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

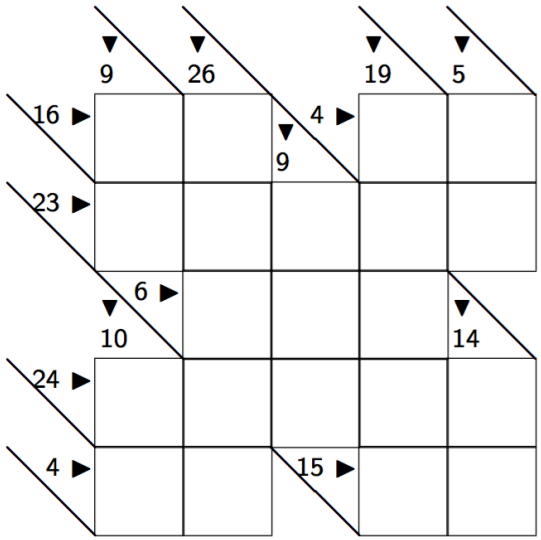
17th May 2017

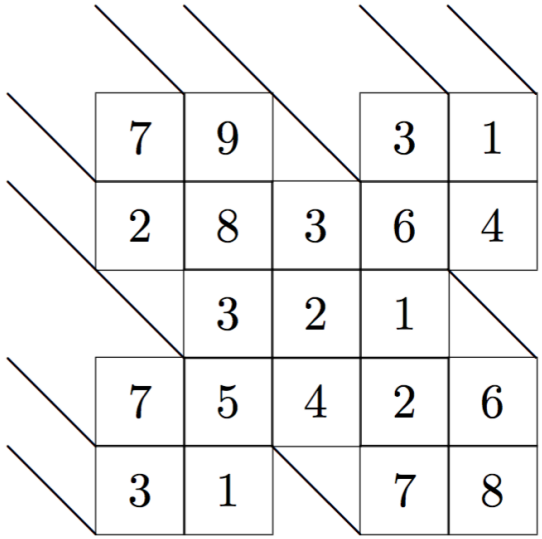
Supervisor: Mats Carlsson
Reviewer: Pierre Flener

Outline

Outline

Kakuro puzzle





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

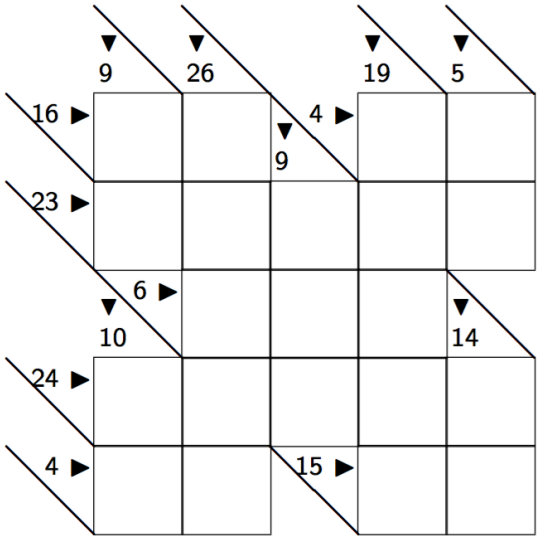
- digits are distinct, and
- the sum of the digits is equal to the *clue*

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.

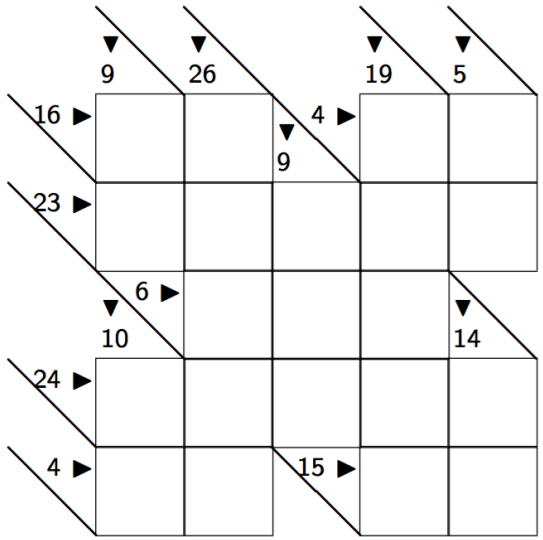


Kakuro puzzle as a constraint problem (2)

Variables One per cell.

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

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

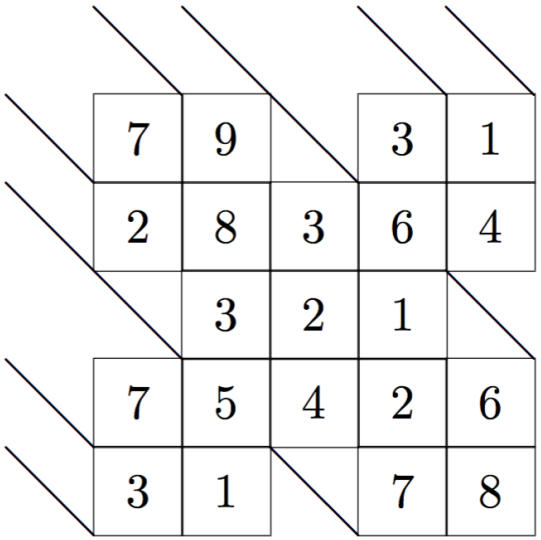


For an entry of size 2 and clue 4: $\langle 1,3 \rangle$ and $\langle 3,1 \rangle$ are the only combinations.

Solving Constraint Problems

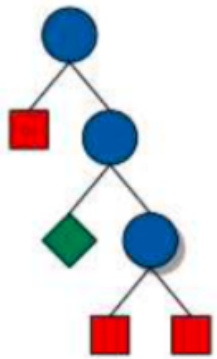
Solution A complete variable-value assignment satisfying the constraints.

(Sometimes: maximises/minimises given function.)



7	9		3	1
2	8	3	6	4
	3	2	1	
7	5	4	2	6
3	1		7	8

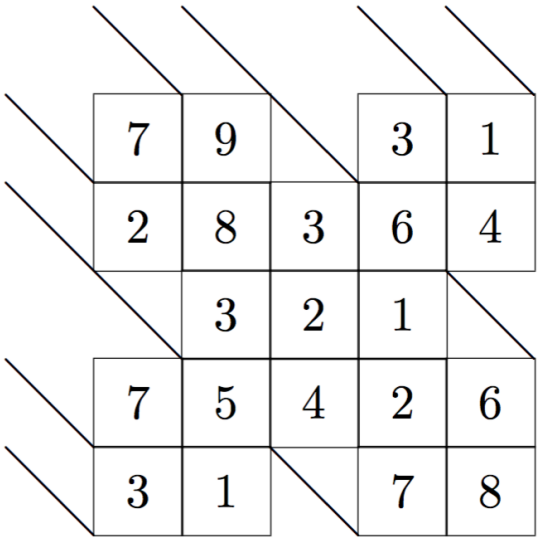
- Solutions are found by **search**
  - Propagation
  - Branching



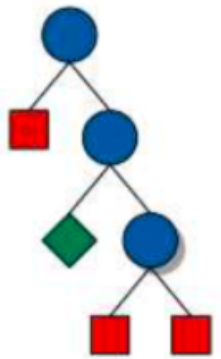
Solving Constraint Problems

Solution A complete variable-value assignment satisfying the constraints.

(Sometimes: maximises/minimises given function.)



- Solutions are found by **search**
  - **Propagation**
  - Branching



UPPSALA UNIVERSITET

Constraint Propagation

Important concepts:

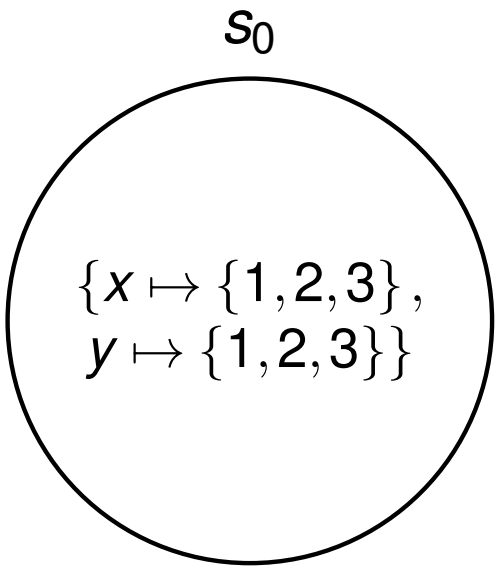
- Constraint store
- Propagator

Constraint Stores

Definition (Constraint store)

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

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



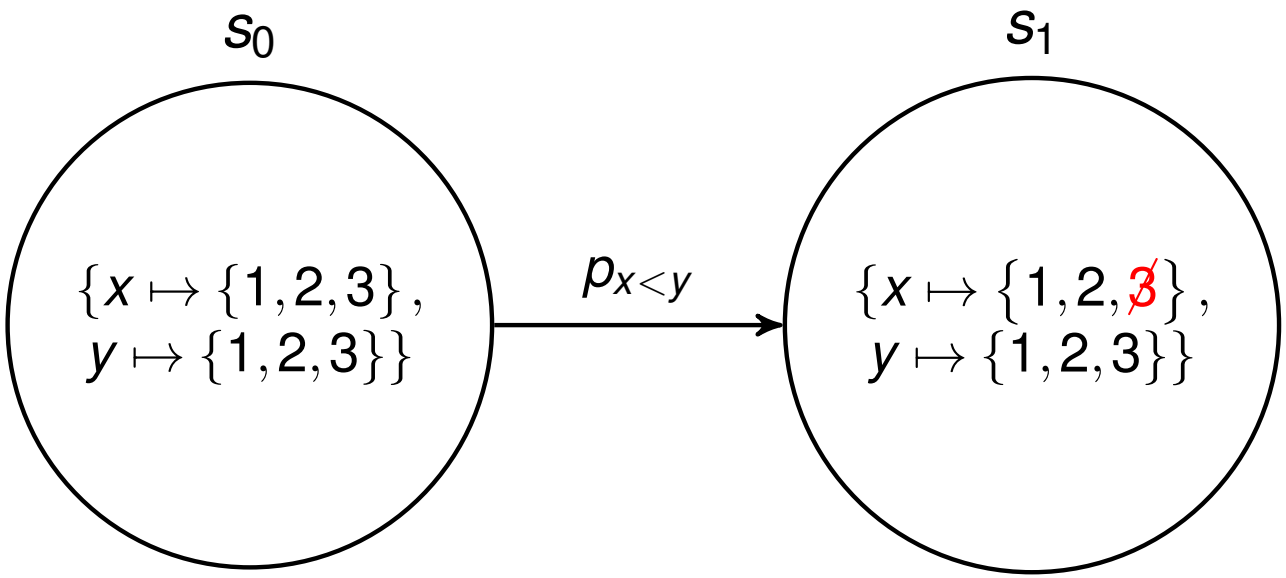
Propagators

Definition (Propagator)

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

$$p : store \mapsto store$$

- Implement constraints



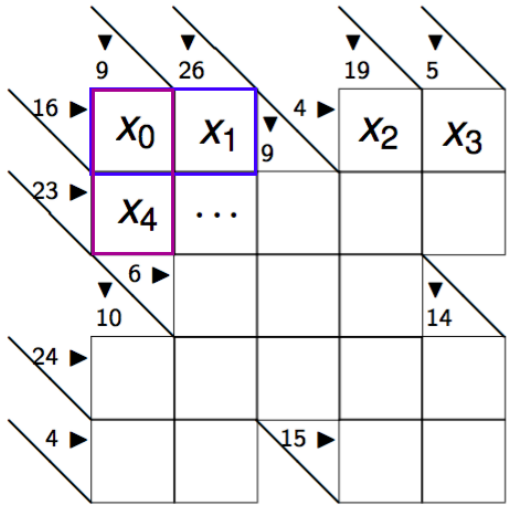
Constraint Propagation

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

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

...



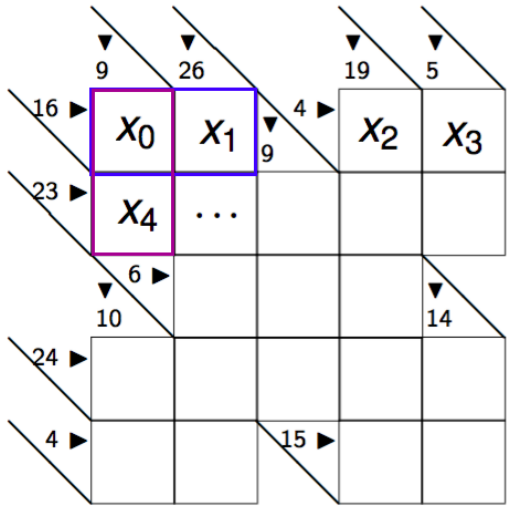
Constraint Propagation

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

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~~

...



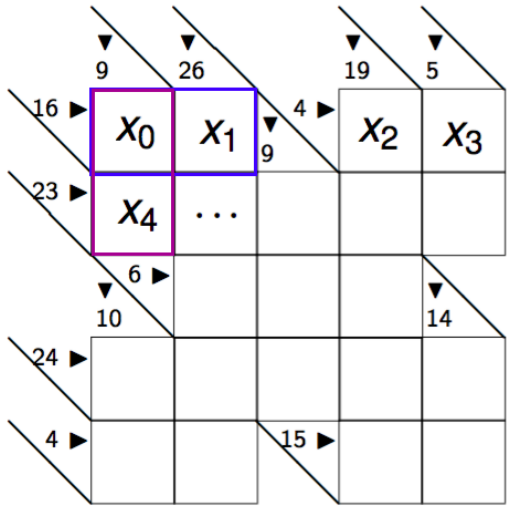
Constraint Propagation

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

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

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

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~~

...

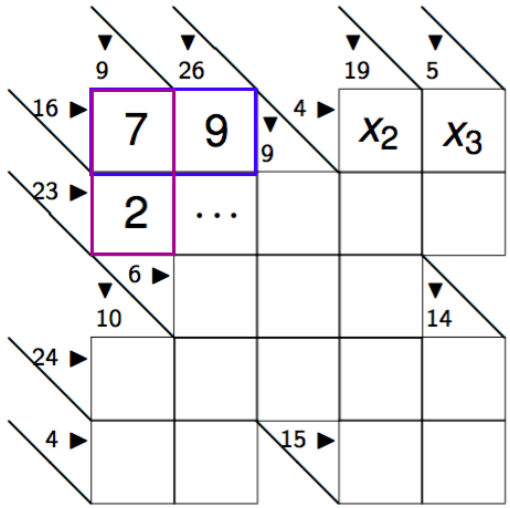


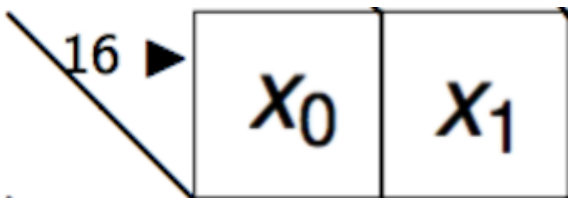
TABLE constraints

Definition (TABLE constraint)

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

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

generic
constraint
development
environment

Compact Table

Compact-Table

- A new propagation algorithm for the TABLE constraint.
- First implemented in OR-tools
- Published in a 2016 paper
- Promising results
- 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

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

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

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

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

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

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**
	0	1	2	3	4	5	6	7

The Compact-Table Algorithm

Initialisation

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

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

}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**
	0	1	2	3	4	5	6	7

The Compact-Table Algorithm

Initialisation

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

	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	} validTuples
$\langle x_0, 1\rangle$	0	**1**	0	0	**1**	0	0	0	} supports
$\langle x_0, 2\rangle$	**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**
	0	1	2	3	4	5	6	7

The Compact-Table Algorithm

Initialisation

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

	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	} validTuples
$\langle x_0, 1 \rangle$	0	**1**	0	0	**1**	0	0	0	} supports
$\langle x_0, 2 \rangle$	**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**
	0	1	2	3	4	5	6	7

The Compact-Table Algorithm

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

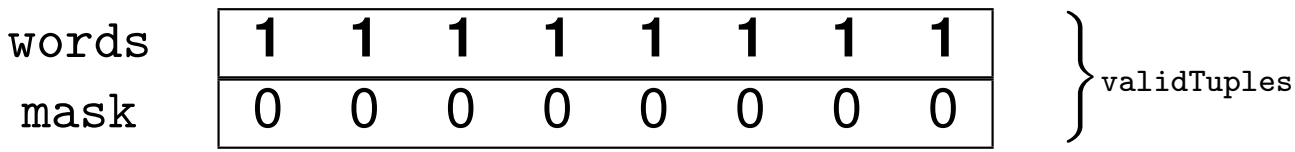
The Compact-Table Algorithm

When a variable is modified

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

The Compact-Table Algorithm

When a variable is modified



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

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

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

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

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

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

The Compact-Table Algorithm

When a variable is modified

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

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

The Compact-Table Algorithm

When a variable is modified

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

`words = words & mask`

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

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

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

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

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**

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

The Compact-Table Algorithm

When a variable is modified

PROCEDURE UPDATETABLE(s: store, x: variable)

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

The Compact-Table Algorithm

- 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

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

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

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

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

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

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

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

Compact-Table Algorithm

Filtering out values

PROCEDURE FILTERDOMAINS(s) : store

1: **foreach** variable $x \in s$ such that $|s(x)| > 1$ **do**
2: **foreach** value $a \in s(x)$ **do**
3: $index \leftarrow \texttt{residues}[x, a]$ // remembered last index
4: **if** $\texttt{validTuples}[index]$ & $\texttt{supports}[x, a][index] = 0$ **then**
5: $index \leftarrow \texttt{validTuples}.\text{intersectIndex}(\texttt{supports}[x, a])$
6: **if** $index \neq -1$ **then**
7: $\texttt{residues}[x, a] \leftarrow index$
8: **else**
9: $s \leftarrow s[x \mapsto s(x) \setminus \{a\}]$
10: **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$\rangle$

```
 1: if the propagator is being posted then
 2:     s ← InitialiseCT(s, T_0)
 3:     if s = ∅ then
 4:         return ⟨FAIL, ∅⟩
 5: else
 6:     foreach variable x ∈ s whose domain has changed since
        last time do
 7:         UpdateTable(s, x)
 8:         if validTuples.isEmpty() then
 9:             return ⟨FAIL, ∅⟩
10:     if validTuples has changed since last time then
11:         s ← FilterDomains(s)
12: if there is at most one unassigned variable left then
13:     return ⟨SUBSUMED, s⟩
14: else
15:     return ⟨FIX, s⟩
```

Outline

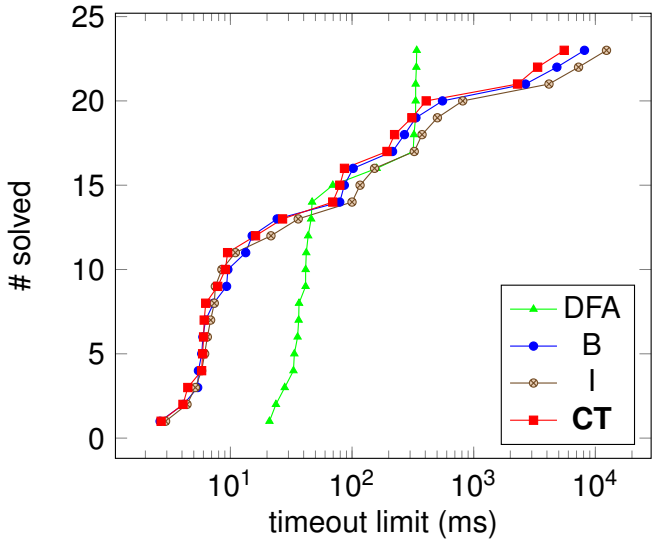
Experiments

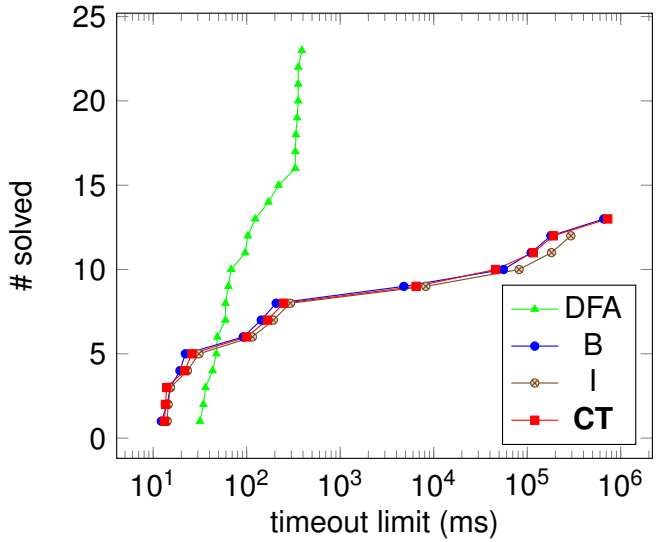
- Compared CT against
  - 2 previously existing propagators for TABLE
  - 1 propagator for REGULAR
- 1507 instances over 30 series.

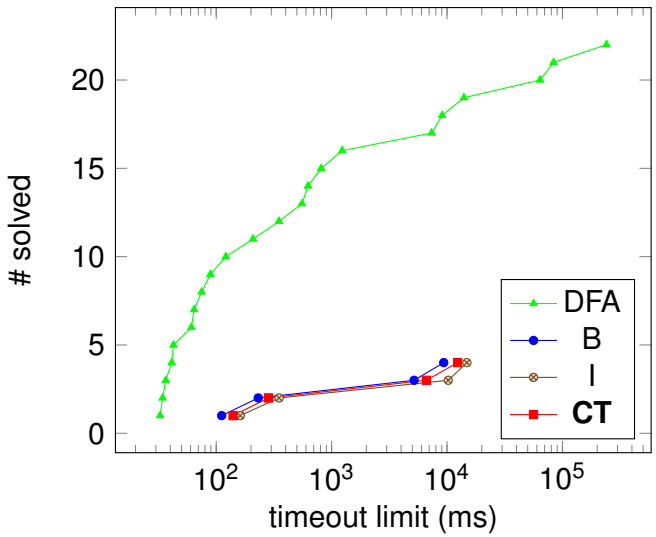
Annotations

CT	Compact-Table propagator
DFA	Layered graph (DFA) propagator
B	Basic tuple set propagator
I	Incremental tuple set propagator

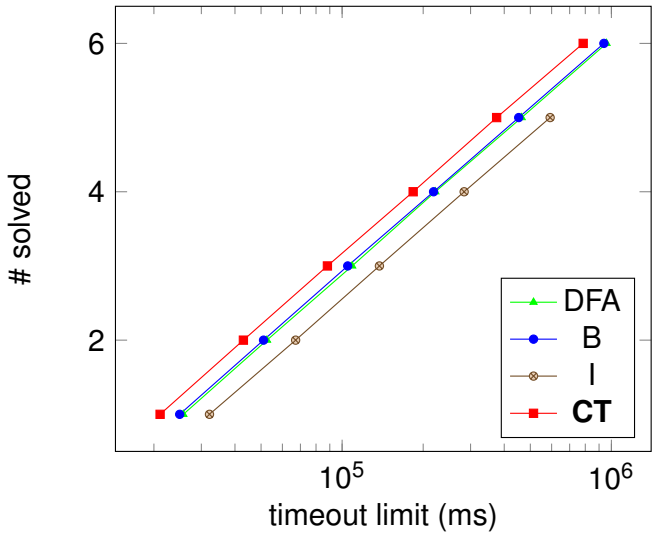
Small tables

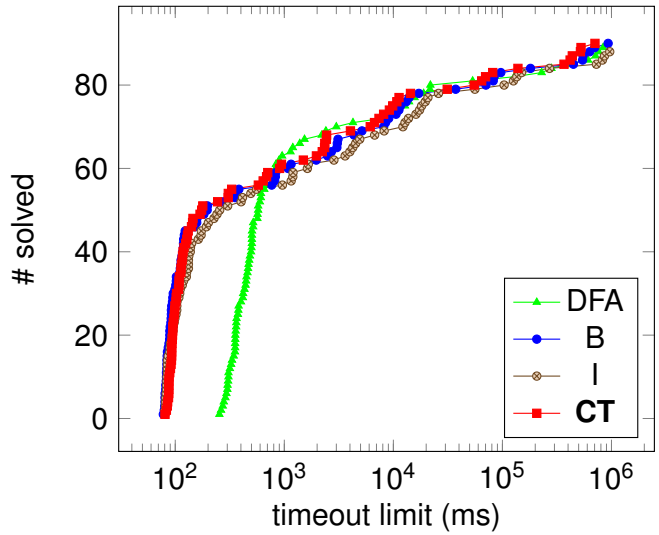




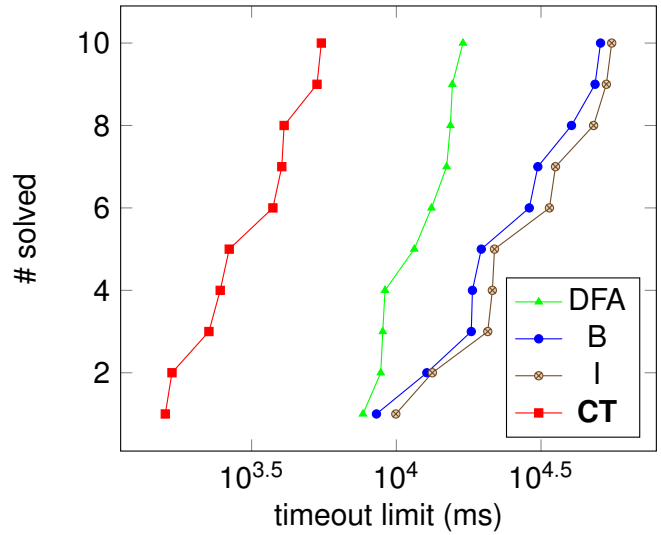


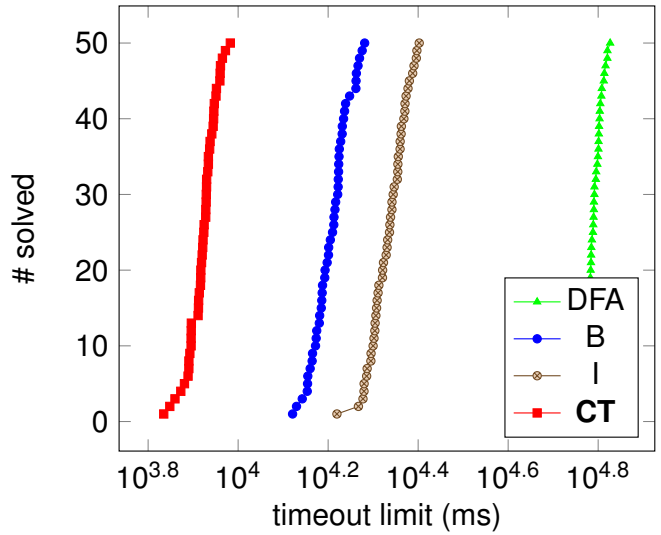
Small tables

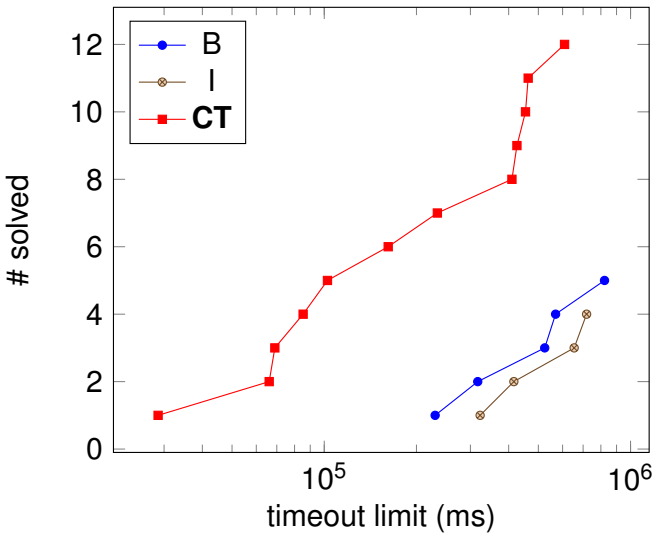


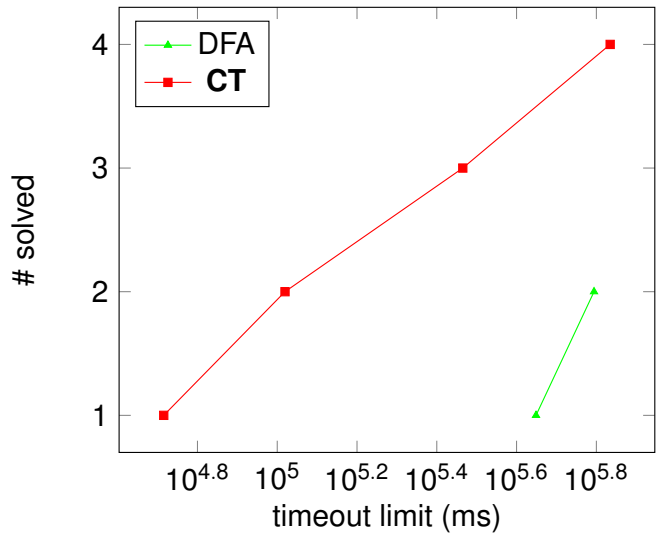


Large tables

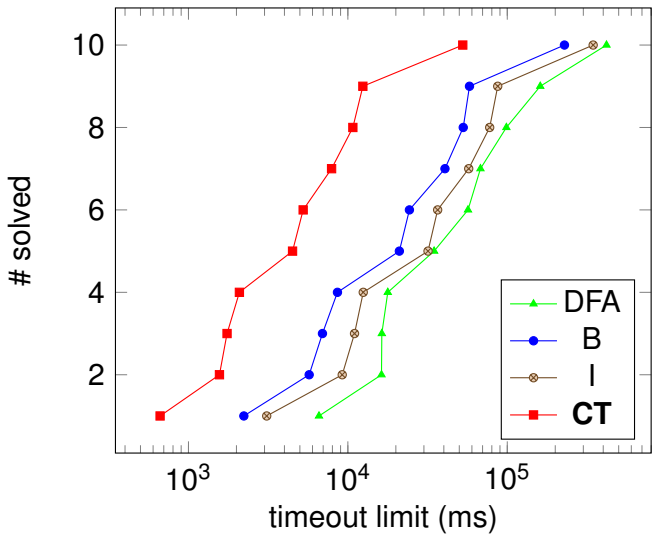


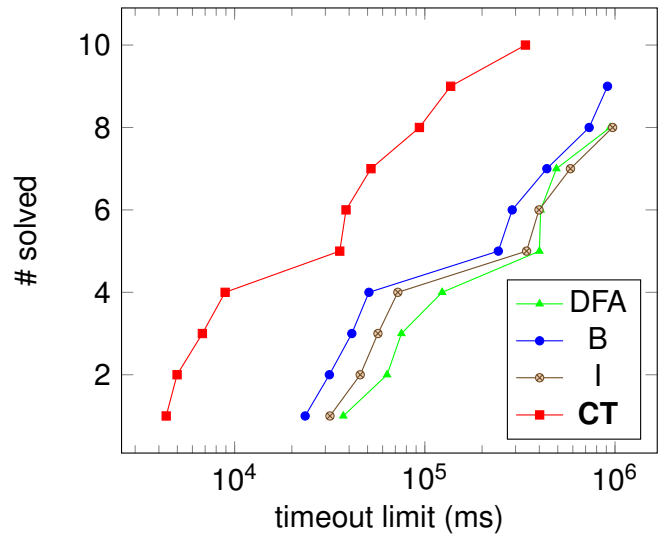


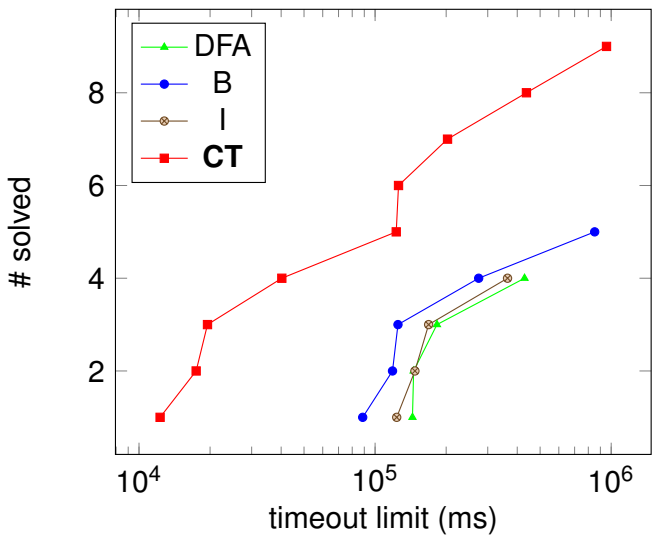


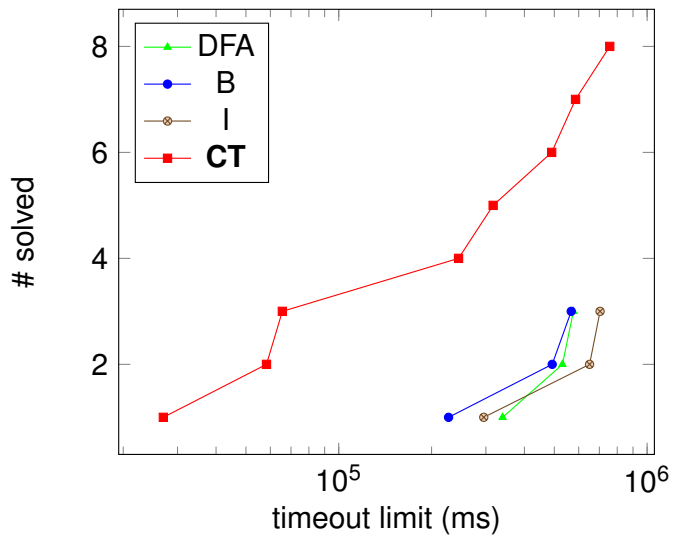


Large tables

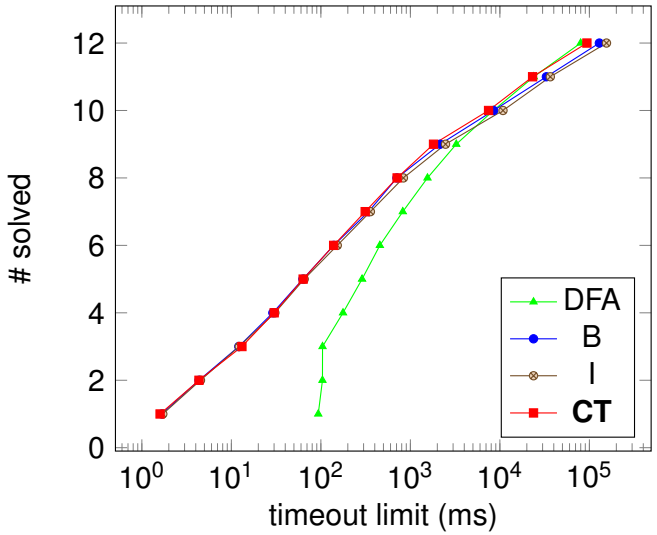


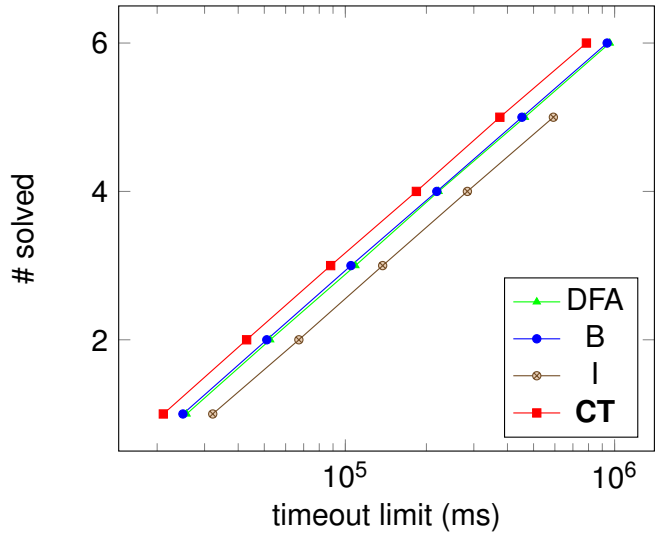


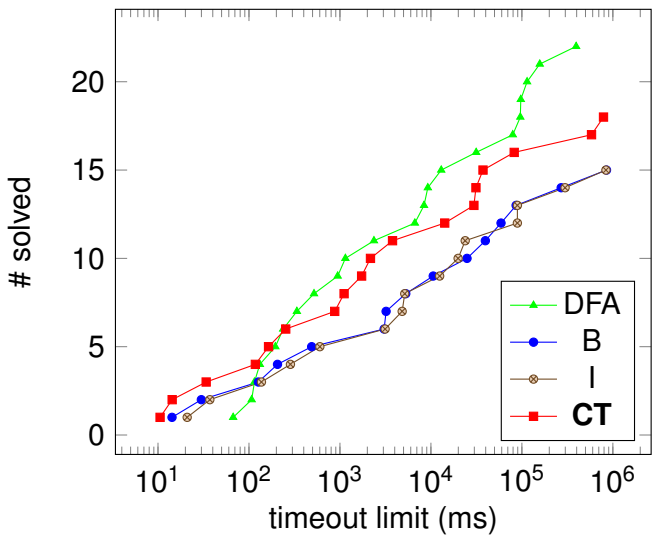


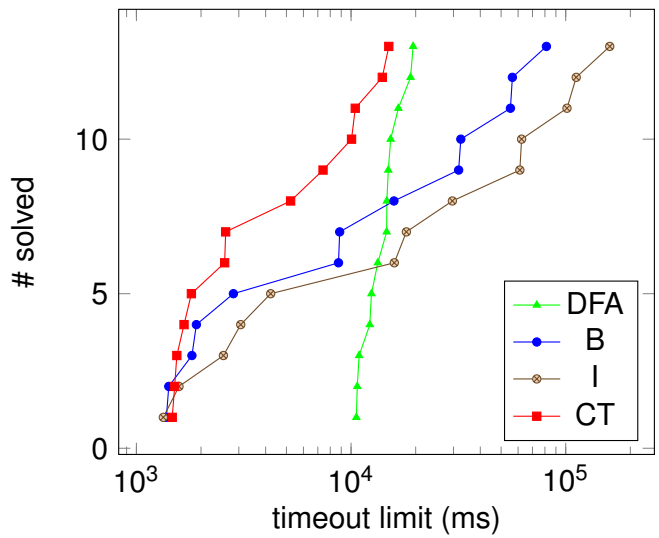


Low arities

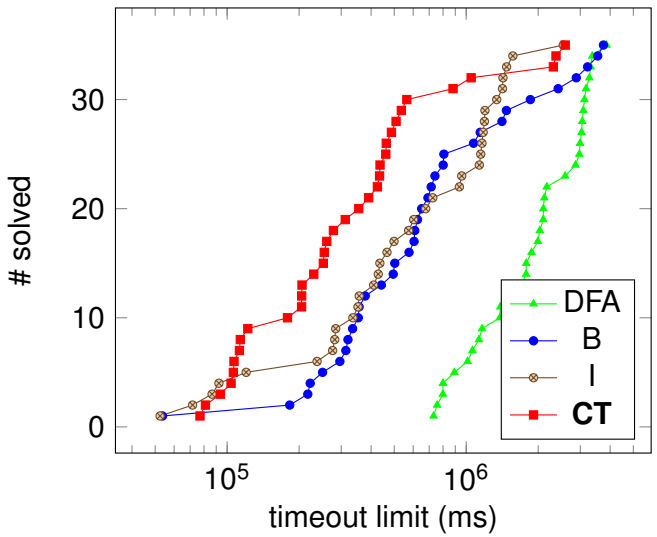

12
10
8
6
4
2
0
solved
DFA
B
I
CT
10^0 10^1 10^2 10^3 10^4 10^5
timeout limit (ms)

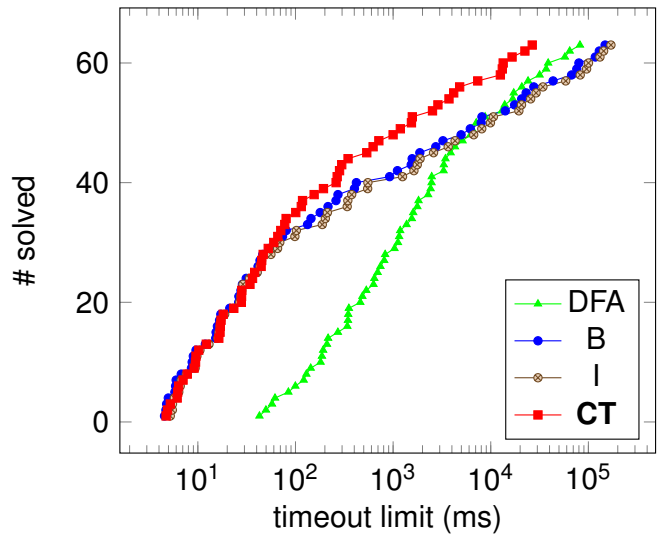

6
4
2
solved
DFA
B
I
CT
10^5 10^6
timeout limit (ms)

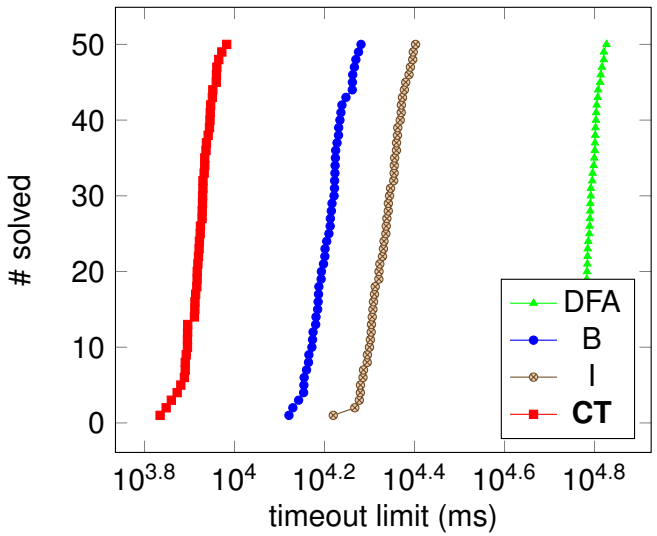

20
15
10
5
0
solved
DFA
B
I
CT
10^1 10^2 10^3 10^4 10^5 10^6
timeout limit (ms)

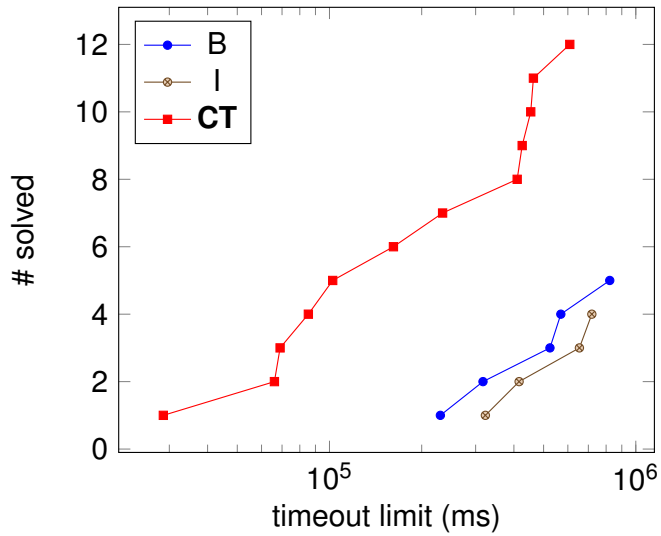

10
5
0
solved
DFA
B
I
CT
10^3 10^4 10^5
timeout limit (ms)

High arities

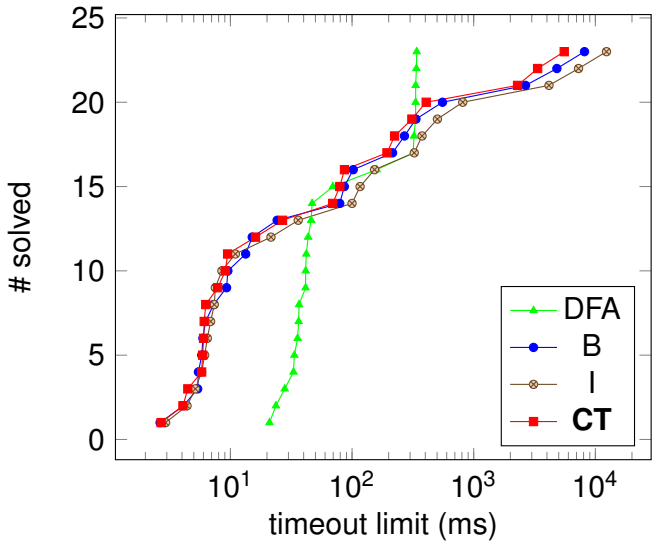


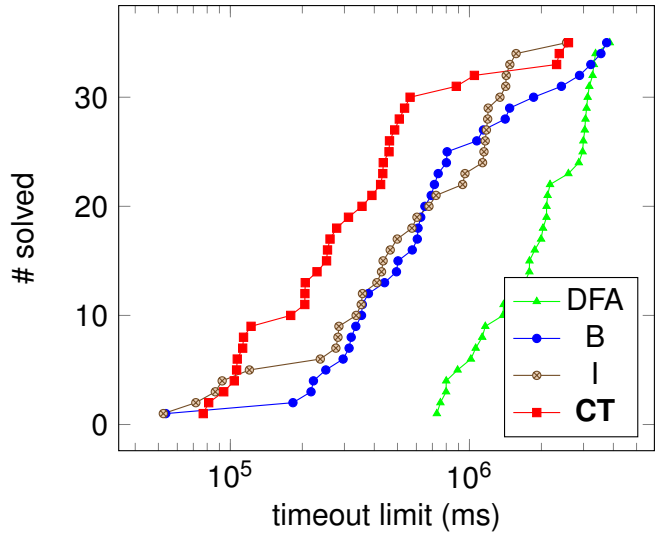




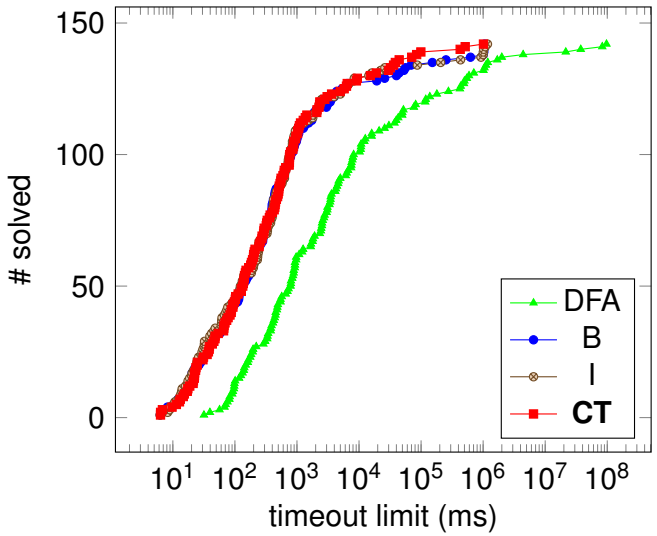


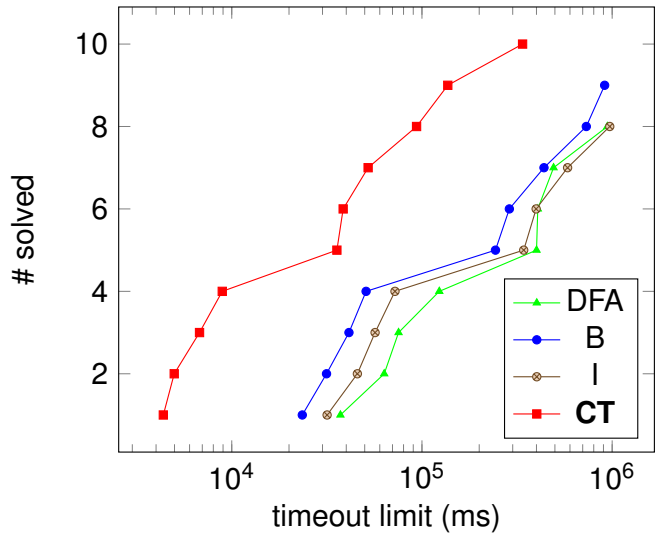
Small domains





Large domains





Results

- On all series, CT either outperforms or performs as well as the existing propagators for TABLE
- DFA overall slowest, but fastest on some series
- Largest performance gain on series with large tables

Outline

Summary and Conclusions

- What I have done:
  - Implemented Compact-Table (CT) in Gecode
  - Compared CT against existing propagators for TABLE, and with REGULAR
- The results:
  - CT seems to outperform existing propagators for TABLE
- Future work:
  - Inclusion into Gecode?
  - Implement and evaluate generalisations of CT described in another article