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 and may be more appropriate for hard problems, but implementing such algorithms to be competitive with the state-of-the-art solvers employing GAC may be a challenge. They have been the subject of recent works [Bessière *et al.*, 2008; Karakashian *et al.*, 2010; Schneider *et al.*, 2014; Paparrizou and Stergiou, 2012; Lecoutre *et al.*, 2013], which propose propagation algorithms for higher-order consistencies such as relational consistency, max-restricted pairwise consistency, and full pairwise consistency (FPWC). Some of the new algorithms are based on well-established GAC algorithms, e.g. the Simple Tabular Reduction algorithm [Ullmann, 2007; Lecoutre, 2011] was extended to cover FPWC in [Lecoutre *et al.*, 2013].

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 $\mathcal{P}$ 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 $scp(c)$ =

$\{x_{i_1}, \ldots, x_{i_r}\}$, $rel(c) \subseteq \prod_{j=1}^{r} D(x_{i_j})$ represents the set of satisfying combinations of values for the variables in $scp(c)$. We may also refer to c by $c(x_{i_1}, \ldots, x_{i_r})$. The dual graph of $\mathcal{P}$ 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 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 a tuple over $\{x_{i_1}, \ldots, x_{i_k}\}$. 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 τ 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 generalized arc-consistent (GAC) [Dechter, 2003] iff for any constraint c involving x, there exists at least one valid support τ for (x, a) in c. A variable x is GAC iff (x, a) is GAC for each $a \in D(x)$. A CSP $\mathcal{P}$ is GAC iff all variables are GAC.*

A *solution* to $\mathcal{P}$ is a valid tuple over $\mathcal{X}$ such that every constraint is satisfied. $\mathcal{P}$ is *satisfiable* iff a solution exists.

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 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 $\sigma(\lambda_c(S)) = S$. 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, σ 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) [Bessière* et al.*, 2008] 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 $\tau_i[scp(c_i) \cap scp(c_j)] = \tau_j[scp(c_i) \cap scp(c_j)]$. A CSP $\mathcal{P}$ is maxRPWC iff all values are maxRPWC.*

Definition 3 (PWC) *A CSP $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ is* pairwise consistent *(PWC) [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 $\tau_i[scp(c_i) \cap scp(c_j)] = \tau_j[scp(c_i) \cap scp(c_j)]$.*

Definition 4 (FPWC) *A CSP $\mathcal{P}$ is full pairwise consistency (FPWC) [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 $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ in Fig. 1(a), where $\mathcal{X} = \{x, y, u, v, w\}$, $D(x) = D(y) = \{0, 1\}$, $D(u) = D(v) = D(w) = \{0\}$, and $\mathcal{C} = \{c_1, c_2, c_3\}$. It is trivial to check that $\mathcal{P}$ is GAC. Let us consider FPWC. Only the first tuple of each constraint can be extended to other constraints.*

c_1			c_2			c_3		
x	y	u	x	y	v	x	y	w
0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	1	0	0
1	1	0	1	0	0	1	1	0

(a) Original

c_1^*				c_2^*				c_3^*			
x	y	u	f	x	y	v	f	x	y	w	f
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	1	0	0	3
1	1	0	2	1	0	0	3	1	1	0	2

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

Given $\mathcal{P}$= $(\mathcal{X}, \mathcal{C})$ 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

$\mathcal{W}$ = $\{ \lambda_{c_i}(S), \lambda_{c_j}(S) \mid S = scp(c_i) \cap scp(c_j)$ for all $i \neq j \wedge |S| > 1\}$

and 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 $\tau \in rel(c_i)$, let $ext(c_i^*, \tau)$ be a tuple extended from τ such that
  - $ext(c_i^*, \tau)[x] = \tau[x]$ for any $x \in scp(c_i)$
  - for any $\lambda(S) \in scp(c_i^*)$, $ext(c_i^*, \tau)[\lambda(S)] = ext(c_i^*, \tau)[S] (= \tau[S])$

then, $rel(c_i^*)$ = $\{ext(c_i^*, \tau) \mid \tau \in rel(c_i)\}$.

We call the compound variables in $\mathcal{W}^*$ *factor variables*. $\mathcal{P}^*$ is also referred to as *fe*($\mathcal{P}$). 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 $\mathcal{P}$ in Example 1. Fig. 1(b) shows fe($\mathcal{P}$), 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 ([Likitvivatanavong *et al.*, 2014]) *fe($\mathcal{P}$) is GAC if and only if $\mathcal{P}$ is FPWC.*

Consider *fe*($\mathcal{P}$) in Fig. 1(b) for example. Enforcing GAC on *fe*($\mathcal{P}$) 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 $\mathcal{P}$.

c'_1		c'_2		c'_3	
u	f	v	f	w	f
0	0	0	0	0	0
0	1	0	1	0	3
0	2	0	3	0	2

c_f		
f	x	y
0	0	0
1	0	1
2	1	1
3	1	0

Figure 2: Example for the factor decomposition encoding.

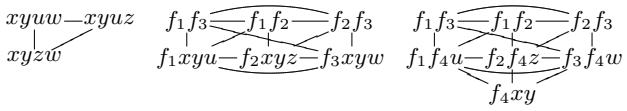

Figure 3: The dual graph of a network $\mathcal{P}$ (left), *fde*($\mathcal{P}$) (middle), and *fde*(*fde*($\mathcal{P}$)) (right), where $f_1 = xyu$, $f_2 = xyz$, $f_3 = xyw$, and $f_4 = xy$.

4 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 $\mathcal{P}$,

1. Construct *fe*($\mathcal{P}$)
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 $rel(c_f) = \{(t, a_{i_1}, \ldots, a_{i_{|\sigma(f)|}}) \mid t \in D(f) \wedge t = (a_{i_1}, \ldots, a_{i_{|\sigma(f)|}})\}$.

The resulting constraint network is denoted by *fde*($\mathcal{P}$).

Example 3 *Given $\mathcal{P}$ in Example 1, fde($\mathcal{P}$) in shown in Fig. 2. Constraints c_1, c_2, c_3 in $\mathcal{P}$ are reduced to binary constraints c'_1, c'_2, c'_3 due to the removal of $\{x, y\}$. A new constraint $c_f(f, x, y)$ that maintains the connection between f and x, y is introduced. Note that the table relations of fde($\mathcal{P}$) (30 cells) are smaller than those of fe($\mathcal{P}$) (36 cells).*

Theorem 2 *fde($\mathcal{P}$) is GAC iff fe($\mathcal{P}$) 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 between the pair (c, c').

(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($\mathcal{P}$) is GAC iff $\mathcal{P}$ is FPWC.*

The FDE keeps all ordinary variables but remove some columns from original table relations. In extreme cases, original non-binary constraints are turned into binary constraints. These happen when variables in the original constraint are replaced with two factor variables, or when there is one ordinary and one factor variable. Note that although the new constraint created by step 2(b) cannot have larger arity than the original pair that produces the factor variable, it is possible for a group of large-arity constraints to generate a large number of new constraints with only slightly smaller arity, so that on the whole, the average arity increases.

Proposition 1 *Compared to fe($\mathcal{P}$), the average constraint arity may increase in fde($\mathcal{P}$).*

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*($\mathcal{P}$) 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($\mathcal{P}$)). The diagram on the left is the dual graph for some constraint network $\mathcal{P}$, 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 $\mathcal{P}_i$=$(\mathcal{X}_i, \mathcal{C}_i)$ with $\mathcal{C}_i = \{c_1, \ldots, c_m\}$, then $\mathcal{P}_{i+1}$:= $(\mathcal{X}_{i+1}, \mathcal{C}_{i+1})$ where* $\mathcal{C}_{i+1} = \{c_{j,k} \mid 1 \leq j \leq m \wedge 1 \leq k \leq 2 \wedge scp(c_{j,k}) = scp(c_j) \cup \{z^i_{j,k}\}\}$ *and* $\mathcal{X}_{i+1} = \mathcal{X}_i \cup \{z^i_{j,k} \mid 1 \leq j \leq m \wedge 1 \leq k \leq 2\}$.

Proposition 2 *Let $fde^k(\mathcal{P})$ denote $fde(fde(\ldots fde(\mathcal{P})\ldots))$ (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 $\mathcal{P}$ is FPWC.*

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[1] 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[2] 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.[3] For this reason, we employed four well-known and commonly used variable selection heuristics in our experiments, $dom/ddeg$, $dom/wdeg$ [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[4] 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-**. The main reason for the good performance is that the FDE converts most ternary constraints into binary, so a GAC algorithm is not

[1] Available 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.

[2] We 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 black-box 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).

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

[4] In AbsCon, instantiating a variable is counted as one search node even if there is only a single value in the domain.

series	enc	arity	dom/ddeg		dom/wdeg		impact		activity	
			nodes	time	nodes	time	nodes	time	nodes	time
rand-3-20-20	-	3.0	179540	85.66	152566	85.92	1376304	424.43	483720	179.42
(#50)	fe	4.8	58710	52.54	75850	80.74	745742	318.18	165385	110.25
	fde	2.6	69550	**38.55**	81524	**51.85**	745742	**208.47**	165385	**70.96**
rand-3-20-20-	-	3.0	129563	57.88	96960	52.75	967188	287.24	282616	114.15
fcd (#50)	fe	4.8	35404	30.17	43332	45.33	588075	228.30	101036	70.57
	fde	2.6	35904	**20.92**	44754	**28.78**	588075	**150.48**	101036	**44.30**
rand-8-20-5	-	8.0	101301	18.28	85703	33.70	396528	117.66	214304	89.57
(#20)	fe	23.3	4985	**14.90**	3540	**21.93**	3525	**100.60**	2917	**38.90**
	fde	5.7	6449	16.88	2545	22.59	3525	109.56	2917	40.14
rand-10-20-10	-	10.0	830	**0.56**	1028	**0.70**	2639	1.05	1517	0.89
(#20)	fe	14.0	0	0.63	0	0.77	0	**0.71**	0	**0.73**
	fde	5.5	0	0.81	0	0.90	0	0.91	0	0.90
dagrand	-	15.0	57969	21.54	74880	36.54	93786	54.77	97718	56.41
(#25)	fe	30.0	0	**12.18**	0	**11.54**	0	**11.52**	0	**11.50**
	fde	11.3	0	21.98	0	20.77	0	20.80	0	20.64
mddhalf	-	7.0	(7 time-out)		(15 time-out)		**(20 time-out)**		**(11 time-out)**	
(#25)	fe	34.1	**(5 time-out)**		(12 time-out)		(22 time-out)		**(11 time-out)**	
	fde	6.0	**(5 time-out)**		**(11 time-out)**		(21 time-out)		(12 time-out)	
rand-3-24-24	-	3.0	(36 time-out)		(36 time-out)		(50 time-out)		(44 time-out)	
(#50)	fe	4.6	(30 time-out)		(34 time-out)		(49 time-out)		(42 time-out)	
	fde	2.6	**(26 time-out)**		**(29 time-out)**		**(48 time-out)**		**(37 time-out)**	
rand-3-24-24-	-	3.0	(25 time-out)		(27 time-out)		(43 time-out)		(35 time-out)	
fcd (#50)	fe	4.6	(25 time-out)		(24 time-out)		(46 time-out)		(30 time-out)	
	fde	2.6	**(13 time-out)**		**(18 time-out)**		**(39 time-out)**		**(22 time-out)**	
rand-3-28-28	-	3.0	(49 time-out)		(49 time-out)		(50 time-out)		(50 time-out)	
(#50)	fe	4.5	**(48 time-out)**		(49 time-out)		(50 time-out)		(50 time-out)	
	fde	2.6	**(48 time-out)**		(49 time-out)		(50 time-out)		(50 time-out)	
rand-3-28-28-	-	3.0	**(48 time-out)**		**(47 time-out)**		(50 time-out)		(49 time-out)	
fcd (#50)	fe	4.5	**(48 time-out)**		(48 time-out)		(50 time-out)		(49 time-out)	
	fde	2.6	(49 time-out)		(48 time-out)		(50 time-out)		(49 time-out)	

Table 1: Mean results for unstructured benchmarks.

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 120 extra tables for the FDE. Overall, 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 so 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[5] 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 shows significant improvement for $dom/ddeg$ and $dom/wdeg$, some improvement for *impact*, and on par with the FE for *activity*. In particular, the

[5]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.

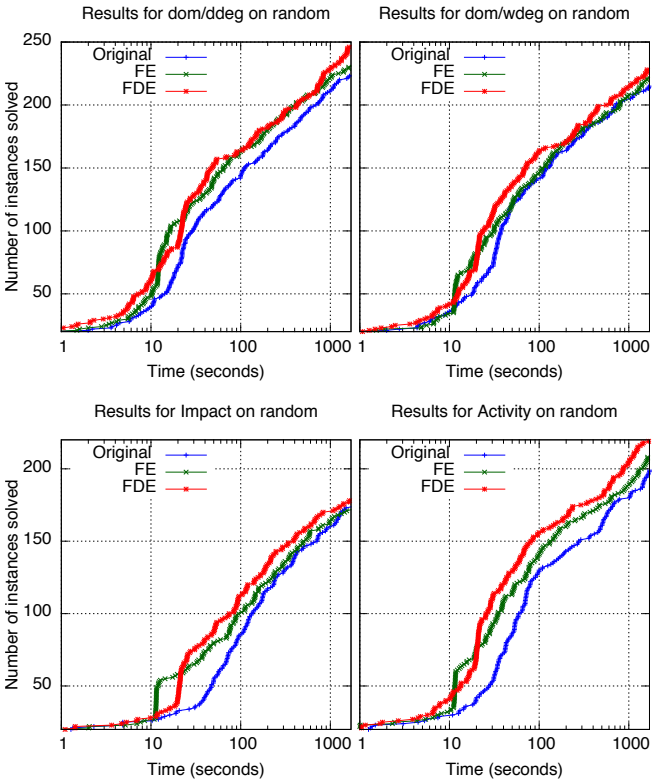

Figure 4: Runtime distribution for unstructured instances. In all four graphs, the line near the top is the FDE, the next line down is the FE, while the line near the bottom is Original.

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[6] 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 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

[6]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.

series	enc	arity	dom/ddeg		dom/wdeg		impact		activity	
			nodes	time	nodes	time	nodes	time	nodes	time
aim-50	-	3.0	14105	0.52	130	**0.27**	319	**0.29**	167	**0.27**
(#24)	fe	4.3	2665	**0.42**	2665	0.32	114	0.31	127	0.31
	fde	2.5	6832	0.52	133	0.34	114	0.32	127	0.32
aim-100	-	3.0	16839983	191.35	749	**0.36**	1977	**0.40**	779	**0.37**
(#24)	fe	4.3	56463	3.67	317	0.44	316	0.41	350	0.44
	fde	2.5	60129	**2.66**	515	0.45	316	0.44	350	0.45
aim-200	-	3.0	656012	65.52	2161	0.73	15194	1.48	17133	1.52
(#24)	fe	3.9	14786	**4.07**	660	0.70	1277	**0.81**	3669	**1.04**
	fde	2.6	21110	5.27	676	**0.69**	1277	**0.81**	3669	1.11
dubois	-	3.0	33030143	204.85	15626398	110.39	69682962	146.21	161463141	432.61
(#13)	fe	3.1	5505023	**27.67**	6293592	47.06	12602606	36.52	56717155	**177.75**
	fde	2.9	8257535	39.37	5031230	**37.70**	12602606	**36.35**	56717155	179.32
modR	-	4.3	1248820	151.38	297	**0.80**	3056	**1.03**	309	**0.77**
(#50)	fe	8.3	370.77	1.59	133	1.38	203	1.42	140	1.42
	fde	4.2	324.68	**1.49**	137	1.37	203	1.38	140	1.40
fgpa	-	8.0	(21 time-out)		(18 time-out)		(1 time-out)		**(1 time-out)**	
(#21)	fe	8.8	(21 time-out)		(18 time-out)		(1 time-out)		(13 time-out)	
	fde	3.9	**(7 time-out)**		**(1 time-out)**		(1 time-out)		(13 time-out)	
allInter	-	2.1	219393	6.06	60122	2.50	527807	**12.32**	3599359	80.33
(#25)	fe	2.2	179884	8.07	15428	1.30	390914	15.55	675806	22.60
	fde	2.0	33	**0.27**	32	**0.27**	390914	15.12	675806	**21.47**
lemma	-	3.0	143440	**12.16**	143240	**12.98**	156360	**27.33**	178022	**15.82**
(#10)	fe	4.9	164467	65.99	125904	78.79	218771	68.80	316163	79.99
	fde	2.6	164524	61.44	115739	67.74	218771	69.06	316163	78.48
jnh	-	4.6	3022	**2.38**	385	**0.80**	1379	**0.99**	2595	**1.26**
(#16)	fe	8.3	4596	7.48	1910	1.86	3494	3.68	6178	3.34
	fde	3.7	6865	12.87	2046	2.66	3494	4.19	6178	4.16
chess	-	4.0	94844	**25.45**	10406	**1.57**	26458	**1.96**	10249	**1.17**
(#16)	fe	8.0	100312	94.65	10349	3.26	370491	16.47	6175	1.75
	fde	3.7	100312	87.23	9568	3.03	370491	19.29	6175	1.87
ii-8	-	2.3	**(4 time-out)**		1266	**0.90**	2338	**1.12**	1858	**1.02**
(#14)	fe	3.0	(5 time-out)		2491	1.57	3551	2.40	3705	2.00
	fde	3.2	(13 time-out)		(7 time-out)		3551	3.29	3705	2.85
socG	-	3.2	(6 time-out)		(6 time-out)		**(3 time-out)**		(5 time-out)	
(#6)	fe	5.1	(6 time-out)		(6 time-out)		(4 time-out)		(5 time-out)	
	fde	3.1	(6 time-out)		(6 time-out)		(4 time-out)		(5 time-out)	

Table 2: Mean results for structured benchmarks. "modR", "allInter", "lemma", and "socG" stand for *modRenault*, *allInterval*, *Schur's lemma*, and *socialGolfer*.

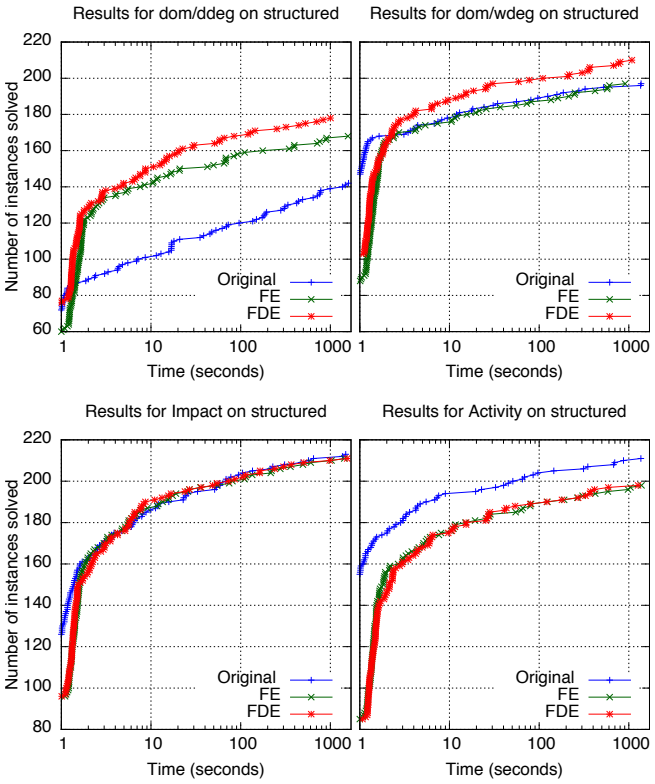

Figure 5: Runtime distribution for structured instances. For *dom/ddeg*, line ordering from top to bottom is FDE, FE, and Original. For *dom/wdeg*, the ordering is FDE, Original, and FE. For *activity*, the top line is Original. Other lines are not distinguishable.

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 decomposition changes 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.

#I	converter		*dom/ddeg*				*dom/wdeg*						*impact*						*activity*					
	fe	fde	-	fe	fde		-		fe		fde		-		fe		fde		-		fe		fde	
	time	time			nodes—	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time
10-8	0.00	0.00	x	x	397811	**6.75**	5.31M	124.44	5.32M	162.04	1064	**0.47**	21529	0.78	3699	**0.52**	3699	0.75	1358	**0.41**	755317	7.62	755317	5.99
10-9	0.01	0.00	x	x	145	**0.30**	446599	12.01	430086	15.60	145	**0.34**	135	**0.32**	145	0.38	145	0.33	135	**0.33**	4.74M	63.15	4.74M	41.06
10-10	0.01	0.00	x	x	160	**0.34**	5.32M	157.78	5.36M	213.63	160	**0.36**	150	0.33	160	0.39	160	**0.31**	150	**0.33**	259724	5.46	259724	3.04
11-9	0.01	0.01	x	x	3.15M	**51.06**		x		x	5603	**0.92**	180123	3.87	20200	1.13	20200	**0.95**	7243	**0.72**	5.60M	79.71	5.60M	43.20
11-10	0.01	0.01	x	x	25.97M	**444.41**		x		x	5520	**0.52**	326837	6.96	25048	1.54	25048	**0.75**	7804	**0.64**	249987	5.65	249987	2.95
11-11	0.02	0.01	x	x	193	**0.34**		x		x	193	**0.35**	182	**0.36**	193	0.51	193	0.38	182	**0.35**	2.62M	65.78	2.62M	28.19
12-10	0.02	0.02	x	x		x		x		x	7671	**0.70**	384911	7.89	27666	1.61	27666	**1.10**	10272	**0.78**		x		x
12-11	0.03	0.02	x	x	210	**0.37**		x		x	210	**0.38**	198	**0.34**	210	0.50	210	0.42	198	**0.40**		x		x
12-12	0.03	0.02	x	x	228	**0.41**		x		x	228	**0.46**	216	0.54	228	0.45	228	**0.41**	216	**0.42**	27.76M	1043.05	27.76M	372.65
13-11	0.05	0.04	x	x		x		x		x	48823	**1.82**	2.93M	68.28	213263	12.06	213263	**3.53**	86660	**2.85**		x		x
13-12	0.05	0.04	x	x		x		x		x	51164	**2.03**	2.44M	68.76	517874	17.45	517874	**5.14**	91543	**3.25**	27.79M	980.00	27.79M	368.82
13-13	0.07	0.04	x	x	267	**0.50**		x		x	267	**0.49**	254	0.51	267	0.65	267	**0.49**	254	**0.53**		x		x
14-12	0.10	0.08	x	x		x		x		x	65417	**2.28**	2.18M	58.67	553929	18.97	553929	**5.70**	118746	**4.22**		x		x
14-13	0.11	0.08	x	x	287	**0.59**		x		x	287	**0.54**	273	**0.55**	287	0.74	287	0.57	273	**0.62**		x		x
14-14	0.13	0.08	x	x	308	**0.62**		x		x	308	**0.61**	294	**0.58**	308	0.80	308	**0.58**	294	**0.70**		x		x
15-13	0.19	0.18	x	x		x		x		x	499483	**13.53**	19.95M	570.52	3.18M	216.39	3.18M	**48.53**	1.12M	**40.14**		x		x
15-14	0.21	0.18	x	x		x		x		x	512317	**15.25**	29.38M	1481.88	29.25M	604.42	29.25M	**164.66**	1.16M	**45.22**		x		x
15-15	0.25	0.19	x	x	353	**0.75**		x		x	353	**0.77**	338	**0.60**	353	0.94	353	0.74	338	**0.71**		x		x
20-18	20.83	21.02	x	x		x		x		x		x		x		x		x		x		x		x
20-19	21.33	21.02	x	x	590	**8.67**		x		x	590	**6.31**	570	**6.03**	590	6.86	590	7.85	570	**7.28**		x		x
20-20	22.00	20.92	x	x	620	**6.53**		x		x	620	**6.44**	600	**6.34**	620	7.73	620	6.56	600	**7.57**		x		x

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

[Bacchus *et al.*, 2002] F. Bacchus, X. Chen, P. van Beek, and T. Walsh. Binary vs. non-binary constraints. *Artificial Intelligence*, 140(1–2):1–37, 2002.

[Bessière *et al.*, 2008] C. Bessière, K. Stergiou, and T. Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6–7):800–822, 2008.

[Boussemart *et al.*, 2004] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proc. of ECAI-04*, pages 146–150, 2004.

[Dechter, 2003] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[Janssen *et al.*, 1989] 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 *Proc. of IEEE Workshop on Tools for Artificial Intelligence*, pages 420–427, 1989.

[Karakashian *et al.*, 2010] S. Karakashian, R. Woodward, C. Reeson, B. Y. Choueiry, and C. Bessiere. A first practical algorithm for high levels of relational consistency. In *Proc. of AAAI-10*, pages 101–107, 2010.

[Lecoutre and Vion, 2008] C. Lecoutre and J. Vion. Enforcing arc consistency using bitwise operations. *Constraint Programming Letters*, 2:21–35, 2008.

[Lecoutre *et al.*, 2013] C. Lecoutre, A. Paparrizou, and K. Stergiou. Extending STR to a higher-order consistency. In *Proc. of AAAI-13*, pages 576–582, 2013.

[Lecoutre, 2011] C. Lecoutre. STR2: Optimized simple tabular reduction for table constraints. *Constraints*, 16(4):341–371, 2011.

[Likitvivatanavong *et al.*, 2014] C. Likitvivatanavong, W. Xia, and R. H. C. Yap. Higher-order consistencies through GAC on factor variables. In *Proc. of CP-14*, pages 497–513, 2014.

[Mairy *et al.*, 2014] J. Mairy, Y. Deville, and C. Lecoutre. Domain k-wise consistency made as simple as generalized arc consistency. In *Proc. of CPAIOR-14*, pages 235–250, 2014.

[Merchez *et al.*, 2001] S. Merchez, C. Lecoutre, and F. Boussemart. AbsCon: a prototype to solve CSPs with abstraction. In *Proc. of CP-01*, pages 730–744, 2001.

[Michel and Hentenryck, 2012] L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Proc. of CPAIOR-12*, pages 228–243, 2012.

[Paparrizou and Stergiou, 2012] A. Paparrizou and K. Stergiou. An efficient higher-order consistency algorithm for table constraints. In *Proc. of AAAI-12*, pages 535–541, 2012.

[Refalo, 2004] P. Refalo. Impact-based search strategies for constraint programming. In *Proc. of CP-04*, pages 557–571, 2004.

[Samaras and Stergiou, 2005] N. Samaras and K. Stergiou. Binary encoding of non-binary constraint satisfaction problems: Algorithms and experimental results. *JAIR*, 24:641–684, 2005.

[Schneider *et al.*, 2014] A. Schneider, R. J. Woodward, B. Y. Choueiry, and C. Bessiere. Improving relational consistency algorithms using dynamic relation partitioning. In *Proc. of CP-14*, pages 688–704, 2014.

[Ullmann, 2007] J. R. Ullmann. Partition search for non-binary constraint satisfaction. *Information Science*, 177(18):3639–3678, 2007.

[Woodward *et al.*, 2014] R. J. Woodward, A. Schneider, B. Y. Choueiry, and C. Bessiere. Adaptive parameterized consistency for non-binary CSPs by counting support. In *Proc. of CP-14*, pages 755–764, 2014.