Relax!

Éric Grégoire, Jean-Marie Lagniez, Bertrand Mazure

Relax!

Éric Grégoire[1] Jean-Marie Lagniez[2] Bertrand Mazure[1]

CRIL-CNRS UMR 8188 [1]
Université d'Artois
F-62307 Lens Cedex, France
{eric.gregoire,bertrand.mazure}@cril.fr

Institute for Formal Models
and Verification [2]
Johannes Kepler University,
AT-4040 Linz, Austria
Jean-Marie.Lagniez@jku.at

Abstract

This paper is concerned with a form of relaxation of constraint networks. The focus is on situations where additional constraints are intended to extend a non-empty set of preexisting solutions. These constraints require a specific treatment since merely inserting them inside the network would lead to their preemption by more restrictive ones. Several approaches to handle these additional constraints are investigated from conceptual and experimental points of view.

1 Introduction

CSPs (Constraints Satisfaction Problems) have emerged as one of the most fertile research fields of applied Artificial Intelligence (see for example as major conferences, journals and handbook in the domain [15, 4, 17, 19]). A CSP concerns a constraint network that models a problem through a set of constraints linking various variables, each variable exhibiting its own instantiation domain. The CSP might consist in verifying whether the set of constraints exhibits solutions or not. In the positive case, it can consist in computing one solution.

In this paper, the focus is on *relaxing* non-empty sets of solutions of constraint networks when an additional, *more permissive*, constraint C must be taken into account in such a way that C prevails over the more restrictive pre-existing ones.

Assume for example that a complex constraint network determines under which circumstances some specific retirement benefits can be claimed and delivers as solutions all situations where the applicant is the parent of at least 3 children. These solutions are not necessarily explicit in the network but can result from the interaction of several constraints. Now assume that the legislation has changed and that a more permissive rule is adopted allowing these benefits to be claimed by parents of at least two children. If this new constraint is merely inserted as an additional one within the initial network then it will be preempted by the constraints requiring at least three children. Being the parent of two children would still not allow the aforementioned benefits. Indeed, since solving the network will try to satisfy all constraints together, the additional solution translated by the new more permissive rule will be subsumed in the process by the more restrictive ones: having at least two children is satisfied whenever having at least three children is satisfied. Accordingly, we investigate several approaches to handle in an adequate way additional constraints that, at the same time, are more permissive and must prevail.

The paper is organized as follows. In the next section, basic concepts about CSP are recalled. Section 3 briefly presents the MUC (Minimal Unsatisfiable Core) and MSS (Maximal Satisfiable Sub-network) concepts, which are central in this study. Relaxing a constraint network is defined more formally in section 4. In sections 5 and 6, several approaches to address this issue are investigated. In section 7, experimental results are presented about the practical efficiency of the approaches whereas the application of the technique in other circumstances is discussed in section 8. The conclusive section elaborates on promising perspectives for further research.

2 Constraints networks

A *constraint network* $\mathcal{P}$ is a pair $(\mathcal{X}, \mathcal{V})$ where $\mathcal{X}$ a finite set of m constraints over a finite set $\mathcal{V}$ of discrete variables. Each variable $V \in \mathcal{V}$ has its own instantiation domain, denoted $dom(V)$.

Each constraint $C \in \mathcal{X}$ involves a subset of variables of $\mathcal{V}$, called *scope* and noted $var(C)$. A relation, de-

noted $rel(C)$, contains the set of tuples that C allows for the variables in $var(C)$ and is called the support of C. In a dual way, the relation $for(C)$ exhibits the set of tuples of values forbidden by C.

Example 1. *Let the constraint* $(x < y - 1)$ *with* $dom(x) = \{0, 1\}$ *and* $dom(y) = \{1, 2, 3\}$*, we have:* $var(x < y - 1) = \{x, y\}$*,* $rel(x < y - 1) = \{(x = 0, y = 2), (x = 0, y = 3), (x = 1, y = 3)\}$ *and* $for(x < y - 1) = \{(x = 0, y = 1), (x = 1, y = 1), (x = 1, y = 2)\}$*.*

A *solution to a constraint* C is any partial assignment that instantiates all the variables occurring in C in such a way that C is satisfied. A solution to a constraint C is an element of $rel(C)$.

A *solution to* $\mathcal{P}$ is an assignment of a value to each variable such that all the constraints are satisfied. A constraint network is said satisfiable iff it admits at least one solution.

The *Constraint Satisfaction Problem* (CSP) is the NP-hard task of determining whether a given constraint network is satisfiable or not. It can also consist in computing one solution in the positive case.

Example 2. *Let* $\mathcal{P} = (\mathcal{X}, \mathcal{V})$ *with* $\mathcal{V} = \{i, j, k, l, m\}, dom(i) = dom(j) = dom(k) = dom(l) = dom(m) = \{1, 2, 3\}$ *and* $\mathcal{C} = \{(j < k), (k < i), (k \neq l), (j \geq l), (m \neq l + 1), (m + i < 5)\}$*. This constraint network is satisfiable:* $\{i = 3,\ j = 1,\ k = 2,\ l = 1,\ m = 1\}$ *is an assignment that satisfies all constraints of* $\mathcal{X}$*.*

Let $\mathcal{P}' = (\mathcal{X} \cup \{(i < j)\}, \mathcal{V})$*. This constraint network has no solution; it is unsatisfiable.*

3 MUC and MSS

Although the focus is on satisfiable constraint networks, the following concept related to unsatisfiability will be central in this paper.

Any unsatisfiable constraint network $\mathcal{P} = (\mathcal{X}, \mathcal{V})$ involves at least one MUC (*Minimal Unsatisfiable Core*), also called *core*. A MUC is a subset of constraints of $\mathcal{X}$ that, at the same time, is unsatisfiable and that is such that each proper subset is satisfiable. A MUC is thus a smallest subset of constraints that cannot be accommodated together.

Example 3. $\mathcal{P}'$ *defined in the previous example admits only one MUC, namely* $\{(j < k), (k < i), (i < j)\}$*.*

In a dual way, a *Maximal Satisfiable Sub-network* (MSS) of $\mathcal{P} = (\mathcal{X}, \mathcal{V})$ is defined as any satisfiable $\mathcal{P}' = (\mathcal{X}', \mathcal{V})$ such that $\forall C \in \mathcal{X} \setminus \mathcal{X}'$, the constraint network $(\mathcal{X}' \cup \{C\}, \mathcal{V})$ is unsatisfiable.

Example 4. *In the Example 2,* $\mathcal{P}$ *is a maximal satisfiable sub-network of* $\mathcal{P}'$*.*

Checking whether a constraint belongs to a MUC or not is in Σ_2^P [5]. In the worst case, the number of MUCs can be exponential in the number m of constraints (it is in $\mathcal{O}(C_m^{m/2})$). Since MUCs can intersect, the concept of *cover of MUCs* has been introduced to alleviate the possible high number of MUCs, at least to some extent [7]. Despite those bad worst-case complexity results, in many real-life situations, the number of MUCs can remain of a manageable size and both MUCs and MSSes can be computed in realistic time (see for example approaches in [7] and [8]).

One specific well-studied problem about MSS is called WCSP (as Weighted CSP) which consists in delivering one MSS of a constraint network following a priority scale between constraints. Each constraint is given a numerical value and constraints with the higher values are preferred candidates for belonging to the resulting MSS. More on MUC and MSS can be found in [3, 9, 13, 11, 18, 12, 10, 7, 8].

4 Relaxing satisfiable networks

From now on, we assume that the constraint network $\mathcal{P} = (\mathcal{X}, \mathcal{V})$ is satisfiable and that a constraint C over variables from $\mathcal{V}$ is the additional more permissive one that must prevail.

First, let us define what *more permissive* and *prevail* mean in this study. Remember that, on the one hand, a solution to a constraint C is any partial assignment that instantiates all the variables occurring in C in such a way that C is satisfied. On the other hand, a solution to $\mathcal{P}$ is any assignment of all the variables of $\mathcal{V}$ that satisfies all constraints in $\mathcal{X}$.

Definition 1. *A constraint* C *is more permissive than a constraint network* $\mathcal{P}$ *iff*
1. there exists at least one solution to C *that cannot be extended into an a solution to* $\mathcal{P}$*, and*
2. any solution to $\mathcal{P}$ *restricted to the scope of* C *is a solution to* C*.*

Definition 2. *A constraint* C *prevails in* $\mathcal{P}$ *iff all solutions to* C *can be extended into solutions to* $\mathcal{P}$*.*

Thus, relaxing $\mathcal{P}$ by a more permissive constraint C must lead to a new network $\mathcal{P}'$ such that any solution to C has been extended into a solution to $\mathcal{P}'$.

Definition 3. *A constraint network* $\mathcal{P}' = (\mathcal{X}', \mathcal{V})$ *is a relaxed network of* $\mathcal{P} = (\mathcal{X}, \mathcal{V})$ *by* C *iff any solution to* C *can be extended into a solution to* $\mathcal{P}'$*.*

Note that forcing all solutions of C to be adopted might lead to the existence of additional solutions related to variables outside the scope of C through a domino effect within $\mathcal{P}$. Hence, $\mathcal{P}'$ can exhibit new solutions about variables not occurring in C in addition to those provided by $\mathcal{P}$. For a similar reason, we cannot require in the general case that any solution of $\mathcal{P}$ is a solution of $\mathcal{P}'$.

5 MUC-based solutions

Several constraints in $\mathcal{P}$ can prevent solutions to C from prevailing in $\mathcal{P}$. A first approach provides the user will a full-fledged explanation of the reasons for this, under the form of the minimal sets of constraints that cause the problem. It is based on MUC-finding algorithms and the main idea is intuitively as follows.

Whenever a single element of $rel(C)$ is introduced as a new constraint in $\mathcal{P}$ and when this leads to an unsatisfiable network $\mathcal{P}'$, this means that $\mathcal{P}$ does not authorize $rel(C)$. Accordingly, we compute and exhibit MUCs in $\mathcal{P}'$. A least one constraint per MUC is expelled (or relaxed) from $\mathcal{P}$. This is iterated until the network $\mathcal{P}'$ becomes satisfiable. The whole process is then iterated for all elements of $rel(C)$. Thereafter, the constraint C is just safely added since we are sure that all its solutions can be now accommodated. When we know which elements of $rel(C)$ cannot be extended into solutions to $\mathcal{P}$, it is sufficient for the procedure to consider these elements, only.

Accordingly, any detected MUC provides a minimal set of constraints preventing an additional solution given by C from prevailing in $\mathcal{P}$. Algorithm relaxMUC describes the skeleton of an iterative approach finding out and "breaking" in an automatic way MUCs iteratively, until no MUC remains. Note that the depicted algorithm does not return the set of detected MUCs but expels one constraint per MUC and simply delivers the final resulting relaxed constraint network. Note that since MUCs can intersect, a cover of MUCs can be delivered and amended as an alternative approach.

6 MSS-based solutions

When not all MUCs must be exhibited to the user. the dual concept of Maximal Satisfiable Sub-networks (MSS) can also be used. Note that MUCs and MSSes can be easily computed one from the other, or be computed separely using their own specific algorithmic paradigms [2].

Definition 4. *A constraint network $\mathcal{P}' = (\mathcal{X}' \cup \{C\})$ is a MSS-relaxed network of $\mathcal{P} = (\mathcal{X}, \mathcal{V})$ by C iff $\mathcal{X}' \subset \mathcal{X}$ and $(\mathcal{X}' \cup \{t\}, \mathcal{V})$ is satisfiable for every t in $rel(C)$.*

Function `relax-MUC(`$(\mathcal{X}, \mathcal{V})$, C`)`: $(\mathcal{X}', \mathcal{V})$

input : $P = (\mathcal{X}, \mathcal{V})$: a constraint network,
C: the new constraint
output: $(\mathcal{X}', \mathcal{V})$: a relaxed constraint network of P by C

$\mathcal{X}' \leftarrow \mathcal{X}$;
foreach $t \in rel(C)$ **do**
  $C_t \leftarrow \{t\}$ with $dom(C_t) = dom(C)$;
  while $(\mathcal{X}' \cup \{C_t\}, \mathcal{V})$ *is unsat* **do**
    $(\mathcal{X}'', \mathcal{V}'') \leftarrow$ `MUC(`$(\mathcal{X}' \cup \{C_t\}), \mathcal{V})$;
    select one constraint C' in $\mathcal{X}''$ s.t. $C' \neq C_t$;
    $\mathcal{X}' \leftarrow \mathcal{X}' \setminus \{C'\}$;
  end
end
$\mathcal{X}' \leftarrow \mathcal{X}' \cup \{C\}$;
return $(\mathcal{X}', \mathcal{V})$;

Definition 5. *A MSS-relaxed constraint network $\mathcal{P}' = (\mathcal{X}' \cup \{C\})$ of $\mathcal{P}$ by C is maximal iff there does not exist any other MSS-relaxed network $\mathcal{P}$" $= (\mathcal{X}$" $\cup \{C\})$ of $\mathcal{P}$ by C such that $\mathcal{X}' \subset \mathcal{X}$".*

Proposition 1. *Every MSS-relaxed network of $\mathcal{P}$ by C is a relaxed network of $\mathcal{P}$ by C.*

Proposition 2. *Not all MSS-relaxed networks are maximal in the general case.*

Indeed, a MSS-relaxed network can depend both on the order according to which the solutions of $rel(X)$ are considered and on the actual selection of constraints that are dropped. For example, dropping a constraint that belongs to the intersection of several MUCs (each of them related to a different solution of $rel(X)$) conducts larger MSS-related networks to be generated.

The relax-MSS algorithm describes such a function computing *one* MSS-relaxed network of $\mathcal{P}$ by C. Note that the computation of one MSS can be performed though WCSP.

7 Relax in other circumstances, too

Interestingly, we can apply this relaxation technique when the new constraint C is not more permissive. In this case, the technique simply leads to add C into $\mathcal{P}$.

Also, it can be applied to unsatisfiable networks as well. In this case, it does not only restore consistency but also enforces C as a more permissive constraint. Note that in such a case the computed MUCs do not necessarily contain the enforced constraint made of one

Function `relax-MSS(`$(\mathcal{X},\mathcal{V})$`,` C`)`: $(\mathcal{X}', \mathcal{V})$

input : $P = (\mathcal{X}, \mathcal{V})$: a constraint network,
C: the new constraint
output: $(\mathcal{X}', \mathcal{V})$: a relaxed constraint network of P by C

$\mathcal{X}' \leftarrow \mathcal{X}$;
foreach $t \in rel(C)$ **do**
  $C_t \leftarrow \{t\}$ with $dom(C_t) = dom(C)$;
  $\mathcal{X}' \leftarrow$ `MSS(`$(\mathcal{X}' \cup \{C_t\}, \mathcal{V})$`)` ;
  $\mathcal{X}' \leftarrow \mathcal{X}' \setminus \{C_t\}$;
end
$\mathcal{X}' \leftarrow \mathcal{X}' \cup \{C\}$;
return $(\mathcal{X}', \mathcal{V})$;

value of $rel(C)$. Similarly, constraints not included in a MSS need not belong to a MUC containing this additional constraint.

Also note that WCSP, which can be used to compute MSSes, can take into account an a priori preference ordering between constraints through numerical coefficients. Accordingly, it allows to respect a preference among constraints in order to designate the ones to be dropped.

8 Experimental results

All experimental results presented in this section have been obtained on a Quad-core Intel XEON X5550 with 32Gb of memory. The CPU time was limited to 7200 seconds. We have implemented our algorithms on top of the CSP-solving platform Abscon `http://www.cril.univ-artois.fr/~lecoutre/software.html`

We have run our algorithms on more than 600 instances taken from the satisfiable benchmarks of the last CSP competitions `http://cpai.ucc.ie/08/` et `http://cpai.ucc.ie/09/`. In the Tables we present results for 53 instances from the 204 ones that were not time-out for neither of our approaches (the full tables are available from `https://www.dropbox.com/sh/q579823oyfr0g9z/66RxbrNYCC`). We generated the additional constraints in the following random way so that they were necessarily more permissive ones. One constraint of the instance was selected in an random fashion; its allowed and forbidden tuples computed. Then, a random number of forbidden tuples changed of status and became allowed. The same data have been submitted to both relax-MUC and relax-MSS.

The columns from Tables 1 and 2 provide the following information.

- name: instance name
- $\#\mathcal{V}$: number of variables of the instance
- $\#\mathcal{C}$: number of constraints of the instance
- $|rel(C)|$: size of the support of the additional constraint
- $\#ns$: number of tuples of $rel(C)$ to be relaxed, i.e., the number of tuples of $rel(C)$ that are forbidden by other constraints of the instance (this information is not an input of the algorithms)
- $|var(C)|$: size of scope of the additional constraint
- $\#rc$: number of expelled constraints
- $\#rc_{avg}$: average number of constraints per tuple to be relaxed
- T_1: average CPU time to relax one tuple
- T_2: total CPU time to relax all tuples
- T_3: total CPU time for the method

The Tables illustrate the practical feasibility of the approaches for *complete* methods computing MUCs and MSSes. Let us stress on some lessons from these experimentations.

Relax-MSS allows to compute a minimal number of constraints to be expelled so that the instance is relaxed with respect to the additional constraint C. Consequently, the number of constraints expelled by this method is always lower or equal to the number of constraints expelled by relax-MUC. However, relax-MUC provides results that exhibit a number of constraints that is often close to this minimal number (although exceptions exist: see for example instance e64-b).

On the other hand, this minimization is computationally costly: in this respect, it is not a surprise to see that relax-MUC is generally more time efficient (although exceptions exist: see for example instance e64-b again). Relax-MUC allowed 72 additional instances to be relaxed. Technically, this is due to the fact that computing a MSS delays branches-pruning decisions in the search trees (the descent is not stopped when a constraint is falsified but only when a specific number of constraints are falsified). In this respect, one possible improvement w.r.t. the MSS-computation would consist in running a local search procedure to find out one "good" first boundary.

The average computing time for relax-MUC is 780 secs when only the successfully relaxed instances are taken into account whereas it is 691 secs for relax-MSS when only the solved instances by this method were

considered. This apparently better result in favor of relax-MSS is due to the fact this method mainly solved the simple instances and did not solve a large number of instances where a large number of constraints need be expelled.

Conclusion

Some past research efforts in the CSP research community have already concerned the problem of constraint relaxation in the framework of unsatisfiable constraint networks[14, 6, 1, 16]. Quite surprisingly, to the best of our knowledge, this issue has not been investigated so far in the context of satisfiable networks that must accommodate an additional constraint whose solutions must not be lost. This study is a first contribution to fill this gap. We believe that more research efforts should be devoted about the dynamics of networks. In particular, the problem investigated in this paper is even more crucial and ubiquitous when several networks need to be merged.

Several paths for further extensions of this study can be envisioned. First, instead of expelling constraints, we may want to amend constraints that forbid additional solutions to C from being extendable into solutions to $\mathcal{P}$. Also, instead of considering computationally costly complete techniques computing MUCs and MSSes we might want to build and experiment approximate methods that attempt to provide, for example, a superset of a MUC that would normally not be significantly larger than the MUC itself.

This study also sheds light on the necessity to distinguish between several forms of constraints in the following sense. Usually, all constraints are restrictive in the sense that each of them can restrict the space of possible solutions provided by other constraints. As illustrated in this paper, we can also need to represent *permissive* constraints instead of restrictive ones. Permissive constraints appear as hard constraints that must be satisfied in all circumstances. Moreover, a permissive constraint translates solutions that cannot be lost due to other (restrictive) constraints. Considering permissive constraints in the traditional algorithms for solving CSPs opens new exciting and very substantial perspectives.

We believe that these are promising paths for further research.

Acknowledgements

This work has been supported in part by the *Conseil Régional du Nord/Pas-de-Calais* , the EC through a FEDER grant and FWF, NFN Grant S11408-N23 (RiSE). We thank C. Lecoutre for making Abscon available and for his help in programming it.

References

[1] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic csps—application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.

[2] J. Bailey and P. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proceedings of PADL'05*, pages 174–186, 2005.

[3] R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In M. Kaufmann, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, volume 1, pages 276–281, 1993.

[4] N. Beldiceanu, N. Jussien, and E. Pinson, editors. *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings*, volume 7298 of *Lecture Notes in Computer Science*. Springer, 2012.

[5] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992.

[6] Y. Georget, P. Codognet, and F. Rossi. Constraint retraction in clp(fd): Formal framework and performance results. *Constraints*, 4(1):5–42, 1999.

[7] É. Grégoire, B. Mazure, and C. Piette. Must: Provide a finer-grained explanation of unsatisfiability. In C. Bessière, editor, *13th International Conference on Principles and Practice of Constraint Programming(CP'07)*, pages 317–331, Providence (USA), sep 2007. LNCS 4741.

[8] É. Grégoire, B. Mazure, and C. Piette. On finding minimally unsatisfiable cores of csps. *International Journal on Artificial Intelligence Tools (IJAIT)*, 17(4):745 – 763, aug 2008.

[9] B. Han and S. Lee. Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 29, pages 281–286. IEEE, 1999.

[10] F. Hemery, C. Lecoutre, L. Saïs, and F. Boussemart. Extracting mucs from constraint networks. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 113–117, 2006.

[11] U. Junker. Quickxplain : Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, 2001.

[12] U. Junker. Quickxplain : Preferred explanations and relaxations for over-constrained problems. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 167–172, 2004.

Instance			New constraint			MUC		Time in seconds		
name	$\#\mathcal{V}$	$\#\mathcal{X}$	$\|rel(C)\|$	$\#ns$	$\|var(C)\|$	$\#rc$	$\#rc_{avg}$	T_1	T_2	T_3
9symml	651	1648	22	2	6	–	–	–	–	–
C17	15	20	4	1	3	1	1.00	.58	.585	2
aim-100-1-6-4	200	260	8	2	4	2	11.00	3.86	7.720	14
aim-100-2-0-3	200	300	8	2	4	2	54.50	35.06	70.124	77
aim-100-3-4-4	200	440	8	2	4	10	184.20	244.22	2442.238	2451
aim-200-1-6-4	400	520	4	2	3	2	60.50	57.02	114.042	144
aim-200-2-0-4	400	600	8	2	4	2	33.00	52.07	104.153	219
aim-50-1-6-4	100	130	4	2	3	2	52.50	29.50	59.013	62
aim-50-2-0-4	100	150	8	2	4	2	11.00	3.73	7.475	11
aim-50-3-4-1	100	220	8	3	4	6	73.00	33.15	198.917	204
aim-50-6-0-4	100	350	8	3	4	14	57.78	24.14	338.075	345
b1	24	45	8	1	4	1	1.00	.69	.694	4
blast-floppy1-4	237	196	8	2	4	67	1.00	.93	62.671	123
blast-floppy1-6	719	592	1	1	4	–	–	–	–	–
c8	239	523	4	1	3	1	1.00	1.68	1.684	5
cc	133	219	2	0	3	0	0.00	0.00	0	1
cm42a	99	185	8	1	4	1	1.00	.99	.997	5
cmb	304	670	4	1	3	1	1.00	2.57	2.571	7
e64-b	607	1022	22	1	8	807	1.00	1.87	1510.178	1696
elf-rf6	67	131	4	4	3	70	2.07	.79	55.999	72
enigma	100	42	5	1	11	2	1.00	1.99	3.995	19
fpga-10-9	135	118	22	1	11	1	18.00	49.47	49.477	77
g-100x100	10000	10000	22	1	6	1	1.00	20.49	20.494	123
graceful–K3-P2	15	60	48	5	3	25	41.04	6.98	174.590	197
graceful–K4-P2	24	164	169	4	3	41	131.36	86.92	3563.927	3670
haystacks-05	25	54	18	2	3	2	21.00	6.66	13.321	21
ii-32e2	534	3013	4	1	3	1	1.00	7.01	7.010	241
lseu	89	28	22	1	8	1	1.00	1.15	1.151	14
mod008	319	6	22	0	197	0	0.00	0.00	0	48
mps-p0282	282	221	4	1	3	1	1.00	3.62	3.622	15
os-taillard-5-100-9	25	100	1531	4	3	8	44.87	374.56	2996.515	5546
os-taillard-5-105-9	25	100	1678	3	3	3	2.66	6.77	20.335	3135
os-taillard-5-95-9	25	100	1391	9	3	16	31.06	141.30	2260.807	4245
p0033	33	15	8	1	4	1	1.00	.70	.705	4
p0201	195	133	14	3	10	–	–	–	–	–
p0282	282	221	4	1	3	1	1.00	3.28	3.286	15
par-8-1-c	128	318	8	4	4	8	74.87	30.96	247.751	254
par-8-5-c	150	373	8	4	4	7	74.71	32.76	229.353	235
pk1	86	60	4	1	3	1	1.00	1.24	1.247	4
primes-15-20-3-5	100	20	450	2	5	3	1.00	3.59	10.797	595
primes-15-40-3-1	100	40	417	1	5	7	2.00	1.50	10.526	342
primes-15-60-2-1	100	60	331	2	4	5	1.00	.74	3.705	156
primes-15-80-2-1	100	80	295	2	3	8	1.87	.95	7.619	147
radar-10-10-4.5-0.95-100	435	500	4	1	3	1	1.00	1.36	1.364	4
radar-10-20-4.5-0.95-100	934	1062	4	1	3	–	–	–	–	–
radar-8-30-3-0-14	180	64	41	4	13	6	1.00	272.82	1636.930	2106
ruler-34-8-a3	36	434	365	7	3	195	24.77	6.30	1228.791	1509
sao2-b	372	765	22	1	10	–	–	–	–	–
scen2	200	1235	483	1	3	1	1.00	2.77	2.772	398
scen6-w1-f2	200	319	345	4	3	5	7.00	2.77	13.853	194
scen7-w1-f5	400	660	270	4	3	9	6.33	2.81	25.331	206
stein27	27	118	13	5	28	52	1.00	.55	29.049	46
stein45	45	331	8	1	4	1	1.00	1.00	1.005	5

Table 1. Results for relax-MUC

Instance			New constraint			MSS		Time in seconds		
name	$\#\mathcal{V}$	$\#\mathcal{X}$	$\|rel(C)\|$	$\#ns$	$\|var(C)\|$	$\#rc$	$\#rc_{avg}$	T_1	T_2	T_3
9symml	651	1648	22	2	6	3	1.50	16.52	33.050	63
C17	15	20	4	1	3	1	1.00	.14	.141	2
aim-100-1-6-4	200	260	8	2	4	2	1.00	26.23	52.473	57
aim-100-2-0-3	200	300	8	2	4	2	1.00	112.30	224.605	232
aim-100-3-4-4	200	440	8	2	4	–	–	–	–	–
aim-200-1-6-4	400	520	4	2	3	–	–	–	–	–
aim-200-2-0-4	400	600	8	2	4	–	–	–	–	–
aim-50-1-6-4	100	130	4	2	3	2	1.00	.56	1.124	3
aim-50-2-0-4	100	150	8	2	4	2	1.00	1.06	2.121	6
aim-50-3-4-1	100	220	8	2	4	2	1.00	40.62	81.240	85
aim-50-6-0-4	100	350	8	3	4	–	–	–	–	–
b1	24	45	8	1	4	1	1.00	.16	.162	3
blast-floppy1-4	237	196	8	4	4	4	1.00	1.39	5.586	38
blast-floppy1-6	719	592	1	1	4	1	1.00	1704.67	1704.67	1809
c8	239	523	4	1	3	1	1.00	1.74	1.749	5
cc	133	219	2	0	3	–	–	–	–	–
cm42a	99	185	8	1	4	1	1.00	.64	.644	4
cmb	304	670	4	1	3	1	1.00	1.43	1.433	6
e64-b	607	1022	22	1	8	1	1.00	9.11	9.119	41
elf-rf6	67	131	4	4	3	–	–	–	–	–
enigma	100	42	5	1	11	1	1.00	23.94	23.941	38
fpga-10-9	135	118	22	1	11	–	–	–	–	–
g-100x100	10000	10000	22	1	6	1	1.00	92.72	92.723	186
graceful–K3-P2	15	60	48	4	3	4	1.00	.21	.876	17
graceful–K4-P2	24	164	169	4	3	–	–	–	–	–
haystacks-05	25	54	18	2	3	2	1.00	1.14	2.297	9
ii-32e2	534	3013	4	1	3	1	1.00	562.45	562.453	793
lseu	89	28	22	1	8	1	1.00	.59	.593	13
mod008	319	6	22	0	197	–	–	–	–	–
mps-p0282	282	221	4	1	3	–	–	–	–	–
os-taillard-5-100-9	25	100	1531	5	3	6	1.20	28.51	142.590	4161
os-taillard-5-105-9	25	100	1678	3	3	3	1.00	1.75	5.265	4614
os-taillard-5-95-9	25	100	1391	9	3	–	–	–	–	–
p0033	33	15	8	1	4	1	1.00	.23	.23	3
p0201	195	133	14	3	10	4	1.33	5.17	15.519	25
p0282	282	221	4	1	3	–	–	–	–	–
par-8-1-c	128	318	8	4	4	–	–	–	–	–
par-8-5-c	150	373	8	5	4	5	1.00	5.45	27.264	32
pk1	86	60	4	1	3	1	1.00	.63	.634	3
primes-15-20-3-5	100	20	450	2	5	–	–	–	–	–
primes-15-40-3-1	100	40	417	1	5	–	–	–	–	–
primes-15-60-2-1	100	60	331	2	4	5	2.50	43.52	87.055	240
primes-15-80-2-1	100	80	295	1	3	3	3.00	14.12	14.122	155
radar-10-10-4.5-0.95-100	435	500	4	1	3	1	1.00	27.89	27.897	30
radar-10-20-4.5-0.95-100	934	1062	4	1	3	1	1.00	2082.93	2082.935	2088
radar-8-30-3-0-14	180	64	41	4	13	–	–	–	–	–
ruler-34-8-a3	36	434	365	7	3	–	–	–	–	–
sao2-b	372	765	22	1	10	1	1.00	3.54	3.543	34
scen2	200	1235	483	1	3	–	–	–	–	–
scen6-w1-f2	200	319	345	6	3	7	1.16	5.99	35.984	210
scen7-w1-f5	400	660	270	4	3	–	–	–	–	–
stein27	27	118	13	5	28	52	10.40	.17	.873	9
stein45	45	331	8	1	4	1	1.00	1022.56	1022.563	1026

Table 2. Results for relax-MSS

[13] N. Jussien and V. Barichard. The palm system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP'00*, pages 118–133, 2000.

[14] N. Jussien and P. Boizumault. Maintien de déduction pour la relaxation de contraintes. In J.-L. Imbert, editor, *JFPLC*, pages 239–254. Hermes, 1996.

[15] J. H.-M. Lee, editor. *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*. Springer, 2011.

[16] T. Nordlander, K. Brown, and D. Sleeman. Constraint relaxation techniques to aid the reuse of knowledge bases and problem solvers. In *Proceedings of the Twenty-third SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 323–335, 2003.

[17] G. Pesant, editor. *Constraints (Journal)*, volume 17. Springer, 2012.

[18] T. Petit, C. Bessière, and J. Régin. A general conflict-set based framework for partial constraint satisfaction. In *Proceedings of SOFT'03: Workshop on Soft Constraints held with CP'03*, 2003.

[19] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier Science, 2006.