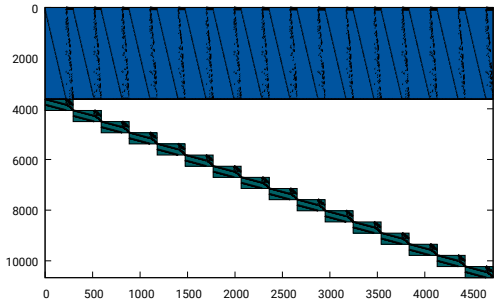
Modular Detection of Model Structure in Integer Programming

Michael Bastubbe & Marco Lübbecke

SCIP Workshop 2018
March 8th, 2018

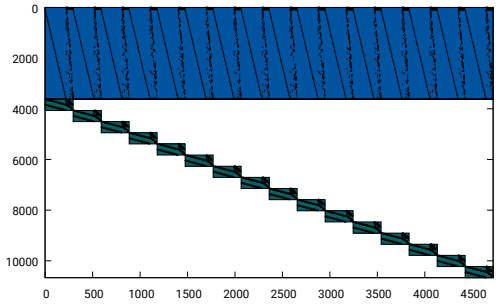
Brief overview: Dantzig-Wolfe Reformulation

- Partition constraints of a MILP:
  - Master problem $Ax \geq b$
  - Pricing problems $D_i x_i \geq d_i$



Brief overview: Dantzig-Wolfe Reformulation

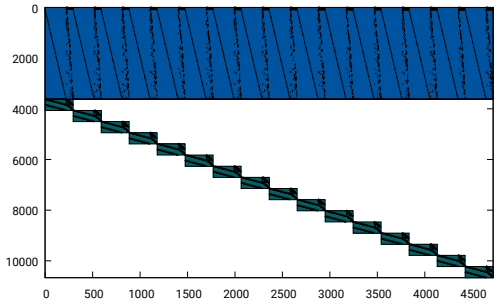
- Partition constraints of a MILP:
  - Master problem $Ax \geq b$
  - Pricing problems $D_i x_i \geq d_i$



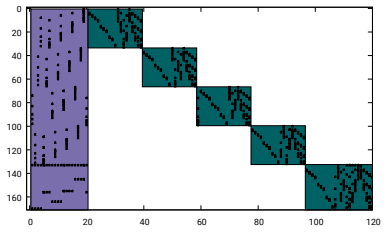
- Reformulate master problem: combination of extreme points and rays ("convexification") of pricing problems
- Solve master problem with Branch-Price-and-Cut

Brief overview: Dantzig-Wolfe Reformulation

- Partition constraints of a MILP:
  - Master problem $Ax \geq b$
  - Pricing problems $D_i x_i \geq d_i$



- Reformulate master problem: combination of extreme points and rays ("convexification") of pricing problems
- Solve master problem with Branch-Price-and-Cut
- Allow linking variables (each is copied for every linked block)

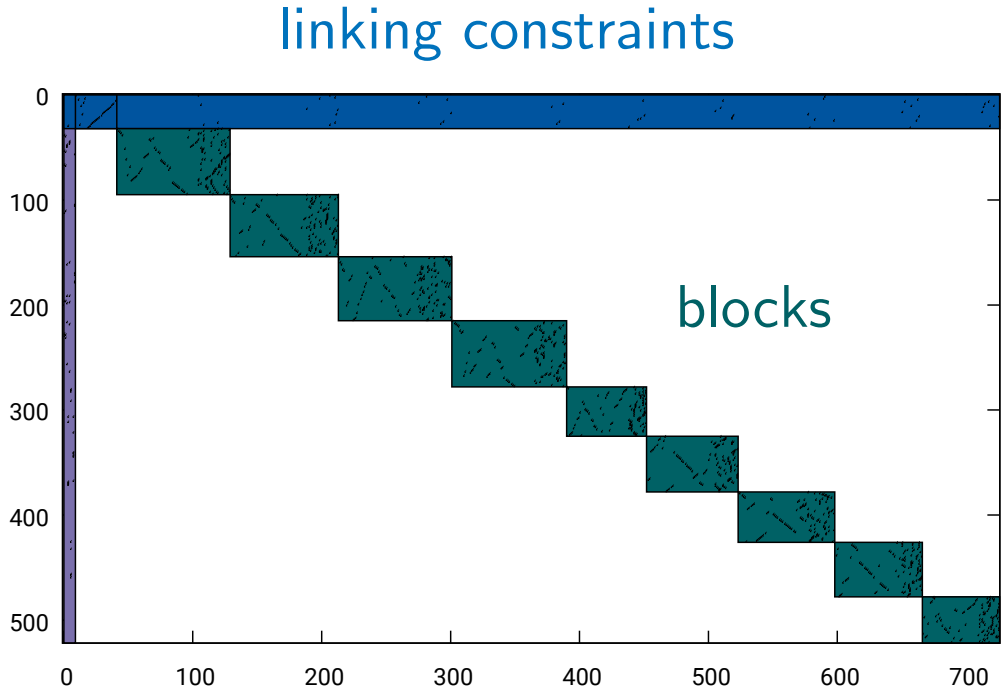


Outline: What is new?

- modular detection scheme
- constraint and variable classification
- consider information from original problem
- guess meaningful number of blocks
- pluggable score to evaluate decompositions
- user interaction

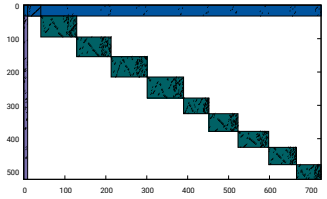
What are we looking for?

find:



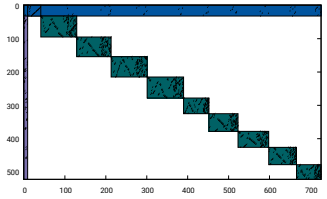
Modular scheme

find:



Modular scheme

find:

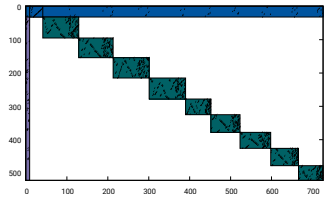


past:

- each detector is called once; returns set of decompositions
- no interaction between detectors
- “connected” detector implemented a mix of two ideas

Modular scheme

find:



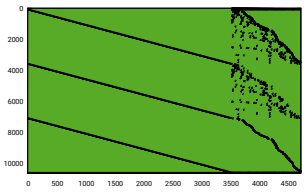
past:

- each detector is called once; returns set of decompositions
- no interaction between detectors
- "connected" detector implemented a mix of two ideas

recent development:

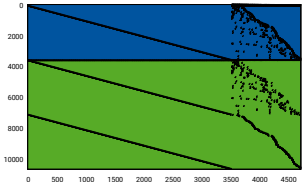
- each detector persues one atomic idea
- mixtures are handled in a general scheme
- new concept: partially decided decomposition [aka partials]
- partials are refined round wise

Example for partially refinement



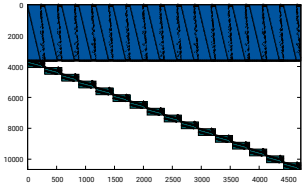
id 2; nB 0; maxW$\geq$ 0.00

consclass_constypes:setpartitioning-linear-logicor-knapsack



id 25; nB 0; maxW$\geq$ 0.34

connected



id 122; nB 16; maxW 0.38

Overview: What is new?

recent dev:

- new concept: partially decided decomposition [aka partials]
- partials are refined round wise
- detectors can implement three different callbacks:

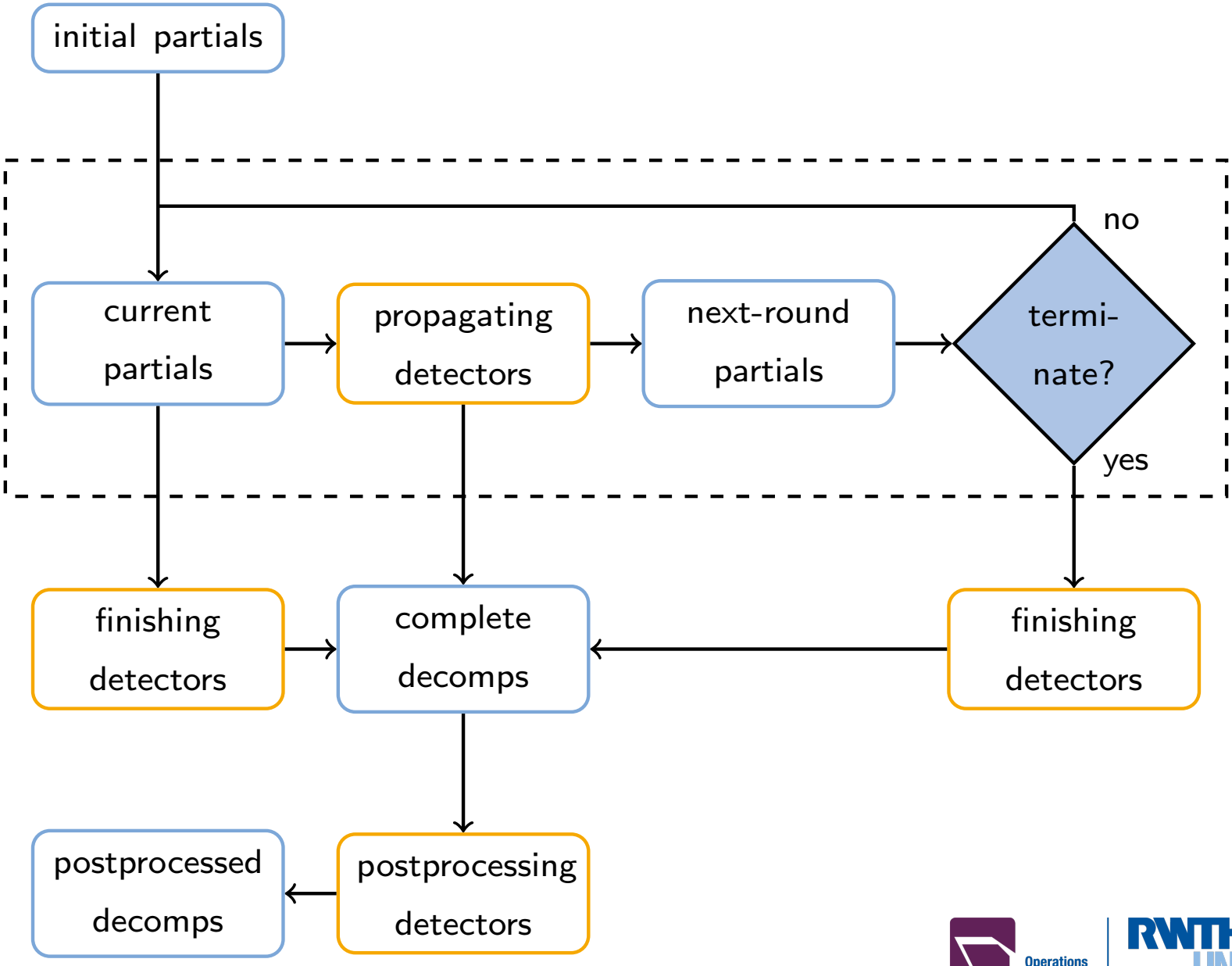
Overview: What is new?

recent dev:

- new concept: partially decided decomposition [aka partials]
- partials are refined round wise
- detectors can implement three different callbacks:

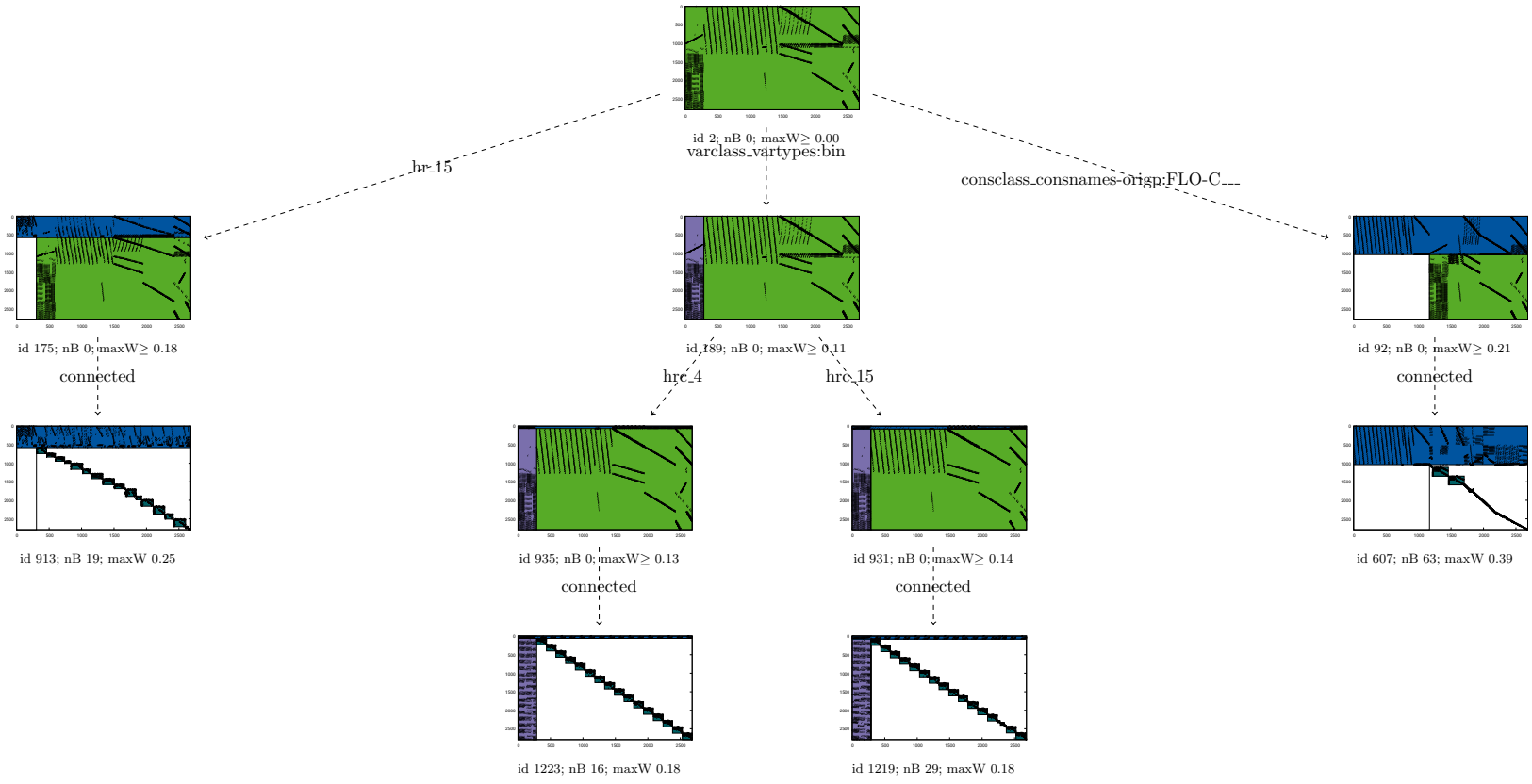
callback	input	output	revoke decisions?
propagate	partial	mix of complete/partials	no
finish	partial	set of complete	no
postprocess	complete	set of complete	yes

Detection overview: detection loop



Modular Detection: Example *b2c1s1*

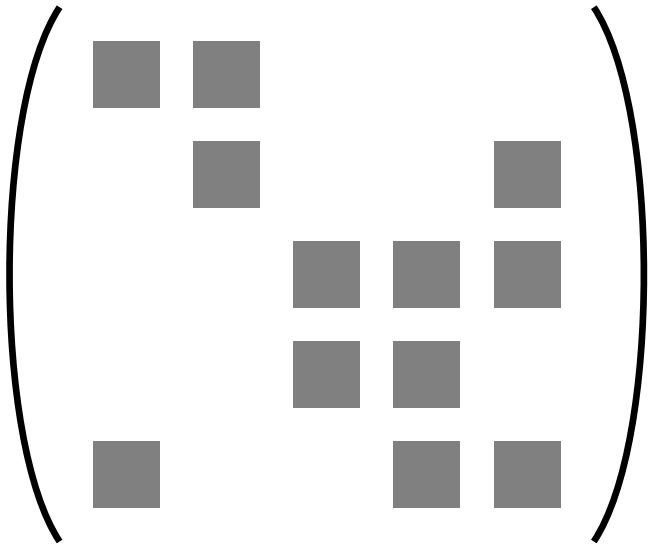
- resulting decompositions can have common ancestors



Operations Research | RWTH AACHEN UNIVERSITY

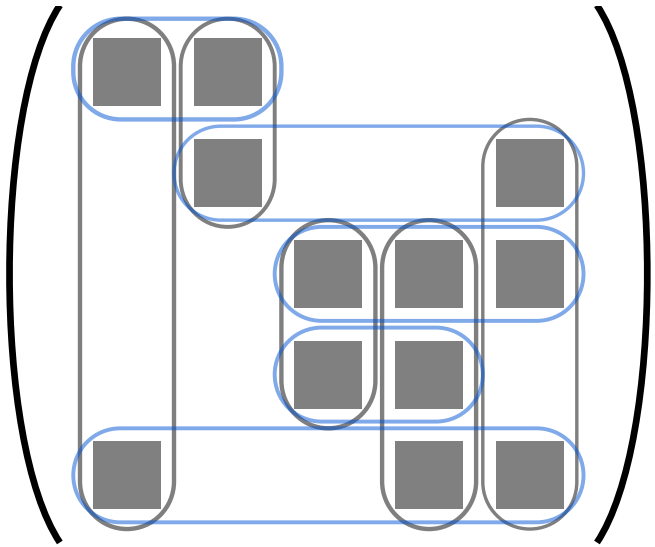
Example Propagating Detector: hrcgpartition

- Row-column-net hypergraph used in Bergner et al. (2015)
- Every nonzero entry a_{ij} is a vertex
- Every row and column is a hyperedge
- Solve balan. $\min$-k way cut problem
- Hyperedges between partitions in border



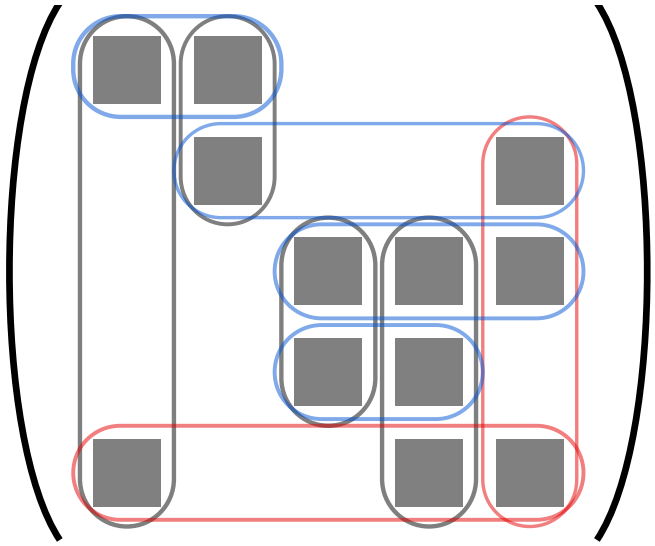
Example Propagating Detector: hrcgpartition

- Row-column-net hypergraph used in Bergner et al. (2015)
- Every nonzero entry a_{ij} is a vertex
- Every row and column is a hyperedge
- Solve balan. $\min$-k way cut problem
- Hyperedges between partitions in border



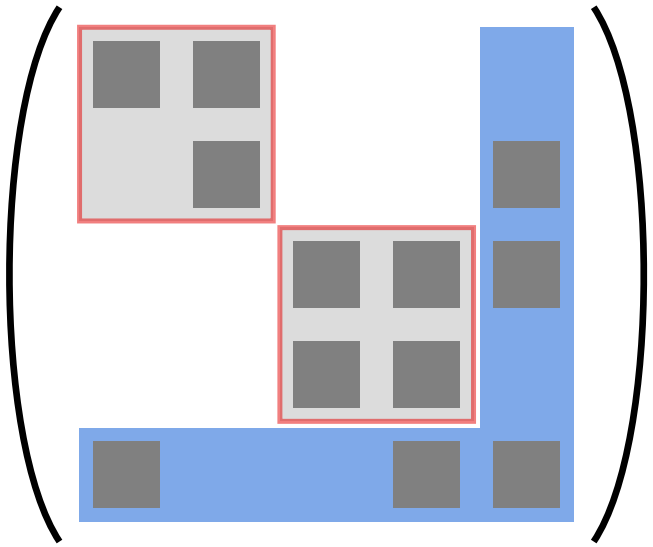
Example Propagating Detector: hrcgpartition

- Row-column-net hypergraph used in Bergner et al. (2015)
- Every nonzero entry a_{ij} is a vertex
- Every row and column is a hyperedge
- Solve balan. $\min$-k way cut problem
- Hyperedges between partitions in border



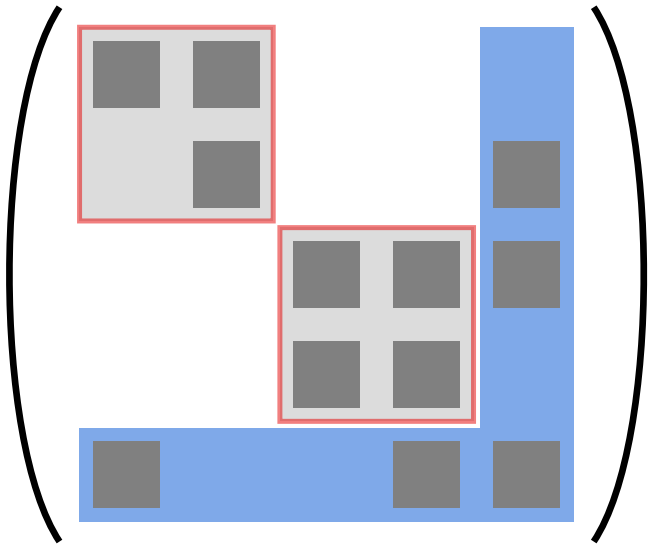
Example Propagating Detector: hrcgpartition

- Row-column-net hypergraph used in Bergner et al. (2015)
- Every nonzero entry a_{ij} is a vertex
- Every row and column is a hyperedge
- Solve balan. $\min$-k way cut problem
- Hyperedges between partitions in border



Example Propagating Detector: hrcgpartition

- Row-column-net hypergraph used in Bergner et al. (2015)
- Every nonzero entry a_{ij} is a vertex
- Every row and column is a hyperedge
- Solve balan. $\min$-k way cut problem
- Hyperedges between partitions in border

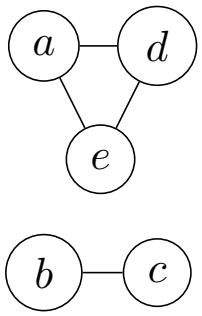


- Note:
  - returns complete and partial (only master and linking is assigned)
  - k is not known
  - problem is $\mathcal{NP}$-hard

Example finishing detector: Connected

- connected component in row-adjacency graph yields a block

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 1 & & & & & & 1 \\ & 1 & & 1 & & 1 & \\ & & & 1 & 1 & 1 & \\ 1 & & 1 & & & & \\ & & 1 & & & & 1 \end{pmatrix} \end{array}$$



$$\begin{array}{c} \\ a \\ d \\ e \\ b \\ c \end{array}\begin{array}{c} \begin{array}{ccccccc} 1 & 3 & 7 & 2 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & & 1 & & & & \\ 1 & 1 & & & & & \\ & 1 & 1 & & & & \\ & & & 1 & 1 & & 1 \\ & & & & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Example postprocessing detector

- simple idea: check for each master constraint if it can be assigned to exactly one block

Constraint Classification: Binpacking example

$$
\begin{aligned}
\text{min } & y_1 + y_2 + y_3 \\
\text{s.t. } & -100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} \leq 0 \\
& -100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} \leq 0 \\
& -100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} \leq 0 \\
& x_{11} + x_{12} + x_{13} \geq 1 \\
& x_{21} + x_{22} + x_{23} \geq 1 \\
& x_{31} + x_{32} + x_{33} \geq 1 \\
& x_{41} + x_{42} + x_{43} \geq 1 \\
& \vdots
\end{aligned}
$$

Constraint Classification: Binpacking example

$$
\begin{aligned}
\min\ & y_1 + y_2 + y_3 \\
\text{s.t.}\ & -100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} && \leq && 0 \\
& -100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} && \leq && 0 \\
& -100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} && \leq && 0 \\
& x_{11} + x_{12} + x_{13} && \geq && 1 \\
& x_{21} + x_{22} + x_{23} && \geq && 1 \\
& x_{31} + x_{32} + x_{33} && \geq && 1 \\
& x_{41} + x_{42} + x_{43} && \geq && 1 \\
& \vdots
\end{aligned}
$$

Constraint Classification: Binpacking example

$$
\begin{aligned}
\min\ & y_1 + y_2 + y_3 \\
\text{s.t.}\ & -100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} \leq 0 \\
& -100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} \leq 0 \\
& -100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} \leq 0 \\
& x_{11} + x_{12} + x_{13} \geq 1 \\
& x_{21} + x_{22} + x_{23} \geq 1 \\
& x_{31} + x_{32} + x_{33} \geq 1 \\
& x_{41} + x_{42} + x_{43} \geq 1 \\
& \vdots
\end{aligned}
$$

Constraint Classification: Binpacking example

$$\begin{aligned}
\min\ & y_1 + y_2 + y_3 \\
\text{s.t.}\ & -100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} \leq 0 \\
& -100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} \leq 0 \\
& -100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} \leq 0 \\
& x_{11} + x_{12} + x_{13} \geq 1 \\
& x_{21} + x_{22} + x_{23} \geq 1 \\
& x_{31} + x_{32} + x_{33} \geq 1 \\
& x_{41} + x_{42} + x_{43} \geq 1 \\
& \vdots
\end{aligned}$$

Constraint Classification: Binpacking example

$$\min\ y_1 + y_2 + y_3$$

$$\text{s.t.}\quad
\begin{array}{lcl}
-100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} & \leq & 0 \\
-100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} & \leq & 0 \\
-100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} & \leq & 0 \\
x_{11} + x_{12} + x_{13} & \geq & 1 \\
x_{21} + x_{22} + x_{23} & \geq & 1 \\
x_{31} + x_{32} + x_{33} & \geq & 1 \\
x_{41} + x_{42} + x_{43} & \geq & 1 \\
\vdots & &
\end{array}$$

knapsack

Constraint Classification: Binpacking example

$$\min\ y_1 + y_2 + y_3$$

$$\text{s.t.}\quad \begin{array}{lcl}
-100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} & \leq & 0 \\
-100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} & \leq & 0 \\
-100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} & \leq & 0 \\
x_{11} + x_{12} + x_{13} & \geq & 1 \\
x_{21} + x_{22} + x_{23} & \geq & 1 \\
x_{31} + x_{32} + x_{33} & \geq & 1 \\
x_{41} + x_{42} + x_{43} & \geq & 1 \\
\vdots & &
\end{array}$$

knapsack

setcover

Constraint Classification: Binpacking example

$$\begin{aligned}
\min\ & y_1 + y_2 + y_3 \\
\text{s.t.}\ & -100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} \leq 0 \\
& -100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} \leq 0 \\
& -100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} \leq 0 \\
& x_{11} + x_{12} + x_{13} \geq 1 \\
& x_{21} + x_{22} + x_{23} \geq 1 \\
& x_{31} + x_{32} + x_{33} \geq 1 \\
& x_{41} + x_{42} + x_{43} \geq 1 \\
& \vdots
\end{aligned}$$

5 nonzeros

Constraint Classification: Binpacking example

$$\min\ y_1 + y_2 + y_3$$

$$\text{s.t.}\quad \begin{array}{lcl}
-100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} & \leq & 0 \\
-100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} & \leq & 0 \\
-100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} & \leq & 0 \\
x_{11} + x_{12} + x_{13} & \geq & 1 \\
x_{21} + x_{22} + x_{23} & \geq & 1 \\
x_{31} + x_{32} + x_{33} & \geq & 1 \\
x_{41} + x_{42} + x_{43} & \geq & 1 \\
\vdots & &
\end{array}$$

5 nonzeros

3 nonzeros

Constraint Classification: Binpacking example

$$
\begin{aligned}
\min\ & y_1 + y_2 + y_3 \\
\text{s.t.}\ & -100y_1 + 54x_{11} + 33x_{21} + 34x_{31} + 72x_{41} \leq 0 \\
& -100y_2 + 54x_{12} + 33x_{22} + 34x_{32} + 72x_{42} \leq 0 \\
& -100y_3 + 54x_{13} + 33x_{23} + 34x_{33} + 72x_{43} \leq 0 \\
& x_{11} + x_{12} + x_{13} \geq 1 \\
& x_{21} + x_{22} + x_{23} \geq 1 \\
& x_{31} + x_{32} + x_{33} \geq 1 \\
& x_{41} + x_{42} + x_{43} \geq 1 \\
& \vdots
\end{aligned}
$$

5 nonzeros

3 nonzeros

- before detection loop: classify constraints
- several classifications (aka classifier) possible
- during detection loop: a propagating detector assigns combinations of classes to the master for each classifier

Constraint classification

- before detection loop: classify constraints
- several classifications (aka classifier) possible
- during detection loop: a propagating detector assigns combinations of classes to the master for each classifier

Constraint classification

- before detection loop: classify constraints
- several classifications (aka classifier) possible
- during detection loop: a propagating detector assigns combinations of classes to the master for each classifier

- several constraint classifiers:
  - same number of nonzero entries
  - type found by *SCIP*
  - type according to *MIPLIB2010*
  - names differ only by digits
  - ...

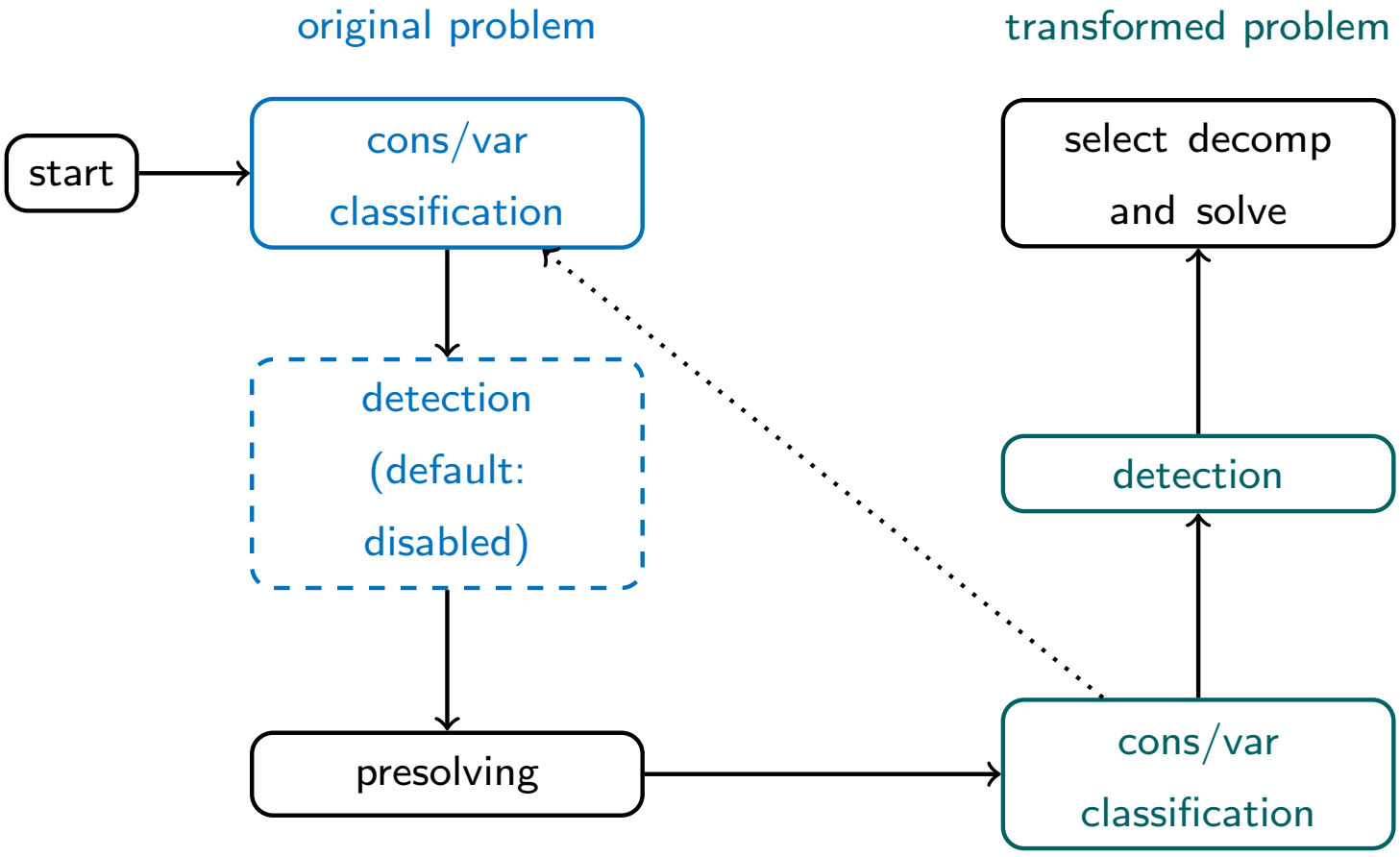
Variable classification

- Several variable classifiers:
  - type according to *SCIP*
  - objective function coefficient
  - sign of objective function coefficient
  - ...

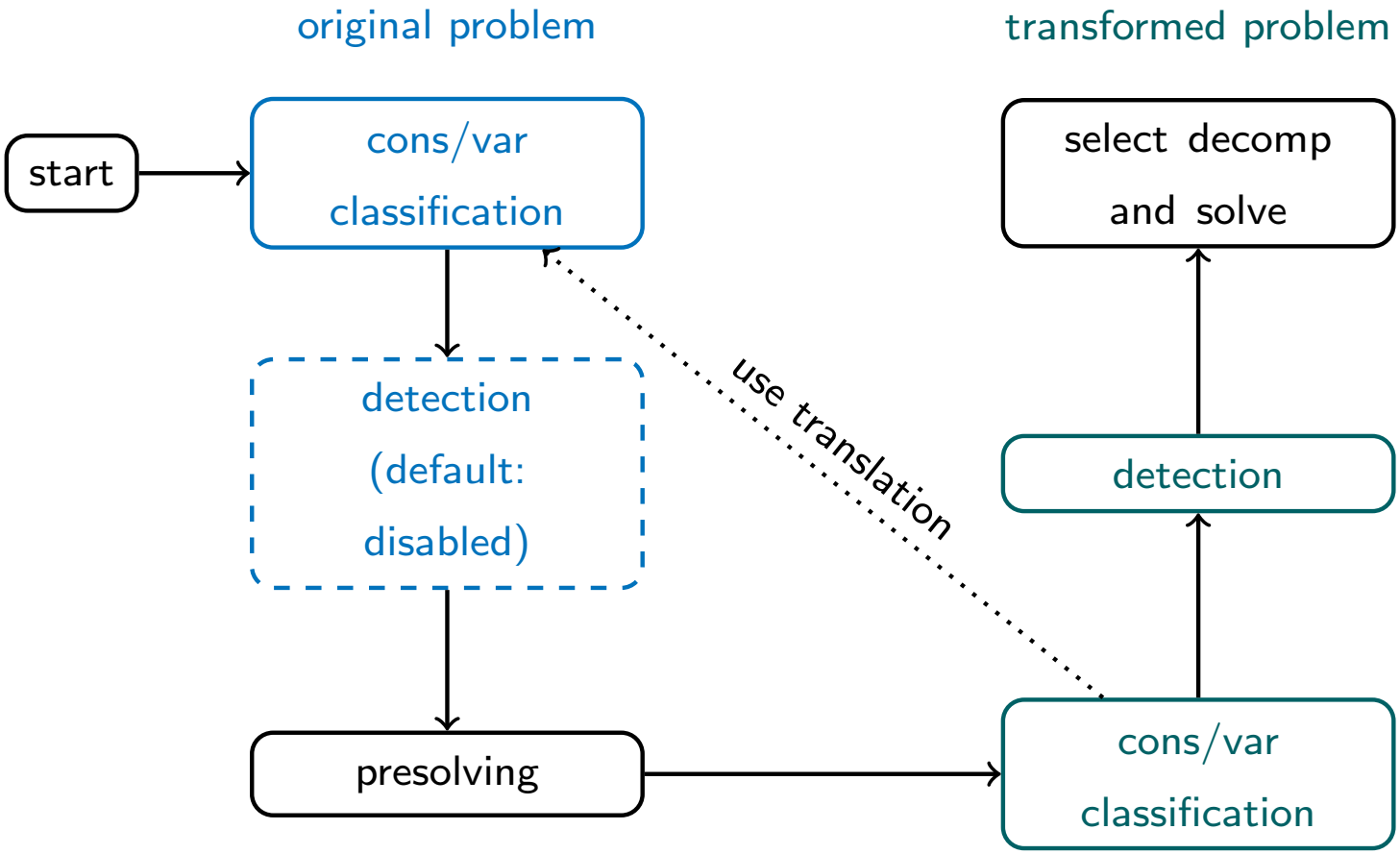
Variable classification

- Several variable classifiers:
  - type according to *SCIP*
  - objective function coefficient
  - sign of objective function coefficient
  - ...
- Set every combination of classes as linking vars and master-only variables

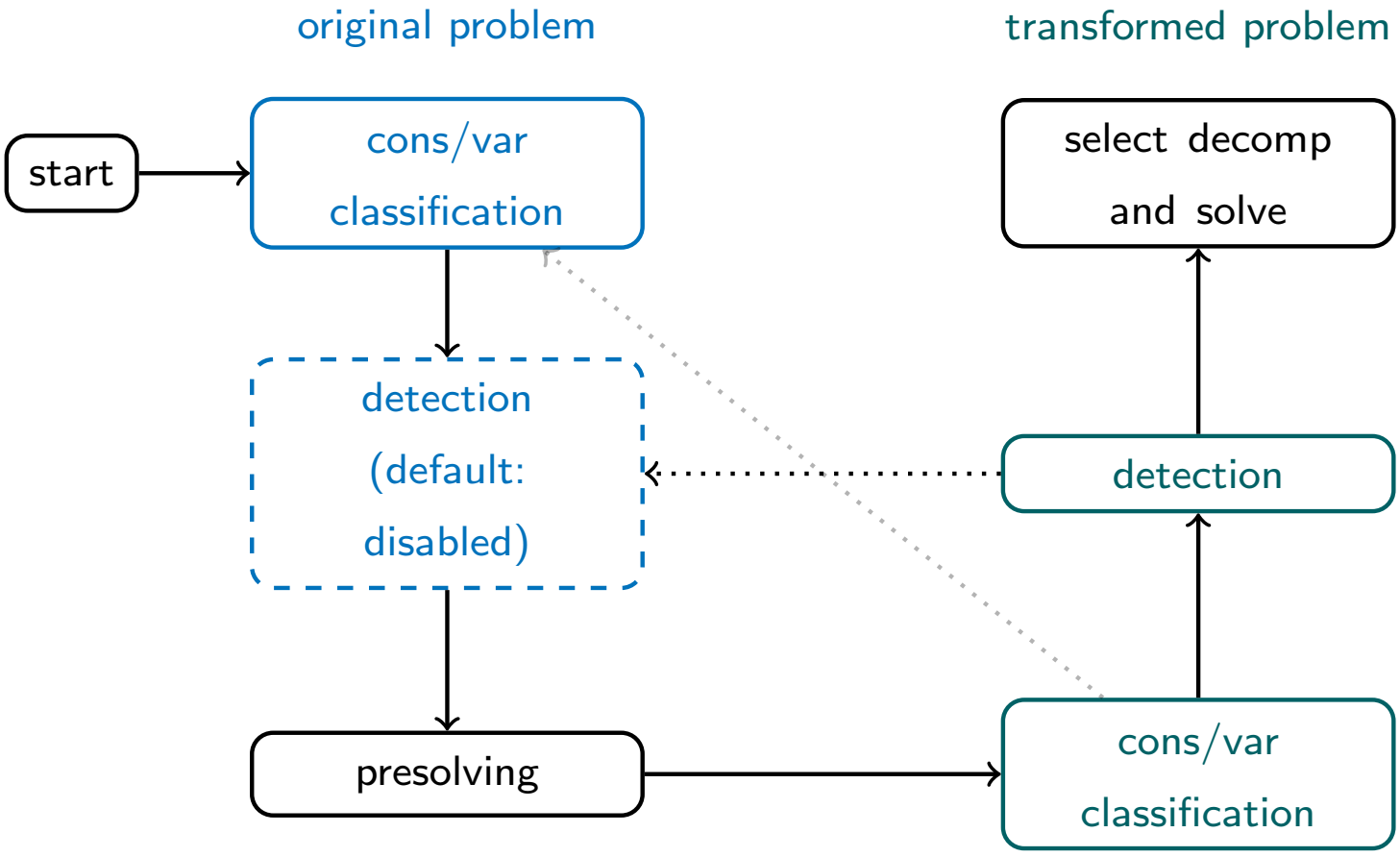
Information from original problem can be preserved



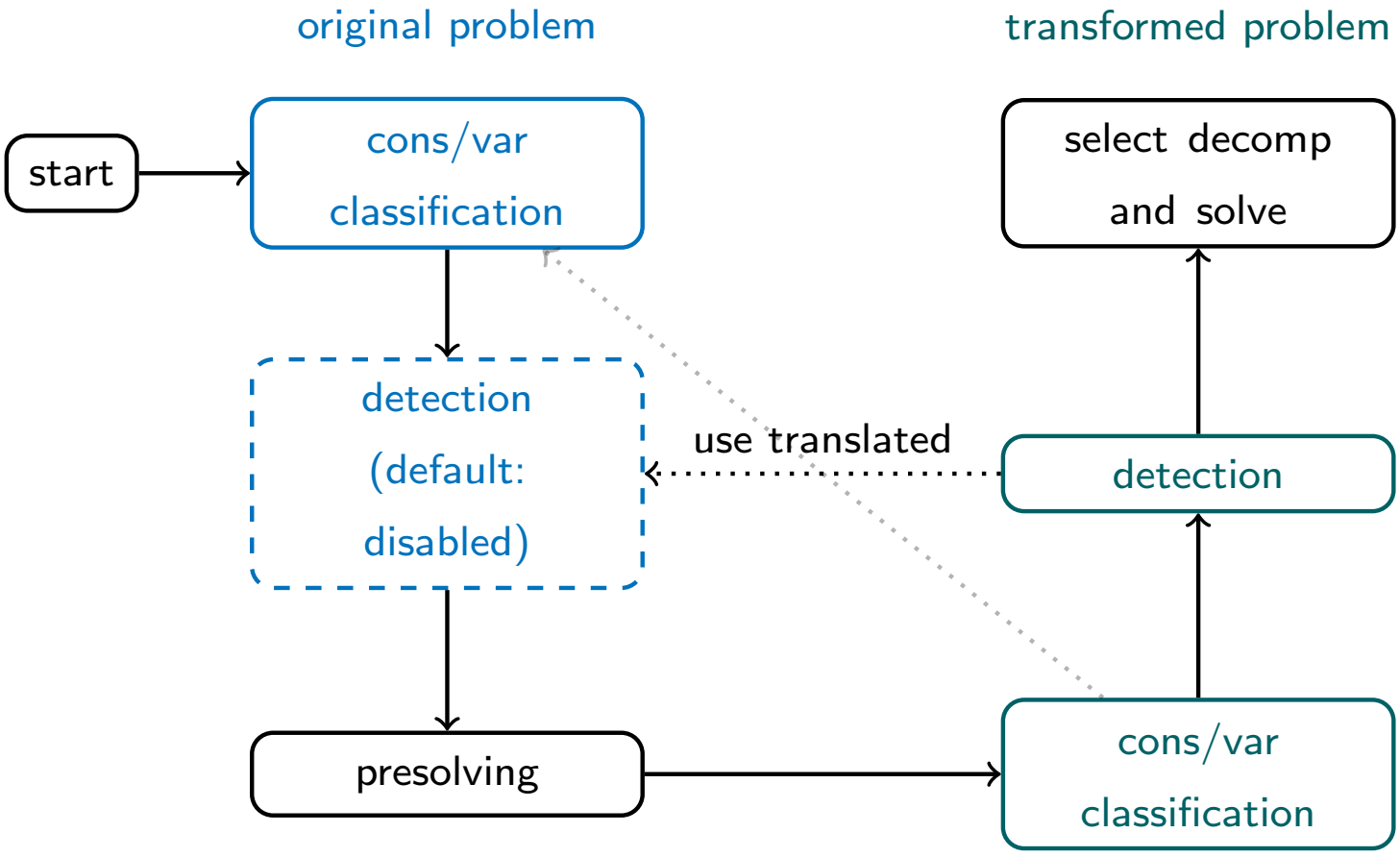
Information from original problem can be preserved



Information from original problem can be preserved



Information from original problem can be preserved



Guess meaningful number of blocks

- motivation: detectors relying on graph partitioning need number of block as input

Guess meaningful number of blocks

- motivation: detectors relying on graph partitioning need number of block as input
- past: try $2, \ldots, 20$

Guess meaningful number of blocks

- motivation: detectors relying on graph partitioning need number of block as input
- past: try $2, \ldots, 20$
- idea: use classification information to make educated guess
- originally: CPAIOR13 talk related to Wang and Ralphs (2013) proposed a frequency table/histogram (this is under the assumption that blocks are identical): count how many rows have 1, 2, 3, ... many non-zeros, e.g., for atm**20**-100:

# of Nonzeros	2	11	12	13	24	40	100
# of Rows	2220	**20**	**20**	2	1998	100	**20**

Guess meaningful number of blocks

- motivation: detectors relying on graph partitioning need number of block as input
- past: try $2, \ldots, 20$
- idea: use classification information to make educated guess
- originally: CPAIOR13 talk related to Wang and Ralphs (2013) proposed a frequency table/histogram (this is under the assumption that blocks are identical): count how many rows have 1, 2, 3, ... many non-zeros, e.g., for atm**20**-100:

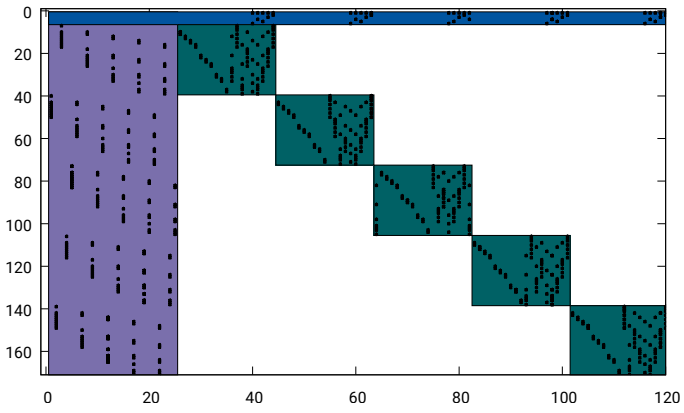
# of Nonzeros	2	11	12	13	24	40	100
# of Rows	2220	**20**	**20**	2	1998	100	**20**

- we calculate greatest common divisors of constraint/variable classes for all classifiers
- thus get a voting for the number of blocks

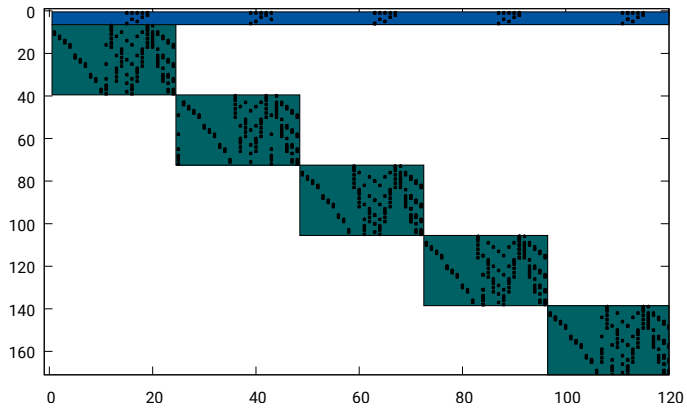
Pluggable scores

score:

- maximize fraction of non-colored area:



score of 0.611

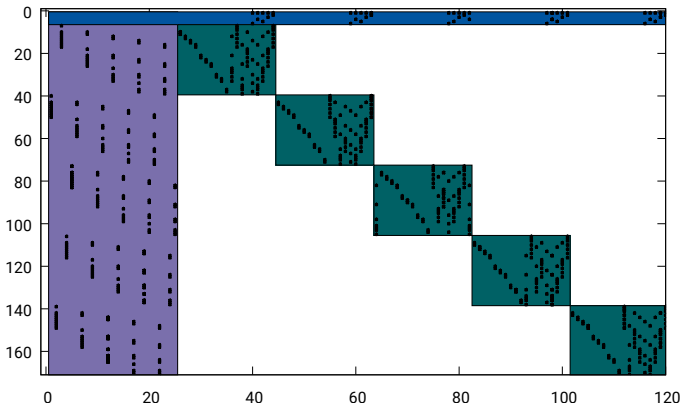


score of 0.77

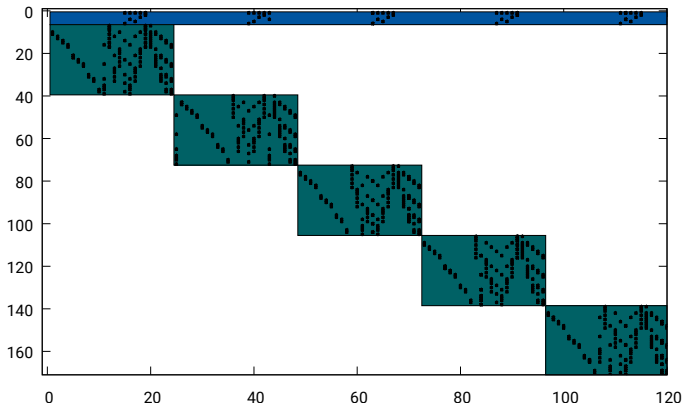
Pluggable scores

score:

- maximize fraction of non-colored area:



score of 0.611



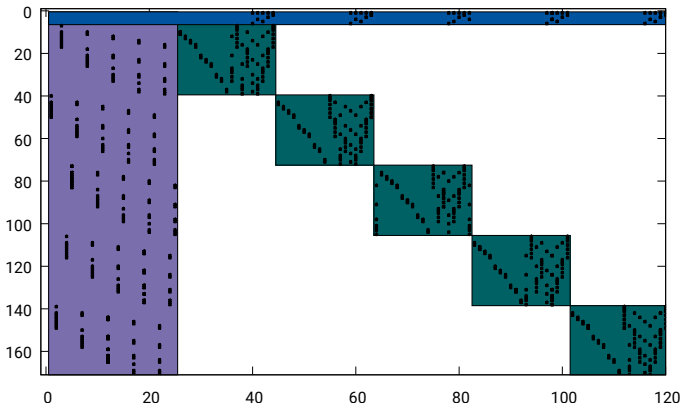
score of 0.77

- currently: 7 different scores, combinations of:
  - consider copied linking vars
  - consider if master consists only of several constraint types
  - consider aggregation information

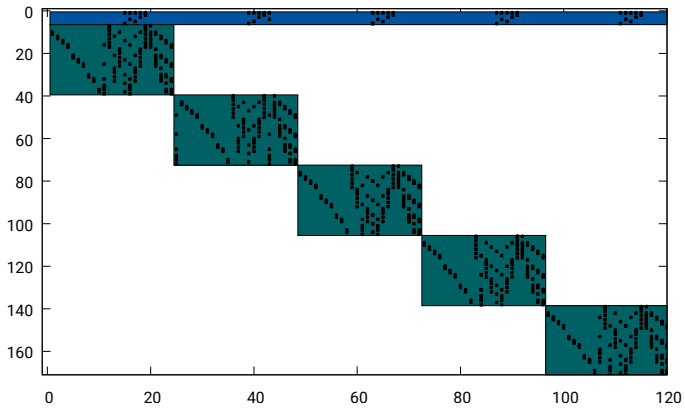
Pluggable scores

score:

- maximize fraction of non-colored area:



score of 0.611



score of 0.77

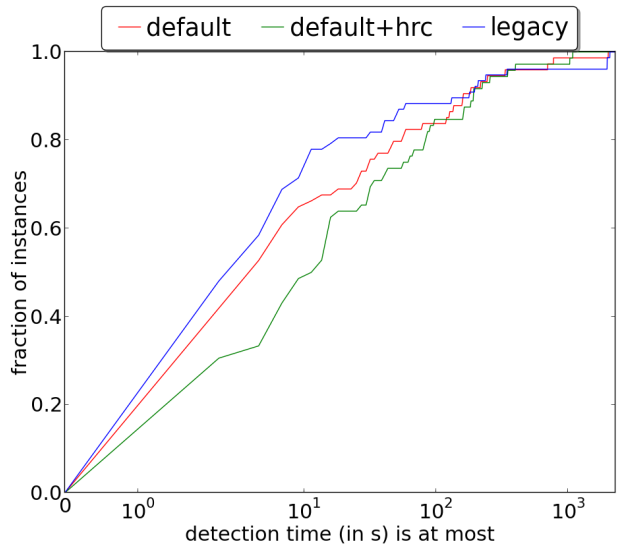
- currently: 7 different scores, combinations of:
  - consider copied linking vars
  - consider if master consists only of several constraint types
  - consider aggregation information
- wip: "strong detection" score (expensive: on demand)

User interface

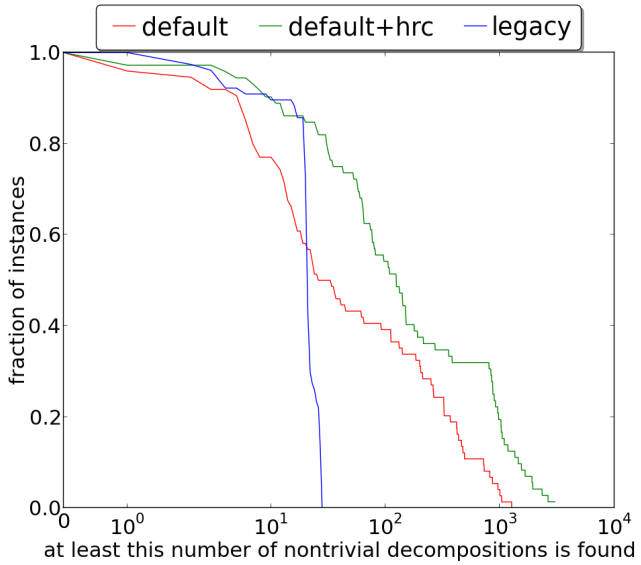
- partial decompositions can be given by user
- toolbox (in development) to call single detectors or assign specific conss or vars by name (regex)
- user can give candidates number of blocks
- browse found decompositions inside gcg and select interesting (for visualization, solving, writing)

Preliminary computational results

- testset: benchmark subset of miplib2010
- three settings: *default*, *default+hrcgpartition*, *legacy*



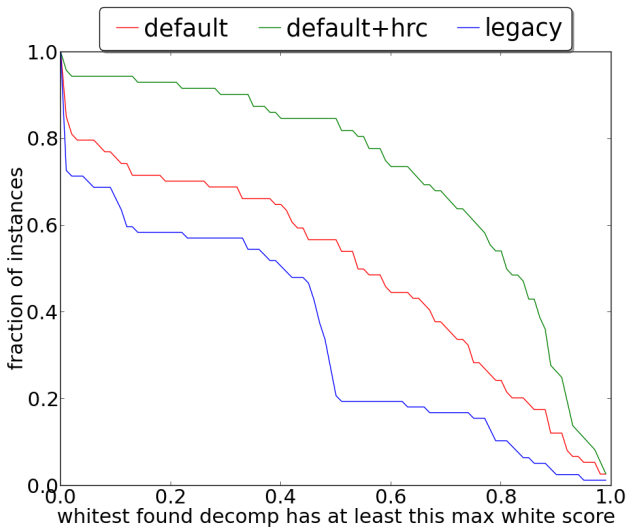
detection time



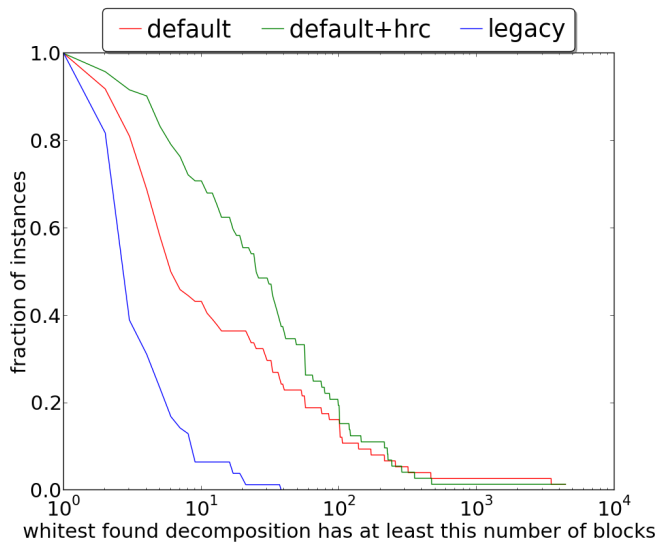
number of found decompositions

Preliminary computational results

- testset: benchmark subset of miplib2010
- three settings: *default*, *default+hrcgpartition*, *legacy*



max white score



nblocks of whitest

Outlook

- tests with termination conditions
- massive tests on resulting reformulations
- test strong decomposition score
- finish toolbox

References

Bergner, M., Caprara, A., Ceselli, A., Furini, F., Lübbecke, M., Malaguti, E., and Traversi, E. (2015). Automatic Dantzig-Wolfe reformulation of mixed integer programs. *Math. Prog.*, 149(1–2):391–424.

Wang, J. and Ralphs, T. (2013). Computational experience with hypergraph-based methods for automatic decomposition in discrete optimization. In Gomes, C. and Sellmann, M., editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 394–402. Springer, Berlin, Heidelberg.