Table Constraints for Integer Programming (CP'26)

OPTIMA Seminar, June 10, 2026

Hendrik Bierlee[1] Wout Piessens[1] Tias Guns[1] Peter J. Stuckey[2,3]

[1]KU Leuven, Belgium

[2]Monash University, Melbourne, Australia

[3]OPTIMA ITTC, Melbourne, Australia

Paper

Slides

Funded by ERC (CHAT-Opt, 101002802), FWO (G020525N), and ARC (OPTIMA).

Table Constraints in CP

- Solver-independent modelling languages (e.g. MiniZinc [7], CPMpy [4]) solve CP models using ILP solvers
- But there is a large perfomance gap between ILP and e.g. CP-SAT
  1. Because important global constraints are not linearized as well as they can be
  2. Because we do not use the full ILP toolset

Running example: $\texttt{table}([x, y, z], T)$ with enforces the variables to take one row of T:

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

Integer encoding (INT)

- Standard encoding by Belov *et al.* [2] (default in MiniZinc)
- Binary variables $r_i \in \{0, 1\}$ for each row i, s.t. $r_i = 1$ iff row i chosen

Table T:

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

Encoding:

$$\sum_{l \in 1..m} r_l = 1$$

$$\sum_{l \in 1..m} T_{l,i} r_l = x_i, \quad i \in 1..k$$

Example:

$$\begin{array}{rrrrrcl} r_1 & +r_2 & +r_3 & +r_4 & +r_5 & = & 1 \\ 2r_1 & +3r_2 & +4r_3 & +1r_4 & +2r_5 & = & x \\ 1r_1 & +2r_2 & +3r_3 & +2r_4 & +1r_5 & = & y \\ 1r_1 & +2r_2 & +3r_3 & +3r_4 & +2r_5 & = & z \end{array}$$

Integer encoding (INT)

- Standard encoding by Belov *et al.* [2] (default in MiniZinc)
- Binary variables $r_i \in \{0,1\}$ for each row i, s.t. $r_i = 1$ iff row i chosen

Table T:

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

Encoding:

$$\sum_{l \in 1..m} r_l = 1$$

$$\sum_{l \in 1..m} T_{l,i} r_l = x_i, \quad i \in 1..k$$

Example:

$$\begin{array}{rrrrrcl} r_1 & +r_2 & +r_3 & +r_4 & +r_5 & = & 1 \\ 2r_1 & +3r_2 & +4r_3 & +1r_4 & +2r_5 & = & x \\ 1r_1 & +2r_2 & +3r_3 & +2r_4 & +1r_5 & = & y \\ 1r_1 & +2r_2 & +3r_3 & +3r_4 & +2r_5 & = & z \end{array}$$

Booleanized encoding (BOOL) [8]

We booleanize x with binary variables $[\![x = j]\!] \in \{0, 1\}, j \in D(x)$ such that $[\![x = j]\!] \Leftrightarrow x = j$:

$$\sum_{j \in D(x)} [\![x = j]\!] = 1, \qquad \sum_{j \in D(x)} j \cdot [\![x = j]\!] = x$$

E.g. $[\![x = 1]\!] + [\![x = 2]\!] + [\![x = 3]\!] + [\![x = 4]\!] = 1, 1 \cdot [\![x = 1]\!] + 2 \cdot [\![x = 2]\!] + 3 \cdot [\![x = 3]\!] + 4 \cdot [\![x = 4]\!] = x$

Booleanized encoding (BOOL) [8]

We booleanize x with binary variables $[\![x = j]\!] \in \{0, 1\}, j \in D(x)$ such that $[\![x = j]\!] \Leftrightarrow x = j$:

$$\sum_{j \in D(x)} [\![x = j]\!] = 1, \qquad \sum_{j \in D(x)} j \cdot [\![x = j]\!] = x$$

E.g. $[\![x = 1]\!] + [\![x = 2]\!] + [\![x = 3]\!] + [\![x = 4]\!] = 1, 1 \cdot [\![x = 1]\!] + 2 \cdot [\![x = 2]\!] + 3 \cdot [\![x = 3]\!] + 4 \cdot [\![x = 4]\!] = x$

And booleanize T as a binary table $T^{\mathbb{B}}$:

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

$\equiv$

$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$
0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1
1	0	0	0	0	1	0	0	0	1
0	1	0	0	1	0	0	0	1	0

Booleanized encoding (BOOL) [8]

We booleanize x with binary variables $[\![x = j]\!] \in \{0, 1\}, j \in D(x)$ such that $[\![x = j]\!] \Leftrightarrow x = j$:

$$\sum_{j \in D(x)} [\![x = j]\!] = 1, \qquad \sum_{j \in D(x)} j \cdot [\![x = j]\!] = x$$

E.g. $[\![x = 1]\!] + [\![x = 2]\!] + [\![x = 3]\!] + [\![x = 4]\!] = 1, 1 \cdot [\![x = 1]\!] + 2 \cdot [\![x = 2]\!] + 3 \cdot [\![x = 3]\!] + 4 \cdot [\![x = 4]\!] = x$

And booleanize T as a binary table $T^{\mathbb{B}}$:

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

$\equiv$

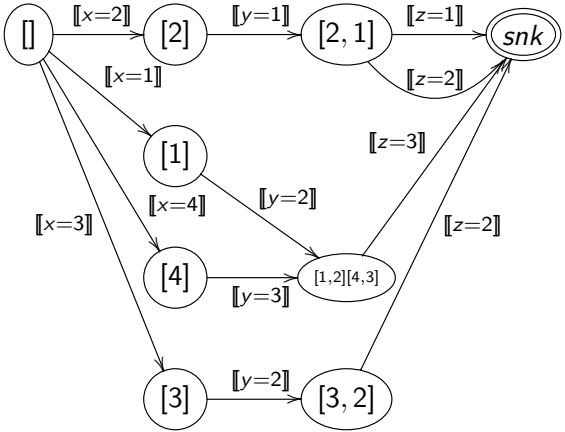
$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$
0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	0	1
1	0	0	0	0	1	0	0	0	1
0	1	0	0	1	0	0	0	1	0

MDD-based Flow Encoding (MDD)

We can reformulate $T^{\mathbb{B}}$ as a reduced Multi-valued Decision Diagram (MDD) [3]

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

$\equiv T^{\mathbb{B}} \equiv$



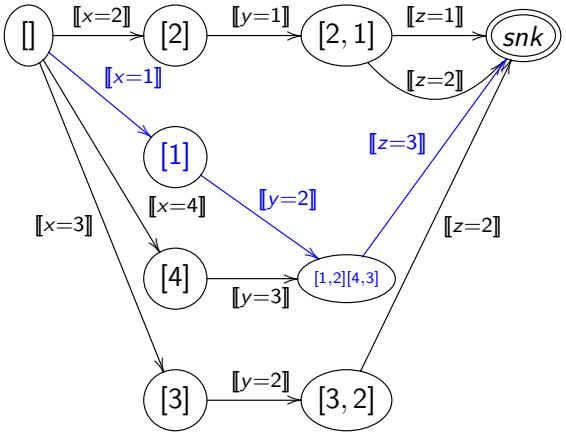
$\equiv$ flow constraints

MDD-based Flow Encoding (MDD)

We can reformulate $T^{\mathbb{B}}$ as a reduced Multi-valued Decision Diagram (MDD) [3]

x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

$\equiv T^{\mathbb{B}} \equiv$



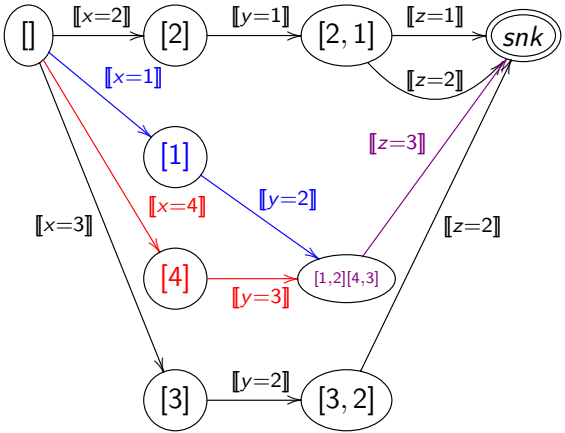
$\equiv$ flow constraints

MDD-based Flow Encoding (MDD)

We can reformulate $T^{\mathbb{B}}$ as a reduced Multi-valued Decision Diagram (MDD) [3]

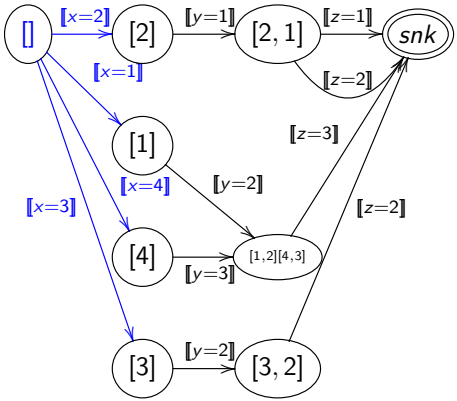
x	y	z
2	1	1
3	2	2
4	3	3
1	2	3
2	1	2

$\equiv T^{\mathbb{B}} \equiv$



$\equiv$ flow constraints

MDD to flow constraints



Flow constraints:

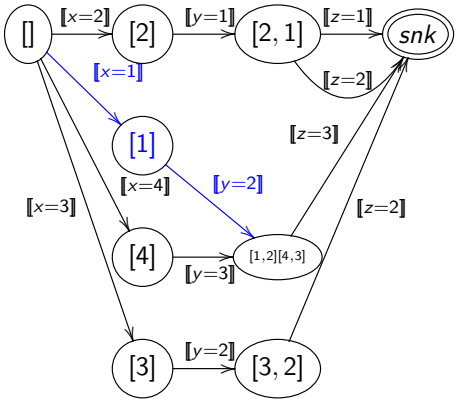
$$[\![x=1]\!] + [\![x=2]\!] + [\![x=3]\!] + [\![x=4]\!] = 1 \quad \text{src}$$

$$\begin{aligned}
[\![x=1]\!] &= e_{([1],2)} && [1] \\
e_{([1],2)} + [\![y=3]\!] &= [\![z=3]\!] && [1,2][4,3] \\
[\![x=2]\!] &= [\![y=1]\!] && [2] \\
[\![y=1]\!] &= [\![z=1]\!] + e_{([2,1],2)} && [2,1] \\
[\![x=4]\!] &= [\![y=3]\!] && [4] \\
[\![x=3]\!] &= e_{([3],2)} && [3] \\
e_{([3],2)} &= e_{([3,2],2)} && [3,2]
\end{aligned}$$

$$e_{([2,1],2)} + [\![z=1]\!] + [\![z=3]\!] + e_{([3,2],2)} = 1 \quad \text{snk}$$

$$\begin{aligned}
e_{([1],2)} + e_{([3],2)} &= [\![y=2]\!] && \text{channel} \\
e_{([2,1],2)} + e_{([3,2],2)} &= [\![z=2]\!] && \text{channel}
\end{aligned}$$

MDD to flow constraints



Flow constraints:

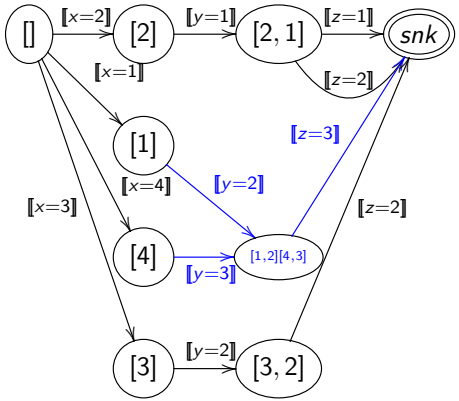
$$[\![x=1]\!] + [\![x=2]\!] + [\![x=3]\!] + [\![x=4]\!] = 1 \quad \text{src}$$

$$\begin{aligned}
[\![x=1]\!] &= e_{([1],2)} && [1] \\
e_{([1],2)} + [\![y=3]\!] &= [\![z=3]\!] && [1,2][4,3] \\
[\![x=2]\!] &= [\![y=1]\!] && [2] \\
[\![y=1]\!] &= [\![z=1]\!] + e_{([2,1],2)} && [2,1] \\
[\![x=4]\!] &= [\![y=3]\!] && [4] \\
[\![x=3]\!] &= e_{([3],2)} && [3] \\
e_{([3],2)} &= e_{([3,2],2)} && [3,2]
\end{aligned}$$

$$e_{([2,1],2)} + [\![z=1]\!] + [\![z=3]\!] + e_{([3,2],2)} = 1 \quad \text{snk}$$

$$\begin{aligned}
e_{([1],2)} + e_{([3],2)} &= [\![y=2]\!] && \text{channel} \\
e_{([2,1],2)} + e_{([3,2],2)} &= [\![z=2]\!] && \text{channel}
\end{aligned}$$

MDD to flow constraints



Flow constraints:

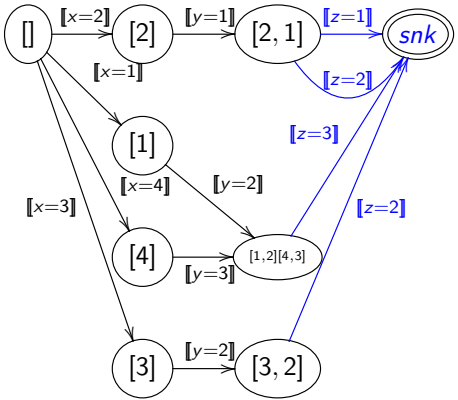
$$[\![x=1]\!]+[\![x=2]\!]+[\![x=3]\!]+[\![x=4]\!]=1 \quad \text{src}$$

$$\begin{aligned}
[\![x=1]\!] &= e_{([1],2)} && [1]\\
e_{([1],2)}+[\![y=3]\!] &= [\![z=3]\!] && [1,2][4,3]\\
[\![x=2]\!] &= [\![y=1]\!] && [2]\\
[\![y=1]\!] &= [\![z=1]\!]+e_{([2,1],2)} && [2,1]\\
[\![x=4]\!] &= [\![y=3]\!] && [4]\\
[\![x=3]\!] &= e_{([3],2)} && [3]\\
e_{([3],2)} &= e_{([3,2],2)} && [3,2]
\end{aligned}$$

$$e_{([2,1],2)}+[\![z=1]\!]+[\![z=3]\!]+e_{([3,2],2)}=1 \quad \text{snk}$$

$$\begin{aligned}
e_{([1],2)}+e_{([3],2)} &= [\![y=2]\!] && \text{channel}\\
e_{([2,1],2)}+e_{([3,2],2)} &= [\![z=2]\!] && \text{channel}
\end{aligned}$$

MDD to flow constraints



Flow constraints:

$$\llbracket x=1 \rrbracket + \llbracket x=2 \rrbracket + \llbracket x=3 \rrbracket + \llbracket x=4 \rrbracket = 1 \quad \text{src}$$

$$\begin{aligned}
\llbracket x=1 \rrbracket &= e_{([1],2)} && [1] \\
e_{([1],2)} + \llbracket y=3 \rrbracket &= \llbracket z=3 \rrbracket && [1,2][4,3] \\
\llbracket x=2 \rrbracket &= \llbracket y=1 \rrbracket && [2] \\
\llbracket y=1 \rrbracket &= \llbracket z=1 \rrbracket + e_{([2,1],2)} && [2,1] \\
\llbracket x=4 \rrbracket &= \llbracket y=3 \rrbracket && [4] \\
\llbracket x=3 \rrbracket &= e_{([3],2)} && [3] \\
e_{([3],2)} &= e_{([3,2],2)} && [3,2]
\end{aligned}$$

$$e_{([2,1],2)} + \llbracket z=1 \rrbracket + \llbracket z=3 \rrbracket + e_{([3,2],2)} = 1 \quad \text{snk}$$

$$\begin{aligned}
e_{([1],2)} + e_{([3],2)} &= \llbracket y=2 \rrbracket && \text{channel} \\
e_{([2,1],2)} + e_{([3,2],2)} &= \llbracket z=2 \rrbracket && \text{channel}
\end{aligned}$$

Results – Encodings (CSP, 180 instances)

Benchmarks XCSP3 competition instances (2022–2025) [1], 180 CSP instances

Solver Gurobi 13.0.1 [5], single core, 600s time limit, 8GB memory

	POST	FEAS	SOLVED	time	constraints	cuts

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – Encodings (CSP, 180 instances)

Benchmarks XCSP3 competition instances (2022–2025) [1], 180 CSP instances

Solver Gurobi 13.0.1 [5], single core, 600s time limit, 8GB memory

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – Encodings (CSP, 180 instances)

Benchmarks XCSP3 competition instances (2022–2025) [1], 180 CSP instances

Solver Gurobi 13.0.1 [5], single core, 600s time limit, 8GB memory

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – Encodings (CSP, 180 instances)

Benchmarks XCSP3 competition instances (2022–2025) [1], 180 CSP instances

Solver Gurobi 13.0.1 [5], single core, 600s time limit, 8GB memory

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – Encodings (CSP, 180 instances)

Benchmarks XCSP3 competition instances (2022–2025) [1], 180 CSP instances

Solver Gurobi 13.0.1 [5], single core, 600s time limit, 8GB memory

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Lazy Cut Generation for Tables

Instead of an eager encoding, we can generate cuts **lazily** (i.e. row generation, logic-based Benders decomposition [6])

Lazy Cut Generation for Tables

Instead of an eager encoding, we can generate cuts **lazily** (i.e. row generation, logic-based Benders decomposition [6])

1. Solve without encoding tables, get assignment $\hat{v}$ (e.g. $\hat{v} = [2, 2, 2]$)

Lazy Cut Generation for Tables

Instead of an eager encoding, we can generate cuts **lazily** (i.e. row generation, logic-based Benders decomposition [6])

1. Solve without encoding tables, get assignment $\hat{v}$ (e.g. $\hat{v} = [2, 2, 2]$)
2. If $\hat{v}$ violates a table, generate a **cut**
   - A valid cut violates $\hat{v}$, but does not violate the table constraint
   - E.g. $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

Lazy Cut Generation for Tables

Instead of an eager encoding, we can generate cuts **lazily** (i.e. row generation, logic-based Benders decomposition [6])

1. Solve without encoding tables, get assignment $\hat{v}$ (e.g. $\hat{v} = [2, 2, 2]$)
2. If $\hat{v}$ violates a table, generate a **cut**
   - A valid cut violates $\hat{v}$, but does not violate the table constraint
   - E.g. $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$
3. Repeat until $\hat{v}$ satisfies all tables

Can we generate a stronger cut, e.g. $[\![x = 2]\!] + [\![y = 2]\!] \leq 1$?

Lazy Cut Generation for Tables

Instead of an eager encoding, we can generate cuts **lazily** (i.e. row generation, logic-based Benders decomposition [6])

1. Solve without encoding tables, get assignment $\hat{v}$ (e.g. $\hat{v} = [2, 2, 2]$)
2. If $\hat{v}$ violates a table, generate a **cut**
   - A valid cut violates $\hat{v}$, but does not violate the table constraint
   - E.g. $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$
3. Repeat until $\hat{v}$ satisfies all tables

Can we generate a stronger cut, e.g. $[\![x = 2]\!] + [\![y = 2]\!] \leq 1$?

Note: cuts with $\sum n$ unit terms $\leq n - 1$ are logical clauses, e.g.:

$$[\![x = 2]\!] + [\![y = 2]\!] \leq 1 \equiv \neg[\![x = 2]\!] \vee \neg[\![y = 2]\!]$$

GENERATE Example: $\hat{v} = (2, 2, 2)$

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$
1	0	1	0	0	1	0	0	1	0	0
2	0	0	1	0	0	1	0	0	1	0
3	0	0	0	1	0	0	1	0	0	1
4	1	0	0	0	0	1	0	0	0	1
5	0	1	0	0	1	0	0	0	1	0
$\hat{v}$	0	1	0	0	0	1	0	0	1	0

Cut: $0 \leq -1$ (all rows violated)

GENERATE Example: $\hat{v} = (2, 2, 2)$

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$
1	0	1	0	0	1	0	0	1	0	0
2	0	0	1	0	0	1	0	0	1	0
3	0	0	0	1	0	0	1	0	0	1
4	1	0	0	0	0	1	0	0	0	1
5	0	1	0	0	1	0	0	0	1	0
$\hat{v}$	0	1	0	0	0	1	0	0	1	0

Cut: $0 \leq -1$ (all rows violated)

GENERATE Example: $\hat{v} = (2, 2, 2)$

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$
1	0	1	0	0	1	0	0	1	0	0
2	0	0	1	0	0	1	0	0	1	0
3	0	0	0	1	0	0	1	0	0	1
4	1	0	0	0	0	1	0	0	0	1
5	0	1	0	0	1	0	0	0	1	0
$\hat{v}$	0	1	0	0	0	1	0	0	1	0

Choose x, add $[\![x = 2]\!]$: $\quad [\![x = 2]\!] \leq 0 \quad$ (rows 1 and 5 still violated)

GENERATE Example: $\hat{v} = (2, 2, 2)$

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$
1	0	1	0	0	1	0	0	1	0	0
2	0	0	1	0	0	1	0	0	1	0
3	0	0	0	1	0	0	1	0	0	1
4	1	0	0	0	0	1	0	0	0	1
5	0	1	0	0	1	0	0	0	1	0
$\hat{v}$	0	1	0	0	0	1	0	0	1	0

Choose y, add $[\![y = 2]\!]$: $\quad [\![x = 2]\!] + [\![y = 2]\!] \leq 1$

Extensions: CUTLIFT and FRACTIONAL

CUTLIFT Strengthen valid cut by **lifting** variables (*without* incrementing RHS)

Input: $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

Extensions: Cutlift and Fractional

Cutlift Strengthen valid cut by **lifting** variables (*without* incrementing RHS)

Input: $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$
Output: $[\![x = 1]\!] + [\![x = 2]\!] + 2[\![x = 4]\!] + [\![y = 2]\!] + [\![z = 1]\!] + [\![z = 2]\!] \leq 2$

Extensions: CUTLIFT and FRACTIONAL

CUTLIFT Strengthen valid cut by **lifting** variables (*without* incrementing RHS)

Input: $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$
Output: $[\![x = 1]\!] + [\![x = 2]\!] + 2[\![x = 4]\!] + [\![y = 2]\!] + [\![z = 1]\!] + [\![z = 2]\!] \leq 2$

FRACTIONAL Generate cuts from **fractional** LP solutions

Input: $\hat{v} = [0, 0.5, 0.5, 0, 0, 1, 0, 0.5, 0.5, 0]$

Extensions: CUTLIFT and FRACTIONAL

CUTLIFT Strengthen valid cut by **lifting** variables (*without* incrementing RHS)

Input: $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$
Output: $[\![x = 1]\!] + [\![x = 2]\!] + 2[\![x = 4]\!] + [\![y = 2]\!] + [\![z = 1]\!] + [\![z = 2]\!] \leq 2$

FRACTIONAL Generate cuts from **fractional** LP solutions

Input: $\hat{v} = [0, 0.5, 0.5, 0, 0, 1, 0, 0.5, 0.5, 0]$
Output: $[\![x = 2]\!] + [\![y = 2]\!] \leq 1$

Example: CUTLIFT on $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

- A tight row assigns variables such that the cut has LHS = RHS (e.g. row 2)
- A tight column is set to 1 in a tight row (e.g. $[\![x = 3]\!]$)
  - $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] + [\![\mathbf{x = 3}]\!] \leq 2$ would violate row 2!

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$	*RS*
1	0	1	0	0	1	0	0	1	0	0	1
2	0	0	1	0	0	1	0	0	1	0	**2**
3	0	0	0	1	0	0	1	0	0	1	0
4	1	0	0	0	0	1	0	0	0	1	1
5	0	1	0	0	1	0	0	0	1	0	**2**

Cut: $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

Example: CUTLIFT on $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

- A tight row assigns variables such that the cut has LHS = RHS (e.g. row 2)
- A tight column is set to 1 in a tight row (e.g. $[\![x = 3]\!]$)
  - $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] + [\![\mathbf{x = 3}]\!] \leq 2$ would violate row 2!

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$	*RS*
1	0	1	0	0	1	0	0	1	0	0	1
2	0	0	1	0	0	1	0	0	1	0	**2**
3	0	0	0	1	0	0	1	0	0	1	1
4	1	0	0	0	0	1	0	0	0	1	1
5	0	1	0	0	1	0	0	0	1	0	**2**

Lift $[\![x = 4]\!]$: $[\![x = 2]\!] + [\![x = 4]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

Example: CUTLIFT on $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

- A tight row assigns variables such that the cut has LHS = RHS (e.g. row 2)
- A tight column is set to 1 in a tight row (e.g. $[\![x = 3]\!]$)
  - $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] + [\![\mathbf{x = 3}]\!] \leq 2$ would violate row 2!

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$	*RS*
1	0	1	0	0	1	0	0	1	0	0	1
2	0	0	1	0	0	1	0	0	1	0	**2**
3	0	0	0	1	0	0	1	0	0	1	**2**
4	1	0	0	0	0	1	0	0	0	1	1
5	0	1	0	0	1	0	0	0	1	0	**2**

Lift $[\![x = 4]\!]$ (again): $[\![x = 2]\!] + 2[\![x = 4]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$ row 3 now tight

Example: CUTLIFT on $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

- A tight row assigns variables such that the cut has LHS = RHS (e.g. row 2)
- A tight column is set to 1 in a tight row (e.g. $[\![x = 3]\!]$)
  - $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] + [\![\mathbf{x = 3}]\!] \leq 2$ would violate row 2!

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$	*RS*
1	0	1	0	0	1	0	0	1	0	0	1
2	0	0	1	0	0	1	0	0	1	0	**2**
3	0	0	0	1	0	0	1	0	0	1	**2**
4	1	0	0	0	0	1	0	0	0	1	**2**
5	0	1	0	0	1	0	0	0	1	0	**2**

Lift $[\![z = 1]\!]$: $[\![x = 2]\!] + 2[\![x = 4]\!] + [\![y = 2]\!] + [\![z = 1]\!] + [\![z = 2]\!] \leq 2$ row 1 now tight

Example: CUTLIFT on $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] \leq 2$

- A tight row assigns variables such that the cut has LHS = RHS (e.g. row 2)
- A tight column is set to 1 in a tight row (e.g. $[\![x = 3]\!]$)
  - $[\![x = 2]\!] + [\![y = 2]\!] + [\![z = 2]\!] + [\![\mathbf{x = 3}]\!] \leq 2$ would violate row 2!

	$[\![x{=}1]\!]$	$[\![x{=}2]\!]$	$[\![x{=}3]\!]$	$[\![x{=}4]\!]$	$[\![y{=}1]\!]$	$[\![y{=}2]\!]$	$[\![y{=}3]\!]$	$[\![z{=}1]\!]$	$[\![z{=}2]\!]$	$[\![z{=}3]\!]$	*RS*
1	0	1	0	0	1	0	0	1	0	0	**2**
2	0	0	1	0	0	1	0	0	1	0	**2**
3	0	0	0	1	0	0	1	0	0	1	**2**
4	1	0	0	0	0	1	0	0	0	1	**2**
5	0	1	0	0	1	0	0	0	1	0	**2**

Lift $[\![x = 1]\!]$: $[\![x = 1]\!] + [\![x = 2]\!] + 2[\![x = 4]\!] + [\![y = 2]\!] + [\![z = 1]\!] + [\![z = 2]\!] \leq 2$ all rows tight

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–
GENERATE	174		36	988.6	8639.7	1857.5

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–
GENERATE	174		36	988.6	8639.7	1857.5
FRACTIONAL	174		36	980.9	8639.7	3202.6

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–
GENERATE	174		36	988.6	8639.7	1857.5
FRACTIONAL	174		36	980.9	8639.7	3202.6
CUTLIFT	174		45	925.0	8639.7	847.1

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–
GENERATE	174		36	988.6	8639.7	1857.5
FRACTIONAL	174		36	980.9	8639.7	3202.6
CUTLIFT	174		45	925.0	8639.7	847.1
ALL	174		55	871.5	8639.7	1326.7

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Eager vs. Lazy – Scatter Plots

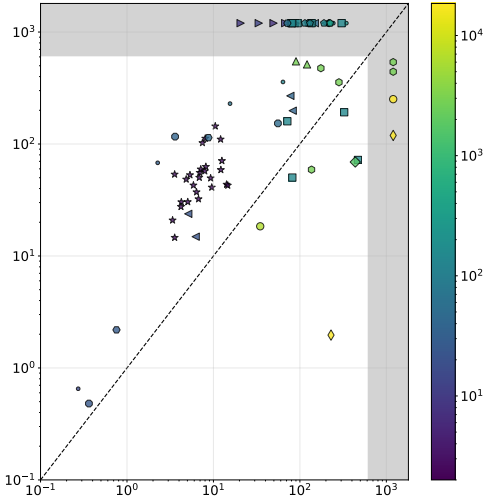


Figure: Solve time [s.]: Eager (top-left) – Lazy (bottom-right). Light colour = big table

Results – CSP (180 instances)

	POST	FEAS	SOLVED	time	constraints	cuts
INT	171		55	851.7	8249.3	–
BOOL	147		69	771.5	25670.4	–
MDD	154		67	780.8	38800.8	–
MDD-DOM	154		76	728.5	30884.2	–
GENERATE	174		36	988.6	8639.7	1857.5
FRACTIONAL	174		36	980.9	8639.7	3202.6
CUTLIFT	174		45	925.0	8639.7	847.1
ALL	174		55	871.5	8639.7	1326.7
HYBRID (1000)	174		79	717.5	29635.4	49.9
HYBRID (2000)	174		**80**	**715.6**	29912.8	49.9
HYBRID (3000)	**175**		78	716.7	30243.3	37.3

POST	#instances posted	FEAS	#feasible instances found
SOLVED	#instances solved	time	PAR2 avg [s]
constraints	avg #constraints added	cuts	avg #cuts generated

Conclusion & Future Work

Summary:

- Studied eager encodings and lazy cuts for table constraints
- Novel MDD-based encodings outperform standard methods
- Cutlifting and fractional cuts improve cut generation
- Hybrid approach combines the strengths of both paradigms

Future work:

- Better understanding of which tables suit which method (e.g. sparsity)
- Extend to *negative*, *smart*, *and reified tables*

Paper

Code

My website

References I

Gilles Audemard, Christophe Lecoutre, and Emmanuel Lonca.
Proceedings of the 2025 xcsp3 competition, 2025.
URL: https://arxiv.org/abs/2511.06918, arXiv:2511.06918.

Gleb Belov, Peter J. Stuckey, Guido Tack, and Mark Wallace.
Improved Linearization of Constraint Programming Models.
In Michel Rueher, editor, *Principles and Practice of Constraint Programming*, pages 49–65, Cham, 2016. Springer International Publishing.
doi:10.1007/978-3-319-44953-1_4.

Kenil C. K. Cheng and Roland H. C. Yap.
An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints.
Constraints An Int. J., 15(2):265–304, 2010.
URL: https://doi.org/10.1007/s10601-009-9087-y, doi:10.1007/S10601-009-9087-Y.

Tias Guns.
Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example.
In *Proceedings of the 18th workshop on Constraint Modelling and Reformulation at CP (Modref 2019)*, volume 19, 2019.

Gurobi Optimization, LLC.
Gurobi Optimizer Reference Manual, 2026.
URL: https://www.gurobi.com.

John N. Hooker.
Planning and scheduling by logic-based Benders decomposition.
Oper. Res., 55(3):588–602, 2007.
URL: https://doi.org/10.1287/opre.1060.0371, doi:10.1287/OPRE.1060.0371.

References II

N. Nethercote, P.J. Stuckey, R. Becket, S. Brand, G.J. Duck, and G. Tack.

Minizinc: Towards a standard CP modelling language.

In C. Bessiere, editor, *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, pages 529–543. Springer-Verlag, 2007.

Thierry Petit.

On constraint linear decompositions using mathematical variables.

In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pages 123–130. IEEE Computer Society, 2017.

doi:10.1109/ICTAI.2017.00030.