Learning to Branch in MILP Solvers

Maxime Gasse, Didier Chetelat, Laurent Charlin, Andrea Lodi

maxime.gasse@polymtl.ca

TTI-C Workshop on Automated Algorithm Design
Chicago, August 7-9th 2019

POLYTECHNIQUE MONTRÉAL

GERAD
GROUPE D'ÉTUDES ET DE RECHERCHE EN ANALYSE DES DÉCISIONS

DATA SCIENCE FOR REAL-TIME DECISION-MAKING

Mila

Overview

The Branching Problem

Mixed-Integer Linear Program (MILP)

$$\begin{aligned} \arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \\ & \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned}$$

- $\mathbf{c} \in \mathbb{R}^n$ the objective coefficients
- $\mathbf{A} \in \mathbb{R}^{m \times n}$ the constraint coefficient matrix
- $\mathbf{b} \in \mathbb{R}^m$ the constraint right-hand-sides
- $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$ the lower and upper variable bounds
- $p \leq n$ integer variables

NP-hard problem.

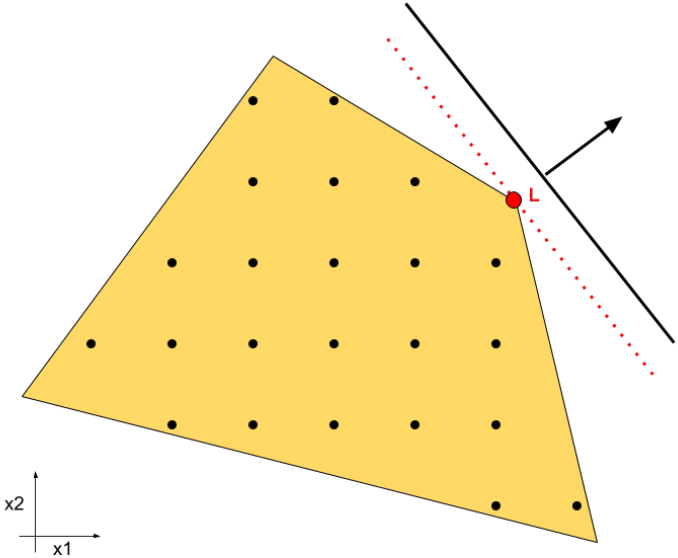
Linear Program (LP) relaxation

$$
\begin{aligned}
\arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \\
& \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \\
& \mathbf{x} \in \mathbb{R}^n.
\end{aligned}
$$

Convex problem, efficient algorithms (e.g., simplex).

- $\mathbf{x}^\star \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$ (lucky) $\rightarrow$ solution to the original MILP
- $\mathbf{x}^\star \notin \mathbb{Z}^p \times \mathbb{R}^{n-p}$ $\rightarrow$ lower bound to the original MILP

Linear Program (LP) relaxation



Branch-and-Bound

Split the LP recursively over a non-integral variable, i.e. $\exists i \leq p \mid x_i^\star \notin \mathbb{Z}$

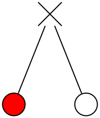
$$x_i \leq \lfloor x_i^\star \rfloor \quad \vee \quad x_i \geq \lceil x_i^\star \rceil.$$

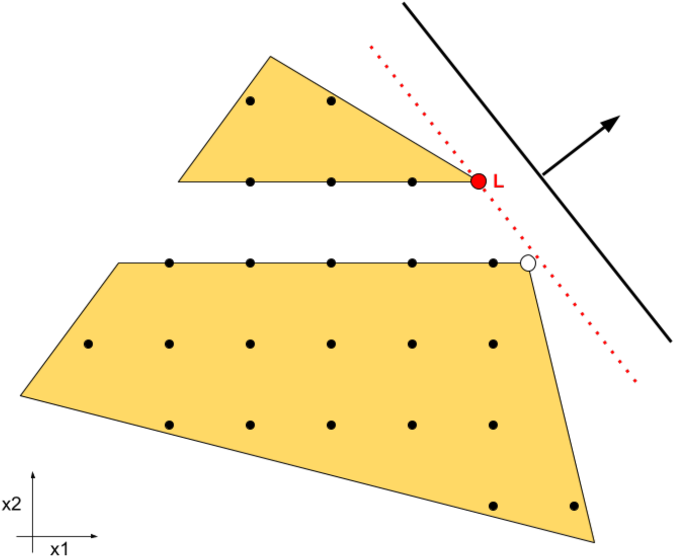
Lower bound (L): minimal among leaf nodes.
Upper bound (U): minimal among integral leaf nodes.

Stopping criterion:

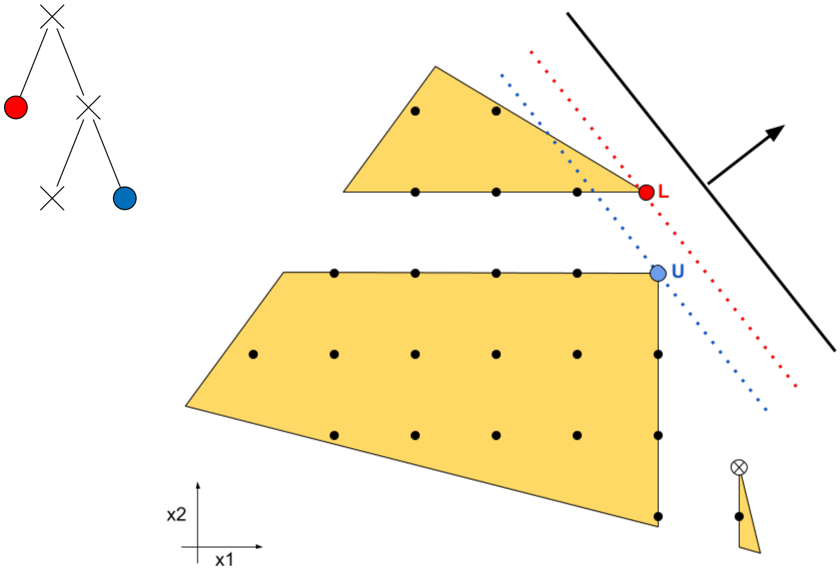
- L = U (optimality certificate)
- L = ∞ (infeasibility certificate)
- L - U < threshold (early stopping)

Branch-and-Bound

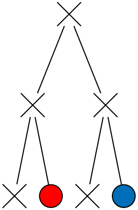


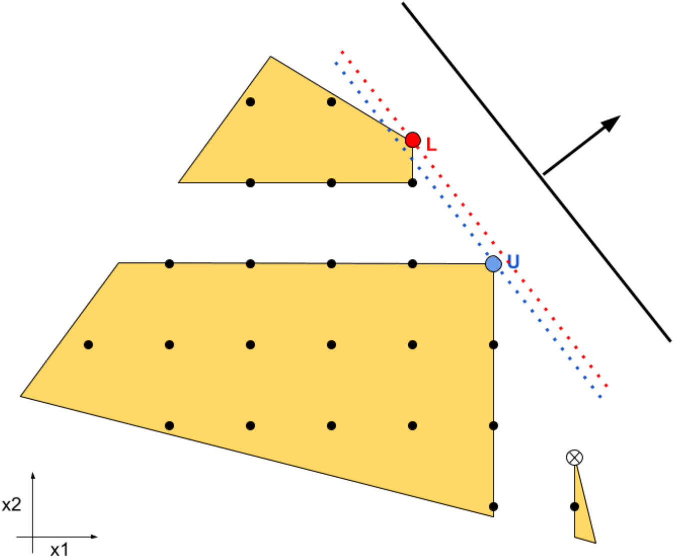


Branch-and-Bound

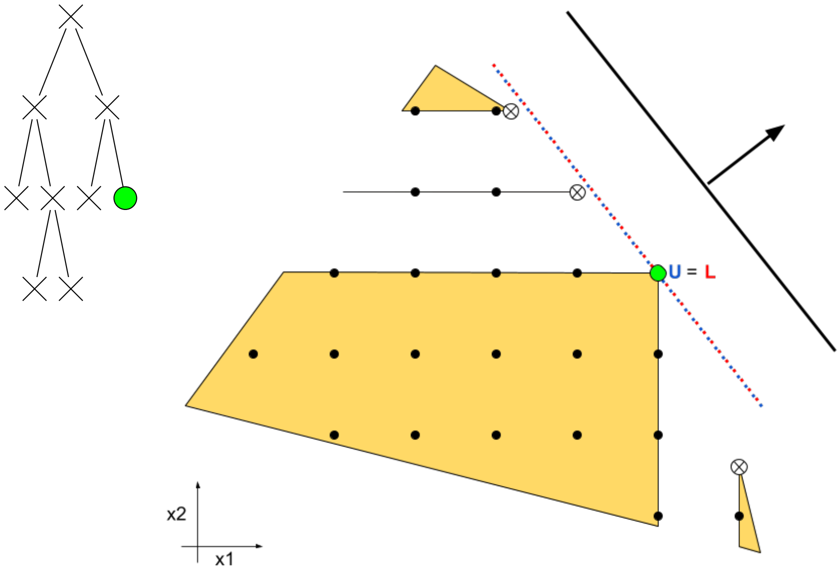


Branch-and-Bound





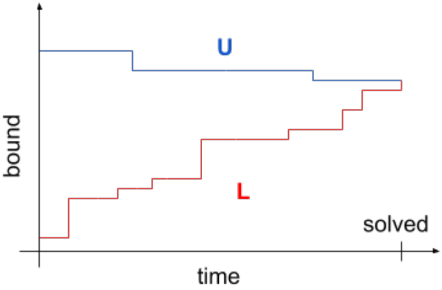
Branch-and-Bound



Branch-and-bound: a sequential process

Sequential decisions:

- node selection
- variable selection (branching)
- *cutting plane selection*
- *primal heuristic selection*
- *simplex initialization*
- ...



Objective: no clear consensus

- L = U fast ?
- U - L ↘ fast ?
- L ↗ fast ?
- U ↘ fast ?

State-of-the-art in B&B solvers: expert rules

Markov Decision Process

Objective: take actions which maximize the long-term reward

$$\sum_{t=0}^{\infty} r(\mathbf{s}_t),$$

with $r : \mathcal{S} \to \mathbb{R}$ a reward function.

Branching as a Markov Decision Process

State: the whole internal state of the solver, $\mathbf{s}$.
Action: a branching variable, $a \in \{1, \dots, p\}$.

Trajectory: $\tau = (\mathbf{s}_0, \dots, \mathbf{s}_T)$

- initial state $\mathbf{s}_0$: a MILP $\sim p(\mathbf{s}_0)$;
- terminal state $\mathbf{s}_T$: the MILP is solved;
- intermediate states: branching

$$\mathbf{s}_{t+1} \sim p_\pi(\mathbf{s}_{t+1}|\mathbf{s}_t) = \sum_{a \in \mathcal{A}} \underbrace{\pi(a|\mathbf{s}_t)}_{\text{branching policy}} \underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t, a)}_{\text{solver internals}}.$$

Branching problem: solve

$$\pi^\star = \arg\max_\pi \mathop{\mathbb{E}}_{\tau \sim p_\pi} [r(\tau)],$$

with $r(\tau) = \sum_{\mathbf{s} \in \tau} r(\mathbf{s})$.

The branching problem: considerations

A policy $\pi^\star$ may not be optimal in two distinct configurations.

Initial distribution $p(\mathbf{s}_0)$?

- Collection of MILPs of interest.

Transition distribution $p(\mathbf{s}_{i+1}|\mathbf{s}_i, a)$?

- Solver internals + parameterization.

Reward function $r(\tau)$?

- negative running time $\implies$ solve quickly
- negative duality gap integral $\implies$ fast gap closing
- negative upper bound integral $\implies$ diving heuristic
- lower bound integral $\implies$ fast relaxation tightening

Expert branching rules: state-of-the-art

Strong branching: one-step forward looking

- solve both LPs for each candidate variable
- pick the variable resulting in tightest relaxation
- \+ small trees
- − computationally expensive

Pseudo-cost: backward looking

- keep track of tightenings in past branchings
- pick the most promising variable
- \+ very fast, almost no computations
- − cold start

Reliability pseudo-cost: best of both worlds

- compute SB scores at the beginning
- gradually switches to pseudo-cost (+ other heuristics)
- \+ best overall solving time trade-off (on MIPLIB)

Machine learning approaches

Node selection

- He et al., 2014
- Song et al., 2018

Variable selection (branching)

- Khalil, Le Bodic, et al., 2016 $\Longrightarrow$ "online" imitation learning
- Hansknecht et al., 2018 $\Longrightarrow$ offline imitation learning
- Balcan et al., 2018 $\Longrightarrow$ theoretical results

Cut selection

- Baltean-Lugojan et al., 2018
- Tang et al., 2019

Primal heuristic selection

- Khalil, Dilkina, et al., 2017
- Hendel et al., 2018

Challenges

MDP $\implies$ Reinforcement learning (RL) ?

State representation: $\mathbf{s}$

- global level: original MILP, tree, bounds, focused node...
- node level: variable bounds, LP solution, simplex statistics...
- – dynamically growing structure (tree)
- – variable-size instances (cols, rows) $\implies$ Graph Neural Network

Sampling trajectories: $\tau \sim p_\pi$

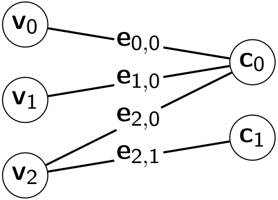
- collect one τ = solving a MILP (with π likely not optimal)
- – expensive $\implies$ train on small instances, use pre-trained policy

The Graph Convolution Neural Network Model

Node state encoding

Natural representation : variable / constraint bipartite graph

$$
\begin{aligned}
\arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} \le \mathbf{b}, \\
& \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \\
& \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}.
\end{aligned}
$$

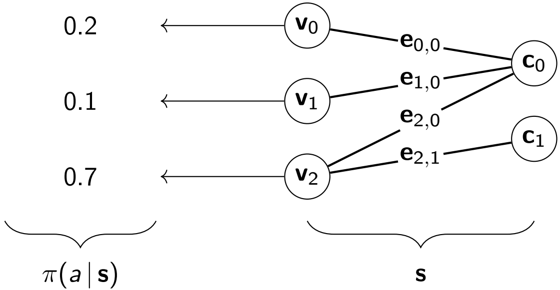


- $\mathbf{v}_i$: variable features (type, coef., bounds, LP solution...)
- $\mathbf{c}_j$: constraint features (right-hand-side, LP slack...)
- $\mathbf{e}_{i,j}$: non-zero coefficients in $\mathbf{A}$

D. Selsam et al. (2019). Learning a SAT Solver from Single-Bit Supervision.

Branching Policy as a GCNN Model

Neighbourhood-based updates: $\mathbf{v}_i \leftarrow \sum_{j \in \mathcal{N}_i} \mathbf{f}_\theta(\mathbf{v}_i, \mathbf{e}_{i,j}, \mathbf{c}_j)$



Natural model choice for graph-structured data

- permutation-invariance
- benefits from sparsity

T. N. Kipf et al. (2016). Semi-Supervised Classification with Graph Convolutional Networks.

Experiments: Imitation Learning

Strong Branching approximation

Full Strong Branching (FSB): good branching rule, but expensive.
Can we learn a fast, good-enough approximation ?

Not a new idea

- Alvarez et al., 2017 predict SB scores, XTrees model
- Khalil, Le Bodic, et al., 2016 predict SB rankings, SVMrank model
- Hansknecht et al., 2018 do the same, λ-MART model

Behavioural cloning

- collect $\mathcal{D} = \{(\mathbf{s}, a^\star), \dots\}$ from the expert agent (FSB)
- estimate $\pi^\star(a \mid \mathbf{s})$ from $\mathcal{D}$

+ no reward function, supervised learning, well-behaved

– will never surpass the expert...

Implementation with the open-source solver SCIP[1]

[1] A. Gleixner et al. (2018). The SCIP Optimization Suite 6.

Minimum set covering[2]

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	20.19	0 / 100	16	282.14	0 / 100	215	3600.00	0 / 0	n/a
RPB	13.38	1 / 100	**63**	66.58	9 / 100	2327	1699.96	27 / 65	51 022
XTrees	14.62	0 / 100	199	106.95	0 / 100	3043	2726.56	0 / 36	58 608
SVMrank	13.33	1 / 100	157	89.63	0 / 100	2516	2401.43	0 / 48	42 824
λ-MART	**12.20**	**59** / 100	161	72.07	12 / 100	2584	2177.72	0 / 54	48 032
GCNN	12.25	39 / 100	130	**59.40**	**79** / 100	**1845**	**1680.59**	**40** / 64	**34 527**

3 problem sizes

- 500 rows, 1000 cols (easy), training distribution
- 1000 rows, 1000 cols (medium)
- 2000 rows, 1000 cols (hard)

Pays off: better than SCIP's default in terms of solving time.
Generalizes to harder problems !

[2] E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.

Maximum independent set[3]

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	34.82	5 / 100	7	2434.80	0 / 52	67	3600.00	0 / 0	n/a
RPB	12.01	3 / 100	**20**	175.00	28 / 100	1292	2759.82	11 / 34	8156
XTrees	11.77	4 / 100	79	1691.76	0 / 44	9441	3600.03	0 / 0	n/a
SVMrank	9.70	9 / 100	43	434.34	0 / 80	**867**	3499.30	0 / 4	10 256
λ-MART	8.36	18 / 100	48	318.38	6 / 84	1042	3493.27	0 / 3	15 368
GCNN	**7.81**	**61** / 100	38	**149.12**	**66** / 93	955	**2281.58**	**28** / 32	**5070**

3 problem sizes, Barabási-Albert graphs (affinity=4)

- 500 nodes (easy), training distribution
- 1000 nodes (medium)
- 1500 nodes (hard)

[3] D. Chalupa et al. (2014). On the Growth of Large Independent Sets in Scale-Free Networks.

Combinatorial auction[4]

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	7.27	0 / 100	5	92.49	0 / 100	72	1845.19	0 / 67	395
RPB	4.49	3 / 100	**8**	18.45	0 / 100	**630**	140.13	13 / 100	5440
XTrees	3.58	0 / 100	82	23.67	0 / 100	944	481.11	0 / 95	10 752
SVMrank	3.58	0 / 100	71	25.81	0 / 100	864	401.08	0 / 98	6353
λ-MART	**2.86**	66 / 100	70	15.23	3 / 100	849	227.44	1 / 100	6878
GCNN	2.88	31 / 100	64	**11.23**	**97** / 100	661	**118.74**	**86** / 100	**4912**

3 problem sizes

- 100 items, 500 bids (easy), training distribution
- 200 items, 1000 bids (medium)
- 300 items, 1500 bids (hard)

[4] K. Leyton-Brown et al. (2000). Towards a Universal Test Suite for Combinatorial Auction Algorithms.

Capacitated facility location[5]

	Easy			Medium			Hard		
Model	Time	Wins	Nodes	Time	Wins	Nodes	Time	Wins	Nodes
FSB	30.86	5 / 100	8	237.14	3 / 97	66	1231.37	1 / 92	81
RPB	28.12	23 / 100	**13**	182.31	1 / 100	**127**	829.54	3 / 100	**149**
XTrees	28.88	15 / 100	105	191.95	0 / 100	481	895.37	5 / 100	495
SVMrank	26.43	11 / 100	89	152.28	20 / 100	373	**726.79**	25 / 100	395
λ-MART	26.21	13 / 100	88	149.60	23 / 100	367	733.48	31 / 100	395
GCNN	**26.01**	**33** / 100	82	**147.22**	**53** / 100	365	761.88	**35** / 100	388

3 problem sizes

- 100 facilities, 100 customers (easy), training distribution
- 100 facilities, 200 customers (medium)
- 100 facilities, 400 customers (hard)

[5] G. Cornuejols et al. (1991). A comparison of heuristics and relaxations for the capacitated plant location problem.

Experiments: Reinforcement Learning

RL with actor-critic

Actor-critic policy gradient (state-of-the-art)

- Actor $\pi(a|\mathbf{s})$: policy
- Critic $Q(\mathbf{s}_i)$: value-function $\sum_{j=i}^{\infty} r(\mathbf{s}_j)$ $\approx$ running time prediction

Sample a dataset $\mathcal{D}$ of state-action trajectories

- $\tau = (\mathbf{s}_0, \ldots, \mathbf{s}_i, a_i, \mathbf{s}_{i+1}, \ldots, \mathbf{s}_T) \sim p_\pi$

Update the critic: $Q \leftarrow Q - \eta \nabla_Q$

- $\mathbb{E}_{\tau}^{\mathcal{D}} \left[\mathbb{E}_{\mathbf{s}_i}^{\tau} \left[(Q(\mathbf{s}_i) - \sum_{j=i}^{t} r(\mathbf{s}_j))^2 \right] \right]$

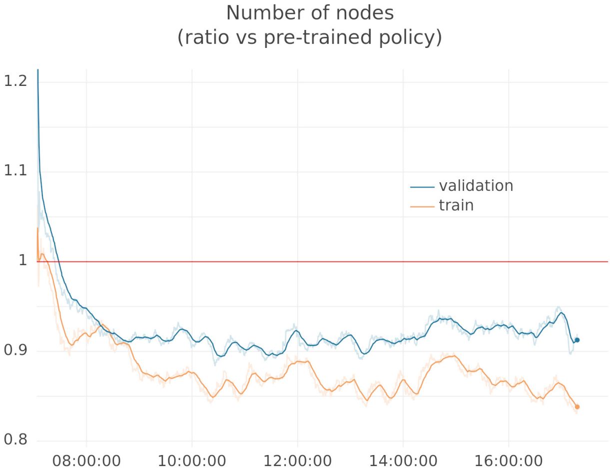
Update the actor: $\pi \leftarrow \pi + \eta \nabla_\pi$

- $\mathbb{E}_{\tau}^{\mathcal{D}} \left[\mathbb{E}_{\mathbf{s}_i, a_i, \mathbf{s}_{i+1}}^{\tau} \left[\log \pi(a_i|\mathbf{s}_i) Q(\mathbf{s}_{i+1}) \right] \right]$

Open question: good architecture / good features for the critic ?

RL with actor-critic

Early results: set covering problem



Reward: negative number of nodes

Proximal Policy Optimization (PPO)

Challenging. . . but promising !

Conclusion

Heuristic vs data-driven branching:

+ tune B&B to your problem of interest automatically
− no guarantees outside of the training distribution
− requires training instances

What next:

- real-world problems
- other solver components: node selection, cut selection...
- reinforcement learning: still a lot of challenges
- interpretation: which variables are chosen ? Why ?
- provide an clean API + benchmarks for MILP adaptive solving (based on the open-source SCIP solver)

Code online: `https://github.com/ds4dm/learn2branch`

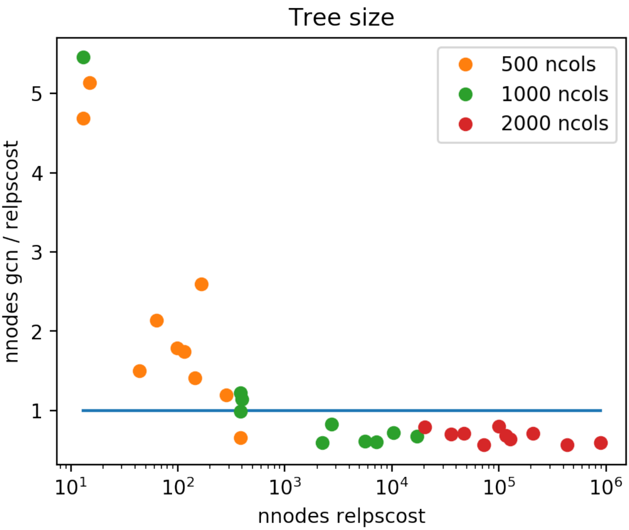
Learning to Branch in MILP Solvers

Thank you!

Maxime Gasse, Didier Chetelat, Laurent Charlin, Andrea Lodi

maxime.gasse@polymtl.ca

Learned Policy vs Reliability Pseudocost (SCIP default)

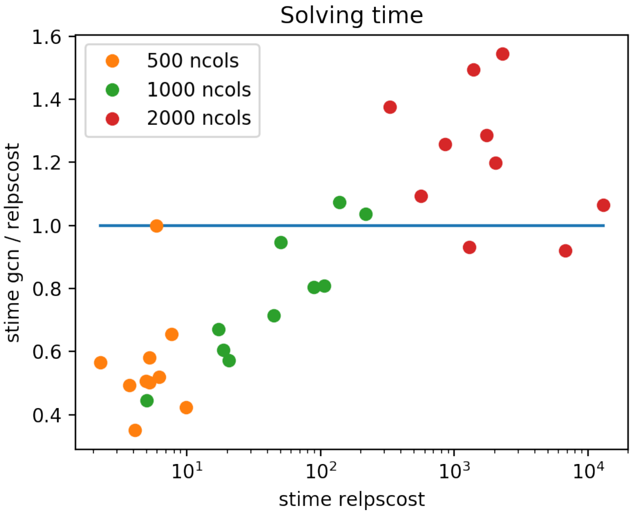


Trained on 500 cols only

Extrapolates to harder instances

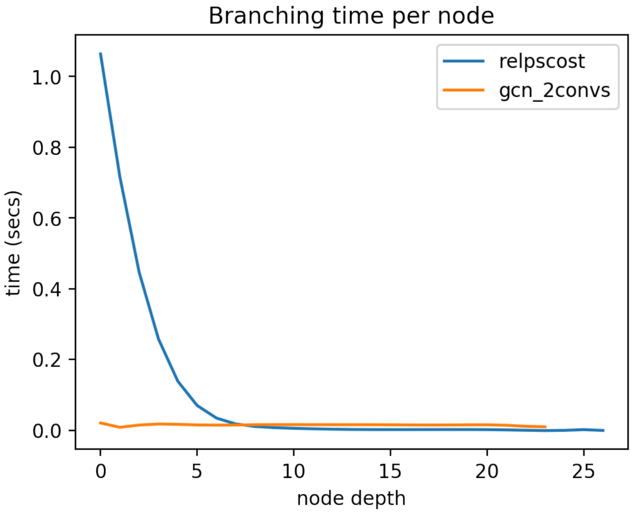
About 30-40% node reduction on those

Learned Policy vs Reliability Pseudocost (SCIP default)



Fewer nodes, but higher solving times...

Learned Policy vs Reliability Pseudocost (SCIP default)



Time delta:
- python overhead
- data extraction (**s**)
- model evaluation

Close the gap:
- engineering ?
- efficient heuristics (reliability) ?