search, in order to store the decisions and deductions, we use a special type of relation over \(\mathcal{X}\), the domain \(\mathcal{D}\). There is not a unique type of relation used for that purpose, for instance the search space can be encoded as a Boolean Decision Diagram [Hadzic and Hooker, 2007]. However we consider here the canonical finite discrete domain where the \(\mathcal{D}\) relation is restricted to conjunctions of finite unary relations, one for every variable \(x \in \mathcal{X}\), which we denote \(D(x)\). We write \(D' \subseteq D\) as a shortcut for \(\forall x \in \mathcal{X}\ D'(x) \subseteq D(x)\).

**Definition 2.** A constraint \(C\) is said domain consistent on a finite discrete domain \(\mathcal{D}\) if and only if

\[
\forall D' \subseteq D \exists \tau \in R_C \cap (D(S_C) \setminus D'(S_C))
\]

The problem of achieving domain consistency of a domain \(D\) with respect to a constraint \(C\) consists in finding the largest domain \(D' \subseteq D\) such that \(C\) is arc consistent on \(D'\). There is a unique largest arc consistent closure of a finite discrete domain \(D\) with respect to a constraint \(C\).

Notice that achieving domain consistency is polynomially reducible to checking the satisfiability of a constraint, i.e., finding a tuple \(\tau \in D \cap C\). Given an algorithm for checking satisfiability, for every variable \(x\) and every value \(v \in D(x)\) one can check the satisfiability of \(C\) w.r.t. \(D'\) where \(D'(x) = \{v\}\) and \(D' = D\) otherwise. If the constraint is not satisfiable, then the domain consistent closure of \(D\) is such that \(v \notin D(x)\) and we can change \(D\) accordingly. Otherwise the domain consistent closure is such that \(v \in D(x)\) and we leave \(D\) unchanged. This procedure terminates after \(\sum_{x \in S_C} |D(x)|\) steps. The converse is trivial, since a constraint is satisfiable if and only if its domain consistent closure is not empty. We therefore say that a constraint is NP-complete if checking its satisfiability is NP-complete, even though achieving domain consistency is not a decision problem.

The constraint NVALUE is NP-complete since checking if there exists a tuple in a domain \(D\) such that \(|\{x_i | 1 \leq i \leq n\}| \leq y\) is equivalent to finding a hitting set of the collection \(\{D(x_1), \ldots, D(x_n)\}\) of size \(\max D(y)\) [Bessiere et al., 2006a]. Therefore, achieving the domain consistent closure is often considered too costly for this constraint. For NVALUE and other NP-complete constraints incomplete approaches are used instead.

**Relaxing the domain:** A common way to reduce the complexity of propagating a constraint is to relax the domain relation \(D\). For instance, the notion of bounds consistency is widely used in this situation. Let interval domains be the class of relations restricted to conjunctions of finite unary relations, one for every variable \(x \in \mathcal{X}\), of the form \(l \leq x \leq u\). Bounds consistency is the property obtained by applying Definition 2 to interval, instead of discrete, domains.

The hitting set problem has a polynomial algorithm when the sets are discrete intervals. Therefore, the relaxation from finite discrete domains to interval domains makes satisfiability checking and thus bounds consistency polynomial. The best algorithm achieves both in \(O(|\mathcal{X}| \log |\mathcal{X}|)\) [Beldiceanu, 2001]. This is true for many NP-complete global constraints. For instance the constraint GCC, which channels the number of occurrences of values in a set of variables \(\mathcal{X}\) to another set of variables \(\mathcal{Y}\) is NP-complete for discrete domains [Quimper et al., 2004], but polynomial if the domain relation is relaxed to intervals for the variables \(\mathcal{Y}\) [Regin, 1996].

**Decomposing the constraint:** Another common way to deal with NP-completeness is to decompose the constraint. A constraint network \(T = (\mathcal{X}, \mathcal{D}, \mathcal{C})\) is a decomposition of a constraint \(C\) if and only if \(S_C \subseteq \mathcal{X}\) and \(\sigma\) is a solution of \(T\) if and only if \(\sigma(S_C) \in R_C\). In other words, it is a constraint network on a superset of \(S_C\) such that its set of solutions, when projected onto \(S_C\) is exactly the set of tuples in \(R_C\).

Decompositions are used in countless constraint programming approaches. A lot of effort has been put into the study of decompositions and in particular when it is or when it is not possible to emulate algorithms through decompositions, both for polynomial and NP-complete constraints [Narodytska, 2011]. For instance, it was shown that propagation algorithms for the NVALUE constraint can be emulated for the same time complexity, however at the cost of a much higher space complexity [Bessiere et al., 2010].

Moreover, decompositions are sometimes used as a way to relax the problem of achieving the domain consistent closure, even though an efficient dedicated algorithm is then proposed to compute the closure. For instance the \(O(n \log n)\) “Timetabling” method for reasoning about a set of \(n\) tasks consuming energy from a cumulative resource [Le Pape, 1988; Beldiceanu and Carlsson, 2001] over time can be seen as achieving bounds consistency on a decomposition involving Boolean variables \(x_{ij}\) standing for whether task \(i\) is processed at time \(j\), however with a far lower time complexity. Similarly, “Edge Finding” algorithms [Nuijten and Aarts, 1994; Vilím, 2009] achieve bounds consistency (in polynomial time) on a decomposition (of exponential size). Lastly, Ouellet and Quimper’s algorithm [Ouellet and Quimper, 2013] achieves bounds consistency on the conjunction of these two decompositions in \(O(kn \log n)\) time with \(k\) the number of different tasks’ consumptions.

**Approximation:** Often, NP-complete constraints on a set of variables \(\mathcal{X}\) can be easily seen as enforcing that an \(|\mathcal{X}|\)-ary *cost function* \(\pi : D^{|\mathcal{X}|} \rightarrow \mathbb{Q} \cup \{-\infty, +\infty\}\) is non-positive.

\(^2\)This is not restrictive as any relation maps tuples to \(0/1\).
The size of a vertex cover can never be increased by removing all vertices from \( I \) and adding all vertices from \( W \). However, this is a correct dominance rule, but not a correct propagation rule: feasible or even minimal solutions might involve vertices from \( I \). Similarly, an efficient kernelization using dominance was proposed for the NVALUE constraint [Gaspers and Szeider, 2011]. For the same reason, this method is to be used within the probing procedure described in Section 2 to achieve domain consistency through satisfiability checks: for every variable-value pair, the problem of deciding if the constraint is satisfiable when restricting the domain accordingly is solved on the kernel. In other words, several exponential satisfiability checks have to be performed.

In [Carbonnel and Hebrard, 2016; 2017] we proposed an approach to propagating NP-complete constraints based on a new definition of “loss-less” kernelization tailored for constraint propagation. Intuitively, \( z \)-loss-less kernels are kernels with an extra loss-lessness property: Whereas solving a classical kernel is sufficient to solve the original instance, achieving domain consistency on a loss-less kernel is sufficient to achieve domain consistency on the original instance. In other words, there exists a polynomial algorithm, which, given the domain consistent closure of the loss-less kernel, computes it for the original instance. This is much more consistent with the spirit of kernelization, extends smoothly constraints with polynomial-time propagators and yields a propagation algorithm with running time \( O(g(p) + |I|^{O(1)}p^{O(1)}) \), instead of \( O(|I|^{O(1)}g(p)) \) for probing plus classical kernelization.

There is caveat, however. Consider again constraints defined as minimizing a cost function, for instance the NVALUE constraint which ensures that the cardinality of the set of values taken by the variables \( x_1, \ldots, x_n \) is at most \( y \). As long as the domain of \( y \) contains high values, the constraint is unlikely to propagate much. In fact, if \( n \in D(y) \) then the constraint is completely inoperative. More generally, kernels whose size decrease when the constraint get tighter should be prioritized since they are the most useful with respect to propagation. When defining constraints as minimizing a cost function such as the NVALUE constraint, the tightness is very clearly related to the gap \( z \) from 0 to the minimum of \( \pi_D(X) \) under the current domain \( D \). We therefore use this extra parameter \( z \) in the definition of loss-less kernels:

**Definition 5.** Let \( \Pi \) be a set of cost functions, and \( C \) a global constraint defined as the relations \( \pi(X) \leq 0 \) for \( \pi \in \Pi \).

A \( z \)-loss-less kernelization of \( C \) with parameter \( p \) is a polynomial-time computable function mapping each instance \( (\pi, X, D) \) and parameter value \( k \) to an instance \( (\pi', X', D') \) and parameter \( k' \) of the same constraint such that if \( \min(\pi_D(X)) \leq z \), then there is a polynomial algorithm achieving domain consistency of \( D \) w.r.t. \( \pi(X) \leq 0 \) given the the domain consistent closure of \( D' \) w.r.t. \( \pi'(X') \leq 0 \), \( |\pi'| + |D'| + |X'| \leq f(k) \), and \( k' \leq g(k) \) for some computable functions \( f, g \).

The results in [Carbonnel and Hebrard, 2017] show that loss-less kernels exist: There is a \( (z + 2)k \)-loss-less kernel for the vertex cover problem parameterized by the size \( k \) of the cover, thus matching the result of Nemhauser [Nemhauser...
and Trotter Jr. 1975] for \( z = 0 \). Similarly, there is a \( \infty \)-lossless kernel \( \max(6k, k^2/2 + 7k/2) \) kernel for the edge dominating set parameterized by the size of the dominating set, thus matching the result of Hagerup [Hagerup, 2012].

Moreover, they are not mere theoretical curiosities: kernel-based propagation was successfully applied to a constrained vertex cover problem [Carbonnel and Hebrard, 2016].

4 Conclusions

In this paper we have surveyed different approaches to propagating NP-complete constraints and argued that designing dedicated methods is extremely valuable. Moreover, among the possible ways of tackling the propagation of NP-complete constraints (approximation, relaxation, decomposition, etc.), we argue that fixed parameterized complexity and in particular kernelization offers several extremely relevant features:

Firstly, the value of the parameter, as well as the size of the gap \( z \), changes during search. Therefore, it is possible to target when kernelization is most likely to being beneficial and thus use it in an opportunistic way.

Secondly, loss-less kernels are designed to achieve domain consistency of the full instance with a single call to a possibly exponential algorithm to compute the closure of the kernel. However, in practice, the kernelization procedure in itself makes some incomplete inference while being polynomial. The kernelization process in itself can therefore be used as a polynomial propagation procedure.

Finally, the guarantee on the size of the kernels entails a guarantee on the strength of this inference. In other words, the size of the kernel is a valuable criterion to compare the achieved level of consistency.

References


