Domain k-Wise Consistency Made as Simple as Generalized Arc Consistency

Jean-Baptiste Mairy[1], Yves Deville[1], and Christophe Lecoutre[2]

[1] ICTEAM, Université catholique de Louvain, Belgium
{jean-baptiste.mairy,yves.deville}@uclouvain.be
[2] CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France
lecoutre@cril.fr

Abstract. In Constraint Programming (CP), Generalized Arc Consistency (GAC) is the central property used for making inferences when solving Constraint Satisfaction Problems (CSPs). Developing simple and practical filtering algorithms based on consistencies stronger than GAC is a challenge for the CP community. In this paper, we propose to combine k-Wise Consistency (kWC) with GAC, where kWC states that every tuple in a constraint can be extended to every set of $k-1$ additional constraints. Our contribution is as follows. First, we derive a domain-filtering consistency, called Domain k-Wise Consistency (DkWC), from the combination of kWC and GAC. Roughly speaking, this property corresponds to the pruning of values of GAC, when enforced on a CSP previously made kWC. Second, we propose a procedure to enforce DkWC, relying on an encoding of kWC to generate a modified CSP called k-interleaved CSP. Formally, we prove that enforcing GAC on the k-interleaved CSP corresponds to enforcing DkWC on the initial CSP. Consequently, we show that the strong DkWC can be enforced very easily in constraint solvers since the k-interleaved CSP is rather immediate to generate and only existing GAC propagators are required: in a nutshell, DkWC is made as simple and practical as GAC. Our experimental results show the benefits of our approach on a variety of benchmarks.

1 Introduction

Constraint Propagation is a key concept to Constraint Programming (CP). Interleaved with (backtrack) search decisions such as classical variable assignments, it typically discards many useless substantial parts of the search space of Constraint Satisfaction Problems (CSPs) by filtering out inconsistent values and/or tuples. Different levels of filtering exist, and usually they can be characterized by properties, called consistencies, of constraints or constraint networks. The central consistency in CP is Generalized Arc Consistency (GAC) [13], also called Domain Consistency (DC): it is the highest filtering level of variable domains when constraints are considered one at a time. Consistencies weaker than GAC are cheaper to enforce but they lose ground progressively, at least for binary and table constraints[1], as they reduce (far) less the search space of CSPs. On the other hand, consistencies stronger than GAC are more and more studied, and often tested on difficult problem instances, where the cost of enforcing them can be counterbalanced by their large inference capabilities. However, such strong consistencies

[1] For example, this is the case of the partial form of GAC maintained in Forward Checking.

H. Simonis (Ed.): CPAIOR 2014, LNCS 8451, pp. 235–250, 2014.

need to reason with several constraints simultaneously, which makes the development of filtering algorithms complex, especially for integration into existing CP solvers.

Most of the consistencies can also be classified in two categories: domain-based (or domain-filtering) consistencies and constraint-based ones. Domain-based consistencies identify inconsistent values that can be removed from the domains of variables whereas constraint-based ones identify inconsistent tuples in the constraints, for which the removal is not always a possible option in constraint solvers. Different examples of such consistencies can be found in [1,4,3,8,10,17]. Interestingly enough, combining a constraint-based consistency with a domain-based one such as GAC allows the pruning achieved in term of tuples to further prune variable domains. This is what we propose in this paper by combining k-Wise Consistency (kWC) with GAC. The constraint-based kWC states that every tuple in the scope of a constraint can be extended to every set of $k-1$ other constraints. Note that kWC and GAC have already been theoretically combined in [7,6]. They have also been practically combined under weaker forms in [3,17] and a practical sophisticated algorithm for the full combination has been proposed in [8].

Our contribution in this paper is two-fold. First, we derive a domain-filtering consistency, called Domain k-Wise Consistency (DkWC), from the combination of kWC and GAC. Roughly speaking, this property corresponds to kWC and GAC combined, but where only the outcome in term of pruned values is considered. Second, we propose a simple and practical filtering procedure to enforce DkWC on a given initial CSP containing table constraints, relying on an encoding of kWC to generate a modified CSP, called k-interleaved CSP. This encoding allows invalidating k-wise inconsistent tuples by means of dual variables, without effectively removing them from constraint scopes, as pure k-wise consistency would do. Formally, we prove that enforcing GAC on the k-interleaved CSP corresponds to enforcing DkWC on the initial CSP. Consequently, we show that the strong DkWC can be enforced very easily in constraint solvers since the k-interleaved CSP is rather immediate to generate (and only once) and only existing GAC propagators are required: in a nutshell, DkWC is made as simple and practical as GAC. We also define two weaker variants of our filtering procedure, that can be used when the problems are too large for the full filtering. Our experimental results show the benefits of our approach on a variety of benchmarks including table constraints.

2 Background

A Constraint Satisfaction Problem (CSP) $P = (X, D, C)$ is composed of an ordered set of n *variables* $X = \{x_1, \ldots, x_n\}$, a set of *domains* $D = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible *values* for variable x_i, and a set of e *constraints* $C = \{c_1, \ldots, c_e\}$, where each constraint c_j restricts the possible combinations of values, called *allowed tuples*, on a subset of variables of X ; this subset is called the *scope* of c_j and denoted by $scp(c_j)$. Because variable domains may evolve (be reduced), $D(x)$ is referred to as the *current domain* of x, which is a subset of the *initial domain* of x denoted by $D^{init}(x)$. If $Y \subseteq X$ then $D[Y]$ is the restriction of D to variables in Y. For any value refutation $x \neq a$, $P|_{x \neq a}$ denotes the CSP P where the value a is removed from $D(x)$, and for any set of value refutations Δ, $P|_{\Delta}$ is defined similarly. The *arity*

of a constraint c is $|scp(c)|$, i.e., the number of variables involved in c. In this paper, we shall refer to (positive) table constraints where a *table constraint* c is a constraint given in extension by its explicit list $rel(c)$ of allowed tuples. A table constraint c holds iff $(x_1, \ldots, x_r) \in rel(c)$, where $scp(c) = \{x_1, \ldots, x_r\}$. The size of a constraint c corresponds to its number of allowed tuples and will be denoted by $|rel(c)|$. If τ is a r-tuple $(a_1, \ldots, a_r)$ then $\tau \odot b$ denotes the $r+1$ tuple $(a_1, \ldots, a_r, b)$.

We assume an implicit total ordering on X, and given $Y = \{y_1, \ldots, y_k\} \subseteq X$, the set of tuples in $D(y_1) \times \ldots \times D(y_k)$ will be denoted by $D(Y)$, and the set of tuples in $D^{init}(y_1) \times \ldots \times D^{init}(y_k)$ will be denoted by $D^{init}(Y)$. A constraint c is satisfied by a tuple $\tau \in D^{init}(scp(c))$ iff τ is *allowed* by c ; the test evaluating to true iff τ is allowed by c is denoted by $c(\tau)$, or equivalently by $\tau \in c$. We shall use the term literal to refer to a variable value pair ; a literal of a constraint c is a pair (x, a) where $x \in scp(c)$ and $a \in D(x)$. An assignment of a set of variables $Y = \{y_1, \ldots, y_k\}$ is a set of literals $\{(y_1, v_1), \ldots, (y_k, v_k)\}$ with $(v_1, \ldots, v_k) \in D^{init}(Y)$; it is a *valid* assignment if $(v_1, \ldots, v_k) \in D(Y)$. Note that any tuple τ in $D^{init}(Y)$ can be seen as an assignment of Y. Actually, for simplicity, we shall use both concepts (assignments and tuples) interchangeably, with the same notations τ, τ', ... For any tuple or assignment τ, the ith value in τ, associated with the variable y_i, will be denoted by $\tau[y_i]$. A solution of P is a valid assignment of X that satisfies all constraints of P. Two CSPs P and P' are equivalent iff they have the same set of solutions.

Now, let us turn to consistencies. On the one hand, Generalized Arc Consistency (GAC), also called Domain Consistency (DC) in the literature, is a well-known domain-filtering consistency. To define it, we need first to introduce the notion of support as follows: a *support* on a constraint c is a tuple $\tau \in D(scp(c))$ such that $c(\tau)$, and a support (on c) for a literal (x, a) of c is a support τ on c such that $\tau[x] = a$. Note that supports are *valid* tuples, meaning that involved values are necessarily present in the current domains.

Definition 1. (GAC) *A constraint c is* generalized arc consistent *(GAC) iff there exists at least one support for each literal of c. A CSP P is GAC iff every constraint of P is GAC.*

Enforcing GAC is the task of removing from domains all values that have no support on a constraint. Many algorithms have been devised for establishing GAC according to the nature of the constraints.

On the other hand, k-Wise Consistency (kWC) [7,1] can be classified as a constraint-based consistency because it allows us to identify inconsistent tuples (initially accepted by constraints) instead of inconsistent values. It is based on the idea of extending (valid) assignments.

Definition 2. (Extension) *Let Y and Z be two sets of variables. An assignment τ' of $Y \cup Z$ is an* extension *on $Y \cup Z$ of an assignment τ of Y iff $\tau'[y] = \tau[y], \forall y \in Y$.*

Of course, a valid extension is simply an extension that corresponds to a valid assignment. We can now define k-wise consistency, which basically guarantees that every support on a constraint can be extended to any set of $k-1$ additional constraints. This kind of property allows us to reason about connections between constraints through shared variables.

Definition 3. *(*kWC) *A CSP* $P = (X, D, C)$ *is* k-wise consistent *(kWC) iff* $\forall c_1 \in C$, $\forall \tau \in c_1 : \tau \in D(scp(c_1))$, $\forall c_2, \ldots, c_k \in C, \exists \tau'$ *valid extension of* τ *on* $\bigcup_{i=1}^{k} scp(c_k)$ *satisfying* $c_2, \ldots, c_k$.

Note that k-wise consistency is called pairwise consistency for k=2 and three-wise consistency for k=3. It is immediate that k-wise consistency implies (k-1)-wise consistency. Enforcing kWC on a CSP involves removing from the constraints (i.e., considering as no more allowed) the tuples that cannot be extended. It thus modifies constraints, not domains. As a result, kWC is incomparable with GAC : a CSP can be kWC but not GAC and reciprocally [6]. However, combining both consistencies allows us to make more pruning of the domains than domain consistency alone. In this paper, we consider such a combination.

Definition 4. (GAC+kWC) *A CSP* P *is GAC+kWC iff* P *is both GAC and kWC.*

At this stage, although already suggested earlier, we can observe that GAC, kWC and GAC+kWC are well-behaved consistencies. We recall that a consistency ψ is well-behaved [10] when for any CSP P, the ψ-closure of P exists, where the ψ-closure of P is the greatest CSP, denoted by $\psi(P)$, which is both ψ-consistent and equivalent to P. The underlying partial order on CSPs is: $P' = (X, D', C') \preceq P = (X, D, C)$ iff $\forall x \in X, D'(x) \subseteq D(x)$ and there exists a bijection μ from C to C' such that $\forall c \in C, \mu(c) \subseteq c$. Enforcing ψ on P means computing $\psi(P)$, and an algorithm that enforces ψ is called a *ψ-algorithm.*

Interestingly, from GAC+kWC, we can derive a domain-filtering consistency, called *domain k-wise consistency*, or DkWC in short. When a CSP P is domain k-wise consistent, it means that all variable domains of P cannot be reduced when enforcing GAC+kWC.

Definition 5. (DkWC) *A CSP* $P = (X, D, C)$ *is* domain k-wise consistent *(DkWC) iff GAC+kWC*(P) *is a CSP* $Q = (X, D^Q, C^Q)$ *such that* $D = D^Q$.

GAC+kWC is both domain-filtering and constraint-filtering, which may render uneasy its implementation in constraint solvers, whereas DkWC is only domain-filtering. In this paper, we propose to enforce DkWC indirectly by considering a reformulation of the CSP to be solved.

3 Enforcing kWC Using k-dual CSPs

This section presents a filtering process for achieving kWC. The filtering procedure is a generalization to the k-wise case of the filtering process presented in [6], and different from the one presented in [7]. It will be useful for our DkWC algorithm, presented in the next section. Due to explicit access to the list of allowed tuples, table constraints are particularly adapted for strong constraint-based consistencies. The filtering procedures proposed in this paper are thus designed for such table constraints. From now on, all constraints will be assumed to be table constraints.

As kWC is a constraint-based consistency, the idea is to define and use a special dual form of the given CSP in order to obtain kWC by simply enforcing GAC on the dual representation. Because this dual form depends on the value of k, we call it *k-dual* CSP. This is a generalization of the dual used in [6] that is equivalent to the *order*

k *constraint graph* defined in [7]. Specifically, the k-dual of a CSP contains one *dual* variable x'_i per constraint c_i in the original CSP and one *k-dual* constraint c'_j per group of k original distinct constraints. Each variable x'_i has a domain which is the set of indexes of the tuples in the original constraint c_i, and each constraint c'_j is a table constraint representing the join of k original constraints. Note that the tuples in those new tables are represented with the indexes of the tuples in the original constraints, which allows the new constraints to have arity k only.

Definition 6. (k-dual CSP) *Let $P = (X, D, C)$ be a CSP. The* k-dual *of P is the CSP $P^{kd} = (X^{kd}, D^{kd}, C^{kd})$ where:*

- *for each constraint $c_i \in C$, X^{kd} contains a variable x'_i with its domain defined as $D^{kd}(x'_i) = \{1, 2, \ldots, |rel(c_i)|\}$,*
- *for each subset S of k constraints of C, C^{kd} contains a constraint c' such that $scp(c') = \{x'_i \mid c_i \in S\}$ and c' is a k-ary table constraint containing the join of all constraints in S (represented with the indexes of the original tuples).*

If P^{kd} is the k-dual of P, then variables and constraints of P are said to be *original* whereas variables and constraints of P^{kd} are said to be *dual*. An example of a k-dual CSP for $k = 3$ can be found in Example 1.

Example 1. Let $P = (X, D, C)$ be a CSP such that $X = \{u, v, w, x, y, z\}$, $D = \{1, 2, 3, 4\}^6$ and $C = \{c_1, c_2, c_3\}$ where:
- $scp(c_1) = \{u, v, w\}$ and $rel(c_1) = \{(1, 2, 3), (1, 2, 4)\}$,
- $scp(c_2) = \{u, x, y\}$ and $rel(c_2) = \{(1, 3, 4), (2, 3, 4)\}$,
- $scp(c_3) = \{v, x, z\}$ and $rel(c_3) = \{(2, 3, 1), (3, 3, 2)\}$.

The 3-dual of P is a CSP $P^{kd} = (X^{kd}, D^{kd}, C^{kd})$ such that $X^{kd} = \{x'_1, x'_2, x'_3\}$, $D^{kd} = \{1, 2\}^3$, and $C^{kd} = \{c'\}$ with $scp(c') = \{x'_1, x'_2, x'_3\}$ and $rel(c') = \{(1, 1, 1), (2, 1, 1)\}$. It represents the full join of the original constraints on $\{u, v, w, x, y, z\}$, which is composed of the tuples $(1, 2, 3, 3, 4, 1)$ and $(1, 2, 4, 3, 4, 1)$. For example, the first tuple is obtained by joining the first tuple of c_1, the first tuple of c_2 and the first one of c_3.

Property 1. A CSP is kWC iff its k-dual CSP is GAC. [6,7].

Property 1 is introduced in [6] for $k = 2$ and the general result is established in [7] for a similar k-dual CSP. A filtering procedure to enforce GAC+2WC (i.e., both pairwise consistency and generalized arc consistency) on a CSP P consists in (1) enforcing GAC on the 2-dual of P, then (2) restraining the constraints of P in order to only contain tuples corresponding to valid dual values, and finally (3) establishing GAC on P [7,3]. The generalization of this procedure to the k-wise case uses the k-dual instead of the 2-dual.

4 Enforcing DkWC Using k-interleaved CSPs

In this section, we propose to reformulate the CSP P to be solved in order to be able to enforce DkWC in a single step, just by applying classical GAC. Basically, to enforce DkWC with GAC propagators only, we first add to P all variables and all constraints

from the k-dual CSP of P. Then, we link dual variables and original constraints because, otherwise, the removal of a value from a dual variable would not leverage its corresponding original constraint (and reciprocally). In the definition of GAC+kWC (on which DkWC is based), only valid tuples can serve as supports either for values (generalized arc consistency part) or for other tuples (k-wise consistency part). The link we make guarantees that original tuples corresponding to invalid dual values are invalidated, and reciprocally, ensuring that original constraints and dual variables keep the same pace during filtering. The link we propose involves transforming each original constraint c_i into a new *hybrid* constraint $\phi(c_i)$ involving the original variables in the scope of c_i as well as the dual variable x'_i that is associated with c_i. For each tuple τ in $rel(c_i)$, we generate a tuple in $rel(\phi(c_i))$ by simply appending to τ its position in $rel(c_i)$.

Definition 7. (Hybrid Constraints) *Let $P = (X, D, C)$ be a CSP. The set of hybrid constraints $\phi(C)$ of P is the set $\{\phi(c_i) \mid c_i \in C\}$ where:*
- *$scp(\phi(c_i)) = scp(c_i) \cup \{x'_i\}$*
- *$rel(\phi(c_i)) = \{\tau_j \odot j \mid \tau_j$ is the j^{th} tuple of $rel(c_i)\}$*

with x'_i denoting the dual variable associated with c_i.

In this way, the removal of a value j from $D(x'_i)$ will be reflected in $\phi(c_i)$, as the tuple $\tau_j \odot j$ will be invalidated. Also, when the tuple $\tau_j \odot j$ becomes invalid due to a value removed from the domain of an original variable, j will be removed from $D(x'_i)$. We can now introduce *k-interleaved CSPs* .

Definition 8. (k-Interleaved CSP) *Let $P = (X, D, C)$ be a CSP. The* k-interleaved *of P is the CSP $P^{ki} = (X^{ki}, D^{ki}, C^{ki}) = (X \cup X^{kd}, D \cup D^{kd}, \phi(C) \cup C^{kd})$ where (X^{kd}, D^{kd}, C^{kd}) is the k-dual of P and $\phi(C)$ the hybrid constraints of P.*

The following property shows an interesting connection: enforcing GAC on the k-interleaved CSP of a CSP P is equivalent to enforcing GAC+kWC on P, when the focus is only on the domains of the variables of P.

Property 2. Let $P = (X, D, C)$ be a CSP and $P^{ki} = (X^{ki}, D^{ki}, C^{ki})$ be the k-interleaved CSP of P. If $Q = (X, D^Q, C^Q)$ is the GAC+kWC-closure of P and $R = (X^{ki}, D^R, C^{ki})$ is the GAC-closure of P^{ki}, then we have $D^Q = D^R[X]$ (i.e., $D^Q(x) = D^R(x), \forall x \in X$).

The intuition of the proof is as follows. On the one hand, each literal (x, a) of D^Q is supported on each constraint c^Q involving x by a valid tuple in Q. This tuple is k-wise consistent in Q. By Property 1, the dual variables in X^{ki} precisely encode this k-wise consistency. On the other hand, each literal (y, b) of $D^R[X]$ is supported on each constraint c^{ki} involving y by a valid tuple in R. As all supports on constraints of C^{ki} include a valid dual variable, we have that $D^Q = D^R[X]$.

Then, we can deduce the following corollary.

Corollary 1. *If the k-interleaved CSP of a CSP P is GAC then P is DkWC.*

It is important to note that "k-interleavedness" is preserved after refuting any value. This is stated by the following property (whose proof is omitted).

Property 3. Let $P = (X, D, C)$ be a CSP and P^{ki} be the k-interleaved CSP of P. $\forall x \in X, \forall a \in D(x)$, $P^{ki}|_{x \neq a}$ is the k-interleaved CSP of $P|_{x \neq a}$.

From Properties 2 and 3, we can derive the following important corollary.

Corollary 2. *Let $P = (X, D, C)$ be a CSP and $P^{ki} = (X^{ki}, D^{ki}, C^{ki})$ be the k-interleaved CSP of P. Let Δ be a set of value refutations on variables of X. If $Q = (X, D^Q, C^Q)$ is the GAC+kWC-closure of $P|_\Delta$ and $R = (X^{ki}, D^R, C^{ki})$ is the GAC-closure of $P^{ki}|_\Delta$, then we have $D^Q = D^R[X]$.*

Corollary 2 is central to our approach. It allows us to achieve DkWC indirectly using GAC, and at any stage of a backtrack search. So, it is important to note that the generation of the k-interleaved CSP is only performed once since it can be used during the whole search.

The complexity of enforcing DkWC is the complexity of enforcing GAC on the k-interleaved CSP. As the k-interleaved CSP only contains table constraints, the complexity analysis will use the optimal time complexity for a table constraint given in [14]. Let P be a CSP with n variables, a maximum domain size d, e constraints, a maximum number t of tuples allowed by a constraint, and a maximum constraint arity r. The k-interleaved CSP of P is a CSP with $n' = n + e$ variables, a maximum domain size $d' = max(d, t)$, $e' = e + \binom{e}{k}$ constraints[2], an upper bound $t' = t^k$ of the maximum number of allowed tuples by a constraint, and a maximum constraint arity $r' = max(r + 1, k)$ Enforcing GAC on the k-interleaved CSP with optimal table constraint propagators has a complexity of $O(e' \cdot (r' \cdot t' + r' \cdot d')) = O((\binom{e}{k} + e) \cdot (r' \cdot t' + r' \cdot d'))$.

Necessity of Hybrid Constraints. It is important to note that the filtering procedure for DkWC presented in this paper is stronger than the propagation that would be obtained by simply replacing the original constraints by their joins. The reason is that, in the second setting, the invalidation of a tuple in a join is not reflected in the other joins, whereas with the k-interleaved CSP, the supports for the tuples on a join must themselves be supported. This is illustrated by Example 2.

Example 2. Let $P = (X, D, C)$ be a CSP such that $X = \{x, y, u, v\}$, $D = \{0, 1\}^4$ and $C = \{c_1, c_2, c_3\}$ with $scp(c_1) = \{x, y, u, v\}$, $scp(c_2) = \{x, y\}$ and $scp(c_3) = \{u, v\}$, and $rel(c_1)$, $rel(c_2)$ and $rel(c_3)$ defined as in Figure 2(a). Let us compare domain pairwise consistency (D2WC) with the joins of any two pairs of constraints: in this CSP, the two possible joins and the 2-interleaved CSP are depicted in Figure 1. On the one hand, enforcing GAC on the two join constraints J_{12} and J_{13} has no effect (observe that values 0 and 1 are present in each column of both tables). On the other hand, enforcing GAC on the 2-interleaved CSP reduces $D(y)$ to $\{1\}$ and $D(v)$ to $\{0\}$. The reduction of $D(y)$ comes from the tuple $(0, 0)$ in $rel(c_2)$ which is the only support for $y = 0$ on c_2. This tuple is only supported in J'_{12} by the second tuple of c_1: $(0, 0, 0, 1)$. As $(0, 0, 0, 1)$ has no support on J'_{13}, we can safely remove 0 from $D(y)$.

5 Practical Use of k-interleaved CSPs

Enforcing GAC on the k-interleaved CSP may be expensive. One possible cause is the number of constraints from the k-dual CSP that are added to the k-interleaved CSP:

[2] $\binom{e}{k}$ is the binomial coefficient corresponding to the number of subsets of size k that can be formed using elements from a set of size e.

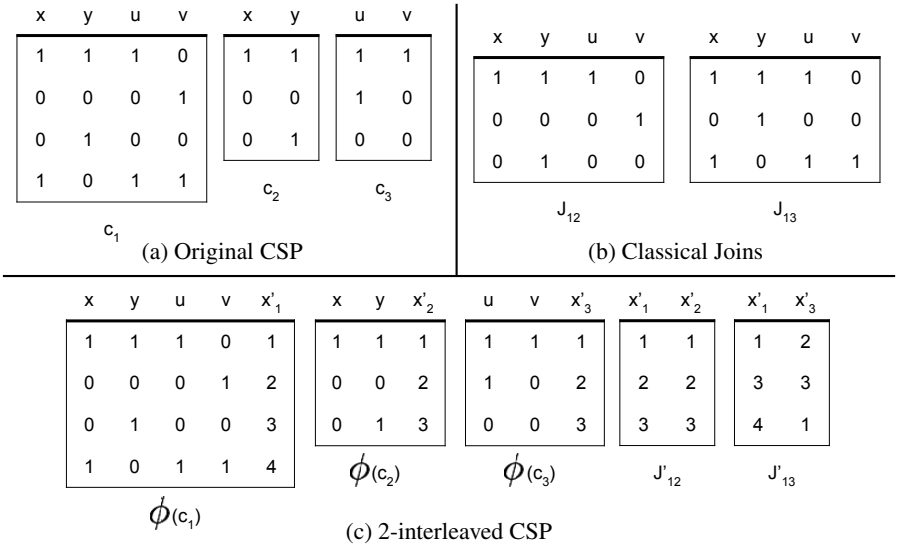

Fig. 1. Illustration of (a) the CSP, (b) the classical joins and (c) the 2-interleaved CSP from Example 2

$\binom{e}{k}$ for an original CSP with e constraints. Some of those constraints can safely be ignored. For instance, this is trivially the case for k-dual constraints that are based on original constraints sharing no variables. Of course, a trade-off can be made between propagation strength and time complexity by integrating only a subset of all possible $\binom{e}{k}$ constraints. In that case, we shall not achieve DkWC completely. Suppose that we limit the integration to the p most promising constraints from the k-dual CSP. The complexity, following our analysis performed above, becomes $O((e+p)\cdot(r'\cdot t'+r'\cdot d'))$: the term $\binom{e}{k}$ has been replaced by p. The most promising constraints can be selected, for example, according to the size of the joins. Indeed, small joins will induce more pruning whereas using large joins is another cause of inefficiency, as it is related to t'.

Following this discussion, we propose two weak variants of DkWC, and refer to them as *weak* DkWC: they only consider a subset of all possible k-dual constraints. The first one, called DkWC^{cy} only considers constraints from the k-dual CSP corresponding to cycles of original constraints (i.e., sequences of constraints at least sharing variables with previous and next constraints in a circular manner). There are typically far less cycles of k constraints than combinations of k constraints and besides they usually form smaller joins. The consistency level attained by DkWC^{cy} is weaker than DkWC but in practice, as we shall see, it shows good performances. Since all the original constraints are included in DkWC^{cy}, the consistency level attained is stronger than GAC. Unsurprisingly, for some problems, the size of the joins of some cycle constraints may be too large to be treated efficiently. For instance, in the modifiedRenault benchmark, some joins computed from cycles of length 3 exceed 10^6 tuples. This is why the second variant of DkWC, called DkWC^{cy-}, only considers constraints from the k-dual CSP

corresponding to cycles of original constraints and admitting a join size smaller than a specified parameter (e.g., a percentage of the size of the largest table). In other words, a maximum size is imposed on the size of the joins. The consistency level attained by DkWC^{cy-} is weaker than DkWC and DkWC^{cy}, but its practical interest will be shown on some problems. DkWC^{cy-} attains a level of consistency stronger than GAC.

6 Related Work

GAC has already been combined with 2WC, 3WC and kWC [7,6,3,17,8,16]. A first approach consists in weakening the combination, to obtain a pure domain-based consistency. We obtain then max-restricted pairwise consistency (maxRPWC) [16,17,15].

Definition 9. (maxRPWC) *A CSP $P = (X, D, C)$ is max-restricted pairwise consistent (maxRPWC) iff $\forall x \in X$, $\forall a \in D(x)$, $\forall c \in C \mid x \in scp(c)$, $\exists \tau \in D(scp(c))$ such that $\tau[x] = a, \tau \in c$ and $\forall c' \in C$ there exists a valid extension of τ on $scp(c) \cup scp(c')$ satisfying c'.*

MaxRPWC is a domain-filtering consistency, close to the idea of DkWC and GAC+ 2WC but weaker than GAC+2WC [3]. In [11], the authors propose a specialized filtering procedure, called *eSTR*, for enforcing GAC+2WC (called full pairwise consistency in their work) on table constraints. Two techniques are combined: simple tabular reduction (STR) and tuple counting. This allows eSTR to keep and update a counter, for each tuple of each table, of the number of supports it has in the other tables. This counter can be used to detect and remove unsupported tuples. This approach is orthogonal to the one presented in this paper. Indeed, in [11], the authors lift up an existing GAC propagator, STR, to eSTR. Our approach is to propose a filtering procedure, relying on a modified CSP, only using existing pure GAC propagators and we are not restricted to GAC+2WC.

Other approaches compute the kWC-closure of a CSP in a first step and then apply GAC in a second step, as proposed in [7,6,3] for 2WC and [8] for kWC. The approach in [8] relies on a specialized propagator, inspecting each constraint with respect to each relevant group of k constraints. Inspecting a constraint means searching, for each tuple of the constraint, a support in each group. This search for support is performed using a backtracking search (Forward Checking), on the dual encoding of the CSP. This whole process is sped up by memorizing, for each constraint and each group, the last encountered support. A similar approach is developed in [20] for relational neighborhood inverse consistency. In [8], the authors also propose a slightly weaker consistency, considering only groups of constraints forming connected components in the minimal dual graph. Although attractive, these original forms of propagators can be hard to include in existing constraint solvers. For instance, in *Comet*, the context management system makes the start of an independent search inside a propagator impossible.

Other related approaches exist, although not trying to enforce directly GAC+kWC. In [12], the authors propose a consistent reformulation for the conjunction of two tables sharing more than one variable, keeping the space complexity low. In [2], the authors propose an algorithm to achieve GAC on global constraints. In this work, the global constraints are perceived as groups of constraints and the CSPs they define are solved

on the fly to achieve GAC on them. GAC+kWC on a group of k constraints can be seen as solving the subproblem they define, but in our approach, the subproblems are not solved on the fly.

The easy integration of strong levels of consistency into existing solvers has been studied in [19]. The integration is performed within a generic scheme, incorporating the subset of the constraints involved in the local consistency into a global constraint.

7 Experimental Results

This section presents some experimental results concerning DkWC. For each test, we propose to maintain this property on k-interleaved CSPs at each node of the search trees developed by a backtrack search. However, as discussed in Section 5, including all k-dual constraints is unpractical for many problems, because of the number of additional constraints and/or because of their size. So, in our experiments, we have only used the weaker versions defined in Section 5, namely, DkWC^{cy} and DkWC^{cy-}, and we have focused our attention to weak D3WC and weak D4WC. Those values of k allow a significant search space reduction with respect to GAC while keeping the number and size of k-dual constraints tractable. Notice that labeling is only performed for original variables during search, and that all solutions are searched for. The GAC propagator used for the original constraints as well as the k-dual ones is the optimal state-of-the-art propagator from [14].

Our filtering procedure is compared with the GAC propagator from [14], the max-RPWC3 procedure from [3] and the state-of-the-art eSTRw propagator from [11]. The eSTRw propagator is weaker than eSTR but easier to incorporate into an existing solver, and is at least as good as eSTR on the benchmarks used in [11]. All the algorithms are (re-)implemented on top of Comet, but as mentioned in Section 6, it is unfortunately impossible to implement the filtering algorithm from [8] in Comet. Eight different benchmarks have been used. Two of them contain only binary table constraints, five of them contain binary and ternary table constraints and the last benchmark contains table constraints up to arity 10. The tests are executed on an Intel Xeon 2.53GHz using Comet 2.1.1. A timeout of 20 minutes on the total execution time is used for each instance. When comparing different techniques in terms of CPU time and search space sizes, we can only use the subset of instances for which none of the techniques timed out. In the results, we thus do not report measurements for some of the techniques on some benchmarks because including them would cause the common instance set to be empty or too small for a meaningful comparison. In the tables, a '-' thus represents a technique that timed out on the set of instances considered. The percentage of the instance set that is solved is however given for each technique on each benchmark.

The results are presented in Table 1. For each instance set and each technique, we report means of different quantities (times are in seconds): the execution time (T), the "posting" time (pT), the join selection time (jST) that corresponds to the amount of time used to select the joins for the k-interleaved CSPs, the join computation time (jT), the number of propagator calls (nP), the number of fails (nF) and the number of choice points (nC). Table 1 also reports the percentage to the best with respect to execution time (%b), the mean of the percentage to the best instance by instance (μ%b) and the

percentage of instances from the sets that are solved (%sol). The total time (T) includes all precomputations the algorithms have to perform before search. This means that both times of join selection (jST) and join computation (jT) for our DkWC algorithm are included in T. The posting time (pT) is the time taken between the loading of the instance file and the start of search without the time for jT. It thus includes time for all precomputations except for join computation. The difference between %b and μ%b is the following. For %b, all execution times are averaged before computing it: there is thus one identified best algorithm. For μ%b, the percentages are first computed instance by instance, and then aggregated with a geometrical mean (as suggested in [5]): this measure takes into account the fact that different instances may have different best algorithms.

Binary Random Instances. This instance set contains 50 instances involving binary table constraints. These instances have 50 variables, a uniform domain size of 10 and 166 constraints whose proportion of allowed tuples is 0.5. They have been generated using the model RD [21], in or close to the phase transition. The search strategy used to solve them (for all techniques) is a lexicographic variable and value ordering. On this benchmark, D3WC^{cy} includes on average 48.4 3-dual constraints, and their tables contain on average 111.5 tuples. We can see that the pruning obtained by D3WC^{cy} on this benchmark allows it to reduce drastically the search space. Moreover, since the mean number of added constraints from the 3-dual CSP and their size is small, D3WC^{cy} has the lowest overall computation time. Propagators maxRPWC3 and eSTRw also reduce the search space with respect to GAC (partly due to the presence of constraints with identical scopes), but this reduction comes at the price of a greater total computation time.

Ternary Random Instances. This instance set contains 50 instances involving ternary table constraints. These instances have 50 variables, a uniform domain size of 5 and 75 constraints whose proportion of allowed tuples is 0.66. They have been generated using the model RD [21], in or close to the phase transition. The search strategy used to solve them is a lexicographic variable and value ordering. On this benchmark, D3WC^{cy} includes, on average 112.3 3-dual constraints, and their tables contain on average 529.5 tuples. As for the binary case, the search space reduction obtained by D3WC^{cy} is important. On this benchmark, the size and number of added constraints is small enough to allow D3WC^{cy} to be the fastest technique. Note that the search space reduction obtained by maxRPWC3 doesn't repay its cost, contrary to eSTRw.

AIM Instances. This instance set contains 24 instances from the AIM series used in the CSP solver competition [18] (100 variables, a majority of ternary constraints and binary ones). The search strategy used is a lexicographic variable and value ordering. D3WC^{cy} includes 3000 constraints from the 3-dual, on average, and the added constraints contain on average 30.3 tuples. On this benchmark, the filtering obtained by maxRPWC3, eSTRW and D3WC^{cy} allows each of them to be significantly faster than GAC. Although D3WC^{cy} achieves the best search space reduction, eSTRw remains the fastest technique. The greater computation time for D3WC^{cy} is due to the number of 3-dual constraints included in the 3-interleaved CSPs: 3000 on average while the number of original constraints lies between 150 and 570. Even if the 3-dual constraints have small tables, they still have to be propagated during search. Interestingly, D3WC^{cy} solves significantly more instances than the other techniques.

Table 1. Results of the experiments on the different benchmarks. T is the mean time in seconds, pT is the mean posting time in seconds, jST is the mean join selection time, jT is the mean join time, nP is the number of calls to the propagators, nF is the number of fails during the search, nC is the number of choice points during the search, %b is the percentage to the best, μ%b is the mean percentage to the best and %sol is the percentage of instances solved.

propagator	T	pT	jST	jT	nP	nF	nC	%b	μ%b	%sol
Binary Random										
GAC	9.9	0.0	0.0	0.0	3 M	7.7 k	1 213.2	337	218	**100**
maxRPWC3	72	0.7	0.0	0.0	148 k	2.2 k	340.1	2448	1833	98
eSTRw	11.4	0.1	0.0	0.0	293 k	2.2 k	340.1	389	283	**100**
D3WC^{cy}	**2.9**	0.1	0.0	0.1	963 k	0.5 k	68.4	100	113	**100**
Ternary Random										
GAC	23.1	0.0	0.0	0.0	4 M	42.4 k	11.8 k	183	223	**100**
maxRPWC3	124	0.2	0.0	0.0	237 k	8.2 k	2.2 k	982	1455	90
eSTRw	16.6	0.0	0.0	0.0	409 k	7.7 k	2.1 k	131	189	**100**
D3WC^{cy}	**12.6**	0.5	0.1	0.4	2 M	0.6 k	0.1 k	100	143	**100**
AIM										
GAC	82	0.1	0.0	0.0	35 M	941.6 k	522 k	6745	460	46
maxRPWC3	14.7	1.0	0.0	0.0	46 k	1.2 k	0.7 k	1204	1481	46
eSTRw	**1.2**	0.3	0.0	0.0	35 k	0.7 k	0.4 k	100	208	50
D3WC^{cy}	3.4	2.4	1.2	0.5	139 k	0.1 k	0.1 k	279	497	**88**
Pret										
GAC	160	0.0	0.0	0.0	58 M	7 M	5 M	121	121	**50**
maxRPWC3	977	0.0	0.0	0.0	26 M	7 M	5 M	741	741	**50**
eSTRw	504	0.0	0.0	0.0	30 M	7 M	5 M	382	382	**50**
D3WC^{cy}	**132**	0.0	0.0	0.0	57 M	4 M	3 M	100	100	**50**
Langford-2										
GAC	**0.5**	0.0	0.0	0.0	171 k	1.4 k	1 k	100	100	**58**
maxRPWC3	44.6	2.2	0.0	0.0	43 k	1.4 k	1 k	9637	3863	46
eSTRw	1.5	0.1	0.0	0.0	65 k	1.4 k	1 k	326	233	54
D3WC^{cy}	10.5	0.9	0.1	3.2	2 M	0.7 k	0.7 k	2270	1782	50
Dubois										
GAC	793	0.0	0.0	0.0	158 M	42 M	37 M	394	390	15
maxRPWC3	-	-	-	-	-	-	-	-	-	8
eSTRw	598	0.0	0.0	0.0	37 M	5 M	3 M	297	294	15
D4WC^{cy}	**201**	0.1	0.0	0.0	68 M	2 M	1 M	100	100	**30**
TSP-20										
GAC	**52**	0.5	0.0	0.0	14 M	17 k	7 k	100	100	**93**
maxRPWC3	-	-	-	-	-	-	-	-	-	33
eSTRw	233	1.5	0.1	0.0	5 M	17 k	7 k	447	438	80
D3WC^{cy}	-	-	-	-	-	-	-	-	-	40
D3WC^{cy-}	94	4.3	0.6	0.1	40 M	17 k	7 k	180	270	**93**
Modified Renault										
GAC	-	-	-	-	-	-	-	-	-	6
eSTRw	-	-	-	-	-	-	-	-	-	0
maxRPWC3	743	0.0	0.0	0.0	23.7	0.0	0.0	148	417	26
D3WC^{cy}	-	-	-	-	-	-	-	-	-	0
D3WC^{cy-}	**502**	497	3.9	5.1	33 k	0.0	0.0	100	111	**34**

Pret Instances. This instance set also comes from the CSP solver competition [18] and counts 8 instances (only ternary table constraints). The search strategy used is a lexicographic variable and value ordering. D3WC^{cy} includes on average 13 constraints that have a mean size of 8 tuples. On this benchmark, neither maxRPWC3 nor eSTRw is able to reduce the search space with respect to GAC. Their additional computations make them slower than GAC. The small number of small constraints from the 3-dual CSP included by D3WC^{cy} allows it to significantly reduce the search space and to be the fastest on this series. The mean percentage to the best (μ%b) of D3WC^{cy} means that it is the best technique on average but also on each instance. Note that the join selection, join computation and posting times are negligible on this problem.

Langford Problem. Langford number problem is Problem 24 of CSPLIB[3], here modeled with binary table constraints only. We used the set Langford-2 containing 24 instances that can be found in [9]. The search strategy used is *dom/deg* combined with a lexicographic value ordering. On this set, D3WC^{cy} includes, on average, 328 3-dual constraints whose tables contain 274.7 tuples on average. On this benchmark, GAC is the fastest technique on average (actually, it is the fastest technique on each instance). Neither maxRPWC3 nor eSTRw are able to reduce the search space with respect to GAC. However, they have a lower propagator call count. This is due to their ability to reach the fixed point faster. The number of constraints added from the 3-dual by D3WC^{cy} is large comparatively to the number of original constraints (the non-timeout instances are the smallest ones). The small search space reduction obtained by D3WC^{cy} does not compensate the cost to propagate all the added constraints. The number of propagator calls is significantly larger for D3WC^{cy}. We can also see that, on this benchmark, the time required to compute the joins is larger than the time required by GAC to solve the instances.

Dubois Instances. Those 13 instances also comes from the CSP solver competition [18] (ternary table constraints). These instances do not contain any cycle of original constraints of length 3. We thus present the results of D4WC^{cy}. On this series, D4WC^{cy} adds on average 156 4-dual constraints and their tables contain, on average, 15.2 tuples. Clearly, D4WC^{cy} is the fastest approach here and solves more instances than the other techniques. D4WC^{cy} is also the fastest on each instance, as shown by the mean percentage to the best (μ%b). The search space reduction obtained by eSTRw is less than that obtained by D4WC^{cy} but it allows it to be faster than GAC.

Travelling Salesman Problem. We used the set of 15 Travelling Salesman satisfaction instances *tsp-20* from [9] (table constraints of arity 2 and 3). The search strategy used here is *dom/deg* combined with a lexicographic value ordering. On this instance set, there is, on average, 1000 cycles of length 3 in the 3-dual CSP and they contain up to 2000 tuples. In that context, D3WC^{cy} only solves 40% of the instances. We thus present the results for D3WC^{cy-} where the limit on the size of the joins is set to one percent of the maximal original constraint size (200). D3WC^{cy-} includes 59.8 constraints from the 3-dual CSP on average, and their tables contain, on average, 26 tuples. As we can see, on those instances, neither eSTRw nor D3WC^{cy-} is able to reduce the search space. The extra computations of eSTRw and the extra propagation effort of D3WC^{cy-} make them slower than GAC (which is also the fastest approach on each

[3] www.csplib.org

Table 2. Summary of the results of the experimental section. T is the total solving time in seconds and %sol is the percentage of the instances solved.

Benchmark	GAC		maxRPWC3		eSTRw		wDkWC	
	T	%sol	T	%sol	T	%sol	T	%sol
Binary Random	9.9	**100**	72	98	11.4	**100**	**2.9**	**100**
Ternary Random	23.1	**100**	124	90	16.6	**100**	**12.6**	**100**
AIM	82	46	14.7	46	**1.2**	50	3.4	**88**
Pret	160	**50**	977	**50**	504	**50**	**132**	**50**
Langford2	**0.5**	**58**	44.6	46	1.5	54	10.5	50
Dubois	793	15	-	8	598	15	**201**	**30**
TSP-20	**52**	**93**	-	33	233	80	94	**93**
ModRenault	-	6	743	0	-	26	**502**	**34**

instance). However, D3WC^{cy-} is faster than eSTRw and it is the only one able to solve the same number of instances as GAC.

Modified Renault Problem. The modified Renault problem instances originate from a real Renault Megane configuration problem, modified to generate 50 instances[9] (large tables and arities up to 10). The search strategy used is *dom/deg* variable ordering combined with a lexicographic value ordering. Since the tables of the original problem can count up to 50K tuples, D3WC^{cy} is unpractical because of the size of the joins. We thus present the results for D3WC^{cy-} where the limit on the size of the joins has been set to one percent of the largest original constraint size (500), as in the TSP benchmark. On those instances, D3WC^{cy-} includes 481.6 3-dual constraints on average, and their tables contain on average 253.9 tuples. As we can see, both maxRPWC3 and D3WC^{cy-} detect the inconsistencies of all instances without performing any search. However, despite the fact that D3WC^{cy-} has a larger propagator call count, it is faster than maxRPWC3. D3WC^{cy-} is also able to solve more instances than maxRPWC3.

Summary of the Experimental Results. A summary of the experimental results can be found in Table 2. This table contains the total execution time (T) and the percentage of instances solved (% sol) for each technique. The column wDkWC represents our weak DkWC approach: it is D3WC^{cy} for binary random, ternary random, AIM, Pret and Langford-2 instances, D4WC^{cy} for Dubois instances and D3WC^{cy-} for TSP-20 and modified Renault instances. Weak DkWC is faster than maxRPWC3 and eSTRw, except for two benchmarks. It is also faster than GAC on all but two benchmarks, where GAC is faster than all strong consistencies. Weak DkWC is also the strong consistency leading to the largest reductions of search space. Except on Langford-2, weak DkWC solves the largest number of instances within the time limit.

For all these benchmarks, we insist that (full) DkWC is unpractical because of the number of possible joins and/or their size. This is the reason why we have introduced weak DkWC. On the majority of benchmarks, we used DkWC^{cy} but on two benchmarks, even D3WC^{cy} suffers from the number of 3-dual constraints and their sizes. Consequently, we also used DkWC^{cy-}, for which the best limit on the joins size has been empirically found to be equal to 1 percent of the maximum original constraint size. This parameter value allows D3WC^{cy-} to include a significant number of small

(highly filtering) 3-dual constraints without including too many of them. All these results show, on a large variety of benchmarks with constraints of various arities, that the weak DkWC filtering procedures defined in this paper are competitive.

8 Conclusion

In this paper, we have derived a domain-filtering consistency, DkWC, from the combination of kWC and GAC. We have shown how to establish and maintain this strong consistency by simply establishing and maintaining GAC on so-called k-interleaved CSPs. Such reformulated CSPs, which integrate dual variables, hybrid constraints and k-dual constraints, are simple to generate, and need to be generated only once before search. To manage the complexity of join operations, we have proposed a few solutions such as the ones relying on the presence of cycles or on the use of a limit on the maximal size of joins. The experimental results that we have obtained show, on a large variety of problems, that our weak DkWC filtering procedures are competitive.

Acknowledgments. The first author is supported as a Research Assistant by the Belgian FNRS. This research is also partially supported by the FRFC project 2.4504.10 of the Belgian FNRS, and by the UCLouvain Action de Recherche Concertée ICTM22C1. The third author benefits from the financial support of both CNRS and OSEO within the ISI project 'Pajero'.

References

1. Bessiere, C.: Constraint propagation. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. Elsevier, New York (2006)
2. Bessière, C., Régin, J.-C.: Enforcing arc consistency on global constraints by solving subproblems on the fly. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 103–117. Springer, Heidelberg (1999)
3. Bessiere, C., Stergiou, K., Walsh, T.: Domain filtering consistencies for non-binary constraints. Artificial Intelligence 72(6-7), 800–822 (2008)
4. Debruyne, R., Bessière, C.: Domain filtering consistencies. Journal of Artificial Intelligence Research 14, 205–230 (2001)
5. Fleming, P., Wallace, J.: How not to lie with statistics: the correct way to summarize benchmark results. Communications of the ACM 29(3), 218–221 (1986)
6. Janssen, P., Jégou, P., Nouguier, B., Vilarem, M.-C.: A filtering process for general constraint-satisfaction problems: achieving pairwise-consistency using an associated binary representation. In: Proceedings of IEEE Workshop on Tools for Artificial Intelligence, pp. 420–427 (1989)
7. Jégou, P.: Contribution à l'étude des Problèmes de Satisfaction de Contraintes: Algorithmes de propagation et de résolution. Propagation de contraintes dans les réseaux dynamique. PhD thesis, Université de Montpellier II (1991)
8. Karakashian, S., Woodward, R., Reeson, C., Choueiry, B., Bessiere, C.: A first practical algorithm for high levels of relational consistency. In: Proceedings of AAAI 2010, pp. 101–107 (2010)

9. Lecoutre, C.: Instances of the Constraint Solver Competition, http://www.cril.fr/~lecoutre/
10. Lecoutre, C.: Constraint Networks: Techniques and Algorithms. ISTE/Wiley (2009)
11. Lecoutre, C., Paparrizou, A., Stergiou, K.: Extending STR to a higher-order consistency. In: Proceedings of AAAI 2013, pp. 576–582 (2013)
12. Lhomme, O.: Practical reformulations with table constraints. In: Proceedings of ECAI 2012, pp. 911–912 (2012)
13. Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence 8(1), 99–118 (1977)
14. Mairy, J.-B., Van Hentenryck, P., Deville, Y.: An optimal filtering algorithm for table constraints. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 496–511. Springer, Heidelberg (2012)
15. Paparrizou, A., Stergiou, K.: An efficient higher-order consistency algorithm for table constraints. In: Proceedings of AAAI 2012, pp. 335–541 (2012)
16. Stergiou, K.: Strong inverse consistencies for non-binary CSPs. In: Proceedings of ICTAI 2007, pp. 215–222 (2007)
17. Stergiou, K.: Strong domain filtering consistencies for non-binary constraint satisfaction problems. International Journal on Artificial Intelligence Tools 17(5), 781–802 (2008)
18. van Dongen, M., Lecoutre, C., Roussel, O.: CSP solver competition (2008), http://www.cril.univ-artois.fr/CPAI08/
19. Vion, J., Petit, T., Jussien, N.: Integrating strong local consistencies into constraint solvers. In: Larrosa, J., O'Sullivan, B. (eds.) CSCLP 2009. LNCS (LNAI), vol. 6384, pp. 90–104. Springer, Heidelberg (2011)
20. Woodward, R., Karakashian, S., Choueiry, B., Bessiere, C.: Solving difficult CSPs with relational neighborhood inverse consistency. In: Proceedings of AAAI 2011, pp. 112–119 (2011)
21. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random constraint satisfaction: easy generation of hard (satisfiable) instances. Artificial Intelligence 171(8-9), 514–534 (2007)