Bipartite Encoding: A New Binary Encoding for Solving Non-Binary CSPs

Ruiwei Wang and **Roland H.C. Yap**
School of Computing, National University of Singapore
{ruiwei,ryap}@comp.nus.edu.sg

Abstract

Constraint Satisfaction Problems (CSPs) are typically solved with Generalized Arc Consistency (GAC). A general CSP can also be encoded into a binary CSP and solved with Arc Consistency (AC). The well-known Hidden Variable Encoding (HVE) is still a state-of-the-art binary encoding for solving CSPs. We propose a new binary encoding, called Bipartite Encoding (BE) which uses the idea of partitioning constraints. A BE encoded CSP can achieve a higher level of consistency than GAC on the original CSP. We give an algorithm for creating compact bipartite encoding for non-binary CSPs. We present a AC propagator on the binary constraints from BE exploiting their special structure. Experiments on a large set of non-binary CSP benchmarks with table constraints using the Wdeg, Activity and Impact heuristics show that BE with our AC propagator can outperform existing state-of-the-art GAC algorithms (CT, STRbit) and binary encodings (HVE with HTAC).

1 Introduction

Many combinatorial problems from AI can be naturally modelled as Constraint Satisfaction Problems (CSPs) with constraints. Typically solving a (non-binary) CSP uses Generalized Arc Consistency (GAC) on the (non-binary) constraints. Alternatively, a non-binary CSP can be transformed with binary encoding into a CSP with only binary constraints so that Arc Consistency (AC) is applicable. While AC is simpler than GAC and binary encodings are well known, it is only recently that solving with binary encoding is shown to be practical. The hidden variable encoding with the AC algorithm HTAC [Wang and Yap, 2019] was shown to be competitive with state-of-the-art GAC algorithms on non-binary CSPs.

Bit representation is the key to the efficiency of many state-of-the-art GAC/AC algorithms for table constraints. A recent survey describes improvements in GAC algorithms [Yap *et al.*, 2020]). Here, we summarize some recent GAC algorithms and ideas. Bit sets are used to represent variable domains in AC3^{bit}[Lecoutre and Vion, 2008]. GAC algorithms STRbit [Wang *et al.*, 2016] and CT [Demeulenaere *et al.*, 2016] optimize the simple tabular reduction algorithms [Lecoutre, 2011; Lecoutre *et al.*, 2012] representing constraint relations as bit sets. Recently, bit representation were extended to handle compact representations and high-order consistencies. However, experiments show that the CT algorithm is still overall faster than these newer algorithms (see the experimental results of PW-CT [Schneider and Choueiry, 2018], compact-MDD [Verhaeghe *et al.*, 2018] and smart MDD [Verhaeghe *et al.*, 2019]).

The hidden variable encoding (HVE) [Rossi *et al.*, 1990], dual encoding [Dechter and Pearl, 1989] and double encoding [Stergiou and Walsh, 1999] are well known binary encodings for reducing non-binary CSPs to binary CSPs. AC on a HVE instance achieves GAC on the original CSP but is weaker than AC on the dual/double encoding instance [Bessiere *et al.*, 2008]. However, the drawback is that dual/double encoding instances may be much larger than HVE instances. HVE, proposed 30 years ago, is the state-of-the-art binary encoding [Wang and Yap, 2019]. We observe that binary encoding instances have special structure. Specialized AC algorithms, such as HAC, PW-AC [Samaras and Stergiou, 2005] and HTAC have been proposed to handle binary encoded instances. In particular, HTAC is competitive with CT, suggesting that AC algorithms on binary encoded instances have the potential to outperform the state-of-the-art GAC algorithms.

In this paper, we propose a new binary encoding, called bipartite encoding (BE), which encodes constraints as binary constraints between factor variables which partition the constraint scopes. AC on a BE encoded CSP is stronger than GAC on the original CSP. We give an algorithm to construct BE instances, followed by a AC propagator AC-BE which treats a connected component in the binary encoding as a "special constraint". We evaluate the bipartite encoding to solve non-binary CSPs comparing with state-of-the-art GAC algorithms and the binary encoding HVE (with HTAC) on a large set of benchmarks. The results show that the BE propagator outperforms CT, STRbit and HTAC across many variable heuristics (Wdeg, Activity, Impact). The usefulness of higher consistency is also shown to be effective making some instances backtrack free.

The remainder of the paper is organized as follows. Section 2 provides preliminaries. The bipartite encoding, an algorithm to create BE instances and a special propagator for BE are given in Sections 3, 4 and 5. Experiments are presented in Section 6 and Section 7 concludes.

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

(a) c_1

x_1	x_2	x_5	x_6
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	1	0	0

(b) c_2

fv_1	fv_2
0	1
0	2
1	0

(c) c_1^*

fv_1	fv_3
0	1
0	3
1	1
1	2

(d) c_2^*

fv_1	x_1^f
0	0
1	0

(e) $c_1^{x_1}$

fv_1	x_2^f
0	0
1	1

(f) $c_1^{x_2}$

fv_2	x_3^f
0	0
1	0
2	1

(g) $c_2^{x_3}$

fv_2	x_4^f
0	0
1	1
2	0

(h) $c_2^{x_4}$

fv_3	x_5^f
1	0
2	1
3	1

(i) $c_3^{x_5}$

fv_3	x_6^f
1	1
2	0
3	1

(j) $c_3^{x_6}$

Figure 1: Bipartite encoding example

2 Preliminaries

A CSP $\mathcal{P}$ is a pair $(\mathcal{X},\mathcal{C})$ where $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ is a set of n variables, $\mathcal{D}(x_i)$ is the domain of variable x_i, and $\mathcal{C} = \{c_1, c_2, \ldots, c_e\}$ is a set of e constraints. In this paper, we assume that $\mathcal{D}(x)$ is finite, also known as *finite domain*. A *literal* of a variable x is a variable value (x, a). A tuple over a set of variables $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ is a set of literals $\{(x_{i_1}, a_1), (x_{i_2}, a_2), \ldots, (x_{i_r}, a_r)\}$. Each constraint c_j consists of a constraint scope $scp(c_j) \subseteq \mathcal{X}$ and a relation $rel(c_j)$ which is defined by a set of tuples over $scp(c_j)$. The arity of constraint c is the number of variables in the constraint scope, i.e. $|scp(c)|$. $\mathcal{P}$ is a r-arity CSP if the largest constraint arity is r. A 2-arity CSP is also called a *binary CSP*. Without loss of generality, all 1-arity constraints are removed by modifying variable domains. A *non-binary* CSP is a r-arity CSP where $r > 2$. We assume the CSP is normalized, i.e. for each constraint $c_i \in \mathcal{C}$ such that $scp(c_i) \subseteq scp(c_j)$, c_i is removed by joining c_i with c_j.

The *projection* of a tuple τ (or a set T of tuples) on a set S of variables, denoted by $\tau[S]$ (or $T[S]$), is $\{(x, a) \in \tau | x \in S\}$ (or $\{\tau[S] | \tau \in T\}$). A tuple τ is *valid* iff for all literals $(x, a) \in \tau$, the value a is in $\mathcal{D}(x)$ and x only appears once in τ. A tuple τ over variables $X \subseteq \mathcal{X}$ is *consistent* iff $\tau[scp(c)] \in rel(c)$ for all constraints $c \in \mathcal{C}$ such that $scp(c) \subseteq X$. A *solution* of P is a consistent and valid tuple over $\mathcal{X}$. $\mathcal{P}$ is satisfiable iff there is a solution over $\mathcal{X}$. A *support of a value* $a \in \mathcal{D}(x)$ on a constraint c is a tuple $\tau \in rel(c)$ such that $(x, a) = \tau[x]$. A *support of a tuple* $\tau_1 \in rel(c_i)$ on a constraint c_j is a tuple $\tau_2 \in rel(c_j)$ such that $\tau_2[scp(c_i)] = \tau_1[scp(c_j)]$. For binary CSPs, $b \in \mathcal{D}(y)$ is a support of $a \in \mathcal{D}(x)$ on y if the tuple $\{(x, a), (y, b)\}$ is a support of a on the constraints between x and y.

A variable $x \in scp(c_i)$ is *Generalized Arc Consistent* (GAC) on a constraint $c_i \in \mathcal{C}$ if a has a valid support on c_i for all $a \in \mathcal{D}(x)$. A constraint c_i is GAC if all variables in $scp(c_i)$ are GAC on c_i. A CSP $(\mathcal{X},\mathcal{C})$ is GAC if all constraints in $\mathcal{C}$ are GAC. For binary CSPs, GAC is also called *Arc Consistent* (AC). GAC is a first-order consistency, filtering variable domains. We also introduce a higher-order consistency to further filter inconsistent tuples. A tuple $\tau \in rel(c_i)$ is *Pairwise Consistent* (PWC) if τ has a valid support on c_j for all $c_j \in \mathcal{C}$ [Janssen *et al.*, 1989]. A constraint c_i is PWC if all tuples $\tau \in rel(c_i)$ are PWC. A CSP $(\mathcal{X},\mathcal{C})$ is PWC if all constraints in $\mathcal{C}$ are PWC. A CSP $\mathcal{P}$ is *Full Pairwise Consistent* (FPWC) if $\mathcal{P}$ is PWC and GAC [Lecoutre *et al.*, 2013; Likitvivatanavong *et al.*, 2014].

3 Bipartite Encoding

We start with an observation that in a binary CSP, every binary constraint is between two variables which partition the constraint scope. So we can think of a binary CSP as being encoded with such a partitioning. We generalize this idea of encoding to a new binary encoding of non-binary CSPs called bipartite encoding which can also give stronger consistency than GAC.

Definition 1. *Given a CSP $\mathcal{P} = (\mathcal{X},\mathcal{C})$, a* factor variable fv *over a non-empty set of variables $S \subseteq \mathcal{X}$ is a new variable such that $\mathcal{D}(fv)$ is a set of tuples over S, and $\tau[S] \in \mathcal{D}(fv)$ for all solutions τ of $\mathcal{P}$. We use $scp(fv) = S$ to denote the scope of the variables covered by fv. A factor variable fv is* original *if $|scp(fv)| = 1$ and* compound *if $|scp(fv)| > 1$.*

The minimum domain of a factor variable fv consists of all tuples τ over $scp(fv)$ which can be extended to solutions of the CSP. As such, it is NP-hard to find the minimum domain of fv. We propose to use some local consistent tuples to construct $\mathcal{D}(fv)$.

Definition 2. *A* bipartite encoding $BE(\mathcal{P})$ *of a CSP $\mathcal{P} = (\mathcal{X},\mathcal{C})$ is a CSP $(\mathcal{X}^+ \cup \mathcal{X}^*, \mathcal{C}^+ \cup \mathcal{C}^*)$, where*

- *Variables are either compound or original factor variables denoted by $\mathcal{X}^*$ and $\mathcal{X}^+$ respectively and for all variables $fv_i, fv_j \in \mathcal{X}^* \cup \mathcal{X}^+$, $scp(fv_i) \neq scp(fv_j)$ if $i \neq j$.*
- Partition constraints $\mathcal{C}^*$*: for each $c_i \in \mathcal{C}$, $c_i^* \in \mathcal{C}^*$ is a binary constraint such that $scp(c_i^*)$=$\{fv_j, fv_k\}$, $\{scp(fv_j), scp(fv_k)\}$ is either a disjoint or non-disjoint partition of $scp(c_i)$, and $rel(c_i^*)$=$\{\{\tau_1, \tau_2\} | \tau_1 \in \mathcal{D}(fv_j), \tau_2 \in \mathcal{D}(fv_k), (\tau_1 \cup \tau_2) \in rel(c_i)\}$.*
- Mapping constraints $\mathcal{C}^+$*: for each $fv_i \in \mathcal{X}^*$ and $x \in scp(fv_i)$, $c_i^x \in \mathcal{C}^+$ is a binary constraint such that $scp(c_i^x) = \{fv_i, fv_j\}$, $scp(fv_j) = \{x\}$, and $rel(c_i^x) = \{\{\tau_1, \tau_2\} | \tau_1 \in \mathcal{D}(fv_j), \tau_1 \in \mathcal{D}(fv_j), \tau_2 = \tau_1[scp(fv_j)]\}$.*

For all factor variables fv_i, the information of the tuples in the domain $\mathcal{D}(fv_i)$ is recorded in the mapping constraints $\{c_i^x | x \in scp(fv_i)\}$, which ensures that the tuple $\tau_1 \cup \ldots \cup \tau_m$ is an assignment of $\mathcal{P}$ if the tuple $\{(fv_1, \tau_1),\ldots(fv_m, \tau_m)\}$ is consistent on $BE(\mathcal{P})$. We remark that the mapping constraints used are similar to the constraints in HVE encoding and there may be other mapping constraints ensuring the same property. Additionally, for each partition constraint $c_i^* \in \mathcal{C}^*$, all tuples $\{(fv_j, \tau_1), (fv_k, \tau_2)\}$ in $rel(c_i^*)$ correspond to the tuples $\tau_1 \cup \tau_2$ in $rel(c_i)$, which ensures that the tuple $\tau_1 \cup \ldots \cup \tau_m$ is a solution of $\mathcal{P}$ if the tuple $(fv_1, \tau_1),\ldots(fv_m, \tau_m)\}$ is a solution of $BE(\mathcal{P})$.

Algorithm 1: BE($\mathcal{X}$,$\mathcal{C}$)

1 $scpp \leftarrow partition(\mathcal{C})$;
  $S \leftarrow (\bigcup_{c_i \in \mathcal{C}} scpp[c_i]) \cup (\bigcup_{x \in \mathcal{X}} \{\{x\}\})$;
2 $V \leftarrow \emptyset$;
  for $S_j \in S$ **do**
    Generate a factor variable fv_j & domain on S_j;
3   $V \leftarrow V \cup \{fv_j\}$;
4 $\mathcal{C}^* \leftarrow \emptyset$;
  for $c_i \in \mathcal{C}$ **do**
    $\{S_j, S_k\} \leftarrow scpp[c_i]$;
    Construct a constraint c_i^* between fv_j and fv_k;
    $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup \{c_i^*\}$;
5 Construct constraints $\mathcal{C}^+$ with factor variables in V;
  return $(V, \mathcal{C}^* \cup \mathcal{C}^+)$;

Algorithm 2: partition($\mathcal{C}$)

  $cur \leftarrow \mathcal{C}$;
  while $cur \neq \emptyset$ **do**
1   $E \leftarrow maxEdge(cur)$;
    if $E = \emptyset$ **then**
      break;
2   **for** $\{c_i, c_j\} \in E$ **do**
      $S_k \leftarrow scp(c_i) \cap scp(c_j)$;
      $scpp[c_i] \leftarrow \{S_k, scp(c_i) \setminus S_k\}$;
3     $scpp[c_j] \leftarrow \{S_k, scp(c_j) \setminus S_k\}$;
4 **for** $c_i \in cur$ **do**
    Select a variable x from $scp(c_i)$;
5   $scpp[c_i] = \{scp(c_i) \setminus \{x\}, \{x\}\}$;
  return $scpp$;

Algorithm 3: maxEdge(cur)

  Let E be all maximum edges in cur;
  $S \leftarrow \{scp(c_i) \cap scp(c_j) | \{c_i, c_j\} \in E\}$;
  for $S_k \in S$ **do**
1   $size[S_k] \leftarrow$ domain size of a factor variable on S_k;
2   $N[S_k] \leftarrow \bigcup\{e \in E | e \subseteq cur, scp(e) = S_k\}$;
  $E_r \leftarrow \emptyset$;
  while $S \neq \emptyset$ **do**
3   Let M be the maximum cardinality subsets in S;
4   Select S_k from M such that $\frac{size[S_k]}{|N[S_k]|}$ is minimum;
    $S \leftarrow S \setminus \{S_k\}$;
5   $A \leftarrow \{c \in (N[S_k] \cap cur) |$ c is size_splittable by $S_k\}$;
    $E_A \leftarrow \{e \in E | scp(e) = S_k, e \subseteq A\}$;
6   $E_r \leftarrow E_r \cup E_A$;
    $cur \leftarrow cur \setminus (\bigcup_{e \in E_A} e)$;
  return E_r;

Example 1. *Let CSP $\mathcal{P}$=($\mathcal{X}$, $\mathcal{C}$) where $\mathcal{C}$={c_1, c_2}, $\mathcal{X}$={x_1, $\cdots$, x_6} and domains are {0,1}. Constraint scopes are: $scp(c_1)$={x_1, $\cdots$, x_4} and $scp(c_2)$={x_1, x_2, x_5, x_6} with relations given in Figure 1a, 1b. A bipartite encoding ({fv_1, fv_2, fv_3, x_1^f, $\cdots$, x_6^f}, {c_1^*, c_2^*, $c_1^{x_1}$, $\cdots$, $c_3^{x_6}$}) of $\mathcal{P}$ is shown in Figures 1c,$\cdots$,1j. Constraints c_1 and c_2 are represented as c_1^* and c_2^*, where $scp(c_1^*) = \{fv_1, fv_2\}$ and $scp(c_2^*) = \{fv_1, fv_3\}$. fv_1, fv_2 and fv_3 are compound factor variables on $S_1 = \{x_1, x_2\}$, $S_2 = \{x_3, x_4\}$ and $S_3 = \{x_5, x_6\}$. x_i^f is an original factor variable over $\{x_i\}$ for $1 \leq i \leq 6$. For each factor variable on $\{x, y\}$, we use values 0, 1, 2 and 3 to denote tuples $\{(x,0),(y,0)\}$, $\{(x,0),(y,1)\}$, $\{(x,1),(y,0)\}$ and $\{(x,1),(y,1)\}$, respectively. For each original factor variable x_i^f, value a denotes $\{(x_i, a)\}$. Each solution of $BE(\mathcal{P})$ corresponds to a solution of $\mathcal{P}$, e.g. {(fv_1,0), (fv_2,1), (fv_3,3), (x_1^f,0), (x_2^f,0), (x_3^f,0), (x_4^f,1), (x_5^f,1), (x_6^f,1)} corresponds to {(x_1,0), (x_2,0), (x_3,0), (x_4,1), (x_5,1), (x_6,1)}.*

For a bipartite encoding $BE(\mathcal{P}) = (\mathcal{X}^* \cup \mathcal{X}^+, \mathcal{C}^* \cup \mathcal{C}^+)$, every original factor variable in $\mathcal{X}^+$ (or constraint in $\mathcal{C}^*$) corresponds to a variable (or constraint) in $\mathcal{P}$. An *image* $\mathcal{P}'$ of $BE(\mathcal{P})$ is a CSP $(\mathcal{X}', \mathcal{C}')$ such that $\mathcal{X}' = \{x' | x \in \mathcal{X}\}$ and $\mathcal{C}' = \{c_i' | c_i \in \mathcal{C}\}$, where $\mathcal{D}(x')$={$a | (x,a) \in \mathcal{D}(x^f)$}, $scp(c_i') = \{x' | x \in scp(c_i)\}$ and $rel(c_i') = \{\{(x',a) | (x,a) \in \tau_1 \cup \tau_2\} \mid \{(fv_j, \tau_1), (fv_k, \tau_2)\} \in rel(c_i^*)\}$. The image $\mathcal{P}'$ of $BE(\mathcal{P})$ has the same variables and constraints as $\mathcal{P}$, except the domains and relations of $\mathcal{P}'$ may be reduced. $\mathcal{P}'$ is useful for comparing the consistencies on $BE(\mathcal{P})$ with that on $\mathcal{P}$.

Proposition 1. *The image $\mathcal{P}'$ of $BE(\mathcal{P})$ is GAC if $BE(\mathcal{P}) = (\mathcal{X}^* \cup \mathcal{X}^+, \mathcal{C}^* \cup \mathcal{C}^+)$ is AC.*

Proof. Assume $scp(c_i^*) = \{fv_j, fv_k\}$, $|scp(fv_j)| > 1$, $x \in scp(fv_j)$, and $a \in \mathcal{D}(x')$. $BE(\mathcal{P})$ is AC, hence, $(x^f, \{(x,a)\})$ has a valid support on $\tau_1 \in \mathcal{D}(fv_j)$ and τ_1 has a valid support $\tau_2 \in \mathcal{D}(fv_k)$. So $\{(y',b) | (y',b) \in \tau_1 \cup \tau_2\}$ is a valid support of (x',a) on c_j'. ☐

AC on $BE(\mathcal{P})$ can be stronger than GAC on $\mathcal{P}$ (see Proposition 2). For example, $(x_1^f, 1)$ is not AC on the BE instance given in Figure 1, but the original CSP is GAC.

4 A Bipartite Encoding Algorithm

We present Algorithm 1 for constructing a bipartite encoding instance from a CSP $(\mathcal{X}, \mathcal{C})$. It is based on partitions of constraint scopes and has three parts given in Algorithm 1:

(i) For each constraint $c_i \in \mathcal{C}$, the partition function split $scps(c_i)$ into 2 subsets $scpp[c_i]$ (Line 1—see Sec. 4.1).

(ii) Generating factor variables (Lines 2-3). We use the projection $\bigcap\{rel(c)[scp(fv)] | c \in \mathcal{C}, scp(fv) \subseteq scp(c)\}$ as the domain of a factor variable fv, and remove invalid tuples from constraint relations after generating each factor variable, e.g. for the CSP in Figure 1, we can remove the tuple $\{(x_1,1),(x_2,1),(x_5,0),(x_6,0)\}$ from $rel(c_2)$ after generating the factor variable fv_1 on $\{x_1, x_2\}$. As $\{(x_1,1),(x_2,1)\}$ is removed, then $\{(x_5,0),(x_6,0)\}$ is not included in the domain of the factor variable fv_3 on $\{x_5, x_6\}$.

(iii) Constructing bipartite encoding instance based on the generated factor variables and partitions of constraint scopes (Lines 4-5)—this is a straightforward construction using Definition 2.

The number of possible partitions of constraint scopes is exponential in the number of constraints and constraint arity. Thus, many BE instances can be constructed for a CSP by applying different partitions. We introduce a heuristic used to generate some disjoint partitions in the following subsection.

4.1 Maximum Edge Partition

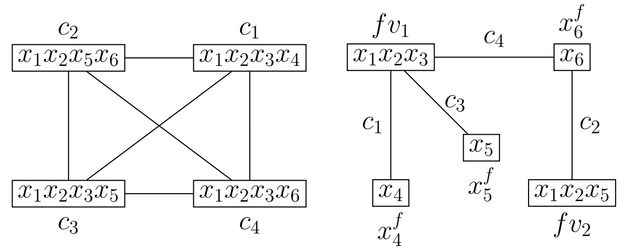
We now give a method to partition the non-binary constraints taking into account obtaining higher level consistency where feasible. We first introduce some notations. A dual graph [Dechter and Pearl, 1989] of a CSP $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ is a undirected graph such that constraints $\mathcal{C}$ are nodes and edges are $\{\{c_i, c_j\} \subseteq \mathcal{C} | scp(c_i) \cap scp(c_j) \neq \emptyset\}$. We use $scp(e) = scp(c_i) \cap scp(c_j)$ to denote the scope of variables covered by a edge $e = \{c_i, c_j\}$. The size of an edge e is the cardinality of $scp(e)$. The edge $e = \{c_i, c_j\}$ is a maximum edge of c_i in $C \subseteq \mathcal{C}$ if the size of e is greater than 1 and largest on all edges in C including c_i, and e is a maximum edge in C if e is a maximum edge of c_i or c_j in C. A bipartite encoding $BE(\mathcal{P})$ covers an edge $e = \{c_i, c_j\}$ if there exists a factor variable fv in $scp(c_i^*) \cap scp(c_j^*)$ such that $scp(fv) = scp(e)$, e.g. the bipartite encoding given in Figure 1 covers the edge $\{c_1, c_2\}$, since $scp(c_1^*) \cap scp(c_2^*)$ includes a factor variable fv_1 on $\{x_1, x_2\}$. Our strategy is that for overall propagation efficiency, the size of the encoding should be compact, in particular, we will focus on bit representations used in GAC/AC propagators. We propose a bipartite encoding heuristic where the encoding is compact and can cover some maximum edges for higher level consistency (see Proposition 2).

Proposition 2. *The image $\mathcal{P}'$ of $BE(\mathcal{P})$ is FPWC if $BE(\mathcal{P})$ is AC and covers all edges whose sizes are greater than 1.*

Proof. For all $c_i', c_j' \in \mathcal{C}'$ and $\tau \in rel(c_i')$ such that $S = scp(c_i) \cap scp(c_j)$ includes at least 2 variables, τ has a valid support on c_j', since $\tau[S]$ is in $\mathcal{D}(fv_k)$ and has valid supports on c_j^*, where $fv_k \in scp(c_i^*) \cap scp(c_j^*)$ is a factor variable on S. Recall, $\mathcal{P}'$ is AC (Proposition 1), so $\mathcal{P}'$ is FPWC. $\square$

Algorithm 2 generates partitions of constraint scopes. The data structure $scpp[c_i]$ records the partition of $scp(c_i)$. Algorithms 2 and 3 use a global set cur to record a set of constraints which are not partitioned. At Line 1, Algorithm 3 is called to generate a subset E of maximum edges in cur such that: (i) for each $c_i \in cur$, there is at most one partition of $scp(c_i)$ based on E, i.e. $|\{scp(e)|e \in E, c_i \in e\}| \leq 1$; and (ii) for each edge $e = \{c_i, c_j\}$ in E, the constraints c_i and c_j are size_splittable by $scp(e)$. If E is not empty, we use the maximum edges in E to partition some constraint scopes (Lines 2-3), otherwise we use a basic partitioning—for each constraint c_i in cur, we select the last variable x in $scp(c_i)$ and partition $scp(c_i)$ (between Lines 4 and 5).

A constraint c_k is size_splittable by a variable subset $S_i \subset scp(c_k)$ or $S_j = scp(c_k) \setminus S_i$ if the size of the bit representation (see data structures used in HTAC [Wang and Yap, 2019]) of c_k is greater than or equal to that of the corresponding constraints in the BE encoding, namely, the sum of the sizes of the binary constraints c_k^* and $\{c_l^x \in \mathcal{C}^+ | scp(fv_l) \in \{S_i, S_j\}, x \in scp(fv_l)\}$.[1] We propose to evaluate the size of the original and bipartite instance as follows. We use $|rel(c_k)|(\sum_{x \in scp(c_k)} |\mathcal{D}(x)|)$ to evaluate the size of the bit representation of c_k. For each binary constraint in the BE encoding, we use $|\mathcal{D}(fv_{l_1})| \times |\mathcal{D}(fv_{l_2})|$ to estimate the size of the constraint, where $\{fv_{l_1}, fv_{l_2}\}$ is the constraint scope.



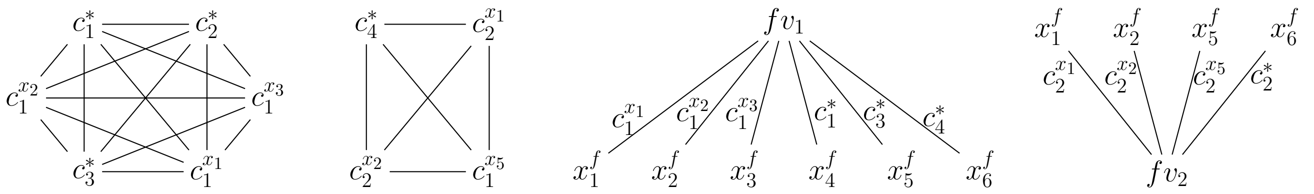
(a) Dual graph (b) Partitions

Figure 2: Maximum edge partition

Algorithm 3 gives our maximum edges generation algorithm. The data structure $size[S_k]$ records the domain size of the factor variable fv_k on S_k (at Line 1). At Line 2, $N[S_k]$ is a set of constraints where the constraint scopes may be partitioned by S_k. At Lines 3 and 4, we first select a variable subset S_k from S with maximum cardinality, and further select those with minimum $\frac{size[S_k]}{N[S_k]}$. Lines 5-6 evaluate which constraints in $N[S_k]$ are size_splittable by S_k, recording the corresponding maximum edges in E_r. We remark that we also delete some invalid tuples when calculating the domain size of a factor variable.

Example 2. *Let $\mathcal{P}$ be a CSP with constraints $\{c_1, \cdots, c_4\}$, variables $\{x_1, \cdots, x_6\}$ and domains $\{0, 1\}$. c_1, c_2, c_3 and c_4 are the linear equations: $x_1+x_2+x_3+x_4$=1, $x_1+x_2+x_5+x_6$=2, $x_1+x_2+x_3+x_5$=2 and $x_1+x_2+x_3+x_6$=1. Figure 2a is the dual graph of $\mathcal{P}$. Figure 2b gives the partitions generated by our algorithm. There are 5 maximum edges, the scopes are S_1=$\{x_1, x_2, x_3\}$, S_2=$\{x_1, x_2, x_5\}$ and S_3=$\{x_1, x_2, x_6\}$ with $N[S_1]$=$\{c_1, c_3, c_4\}$, $N[S_2]$=$\{c_2, c_3\}$ and $N[S_3]$=$\{c_2, c_4\}$. We generate the factor variables on S_1, S_2 and S_3, and then record the corresponding domain sizes. At Line 4 in Algorithm 3, S_1 is selected. c_1, c_3 and c_4 are size_splittable by S_1, thus, $\{c_1, c_3\}$, $\{c_1, c_4\}$ and $\{c_3, c_4\}$ are the maximum edges selected. Then $\{S_1, \{x_4\}\}$, $\{S_1, \{x_5\}\}$ and $\{S_1, \{x_6\}\}$ are the partitions of $scp(c_1)$, $scp(c_3)$ and $scp(c_4)$. After partitioning c_1, c_3 and c_4, there are no more maximum edges, so we select x_6 from $scp(c_2)$, with partition of $scp(c_2)$ being $\{\{x_1, x_2, x_5\}, \{x_6\}\}$. Then we construct a BE instance ($\mathcal{X}^* \cup \mathcal{X}^+$, $\mathcal{C}^* \cup \mathcal{C}^+$) such that $\mathcal{X}^*$=$\{fv_1, fv_2\}$, $\mathcal{X}^+$=$\{x_1^f, x_2^f, x_3^f, x_4^f, x_5^f, x_6^f\}$, $\mathcal{C}^*$=$\{c_1^*, c_2^*, c_3^*, c_4^*\}$, and $\mathcal{C}^+$=$\{c_1^{x_1}, c_1^{x_2}, c_1^{x_3}, c_2^{x_1}, c_2^{x_2}, c_2^{x_5}\}$, where $scp(fv_1)$=S_1, $scp(fv_2)$=S_2, $scp(x_i^f)$=$\{x_i\}$ for $1 \leq i \leq 6$, $scp(c_1^*)$=$\{fv_1, x_4^f\}$, $scp(c_2^*)$=$\{fv_2, x_6^f\}$, $scp(c_3^*)$=$\{fv_1, x_5^f\}$ and $scp(c_4^*)$=$\{fv_1, x_6^f\}$.*

[1] We remark that experiments show HTAC on the HVE encoding is competitive with CT on the original CSP, and as the experiments show, we want to improve on both. This means that attention is also needed on the size of the bit representations of the constraints.



(a) Component 1 (b) Component 2 (c) Component 1 (primal graph) (d) Component 2 (primal graph)

Figure 3: Graphs of components

5 AC on Bipartite Encoding Instances

HTAC is a state-of-the-art specialized AC algorithm for non-binary CSPs encoded by HVE. It exploits the structure of the binary CSP arising from the encoding— the set of binary constraints from encoding a non-binary constraint is regarded as a special subset of the constraints in HVE instance. Correspondingly, a specialized propagator AC-H is used for each subset with a star structure in the constraint graph. HTAC only needs to search on the original variables.

A BE instance is a binary CSP so any AC algorithm can be applied. However, in the same way that HTAC uses a specialized propagator, our AC-BE algorithm is a specialized propagator exploiting the structure of a BE instance $(\mathcal{X}^+ \cup \mathcal{X}^*, \mathcal{C}^+ \cup \mathcal{C}^*)$ extending the framework of HTAC. We partition binary constraints in the BE instance into a set of connected components in an undirected graph $(\mathcal{C}_1, E)$, and employ special propagators for the components, where $\mathcal{C}_1 = \mathcal{C}^+ \cup \mathcal{C}^*$ and $E = \{\{c_i, c_j\} \subseteq \mathcal{C}_1 | scp(c_i) \cap scp(c_j) \cap \mathcal{X}^* \neq \emptyset\}$, and every component is treated as a set of constraints. Meanwhile, we also only search on the original factor variables in $\mathcal{X}^+$. Due to lack of space, we refer readers to [Wang and Yap, 2019] for details on algorithms and data structures in HTAC.

Example 3. *There are 2 connected components for the BE instance given in Example 2. Figure 3a is the first component* $\{c_1^{x_1}, c_1^{x_2}, c_1^{x_3}, c_1^*, c_2^*, c_3^*\}$. *The constraint scopes of all constraints in the first component include the factor variable* fv_1. *Figure 3b shows the second component* $\{c_2^{x_1}, c_2^{x_2}, c_2^{x_5}, c_4^*\}$. *The compound factor variable* fv_2 *is included in the constraint scopes of all constraints in the second component. Note that the constraints in different components only share some original factor variables.*

Algorithm 4: AC-BE(C)

$V \leftarrow \bigcup\{scp(c) | c \in C\}$;
1 $T \leftarrow \emptyset, U \leftarrow C$;
while $\exists x \in V$ *s.t.* $|\{c \in U | x \in scp(c)\}| = 1$ **do**
  $T \leftarrow T \cup \{c \in U | x \in scp(c)\}$;
2   $U \leftarrow U \setminus \{c \in U | x \in scp(c)\}$;
3 **if** $\neg propagationUp(T)$ **then return** false ;
4 **if** $\neg AC(U)$ **then return** false ;
5 **if** $\neg propagationDown(T)$ **then return** false ;
for $x \in V \cap \mathcal{X}^+$ *s.t.* $\mathcal{D}(x)$ *is changed* **do**
6   Add x to the propagation queue;
return true;

Every connected component C in the graph $(\mathcal{C}_1, E)$ is regarded as a single "special constraint". Algorithm 4 presents a specialized propagation function for this component which enforces AC on all binary constraints in the component as follows. We first partition the binary constraints in C into two subsets T and U (between Lines 1 and 2) such that: (i) the primal graph of T is acyclic and (ii) every node in the primal graph of U is included in at least one cycle and (iii) every connected component in the primal graph of T has at most one node which is also included in the primal graph of U, where the primal graph of a set of binary constraints C' is a undirected graph $(\bigcup_{c \in C'} scp(c), \{scp(c) | c \in C'\})$. If U is empty, then the primal graph of C is acyclic, otherwise we have a special "star structure" such that the primal graph of U is the "root" and the trees included in the primal graph of T are the "leaves".

While an AC propagator can be used on the entire component C, we use a more efficient approach recognizing the tree structure in T. For the binary constraints in T, we call the revise functions to update the variable domains from leaves to the roots (the propagationUp function at Line 3), and then from roots to leaves (in the propagationDown function at Line 5), where for each connected component in the primal graph of T, a node is set as the root of the component, such that the nodes included in both the primal graphs of U and T are root nodes. For the binary constraints in U, we use a queue to record the variables which may cause reductions of other variable domains (same as the queue used in AC3^{bit}), and then iteratively propagate the variables in the queue (in the AC function at Line 4 which is a normal AC propagator). The propagationUp, propagationDown and AC functions employ some revise functions to update variable domains, and return false if there is a wipe-out of a domain, otherwise return true. At Line 6, if the domain of a variable $x \in \mathcal{X}^+$ is changed, then x is added to a propagation queue shared by all connected components of the graph $(\mathcal{C}_1, E)$, thus propagating to other constraints in $\mathcal{C}_1$.

Comparing to the normal AC propagators liking AC3^{bit}, the specialized AC propagator for BE instances has a different propagation ordering which iteratively uses Algorithm 4 to enforce AC on a connected component of the graph $(\mathcal{C}_1, E)$ until all connected components of $(\mathcal{C}_1, E)$ are AC, where a connected component is AC if all binary constraints in the connected component are AC.

Example 4. *Figure 3c and 3d shows the primal graph of the component given in Figure 3a and 3b respectively. Every edge in the primal graphs corresponds to a binary constraint* *in the BE instance. The primal graph of each component is acyclic and only includes one compound factor variable node. We set the compound factor variables as roots in the primal graphs. Correspondingly the propagationUp function updates the domains of compound factor variables (roots) based on the domains of original factor variables (leaves) and the propagationDown function updates the domains of leaves based on the domain of roots.*

In our implementation, as common with the implementation of AC propagators, some revise functions are used. Our revise functions are based on those from HTAC which use the following functions to update a variable domain $\mathcal{D}(x)$ based on another variable domain $\mathcal{D}(y)$: (i) *seekSupport* scans all values in the domain $\mathcal{D}(x)$, removing values which do not have any valid support on y; (ii) *reset* updates $\mathcal{D}(x)$ by using the union of all supports on x of the values in $\mathcal{D}(y)$; and (iii) *delete* removes all supports on x of the values which are removed from $\mathcal{D}(y)$. For a binary constraint with $scp(c) = \{x, y\}$, our implementation uses the revise operations as follows. The *seekSupport* operation is used when $x \in \mathcal{X}^+$ or $|\mathcal{D}(x)| \leq |\mathcal{D}(y)|$. Otherwise, *reset* or *delete* operations are used to update the domain $\mathcal{D}(x)$. We use the *delete* operation as an optimization when the values in $\mathcal{D}(x)$ have at most 1 support in $\mathcal{D}(y)$. We adapt data structures employing sparse sets [Briggs and Torczon, 1993], sparse bit sets [Demeulenaere *et al.*, 2016; Wang and Yap, 2019] and "ordered link" data structures [Lecoutre and Szymanek, 2006; Wang and Yap, 2019].

6 Experiments

Experiments presented in recent GAC algorithms (compact-MDD, smart MDD and PW-CT (enforces FPWC)) show CT to be state-of-the-art. HTAC is also shown to be comparable to CT, and better than STRbit. We compare our algorithm BE (AC on BE instances) with CT, STRbit and HTAC (AC on HVE instances). The experiments were run on a 3.20GHz Intel i7-8700 machine. All algorithms are implemented in the Abscon solver (https://www.cril.univ-artois.fr/~lecoutre/#/softwares). We tested with the 3 well known variable search heuristics Wdeg/Dom (Wdeg) [Boussemart *et al.*, 2004], Activity [Michel and Van Hentenryck, 2012] and Impact [Refalo, 2004] with the binary branching MAC and geometric restart strategy.[2] The value heuristic used is lexical value order. We measure CPU timings as: (i) solving time: time for solving the CSP; (ii) total time: initialization time (includes I/O, binary encoding, data structures) + solving time. Total CPU time is limited to 10 minutes per instance and memory to 8GB. We tested all 2559 non-binary instances,[3] which only employ table constraints, from the XCSP3 website (http://xcsp.org).

		STRbit	CT	HTAC	BE
Wdeg (682)	AvgR	3.50	2.68	1.99	-
	MaxR	91.46	80.98	19.2	-
	#F	3	37	89	**553**
	#TO	45	36	38	**3**
	#BF	0	0	0	**148**
Activity (655)	AvgR	3.58	2.76	2.80	-
	MaxR	90.54	80.01	77.83	-
	#F	4	87	41	**539**
	#TO	51	47	48	**3**
Impact (691)	AvgR	4.73	3.52	3.64	-
	MaxR	149.78	129.44	138.61	-
	#F	0	89	33	**569**
	#TO	59	44	43	**9**
	#BF	0	0	0	**175**
Initial Time (s)		1.39	1.37	1.38	1.65

Table 1: Relative comparison with Total Times

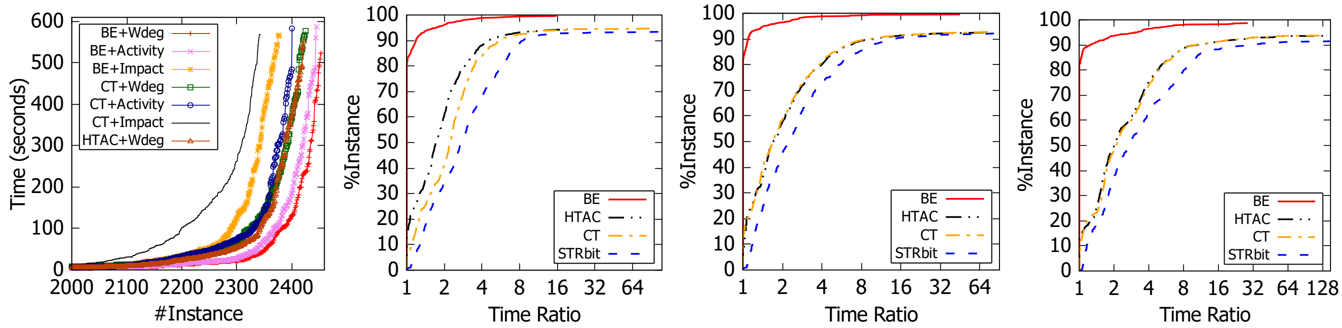
We first evaluate the relative performance of the GAC/AC algorithmic on each variable heuristic. To avoid timeout and small timings, for each variable heuristic, the trivial instances where all algorithms timeout or the solving time of the slowest algorithm is less than 2 seconds are ignored. Thus, different search heuristics have a different number of non-trivial instances. Table 1 gives a relative comparison between BE and other algorithms per search heuristic using *total time*: 682 instances for Wdeg; 655 for Activity; and 691 for Impact. AvgR is the average ratio of the total time of an algorithm to BE, and MaxR is the maximum total time ratio. The average speedup of BE is between 1.99 to 4.7X and maximum speedup between 19 to 149X. The *Initial Time* row gives the average initialization time on all 2559 instances, and BE is higher due to the time needed for encoding. #F (fastest) is the number of instances on which an algorithm has the smallest total time. The number of instances on which an algorithm is backtrack free is #BF and timeout is #TO. Overall BE solves more instances than CT, HTAC and STRbit. We highlight that BE is backtrack free on many instances showing that a higher consistency level is effective since GAC has #BF=0 on these instances.

Figure 4a shows the runtime distribution of the *total time* for all 2559 instances. It shows BE solves more instances than CT/HTAC with Wdeg and better than Impact and Activity, e.g. BE solves 116 more instances than CT with Wdeg for a time limit of 30 seconds/instance. Figure 4b, 4c and 4d show the differences more clearly with performance profiles [Dolan and Moré, 2002] comparing the algorithms for each search heuristic on the corresponding non-trivial instances. The y-axis is the percentage of instances and x-axis is the ratio of the CPU time of an algorithm to the virtual best algorithm. BE dominates CT, HTAC and STRbit for all heuristics tested.

Figure 5 gives scatter plots comparing *solving times*. The solving time of BE is less than that of CT and HTAC on most instances, except instances where BE has the same or more

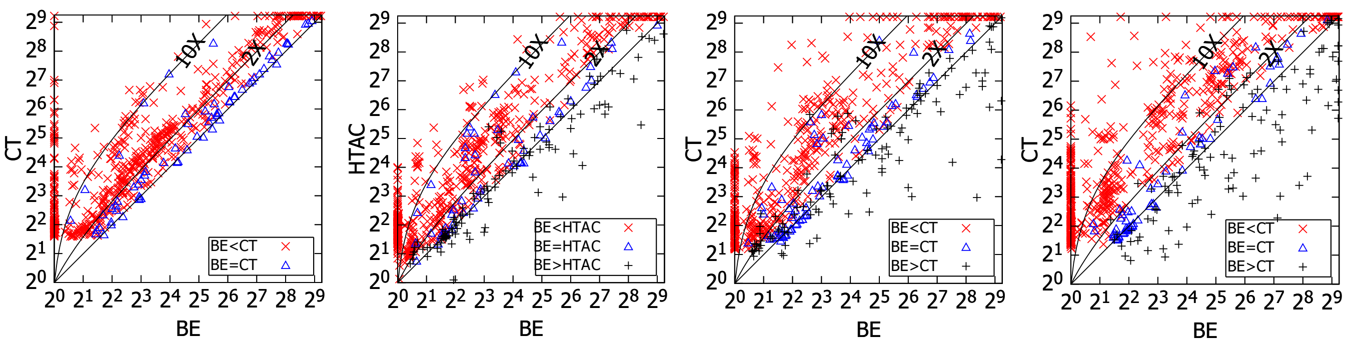
[2] The initial $cutoff = 10$ and $\rho = 1.1$. For each restart, $cutoff$ is the allowed number of failed assignments and $cutoff$ increases by $(cutoff \times \rho)$ after restart.

[3] 43 CSP series: MaxCSP-{cnf-3-40, cnf-3-80, pedigree, spot5}, Kakuro-{easy, medium, hard}, Dubois, PigeonsPlus, Renault, Aim-{50, 100, 200}, Jnh, Cril, Tsp-{20, 25, 4-20}, Various, Nonogram, Bdd-{15, 18}-21, mdd-7-25-{5, 5-p7, 5-p9}, Reg2ext, Rand-3-24-{24, 24f}, Rand-3-28-{28, 28f}, Rand-5-12-{12, 12t}, Rand-5-{2, 4, 8}X, Rand-10-20-10, Rand-10-20-60, Rand-15-23-3, Rand-5-10-10, Rand-7-40-8t, Rand-8-20-5, Rand-3-20-{20, 20f}.



(a) all instances (b) Wdeg (c) Activity (d) Impact

Figure 4: Overall *total time* comparison



(a) DDeg+O (b) Wdeg (c) Activity (d) Impact

Figure 5: *Solving time* comparison: Each dot denotes an instance, the time on the x-axis and y-axis is (1 + solving time) to enable logarithmic scales, "A=B" ("A>B" and "A<B") means the number of search nodes of Algorithm A is within 2% (greater than 1.02X and less than 1.02X) than that of B. The 10X (and 2X) line means BE is 10X (and 2X) faster than CT or HTAC.

search nodes. A few instances are slower—it turns out that the higher consistency level change the search tree, i.e. filtering impacts the search. To explore this effect, we tested the DDeg/Dom (DDeg) heuristic [Smith and Grant, 1998]. Figure 5a compares BE with CT using the DDeg+O heuristic (for DDeg+O the BE algorithm uses the structure of the original instances to calculate the value of DDeg). With DDeg+O, the number of search nodes of BE is less than or equal to that of CT on all instances tested. The solving time of BE is less than that of CT on all but 10 "BE=CT" instances. For the DDeg heuristic, the solving time of BE is 10X (2X) less than that of CT on 23% (71%) instances. Figures 5b, 5c and 5d give the results of Wdeg, Activity and Impact. For the heuristic Wdeg, Activity and Impact, the solving time of BE is 10X (2X) less than that of HTAC and CT on 27%, 31% and 35% (68%, 69% and 71%) instances, respectively. We see that the Impact heuristic is more affected by consistency levels. We remark that the backtrack free instances with BE can be seen as the points which are on the y-axis.

7 Conclusion

GAC algorithms have been the mainstay of CSP solvers for non-binary CSPs. Recently, the well known hidden variable encoding was shown through the HTAC algorithm to be competitive with state-of-the-art GAC algorithms such as CT on the original non-binary CSP. This is surprising since encoding a non-binary CSP into a binary CSP allowing the use of AC algorithm has been thought to be inferior to GAC on the original non-binary CSP. We show how to further improve the performance of binary encoding with a new binary encoding, the bipartite encoding based on partitioning constraint scopes. Ensuring AC on BE encoded instances not only gives GAC on the original non-binary CSP but may also have higher consistency than GAC. We present an algorithm to construct BE instances with heuristics to keep the representation of the binary constraints compact while encouraging the possibility of higher consistency. We also give a new AC propagator, AC-BE, which exploits the structure of BE instances working on connected components of the encoded constraint graph. Extensive experiments comparing solving CSPs with BE versus state-of-the-art GAC algorithms (CT, HTAC and STRbit) on the well known variable search heuristics (Wdeg, Activity, Impact) demonstrate the superiority of solving the CSP with BE. Furthermore, the higher consistency which is achieved also contributes to faster solving and in some cases makes the CSP backtrack free.

Acknowledgments

We acknowledge the support from grant R-726-000-006-646.

References

[Bessiere *et al.*, 2008] Christian Bessiere, Kostas Stergiou, and Toby Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 172(6-7):800–822, 2008.

[Boussemart *et al.*, 2004] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *European Conference on Artificial Intelligence*, 2004.

[Briggs and Torczon, 1993] Preston Briggs and Linda Torczon. An efficient representation for sparse sets. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 2(1-4):59–69, 1993.

[Dechter and Pearl, 1989] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38(3):353–366, 1989.

[Demeulenaere *et al.*, 2016] Jordan Demeulenaere, Renaud Hartert, Christophe Lecoutre, Guillaume Perez, Laurent Perron, Jean-Charles Régin, and Pierre Schaus. Compact-Table: efficiently filtering table constraints with reversible sparse bit-sets. In *International Conference on Principles and Practice of Constraint Programming*, 2016.

[Dolan and Moré, 2002] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[Janssen *et al.*, 1989] Philippe Janssen, Philippe Jégou, Bernard Nouguier, and Marie-Catherine Vilarem. A filtering process for general constraint-satisfaction problems: achieving pairwise-consistency using an associated binary representation. In *IEEE International Workshop on Tools for Artificial Intelligence*, 1989.

[Lecoutre and Szymanek, 2006] Christophe Lecoutre and Radoslaw Szymanek. Generalized arc consistency for positive table constraints. In *International Conference on Principles and Practice of Constraint Programming*, 2006.

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

[Lecoutre *et al.*, 2012] Christophe Lecoutre, Chavalit Likitvivatanavong, and Roland H.C. Yap. A path-optimal GAC algorithm for table constraints. In *European Conference on Artificial Intelligence*, 2012.

[Lecoutre *et al.*, 2013] Christophe Lecoutre, Anastasia Paparrizou, and Kostas Stergiou. Extending STR to a higher-order consistency. In *AAAI Conference on Artificial Intelligence*, 2013.

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

[Likitvivatanavong *et al.*, 2014] Chavalit Likitvivatanavong, Wei Xia, and Roland H.C. Yap. Higher-order consistencies through GAC on factor variables. In *International Conference on Principles and Practice of Constraint Programming*, 2014.

[Michel and Van Hentenryck, 2012] Laurent Michel and Pascal Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming*, 2012.

[Refalo, 2004] Philippe Refalo. Impact-based search strategies for constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, 2004.

[Rossi *et al.*, 1990] Francesca Rossi, Charles J Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In *European Conference on Artificial Intelligence*, 1990.

[Samaras and Stergiou, 2005] Nikolaos Samaras and Kostas Stergiou. Binary encodings of non-binary constraint satisfaction problems: Algorithms and experimental results. *Journal of Artificial Intelligence Research*, 24:641–684, 2005.

[Schneider and Choueiry, 2018] Anthony Schneider and Berthe Y Choueiry. PW-CT: extending Compact-Table to enforce pairwise consistency on table constraints. In *International Conference on Principles and Practice of Constraint Programming*, 2018.

[Smith and Grant, 1998] Barbara M Smith and Stuart A Grant. Trying harder to fail first. In *European Conference on Artificial Intelligence*, 1998.

[Stergiou and Walsh, 1999] Kostas Stergiou and Toby Walsh. Encodings of non-binary constraint satisfaction problems. In *National Conference on Artificial Intelligence*, 1999.

[Verhaeghe *et al.*, 2018] Hélene Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Compact-MDD: Efficiently filtering (s)MDD constraints with reversible sparse bit-sets. In *International Joint Conference on Artificial Intelligence*, 2018.

[Verhaeghe *et al.*, 2019] Hélène Verhaeghe, Christophe Lecoutre, and Pierre Schaus. Extending Compact-Diagram to basic smart multi-valued variable diagrams. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2019.

[Wang and Yap, 2019] Ruiwei Wang and Roland H.C. Yap. Arc consistency revisited. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2019.

[Wang *et al.*, 2016] Ruiwei Wang, Wei Xia, Roland H.C. Yap, and Zhanshan Li. Optimizing simple tabular reduction with a bitwise representation. In *International Joint Conference on Artificial Intelligence*, 2016.

[Yap *et al.*, 2020] Roland H.C. Yap, Wei Xia, and Ruiwei Wang. Generalized arc consistency algorithms for table constraints: A summary of algorithmic ideas. In *AAAI Conference on Artificial Intelligence*, 2020.