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
{ruwei,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 AC3bit [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.
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 $\{x_1, x_2, \ldots, x_i\}$ is a set of literals $\{(x_1, a_1), (x_2, a_2), \ldots, (x_i, a_i)\}$. Each constraint $c_i$ consists of a constraint scope scp($c_i$) $\subseteq \mathcal{X}$ and a relation rel($c_i$) which is defined by a set of tuples over scp($c_i$). The arity of constraint $c$ is the number of variables in the constraint scope, i.e., |scp($c_i$)|. $\mathcal{P}$ 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 $\tau$ (or a set $T$ of tuples) on a set $S$ of variables, denoted by $\tau[S]$ (or $T[S]$), is $\{(x, a) \in \tau \mid x \in S\}$ (or $\{\tau[x] \mid x \in T\}$). A tuple $\tau$ is valid iff for all literals $(x, a) \in \tau$, the value $a$ is in $\mathcal{D}(x)$ and $x$ only appears once in $\tau$. A tuple $\tau$ over variables $X \subseteq \mathcal{X}$ is consistent iff $\tau[scp(c)] \subseteq$ rel($c$) for all constraints $c \in \mathcal{C}$ such that scp($c_i$) $\subseteq X$. A solution of $\mathcal{P}$ 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($c$) on a constraint $c_j$ is a tuple $\tau_2 \in$ rel($c_j$) such that $\tau_2[scp(c_j)] = \tau_1[scp(c_j)]$. For binary CSPs, $a \in \mathcal{D}(y)$ 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 $\tau$ 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 $f$ over a non-empty set of variables $S \subseteq \mathcal{X}$ is a new variable such that $\mathcal{D}(f) = \mathcal{D}(f)$ is a set of tuples over $S$, and $\tau[S] \in \mathcal{D}(f)$ for all solutions $\tau$ of $\mathcal{P}$. We use $\mathcal{D}(f) = S$ to denote the scope of the variables covered by $f$. A factor variable $f$ is original if $|\mathcal{D}(f)| = 1$ and compound if $|\mathcal{D}(f)| > 1$.

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

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 $f \in \mathcal{X}^+ \cup \mathcal{X}^*$, scp($f$) $\neq$ scp($f$) 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($c_i^*$) $= \{f_{v_i}, f_{v_j}\}$, scp($f_{v_i}$), scp($f_{v_j}$) is either a disjoint or non-disjoint partition of scp($c_i$), and rel($c_i^*$) $= \{\tau_1, \tau_2\} | \tau_1 \in \mathcal{D}(f_{v_1}), \tau_2 \in \mathcal{D}(f_{v_2}) \equiv \tau_1 \cup \tau_2 \in$ rel($c_i$).
- Mapping constraints $\mathcal{C}^+$: for each $f \in \mathcal{X}^+ \cup \mathcal{X}^*$ and $x \in$ scp($f$), $c_i^*$ $\in$ $\mathcal{C}^*$ is a binary constraint such that scp($c_i^*$) $= \{f_{v_1}, f_{v_j}\}$, scp($f_{v_i}$) $= \{x\}$, and rel($c_i^*$) $= \{\tau_1, \tau_2\} | \tau_1 \in \mathcal{D}(f_{v_1}), \tau_2 \in \mathcal{D}(f_{v_2}) \equiv \tau_1 \cup \tau_2 \in$ scp($f_i$).

For all factor variables $f$, the information of the tuples in the domain $\mathcal{D}(f)$ is recorded in the mapping constraints $\{c_i^* | x \in$ scp($f_i$)$\}$, which ensures that the tuple $\tau_1 \cup \ldots \cup \tau_m$ is an assignment of $\mathcal{P}$ if the tuple $(\{f_{v_1}, \tau_1\}, \ldots, \{f_{v_m}, \tau_m\})$ 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 $(\{f_{v_j}, \tau_1\}, \ldots, \{f_{v_k}, \tau_m\})$ 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 $(\{f_{v_1}, \tau_1\}, \ldots, \{f_{v_m}, \tau_m\})$ is a solution of BE($\mathcal{P}$).

Figure 1: Bipartite encoding example

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\hline
$x_1$ & $x_2$ & $x_3$ & $x_4$ & $x_5$ & $x_6$ \\
\hline
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
\hline
\end{tabular}
\end{figure}
Example 1. Let CSP \(\mathcal{P}=(\mathcal{X}, \mathcal{C})\) where \(\mathcal{C} = \{c_1, c_2\}\), \(\mathcal{X} = \{x_1, \ldots, x_6\}\) and domains are \([0,1]\). Constraint scopes are: \(scp(c_1) = \{x_1, \ldots, x_4\}\) and \(scp(c_2) = \{x_1, x_2, x_5, x_6\}\) with relations given in Figure 1a, 1b. A bipartite encoding \(\{(f_{v_1}, f_{v_2}, f_{v_3}, x^f_1, \ldots, x^f_6), \{c^*_1, c^*_2, c^*_3, \ldots, c^*_8\}\}\) of \(\mathcal{P}\) is shown in Figures 1c, 1j. Constraints \(c_1\) and \(c_2\) are represented as \(c_1^*\) and \(c_2^*\), where \(scp(c_1^*) = \{f_{v_1}, f_{v_2}\}\) and \(scp(c_2^*) = \{f_{v_1}, f_{v_1}, f_{v_2}\}\). \(f_{v_1}, f_{v_2}, f_{v_3}\) are compound factor variables on \(S_1 = \{x_1, x_2\}\), \(S_2 = \{x_3, x_4\}\) and \(S_3 = \{x_5, x_6\}\), \(x^f\) is an original factor variable over \(\mathcal{X}\) 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 \(x^f\), value \(a\) denotes \(\{(x, a)\}\). Each solution of \(BE(\mathcal{P})\) corresponds to a solution of \(\mathcal{P}\), e.g. \(\{(f_{v_1}, 0), (f_{v_2}, 1), (f_{v_3}, 3), (x^f_1, 0), (x^f_2, 0), (x^f_3, 0), (x^f_4, 1), (x^f_5, 1), (x^f_6, 1)\}\) 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 \(\mathcal{C}' = \{c'|c \in \mathcal{C}\}\), where \(\mathcal{D}(x') = \{a | (x, a) \in \mathcal{D}(x)\}\). Let \(\mathcal{P}'\) be the image 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 \(scp(c^*_i) = \{f_{v_j}, f_{v_k}\}\), \(scp(f_{v_j}) > 1\), \(x \in scp(f_{v_j})\), and \(a \in \mathcal{D}(x)\). \(BE(\mathcal{P})\) is AC, hence, \(\{f_{v_j}, \{(x, a)\}\}\) has a valid support on \(\tau_1 \in \mathcal{D}(f_{v_j})\) and \(\tau_1\) has a valid support \(\tau_2 \in \mathcal{D}(f_{v_k})\). 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, \(\{x^f_1, 1\}\) is not AC on the BE instance given in Figure 1, but the original CSP is GAC.

Algorithm 3: maxEdge(cur)
\[
\text{Let } E \text{ be all maximum edges in } \text{cur}; \\
S \leftarrow \{scp(c_1) \cap scp(c_2)| (c_1, c_2) \in E\}; \\
\text{for } S_k \text{ in } S \text{ do}; \\
\text{size}[S_k] \leftarrow \text{domain size of a factor variable on } S_k; \\
\text{N}[S_k] \leftarrow \text{\{e \in E| e \subseteq cur, scp(e) = S_k\};} \\
E_r \leftarrow \emptyset; \\
\text{while } S \neq \emptyset \text{ do}; \\
\text{Select a variable } x \text{ from } scp(c_1); \\
scp[c_1] \leftarrow \text{\{scp(c_1) \cap \{x, \{x\}\\};} \\
\text{return scp;}
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 an undirected graph such that constraints $\mathcal{C}$ are nodes and edges are $\{\{c_1, c_j\} \subseteq C | \text{scp}(c_i) \cap \text{scp}(c_j) \neq \emptyset\}$. We use $\text{scp}(e) = \text{scp}(c_i) \cap \text{scp}(c_j)$ to denote the scope of variables covered by an edge $e = \{c_i, c_j\}$. The size of an edge $e$ is the cardinality of $\text{scp}(e)$. 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 $f_e$ in $\text{scp}(c_i) \cap \text{scp}(c_j)$ such that $\text{scp}(f_e) = \text{scp}(e)$, e.g. the bipartite encoding given in Figure 1 covers the edges $\{c_1, c_2\}$, since $\text{scp}(c_1) \cap \text{scp}(c_2)$ includes a factor variable $f_{v_1}$ 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 $P'$ 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 $c'_i, c'_j \in C'$ and $\tau \in \text{rel}(c'_i)$ such that $S = \text{scp}(c'_i) \cap \text{scp}(c'_j)$ includes at least 2 variables, $\tau$ has a valid support on $c'_j$, since $\tau|S$ is in $D(f_{v_k})$ and has valid supports on $c'_j$, where $f_{v_k} \in \text{scp}(c'_i) \cap \text{scp}(c'_j)$ is a factor variable on $S$. Recall, $P'$ is AC (Proposition 1), so $P'$ is FPWC.

Algorithm 2 generates partitions of constraint scopes. The data structure $\text{scp}(c_i)$ records the partition of $\text{scp}(c_i)$. Algorithms 2 and 3 use a global set $\text{cur}$ 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 $\text{cur}$ such that: (i) for each $c_i \in \text{cur}$, there is at most one partition of $\text{scp}(c_i)$ based on $E$, i.e. $|\{\text{scp}(e) | e \in E, c_i \in e\}| \leq 1$; and (ii) for each edge $e = \{c_i, c_j\}$ in $E$, the constraints $c_i$ and $c_j$ are size splittable by $\text{scp}(e)$. 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 $\text{cur}$, we select the last variable $x$ in $\text{scp}(c_i)$ and partition $\text{scp}(c_i)$ (between Lines 4 and 5).

A constraint $c_k$ is size splittable by a variable subset $S_i \subseteq \text{scp}(c_k)$ or $S_i' = \text{scp}(c_k) \setminus S_i$ if the size of the bit representation (see data structures used in HTAC [Wang and Yap, 2019]) of $c_k$ is greater than equal to that of the corresponding constraints in the BE encoding, namely, the sum of the sizes of the binary constraints $c_k^i$ and $\{|c_k^i| \in C^+ | \text{scp}(f_{v_1}) \in S_i\}$

1. We remark that experiments show HTAC on the HVE encoding is competitive with CT on the original CSP, 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.
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 \((X^+ \cup X^* \cup C^+ \cup C^*)\) extending the framework of HTAC. We partition binary constraints in the BE instance into a set of connected components in an undirected graph \((C_1, E)\), and employ special propagators for the components, where \(C_1 = C^+ \cup C^*\) and \(E = \{\{c_1, c_j\} \subseteq C_1 | scp(c) \cap scp(c_j) \cap X^* \neq \emptyset\}\), and every component is treated as a set of constraints. Meanwhile, we also only search on the original factor variables in \(X^+\). 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 \(\{c_1^1, c_2^1, c_3^1, c_1^2, c_2^2, c_3^2\}\). The constraint scopes of all constraints in the first component include the factor variable \(f_{v_1}\). Figure 3b shows the second component \(\{c_1^*2, c_2^*, c_2^*, c_3^*\}\). The compound factor variable \(f_{v_2}\) is included in the constraint scopes of all connected components in the second component. Note that the constraints in different components only share some original factor variables.

Algorithm 4: AC-BE(C)

\[
V \leftarrow \{scp(c) | c \in C\};
1 \quad T \leftarrow \emptyset, U \leftarrow C;
\]

\[
\text{while } \exists x \in V \text{ s.t. } |\{c \in U | x \in scp(c)\}| = 1 \text{ do}
2 \quad T \leftarrow T \cup \{c \in U | x \in scp(c)\};
3 \quad U \leftarrow U \setminus \{c \in U | x \in scp(c)\};
1 \quad \text{if } \sim propagationUp(T) \text{ then return false ;}
1 \quad \text{if } \sim AC(U) \text{ then return false ;}
1 \quad \text{if } \sim propagationDown(T) \text{ then return false ;}
1 \quad \text{for } x \in V \cap X^+ \text{ s.t. } D(x) \text{ is changed do}
1 \quad \text{Add } x \text{ to the propagation queue;}
1 \quad \text{return true;}
\]

Every connected component \(C\) in the graph \((C_1, E)\) 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)\}. 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 \(x \in X^+\) is changed, then \(x\) is added to a propagation queue shared by all connected components of the graph \((C_1, E)\), thus propagating to other constraints in \(C_1\).

Comparing to the normal AC propagators like 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 \((C_1, E)\) until all connected components of \((C_1, E)\) 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 domain 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 $D(x)$ based on another variable domain $D(y)$: (i) seekSupport scans all values in the domain $D(x)$, removing values which do not have any valid support on $y$; (ii) reset updates $D(x)$ by using the union of all supports on $x$ of the values in $D(y)$; and (iii) delete removes all supports on $x$ of the values which are removed from $D(y)$. 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 $|D(x)| \leq |D(y)|$. Otherwise, reset or delete operations are used to update the domain $D(x)$. We use the delete operation as an optimization when the values in $D(x)$ have at most 1 support in $D(y)$. 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 3.2GHz 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. The value heuristic used is lexical value order. We measure CPU timings as: (i) solving 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, which only employ table constraints, from the XCSP3 website (http://xcsp.org).

---

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.

#### Table 1: Relative comparison with Total Times

<table>
<thead>
<tr>
<th></th>
<th>STRbit</th>
<th>CT</th>
<th>HTAC</th>
<th>BE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wdeg (682)</strong></td>
<td>AvgR</td>
<td>3.50</td>
<td>2.68</td>
<td>1.99</td>
</tr>
<tr>
<td></td>
<td>MaxR</td>
<td>91.46</td>
<td>80.98</td>
<td>19.2</td>
</tr>
<tr>
<td></td>
<td>#F</td>
<td>3</td>
<td>37</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>#TO</td>
<td>45</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>#BF</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Activity (655)</strong></td>
<td>AvgR</td>
<td>3.58</td>
<td>2.76</td>
<td>2.80</td>
</tr>
<tr>
<td></td>
<td>MaxR</td>
<td>90.54</td>
<td>80.01</td>
<td>77.83</td>
</tr>
<tr>
<td></td>
<td>#F</td>
<td>4</td>
<td>87</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>#TO</td>
<td>51</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td><strong>Impact (691)</strong></td>
<td>AvgR</td>
<td>4.73</td>
<td>3.52</td>
<td>3.64</td>
</tr>
<tr>
<td></td>
<td>MaxR</td>
<td>149.78</td>
<td>129.44</td>
<td>138.61</td>
</tr>
<tr>
<td></td>
<td>#F</td>
<td>0</td>
<td>89</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>#TO</td>
<td>59</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>#BF</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Initial Time (s)</strong></td>
<td>1.39</td>
<td>1.37</td>
<td>1.38</td>
<td>1.65</td>
</tr>
</tbody>
</table>

---

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].
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


