

Table Constraints for Integer Programming (CP'26)

JFPC'26

Hendrik Bierlee¹ Wout Piessens¹ Tias Guns¹ Peter J. Stuckey^{2,3}

¹KU Leuven, Belgium

²Monash University, Melbourne, Australia

³OPTIMA ITTC, Melbourne, Australia



Paper



Slides



KU LEUVEN

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 *Constraint Programming* (CP) models using *Integer Linear Programming* (ILP) solvers
- But there is a large performance gap between ILP and e.g. CP-SAT
 - Because globals (e.g. table constraints) are not linearized as well
 - Because we do not use the full ILP toolset

Running example: `table([x, y, z], T)` with enforces \hat{x} to take one row of T :

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

Simple 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:

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

For example, if we choose row 1, then $r_1 = 1$, so $x = 2 \cdot r_1 + 0 + \dots + 0 = 2$

Booleanized Encoding (BOOL) [8]

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

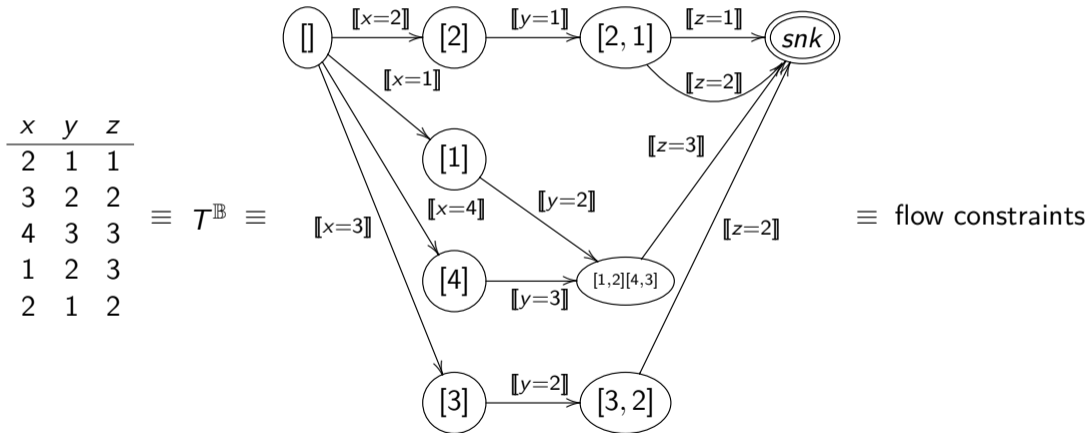
$$\sum_{j \in D(x)} \llbracket x = j \rrbracket = 1, \quad \sum_{j \in D(x)} j \cdot \llbracket x = j \rrbracket = x$$

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

x	y	z		$\llbracket x=1 \rrbracket$	$\llbracket x=2 \rrbracket$	$\llbracket x=3 \rrbracket$	$\llbracket x=4 \rrbracket$	$\llbracket y=1 \rrbracket$	$\llbracket y=2 \rrbracket$	$\llbracket y=3 \rrbracket$	$\llbracket z=1 \rrbracket$	$\llbracket z=2 \rrbracket$	$\llbracket z=3 \rrbracket$
2	1	1		0	1	0	0	1	0	0	1	0	0
3	2	2		0	0	1	0	0	1	0	0	1	0
4	3	3	\equiv	0	0	0	1	0	0	1	0	0	1
1	2	3		1	0	0	0	0	1	0	0	0	1
2	1	2		0	1	0	0	1	0	0	0	1	0

MDD-based Flow Encoding (MDD)

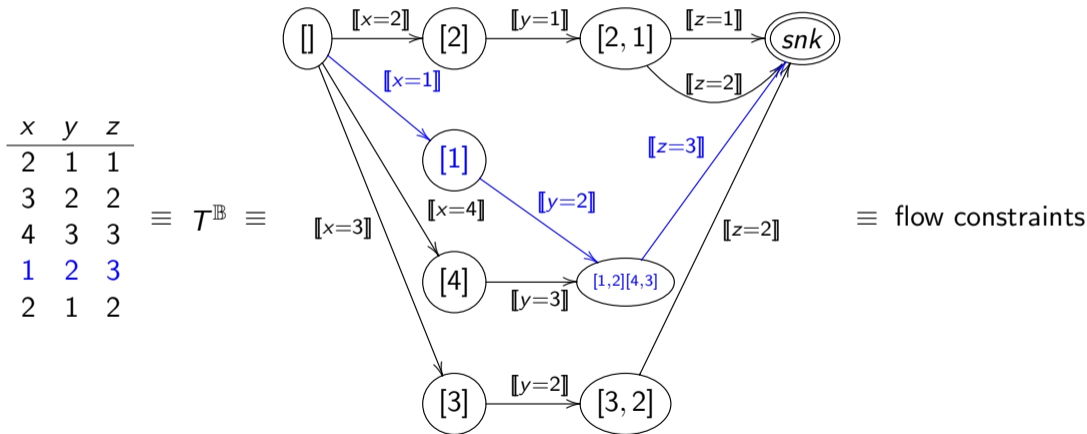
We can reformulate $T^{\mathbb{B}}$ as a reduced Multi-valued Decision Diagram (MDD) [3]



The MDD can be **exponentially smaller**, the flow constraints are **totally unimodular**

MDD-based Flow Encoding (MDD)

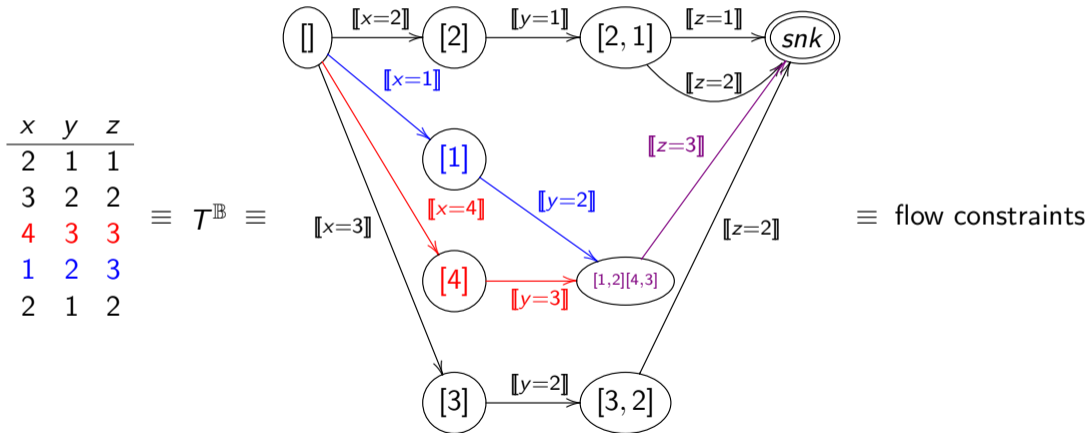
We can reformulate $T^{\mathbb{B}}$ as a reduced Multi-valued Decision Diagram (MDD) [3]



The MDD can be **exponentially smaller**, the flow constraints are **totally unimodular**

MDD-based Flow Encoding (MDD)

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



The MDD can be **exponentially smaller**, the flow constraints are **totally unimodular**

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	–

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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	–

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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	–

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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	–

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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. $\hat{v} = [2, 2, 2]$)

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** (violating \hat{v} but not the table)
 - E.g. $\llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$
 - Note this cut is a logical clause $\neg \llbracket x = 2 \rrbracket \vee \neg \llbracket y = 2 \rrbracket \vee \neg \llbracket z = 2 \rrbracket$

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** (violating \hat{v} but not the table)
 - E.g. $\llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$
 - Note this cut is a logical clause $\neg\llbracket x = 2 \rrbracket \vee \neg\llbracket y = 2 \rrbracket \vee \neg\llbracket z = 2 \rrbracket$
- 3 Repeat until \hat{v} satisfies all 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** (violating \hat{v} but not the table)
 - E.g. $\llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$
 - Note this cut is a logical clause $\neg\llbracket x = 2 \rrbracket \vee \neg\llbracket y = 2 \rrbracket \vee \neg\llbracket z = 2 \rrbracket$
- 3 Repeat until \hat{v} satisfies all tables

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

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 $[[x = 2]]$: $[[x = 2]] \leq 0$ (rows 1,5 still violated)

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

	$\llbracket x=1 \rrbracket$	$\llbracket x=2 \rrbracket$	$\llbracket x=3 \rrbracket$	$\llbracket x=4 \rrbracket$	$\llbracket y=1 \rrbracket$	$\llbracket y=2 \rrbracket$	$\llbracket y=3 \rrbracket$	$\llbracket z=1 \rrbracket$	$\llbracket z=2 \rrbracket$	$\llbracket z=3 \rrbracket$
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 $\llbracket y = 2 \rrbracket$: $\llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket \leq 1 \quad \checkmark$

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

$$\text{Input: } \llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

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

$$\text{Input: } \llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

$$\text{Output: } \llbracket x = 1 \rrbracket + \llbracket x = 2 \rrbracket + 2\llbracket x = 4 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 1 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

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

$$\text{Input: } \llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

$$\text{Output: } \llbracket x = 1 \rrbracket + \llbracket x = 2 \rrbracket + 2\llbracket x = 4 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 1 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

FRACTIONAL Generate cuts from **fractional** LP solutions

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

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

$$\text{Input: } \llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

$$\text{Output: } \llbracket x = 1 \rrbracket + \llbracket x = 2 \rrbracket + 2\llbracket x = 4 \rrbracket + \llbracket y = 2 \rrbracket + \llbracket z = 1 \rrbracket + \llbracket z = 2 \rrbracket \leq 2$$

FRACTIONAL Generate cuts from **fractional** LP solutions

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

$$\text{Output: } \llbracket x = 2 \rrbracket + \llbracket y = 2 \rrbracket \leq 1$$

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

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

Eager vs. Lazy – Scatter Plots

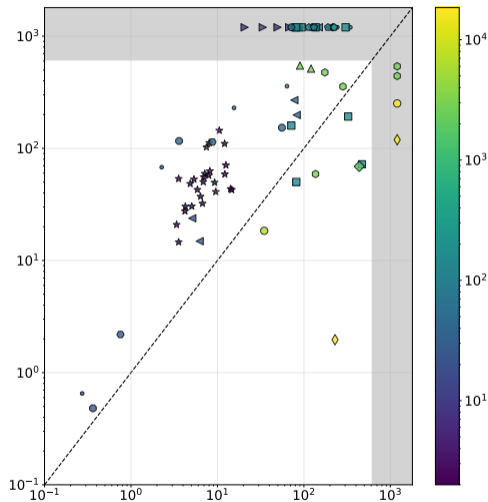


Figure: Solve time [s.]: Eager (top-left) – Lazy (bottom-right). Light colour = big table

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

pst = posted, sat = feasible, sol = solved. Time = PAR2 avg [s.]

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



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.

 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.