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

16th May 2017

Supervisor: Mats Carlsson (SICS)
Reviewer: Pierre Flener

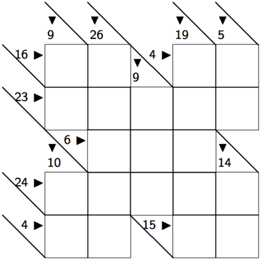
Outline

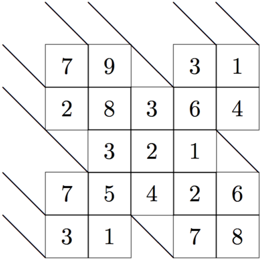
UPPSALA UNIVERSITET

Outline

1. **Background**
   - Constraint Problems
   - Constraint Propagation
   - Gecode
   - The Compact Table algorithm
2. **Algorithms**
   - Sparse bit-set
   - Compact Table
3. **Evaluation**
   - Setup
   - Results
   - Discussion
4. **Conclusions**

Kakuro puzzle





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

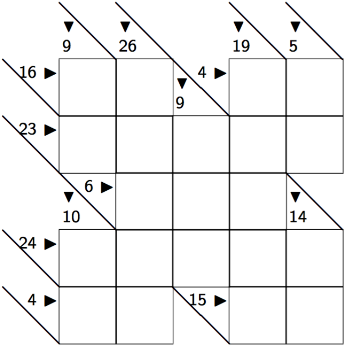
- digits are distinct, and
- the sum of the digits is equal to the *clue*

Kakuro puzzle as a constraint problem (1)

Variables One per empty cell.

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

Constraints For each row and column: *distinct* digits, and the *sum* of the digits is equal to the clue.

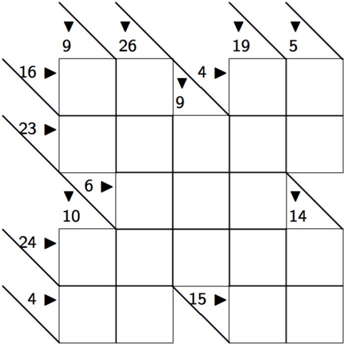


Kakuro puzzle as a constraint problem (2)

Variables One per empty cell.

Domains $\{1...9\}$ 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

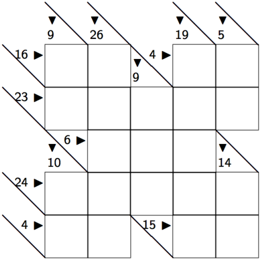
Constraint Stores

Definition (Constraint store)

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

$$s : variables \mapsto domains$$

$s(x_0) = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$s(x_1) = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$\ldots$



Propagators

Definition (Propagator)

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

$$p : store \mapsto store$$

- Implement constraints

$$\{x \mapsto \{1,2,3\},\ y \mapsto \{1,2,3\}\} \xrightarrow{p_{x<y}} \{x \mapsto \{1,2\},\ y \mapsto \{1,2,3\}\}$$

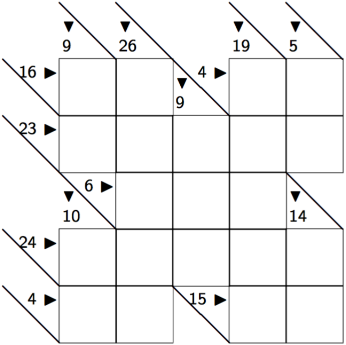
Constraint Propagation

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

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

$\ldots$



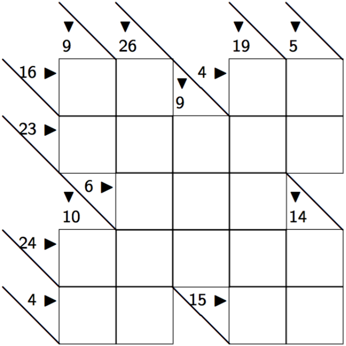
Constraint Propagation

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

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~~

$\ldots$



Constraint Propagation

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

$\dots$

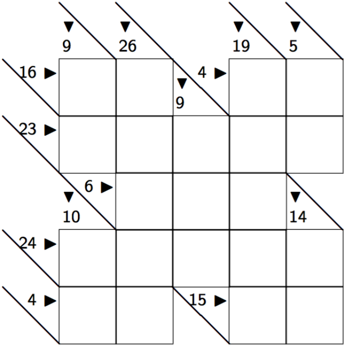
$$x_4 \in \{\cancel{1},\mathbf{2},\cancel{3},\cancel{4},\cancel{5},\cancel{6},\cancel{7},\cancel{8},\cancel{9}\}$$

$\dots$

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~~

$\dots$



Constraint Propagation

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

$\dots$

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

$\dots$

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~~

$\dots$

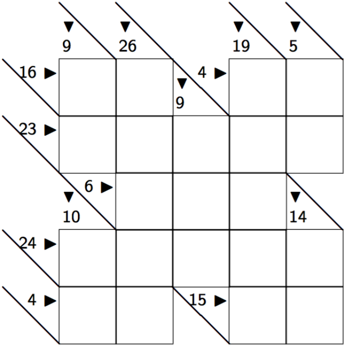


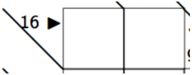
TABLE constraints

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

UPPSALA UNIVERSITET

Compact Table

UPPSALA UNIVERSITET

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

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

x_0	7	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

UPPSALA UNIVERSITET

Background

Algorithms

Sparse bit-set

Compact Table

Evaluation

Conclusions

The Compact Table Algorithm

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

x_0	7	**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

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

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

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

`validTuples`:

`words`	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
$\langle x_0, 4\rangle$	0	0	0	1	0	0	0	1
$\langle x_1, 1\rangle$	1	0	0	0	1	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

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

`validTuples`:

`words`	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
$\langle x_0, 4 \rangle$	0	0	0	1	0	0	0	1
$\langle x_1, 1 \rangle$	1	0	0	0	1	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**

UPPSALA UNIVERSITET

The Compact Table Algorithm

$s(x_0) = \{1, 2, 4\}$

$s(x_1) = s(x_2) = \{1, 2, 3, 4\}$

validTuples:

words	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
$\langle x_0, 4\rangle$	0	0	0	1	0	0	0	1
$\langle x_1, 1\rangle$	1	0	0	0	1	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**

UPPSALA UNIVERSITET

The Compact Table Algorithm

Background

Algorithms

Sparse bit-set

Compact Table

Evaluation

Conclusions

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

UPPSALA UNIVERSITET

The Compact Table Algorithm

validTuples:

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

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

The Compact Table Algorithm

validTuples:

words	1	1	1	1	1	1	1	1
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

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

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

The Compact Table Algorithm

`validTuples:`

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

`words = words && mask`

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

The Compact Table Algorithm

validTuples:

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

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

Procedure for updating `validTuples`

PROCEDURE UPDATETABLE(s: store, x: variable)

1: `validTuples`.clearMask()
2: **foreach** $a \in s(x)$ **do**
3:     `validTuples`.addToMask(`supports`$[x, a]$)
4: `validTuples`.intersectWithMask()

Filtering out values

- 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

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

Filtering out values

- 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

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

Filtering out values

PROCEDURE $\textsc{FilterDomains}(s)$: store

foreach $x \in s$ such that $|s(x)| > 1$ **do**
  foreach $a \in s(x)$ **do**
    $index \leftarrow \texttt{residues}[x, a]$
    if $\texttt{validTuples}[index]$ & $\texttt{supports}[x, a][index] = 0$ **then**
      $index \leftarrow \texttt{validTuples}.\text{intersectIndex}(\texttt{supports}[x, a])$
      if $index \neq -1$ **then**
        $\texttt{residues}[x, a] \leftarrow index$
      else
        $s \leftarrow s[x \mapsto s(x) \setminus \{a\}]$
return s

```
PROCEDURE CompactTable(s : store) : ⟨StatusMsg, store⟩
if the propagator is being posted then
    s ← InitialiseCT(s, T_0)
    if s = ∅ then
        return ⟨FAIL, ∅⟩
else
    foreach variable x ∈ s whose domain has changed since last time do
        UpdateTable(s, x)
        if validTuples.isEmpty() then
            return ⟨FAIL, ∅⟩
    if validTuples has changed since last time then
        s ← FilterDomains(s)
if there is at most one unassigned variable left then
    return ⟨SUBSUMED, s⟩
else
    return ⟨FIX, s⟩
```

Algorithm 1: Compact Table Propagator.

Outline

Outline