

Component Bound Branching in a Branch-and-Price Framework

Til Mohr

`til.mohr@rwth-aachen.de`

Master thesis at the
Chair of Operations Research @ RWTH Aachen

`www.or.rwth-aachen.de`

September 19, 2024

Dantzig-Wolfe Reformulation for *IPs*

- consider the following bounded integer program (*IP*):

$$\begin{aligned} z_{IP}^* = \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s. t.} \quad & \mathbf{Ax} \geq \mathbf{b} \quad [\pi_b] \\ & \mathbf{Dx} \geq \mathbf{d} \quad [\pi_d] \\ & \mathbf{x} \in \mathbb{Z}_+^n \end{aligned}$$

- we want to solve this *IP* using Column Generation (CG)

Dantzig-Wolfe Reformulation for *IPs*

- consider the following bounded integer program (*IP*):

$$\begin{aligned} z_{IP}^* = \min \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s. t.} \quad & \mathbf{Ax} \geq \mathbf{b} \quad [\pi_b] \\ & \mathbf{Dx} \geq \mathbf{d} \quad [\pi_d] \\ & \mathbf{x} \in \mathbb{Z}_+^n \end{aligned}$$

- we want to solve this *IP* using Column Generation (CG)
→ Apply Dantzig-Wolfe Reformulation

Dantzig-Wolfe Reformulation for IPs

- reformulation using discretization yields the master problem (*MP*):

$$\begin{aligned} z_{MP}^* = \min & \quad \sum_{q \in Q} c_q \lambda_q \\ \text{s. t.} & \quad \sum_{q \in Q} \mathbf{a}_q \lambda_q \geq \mathbf{b} \quad [\pi_b] \\ & \quad \sum_{q \in Q} \lambda_q = 1 \quad [\pi_0] \\ & \quad \lambda_q \in \{0, 1\} \quad \forall q \in Q \end{aligned}$$

- and the subproblem (*SP*):

$$\begin{aligned} z_{SP}^* = \min & \quad (\mathbf{c}^\top - \boldsymbol{\pi}_b^\top \mathbf{A}) \mathbf{x} - \pi_0 \\ \text{s. t.} & \quad \mathbf{Dx} \geq \mathbf{d} \quad [\pi_d] \\ & \quad \mathbf{x} \in \mathbb{Z}_+^n \end{aligned}$$

Branch-and-Price

Branch-and-Price Algorithm:

1. solve the relaxed *MP* using CG to optimality
2. branch, if λ^* is fractional
3. repeat

Branch-and-Price

Branch-and-Price Algorithm:

1. solve the relaxed *MP* using CG to optimality
2. branch, if λ^* is fractional
3. repeat

But how to branch?

- branching on a single λ_q is unbalanced:
 $\lambda_q \leq 0$ forbids almost nothing, $\lambda_q \geq 1$ forbids much

Branch-and-Price

Branch-and-Price Algorithm:

1. solve the relaxed *MP* using CG to optimality
2. branch, if λ^* is fractional
3. repeat

But how to branch?

- branching on a single λ_q is unbalanced:
 $\lambda_q \leq 0$ forbids almost nothing, $\lambda_q \geq 1$ forbids much
- branch on a set of columns $Q' \subset Q$, such that:

$$\sum_{q \in Q'} \lambda_q^* =: K \notin \mathbb{Z}$$

Component Bound Sequences - Vanderbeck *et al.* 1996

- to find such a set Q' of columns, we find a component bound sequence:

$$S := \{(x_i, \eta_i, v_i) \mid \eta_i \in \{\leq, \geq\}, v_i \in \mathbb{Z}\}$$

Component Bound Sequences - Vanderbeck *et al.* 1996

- to find such a set Q' of columns, we find a component bound sequence:

$$S := \{(x_i, \eta_i, v_i) \mid \eta_i \in \{\leq, \geq\}, v_i \in \mathbb{Z}\}$$

- a master variable λ_q satisfies a component bound (x_i, η_i, v_i) if:

$$x_{q,i} \ \eta_i \ v_i$$

- we define Q' as the set of columns, that satisfy all component bounds in S
- there always exists a component bound sequence S such that:

$$\sum_{q \in Q'} \lambda_q^* \in (0, 1)$$

Component Bound Sequences

	q_1	q_2	q_3	q_4
x_{1,q_i}	1	1	2	2
x_{2,q_i}	1	2	1	2
$\lambda_{q_i}^*$	0.25	0	0.5	0.25

Component Bound Sequences

	q_1	q_2	q_3	q_4
x_{1,q_i}	1	1	2	2
x_{2,q_i}	1	2	1	2
$\lambda_{q_i}^*$	0.25	0	0.5	0.25

- Find $S = \{(x_1, \leq, 1)\}$
- $Q' = \{q_1, q_2\}$
- $\sum_{q \in Q'} \lambda_q^* = 0.25 \notin \mathbb{Z}$

Introduction
ooooo

Vanderbeck's Generic Branching
●○○

Component Bound Branching Rule
○○○

Generic Mastercuts
○

Evaluation
oooooooo

Conclusion
○

Table of Contents

Introduction

Vanderbeck's Generic Branching

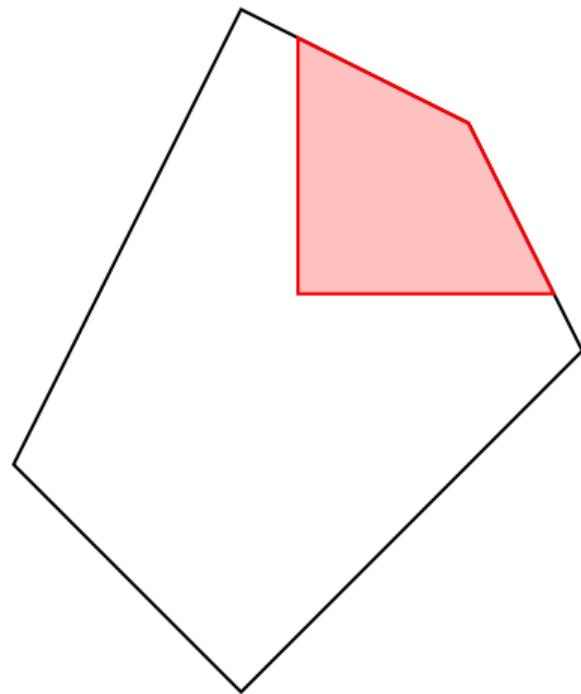
Component Bound Branching Rule

Generic Mastercuts

Evaluation

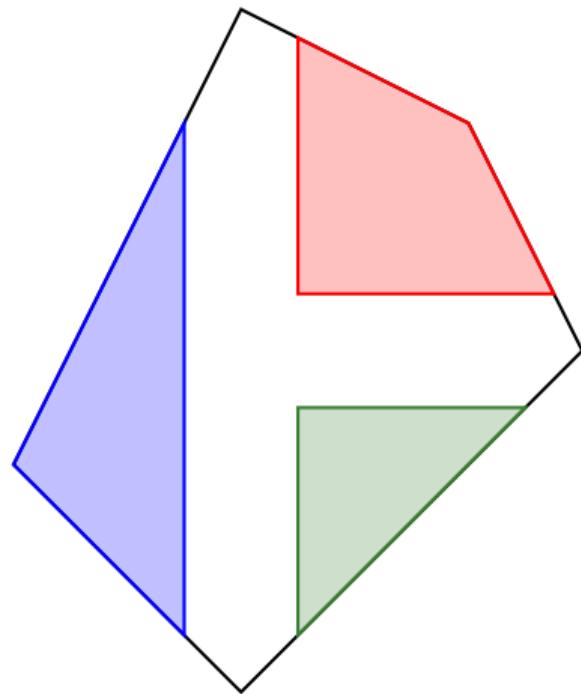
Conclusion

Vanderbeck's Generic Branching Scheme - Vanderbeck 2011



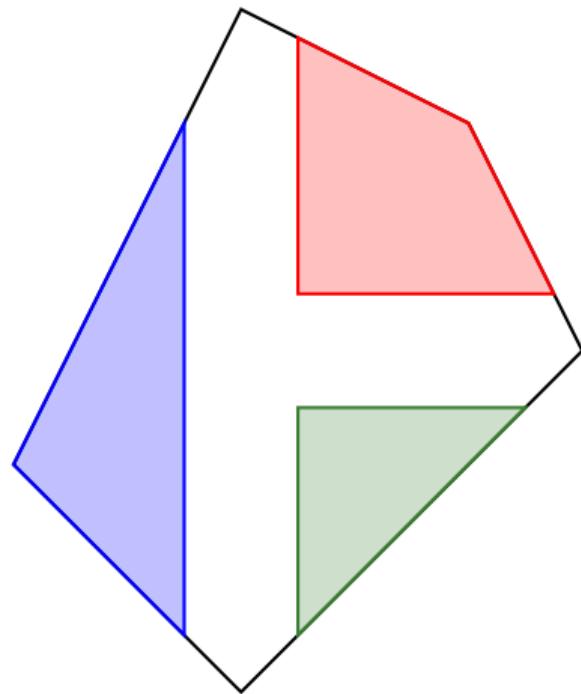
- assume we have found a component bound sequence
- $$S = \{(x_1, \geq, 1), (x_2, \geq, 2)\}$$

Vanderbeck's Generic Branching Scheme - Vanderbeck 2011



- assume we have found a component bound sequence
 $S = \{(x_1, \geq, 1), (x_2, \geq, 2)\}$
- create $|S| + 1 = 3$ child nodes:
 1. $S_1 := \{(x_1, \leq, 0)\}$
 2. $S_2 := \{(x_1, \geq, 1), (x_2, \leq, 1)\}$
 3. $S_3 := \{(x_1, \geq, 1), (x_2, \geq, 2)\}$
- search for solutions only within these regions

Vanderbeck's Generic Branching Scheme - Vanderbeck 2011



- assume we have found a component bound sequence
 $S = \{(x_1, \geq, 1), (x_2, \geq, 2)\}$
- create $|S| + 1 = 3$ child nodes:
 1. $S_1 := \{(x_1, \leq, 0)\}$
 2. $S_2 := \{(x_1, \geq, 1), (x_2, \leq, 1)\}$
 3. $S_3 := \{(x_1, \geq, 1), (x_2, \geq, 2)\}$
- search for solutions only within these regions
- refine the regions in deeper nodes

Enforcing a Component Bound Sequence S_j in Node j

- to the MP add constraint:

$$\sum_{q \in Q'_j} \lambda_q \geq 1$$

- move the bounds in the SP :

$$x_i \geq v_i \quad \forall (x_i, \geq, v_i) \in S_j$$

$$x_i \leq v_i \quad \forall (x_i, \leq, v_i) \in S_j$$

Component Bound Branching Rule - Desrosiers *et al.* 2024

Again:

- find a subset Q' of columns using a component bound sequence S :

$$S := \{(x_i, \eta_i, v_i) \mid \eta_i \in \{\leq, \geq\}, v_i \in \mathbb{Z}\}$$

- such that: $\sum_{q \in Q'} \lambda_q^* \in (0, 1)$

Component Bound Branching Rule - Desrosiers *et al.* 2024

Again:

- find a subset Q' of columns using a component bound sequence S :

$$S := \{(x_i, \eta_i, v_i) \mid \eta_i \in \{\leq, \geq\}, v_i \in \mathbb{Z}\}$$

- such that: $\sum_{q \in Q'} \lambda_q^* \in (0, 1)$

Alternative scheme:

- create binary search tree:

$$\sum_{q \in Q'} \lambda_q \leq 0 \quad [\gamma]$$

$$\sum_{q \in Q'} \lambda_q \geq 1 \quad [\gamma] \quad (1)$$

Component Bound Branching Rule - Desrosiers *et al.* 2024

Again:

- find a subset Q' of columns using a component bound sequence S :

$$S := \{(x_i, \eta_i, v_i) \mid \eta_i \in \{\leq, \geq\}, v_i \in \mathbb{Z}\}$$

- such that: $\sum_{q \in Q'} \lambda_q^* \in (0, 1)$

Alternative scheme:

- create binary search tree:

$$\sum_{q \in Q'} \lambda_q \leq 0 \quad [\gamma] \quad \sum_{q \in Q'} \lambda_q \geq 1 \quad [\gamma] \quad (1)$$

- allows for solutions violating S to be found / generated

Component Bound Branching Rule - Desrosiers *et al.* 2024

- SP determines whether a new column is in Q' , i.e., satisfies S
- create new variable $y \in \{0, 1\}$ along new constraints, such that: $y = 1 \Leftrightarrow x \in S$

Component Bound Branching Rule - Desrosiers *et al.* 2024

- SP determines whether a new column is in Q' , i.e., satisfies S
- create new variable $y \in \{0, 1\}$ along new constraints, such that: $y = 1 \Leftrightarrow x \in S$

$$z_{SP}^* = \min \quad (\mathbf{c}^\top - \boldsymbol{\pi}_b^\top \mathbf{A}) \mathbf{x} - \gamma y - \pi_0$$

s. t.

$$\mathbf{Dx} \geq \mathbf{d}$$

$$y = 1 \Leftrightarrow \sum_{i \in S} y_i = |S|$$

$$y_i = 1 \Leftrightarrow x_i \leq v_i \quad \forall (x_i, \leq, v_i) \in S$$

$$y_i = 1 \Leftrightarrow x_i \geq v_i \quad \forall (x_i, \geq, v_i) \in S$$

$$y \in \{0, 1\}$$

$$y_i \in \{0, 1\} \quad \forall (x_i, \eta_i, v_i) \in S$$

$$\mathbf{x} \in \mathbb{Z}_+^n$$

Differences

Vanderbeck's Generic Branching	Component Bound Branching
tightens bounds in SP → use special algorithms	adds new variables and constraints to SP → fall back to MIP solver

Differences

Vanderbeck's Generic Branching	Component Bound Branching
tightens bounds in SP → use special algorithms	adds new variables and constraints to SP → fall back to MIP solver
refines component bounds → SP becomes faster to solve	does not refine component bounds → SP becomes slower to solve

Master Constraints without Original Formulation

- we want to add constraint MP :

$$\sum_{p \in P} f(p)\lambda_p + \sum_{r \in R} f(r)\lambda_r \leq f \quad [\gamma]$$

- with the following modification to the pricing problem:

$$\begin{aligned} z_{SP}^* = \min \quad & (\mathbf{c}^\top - \boldsymbol{\pi}_b^\top \mathbf{A}) \mathbf{x} - \gamma \mathbf{y} - \pi_0 \\ \text{s. t.} \quad & \mathbf{Dx} \geq \mathbf{d} \quad [\pi_d] \\ & \mathbf{y} = \mathbf{f}(\mathbf{x}) \\ & \mathbf{x} \in \mathbb{Z}_+^n \\ & \mathbf{y} \in \mathcal{Y} \end{aligned}$$

Core Evaluation Questions

- implemented generic mastercuts as a new interface in GCG, Gamrath 2010
- implemented component bound branching using this interface

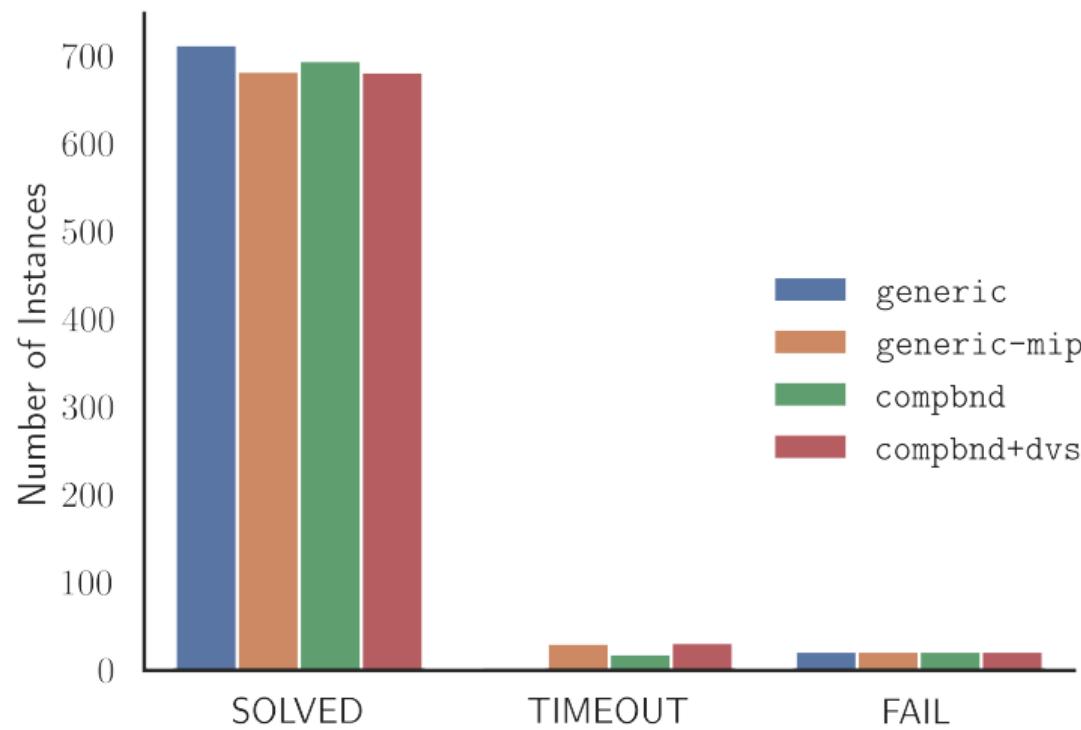
Core Evaluation Questions

- implemented generic mastercuts as a new interface in GCG, Gamrath 2010
 - implemented component bound branching using this interface
1. Vanderbeck's Generic Branching vs. Component Bound Branching Rule
 2. Dual Value Stabilization: Root-Only vs. Full-Tree

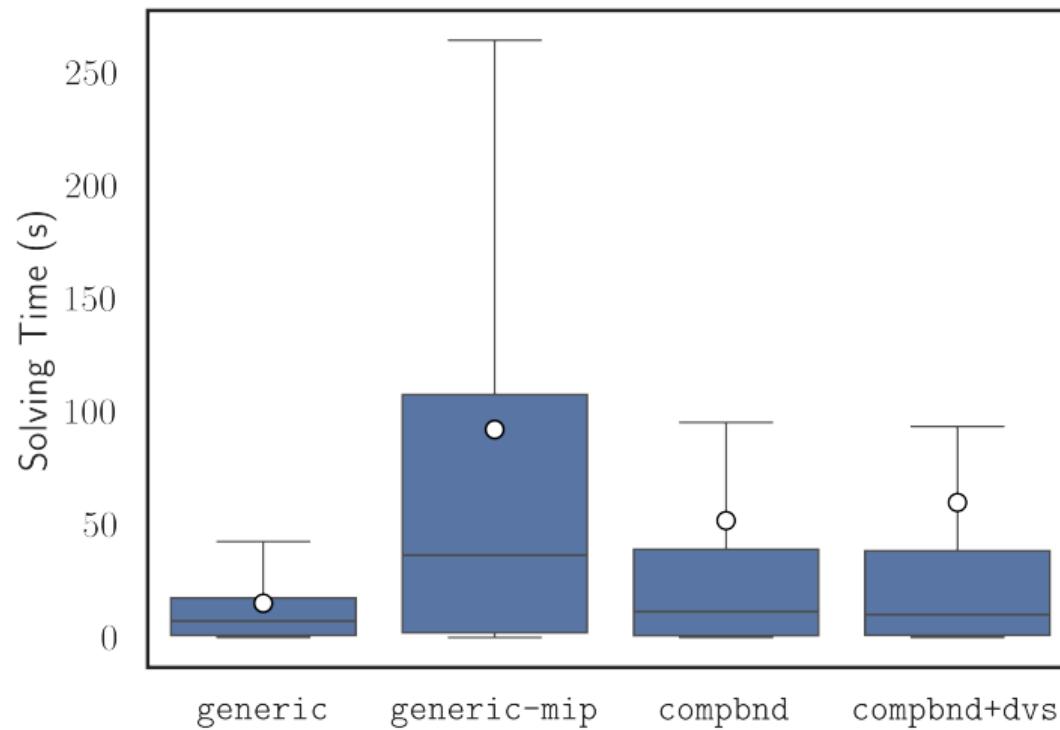
Core Evaluation Questions

- implemented generic mastercuts as a new interface in GCG, Gamrath 2010
 - implemented component bound branching using this interface
1. Vanderbeck's Generic Branching vs. Component Bound Branching Rule
 2. Dual Value Stabilization: Root-Only vs. Full-Tree
- extracted 736 instances from the structured Integer Programming Library (**strIPLib**), Bastubbe *et al.* 2025

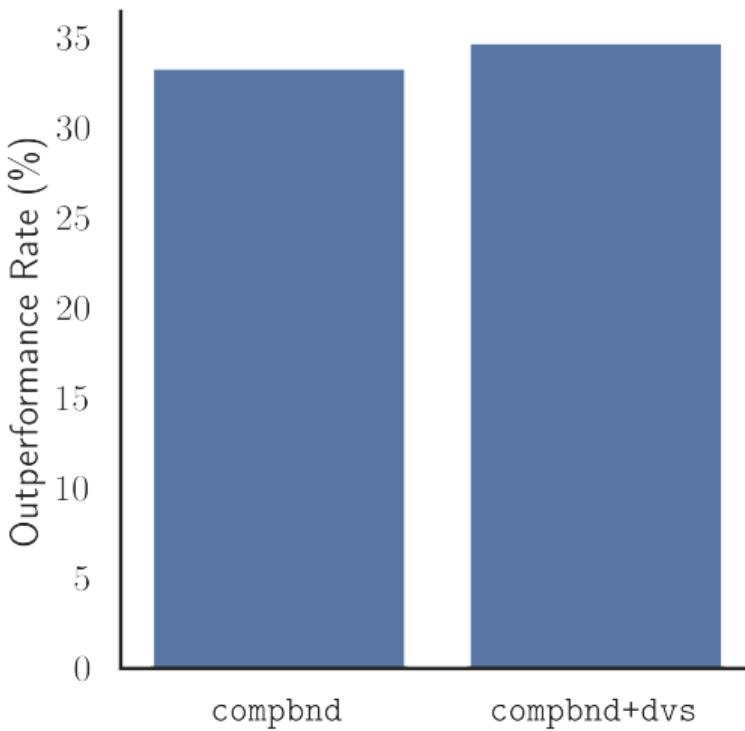
Evaluation · Solving Status



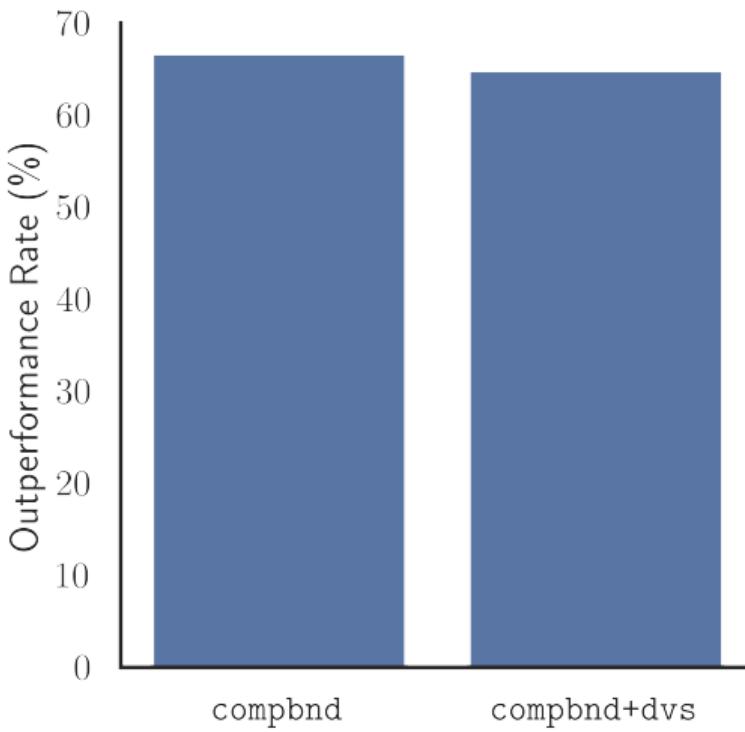
Evaluation · Solving Times



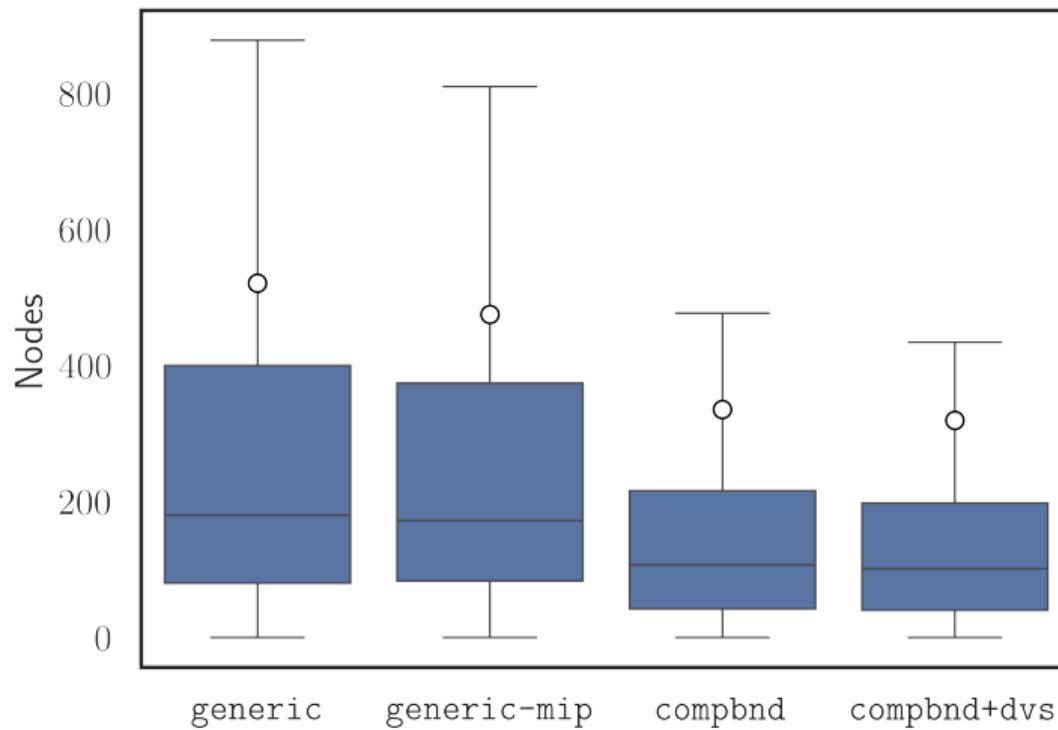
Evaluation · Outperformance Rate over generic



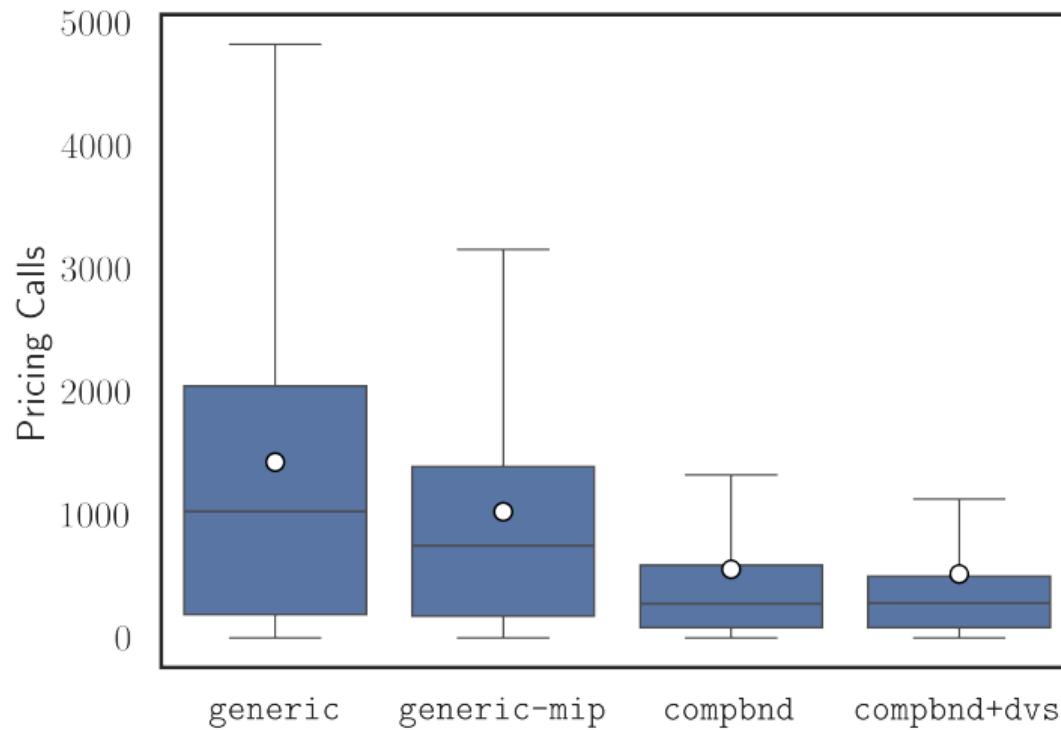
Evaluation · Outperformance Rate over generic-mip



Evaluation · Number of Nodes



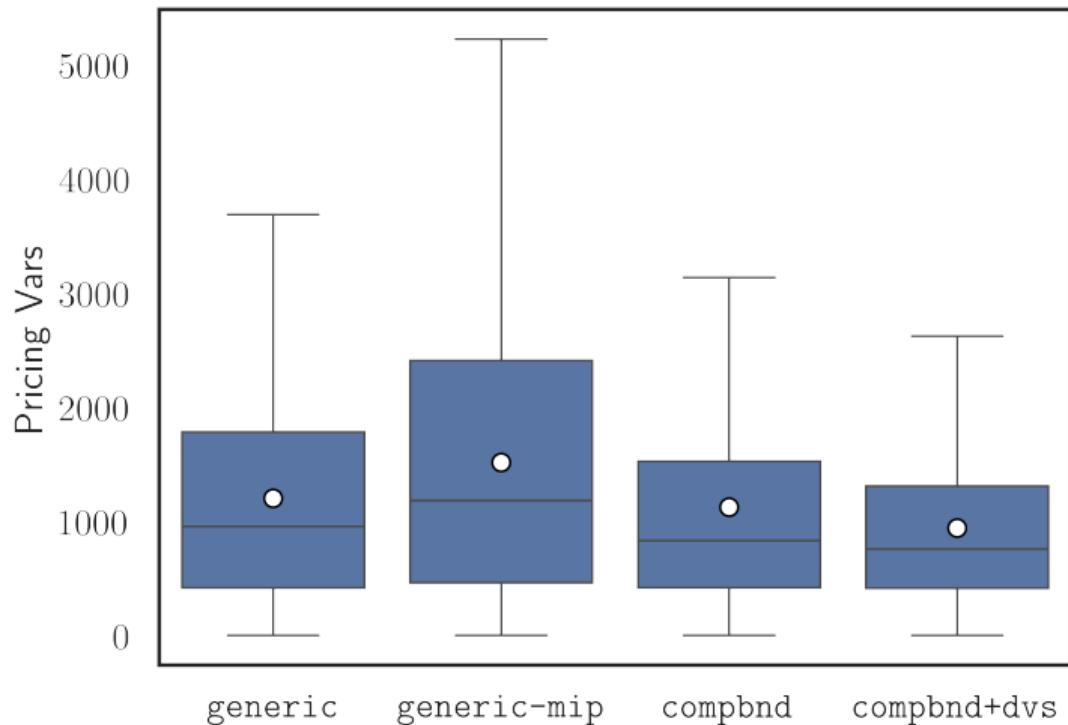
Evaluation · Pricing Calls



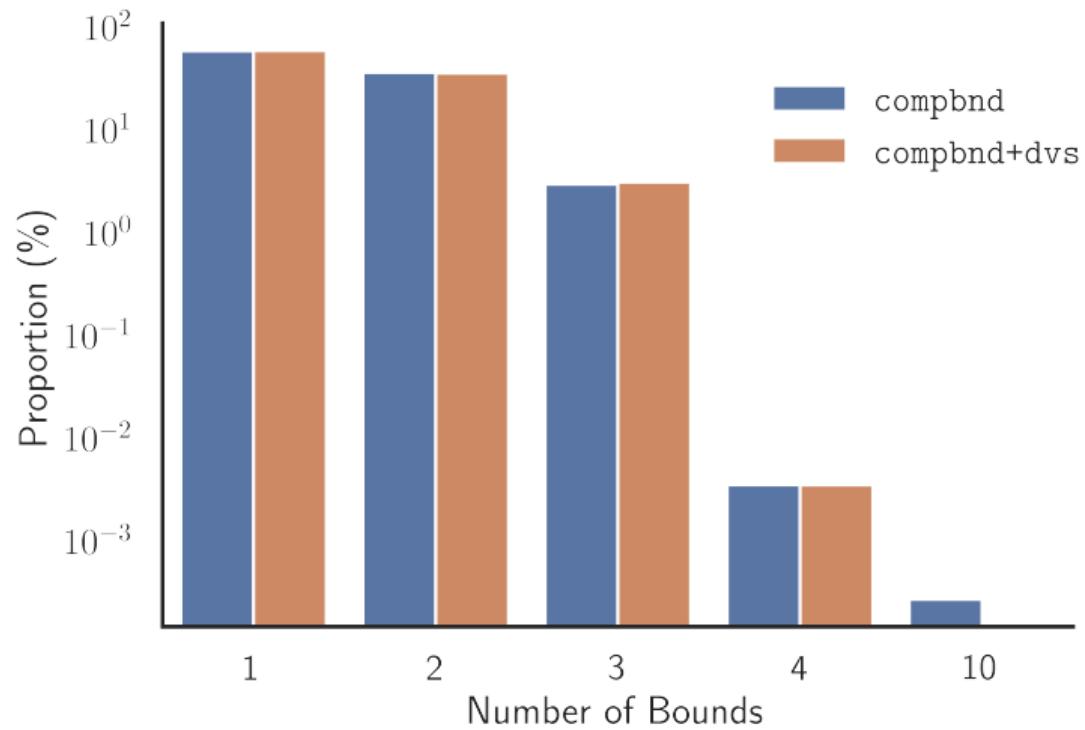
Conclusion

- GCG has a **new interface** for generic master constraints
- GCG has a **new branching rule** operating on component bound sequences
 - opens up new possibilities for future research (e.g., master separators)

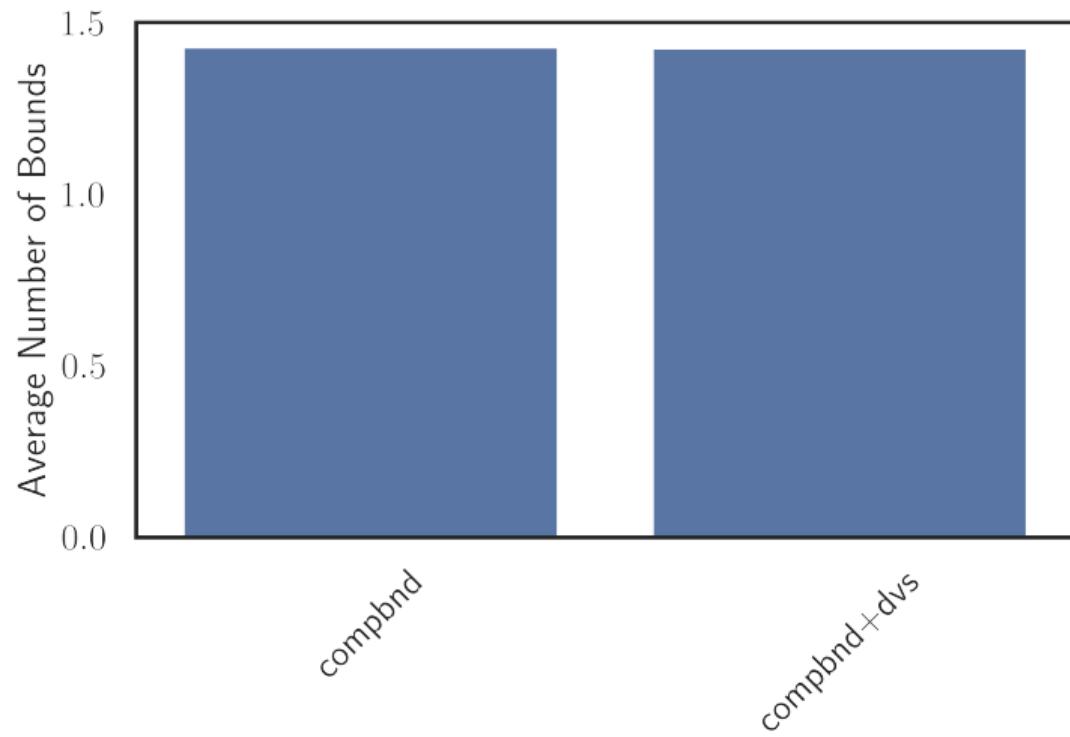
Appendix



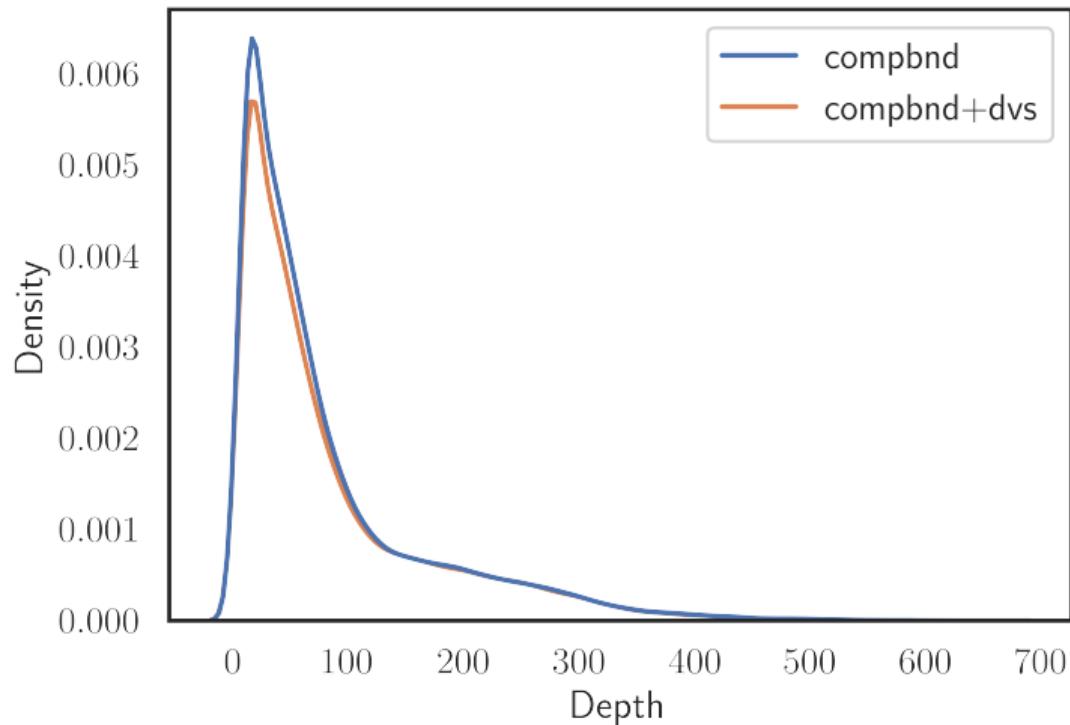
Appendix



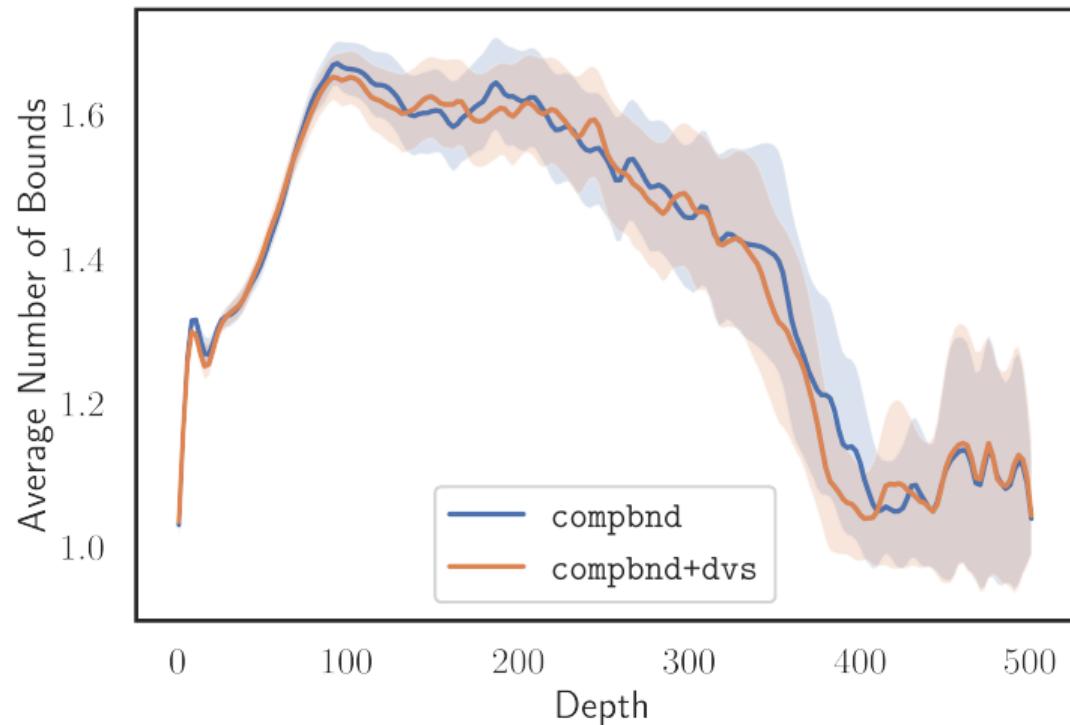
Appendix



Appendix



Appendix



References I

- Achterberg, Tobias (2007). “Constraint integer programming”.
- Achterberg, Tobias, Thorsten Koch, and Alexander Martin (2005). “Branching rules revisited”. *Operations Research Letters* 33.1, pp. 42–54.
- Bastubbe, Michael, Marco E Lübbeke, and Jonas T Witt (2018). “A Computational Investigation on the Strength of Dantzig-Wolfe Reformulations”. *17th International Symposium on Experimental Algorithms (SEA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- Bastubbe, Michael et al. (2025). “strIPlib: Structured Integer Programming Library”.
- Chappell, Nathan (2019). “Minkowski-Weyl Theorem”.
- Dantzig, George B and Mukund N Thapa (1997). *The simplex method*. Springer.
- Desrosiers, Jacques et al. (June 2024). *Branch-and-Price*. Les Cahiers du GERAD G-2024-36. GERAD, Montréal QC H3T 2A7, Canada: Groupe détudes et de recherche en analyse des décisions, pp. 1–689. eprint:
<https://www.gerad.ca/papers/G-2024-36.pdf?locale=fr>. published.

References II

- Gamrath, Gerald (2010). "Generic branch-cut-and-price". MA thesis.
- Matouek, Jií and Bernd Gärtner (2007). *Understanding and using linear programming*. Vol. 1. Springer.
- Pessoa, Artur et al. (2013). "In-out separation and column generation stabilization by dual price smoothing". *Experimental Algorithms: 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings* 12. Springer, pp. 354–365.
- Pessoa, Artur et al. (2018). "Automation and combination of linear-programming based stabilization techniques in column generation". *INFORMS Journal on Computing* 30.2, pp. 339–360.
- Ploskas, Nikolaos and Nikolaos Samaras (2014). "Pivoting rules for the revised simplex algorithm". *Yugoslav Journal of Operations Research* 24.3, pp. 321–332.
- Schmickerath, Marcel (2012). "Experiments on Vanderbeck's generic Branch-and-Price scheme".

References III

- Vanderbeck, François (2011). “Branching in branch-and-price: a generic scheme”. *Mathematical Programming* 130, pp. 249–294.
- Vanderbeck, François and Laurence A Wolsey (1996). “An exact algorithm for IP column generation”. *Operations research letters* 19.4, pp. 151–159.
- (2010). *Reformulation and decomposition of integer programs*. Springer.
- Witt, Jonas T (2013). “Separation of Generic Cutting Planes in Branch-and-Price”. PhD thesis. Master’s thesis. RWTH Aachen University.
- Wolsey, Laurence A and George L Nemhauser (2014). *Integer and combinatorial optimization*. John Wiley & Sons.