Implementation and Evaluation of a Compact-Table Propagator in Gecode

Linnea Ingmar
<linnea.ingmar.3244@student.uu.se>

The ASTRA Group
on Combinatorial Optimisation

Uppsala University

17th May 2017

Supervisor: Mats Carlsson (SICS)
Reviewer: Pierre Flener

UPPSALA UNIVERSITET

Background

The Compact-Table Algorithm

Evaluation

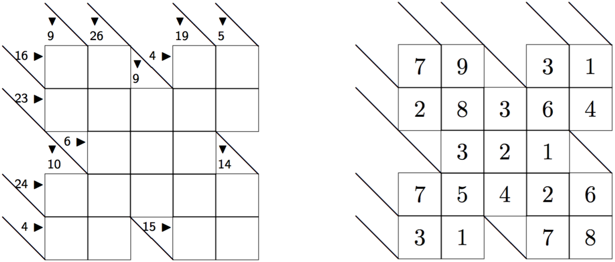
Conclusions

Outline

Outline

 Linnea Ingmar | *linnea.ingmar.3244@student.uu.se*

Kakuro puzzle

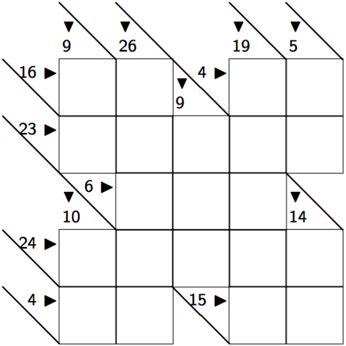


Assign the cells digits from 1 to 9 such that for each row and column:

- digits are distinct, and
- the sum of the digits is equal to the *clue*

Kakuro puzzle as a constraint problem (1)

Variables	One per cell.
Domains	$\{1...9\}$ for all variables.
Constraints	For each row and column: *distinct* digits, and the *sum* of the digits is equal to the clue.

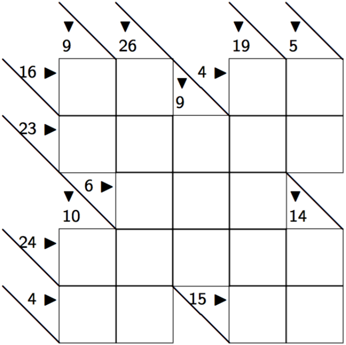


Kakuro puzzle as a constraint problem (2)

Variables One per cell.

Domains $\{1 \ldots 9\}$ for all variables.

Constraints For each row and column: state the *possible combinations of values* that the variables can take.



For an entry of size 2 and clue 4: $\langle 1, 3 \rangle$ and $\langle 3, 1 \rangle$ are the only combinations.

Constraint problems (definition)

Definition (Constraint problem)

A **constraint satisfaction problem (CSP)** is a triple

$$\langle V, D, C \rangle$$

where:

- $V = v_1, \ldots, v_n$ is a finite sequence of variables,
- $D = D_1, \ldots, D_n$ is a finite sequence of domains, that are possible values for the respective variable,
- $C = \{c_1, \ldots, c_m\}$ is a finite set of constraints, each on a subset of V. Express relations among the variables that have to be true.

Constraint Propagation

- Constraint store
- Propagator
- Constraint propagation

Constraint Stores

Definition (Constraint store)

A **constraint store** s is a function mapping variables to domains:

$$s : \mathit{variables} \mapsto \mathit{domains}$$

s_0

$$\{x \mapsto \{1,2,3\},\ y \mapsto \{1,2,3\}\}$$

Propagators

Definition (Propagator)

A **propagator** p is a function mapping stores to stores:

$$p : store \mapsto store$$

- Implement constraints

$$\underset{s_0}{\{x \mapsto \{1,2,3\},\ y \mapsto \{1,2,3\}\}} \xrightarrow{p_{x<y}} \underset{s_1}{\{x \mapsto \{1,2,\cancel{3}\},\ y \mapsto \{1,2,3\}\}}$$

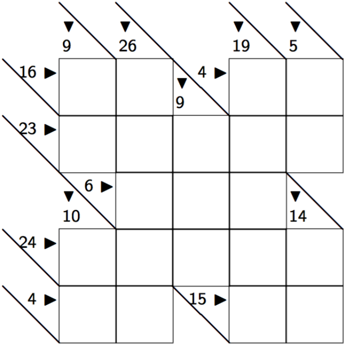
Constraint Propagation

$x_0 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$x_1 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$\ldots$
$x_4 \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$\ldots$

x_0	x_1
7	9
9	7

x_0	x_4
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1

$\ldots$



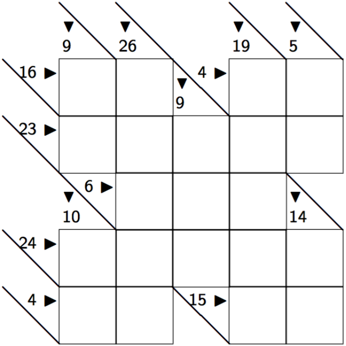
Constraint Propagation

$$x_0 \in \{\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},7,\cancel{8},9\}$$
$$x_1 \in \{\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},7,\cancel{8},9\}$$
$$\ldots$$
$$x_4 \in \{1,2,3,4,5,6,7,8,9\}$$
$$\ldots$$

x_0	x_1
7	9
9	7

x_0	x_4
~~1~~	~~8~~
~~2~~	~~7~~
~~3~~	~~6~~
~~4~~	~~5~~
~~5~~	~~4~~
~~6~~	~~3~~
7	2
~~8~~	~~1~~

...



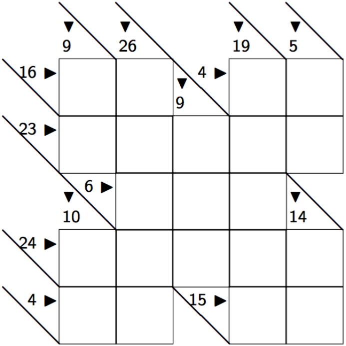
Constraint Propagation

$$x_0 \in \{\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},\mathbf{7},\cancel{8},\cancel{9}\}$$
$$x_1 \in \{\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},7,\cancel{8},9\}$$
$$\ldots$$
$$x_4 \in \{\cancel{1},\mathbf{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},\cancel{7},\cancel{8},\cancel{9}\}$$
$$\ldots$$

x_0	x_1
7	9
9	7

x_0	x_4
~~1~~	~~8~~
~~2~~	~~7~~
~~3~~	~~6~~
~~4~~	~~5~~
~~5~~	~~4~~
~~6~~	~~3~~
7	2
~~8~~	~~1~~

...



Constraint Propagation

$$x_0 \in \{\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},\mathbf{7},\cancel{8},\cancel{9}\}$$
$$x_1 \in \{\cancel{1},\cancel{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},\cancel{7},\cancel{8},\mathbf{9}\}$$
$$\ldots$$
$$x_4 \in \{\cancel{1},\mathbf{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},\cancel{7},\cancel{8},\cancel{9}\}$$
$$\ldots$$

x_0	x_1
7	9
~~9~~	~~7~~

x_0	x_4
~~1~~	~~8~~
~~2~~	~~7~~
~~3~~	~~6~~
~~4~~	~~5~~
~~5~~	~~4~~
~~6~~	~~3~~
7	2
~~8~~	~~1~~

...

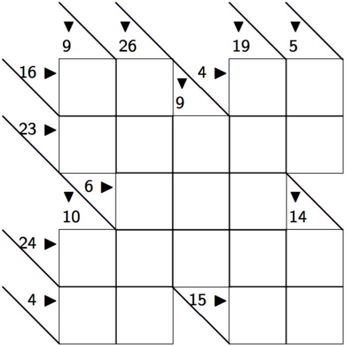


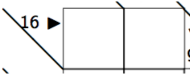
TABLE constraints

Definition (TABLE constraint)

A TABLE constraint lists the possible combinations of values that the variables can take as a sequence of n-tuples.

$$\text{TABLE}(\{x_0, x_1\}, [\langle 7, 9\rangle, \langle 9, 7\rangle])$$

x_0	x_1
7	9
9	7



Gecode

Gecode (Generic Constraint Development Environment) is...

- ...a constraint solver (a software that solves constraint problems).
- ...written in C++, modular, extensible, and has state-of-the-art performance.
- ...supports the programming of new propagators.

Two existing propagators for the TABLE constraint

Compact Table

Compact-Table

- A new propagation algorithm for the TABLE constraint.
- Published in a 2016 paper
- No attempt to implement it in Gecode (until now).

Outline

The Compact-Table Algorithm

- **Initialisation**
- Variable modifications
- Filtering
- Putting it all together

The Compact-Table Algorithm
Initialisation

$$\mathrm{dom}(x_0) = \mathrm{dom}(x_1) = \mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

x_0	1	2	1	2	6	7	4	1	7	8	2	0	2	5	4
x_1	5	1	3	4	5	7	2	1	8	9	2	0	3	8	3
x_2	8	4	2	2	9	8	1	1	9	6	3	0	1	5	1

The Compact-Table Algorithm
Initialisation

$$\mathrm{dom}(x_0) = \mathrm{dom}(x_1) = \mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

x_0	1	**2**	**1**	**2**	6	7	**4**	**1**	7	8	**2**	0	**2**	5	**4**
x_1	5	**1**	**3**	**4**	5	7	**2**	**1**	8	9	**2**	0	**3**	8	**3**
x_2	8	**4**	**2**	**2**	9	8	**1**	**1**	9	6	**3**	0	**1**	5	**1**

The Compact-Table Algorithm
Initialisation

Background

The Compact-Table Algorithm

Evaluation

Conclusions

$$\text{dom}(x_0) = \text{dom}(x_1) = \text{dom}(x_2) = \{1, 2, 3, 4\}$$

x_0	**2**	**1**	**2**	**4**	**1**	**2**	**2**	**4**
x_1	**1**	**3**	**4**	**2**	**1**	**2**	**3**	**3**
x_2	**4**	**2**	**2**	**1**	**1**	**3**	**1**	**1**

The Compact-Table Algorithm
Initialisation

$$\mathrm{dom}(x_0) = \mathrm{dom}(x_1) = \mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

$$\underbrace{\boxed{\mathbf{1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1}}} \} \texttt{validTuples}$$

x_0	**2**	**1**	**2**	**4**	**1**	**2**	**2**	**4**
x_1	**1**	**3**	**4**	**2**	**1**	**2**	**3**	**3**
x_2	**4**	**2**	**2**	**1**	**1**	**3**	**1**	**1**

The Compact-Table Algorithm
Initialisation

$$\mathrm{dom}(x_0) = \mathrm{dom}(x_1) = \mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

validTuples:

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

supports:

$\langle x_0, 1\rangle$	0	**1**	0	0	**1**	0	0	0
$\langle x_0, 2\rangle$	**1**	0	**1**	0	0	**1**	**1**	0
$\langle x_0, 3\rangle$	0	0	0	0	0	0	0	0
$\dots$								
$\langle x_2, 4\rangle$	**1**	0	0	0	0	0	0	0

x_0	**2**	**1**	**2**	**4**	**1**	**2**	**2**	**4**
x_1	**1**	**3**	**4**	**2**	**1**	**2**	**3**	**3**
x_2	**4**	**2**	**2**	**1**	**1**	**3**	**1**	**1**

The Compact-Table Algorithm
Initialisation

$$\mathrm{dom}(x_0) = \{1, 2, \cancel{3}, 4\}$$
$$\mathrm{dom}(x_1) = \mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

validTuples:

validTuples	1	1	1	1	1	1	1	1

supports:

$\langle x_0, 1\rangle$	0	1	0	0	1	0	0	0
$\langle x_0, 2\rangle$	1	0	1	0	0	1	1	0
$\langle x_0, 3\rangle$	0	0	0	0	0	0	0	0
...								
$\langle x_2, 4\rangle$	1	0	0	0	0	0	0	0

x_0	2	1	2	4	1	2	2	4
x_1	1	3	4	2	1	2	3	3
x_2	4	2	2	1	1	3	1	1

The Compact-Table Algorithm

- Initialisation
- **Variable modifications**
- Filtering
- Putting it all together

The Compact-Table Algorithm
When a variable is modified

$$\mathrm{dom}(x_0) = \{1, 2, 4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1}, \cancel{2}, 3, 4\}$$
$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

The Compact-Table Algorithm
When a variable is modified

`words`	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
`mask`	0	0	0	0	0	0	0	0

$\Big\}$ `validTuples`

$$\mathrm{dom}(x_0) = \{1, 2, 4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1}, \cancel{2}, 3, 4\}$$
$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

The Compact-Table Algorithm

When a variable is modified

words	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	validTuples
mask	0	0	0	0	0	0	0	0	

supports[x_1,3]	0	**1**	0	0	0	0	**1**	**1**
supports[x_1,4]	0	0	**1**	0	0	0	0	0

$$\mathrm{dom}(x_0) = \{1,2,4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1},\cancel{2},3,4\}$$
$$\mathrm{dom}(x_2) = \{1,2,3,4\}$$

 Linnea Ingmar | *linnea.ingmar.3244@student.uu.se*

The Compact-Table Algorithm
When a variable is modified

$$\begin{array}{r|cccccccc|l} \texttt{words} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \multirow{2}{*}{$\Big\}\ \texttt{validTuples}$} \\ \texttt{mask} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \end{array}$$

$$\begin{array}{r|cccccccc|} \texttt{supports}[x_1,3] & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\ \texttt{supports}[x_1,4] & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$\texttt{mask} = \texttt{supports}[x_1,3] \mid \texttt{supports}[x_1,4]$$

$$\mathrm{dom}(x_0) = \{1,2,4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1},\cancel{2},3,4\}$$
$$\mathrm{dom}(x_2) = \{1,2,3,4\}$$

UPPSALA UNIVERSITET

Background

The Compact-Table Algorithm

Evaluation

Conclusions

The Compact-Table Algorithm
When a variable is modified

`words`	1	1	1	1	1	1	1	1
`mask`	0	1	1	0	0	0	1	1

(`words` and `mask`: `validTuples`)

`supports`$[x_1, 3]$	0	1	0	0	0	0	1	1
`supports`$[x_1, 4]$	0	0	1	0	0	0	0	0

$$\mathrm{dom}(x_0) = \{1, 2, 4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1}, \cancel{2}, 3, 4\}$$
$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

The Compact-Table Algorithm

When a variable is modified

`words`	1	1	1	1	1	1	1	1
`mask`	0	1	1	0	0	0	1	1

(`words` and `mask`: `validTuples`)

$\texttt{supports}[x_1, 3]$	0	1	0	0	0	0	1	1
$\texttt{supports}[x_1, 4]$	0	0	1	0	0	0	0	0

`words = words & mask`

$$\mathrm{dom}(x_0) = \{1, 2, 4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1}, \cancel{2}, 3, 4\}$$
$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

The Compact-Table Algorithm

When a variable is modified

`words`	0	1	1	0	0	0	1	1
`mask`	0	1	1	0	0	0	1	1

`validTuples`

`supports`$[x_1, 3]$	0	1	0	0	0	0	1	1
`supports`$[x_1, 4]$	0	0	1	0	0	0	0	0

$$\mathrm{dom}(x_0) = \{1, 2, 4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1}, \cancel{2}, 3, 4\}$$
$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

The Compact-Table Algorithm

When a variable is modified

words	0	1	1	0	0	0	1	1	validTuples
mask	0	1	1	0	0	0	1	1	

supports[x_1, 3]	0	1	0	0	0	0	1	1
supports[x_1, 4]	0	0	1	0	0	0	0	0

x_0	2	1	2	4	1	2	2	4
x_1	1	3	4	2	1	2	3	3
x_2	4	2	2	1	1	3	1	1

$$\mathrm{dom}(x_0) = \{1, 2, 4\}$$
$$\mathrm{dom}(x_1) = \{\cancel{1}, \cancel{2}, 3, 4\}$$
$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

The Compact-Table Algorithm

When a variable is modified

PROCEDURE UPDATETABLE(s: store, x: variable)

```
validTuples.clearMask()
foreach a ∈ s(x) do
    validTuples.addToMask(supports[x, a])
validTuples.intersectWithMask()
```

The Compact-Table Algorithm

- Background
- **The Compact-Table Algorithm**
- Evaluation
- Conclusions

- Initialisation
- Variable modifications
- **Filtering**
- Putting it all together

Compact-Table Algorithm

Filtering

- Intersect every support entry with `validTuples`
- Remove value if intersection is empty

`validTuples`:

`words`	0	**1**	**1**	0	0	0	**1**	**1**

&

$\langle x_2, 1\rangle$	0	0	0	**1**	**1**	0	**1**	**1**
$\langle x_2, 2\rangle$	0	**1**	**1**	0	0	0	0	0
$\langle x_2, 3\rangle$	0	0	0	0	0	**1**	0	0
$\langle x_2, 4\rangle$	**1**	0	0	0	0	0	0	0

$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

Compact-Table Algorithm
Filtering

- Intersect every support entry with `validTuples`
- Remove value if intersection is empty

`validTuples`:

`words`	0	**1**	**1**	0	0	0	**1**	**1**

&

$\langle x_2, 1\rangle$	0	0	0	**1**	**1**	0	**1**	**1**
$\langle x_2, 2\rangle$	0	**1**	**1**	0	0	0	0	0
$\langle x_2, 3\rangle$	0	0	0	0	0	**1**	0	0
$\langle x_2, 4\rangle$	**1**	0	0	0	0	0	0	0

$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

Compact-Table Algorithm
Filtering

- Intersect every support entry with `validTuples`
- Remove value if intersection is empty

`validTuples`:

`words`	0	1	1	0	0	0	1	1

&

$\langle x_2, 1\rangle$	0	0	0	1	1	0	1	1
$\langle x_2, 2\rangle$	0	1	1	0	0	0	0	0
$\langle x_2, 3\rangle$	0	0	0	0	0	1	0	0
$\langle x_2, 4\rangle$	1	0	0	0	0	0	0	0

$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

Compact-Table Algorithm
Filtering

- Intersect every support entry with `validTuples`
- Remove value if intersection is empty

`validTuples`:

`words`	0	1	1	0	0	0	1	1

&

$\langle x_2, 1\rangle$	0	0	0	1	1	0	1	1
$\langle x_2, 2\rangle$	0	1	1	0	0	0	0	0
$\langle x_2, 3\rangle$	0	0	0	0	0	1	0	0
$\langle x_2, 4\rangle$	1	0	0	0	0	0	0	0

$$\mathrm{dom}(x_2) = \{1, 2, 3, 4\}$$

Compact-Table Algorithm
Filtering

- Intersect every support entry with `validTuples`
- Remove value if intersection is empty

`validTuples`:

`words`	0	1	1	0	0	0	1	1

&

$\langle x_2, 1\rangle$	0	0	0	1	1	0	1	1
$\langle x_2, 2\rangle$	0	1	1	0	0	0	0	0
$\langle x_2, 3\rangle$	0	0	0	0	0	1	0	0
$\langle x_2, 4\rangle$	1	0	0	0	0	0	0	0

$$\mathrm{dom}(x_2) = \{1,2,3,4\}$$

Compact-Table Algorithm
Filtering

- Intersect every support entry with `validTuples`
- Remove value if intersection is empty

`validTuples`:

`words`	0	1	1	0	0	0	1	1

&

$\langle x_2, 1\rangle$	0	0	0	1	1	0	1	1
$\langle x_2, 2\rangle$	0	1	1	0	0	0	0	0
$\langle x_2, 3\rangle$	0	0	0	0	0	1	0	0
$\langle x_2, 4\rangle$	1	0	0	0	0	0	0	0

$$\mathrm{dom}(x_2) = \{1, 2, \cancel{3}, \cancel{4}\}$$

UPPSALA UNIVERSITET

Compact-Table Algorithm

Filtering out values


```
PROCEDURE FilterDomains(s) : store
foreach x ∈ s such that |s(x)| > 1 do
    foreach a ∈ s(x) do
        index ← residues[x, a]                    // remembered last index
        if validTuples[index] & supports[x, a][index] = 0 then
            index ← validTuples.intersectIndex(supports[x, a])
            if index ≠ −1 then
                residues[x, a] ← index
            else
                s ← s[x ↦ s(x) \ {a}]
return s
```

The Compact-Table Algorithm

- Initialisation
- Variable modifications
- Filtering
- **Putting it all together**

The Compact-Table Algorithm

Putting it all together

PROCEDURE COMPACTTABLE(s : store) : $\langle$*StatusMsg*, store$\rangle$

if the propagator is being posted **then**
    $s \leftarrow$ INITIALISECT(s, T_0)
    if $s = \emptyset$ **then**
        return $\langle$**FAIL**, $\emptyset\rangle$
else
    foreach variable $x \in s$ whose domain has changed since last time **do**
        UPDATETABLE(s, x)
        if `validTuples`.isEmpty() **then**
            return $\langle$**FAIL**, $\emptyset\rangle$
    if `validTuples` has changed since last time **then**
        $s \leftarrow$ FILTERDOMAINS(s)
if there is at most one unassigned variable left **then**
    return $\langle$**SUBSUMED**, $s\rangle$
else
    return $\langle$**FIX**, $s\rangle$

Outline

1. **Background**
   - Constraint Programming
   - Gecode
   - The Compact-Table algorithm
2. **The Compact-Table Algorithm**
3. **Evaluation**
   - Setup
   - Results
   - Discussion
4. **Conclusions**

UPPSALA UNIVERSITET

Outline

1. Background
   - Constraint Programming
   - Gecode
   - The Compact-Table algorithm
2. The Compact-Table Algorithm
3. Evaluation
   - Setup
   - Results
   - Discussion
4. **Conclusions**