Decomposition of the Factor Encoding for CSPs

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
Generalized arc consistency (GAC) is one of the most fundamental properties for reducing the search space when solving constraint satisfaction problems (CSPs). Consistencies stronger than GAC have also been shown useful, but the challenge is to develop efficient and simple filtering algorithms. Several CSP transformations are proposed recently so that the GAC algorithms can be applied on the transformed CSP to enforce stronger consistencies. Among them, the factor encoding (FE) is shown to be promising with respect to recent higher-order consistency algorithms. Nonetheless, one potential drawback of the FE is the fact that it enlarges the table relations as it increases constraint arity. We propose a variation of the FE that aims at reducing redundant columns in the constraints of the FE while still preserving full pairwise consistency. Experiments show that the new approach is competitive over a variety of random and structured benchmarks.

1 Introduction
In order to solve a constraint satisfaction problem (CSP), local consistencies are commonly used to filter out inconsistent parts of the constraint network to reduce the search space during the solving process. Generalized arc consistency (GAC) is usually the local consistency of choice. With so many algorithms having been developed, GAC is invariably implemented in most solvers in one form or another. Stronger consistencies can further reduce the search space when solving constraint satisfaction problems (CSPs). Among these, the factor encoding (FE) is shown to be promising with respect to recent higher-order consistency algorithms. Nonetheless, one potential drawback of the FE is the fact that it enlarges the table relations as it increases constraint arity. We propose a variation of the FE that aims at reducing redundant columns in the constraints of the FE while still preserving full pairwise consistency. Experiments show that the new approach is competitive over a variety of random and structured benchmarks.

Due to the practical challenges in implementing higher-order consistency algorithms, an alternative approach for some higher consistencies is to convert a CSP into another CSP and apply existing GAC propagators on the result, so that it is equivalent to enforcing the stronger consistencies on the original CSP. The k-interleaved encoding (kIL) [Mairy et al., 2014] is one such approach. Enforcing GAC on the kIL is the same as enforcing k-wise consistency on the original problem. A recent transformation is the factor encoding (FE) [Likitvivatanavong et al., 2014], which extracts commonly shared variables between pairs of constraints and forms new variables called factor variables. The factor variables are then augmented to the original constraints. Similar to the kIL, enforcing GAC on the FE is the same as enforcing FPWC on the original problem. However, although the results for the FE transformation [Likitvivatanavong et al., 2014] show that the FE provides an efficient way of achieving FPWC, many new factor variables can be added which can substantially increase the total size of the transformed constraints and increase runtime overheads.

In this paper, we address this shortcoming of the FE and propose a new encoding based on the FE. The idea is to decompose constraints such that factor variables and their corresponding original variables are taken out to form new constraints. We show that this new variant preserves the main property of the FE. We then perform an experimental study on the FE and this new transformation using multiple search heuristics. We show that the transformations can benefit dynamic search heuristics, e.g. dom/ddeg and dom/wdeg, due to the change in constraint networks. This new encoding is competitive with the FE on majority of problem instances and can reduce the search space and speed up the solving on some structured problems by several orders of magnitude.

2 Preliminaries
A constraint network $P$ is a pair $(\mathcal{X}, \mathcal{C})$ 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 $x \in \mathcal{X}$. 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_1, \ldots, x_n\}, rel(c) \subseteq \prod_{j=1}^n D(x_j) \) represents the set of satisfying combinations of values for the variables in scp(c).

We may also refer to \( c \) by \( c(x_1, \ldots, x_n) \). The dual graph of \( P \) is a graph in which vertices represent the constraints and edges connect two vertices whose constraints’ scopes overlap. The arity of \( c \) is \( |scp(c)| \). Given an ordered set \( S \subseteq scp(c) \) and \( \tau \in rel(c) \), the projection of \( \tau \) on \( S \) (\( \tau[S] \)) is the tuple consisting of only the components of \( \tau \) that correspond to the variables in \( S \). A tuple \( \tau = (a_1, \ldots, a_k) \) where \( a_j \in D(x_j) \) is said to be a tuple over \( \{x_1, \ldots, x_n\} \).

When elements in rel(c) are given explicitly, \( c \) is called a (positive) table constraint. A tuple \( \tau \in rel(c) \) is valid iff \( \tau[x] \in D(x) \) for each \( x \in scp(c) \). Otherwise \( \tau \) is invalid. A tuple \( \tau \in rel(c) \) is a support of \((x,a)\) in \( c \) iff \( \tau[x] = a \).

**Definition 1** (GAC) A value \((x,a)\) is a generalized arc-consistent (GAC) \([\text{Dechter}, 2003]\) iff for any constraint \( c \) involving \( x \), there exists at least one valid support \( \tau \) for \((x,a)\) in \( c \). A variable \( x \) is GAC iff \((x,a)\) is GAC for each \( a \in D(x) \).

A CSP \( P \) is GAC iff all variables are GAC.

A solution to \( P \) is a valid tuple over \( X \) such that every constraint is satisfied. \( P \) is satisfiable iff a solution exists.

A compound variable \( X \) is a cross-product composition from \( \{x_1, \ldots, x_m\} \subseteq X \), called X’s signature \( (\sigma(X)) \), where \( D(X) \subseteq \prod_{j=1}^m D(x_j) \) and its values are referred to as compound values. Given a constraint \( c \) and an ordered set \( S = \{x_1, \ldots, x_i\} \subseteq scp(c) \), we denote \( \lambda_i(S) \) to be the compound variable on \( S \) with respect to \( c \) whose domain \( D(\lambda_i(S)) \) is \( \tau[S] | \tau \in rel(c) \). It follows that \( \sigma(\lambda_i(S)) = S \). Note that compound variables and compound values are logical concepts. In practice, they are represented by the standard notion of variables and values (Example 2 will clarify this point). We may drop the subscript and write \( \lambda(S) \) if there is no ambiguity. Non-compound variables are called ordinary variables. For uniformity, \( \sigma \) is defined for all variables, i.e. \( \sigma(x) = \{x\} \) for an ordinary variable \( x \).

**Definition 2** (maxRPWC) A value \((x,a)\) is max-restricted pairwise consistent (maxRPWC) \([\text{Bessi\`ere et al.}, 2008]\) iff for all \( c_i \in C \) where \( x \in scp(c_i) \), \((x,a)\) has a valid support \( \tau \) in \( rel(c_i) \) such that for any other \( c_j \in C \), \( x \) exists a valid tuple \( \tau_j \in rel(c_j) \) and \( \tau_j[scp(c_i) \cap scp(c_j)] = \tau_j[scp(c_i) \setminus scp(c_j)] \). A CSP \( P \) is maxRPWC iff all values are maxRPWC.

**Definition 3** (PWC) A CSP \( P = (X,C) \) is pairwise consistent (PWC) \([\text{Janssen et al.}, 1989]\) iff for any constraint \( c_i \) and any valid tuple \( \tau_i \in rel(c_i) \), for any other constraint \( c_j \), there exists at least one valid tuple \( \tau_j \in rel(c_j) \) such that \( \tau_i[scp(c_i) \cap scp(c_j)] = \tau_j[scp(c_i) \setminus scp(c_j)] \).

**Definition 4** (FPWC) A CSP \( P \) is full pairwise consistency (FPWC) \([\text{Lecoutre et al.}, 2013]\) iff it is GAC and PWC.

FPWC is also equivalent to PWC together with maxRPWC. The following example illustrates the filtering power of GAC and FPWC.

**Example 1** Consider the CSP \( P = (X,C) \) in Fig. 1(a), where \( X = \{x, y, u, v, w\} \), \( D(x) = D(y) = \{0, 1\} \), \( D(u) = D(v) = D(w) = \{0\} \) and \( C = \{c_1, c_2, c_3\} \). It is trivial to check that \( P \) is GAC. Let us consider FPWC. Only the first tuple of each constraint can be extended to other constraints.

<table>
<thead>
<tr>
<th>(c_1)</th>
<th>(c_2)</th>
<th>(c_3)</th>
<th>(c'_1)</th>
<th>(c'_2)</th>
<th>(c'_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x y u</td>
<td>x y v</td>
<td>x y w</td>
<td>x y u</td>
<td>x y v</td>
<td>x y w</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>0 1 0</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

(a) Original  
(b) Factor encoding of (a)

Figure 1: Original network and its factor encoding.

The second and the third tuples of each constraint are thus inconsistent with respect to FPWC. As a result \((x, 1)\) and \((y, 1)\) are no longer FPWC and can be pruned from \(D(x)\) and \(D(y)\).

**3 The factor encoding**

The factor encoding (FE) \([\text{Likitivatanavong et al., 2014}]\) converts a constraint network \( P \) into another network \( P^* \) such that enforcing GAC on \( P^* \) is equivalent to enforcing FPWC on \( P \). This section provides background on the FE.

Given \( P = (X,C) \) the factor encoding (FE) of \( P \) is the network \( P^* = (X \cup \mathcal{W} \cup \mathcal{C}^*, \mathcal{T}^*) \) where,

\[ \mathcal{W}^* = \{\lambda(S) | D(\lambda(S)) = \bigcup_k D(\lambda_{ck}(S)) \} \text{ for all } k \] such that \( \lambda_{ck}(S) \in \mathcal{W} \), and \[ \mathcal{W} = \{\lambda_S, \lambda_{c}, \lambda_{c'} | S = scp(c_i) \cap scp(c_j) \} \text{ for all } i \neq j \land |S| > 1 \]

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

- \( scp(c_i^*) = scp(c_i) \cup \{\lambda(S) | \lambda(S) \in \mathcal{W}^* \land S \subseteq scp(c_i) \} \)
- for any \( \tau \in rel(c_i) \), let \( ext(c_i^*, \tau) \) be a tuple extended from \( \tau \) such that
  \[ - ext(c_i^*, \tau)[x] = \tau[x] \text{ for any } x \in scp(c_i) \]
  \[ - \text{for any } \lambda(S) \in scp(c_i^*) \text{, } ext(c_i^*, \tau)[\lambda(S)] = ext(c_i^*, \tau)[\lambda(S)] \]
  then, \( rel(c_i^*) = \{ext(c_i^*, \tau) | \tau \in rel(c_i)\} \).

We call the compound variables in \( \mathcal{W}^* \) factor variables. \( P^* \) is also referred to as \( \text{fe}(P) \). A pair of constraints is factorable if it generates a factor variable in the FE; a constraint network is factorable if it contains at least one such pair.

**Example 2** Consider \( P \) in Example 1. Fig. 1(b) shows \( \text{fe}(P) \), which involves a factor variable \( f \) where \( \sigma(f) = \{x, y\} \) and \( D(f) = \{(0,0), (0,1), (1,1), (1,0)\} \). For simplicity, \( D(f) \) is normalized to \( \{0, 1, 2, 3\} \).

The factor encoding employs auxiliary variables to represent the intersection between constraints. These additional variables are then grafted onto the constraints where they originate from. The FE, therefore, achieves higher-order consistency at a cost of enlarged tables.

**Theorem 1** \([\text{Likitivatanavong et al., 2014}]\) \( \text{fe}(P) \) is GAC if and only if \( P \) is FPWC.

Consider \( \text{fe}(P) \) in Fig. 1(b) for example. Enforcing GAC on \( \text{fe}(P) \) reduces \( D(f) \) to \( \{0\} \). This makes the second and the third tuples of \( c'_1, c'_2 \), and \( c'_3 \) invalid and leaves \((x, 1)\) and \((y, 1)\) with no valid support. The effect is the same as enforcing FPWC on \( P \).
Factor-decomposition encoding

In this paper, we propose a new encoding that addresses the FE’s disadvantage. It is basically a variation of the FE and the idea behind it is to compress the FE by extracting factor variables and their signatures to form new constraints. The goal is to reduce the arity of the factor-encoded constraints in the FE where constraint arity can be much larger than the original arity. Specifically, a constraint that is augmented with factor variables by the FE is decomposed into multiple smaller constraints. A new constraint is created for each factor variable such that the scope includes the factor variable itself and the variables in its signature. The original constraint is then modified so that any ordinary variable that is a member of some factor variable’s signature is removed. We call this new transformation factor-decomposition encoding (FDE).

We explain the process as follows. Given network $P$,

1. Construct $fe(P)$
2. (Decomposition) For each factor variable $f \in scp(c^*)$,
   a. subtract $\sigma(f)$ from $scp(c^*)$.
   b. add a new constraint $c_f$ such that $scp(c_f) = \{f\}$
      $\cup \sigma(f)$, and $rel(c_f) = \{(t, a_{i_1}, \ldots, a_{i_{\sigma(f)}}) \mid t \in D(f) \land t = (a_{i_1}, \ldots, a_{i_{\sigma(f)}})\}$. The resulting constraint network is denoted by $fde(P)$.

Example 3 Given $P$ in Example 1, $fde(P)$ is shown in Fig. 2. Constraints $c_1, c_2, c_3$ in $P$ are reduced to binary constraints $c_1', c_2', c_3'$ due to the removal of $\{x, y\}$. A new constraint $c_f(x, y)$ that maintains the connection between $f$ and $x, y$ is introduced. Note that the table relations of $fde(P)$ (30 cells) are smaller than those of $fe(P)$ (36 cells).

Theorem 2 $fde(P)$ is GAC iff $fe(P)$ is GAC.

Sketch of Proof: Given variable $x \in scp(c)$ removed by step 2(a), we will show that the FDE keeps the restriction between $c$ and any other constraint $c'$ in the FE whose scope includes $x$. There are three cases.

1. $x \in \sigma(f)$ where $f$ is a factor variable such that $f \in scp(c) \cap scp(c')$. The new constraint $c_f$ added in step 2(b) produces a cycle $c-c'-c_f$ in the dual graph that maintains the effect of $x$ on the pair $(c, c')$. The connection between $c$ and $c'$ in the original dual graph is provided by $x$. Removing $x$ does not take away this connection since it is replaced by the connection between $c$ and $c_f$ (through $f$) and between $c_f$ and $c'$ (through $x$).

2. $x \in \sigma(f')$ for some factor variable $f' \in scp(c)$ and there exists no factor variable $f'' \in scp(c')$ such that $x \in \sigma(f'')$. The connection between $c$ and $c'$ in the original dual graph is provided by $x$. Removing $x$ does not take away this connection since it is replaced by the connection between $c$ and $c_f$ (through $f$) and between $c_f$ and $c'$ (through $x$).

3. $x \in \sigma(f) \cap \sigma(f')$ where $f$ and $f'$ are factor variables such that $f \in scp(c)$ and $f' \in scp(c')$. Step 2(b) ensures that there are constraints $c_f$ and $c_{f'}$ such that $scp(c_f) \supseteq \{f, x\}$ and $scp(c_{f'}) \supseteq \{f', x\}$. That means although $x$ is removed, the effect of $x$ on the pair $(c, c')$ is replicated through the path $(c, c_f), (c_f, c_{f'}), (c_{f'}, c')$ in the dual graph.

The FDE retains the main property of the FE (Theorem 1 [Likitvivatanavong et al., 2014]). We have the following corollary.

Corollary 1 $fde(P)$ is GAC iff $P$ is FPWC.

Although the average arity may increase, we expect the FDE to reduce average arity. This is borne out in the experiments which show that the average constraint arity is reduced in practice. Also the maximum constraint arity of the FDE and the original CSP is always smaller than that of the FE. Since the speed of various GAC algorithms and implementations can depend strongly on the arity, the FDE can be faster than the FE in many cases (experiments in Section 5 show that the arity reduction allows special purpose binary AC algorithms to replace GAC).

We remark that $fde(P)$ may still be factorable even though it includes the FE as part of the process. While it is true that step 1 and 2(a) replace shared variables with factor variables, only the original constraints are affected. For this reason, only the scopes of the original constraints are guaranteed to intersect on fewer than two variables. The new constraints created in step 2(b), however, may contain a factorable pair. In this case, we can re-apply the FDE possibly multiple times until there are no more factorable pairs, a fixpoint.

Example 4 Fig. 3 shows an example of $fde(fde(P))$. The diagram on the left is the dual graph for some constraint network $P$, the middle shows the dual graph for one application of the FDE, and the right shows two successive applications of the FDE, reaching a fixpoint afterward.
The number of times for the FDE to reach a fixpoint is dependent on the network. An example of a network that needs arbitrary applications of the FDE to reach a fixpoint follows.

Example 5 We will construct a network $\mathcal{P}_i$ such that it takes $i$ applications of the FDE to reach a fixpoint. A recursive definition of $\mathcal{P}_i$ is given as follows. $\mathcal{P}_0 = \{(x, y), \{c(x, y)\}\}$ ($\mathcal{P}_0$ has a single constraint involving two variables). Given $\mathcal{P}_i = (X_i, C_i)$ with $C_i = \{c_1, \ldots, c_m\}$, then $\mathcal{P}_{i+1} := (X_i+1, C_{i+1})$ where $C_{i+1} = \{c_{j,k} \mid 1 \leq j \leq m \wedge 1 \leq k \leq 2 \wedge \text{sep}(c_{j,k}) = \text{sep}(c_j) \cup \{z_{j,k}\}$ and $X_{i+1} = X_i \cup \{z_{j,k} \mid 1 \leq j \leq m \wedge 1 \leq k \leq 2\}$. 

**Proposition 2** Let $fde^k(\mathcal{P})$ denote $fde(fde(...fde(\mathcal{P})...))$ (the FDE is applied $k$ times in a row). An upperbound of $k$ is $\max \{|\sigma(f)| - 1 \mid f$ is a factor variable of $fe(\mathcal{P})\}$

Applying the FDE more than once may affect the structure of the network in a positive way since more redundancy is eliminated. In contrast, applying the FE more than once is pointless since it only increases arity and redundancy. Moreover, similar to the FE, multiple applications of the FDE does not affect the level consistency when GAC is enforced.

In the fixpoint of the FDE, no two constraints share more than one variable. The same cannot be said for the FE or $fde^k$ which has not reached fixpoint.

**Proposition 3** $fde^k(\mathcal{P})$ is GAC iff $fde(\mathcal{P})$ is GAC.

**Corollary 2** $fde^k(\mathcal{P})$ is GAC iff $FPW(C)$

5 Experiments

We present extensive experiments that evaluate the new FDE encoding compared with the FE and the original network. All extensional benchmarks from the CSP solver competition that are non-binary and factorable are used. We also convert some intensional structure benchmarks to extensional. In total, we use 633 problem instances from 22 problem series. However, only some instances allow $fde^k$ for $k > 1$ (of all structured problems tested, only $fpga$ and $ii$-8 do). With insufficient benchmarks, experiments involve only one-pass FDE. The experiments were run on a 3.20GHz Intel i7-960 with 64-bit Linux. The converters take an input in the XCSP format and output the result as another text file, also in XCSP. For the FDE, the converter generates the encoding directly (the description of the FDE process in Section 4 in which the FE is generated as a first step is only an exposition). We used AbsCon [Merchez et al., 2001] as the solver where the default GAC algorithm is STR2 [Lecoutre, 2011] and the default arc consistency algorithm is the bitwise AC3 algorithm [Lecoutre and Vion, 2008]. CPU time is limited to 1800 seconds while memory is limited to 8GB.

The encodings alter the problem’s structure in a way that may lead variable ordering heuristics to choose variables differently compared with the original problem. Thus, even though the consistency level is the same, the number of nodes visited could be different. For the FE, new variables are added to the network as well as constraint scopes, increasing constraint arity in the process. For the FDE, although the set of variables is the same as the FE’s the topology of the network is affected even more profoundly given that variables are extracted from one constraint to form an entirely new constraint. Because structural features of a network are often taken into account in search heuristics, the same heuristic could possibly pick different variables for the original CSP, the FE and the FDE. In order to test the performance of these encodings realistically, it would be better to let the heuristics choose freely rather than foisting the same ordering on all of them. For this reason, we employed four well-known and commonly used variable selection heuristics in our experiments, dom/ideg, dom/wdeg [Boussemart et al., 2004], impact [Refalo, 2004], activity [Michel and Hentenryck, 2012], and test them on the original problem as well as the FE and the FDE encodings. We used lex value ordering in all cases.

Table 1 shows the mean results on unstructured problems, calculated from instances that are solved within the time limit by all the combinations of three constraint networks (one original and two encodings) and four heuristics. The first column displays the name of the benchmarks while the number below indicates the number of instances. The second column denotes the encodings, where “-” indicates no encoding (original $\mathcal{P}$). The third column gives the mean constraint arity. The next four columns (eight sub-columns) display the mean number of nodes visited during search and the mean CPU time (in seconds) for the four heuristics (unless too many instances are not solved within the time limit, in which case, the number of time-out instances is reported instead). Encoding times for the FE and the FDE are not included in the CPU time and are not reported in the table due to space restriction (we remark that for over 80% of all problem instances, the conversion time is negligible compared to the solving time, taking only a fraction of a second to generate the XCSP XML). Numbers in bold indicate best results. Graphs in Fig. 4 (in color) show overall performance of different techniques when all random problem instances are taken into account. Each point $(x, y)$ indicates that the corresponding technique is able to solve $y$ instances within $x$ seconds (note that $x$ represents the runtime of each of the $y$ instances, not the total runtime of all $y$ instances).

The FDE shows significant improvement in runtime across all heuristics for most of the series rand-3-8. The main reason for the good performance is that the FDE converts most ternary constraints into binary, so a GAC algorithm is not

---

1Available at http://www.cril.univ-artois.fr/CSC09. We exclude the $bdd$ benchmarks which have very large extensional tables. With the FE and the FDE, the tables are larger than 1GB which is too big to solve with AbsCon.

2We focus on encodings and their relative effectiveness in the experiments. The choice of solver is less important as long as it is valid and robust. We chose AbsCon for its versatility as a blackbox solver: many algorithms and heuristics are implemented and selectable. Compared to another solver such as Mistral, AbsCon handles large tables better, (e.g. AbsCon is 20X faster than Mistral on average to solve $dagrand$ (arity=15, 150000 tuples/table), so it is suitable for testing the FE which increases arities. We used the latest version AbsCon1.41 (not publicly available).

3Using the same ordering simply gives the same search space.

4In AbsCon, instantiating a variable is counted as one search node even if there is only a single value in the domain.
needed and AbsCon employs a bitwise-based AC [Lecoutre and Vion, 2008]. For the FDE of rand-3-20-20, on average 73.9% of original ternary constraints are transformed into binary where most have two factor variables; the FDE of rand-3-20-20 has 41.6% binary constraints and 58.4% ternary constraints on average, which results in 46% runtime savings. For dagrand and rand-10-20-10, FPWC proves the problems unsatisfiable without search and that is why there is little difference across heuristics within the same encoding. Runtimes are dominated by GAC processing which depends on table size. Tables in the FDE can be larger than those in the FE, as is the case with dagrand and rand-10-20-10. For dagrand, there are 120 factor variables compared to 23 ordinary variables, which translates to 42 extra tables for the FDE. Over all, the graphs in Fig. 4 show that the FDE solves more problems within the specified time on all four heuristics.

Table 2 shows the mean results for structured problems. Again, the conversion times for the FE and FDE are negligible for almost all instances and are not reported for space reason. As there are fewer structured benchmarks in extensional form, we have added some structured benchmarks which are given in intensional form in the CSP solver competition benchmarks (separated by double line in the table). The series fpga, allInterval, Schur's lemma, jnh, Chessboard coloration, ii-8, and socialGolfers are converted\(^\text{3}\) from intensional to extensional constraints. Graphs in Fig. 5 show overall performance of different techniques when all structured problem instances are taken into account.

From the graphs, the FDE provides huge speedup for dom/ddeg and dom/wdeg on allInterval and fpga. At the same time, it has an adverse effect on ii-8. Detailed results for fpga are shown\(^\text{4}\) in Table 3. One of the reasons for the striking results on allInterval and fpga can be attributed to scope reduction by the FDE. For allInterval, the original problems contain only binary and ternary constraints. The FDE converts all original ternary constraints into binary. The reduction in fpga is even more notable. Take fpga-12-12 for example. In this instance there are 180 constraints of arity 6, 7, and 12 — all in the same proportion. The FDE of this instance has 192 constraints in total, which 144 are binary. By contrast, the poor performance of dom/ddeg and dom/wdeg on the FDE of ii-8 appears to be idiosyncratic. We found that choosing variables randomly instead always brings down the runtime to the same level as the FE’s on the ii-8 instances. We also tried random heuristic on the allInterval and fpga series but improvement is limited.

The FDE works well on unstructured benchmarks which are generally harder, solving more than the FE and untransformed CSPs. For structured problems, running impact on the original CSPs is the best (213 instances solved), followed by impact on the FDE (211). However, structured benchmarks have many easy instances. Altogether, dom/wdeg on the FDE

\(^{3}\)Conversion costs are given in this table. The converter is external to the solver and reads and writes XCSP XML files, so there are I/O costs for large files when the tables are large.

\(^{4}\)We focus on studying the FE on extensional constraints but also tested the intensional benchmarks on AbsCon. Interestingly, we found that table constraints with the FE variants can be faster than intensional, e.g. allInterval can be 60% faster.
Table 2: Mean results for structured benchmarks. “modR”, “allInt”, “lemma”, and “socG” stand for modRenault, allInterval, Schur’s lemma, and socialGolfer.

solves the most number of instances (445).

6 Related work

Transformation of one CSP to another is well studied. The early and most well-known work is the transformation of non-binary constraints to binary, which includes the dual and the hidden encoding [Bacchus et al., 2002; Samaras and Stergiou, 2005]. The k-interleaved encoding [Mairy et al., 2014] is another way to incorporate k-wise consistency into the network by adding extra k-ary constraints that are basically the index of all feasible combination of tuples. It was shown in [Likitvivatanavong et al., 2014] that the FE consistently outperforms the k-interleaved encoding when k is two. [Bessière et al., 2008] gives an extensive coverage of filtering consistencies for non-binary constraints. Other works on consistency algorithms that are stronger than GAC are [Paparrizou and Stergiou, 2012] (maxRPWC) and [Lecoutre et al., 2013; Karakashian et al., 2010; Schneider et al., 2014; Woodward et al., 2014] (FPWC and k-wise consistency). The FE has proved to be faster than eSTR2 [Lecoutre et al., 2013], an algorithm that enforces FPWC. In [Woodward et al., 2014], an adaptive algorithm that is a compromise between GAC and FPWC was studied, its speed ranging between GAC’s and FPWC’s for the most part.

7 Conclusion

We have introduced a variation of the FE for non-binary constraint networks. Unlike the FE, which merely augments table constraints with factor variables, the FDE also takes further advantage of the factor variables by extracting and replacing original variables with new constraints. Experiments show that the FDE has an edge over the FE on random problems while it can vastly outperform the FE on structured problems with the dom/ddeg and dom/wdeg heuristics.

The focus on network transformation in the literature has been on concrete, theoretical properties associated with the transformation, e.g. transforming non-binary constraints to binary [Bacchus et al., 2002]. The FE encodes PWC while the FDE decomposes the network. These tangible properties are undoubtedly useful, but our experiments suggest that coming up with transformations that can influence variable heuristics down the right path can have a greater effect. Such transformations may not have any identifiable property associated with them, as their sole function is to steer variable heuristics by exposing some hidden features of the network, and would be designed together with the heuristic to be used. The FDE provides a glimpse in this 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.
<table>
<thead>
<tr>
<th>instances/dimencg</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>10-8</td>
<td>10-9</td>
<td>10-10</td>
<td>11-9</td>
<td>11-10</td>
<td>11-11</td>
<td>12-0</td>
<td>12-1</td>
<td>13-1</td>
<td>13-2</td>
<td>13-3</td>
<td>14-0</td>
<td>14-1</td>
<td>14-2</td>
<td>15-0</td>
<td>15-1</td>
<td>15-2</td>
</tr>
<tr>
<td>time</td>
<td>0.10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.18</td>
<td>0.03</td>
<td>0.28</td>
<td>0.06</td>
<td>0.10</td>
<td>287</td>
<td>0.18</td>
<td>0.03</td>
<td>267</td>
<td>0.10</td>
<td>0.04</td>
<td>0.28</td>
<td>0.06</td>
<td>0.10</td>
</tr>
<tr>
<td>nodes</td>
<td>397811</td>
<td>3.13M</td>
<td>124.24</td>
<td>3.32M</td>
<td>162.04</td>
<td>1064.07</td>
<td>21529</td>
<td>0.78</td>
<td>3699.05</td>
<td>0.52</td>
<td>3699.05</td>
<td>0.75</td>
<td>1358.04</td>
<td>0.41</td>
<td>755317</td>
<td>7.62</td>
<td>755317</td>
</tr>
<tr>
<td>nodes</td>
<td>2123.92</td>
<td>1.04M</td>
<td>124.24</td>
<td>3.32M</td>
<td>162.04</td>
<td>1064.07</td>
<td>21529</td>
<td>0.78</td>
<td>3699.05</td>
<td>0.52</td>
<td>3699.05</td>
<td>0.75</td>
<td>1358.04</td>
<td>0.41</td>
<td>755317</td>
<td>7.62</td>
<td>755317</td>
</tr>
<tr>
<td>nodes</td>
<td>2123.92</td>
<td>1.04M</td>
<td>124.24</td>
<td>3.32M</td>
<td>162.04</td>
<td>1064.07</td>
<td>21529</td>
<td>0.78</td>
<td>3699.05</td>
<td>0.52</td>
<td>3699.05</td>
<td>0.75</td>
<td>1358.04</td>
<td>0.41</td>
<td>755317</td>
<td>7.62</td>
<td>755317</td>
</tr>
<tr>
<td>nodes</td>
<td>2123.92</td>
<td>1.04M</td>
<td>124.24</td>
<td>3.32M</td>
<td>162.04</td>
<td>1064.07</td>
<td>21529</td>
<td>0.78</td>
<td>3699.05</td>
<td>0.52</td>
<td>3699.05</td>
<td>0.75</td>
<td>1358.04</td>
<td>0.41</td>
<td>755317</td>
<td>7.62</td>
<td>755317</td>
</tr>
</tbody>
</table>

Table 3: Runtime and the number of nodes for instances in the fpga benchmark. “x” denotes time-out. The second and the third sub-columns show the time it takes to convert an instance.

References


