ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-919

Intermediary Local Consistencies

Thierry Petit[1]

Abstract. We propose a new definition for characterizing levels of consistency. A perspective is to provide new tools for classifying filtering algorithms, including incomplete algorithms based on the semantics of constraints.

1 INTRODUCTION

A filtering algorithm associated with a constraint is complete if it achieves generalized arc-consistency (GAC). This means that the algorithm removes from domains of variables all the values which cannot belong to at least one valid assignment satisfying the constraint. Depending on the size of problems and the nature of constraints, it is not always possible to enforce GAC. For instance, if a constraint involves $n = 500$ variables, using a $O(n^3)$ GAC algorithm is generally not reasonable. The same remark can be made for algorithms using too heavy data structures. In this context, constraints are associated with algorithms that enforce weaker forms of consistency. These intermediary consistencies are not always clearly formalized.

This article provides a new generic definition for characterizing the levels of consistency associated with constraints. We discuss some practical examples and identify the metrics that can be used to classify the different levels of consistency.

2 SUPPORT-DIRECTED CONSISTENCIES

Given a constraint $C(X)$ defined on a set of variables X, a filtering algorithm removes values which, given the current variable domains, cannot belong to a solution satisfying the constraint. Some filtering algorithms evaluate all the values in domains (*e.g.*, GAC), while some other evaluate only the bounds of each domain, *e.g.*, Bounds-Consistency (BC). In both cases, the viability of a value v in the domain $D(x)$ of a variable $x \in X$ is checked by considering either the set of solutions of $C(X)$ according to the current domains, or a superset of these solutions. We call this superset a *relaxation* of $C(X)$. This relaxation can be obtained either by relaxing the constraint $C(X)$ itself, or by adding "virtually" some values in domains, for instance by considering that domains have no holes. If value v cannot belong to at least one solution of the relaxation then it can be removed from $D(x)$. Such a solution is called a *support*. Thus, two main notions characterize the level of consistency of a filtering algorithm: 1. The set of checked values (either all the values in domains or only the bounds). 2. The relaxation of the constraint used to search supports for values. We represent this relaxation by a set of constraints. This point of view leads to a new definition of local consistency, parameterized by the relaxation. We use the notations $\min(x) = \min(D(x))$ and $\max(x) = \max(D(x))$.

Definition 1 *Given a constraint $C(X)$, let $Y = \{y_1, y_2, \ldots, y_n\}$ be a set of variables one-to-one mapped with $X = \{x_1, x_2, \ldots, x_n\}$, such that $\forall x_i \in X, \forall y_i \in Y$, $D(x_i) \subseteq D(y_i)$. Let $\mathcal{C} = \{C_1(Y_1), C_2(Y_2), \ldots, C_m(Y_m)\}$ be a set of constraints such that $Y_1 \cup Y_2 \ldots Y_m \subseteq Y$ and $C(Y) \Rightarrow C_1(Y_1) \wedge C_2(Y_2) \ldots C_m(Y_m)$. Value $v \in D(x_i)$ has a $(\mathcal{C}, Y)$-**support** on $C(X)$ if and only if $\forall C_j(Y_j) \in \mathcal{C}$, either the variable y_i mapped with x_i is not in Y_j or $y_i \in Y_j$ and $C_j(Y_j)$ has a solution with $y_i = v$.*
*Constraint $C(X)$ is $(\mathcal{C}, Y)$-**DC** ($(\mathcal{C}, Y)$-Domain Consistent) if and only if $\forall x_i \in X$, $\forall v \in D(x_i)$, v has a $(\mathcal{C}, Y)$-support on $C(X)$.*
*$C(X)$ is $(\mathcal{C}, Y)$-**BC** ($(\mathcal{C}, Y)$-Bounds Consistent) if and only if $\forall x_i \in X$, $min(x_i)$ and $max(x_i)$ have a $(\mathcal{C}, Y)$-support on $C(X)$.*

Definition 1 can be specialized to the usual notions of GAC, BC and Range-Consistency (RC) [2].

Property 1 *$C(X)$ is* GAC $\equiv C(X)$ *is* $(\{C(X)\}, X)$-DC. *Let Y be a variable set one-to-one mapped with X, with $\forall y_i \in Y, D(y_i) = \{\min(x_i), \min(x_i) + 1, \ldots, \max(x_i)\}$. $C(X)$ is* RC $\equiv C(X)$ *is* $(\{C(Y)\}, Y)$-DC. *$C(X)$ is* BC $\equiv C(X)$ *is* $(\{C(Y)\}, Y)$-BC.

Definition 1 is not restricted to the case where all the variables in Y are involved in constraints of $\mathcal{C}$, but constraints in $\mathcal{C}$ use exclusively variables derived from variables in X. Our goal is to characterize filtering algorithms, not reformulations. We thus consider that a filtering algorithm does not add new variables to the problem. Other generic consistencies can be defined, for instance by relaxing only a subset of variables in X, or by checking real supports (like GAC) only for the bounds ($(\{C(X)\}, X)$-BC). Furthermore, Definition 1 characterizes the level of consistency of some specialized filtering algorithms, which are not always clearly formalized in the literature.

Example 1 *Consider the constraint $s = \sum_{x_i \in X} x_i$.* GAC *is* NP-Hard.[2] *Conversely, enforcing* GAC *on $\sum_{x_i \in X} x_i \leq s$ is in* P [14]. *Therefore, a possible consistency for $s = \sum_{x_i \in X} x_i$ is* $(\mathcal{C}, Y)$-DC *with $Y = X \cup \{s\}$ and $\mathcal{C} = \{\sum_{x_i \in X} x_i \leq s, \sum_{x_i \in X} x_i \geq s\}$.* ⊛

In Example 1, the obtained level of consistency is equivalent to BC. This is not the case for some other filtering algorithms of constraints that use a similar principle of relaxation for checking supports, such as $s = \sum_{x_i \in X} x_i^2$, or the filtering algorithm of *Cost-regular* [3]. With respect to soft global constraints [9, 6], the variable that measures a violation degree is also generally filtered separately from its minimum value and from its maximum value [5]. Many other examples exist, some of them relax both the constraint and the variables.

Comparison with Guido Tack's Dissertation Tack [12] proposed a characterization of propagation levels. The notion of completeness of domain approximations provides a classification. This

[1] TASC (Mines Nantes, LINA, CNRS, INRIA), 4, Rue Alfred Kastler, FR-44307 Nantes Cedex 3, France, email: Thierry.Petit@mines-nantes.fr

[2] Checking the satisfiability of $k = \sum_{x_i \in X} x_i$, where k is an integer, requires in the general case to solve the NP-Hard Subset-sum problem [4, p. 223].

characterization also considers the case of set variables, conversely to Definition 1. It is thus more generic. Definition 1 deals with a *set of constraints* $\mathcal{C}$ implied by the original constraint, which corresponds concretely to many filtering algorithms of constraints, provided by existing solvers. By evaluating properties of the set we obtain some new and pragmatic measures for comparing local consistencies.

3 PROPERTIES AND PERSPECTIVES

We present and discuss some metrics for classifying intermediary local consistencies, using Definition 1.

A. Set of solutions The levels of consistency weaker than GAC cannot be totally ordered with respect to the set of solutions of the relaxations of $C(X)$ considered for checking supports. These sets of are not necessarily included one another. However, some properties exist. Given to levels of consistency Φ_1 and Φ_2 applied to a constraint $C(X)$, we say that $\Phi_1 \leq \Phi_2$ when the set of values removed by Φ_1 from the domains of variables in X is included in the set of values removed by Φ_2. By construction, this relation is transitive.

Property 2 *If $Y \subseteq Y'$ then $(\mathcal{C}, Y')$-DC $\leq (\mathcal{C}, Y)$-DC and $(\mathcal{C}, Y')$-BC $\leq (\mathcal{C}, Y)$-BC.*

Proof (Sketch): The same constraints in $\mathcal{C}$ are checked, with larger domains in Y', compared with Y. Definition 1 imposes complete checks (that is, a value v that does not satisfies a constraint $C_j(Y_j) \in \mathcal{C}$ cannot have a $(\mathcal{C}, Y)$-support). The set of supports of each $v \in D(x_i)$ with Y is included into the set obtained with Y'. □

With respect to a comparison related to different sets $\mathcal{C}$, recall that in Definition 1 all $C_j(Y_j) \in \mathcal{C}$ are considered separately. Therefore, the fact that the set of solutions of the constraint network defined by a set $\mathcal{C}$ is strictly included in the set of solutions obtained with another set of constraints is not sufficient, in the general case, to prove an inclusion. We thus consider a Berge-acyclic constraint network [1].

Property 3 *Given two sets of constraints $\mathcal{C} = \{C_1(Y_1), C_2(Y_2), \ldots, C_m(Y_m)\}$ and $\mathcal{C}' = \{C'_1(Y'_1), C'_2(Y'_2), \ldots, C'_{m'}(Y'_{m'})\}$, such that $Y_1 \cup Y_2 \ldots Y_m = Y'_1 \cup Y'_2 \ldots Y'_{m'} \subseteq Y$, If:*
$C_1(Y_1) \wedge \ldots \wedge C_m(Y_m)$ is a Berge-acyclic constraint network,
and $C_1(Y_1) \wedge \ldots \wedge C_m(Y_m) \Rightarrow C'_1(Y'_1) \wedge \ldots \wedge C'_{m'}(Y'_{m'})$,
then $(\mathcal{C}', Y)$-DC $\leq (\mathcal{C}, Y)$-DC and $(\mathcal{C}', Y)$-BC $\leq (\mathcal{C}, Y)$-BC .

Proof (Sketch): The set of solutions of the network $\mathcal{N}_\mathcal{C} = C_1(Y_1) \wedge \ldots \wedge C_m(Y_m)$ defined by $\mathcal{C}$ is included in the one of $\mathcal{C}'$. If $\mathcal{N}_\mathcal{C}$ is Berge-acyclic then any value v supported by all $C_j(Y_j)$'s belongs to a solution of $\mathcal{N}_\mathcal{C}$. From Definition 1, the property holds. □

Example 2 *The Alldiff(X) constraint is satisfied if and only if all the variables in X are pairwise distinct. In this example, the set Y is defined as in Property 1, n is the size of X and d is the maximum domain size. In the literature, we find some filtering algorithms for Alldiff(X):* GAC *($(\{C(X)\}, X)$-DC) in $O(n^{1.5}d)$* [11]. RC *($(\{C(Y)\}, Y)$-DC) in $O(n^2)$* [7]. BC *($(\{C(Y)\}, Y)$-BC) in $O(n.\log(n))$* [10, 8]. *Some constraint toolkits such as* Choco *have a propagator that simulates a clique of binary constraints of difference: $(\mathcal{C}_{\neq}, X)$-DC with $\mathcal{C}_{\neq} = \{x_i \neq x_j, i \in \{1, 2, \ldots n\}, j \in \{1, 2, \ldots n\}, i \neq j\}$). Its has an $O(n^2)$ time complexity per branch of the search tree (it only reacts on variable assignments), which leads to an average time complexity per node in $O(n)$. We have:*
$(\{C(Y)\}, Y)$-BC $\leq (\{C(Y)\}, Y)$-DC $\leq (\{C(X)\}, X)$-DC.
$(\mathcal{C}_{\neq}, X)$-DC $\leq (\{C(X)\}, X)$-DC.
Observe that since $\mathcal{C}_{\neq}$ does not corresponds to a Berge-acyclic constraint network, the second inclusion cannot be reversed. ⊛

It is possible to compose several propagators for $C(X)$. Given two levels of consistency Φ_1 and Φ_2, $\Phi_1 \leq \Phi_1 \circ \Phi_2$ and $\Phi_2 \leq \Phi_1 \circ \Phi_2$.

B. Time complexity Given two levels of consistency Φ_1 and Φ_2 applied to a constraint $C(X)$, we say that $\Phi_1 >_\circlearrowleft \Phi_2$ when the best known algorithm for achieving Φ_1 has a time complexity stricly greater than the best known algorithm for achieving Φ_2. This notion is not formal but very useful in practice to deal with large problems.

Example 3 *Consider the* Alldiff(X) *constraint of Example 2. Like in the example 2, the set Y is defined as in Property 1. We have:*
$(\{C(X)\}, X)$-DC $>_\circlearrowleft (\{C(Y)\}, Y)$-BC
$(\{C(Y)\}, Y)$-DC $>_\circlearrowleft (\{C(Y)\}, Y)$-BC
and $(\{C(Y)\}, Y)$-BC $>_\circlearrowleft (\mathcal{C}_{\neq}, X)$-DC. ⊛

Time complexity is however not always the single criterion in the context of large problems. Some algorithms involve too complex data structures. In some cases, domains of variables must be represented only by their bounds, for memory reasons (trail). Thus, even when a GAC algorithm has a reasonable time complexity, *e.g.*, less than or equal to $O(n.\log(n))$, implementing a BC algorithm can be useful.

C. Solution Counting The sets of solutions of distinct relaxations of $C(X)$ are not necessarily comparable. Conversely, the *number* of solutions of such relaxations can be ordered. Techniques for evaluating the solution counting information of constraints [13] could be an interesting metric to classify, possibly dynamically, several levels of consistencies characterized with Definition 1 for a given constraint.

REFERENCES

[1] C. Berge, 'Graphs and hypergraphs', *Dunod, Paris*, (1970).
[2] C. Bessière, 'Constraint propagation', Research report 06020 (Chapter 3 of the Handbook of Constraint Programming), LIRMM, (2006).
[3] S. Demassey, G. Pesant, and L.-M. Rousseau, 'A cost-regular based hybrid column generation approach', *Constraints*, **11**(4), 315–333, (2006).
[4] M. R. Garey and D. S. Johnson, 'Computers and intractability : A guide to the theory of NP-completeness', *W.H. Freeman and Company*, **ISBN 0-7167-1045-5**, (1979).
[5] E. Hebrard, D. Marx, B. O'Sullivan, and I. Razgon, 'Soft constraints of difference and equality', *J. Artif. Intell. Res. (JAIR)*, **41**, 97–130, (2011).
[6] W.-J. Van Hoeve, G. Pesant, and L.-M. Rousseau, 'On global warming: Flow-based soft global constraints', *Journal of Heuristics*, **12:4-5**, 475–489, (2006).
[7] M. Leconte, 'A bounds-based reduction scheme for constraints of difference', *Proc. Constraint-96 International Workshop on Constraint-Based Reasoning, Key West, Florida*, 19–28, (1996).
[8] A. López-Ortiz, C.-G. Quimper, J. Tromp, and P. van Beek, 'A fast and simple algorithm for bounds consistency of the alldifferent constraint', *Proc. IJCAI*, 245–250, (2003).
[9] T. Petit, J-C. Régin, and C. Bessière, 'Specific filtering algorithms for over constrained problems', *Proc. CP*, 451–463, (2001).
[10] J-F. Puget, 'A fast algorithm for the bound consistency of alldiff constraints.', *Proc. AAAI*, 359–366, (1998).
[11] J-C. Régin, 'A filtering algorithm for constraints of difference in CSPs', *Proc. AAAI*, 362–367, (1994).
[12] Guido Tack, *Constraint Propagation – Models, Techniques, Implementation*, Doctoral dissertation, Saarland University, 2009.
[13] A. Zanarini and G. Pesant, 'Solution counting algorithms for constraint-centered search heuristics', *Proc. CP*, 743–757, (2007).
[14] Y. Zhang and R. H. C. Yap, 'Arc consistency on n-ary monotonic and linear constraints', *Proc. CP*, 470–483, (2000).