ECAI 2014
T. Schaub et al. (Eds.)

doi:10.3233/978-1-61499-419-0-1017

Enforcing Solutions in Constraint Networks

Éric Grégoire and **Jean-Marie Lagniez** and **Bertrand Mazure**[1]

Abstract. A method is proposed to enforce specific solutions in constraint networks. Contrary to previous approaches, it yields a set of constraints to be dropped whose cardinality is minimal.

1 INTRODUCTION

This paper is concerned with the dynamics of constraint networks (in short, CNs), namely with ways to make CNs evolve to meet changing specifications. Especially, the focus is on situations where a CN must deliver some given additional solutions, or where given values to specific variables must ensure global solutions to the CN. Recently, a family of computational approaches have been proposed in [3, 4] that yield subsets of existing constraints to be dropped to this end. We show that these sets are not (cardinality) minimal in the general case, leading to unnecessary additional destructive impact on the CN. Accordingly, we propose an alternative approach that guarantees this minimality.

2 PROBLEM STATEMENT

A *constraint network* $\mathcal{P}$ is a pair $\langle \mathcal{V}, \mathcal{C} \rangle$, where $\mathcal{V}$ is a finite set of *variables* s.t. each variable $x \in \mathcal{V}$ has a finite instantiation domain, denoted $dom(x)$, and where $\mathcal{C}$ is a finite set of *constraints* s.t. each constraint $c \in \mathcal{C}$ involves a subset of variables of $\mathcal{V}$, called *scope* and denoted $scp(c)$. The relation that contains all the combinations of values for the variables of $scp(c)$ that satisfy c is noted $rel(c)$. A *solution* to $\mathcal{P}$ is any assignment of values to all variables of $\mathcal{V}$ that satisfies every constraint of $\mathcal{C}$. A *partial solution* of P w.r.t. $\mathcal{V}' \subseteq \mathcal{V}$ is any solution of $\mathcal{P}$ filtered w.r.t. the scope of the variables of $\mathcal{V}'$. When $\mathcal{P}$ has no solution, it is said *unsatisfiable* or *over-constrained*.

Example 1 *Figure 1 depicts a CN* $\mathcal{P} = \langle \mathcal{V}, \mathcal{C} \rangle$ *where* $\mathcal{V} = \{x_1, x_2, x_3, x_4\}$ *s.t.* $dom(x_i) = \{0, 1\}$ *and* $\mathcal{C} = \{c_1 : x_1 \neq x_3, c_2 : x_2 \geq x_4, c_3 : x_3 < x_4\}$.

Whereas usual CN relaxation methods [5, 2, 1, 6] concern unsatisfiable CNs, [3, 4] have proposed families of incremental relaxation approaches that also concern satisfiable CNs. The goal is to enforce some additional (partial) solutions to the CN. This can be used to ensure that some specific values for critical variables lead to global solutions to the CN. It is also useful in ensuring solutions that would be overridden by pre-existing stronger constraints otherwise. The canonic example is when a constraint $c : x > 2$ is to be "enforced" whereas the CN actually allows values for $x > 4$, only: $x = 3$ and $x = 4$ must be partial solutions of the modified CN. Note that merely adding c in CN will neither yield these additional solutions, nor even lead to unsatisfiability in the general case. Roughly, the methods in [3, 4] consider in an iterative way each additional tuple of values t of $scp(c)$ that is asked to be an additional partial solution. t is inserted within the CN as a new constraint, leading to unsatisfiability.

[1] CRIL, Univ. d'Artois & CNRS, France {gregoire,lagniez,mazure}@cril.fr

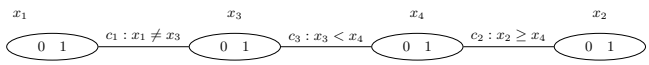

Figure 1. Constraint network of Example 1

According to the extraction of either one maximal (w.r.t. $\subseteq$) satisfiable subsets (MSS), or a cover of minimal unsatisfiable subsets of $\mathcal{C}$ (MUCs), or even one solution to Max-CSP, constraints are dropped to regain satisfiability. This process is iterated until all additional tuples have been considered; c is then inserted within the resulting CN, which yields all required solutions.

This process does not guarantee the extraction of one *minimal* set of constraints to be dropped. Consider Example 1 and assume that all the solutions of an additional constraint $c : (x_1 = 1) \vee (x_2 = 1)$ (that is $rel(c) = \{(1,1), (1,0), (0,1)\}$) must now become partial solutions of the CN, namely that c must be "enforced". $x_1 = x_2 = 1$ is already a partial solution of $\mathcal{P}$. Consider $\langle \mathcal{V}, \mathcal{C} \cup \{c' : (x_1 = 1) \wedge (x_2 = 0)\} \rangle$: it is unsatisfiable. [4] detects one MSS, say $\{c, c_1, c_3\}$ and thus removes c_2 from $\mathcal{P}$. Now, make the same process for c" : $(x_1 = 0) \wedge (x_2 = 1)$; this requires c_3 to be dropped in its turn. Accordingly, both c_2 and c_3 are expelled. c is then inserted within the remaining part of CN. On the contrary, it is easy to see that dropping only c_3 before inserting c would have been sufficient to enforce c.

3 A GLOBAL MAX-CSP-BASED SOLUTION

One direct way to remove a minimal set of constraints of $\mathcal{P}$ would consist in computing all MSSes for each additional considered tuple and then in selecting a minimal set of constraints to drop before c is inserted. This is intractable in the worst case: computing all MSSes (and all MUSes) of a CN is out of reach since the number of MSSes can be exponential in the number of constraints in $\mathcal{P}$ and computing one MSS is in Δ_2^P in the basic Boolean setting.

Instead, we propose an alternative approach, which is best concisely explained through the pseudo-code of Algorithm 1, called MRE for *Minimal Relaxation and Enforcement*. A Boolean flag called (constraint) selector[2] and noted $sel(c_i)$ is created for each constraint c_i of $\mathcal{P}$. A set $\mathcal{C}'$ is created and made of constraints assigning 0 to all those selectors (l. 1). For each tuple $t \in rel(c)$ s.t. $\mathcal{P}$ instantiated to t leads to unsatisfiability (l. 4), a corresponding CN is built by restricting the domains of the variables that are instantiated in t to these instantiation values (l. 4). In the new CN, all variables of $\mathcal{P}$ are given a name that is specific to the tuple t (l. 5) and all constraints of $\mathcal{P}$ get a corresponding new constraint, using these new variables and their domains (l. 6-8) through the use of a $newC(c', r, s)$ function that delivers a novel constraint called c' which satisfies the relation r and s.t. $scp(c) = s$. These constraints are weakened using a disjunct requiring the selector of the corresponding constraint from $\mathcal{P}$

[2] Boolean selectors are of widespread practice in computer science: in Max-CSP-related issues, they were already used in [7] for counting the number of falsified constraints. Their usage here is different.

to be set to 1. Accordingly, every tuple of $rel(c)$ that is incompatible with $\mathcal{P}$ leads to a new CN. These CNs are linked one another through the shared selectors and constraints in $\mathcal{C}'$. This transformation of $\mathcal{P}$ in Example 1 is depicted in Figure 2. The resulting CN is unsatisfiable and one solution to *one* call to a Max-CSP solver (l. 9: noted $MSS \leftarrow MaxCSP(\langle \mathcal{V}', \mathcal{C}' \rangle)$ by abuse of notation) allows us to determine one minimal set of constraints to be removed from $\mathcal{P}$ (l. 11) before c is inserted and the final intended result is obtained.

Algorithm 1: Minimal Relaxation and Enforcement (MRE)

input : $\mathcal{P} = \langle \mathcal{V}, \mathcal{C} \rangle$: a CN; c : a constraint;
output: $\langle \mathcal{V}, \mathcal{C}' \rangle$: minimally modified $\mathcal{P}$ s.t. the partial solutions of $\langle \mathcal{V}, \mathcal{C}' \rangle$ contain $rel(c)$

1 $\langle \mathcal{V}', \mathcal{C}' \rangle \leftarrow \langle \{sel(c_i) | c_i \in \mathcal{C}\}, \{(sel(c_i) = 0) | c_i \in \mathcal{C}\} \rangle$;
2 **foreach** $t \in rel(c)$ **do**
3 **foreach** $x \in scp(c)$ **do** $dom(x) \leftarrow \{t[x]\}$;
   /* $t[x]$is the value of x in t */
4 **if** $\langle \mathcal{V}, \mathcal{C} \rangle$ *is unsatisfiable* **then**
5 **foreach** $x_i \in \mathcal{V}$ **do**
   $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{newVar(x_{it}, dom(x_i))\}$;
6 **foreach** $c_i \in \mathcal{C}$ *s.t.* $scp(c_i) = \{x_{i1}, x_{i2}, \ldots, x_{ik}\}$ **do**
7 $s \leftarrow \{x_{i1t}, x_{i2t}, \ldots, x_{ikt}\}$;
8 $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{(sel(c_i) = 1) \vee newC(c_{it}, rel(c_i), s)\}$
9 $MSS \leftarrow MaxCSP(\langle \mathcal{V}', \mathcal{C}' \rangle)$;
10 **foreach** $c' \in (\mathcal{C}' \setminus MSS)$ **do**
11 $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c_i$ s.t. $sel(c_i) \in \mathcal{C}$ and $c_i \in scp(c')\}$;
12 **return** $\langle \mathcal{V}, \mathcal{C} \cup \{c\} \rangle$;

4 EXPERIMENTAL STUDY

We have conducted extensive experimentations to show both the actual viability of the MRE algorithm and the extent to which it allows a lesser number of constraints to be dropped in practice. We have implemented an incremental method *á la* [3, 4] that computes one Max-CSP per tuple to enforce. We used the SUGAR solver from [8] for Max-CSP computations, available from `http://bach.istc.kobe-u.ac.jp/sugar/`. Experimentations were conducted on Xeon E5-2643 (3.30GHz) processors with 7.6Go RAM on Linux CentOS. We considered instances from the CSP competitions repository `http://cpai.ucc.ie/08/` and `09/` and used the protocol from [4] to generate tuples to be enforced. Table 1 provides a sample of results: the columns indicate the name of the instances $\mathcal{P}$, their extensional or intentional form, maximal arity of their constraints, their numbers of variables and constraints, the number of variables of c generated following [4] ($|scp(c)|$) and the number of tuples from c that do not allow for global solutions to $\mathcal{P}$ ($|rel(c)_\perp|$). The last two columns indicate the number of constraints removed by the incremental method and MRE, each of them followed, between parentheses, by the time in seconds to compute the result. These results illustrate the lighter impact of MRE on $\mathcal{P}$ but the (most often) increased actual computing time by MRE.

5 DISCUSSION AND CONCLUSION

The proposed encoding through the MRE algorithm allows the computation of all MSSes to be avoided. At the same time, it guarantees that one of the smallest possible subsets of constraints is removed, which was ensured by none of the previous approaches. Interestingly, it applies to both satisfiable and unsatisfiable CNs. It is important to note that additional sub-networks are created in the transformed CN only for tuples of $rel(c)$ that are not sub-tuples of global solutions

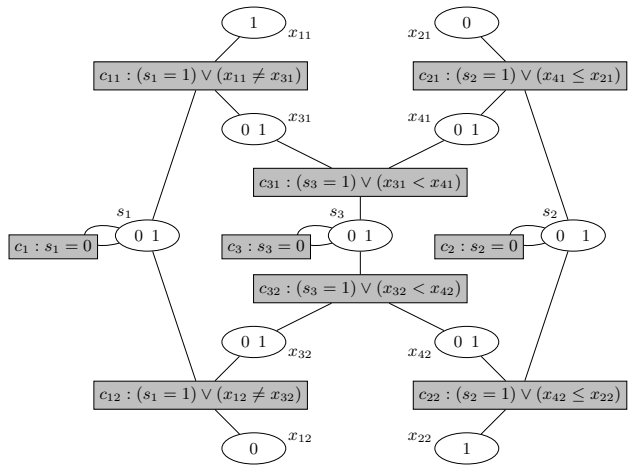

Figure 2. Transposed CN of Example 1

to $\mathcal{P}$. These sub-networks could have been created for every tuple of $rel(c)$ in order to avoid the satisfiability checks to decide of such creations. However, this choice was not selected because the size of the instance for which a solution to Max-CSP is to be found appears to play an even more critical role for the viability of the approach: in MRE, this size is $|rel(c)_\perp|$ times the size of the initial network. In this respect, MRE is only to be recommended when a truly minimal set of constraints to remove must be found and when a limited number of values for some critical variables must belong to global solutions. Also, MRE is best viable when it does not need to be achieved frequently but when it only occurs during "off-line" maintenance operations for which more computational time can be spent.

Name	Type	Arity	$\#\mathcal{V}$	$\#\mathcal{C}$	$\lvert scp(c)\rvert$	$\lvert rel(c)_\perp\rvert$	Incremental	MRE
com'ed-25-10-20-6	ext	2	105	620	2	6	5 (25.38)	3 (98.86)
langford-2-4	ext	2	8	32	2	6	4 (2.13)	3 (0.81)
langford-2-3	int	2	6	18	2	3	3 (1.34)	2 (0.84)
bqwh-15-106-27	ext	2	106	574	2	2	5 (38.56)	4 (212.56)
bqwh-18-141-7	ext	2	141	863	2	2	4 (49.08)	3 (616.82)
bqwh-15-106-69	ext	2	106	608	6	3	6 (24.61)	3 (1721.66)
lemma-23-3	ext	2	23	120	6	11	7 (11.30)	4 (61.21)
qcp-10-67-7	ext	2	100	822	6	4	6 (35.35)	5 (162.81)
series-5	int	3	9	20	6	99	9 (12.64)	8 (41.87)

Table 1. Sample of experimental results

REFERENCES

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis, 'Consistency restoration and explanations in dynamic CSPs - application to configuration', *Artificial Intelligence*, **135**(1-2), 199–234, (2002).
[2] Yan Georget, Philippe Codognet, and Francesca Rossi, 'Constraint retraction in CLP(FD): Formal framework and performance results', *Constraints*, **4**(1), 5–42, (1999).
[3] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure, 'Relax!', in *Proc. of IEEE ICTAI'12*, pp. 146–153, (2012).
[4] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure, 'Preserving partial solutions while relaxing constraint networks', in *Proc. of IJCAI'13*, pp. 552–558, (2013).
[5] Narendra Jussien and Patrice Boizumault, 'Maintien de déduction pour la relaxation de contraintes', in *Actes de JFPLC'96*, pp. 239–254, (1996).
[6] Tomas Nordlander, Ken Brown, and Derek Sleeman, 'Constraint relaxation techniques to aid the reuse of knowledge bases and problem solvers', in *Proc. of the 23rd SGAI Int. Conf. on Innovative Techn. and Appl. of A.I.*, pp. 323–335, (2003).
[7] Jean-Charles Régin, Thierry Petit, Christian Bessière, and Jean-Francois Puget, 'An original constraint based approach for solving over constrained problems', in *Proc. of CP'2000*, pp. 543–548, (2000).
[8] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara, 'Compiling finite linear CSP into SAT', *Constraints*, **14**(2), 254–272, (2009).