Un Algorithme Optimal de Filtrage pour Contraintes Table

JB Mairy Pascal Van Hentenryck Yves Deville

JFPC 2013
Table Constraints

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>

Table constraints can represent anything:

**Non-intentional constraints**
(compatibility, configuration, preference, etc.)

**Intentional constraints**
(regular, etc.)

Importance and difficulty
to design efficient propagators:

No exploitable semantics
Contributions

Existing table constraint propagators

- Index
  - GAC_Allowed
  - GAC_valid
  - GAC_allowed+valid
  - etc.

- Compression
  - Tries
  - MDD

- Dynamic Table
  - STR2

- Optimal Value Based
  - STR3

STR3 is the only above with optimal time complexity

We propose new Propagators

- New efficient propagators
- An optimal propagator
- Better than state of the art on most benchmarks
AC5: a Generic GAC Framework

AC5 is a *generic* framework for GAC propagation algorithms

The concrete propagators must implement 2 methods:

- **post**: initialization of the structures and first pruning
- **valRemove**(in \( y \): Variable; in \( b \): Value;):
  - reflects the deletion of value \( b \) from \( \text{dom}(y) \)
  - removes non-GAC values

Value-Based approach

Our propagators provide implementations of those methods
AC5 Uses a Propagation Queue

Initialization

Propagation

Other Constraints
remove values

(post)

\(Q_c\)

valRemove (var, val)

\((x, d)\)
\((y, b)\)
\((x, a)\)
The First Supports are Maintained

AC5 for **table constraints**:  
the constraint table is static  
⇒ first supports are maintained

First support structure  
called **FS**

**FS**[\(y, b\)] = index of first valid support for \((y, b)\)
Indexing is Used for the Traversals

<table>
<thead>
<tr>
<th>Constraint</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
</tr>
</tbody>
</table>
Indexing is Used for the Traversals

Constraint

next

\[
\begin{array}{ccc}
  x & y & z \\
  1 & a & b \quad a \\
  2 & b & c \quad b \\
  3 & a & a \quad a \\
  4 & c & b \quad b \\
  5 & b & b \quad a \\
  6 & c & a \quad c \\
\end{array}
\]

\[
\begin{array}{ccc}
  x & y & z \\
  3 & 1 \\
  2 & 2 \\
  3 & 3 \\
  4 & 4 \\
  5 & 5 \\
  6 & 6 \\
\end{array}
\]
Indexing is Used for the Traversals

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>

Next:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Indexing is Used for the Traversals

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>a</td>
<td>c</td>
</tr>
</tbody>
</table>

next:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Indexing is Used for the Traversals

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>a</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>T</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Indexing is Used for the Traversals

- Table has to be traversed during propagation steps
- Indexing structure linking together the tuples with the same value for variables
- $\text{next}[x, i] =$ smallest tuple index $j > i$ with the same value for $x$
- Allows to only consider the tuples with the desired value for a variable
AC5TC-Recomp Tests the Validity of Candidate Supports

\textit{valRemoveTC-Recomp}(x, a):

Propagates the deletion of \( a \) from \( \text{dom}(x) \)
in the constraint

Using \textbf{Recomputation} of validity:

- no validity info stored
- tuples tested for validity
AC5TC-Recomp Tests the Validity of Candidate Supports

valRemoveTC-Recomp\((x, a)\):

foreach tuple \(\sigma_i\) where \(x = a\):
    foreach \((y, b)\) in \(\sigma_i\):
        if \(FS[y, b] == i\):
            find new first support

\(FS[x, a]\)
AC5TC-Recomp Tests the Validity of Candidate Supports

valuation of \textit{removal TC-Recomp}(x, a): 

\begin{align*}
\text{valRemoveTC-Recomp}(x, a): \\
\text{foreach tuple } \sigma_i \text{ where } x = a: \\
\quad \text{foreach } (y, b) \text{ in } \sigma_i: \\
\quad \quad \text{if } FS[y, b] = i: \\
\quad \quad \quad \text{find new first support}
\end{align*}

While testing next chain:
valid support found
AC5TC-Recomp Tests the Validity of Candidate Supports

`valRemoveTC-Recomp(x, a)`:

foreach tuple $\sigma_i$ where $x = a$:
  foreach $(y, b)$ in $\sigma_i$:
    if $FS[y, b] == i$:
      find new first support

After testing next chain:
no valid support found
AC5TC-Recomp is not optimal

AC5TC-Recomp is efficient on the benchmarks but not theoretically optimal

The temporal complexity of AC5TC-Recomp is

$\mathcal{O}(r^2 \cdot t + r \cdot d)$

- $r$: arity
- $t$: # tuples
- $d$: max dom size

The spacial complexity of AC5TC-Recomp:

- next: $\Theta(r \cdot t)$, static
- FS: $\Theta(r \cdot d)$, dynamic
AC5TC-Recomp can be Improved

AC5TC-Recomp is efficient on the benchmarks but not theoretically optimal.

The temporal complexity of AC5TC-Recomp is

\[ O(r^2 \cdot t + r \cdot d) \]

\( r \): arity \quad \( t \): \# tuples \quad \( d \): max dom size

Can be improved in two ways because:

- validity of the visited tuples must be checked: \( O(r) \)
- tuples can be visited multiple times (in different chains)
An Optimal Propagator
AC5TC-Tr Changes the next Structure

From validity to $Q$-validity:

- A tuple $\sigma$ is valid iff $\forall x : \sigma[x] \in \text{dom}(x)$
- A tuple $\sigma$ is $Q$-valid iff $\forall x : \sigma[x] \in \text{dom}(x) \cup \{ a | (x, a) \in Q_c \}$

$Q$-validity information included in $\text{next}$:

$\text{next}[x, i] =$

smallest tuple index $j > i$ with the same value for $x$
which is $Q$-valid

Consequences:

- No need to test the validity of the tuples
- Any tuple is considered at most once
- Dynamic structure
AC5TC-Tr Maintains the Next Structure

\textit{valRemoveTC-Tr}(x, a):

\begin{itemize}
  \item foreach tuple $\sigma_i$ where $x = a$:
    \begin{itemize}
      \item foreach $(y, b)$ in $\sigma_i$:
        \begin{itemize}
          \item update \textit{next} chain or $FS[y, b]$
        \end{itemize}
    \end{itemize}
\end{itemize}
valRemoveTC-Tr(x, a):

foreach tuple $\sigma_i$ where $x = a$:

foreach $(y, b)$ in $\sigma_i$:

update next chain or $FS[y, b]$

if $FS[y, b] \neq i$:

update next

$\sigma_i$ never reconsidered for $y$
**AC5TC-Tr Maintains the Next Structure**

\[
\text{\texttt{valRemoveTC-Tr}}(x, a):
\]

\[
\text{foreach tuple } \sigma_i \text{ where } x = a:
\]

\[
\text{foreach } (y, b) \text{ in } \sigma_i:
\]

\[
\text{update } \text{next} \text{ chain or } FS[y, b]
\]

\[
\text{if } FS[y, b] == i:
\]

\[
\text{update } FS
\]
AC5TC-Tr Maintains the Next Structure

\textit{valRemoveTC-Tr}(x, a):

\begin{enumerate}
\item\texttt{foreach} tuple $\sigma_i$ where $x = a$:
\item\texttt{foreach} $(y, b)$ in $\sigma_i$:
\item update \texttt{next} \texttt{chain} or \texttt{FS}[y, b]
\end{enumerate}

If \texttt{FS}[y, b] == i:
update \texttt{FS}

next element in chain is Q-valid
\(b\) removed if next is \(\top\)
AC5TC-Tr is optimal

AC5TC-Tr has an optimal theoretical time complexity per table constraint:

$$O(r \cdot t + r \cdot d)$$

- $r$: arity
- $t$: # tuples
- $d$: max dom size

Optimal because in the worst case, one has:
- to traverse the table
- to suppress all the values
AC5TC-Tr is optimal

AC5TC-Tr has an optimal theoretical time complexity per table constraint:

\[ \mathcal{O}(r \cdot t + r \cdot d) \]

- \( r \): arity
- \( t \): \# tuples
- \( d \): max dom size

The spacial complexity of AC5TC-Tr:

- next: \( \Theta(r \cdot t) \), dynamic
- FS: \( \Theta(r \cdot d) \), dynamic
Experimental results
Random instances

GAC3Allow: GAC3-allowed
AC5-Rec: AC5TC-Recomp
AC5-Tr: AC5TC-Tr
MDD: Multivalued Decision Diagrams
STR2+: Simple Tabular Reduction 2
STR3: Simple Tabular Reduction 3
Langford instances, $k = 4$
Travelling Salesman Problem, 25 cities
Conclusion

- New propagators for table constraints
- A question of tradeoff:

\[\text{AC5TC-Bool} \quad \text{AC5TC-Map} \quad \text{AC5TC-Recomp} \quad \text{AC5TC-Tr}\]

- recomputing \hspace{1cm} maintaining

  - not optimal, redundant
  - optimal theoretical time complexity
  - minimal backtrack
  - backtracks large structure

For random instances:  
do not use our propagators

For non random ones:  
do use them
Un Algorithme Optimal de Filtrage pour Contraintes Table

JB Mairy  Pascal Van Hentenryck  Yves Deville

JFPC 2013

UCL
Université catholique de Louvain