Higher-Order Consistencies Through GAC on Factor Variables

Chavalit Likitvivatanavong, Wei Xia, and Roland H. C. Yap

School of Computing, National University of Singapore, Singapore
{chavalit,xiawei,ryap}@comp.nus.edu.sg

Abstract. Filtering constraint networks to reduce search space is one of the main cornerstones of Constraint Programming and among them (Generalized) Arc Consistency has been the most fundamental. While stronger consistencies are also the subject of considerable attention, none matches GAC's and for this reason it continues to advance at a steady pace and has become the popular choice of consistency for filtering algorithms. In this paper, we build on the success of GAC by proposing a way to transform a constraint network into another such that enforcing GAC on the latter is equivalent to enforcing a stronger consistency on the former. The key idea is to factor out commonly shared variables from constraints' scopes, form new variables, then re-attach them back to the constraints where they come from. Experiments show that this method is inexpensive and outperforms specialized algorithms and other techniques when it comes to full pair-wise consistency (FPWC).

1 Introduction

Generalized arc consistency (GAC) is one of the most studied filtering algorithms for constraint satisfaction problems (CSPs) due to its simplicity and excellent performance in practice. Domain reduction interspersed with GAC during backtracking search has become the foremost method for solving a general CSP [17]. GAC on positive table constraints, in particular, has received a great deal of attention in recent years [3, 4, 9, 10, 12, 14]. These advances in turn provide a basis for many algorithms that enforce even stronger consistencies than GAC to build on.

In [6] it was shown that a network is pairwise consistent (PWC) iff its dual CSP is arc consistent. PWC is a k-wise consistency [5] for the case where $k = 2$. This is one of the earlier works that demonstrates how (G)AC can be used to achieve other types of consistencies. Consistencies of level/order higher than GAC are also the subject of many recent works [8, 11, 16]. Specifically, maxRPWC, PWC, and FPWC are investigated in [2, 11, 16]. Many of the algorithms that enforce these consistencies are based on well-established GAC algorithms. In [16], the authors extended the GACva algorithm [12] to enforce maxRPWC. Subsequently, STR2 was extended to cope with FPWC, resulting in eSTR2 [11] which gets improvement similar to how STR2 outperforms GACva.

As another area of focus in CSPs, researchers have studied how to transform non-binary constraint networks into equivalent binary constraint networks so that the algorithms and methods from the binary case can be applied [1, 18]. Two techniques emerge as a result: the hidden transformation and the dual transformation. Both rely on the dual

variable associated with each constraint, whose domain values have a one-to-one correspondence with the constraint's tuples. Nevertheless, the benefits of the transformation diminish as filtering algorithms for non-binary CSPs get better.

In this paper, we propose to transform a non-binary constraint network into another non-binary constraint network such that the latter is GAC if and only if the former is full pairwise consistency (FPWC), which means both GAC and PWC. In this respect, our intention is similar to that of the authors of [13] who proposed another kind of transformation. Like the transformation from non-binary to binary network, the one in [13] is based on dual variables. But here the dual variable is included into the scope of the constraint it is associated with. The pruning power comes from the join of tables that uses the dual variables as its scope so that propagation can be transmitted directly to other constraints.

Our transformation works in a fundamentally different way. Instead of forming a dual variable for each constraint, we factor out commonly shared variables among them. These variables form new compound variables that will be augmented to only the constraints that are involved. For FPWC, no new constraints are created. We extend this transformation to cover general k-wise consistency, adding new constraints reduced from the join of k tables. Preliminary experiments show that for FPWC our method is faster than both [13] and a specialized FPWC algorithm. For k-wise consistency where $k \geq 3$, our transformation can lower the number of nodes visited during search but it is more costly and thus of limited application unless the search reduction is large.

2 Preliminaries

A constraint network $\mathcal{P}$ is a $(\mathcal{X}, \mathcal{C})$ where $\mathcal{X}$ is a set of n variables $\{x_1, \ldots, x_n\}$ and $\mathcal{C}$ a set of e constraints $\{c_1, \ldots, c_e\}$. $D(x)$ is the domain of $x \in \mathcal{X}$. During search, $D^c(x)$ denotes the current domain of x. If $a \in D^c(x)$, a is said to be *present* in $D(x)$; otherwise a is *absent* from $D(x)$. We use (x, a) to denote the value $a \in D(x)$ (or simply a when the context is clear). Each $c \in \mathcal{C}$ involves two components: a scope ($scp(c)$) which is an ordered subset of variables of $\mathcal{X}$; and a relation over the scope ($rel(c)$). Given *scp*(*c*) = $\{x_{i_1}, \ldots, x_{i_r}\}$, $rel(c) \subseteq \prod_{j=1}^{r} D(x_{i_j})$ represents the set of satisfying combinations of values for the variables in $scp(c)$. We may also refer to c by $c(x_{i_1}, \ldots, x_{i_r})$ to emphasize the scope. A constraint's scope can be made unique by combining the constraint's relation with relations from other constraints with the same scope through intersection. We assume a total ordering for every $rel(c)$ and use $\rho(c, i)$ to denote the i^{th} tuple. The *arity* of c is $|scp(c)|$. Given an ordered set $S \subseteq scp(c)$ and $\tau \in rel(c)$, the projection of τ on S ($\tau[S]$) is the tuple consisting of only the components of τ that correspond to the variables in S. A tuple $\tau = (a_{i_i}, \ldots, a_{i_k})$ where $a_{i_j} \in D(x_{i_j})$ is said to be an tuple over $\{x_{i_1}, \ldots, x_{i_k}\}$. The join of constraints c_i and c_j ($c_i \bowtie c_j$) is a constraint whose scope is $scp(c_i) \cup scp(c_j)$ and whose relation is $\{\tau \mid \tau$ is a tuple over $scp(c_i) \cup scp(c_j) \wedge \tau[scp(c_i)] \in rel(c_i) \wedge \tau[scp(c_j)] \in rel(c_j)\}$. The join of tuples $\tau_i \in rel(c_i)$ and $\tau_j \in rel(c_j)$ ($\tau_i \bowtie \tau_j$) is the tuple τ over $scp(c_i) \cup scp(c_j)$ such that $\tau[scp(c_i)] = \tau_i$ and $\tau[scp(c_j)] = \tau_j$. When elements in $rel(c)$ are given explicitly, c is called a *positive table constraint*. A tuple $\tau \in rel(c)$ is *valid* iff $\tau[x] \in D^c(x)$ for each $x \in scp(c)$. Otherwise τ is *invalid*. A tuple $\tau \in rel(c)$ is a

support of (x, a) in c iff $\tau[x] = a$. A value (x, a) is generalized arc-consistent (GAC) on a constraint c involving x iff there exists a *valid support* τ of (x, a) in c. A value (x, a) is GAC iff it is GAC on every constraint c involving x. A variable x is GAC iff $D^c(x) \neq \emptyset$ and (x, a) is GAC for each $a \in D^c(x)$. $\mathcal{P}$ is GAC iff each of its variables is GAC. A *solution* to $\mathcal{P}$ is a valid tuple over $\mathcal{X}$ such that every constraint is satisfied. $\mathcal{P}$ is *satisfiable* iff one solution exists. The constraint satisfaction problem (CSP) is the NP-hard task of determining whether a given constraint network is satisfiable or not.

A *compound variable* X is a cross-product composition from $\{x_{i_1}, \ldots, x_{i_m}\} \subseteq \mathcal{X}$, called X's *signature* ($\sigma(X)$), where $D(X) \subseteq \prod_{j=1}^{m} D(x_{i_j})$ and its values are sometimes referred to as *compound values*. Given a constraint c and an ordered set $S = \{x_{i_1}, \ldots, x_{i_m}\} \subseteq scp(c)$, we denote $\lambda_c(S)$ to be the compound variable on S with respect to c whose domain $D(\lambda_c(S))$ is $\{\tau[S] \mid \tau \in rel(c)\}$. It follows that $\sigma(\lambda_c(S)) = S$. A value in $D(\lambda_c(S))$ may be written as $\bar{a} = (a_{i_1}, \ldots, a_{i_m})$. We also use $\pi(\lambda_c(S), x_{i_k})$ to denote $\{a_{i_k} \mid \bar{a} \in D(\lambda_c(S))\}$ where $k \in \{1, \ldots, m\}$. Similarly, $\pi^c(\lambda_c(S), x_{i_k}) = \{a_{i_k} \mid \bar{a} \in D^c(\lambda_c(S))\}$. We may drop the subscript and write $\lambda(S)$ if there is no ambiguity. Non-compound variables are called *ordinary* variables. For uniformity, σ is defined for all variables, i.e. $\sigma(x) = \{x\}$ for an ordinary variable x. A value (x, a) is *max-restricted pairwise consistent* (maxRPWC) iff for all $c_i \in \mathcal{C}$ where $x \in scp(c_i)$, (x, a) has a valid support τ_i in $rel(c_i)$ such that for any other $c_j \in \mathcal{C}$ there exists a valid tuple $\tau_j \in rel(c_j)$ and $\tau_i[scp(c_i) \cap scp(c_j)] = \tau_j[scp(c_i) \cap scp(c_j)]$. $\mathcal{P}$ is maxRPWC iff all values are maxRPWC. $\mathcal{P}$ is *k-wise consistent* (kWC) iff given any group of k constraints $\{c_{i_i}, \ldots, c_{i_k}\}$, then for any $\tau \in rel(c_{i_j})$ for some j there exists a valid tuple τ' over $\bigcup_{l=1}^{k} scp(c_{i_l})$ such that $\tau'[scp(c_{i_j})] = \tau$ and $\tau'[scp(c_{i_l})] \in rel(c_{i_l})$ for all $l \in \{1, \ldots, k\}$. If $\mathcal{P}$ is kWC then $\mathcal{P}$ is (k-1)WC. When k is equal to two, it is also called *pairwise consistency* (PWC). $\mathcal{P}$ is *full pairwise consistent* (FPWC) iff it is both GAC and PWC. FPWC is also equivalent to PWC together with maxRPWC [11].

3 Reformulation

First we give a straightforward reformulation of a constraint network that encodes FPWC as follows. Given $\mathcal{P} = (\mathcal{X}, \mathcal{C})$, we construct $\mathcal{P}^+ = (\mathcal{X} \cup \mathcal{W}, \mathcal{C}^+)$ such that $\mathcal{W} = \{ \lambda_{c_i}(S), \lambda_{c_j}(S) \mid S = scp(c_i) \cap scp(c_j))$ for all $i \neq j \wedge |S| > 1\}$ and $\mathcal{C}^+$ includes constraints of the following three types. The first involves a simple extension of constraints in $\mathcal{C}$. For each $c'_i \in \mathcal{C}^+$, $1 \leq i \leq e$, we have,

- $scp(c'_i) = scp(c_i) \cup \{\lambda_{c_i}(S) \mid \lambda_{c_i}(S) \in \mathcal{W} \wedge S \subseteq scp(c_i)\}\}$
- for any $\tau \in rel(c_i)$, $\tau' \in rel(c'_i)$ is a tuple extended from τ such that
  - $\tau'[x] = \tau[x]$ for any $x \in scp(c_i)$
  - for any $\lambda_{c_i}(S) \in scp(c'_i)$, $\tau'[\lambda_{c_i}(S)] = \tau[S]$

The second type of constraints involves equality between $\lambda_{c_i}(S)$ and $\lambda_{c_j}(S)$ in $\mathcal{W}$ for any i, j, and S. The third involves compatibility constraints between a compound variable and each variable in its signature. That is, given $\lambda_c(S)$ such that $S = \{x_{i_1}, \ldots, x_{i_m}\}$, there is a constraint between $\lambda_c(S)$ and each x_{i_k} that forces $\pi^c(\lambda_c(S), x_{i_k}) = D^c(x_{i_k})$. As a result of this construction, in a generalized arc-consistent $\mathcal{P}^+$ any valid tuple in a constraint c can be extended to a valid tuple over $scp(c) \cup scp(c')$ for any other constraint c' through variables in $\mathcal{W}$. The proof is omitted due to space restrictions.

Theorem 1. *$\mathcal{P}^+$ is GAC if and only if $\mathcal{P}$ is FPWC.*

Next we show how $\mathcal{P}^+$ can be simplified while still preserving Theorem 1. Instead of posting an equality constraint between every pair of compound variables with the same signature, we unify all these compound variables into a single variable. Equality constraints are removed. Given $\mathcal{P}$= $(\mathcal{X}, \mathcal{C})$ and $\mathcal{P}^+$= $(\mathcal{X} \cup \mathcal{W}, \mathcal{C}^+)$, the *factor encoding* (FE) of $\mathcal{P}$ is the network $\mathcal{P}^*$ = $(\mathcal{X} \cup \mathcal{W}^*, \mathcal{C}^*)$ where,

$\mathcal{W}^* = \{\lambda(S) \mid D(\lambda(S)) = \bigcup_k D(\lambda_{c_k}(S))$ for all k such that $\lambda_{c_k}(S) \in \mathcal{W}\}$

and for each $c_i^* \in \mathcal{C}^*$, $1 \leq i \leq e$,

- $scp(c_i^*) = scp(c_i) \cup \{\lambda(S) \mid \lambda(S) \in \mathcal{W}^* \wedge S \subseteq scp(c_i)\}\}$
- for any $\tau \in rel(c_i)$, $\tau^* \in rel(c_i^*)$ is a tuple extended from τ such that
  - $\tau^*[x] = \tau[x]$ for any $x \in scp(c_i)$
  - for any $\lambda(S) \in scp(c_i^*)$, $\tau^*[\lambda(S)] = \tau^*[S](= \tau[S])$ (e1)

We call the compound variables in $\mathcal{W}^*$ *factor variables*. $\mathcal{P}^*$ is also referred to as *fe*($\mathcal{P}$). Given $c_k \in \mathcal{C}$, we may denote $c_k^* \in \mathcal{C}^*$ with *fe*(c_k). We observe that the compatibility constraint in $\mathcal{P}^+$ can be decomposed into two conditions. Given $\lambda(S)$ such that S = $\{x_{i_1}, \ldots, x_{i_m}\}$, we have,

(c1) $\bar{a} \in D^c(\lambda(S)) \Rightarrow \forall k \in \{1, \ldots, m\}, a_{i_k} \in D^c(x_{i_k})$
(c2) $a \in D^c(x_{i_k})$ for some $k \in \{1, \ldots, m\} \Rightarrow \exists \bar{a} \in D^c(\lambda(S)), a_{i_k} = a$

We will show that the compatibility constraints in $\mathcal{P}^+$ are actually implied and do not need to be posted explicitly.

Lemma 1 *Enforcing GAC on fe($\mathcal{P}$) imposes the condition $(c1)$ between a factor variable and each ordinary variable in its signature.*

Proof Consider $\lambda(S)$ where S = $\{x_{i_1}, \ldots, x_{i_m}\}$ and $\bar{a} \in D^c(\lambda(S))$. Because *fe*($\mathcal{P}$) is GAC, for any *fe*($c$) such that $\lambda(S) \in scp(fe(c))$, there is a valid support of $\bar{a}$ in $rel(fe(c))$. That is, $\exists \tau \in rel(fe(c))$ such that $\tau[\lambda(S)] = \bar{a}$. Since $\tau[\lambda(S)] = \tau[S]$, $\tau[x_{i_k}] = a_{i_k}$ for $1 \leq k \leq m$, which means a_{i_k} also has a valid support in $rel(fe(c))$. □

Lemma 2 *Enforcing GAC on fe($\mathcal{P}$) imposes the condition $(c2)$ between a factor variable and each ordinary variable in its signature.*

Proof Assume $a_{i_k} \notin \pi^c(\lambda(S), x_{i_k})$ for some a_{i_k}. This indicates that any $\bar{a}$ involving a_{i_k} must be absent from $D(\lambda(S)))$. Due to propagation, every τ in every $rel(fe(c))$ such that $\lambda(S) \in scp(fe(c))$ and $\tau[\lambda(S)] = \bar{a}$ would eventually become invalid. Because $\tau[S] = \tau[\lambda(S)] = \bar{a}$, $\tau[x_{i_k}] = a_{i_k}$. That means such τ is not a valid support of a_{i_k}. Because $D(\lambda(S))$ contains every compound values involving a_{i_k} from all c whose scope subsumes S, there is no other valid tuple τ' such that $\tau'[x_{i_k}] = a_{i_k}$. Hence, $a_{i_k} \notin D^c(x_{i_k})$ after the propagation converges. □

Theorem 2. *fe($\mathcal{P}$) is GAC if and only if $\mathcal{P}$ is FPWC.*

Proof Follows from Theorem 1, and Lemma 1, and 2. □

Theorem 3. *fe($\mathcal{P}$) is GAC if and only if fe($\mathcal{P}$) is FPWC.*

Proof As FPWC is both GAC and PWC, ($\Leftarrow$) is immediate. We will prove the ($\Rightarrow$) direction. Assume *fe*($\mathcal{P}$) is GAC. Let $\tau_i \in$ *fe*(c_i). Now consider another constraint *fe*(c_j) $\neq$ *fe*(c_i). If there is no factor variable in scp(*fe*(c_i)) $\cap$ scp(*fe*(c_j)), then PWC is trivial. Let f be the factor variable[1] in scp(*fe*(c_i)) $\cap$ scp(*fe*(c_j)) such that scp(*fe*(c_i)) $\cap$ scp(*fe*(c_j))$\setminus\sigma(f) = \{f\}$. Since *fe*($\mathcal{P}$) is GAC, $\tau_i[f]$ must have a valid support in *fe*(c_j). Call it τ_j. Because τ_i and τ_j agree on f, by definition of factor variable they must agree on $\sigma(f)$ too, which means they agree on $\sigma(f) \cup \{f\} = scp$(*fe*($c_i$)) $\cap$ scp(*fe*(c_j)). As a result, $\tau_i \bowtie \tau_j$ is well-defined as well as being a tuple extended from τ_i over scp(*fe*(c_i)) $\cup$ scp(*fe*(c_j)). Hence, *fe*($\mathcal{P}$) is PWC. □

Let *fe*^{k}($\mathcal{P}$) denote *fe*(*fe*(...*fe*($\mathcal{P}$)...)) (the FE is applied k times in a row), then

Corollary 1 *For all $k \geq 1$, $\mathcal{P}$ is FPWC if and only if* *fe*^{k}($\mathcal{P}$) *is GAC.*

Proof We consider $k = 2$ as other cases follow from induction. From Theorem 2 and Theorem 3, we have: $\mathcal{P}$ is FPWC iff *fe*($\mathcal{P}$) is FPWC. From this statement and the result of another application of the FE on it we derive: $\mathcal{P}$ is FPWC iff *fe*(*fe*($\mathcal{P}$)) is FPWC. From Theorem 3, *fe*(*fe*($\mathcal{P}$)) is FPWC iff *fe*(*fe*($\mathcal{P}$)) is GAC. □

This shows *fe*^{k}($\mathcal{P}$) for $k \geq 2$ is no different than *fe*($\mathcal{P}$) so applying the FE more than once in succession is pointless. A localized version of this corollary is given as follows.

Corollary 2 *Given any two constraints c_i and c_j, if there exists a factor variable $f \in scp(c_i) \cap scp(c_j)$ such that $scp(c_i) \cap scp(c_j) \setminus \sigma(f) = \{f\}$ then adding the factor variable f' whose signature is $\sigma(f) \cup \{f\}$ to the scopes of both constraints is futile.*

Property 1 *Running GAC on fe($\mathcal{P}$) can be $O(e^2)$ faster and use $O(e^2)$ smaller space than running eSTR2 on $\mathcal{P}$.*

Reasoning: eSTR2 [11] is an extension of STR2 [9] that enforces FPWC. The main difference between enforcing GAC on the FE and enforcing eSTR2 on the original network is the space and time associated with factor variables vs. those associated with the additional data structures for checking PWC in eSTR2. The overhead of running GAC on the FE depends on factor variables, whose number can be lower than the number of intersecting constraints. In eSTR2, the overhead depends on the number of intersecting constraints. If $\mathcal{P}$ consists of only constraints such that a single factor variable is common to all and that no other factor variable exists, the space and time complexity of the GAC on *fe*($\mathcal{P}$) is the same as those on $\mathcal{P}$. By contrast, the space and time of eSTR2 on $\mathcal{P}$ would be at least an order of $O(\binom{e}{2}) = O(e^2)$ larger. □

Property 2 *For any $c \in \mathcal{C}$, $|scp(c)| \leq |scp(fe(c))| \leq |scp(c)| + |\mathcal{C}| - 1$.*

The range is the result of the number of factor variables added. The lower bound is zero, when no other constraint's scope overlaps on more than two variables with $scp(c)$, whereas the upper bound is $|\mathcal{C}| - 1$ when every intersection with another constraint produces a new factor variable.

[1] There may be multiple factor variables if $\mathcal{P}$ itself is the factor encoding of another constraint network, which in turn is the factor encoding of another, and so on (see Corollary 1). The factor variable f is set to be the most recent one.

3.1 Example

We give an example of $\mathcal{P}^*$ and trace some GAC propagation on $\mathcal{P}^*$ in this section. Note that although relations in $\mathcal{P}^*$ are an extension of those in $\mathcal{P}$, the extension to factor variables can be implicit. The expression $\tau[S]$ in (e1) can be given as a function (i.e. the projection) that takes an input S rather than the actual result of the projection of τ on S. Such abstract extension of tuples is demonstrated in this section.

For brevity, compound variables and values are written as a concatenation of ordinary variables and values. Let $\mathcal{P}^* = (\mathcal{X} \cup \mathcal{W}^*, \mathcal{C}^*)$, where $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, $\mathcal{W}^* = \{x_1x_2,\ x_1x_2x_4\}$, $\mathcal{C}^* = \{c_1^*, c_2^*, c_3^*, c_4^*\}$ where $scp(c_1^*) = \{x_1, x_2, x_3,\ x_1x_2\}$, $scp(c_2^*) = \{x_1, x_2, x_4,\ x_1x_2,\ x_1x_2x_4\}$, $scp(c_3^*) = \{x_1, x_2, x_4, x_5,\ x_1x_2,\ x_1x_2x_4\}$, $scp(c_4^*) = \{x_2, x_6\}$. Relations of $\mathcal{P}$ are given as tables for c_i below ($rel(c_i^*)$ will be inferred from $rel(c_i)$). $D^c(x_1) = D^c(x_2) = D^c(x_4) = D^c(x_6) = \{a, b\}$, $D^c(x_3) = D^c(x_5) = \{a, b, c\}$, $D^c(x_1x_2) = \{aa, ab, bb\}$, $D^c(x_1x_2x_4) = \{abb, bba, bbb\}$.

c_1

x_1	x_2	x_3
a	a	a
a	b	a
a	b	c
b	b	b

c_2

x_1	x_2	x_4
a	b	b
b	b	a

c_3

x_1	x_2	x_4	x_5
a	b	b	a
b	b	a	b
b	b	b	c

c_4

x_2	x_6
a	a
b	b

We now look at some GAC propagation on this network. First, we consider whether $(x_1x_2x_4, bbb)$ is GAC. Let $\tau = \rho(c_3, 3) = (b, b, b, c)$. The value $(x_1x_2x_4, bbb)$ is GAC on c_3^* since $\rho(c_3^*, 3) = (\tau[x_1], \tau[x_2], \tau[x_4], \tau[x_5], \tau[x_1x_2], \tau[x_1x_2x_4]) = (b, b, b, c, bb, bbb)$ is found to be a valid support. But $(x_1x_2x_4, bbb)$ is not GAC on c_2^* because no tuple in $rel(c_2^*)$ involves bbb (i.e. $rel(c_2^*) = \{(a, b, b, ab, abb), (b, b, a, bb, bba)\}$), so bbb is removed from $D^c(x_1x_2x_4)$. Propagation leads back to the removal of c from $D^c(x_5)$ as $\rho(c_3^*, 3)$ is no longer valid because $\rho(c_3^*, 3)[x_1x_2x_4] = bbb \notin D^c(x_1x_2x_4)$. Next we look at (x_1x_2, aa). It has no valid support in c_2^* so aa will be removed from the domain of x_1x_2. Because $\rho(c_1^*, 1) = (a, a, a, aa)$, this tuple becomes invalid. Because $\rho(c_1^*, 1)$ is the only tuple involving (x_2, a) in $rel(c_1^*)$, (x_2, a) is no longer GAC on c_1^*. Value a is then removed from $D^c(x_2)$. Further propagation leads to the removal of (x_6, a).

4 The k-interleaved encoding

The k-interleaved encoding (kIL) [13] is closely related to the FE as both try to enlarge constraints with auxiliary variables that represent groups of existing variables. Enforcing GAC on the k-interleaved encoding is equivalent to enforcing kWC on the original network in addition to GAC. The following definitions are taken from [13].

Definition 1 (k-dual encoding). *Let $\mathcal{P} = (\mathcal{X}, \mathcal{C})$. The k-dual encoding of $\mathcal{P}$ is the constraint network $\mathcal{P}^{kd} = (\mathcal{X}^{kd}, \mathcal{C}^{kd})$ where:*

- *for each $c_i \in \mathcal{C}$, $\mathcal{X}^{kd}$ contains a variable x'_i where $D(x'_i) = \{1, \ldots, |rel(c_i)|\}$.*
- *for each subset S of k constraints of $\mathcal{C}$, $\mathcal{C}^{kd}$ 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).*

Definition 2 (Hybrid constraints). *Let $\mathcal{P} = (\mathcal{X}, \mathcal{C})$. The set of hybrid constraints $\phi(\mathcal{C})$ of $\mathcal{P}$ is the set $\{\phi(c_i) \mid c_i \in \mathcal{C}\}$ where:*

- $scp(\phi(c_i)) = scp(c_i) \cup \{x'_i\}$
- *for every* j^{th} *tuple* τ *of* $rel(c_i)$, τ' *is a tuple in* $rel(\phi(c_i))$ *such that* $\tau'[x'_i] = j$ *and* $\tau'[x] = \tau[x]$ *for each* $x \in scp(c_i)$

Definition 3 (k-interleaved encoding). *Let* $\mathcal{P} = (\mathcal{X}, \mathcal{C})$. *The* k*-interleaved encoding of* $\mathcal{P}$ *is the constraint network* $\mathcal{P}^{ki} = (\mathcal{X}^{ki}, \mathcal{C}^{ki}) = (\mathcal{X} \cup \mathcal{X}^{kd}, \phi(\mathcal{C}) \cup \mathcal{C}^{kd})$ *where* $(\mathcal{X}^{kd}, \mathcal{C}^{kd})$ *is the* k*-dual encoding of* $\mathcal{P}$ *and* $\phi(\mathcal{C})$ *the hybrid constraints of* $\mathcal{P}$.

For k = 2, enforcing GAC on the 2IL has the same pruning power as enforcing GAC on the FE, but the FE does not add any new constraint. We now look at an example from [13] for a comparison of the 2IL and the FE. Figure 1a shows three constraints from the original network. Figure 1b shows the FE for these constraints. A factor variable's domain of size d is normalized as $\{1, \ldots, d\}$. As a result, $D^c(xy) = D^c(uv) = \{11, 00, 01, 10\} = \{1, 2, 3, 4\}$. After GAC is established, $D^c(y)$ becomes $\{1\}$ and $D^c(v)$ becomes $\{0\}$. Figure 1c shows the 2IL of 1a [13]. This example shows that while enforcing GAC on the kIL gives identical $D^c(y)$ and $D^c(v)$ to Figure 1b, the kIL can take a longer chain of propagation to do so.

x	y	u	v
1	1	1	0
0	0	0	1
0	1	0	0
1	0	1	1

x	y
1	1
0	0
0	1

u	v
1	1
1	0
0	0

x	y	u	v	xy	uv
1	1	1	0	1	4
0	0	0	1	2	3
0	1	0	0	3	2
1	0	1	1	4	1

x	y	xy
1	1	1
0	0	2
0	1	3

u	v	uv
1	1	1
1	0	4
0	0	2

x	y	u	v	x'_1
1	1	1	0	1
0	0	0	1	2
0	1	0	0	3
1	0	1	1	4

x	y	x'_2
1	1	1
0	0	2
0	1	3

u	v	x'_3
1	1	1
1	0	2
0	0	3

x'_1	x'_2
1	1
2	2
3	3

x'_1	x'_3
1	2
3	3
4	1

(a) Original (b) Factor encoding of (a) (c) 2-interleaved encoding of (a)

Fig. 1: Comparison of two encodings

We compare the complexity of the FE and the kIL as follows. For simplicity, we assume there are e constraints of arity r, each associated with a table containing t tuples and that every pair of constraints shares at least two variables in their scopes.

Property 3 *The extra cells added to the tables by the FE ranges from* $O(et)$ *to* $O(e^2t)$.

Proof In the best case there is only one factor variable. Each constraint will be extended with an extra column so the total extra space is $O(et)$. In the worst case, every pair of constraint produces one additional factor variable. Each of these factor variables will appear in two different tables. Thus, the total is $O(2t\binom{e}{2}) = O(e^2t)$. □

Because an optimal GAC algorithm traverses every cell of every table in the worst case, the worst-case time complexity of GAC on the FE is thus between $O(ert)$ (i.e. no asymptotic difference) and $O(ert + e^2t) = O(e^2t)$ (i.e. assuming $e > r$).

Property 4 *The extra cells added to the tables by the* k*IL is* $O(\binom{e}{k}t^k)$

Proof Each constraint has an extra column for indexing so the space is et. For every subset of $\mathcal{C}$ of size k, a join table of arity k is created. The total space is therefore $O(et + \binom{e}{k}t^k) = O(\binom{e}{k}t^k)$. □

For k = 2, this space becomes $O(e^2t^2)$. As far as GAC is concerned, the 2IL is thus a factor of t more expensive in the worst case than the FE.

5 Enforcing k-wise consistency through reduced join tables

Given *fe*($\mathcal{P}$), we may post additional constraints so that GAC may also enforce kWC. These new constraints are created from a group of existing constraints and this section studies their effect on the consistency level.

Given $C = \{c_{i_1}, \ldots, c_{i_k}\}$ in $\mathcal{P}$ where $k \geq 3$, we define the following notation:

- *mult*$(C) = \{\lambda(S) \mid \lambda(S) \in \mathcal{W}^* \wedge S = scp(c_{i_j}) \cap scp(c_{i_l})$ for $1 \leq j < l \leq k\}$
- *sing*$(C) = \{x \mid x \in \mathcal{X} \wedge \{x\} = scp(c_{i_j}) \cap scp(c_{i_l})$ for $1 \leq j < l \leq k\}$
- *join*$(C) = rel(c_{i_1}) \bowtie \ldots \bowtie rel(c_{i_k})$,

Definition 4. *Given a set C of k constraints ($k \geq 3$), the* factor-reduced join *of C (frj(C)) is a constraint constructed as follows. Let |mult(C)| = o, and |sing(C)| = p,*

- *scp(frj(C)) = mult(C) $\cup$ sing(C) = $\{\lambda(S_1), \ldots, \lambda(S_o)\} \cup \{x_{j_1}, \ldots x_{j_p}\}$*
- *rel(frj(C)) = $\{(\tau[S_1], \ldots, \tau[S_o], \tau[x_{j_1}], \ldots, \tau[x_{j_p}]) \mid \tau \in$ join(C)$\}$*

The factor-reduced join is not a projection of *join*(C) as its scope may include factor variables. Rather, it can be viewed as a projection of $\bowtie_{c \in C}$ *fe*(c). In any case, since it is derived from the join of C, its pruning power cannot be greater.

It should be noted that *frj*(C) may end up having the same scope as another existing constraint or another *frj* constraint. For instance, let C_1 = $\{c_1(x_1, x_2, x_3)$, $c_2(x_1, x_2, x_4)$, $c_3(x_1, x_5)\}$ and C_2 = $\{c_4(x_1, x_2, x_6)$, $c_5(x_1, x_2, x_7)$, $c_3(x_1, x_5)\}$. Let $y_1 = x_1x_2$, it follows that *scp*(*frj*(C_1)) = $\{x_1, y_1\}$ = *scp*(*frj*(C_2)). This can also happen in the case where no factor variables are formed by the FE. For instance, let C_1 = $\{c_1(x_1, x_2)$, $c_2(x_2, x_3)$, $c_3(x_3, x_4)\}$. Then *scp*(*frj*(C_1)) = $\{x_2, x_3\}$ = *scp*(c_2). Both cases can be handled by merging constraints with the same scope afterwards.

We assume that every constraint in C must be relevant. Namely, given $c \in C$ there must exist at least one other $c' \in C$ such that $|scp(c) \cap scp(c')| \geq 1$.

Property 5 *The arity of frj(C) ranges from 2 to $\binom{|C|}{2}$.*

The *fe*($\mathcal{P}$) with the additional constraints *frj*(C) for every group C of size k is called the factor encoding of $\mathcal{P}$ for k-wise consistency (FKWC), also denoted by *fkwc*($\mathcal{P}, k$).

Property 6 *Enforcing GAC on fkwc($\mathcal{P}$,k) is strictly weaker than enforcing both FPWC and kWC on $\mathcal{P}$ and strictly stronger than enforcing FPWC on $\mathcal{P}$.*

We show this by an example. Consider the constraints in Figure 2a and their factor encodings in Figure 2b, where $y_1 = x_2x_3$. The networks in both figures are PWC. The join of the three original constraints is given in Figure 2c. The projection of *join*(C) onto each of the original constraint makes the following tuples 3-wise inconsistent: $(1, 1, 0, 0) \in rel(c_1)$, $(1, 0, 1) \in rel(c_2)$, and $(1, 0, 1) \in rel(c_3)$. Now consider the *frj*(C) in Figure 2d. GAC on {*frj*(C),*fe*(c_1),*fe*(c_2),*fe*(c_3)} leads to the inconsistency of $(1, 1, 0, 0, 2) \in rel($*fe*$(c_1))$ and $(1, 0, 1, 2) \in rel($*fe*$(c_2))$, but not $(1, 0, 1) \in rel($*fe*$(c_3))$.

Although the *fkwc*($\mathcal{P}$,k) encoding is only partial kWC, it subsumes *fe*($\mathcal{P}$) so FPWC is guaranteed by GAC. Together with the fact that kWC implies (k-1)WC, we have

Property 7 *Enforcing GAC on fkwc($\mathcal{P}$,k) is strictly weaker than enforcing GAC on the kIL of $\mathcal{P}$ for $k \geq 3$.*

Theorem 4. *$\mathcal{Q}$ = fe(fkwc($\mathcal{P}, k$))) is GAC if and only if $\mathcal{P}$ is FPWC and kWC.*

c_1			
x_1	x_2	x_3	x_4
0	0	0	0
0	0	1	1
1	1	0	0

c_2		
x_2	x_3	x_5
0	0	0
0	1	1
1	0	1

c_3		
x_4	x_5	x_6
0	0	0
1	0	1
1	1	0

(a) Original: $C = \{c_1, c_2, c_3\}$

fe(c_1)				
x_1	x_2	x_3	x_4	y_1
0	0	0	0	0
0	0	1	1	1
1	1	0	0	2

fe(c_2)			
x_2	x_3	x_5	y_1
0	0	0	0
0	1	1	1
1	0	1	2

fe(c_3)		
x_4	x_5	x_6
0	0	0
1	0	1
1	1	0

(b) Factor-encoded constraints

join(C)					
x_1	x_2	x_3	x_4	x_5	x_6
0	0	0	0	0	0
0	0	1	1	1	0

(c) The join of constraints in C

frj(C)		
x_4	y_1	x_5
0	0	0
1	1	1

(d) The factor-reduced join of C

fe(*fe*(c_1))					
x_1	x_2	x_3	x_4	y_1	z_1
0	0	0	0	0	0
0	0	1	1	1	1
1	1	0	0	2	2

fe(*fe*(c_2))				
x_2	x_3	x_5	y_1	z_2
0	0	0	0	0
0	1	1	1	1
1	0	1	2	2

fe(*fe*(c_3))			
x_4	x_5	x_6	z_3
0	0	0	0
1	0	1	1
1	1	0	2

fe(*frj*(C))					
x_4	y_1	x_5	z_1	z_2	z_3
0	0	0	0	0	0
1	1	1	1	1	2

(e) The FE of (b) and (d) where *fe*(*fe*(c_i)) denotes the FE of constraints from (b) with *frj*(C).

Fig. 2: The pruning power of GAC on (b) + (d) lies between FPWC and FPWC + 3-wise consistency on (a), whereas GAC on (e) is equal to FPWC + 3-wise consistency on (a).

Sketch of Proof: Consider $C = \{c_{i_1}, \ldots, c_{i_k}\}$. It is clear that *join*(C) forces kWC on C through GAC. *fe*(*frj*(C)) $\in \mathcal{Q}$ represents all the articulation points of *join*(C) and we will show that both have the same restricting effect on the rest of the network by showing that the "missing columns" can be "rebuilt" via GAC. The proof for ($\Leftarrow$) is omitted for lack of space. Assume $\mathcal{Q}$ is GAC. Let x be a variable in $scp(\textit{join}(C)) \setminus (\bigcup_{\lambda(S) \in \textit{mult}(C)} S) \setminus \textit{sing}(C)$. It follows that there is exactly one constraint $c_{i_j} \in C$ such that $x \in scp(c_{i_j})$. Suppose $a \in D^c(x)$. Because $\mathcal{Q}$ is GAC, so is a. By definition, there exists a valid tuple $\tau_{i_j} \in rel(\textit{fe}(\textit{fe}(c_{i_j})))$ such that $\tau_{i_j}[x] = a$. Let $H_{i_j} = scp(\textit{fe}(c_{i_j})) \cap scp(\textit{frj}(C))$, $|H_{i_j}| \geq 1$. Because $\mathcal{Q}$ is a factor encoding, there exists a variable $\lambda(H_{i_j})$ in both $scp(\textit{fe}(\textit{fe}(c_{i_j})))$ and $scp(\textit{fe}(\textit{frj}(C)))$ ($\lambda(H_{i_j})$ is either an ordinary or a factor variable). Since H_{i_j} too is GAC, τ_{i_j} is guaranteed to be extendable to *fe*(*frj*(C)). Let φ be such a tuple in *fe*(*frj*(C)) such that $\tau_{i_j} \bowtie \varphi$ is a tuple over $scp(\textit{fe}(c_{i_j})) \cup scp(\textit{frj}(C))$. For each $c_{i_l} \in C \setminus \{c_{i_j}\}$, let $H_{i_l} = scp(\textit{fe}(c_{i_l})) \cap scp(\textit{frj}(C))$. By the same argument, there exists a valid tuple τ_{i_l} in *fe*(*fe*(c_{i_l})) such that $\tau_{i_l}[H_{i_l}] = \varphi[H_{i_l}]$. The join of φ, τ_{i_j}, and every such τ_{i_l} would become a valid support of a in $J = (\bowtie_{c \in C} \textit{fe}(\textit{fe}(c))) \bowtie \textit{fe}(\textit{frj}(C))$. Because the projection of J on $scp(\textit{join}(C))$ is *join*(C) and a is arbitrary, the column x in $scp(\textit{join}(C))$ is thus the same as $D^c(x)$. □

Figure 2e shows another application of the factor encoding on top of *fkwc*($\mathcal{P}, k$), where $z_1 = x_4 y_1$, $z_2 = x_5 y_1$, and $z_3 = x_4 x_5$. GAC on this network would lead to the inconsistency of $(1, 0, 1, 1)$ in the third table. Since $x_2 x_3 y_1$ is redundant according to Corollary 2 we do not add it to $scp(\textit{fe}(\textit{fe}(c_1)))$ and $scp(\textit{fe}(\textit{fe}(c_2)))$.

Property 8 *The arity of fe*(*frj*(C)) *ranges from 2 to* $\binom{|C|}{2} + |\mathcal{C}|$.

Proof The reasoning is similar to the one for Property 2, but here *fe*(*frj*(C)) is not necessarily part of $\mathcal{C}$ so the bound on the number of constraints that it may interact with is $|\mathcal{C}|$ not $|\mathcal{C}| - 1$. Coupled with Property 5, we have,

$$2 \leq |scp(\mathit{frj}(C))| \leq |scp(\mathit{fe}(\mathit{frj}(C)))| \leq |scp(\mathit{frj}(C))| + |\mathcal{C}| \leq \binom{|C|}{2} + |\mathcal{C}| \qquad \square$$

Figure 2e demonstrates: we have $|\mathcal{C}| = |C| = 3$, so the upper bound on the arity of $\mathit{fe}(\mathit{frj}(C))$ is $\binom{|C|}{2} + |\mathcal{C}| = \binom{3}{2} + 3 = 6$, which happens to be the actual arity of $\mathit{fe}(\mathit{frj}(C))$.

6 Experiments

In this section, we present experimental results on the effectiveness of the FE and the FKWC in comparison with the kIL and an FPWC algorithm. We will use kFE to denote $\mathit{fkwc}(\mathcal{P}, k)$. Benchmarks are drawn from the CSP solver competition[2] in addition to randomly generated problems. The experiments were conducted on a 2.6GHz quad-core Intel Core i7 on OS X 10.8. The converters take an input in the XCSP format and output the result as another text file. As such, we are not restricted to any particular GAC algorithm and we shall test the encoding on multiple GAC algorithms. Like [7, 11], the search employed the $dom/ddeg$ variable ordering heuristic and the *lex* value ordering. We used AbsCon [15] as the solver. Conversion time is limited to 30 minutes while memory is limited to 8GB for both the converters and the solver.

Because the kIL and the FE augment the original constraints' scope with new variables, the location to which they are inserted has to be considered. Two natural choices are the front and the back. The front leads to slightly faster running time in our experiments, with a big difference on some problem instances, such as when multi-valued decision diagrams (MDDs) are involved. To simplify the presentation, experiments therefore involve only the front insertion. After the conversion the FE and FKWC converters may have to re-sort the tuples since the front insertion may disrupt the ordering of the input that is already sorted. The reason is that AbsCon happens to need sorted relations for some GAC algorithms such as MDDc [3]. The kIL conversion avoids this overhead because it maintains the tuple ordering of the input. For the 2IL any pair of constraints is joined only if their scopes share more than one variable. Unused dual variables are discarded from the output. For instance, consider the 2IL comprising of $c_1(x, y, z, v_1)$, $c_2(x, y, w, v_1)$, $c_3(w, z, v_3)$, where v_1, v_2, v_3 are the dual variables associated with c_1, c_2, c_3 and the rest are ordinary variables. Only v_1 and v_2 are joined to form a new constraint $c_4(v_1, v_2)$ as c_1 and c_2 share x and y. Because v_3 is not involved in any constraint it will be removed from $\mathcal{X}$ and c_3.

Table 1 shows the mean results on some series of benchmarks while Table 2 shows the results from selected instances. Five algorithms were tested: GACva [12], MDDc [3], STR3[10], STR2 [9], and an AbsCon's implementation[3] of FPWC based on STR, which we will call Fabs. Fabs is not eSTR2w [11] but can be regarded as a variant of eSTRw (or FPWC-STR^w). It exhibits a profile similar to that of eSTR2w when compared to STR2 on common benchmarks, except in a few cases where the difference in performance with respect to STR2 is noticeably smaller (e.g. *aim*) or larger (e.g. *rand-10-20-10*). Among the four GAC algorithms, GACva and STR3 are generally slower than MDDc and STR2 so for succinctness they do not appear in the main results in

[2] Available at http://www.cril.univ-artois.fr/CSC09. The modified renault problems with tables all positive are taken from http://becool.info.ucl.ac.be/resources/positive-table-constraints-benchmarks.

[3] Available in AbsCon 1.418.

Table 1. Their performance on some representative instances can be seen in Table 2. All algorithms were run on the original instance and its two encodings for FPWC: the FE and the 2IL. There are three variants depending on how variables are handled during search. FE-O and 2IL-O (*pref-orig*) force the variable ordering heuristic to choose from the set of original variables until all of them are instantiated before choosing from the set of auxiliary (compound) variables. FE-A and 2IL-A (*pref-aux*) are the opposite, where preference is given to the auxiliary variables. FE-E and 2IL-E (*pref-equal*) give equal treatment to all variables with respect to $dom/ddeg$.

In both tables, tC gives the running time of the converters in seconds, while nV is the number of the variables, e.g. there are 100 ordinary variables in *a2*, 99 extra factor variables in its FE, and 127 extra dual variables in its 2IL. Best times and nodes are set in bold. A node count of zero means unsatisfiability is detected before the search starts. In Table 1, *SS* stands for solving strategy, the combination of GAC algorithm and encoding (if applicable). STR2 is the main GAC algorithm for solving various encodings unless specified otherwise (MDDc is ill-suited to the encodings as will be explained later). (#n) is the number of instances tested in the series. Instances that were not solved within 60 minutes by STR2 (the baseline) or exceeded the memory limit on any solving strategy are excluded, otherwise there is no time limit. Trivial instances (solved within 1 second) of the modified renault benchmark (*modRenault*) are also excluded (17 out of 50). The *mdd-*r*-*n*-*d series are randomly generated, based on the RD model [19], by building an MDD in a post-order manner with probability 0.5 that a previously created sub-MDD is reused [3]. The parameters are: arity (r), number of variables (n), and domain size (d). In Table 2, *ENC* is the encoding used. The columns GACva, MDDc, STR3, STR3, and Fabs give their running times. As Fabs reaches the same node count as GAC on the respective FE/2IL encoding, cells on these rows are left blank.

It is worth mentioning that $dom/ddeg$ may not produce the same search tree in both the original network and in its encoding, even when only the ordinary variables are instantiated. Because the dynamic degree counts the number of constraints in which there are at least two uninstantiated variables, the fact that an encoding's scopes have more variables may steer $dom/ddeg$ to pick a different variable than in the original problem. As a result, a weaker consistency may generate lower nodes than a stronger one (e.g. Fabs's node count on *r19* is lower than GAC's on the FE-O and the 2IL-O). Another source of the difference in node count lies in how AbsCon explicitly instantiates all variables, even the ones that are already singletons. For example, both encodings for *a2* are backtrack-free for GAC, but the node count for the FE is 199 because there are 99 more variables, while the 2IL's is 277 because there are 127 additional dual variables.

We make the following observations on these data:

– The FE's conversion time is mostly inconsequential compared to the solver's running time whereas the 2IL's can be a lot more expensive as it is based on the join of tables. The conversion time of the FE can be improved since we have not made an attempt to optimize our converter. For one thing, it always re-sorts relations before writing the output regardless of the solver's requirement.

– The fastest GAC on the original problem is either MDDc or STR2. The best variant of the FE largely improves the running time of STR2 and STR3, while it may help or hurt MDDc and Fabs. The 2IL has mixed results and the conversion can be very slow due

series	SS	tC	nV	time	nodes
rand-3-20-20-fcd	STR2	–	20	39.61	130,327
(#50)	MDDc	–	20	**21.00**	130,327
	Fabs	–	20	34.28	37,727
	FE-O	0.21	+45	22.07	40,289
	FE-A	0.21	+45	32.94	71,568
	FE-E	0.21	+45	22.82	**36,195**
	2IL-O	2.83	+55	68.79	40,299
	2IL-A	2.83	+55	105.40	59,885
	2IL-E	2.83	+55	73.09	36,227
rand-3-20-20	STR2	–	20	83.27	256,958
(#50)	MDDc	–	20	**40.94**	256,958
	Fabs	–	20	74.39	83,529
	FE-O	0.22	+45	41.76	74,825
	FE-A	0.22	+45	54.85	108,696
	FE-E	0.22	+45	42.53	**66,850**
	2IL-O	2.89	+55	130.38	74,830
	2IL-A	2.89	+55	269.64	137,301
	2IL-E	2.89	+55	129.68	66,853
dubois	STR2	–	71	528.45	100.27M
(#8)	MDDc	–	71	541.67	100.27M
	Fabs	–	71	298.35	75.20M
STR2 {	FE-O	0.00	+2	92.88	41.78M
STR2 {	FE-A	0.00	+2	198.42	66.85M
STR2 {	FE-E	0.00	+2	63.91	16.71M
Fabs +	FE-E	0.00	+2	**50.90**	16.71M
MDDc +	FE-E	0.00	+2	64.63	16.71M
	2IL-O	0.00	+4	96.74	41.78M
	2IL-A	0.00	+4	196.88	66.85M
	2IL-E	0.00	+4	73.79	16.71M
	4FE-O	0.08	+2	81.88	41.78M
	4FE-A	0.08	+2	194.64	66.85M
	4FE-E	0.08	+2	192.52	66.85M
	4IL-O	0.08	+47	89.80	29.25M
	4IL-A	0.08	+47	76.61	20.89M
	4IL-E	0.08	+47	65.68	**8.36M**
aim-200	STR2	–	200	45.54	637,085
(#6)	MDDc	–	200	32.95	637,085
	Fabs	–	200	25.42	377,682
	FE-O	0.03	+354	3.85	32,354
	FE-A	0.03	+354	**0.85**	1,767
	FE-E	0.03	+354	2.96	11,397
	2IL-O	0.03	+551	4.36	32,552
	2IL-A	0.03	+551	3.45	22,968
	2IL-E	0.03	+551	3.72	11,594
	3FE-O	79.03	+354	1.39	5,561
	3FE-A	80.52	+354	2.79	19,002
	3FE-E	79.23	+354	1.00	1,434
	3IL-O	31.32	+769	34.73	4,854
	3IL-A	31.28	+769	8.32	690
	3IL-E	31.64	+769	7.73	**683**

instance	SS	tC	nV	time	nodes
rand-8-20-5	STR2	–	20	**12.50**	101,301
(#20)	MDDc	–	20	22.26	101,301
(2IL T/O)	Fabs	–	20	32.74	18,709
	FE-O	5.87	+130	12.57	5,302
	FE-A	5.87	+130	22.64	**3,111**
	FE-E	5.87	+130	12.73	4,985
mdd-5-15-7	STR2	–	15	18.48	50,402
(#30)	MDDc	–	15	**5.95**	50,402
(e = 42)	Fabs	–	15	36.26	3,996
(t = 8403)	FE-O	1.05	+175	11.80	1,569
(2IL M/O)	FE-A	1.05	+175	13.63	1,816
	FE-E	1.05	+175	12.33	**1,512**
mdd-7-25-4	STR2	–	25	79.18	231,364
(#10)	MDDc	–	25	**26.19**	231,364
(e = 50)	Fabs	–	25	287.36	34,636
(t = 8192)	FE-O	1.85	+466	79.95	12,037
(2IL M/O)	FE-A	1.85	+466	71.90	39,366
	FE-E	1.85	+466	73.99	**10,523**
mdd-9-30-3	STR2	–	30	73.16	349,073
(#10)	MDDc	–	30	**39.00**	349,073
(e = 47)	Fabs	–	30	396.88	66,109
(t = 9,841)	FE-O	2.80	+723	83.68	12,963
(2IL M/O)	FE-A	2.80	+723	79.79	23,578
	FE-E	2.80	+723	84.28	**10,603**
rand-10-20-10	STR2	–	20	0.64	830
(#20)	MDDc	–	20	2.06	830
	Fabs	–	20	**0.60**	**0**
	FE-O	0.24	+10	0.69	**0**
	FE-A	0.24	+10	0.69	**0**
	FE-E	0.24	+10	0.70	**0**
	2IL-O	4.37	+5	1.41	**0**
	2IL-A	4.37	+5	1.45	**0**
	2IL-E	4.37	+5	1.45	**0**
dag-rand	STR2	–	23	17.48	57,969
(#25)	MDDc	–	23	123.83	57,969
	Fabs	–	23	12.56	**0**
(2IL T/O)	FE-O	14.07	+120	9.45	**0**
	FE-A	14.07	+120	9.30	**0**
	FE-E	14.07	+120	**9.12**	**0**
modRenault	STR2	–	110	317.18	6.40M
(#12)	MDDc	–	110	295.45	6.40M
	Fabs	–	110	2.19	**30**
	FE-O	0.71	+102	**1.19**	54
	FE-A	0.71	+102	1.22	58
	FE-E	0.71	+102	1.20	53
	2IL-O	81.16	+148	158.40	66
	2IL-A	81.45	+148	159.93	1023
	2IL-E	80.96	+148	159.14	2411

Table 1: Mean results for selected benchmarks. T/O indicates the converter was timed out. M/O is out-of-memory failure. M stands for millions. For the mdd series, e is the number of constraints while t is the number of tuples in a relation.

instance	ENC	tC	nV	nodes	GACva	MDDc	STR3	STR2	Fabs	nodes
rand-3-20-20-60-632-19	None	–	20	252,803	49.15	**36.59**	82.39	73.05	64.94	**74,509**
(abbrv. as "r19")	FE-O	0.22	+47	87,674	119.53	197.73	90.55	49.78	226.84	
	FE-A	0.22	+47	145,296	145.12	296.61	128.35	73.93	301.83	
	FE-E	0.22	+47	77,483	125.08	197.09	97.32	53.58	207.95	
	2IL-O	3.03	+57	87,674	258.08	341.64	222.87	143.26	218.48	
	2IL-A	3.03	+57	147,978	416.86	488.22	340.66	230.14	395.93	
	2IL-E	3.03	+57	77,483	245.40	355.73	222.59	156.22	243.80	
rand-3-20-20-60-632-26	None	–	20	442,871	74.65	67.24	147.71	127.00	35.34	29,765
(abbrv. as "r26")	FE-O	0.21	+48	34,200	53.73	82.02	32.71	19.54	72.07	
	FE-A	0.21	+48	**23,498**	24.58	52.54	17.64	**10.95**	40.86	
	FE-E	0.21	+48	30,957	53.02	85.44	32.94	19.37	72.55	
	2IL-O	2.73	+57	34,209	117.34	159.62	97.49	62.93	93.67	
	2IL-A	2.73	+57	31,907	117.13	156.49	112.07	66.98	90.20	
	2IL-E	2.73	+57	30,966	117.26	159.61	98.58	63.29	92.36	
dag-rand-1	None	–	23	43,994	74.37	109.04	259.00	15.52	11.68	**0**
(2IL T/O)	FE-O	14.28	+120	**0**	18.44	15.39	12.30	**9.14**	20.57	
	FE-A	14.28	+120	**0**	18.66	15.53	12.54	9.24	21.91	
	FE-E	14.28	+120	**0**	18.74	15.65	13.79	9.91	20.47	
rand-8-20-5-18-800-7	None	–	20	11,063	4.75	4.94	32.95	**4.43**	8.81	980
(2IL T/O)	FE-O	5.71	+128	573	22.77	M/O	10.82	5.98	M/O	
	FE-A	5.71	+128	**177**	6.79	M/O	5.93	4.47	M/O	
	FE-E	5.71	+128	546	23.00	M/O	11.40	6.27	M/O	
aim-100-1-6-sat-2	None	–	100	95.79M	498.67	446.68	1015.90	577.23	163.12	23.11M
(abbrv. as "a2")	FE-O	0.01	+99	**199**	**0.43**	0.50	0.48	0.45	0.47	
	FE-A	0.01	+99	**199**	0.44	0.50	0.46	**0.43**	0.45	
	FE-E	0.01	+99	**199**	0.44	0.49	0.46	0.46	0.47	
	2IL-O	0.00	+127	227	0.47	0.48	0.52	0.49	0.51	
	2IL-A	0.00	+127	227	0.47	0.48	0.49	0.48	0.52	
	2IL-E	0.00	+127	227	0.46	0.47	0.52	0.49	0.51	
mdd-5-15-7-inst-1	None	–	15	9,975	3.56	**2.13**	10.39	4.33	8.88	694
(2IL M/O)	FE-O	1.05	+190	594	16.58	91.49	5.84	4.11	52.74	
	FE-A	1.05	+190	1,383	35.54	201.29	12.10	9.33	155.91	
	FE-E	1.05	+190	**572**	16.50	88.07	6.01	4.35	52.40	

Table 2: Results from selected instances.

to the join. The FE clearly outperforms the 2IL on the same variant and benchmark, but enforcing MDDc on the original problems is frequently faster than any solving strategy (e.g. *rand-3-20-20* and the *fcd* variation). The implication here is that switching GAC algorithm may improve the running time better than equipping a GAC algorithm with stronger consistency. Experiments in previous works [11, 13, 16] neither considered MDDc nor included more than one GAC algorithm in the same study.

– For MDD compression, a larger scope is associated with lower probability of getting well-compacted MDDs. Any transformation which enlarges the scope may be unfavorable to MDDc. This is especially true with the kIL, which interferes directly with the compression by assigning different index to different tuples. The FE too causes the same problem, but to a lesser extent. However, the pruning from FPWC can more than compensate for this drawback in many cases (e.g. *dubois, dag-rand-1*), although it is not enough to win over STR2 on the same encoding. Since auxiliary variables are put

in front of the scope, they will be placed on top of the MDDs by MDDc and this makes the pruning from FPWC more effective. By comparison, putting auxiliary variables in the back of the scope lessens the impact of FPWC to the point where running MDDc on an encoding is always worse off.

– Due to stronger consistency, maintaining Fabs leads to a lower node count than maintaining GAC during search, but the lower number of nodes does not always translate to faster running time. Fabs can be faster or slower than STR2. By contrast, all variants of the FE are faster than Fabs although the node count can be higher.

– When a problem does not present an opportunity for additional pruning beyond GAC, running a stronger algorithm is counterproductive. Given that FPWC is both GAC and PWC, as the FE and the 2IL already builds in PWC propagation into the encoding, the portion of an FPWC algorithm that administers PWC becomes useless and simply incurs overhead when executed. Running an FPWC algorithm on the encoding therefore gets the same number of nodes as running any GAC algorithm on the encoding. It is interesting that the FE can make Fabs faster in some cases. The reason is that Fabs enforces only partial FPWC while the encoding provides complete FPWC. When Fabs's pruning capability happens to reach the level of complete FPWC on the original problem (i.e. its node count is already the lowest or not too much higher) running it on the FE would be slower (e.g. *r19, dag-rand-1*). Otherwise if Fabs's node count is considerably larger, that means there is still room for improvement and running Fabs on the FE (or 2IL) could make it faster (e.g. *dubois, a2*). On *dubois*, the combination of Fabs and FE-E is the fastest, offering an order-of-magnitude improvement over STR2.

– Variable preferences have a strong influence on the performance: the best can be twice as fast and/or halves the node count of the worst. Wide fluctuation also exists within the same series (e.g. in Table 2 FE-A is the best on *r26* but the worst on *r19*). Generally *pref-equal* has an advantage over *pref-orig*, while *pref-aux* is consistently the worst (FE-A on *aim* is the exception). This pattern holds for both the FE and the 2IL.

– As is the case with Fabs, the node count of various encodings does not correlate well with the running time. However, too many overlapping constraints or factor variables clearly has an adverse effect on the running time. The three *mdd* series illustrate. As arity and number of variable increases, so does the number of overlapping constraints and factor variables. Keep in mind that the latter's number can be lower than the former's. For example, the instance *mdd-9-30-3-inst-1* has 47 relations, so the maximum number of intersecting constraints is $\binom{47}{2}$ = 1081, whereas the actual number is 930 and the number of factor variables in the FE is 718. The ratio of the number of factor variables to the number of original variables goes from 11.67 for *mdd-5-15-7* to 18.64 for *mdd-7-25-4* to 24.1 for *mdd-9-30-3*. The ratio of Fabs's running time to STR2's increases accordingly from 1.96 to 3.63 to 5.42. The ratio of the FE's running time increases too, but at a lower pace of 0.54, 0.93, and 1.51 respectively. We also experimented with restricting the number of factor variables allowed in the FE for the mdd series but this does not improve the running time.

We have performed initial experiments with the FKWC and compare it with the kIL. For $k \geq 3$, [13] suggested the cycle heuristic to reduce the number of constraints: each constraint must share at least one variable with the previous and the next constraint in a circular manner. Our converters for the kIL and the FKWC employ this heuristic. Both

the kIL and the FKWC are not practical beyond small k (3 or 4) since they are based on join which suffers from exponential growth in computation. The 3IL and 3FE are either timed out or ran out of memory on all the benchmarks in Table 1 except for *dubois* and *aim-200*. On *dubois*, no new constraint is created by the 3IL and the only constraints created by the 3FE are universal (where every combination of value is allowed) so they are useless and ignored. The 4FE does not improve on the FE. The 4IL is better than the 2IL and has the best node count but it is still slower than the FE. Similarly, the 3FE and the 3IL brings down the node count for *aim* but does not improve the running time. We also tried other benchmarks from the solver competition but most exceeded time or memory limit for conversion. Some benchmarks, such as *pret* or *ramsey*, produce only universal constraints for the 3FE. For the benchmarks that can be converted, we found the FKWC to be slower than the FE although the node count is lower.

7 Conclusion

We have introduced a new encoding for non-binary constraint networks that enables stronger consistencies to be acquired through GAC. Thus, this allows stronger consistencies to be incorporated into existing (state-of-the-art) CP solvers. Our experiments suggest FE to be the better method for achieving FPWC than both the 2IL and AbsCon's FPWC algorithm. Unlike specialized FPWC algorithms which are usually slower than GAC when there is little or no overlapping constraint, the preprocessors like the FE or the 2IL converter do not suffer from such computational overhead. Unlike the 2IL which joins constraints to achieve PWC, the FE is more precise and does not require any new constraint to be posted. As a converter, the FE benefits from flexibility: any solver using any GAC algorithm can be used as long as it is able to read the file in the specified format. At the same time, passing information to the solver this way can become a significant expense when large files are involved. Integrating the converter with the solver would eliminate this problem. As for the encodings for general kWC, we found they are not as effective as the FE. Similar to the kIL, the FKWC encoding has limited practical benefits due to the high cost of joins in both time and space and the need for good heuristics that pick only the useful pieces from the large number of possible joins. Success hinges on fine-tuning these heuristics and implementing better join algorithms. Constructing the *frj* constraints directly through search [8] instead of deriving them from join is also less expensive and could be examined in future works.

The MDDc algorithm is faster than STR2 on some sets of benchmarks but its performance on the FE is generally poor due to the factor variable's larger domains and the drop in compression rate as arity increases. Modifying the MDDc algorithm itself to make it aware of factor variables is a promising direction.

Acknowledgments

We thank Christophe Lecoutre for the permission to use AbsCon in our experiment. This work has been supported by grant MOE2012-T2-1-155.

References

1. Fahiem Bacchus, Xinguang Chen, Peter van Beek, and Toby Walsh. Binary vs. non-binary constraints. *AIJ*, 140(1–2):1–37, 2002.
2. Christian Bessière, Kostas Stergiou, and Toby Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6–7):800–822, 2008.
3. Kenil C. K. Cheng and Roland H. C. Yap. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, 2010.
4. Ian P. Gent, Chris Jefferson, Ian Miguel, and Peter Nightingale. Data structures for generalised arc consistency for extensional constraints. In *Proceedings of AAAI-07*, pages 191–197, Vancouver, Canada, 2007.
5. Marc Gyssens. On the complexity of join dependencies. *ACM Transactions on Database System*, 11(1):81–108, 1986.
6. P. Janssen, P. Jegou, B. Nouguier, and M. C. Vilarem. 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*, pages 420–427, 1989.
7. Shant Karakashian, Robert Woodward, Berthe Y. Choueiry, Steven Prestwich, and Eugene C. Freuder. A partial taxonomy of substitutability and interchangeability. In *CP-10 Workshop on Symmetry in Constraint Satisfaction Problems*, 2010.
8. Shant Karakashian, Robert Woodward, Christopher Reeson, Berthe Y. Choueiry, and Christian Bessiere. A first practical algorithm for high levels of relational consistency. In *Proceedings of AAAI-10*, pages 101–107, 2010.
9. Christophe Lecoutre. STR2: Optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, 2011.
10. Christophe Lecoutre, Chavalit Likitvivatanavong, and Roland H. C. Yap. A path-optimal GAC algorithm for table constraints. In *Proceedings of ECAI-12*, pages 510–515, France, 2012.
11. Christophe Lecoutre, Anastasia Paparrizou, and Kostas Stergiou. Extending STR to a higher-order consistency. In *Proceedings of AAAI-13*, pages 576–582, Washington, U.S., 2013.
12. Christophe Lecoutre and Radoslaw Szymanek. Generalized arc consistency for positive table constraints. In *Proceedings of CP-06*, pages 284–298, 2006.
13. Jean-Baptiste Mairy, Yves Deville, and Christophe Lecoutre. Domain k-wise consistency made as simple as generalized arc consistency. In *Proceedings of CPAIOR-14*, pages 235–250, 2014.
14. Jean-Baptiste Mairy, Pascal Van Hentenryck, and Yves Deville. Optimal and efficient filtering algorithms for table constraints. *Constraints*, 19(1):77–120, 2014.
15. S. Merchez, C. Lecoutre, and F. Boussemart. AbsCon: a prototype to solve CSPs with abstraction. In *Proceedings of CP-01*, pages 730–744, Paphos, Cyprus, 2001.
16. Anastasia Paparrizou and Kostas Stergiou. An efficient higher-order consistency algorithm for table constraints. In *Proceedings of AAAI-12*, pages 535–541, 2012.
17. D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP-94*, pages 10–20, Seattle WA, 1994.
18. Nikolaos Samaras and Kostas Stergiou. Binary encoding of non-binary constraint satisfaction problems: Algorithms and experimental results. *JAIR*, 24:641–684, 2005.
19. Ke Xu, Frederic Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: easy generation of hard (satisfiable) instances. *AIJ*, 171(8–9):514–534, 2007.