Learning to Branch in Mixed Integer Programming

Elias Khalil

School of Computational Science & Engineering
Georgia Institute of Technology

AAAI 2016

Joint work with Pierre Le Bodic, Le Song,
George Nemhauser and Bistra Dilkina

Overview

- Goal: solve Mixed Integer Linear Programs (MIP) "faster"
- Branch-and-Bound (B&B): a general framework for solving MIPs
- B&B components: presolve, cutting planes, primal heuristics, **branching** (this talk)

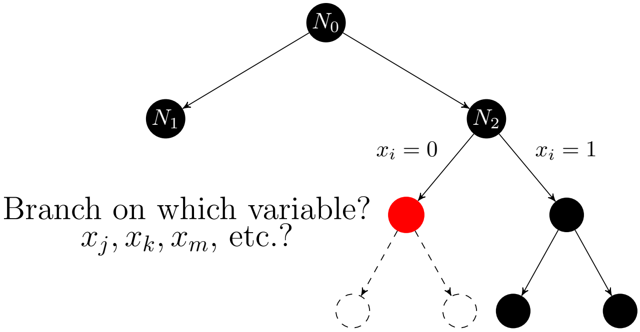


Figure: A B&B tree. A node is a sub-problem, an edge is an additional constraint

Motivation

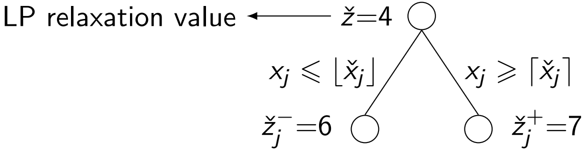
- Branching on the “right” variables can have a dramatic impact on size of B&B tree (i.e. number of nodes)

Motivation

- Branching on the "right" variables can have a dramatic impact on size of B&B tree (i.e. number of nodes)
- Example: suppose we are minimizing and the incumbent value is 5

Motivation

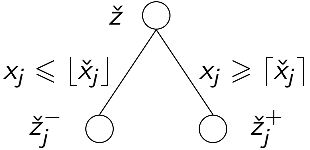
- Branching on the “right” variables can have a dramatic impact on size of B&B tree (i.e. number of nodes)
- Example: suppose we are minimizing and the incumbent value is 5



  - Branching on x_j prunes off the child sub-trees

Quality of a Branch

Measuring the quality of a branch:



- Change in objective value: $\Delta_j^- = \check{z}_j^- - \check{z}$ and $\Delta_j^+ = \check{z}_j^+ - \check{z}$
- Quality of the branch:

$$\text{score}(\Delta_j^-, \Delta_j^+) = \Delta_j^- \cdot \Delta_j^+$$

- Higher score is better

Two Classes of Branching Strategies

- Strong Branching (SB)

Two Classes of Branching Strategies

- Strong Branching (SB)
  - For some of the fractional integer variables at a node, *simulate* their branching quality by solving LPs, branch on var. with best product score

Two Classes of Branching Strategies

- Strong Branching (SB)
  - For some of the fractional integer variables at a node, *simulate* their branching quality by solving LPs, branch on var. with best product score
- Pseudocost Branching (PC)

Two Classes of Branching Strategies

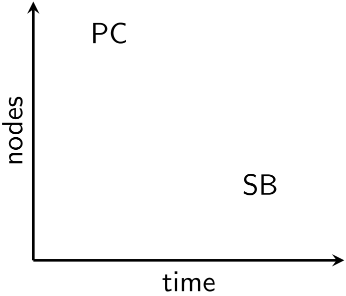
- Strong Branching (SB)
  - For some of the fractional integer variables at a node, *simulate* their branching quality by solving LPs, branch on var. with best product score
- Pseudocost Branching (PC)
  - Maintain some historical averages of the increase in obj. value, one for each of upwards and downwards branchings; combine by product.

Two Classes of Branching Strategies

- Strong Branching (SB)
  - For some of the fractional integer variables at a node, *simulate* their branching quality by solving LPs, branch on var. with best product score
- Pseudocost Branching (PC)
  - Maintain some historical averages of the increase in obj. value, one for each of upwards and downwards branchings; combine by product.
  - Imitate SB without solving many LP sub-problems.

Two Classes of Branching Strategies

- Strong Branching (SB)
  - For some of the fractional integer variables at a node, *simulate* their branching quality by solving LPs, branch on var. with best product score
- Pseudocost Branching (PC)
  - Maintain some historical averages of the increase in obj. value, one for each of upwards and downwards branchings; combine by product.
  - Imitate SB without solving many LP sub-problems.

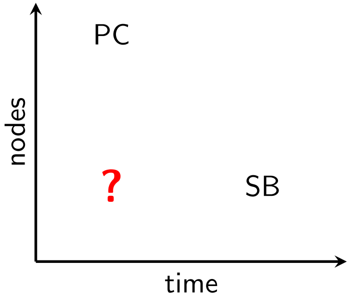


Contribution

- This talk: *can we get the best of both worlds (SB and PC)?*
  - small number of nodes
  - small time
  - as little manual tuning as possible

Contribution

- This talk: *can we get the best of both worlds (SB and PC)?*
  - small number of nodes
  - small time
  - as little manual tuning as possible



Contribution

- This talk: *can we get the best of both worlds (SB and PC)?*
  - small number of nodes
  - small time
  - as little manual tuning as possible

Proposed Method

A Machine Learning (ML) framework for branching that imitates SB well, at a fraction of the computation cost

Learning of Branching Strategies

- Regression:
  - Tempted to set up learning task as fitting a regression model to estimate SB scores directly

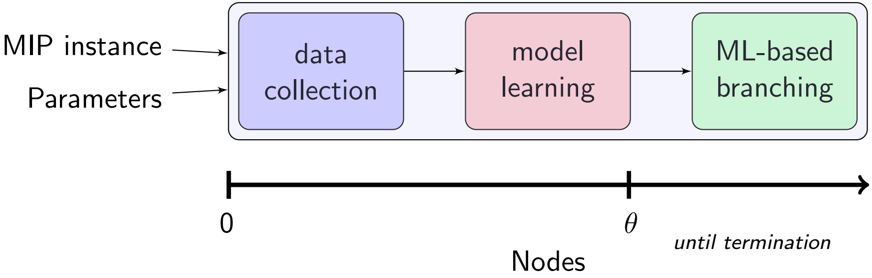
Learning of Branching Strategies

- Regression:
  - Tempted to set up learning task as fitting a regression model to estimate SB scores directly
- Key observation:
  - SB scores are only used to **select** the *single* best variable

Learning of Branching Strategies

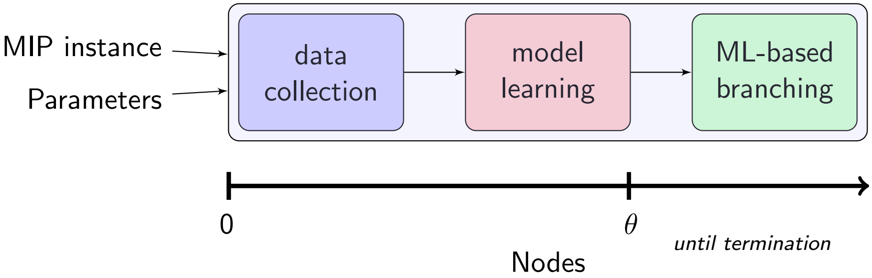
- Regression:
  - Tempted to set up learning task as fitting a regression model to estimate SB scores directly
- Key observation:
  - SB scores are only used to **select** the *single* best variable
- Goal: learn to imitate the correct **ranking** of the candidate variables at each node
  - Collect training data on SB ranking & fit a ranking model

Learning of Branching Strategies



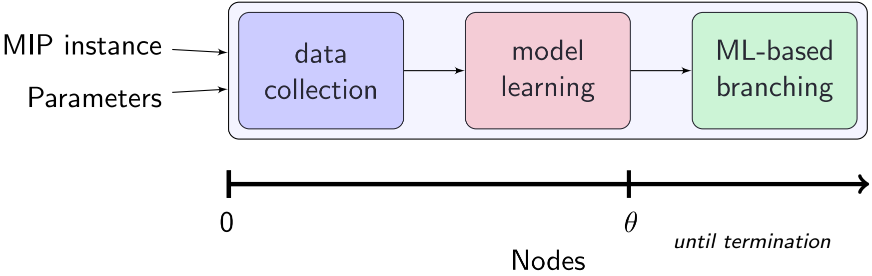
- **On-the-fly**: no upfront offline learning required
- **Instance-specific**: different ML model for each instance
- **No lost work**: data collection is part of the search tree

On-the-fly Learning of Branching Strategies



data collection: run SB for θ nodes, and use SB scores as *labels* for variables within each node; compute *features* describing each variable (dataset $\mathcal{D}$)

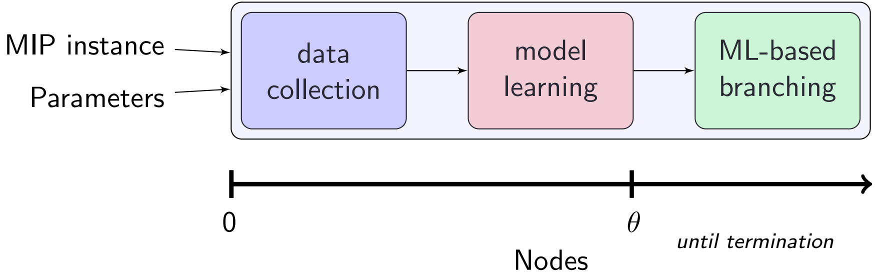
On-the-fly Learning of Branching Strategies



data collection: run SB for θ nodes, and use SB scores as *labels* for variables within each node; compute *features* describing each variable (dataset $\mathcal{D}$)

model learning: learn a model (function f) that **ranks** variables in $\mathcal{D}$ s.t. good ones are ranked better than bad ones

On-the-fly Learning of Branching Strategies



data collection: run SB for θ nodes, and use SB scores as *labels* for variables within each node; compute *features* describing each variable (dataset $\mathcal{D}$)
model learning: learn a model (function f) that **ranks** variables in $\mathcal{D}$ s.t. good ones are ranked better than bad ones
ML-based branching: as of the $(\theta + 1)^{st}$ node, use learned function f to rank variables based on their features

Data Collection

For the first θ nodes, $N_i \in \mathcal{N} = \{N_1, ..., N_\theta\}$, run SB on some fractional integer variables $\mathcal{C}_i$, $|\mathcal{C}_i| \leqslant \kappa$, and collect dataset $\mathcal{D}$ formed of:

Data Collection

For the first θ nodes, $N_i \in \mathcal{N} = \{N_1, ..., N_\theta\}$, run SB on some fractional integer variables $\mathcal{C}_i$, $|\mathcal{C}_i| \leqslant \kappa$, and collect dataset $\mathcal{D}$ formed of:

– *features* describing each candidate variable at that node

Data Collection

For the first θ nodes, $N_i \in \mathcal{N} = \{N_1, ..., N_\theta\}$, run SB on some fractional integer variables $\mathcal{C}_i$, $|\mathcal{C}_i| \leqslant \kappa$, and collect dataset $\mathcal{D}$ formed of:

- *features* describing each candidate variable at that node
- *labels* y_j^i, based on SB scores, s.t. higher is better

Data Collection

For the first θ nodes, $N_i \in \mathcal{N} = \{N_1, ..., N_\theta\}$, run SB on some fractional integer variables $\mathcal{C}_i$, $|\mathcal{C}_i| \leqslant \kappa$, and collect dataset $\mathcal{D}$ formed of:

- *features* describing each candidate variable at that node
- *labels* y_j^i, based on SB scores, s.t. higher is better

Example dataset: $\theta = 2, \kappa = 4$, $p = 4$ features

		y_j^i	feature 1	feature 2	feature 3	feature 4
N_1	x_2	0	0.4	0.4	0.3	0.9
	x_4	1	0.3	0.9	0.4	0.9
	x_5	0	0.2	0.2	0.6	0.7
	x_6	1	0.5	0.8	0.4	0.4
N_2	x_1	1	0.2	0.2	0.6	0.9
	x_2	0	0.5	0.2	0.8	0.7
	x_4	0	0.4	0.9	0.1	0.4

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0, 1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$.
  $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0,1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$.
  $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.
- **Atomic features**:

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0,1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$. $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.
- **Atomic features**:
  - *static* & *dynamic*, taking into account the structure of the node subproblem

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0,1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$.
  $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.
- **Atomic features**:
  - *static* & *dynamic*, taking into account the structure of the node subproblem
  - *easy to compute*, with time complexity $O\big(nnz(A)\big)$

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0,1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$.
  $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.
- **Atomic features**:
  - *static* & *dynamic*, taking into account the structure of the node subproblem
  - *easy to compute*, with time complexity $O\big(nnz(A)\big)$
- **Interaction features**:

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0,1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$. $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.
- **Atomic features**:
  - *static* & *dynamic*, taking into account the structure of the node subproblem
  - *easy to compute*, with time complexity $O\big(nnz(A)\big)$
- **Interaction features**:
  - create new features by multiplying each pair of atomic features.

Features

- Feature map $\phi : \mathcal{X} \times \mathcal{N} \rightarrow [0,1]^p$, where $\mathcal{X} = \{x_1, ..., x_n\}$.
  $\phi(x_j, N_i)$ describes variable x_j at node N_i with p features.
- **Atomic features**:
  - *static* & *dynamic*, taking into account the structure of the node subproblem
  - *easy to compute*, with time complexity $O\big(nnz(A)\big)$
- **Interaction features**:
  - create new features by multiplying each pair of atomic features.
- Scale and normalize all features, *per node*, to $[0,1]$.

Features

We currently use 72 atomic features.

Features

We currently use 72 atomic features.

Feature	Count	Reference
Static Features (18)		
Objective function coefficients	3	
Number of constraints	1	
Statistics for constraint degrees	4	
Statistics for constraint coefficients	10	

Features

We currently use 72 atomic features.

Feature	**Count**	**Reference**
Static Features (18)		
Objective function coefficients	3	
Number of constraints	1	
Statistics for constraint degrees	4	
Statistics for constraint coefficients	10	
Dynamic Features (54)		
Slack and ceil distances	2	
Pseudocosts	5	[Achterberg, 2009]
Infeasibility statistics	4	
Statistics for constraint degrees	7	
Min/max for ratios of constraint coefficients to RHS	4	[Alvarez et al., 2014]
Min/max for one-to-all coefficient ratios	8	[Alvarez et al., 2014]
Stats. for active constraint coefficients	24	[Patel and Chinneck, 2007]

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

- *The SB scores themselves:*

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

- *The SB scores themselves:*
  - Valid, but sensitive to variables with low SB scores (noisy)

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

- *The SB scores themselves:*
  - Valid, but sensitive to variables with low SB scores (noisy)
- *A label 1 to the variable with max. SB score, and a label 0 otherwise:*

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

- *The SB scores themselves:*
  - Valid, but sensitive to variables with low SB scores (noisy)
- *A label 1 to the variable with max. SB score, and a label 0 otherwise:*
  - Too strict, ignores the fact that there may be multiple "good" variables that deserve a label of 1.

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

- *The SB scores themselves:*
  - Valid, but sensitive to variables with low SB scores (noisy)
- *A label 1 to the variable with max. SB score, and a label 0 otherwise:*
  - Too strict, ignores the fact that there may be multiple "good" variables that deserve a label of 1.
- **Relaxed Binary Labels**:

Labels

How do we obtain the *labels* y_j^i? Some alternatives:

- *The SB scores themselves:*
  - Valid, but sensitive to variables with low SB scores (noisy)
- *A label 1 to the variable with max. SB score, and a label 0 otherwise:*
  - Too strict, ignores the fact that there may be multiple "good" variables that deserve a label of 1.
- **Relaxed Binary Labels**:

$$y_j^i = \begin{cases} 1, & \text{if SB score of } x_j \text{ is within } \alpha \text{ of max. SB score at } N_i \\ 0, & \text{otherwise} \end{cases}$$

where $\alpha \in (0, 1)$ is small.

Model Learning

Goal

Learn a function of the features that ranks variables with better labels higher than other variables.

Model Learning

Goal

Learn a function of the features that ranks variables with better labels higher than other variables.

- "Learning to rank with pairwise loss", popular in web search [Joachims, 2002]

Model Learning

Goal

Learn a function of the features that ranks variables with better labels higher than other variables.

- "Learning to rank with pairwise loss", popular in web search [Joachims, 2002]
- Assume the ranking function is linear in the features, i.e. $f : \mathbb{R}^p \rightarrow \mathbb{R}$, and $\mathbf{w} \in \mathbb{R}^p$:

$$f\left(\phi(x_j, N_i)\right) = \mathbf{w}^T \cdot \phi(x_j, N_i)$$

Model Learning

Goal

Learn a function of the features that ranks variables with better labels higher than other variables.

- "Learning to rank with pairwise loss", popular in web search [Joachims, 2002]
- Assume the ranking function is linear in the features, i.e. $f : \mathbb{R}^p \rightarrow \mathbb{R}$, and $\mathbf{w} \in \mathbb{R}^p$:

$$f\left(\phi(x_j, N_i)\right) = \mathbf{w}^T \cdot \phi(x_j, N_i)$$

- Convex SVM optimization problem; can be solved efficiently with cutting plane algorithm of [Joachims, 2006].

Model Learning

Goal

Learn a function of the features that ranks variables with better labels higher than other variables.

- "Learning to rank with pairwise loss", popular in web search [Joachims, 2002]
- Assume the ranking function is linear in the features, i.e. $f : \mathbb{R}^p \rightarrow \mathbb{R}$, and $\mathbf{w} \in \mathbb{R}^p$:

$$f\left(\phi(x_j, N_i)\right) = \mathbf{w}^T \cdot \phi(x_j, N_i)$$

- Convex SVM optimization problem; can be solved efficiently with cutting plane algorithm of [Joachims, 2006].
- Open-source implementation SVM^{rank}: `http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`

ML-based Branching

- After collecting data for the first θ nodes, and learning the ranking model, we start branching based on the learned model.
- No SB anymore.

ML-based Branching

- After collecting data for the first θ nodes, and learning the ranking model, we start branching based on the learned model.
- No SB anymore.
- Complexity (per node):
  - $O\big(nnz(A)\big)$ for feature computation (line 2)
  - $O(p \cdot \kappa)$ for scoring by dot product (line 3)

Algorithm: ML-based Branching

Input: node N_i, candidate set $\mathcal{C}_i, |\mathcal{C}_i| \leqslant \kappa$
Output: variable x_k s.t. $k \in \mathcal{C}_i$

```
1 for each j ∈ C_i do
2     Compute the features by φ(x_j, N_i)
3     Compute the score s_j = f(φ(x_j, N_i))
4 end
5 Branch on x_k, where s_k = max_{j∈C_i} {s_j}
```

Experimental Setup

- All algorithms implemented using control callbacks in CPLEX 12.6.1
- Cuts applied at the root, disabled afterwards
- Optimal value provided as upper cutoff
- Number of simplex iterations for any SB call is 50
- MIPLIB2010 "Benchmark" set with a time cutoff of 5 hours; default gap tolerances

Experimental Setup

- Competing Strategies:

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization
  - SB+PC ($\theta = 500, \kappa = 10$): SB for first θ nodes, PC after

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization
  - SB+PC ($\theta = 500, \kappa = 10$): SB for first θ nodes, PC after
- Other Strategies:

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization
  - SB+PC ($\theta = 500, \kappa = 10$): SB for first θ nodes, PC after
- Other Strategies:
  - CPLEX-D: enter branching callback and branch as CPLEX suggests

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization
  - SB+PC ($\theta = 500, \kappa = 10$): SB for first θ nodes, PC after
- Other Strategies:
  - CPLEX-D: enter branching callback and branch as CPLEX suggests
  - SB ($\kappa = 10$): SB at each node

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization
  - SB+PC ($\theta = 500, \kappa = 10$): SB for first θ nodes, PC after
- Other Strategies:
  - CPLEX-D: enter branching callback and branch as CPLEX suggests
  - SB ($\kappa = 10$): SB at each node

- MIPLIB2010 "Benchmark" set; 84 feasible instances, each with 10 random seeds. From the 840 instances, 523 remain after filtering (too easy, too hard, etc.)

Experimental Setup

- Competing Strategies:
  - SB+ML ($\theta = 500, \kappa = 10$): our method with $\alpha = 0.2$ and SVM parameter $C = 0.1$. Results are consistent for other values of α and C.
  - PC: pseudocost branching with SB initialization
  - SB+PC ($\theta = 500, \kappa = 10$): SB for first θ nodes, PC after
- Other Strategies:
  - CPLEX-D: enter branching callback and branch as CPLEX suggests
  - SB ($\kappa = 10$): SB at each node

- MIPLIB2010 "Benchmark" set; 84 feasible instances, each with 10 random seeds. From the 840 instances, 523 remain after filtering (too easy, too hard, etc.)
- Similar experimental procedures used in [Fischetti and Monaci, 2012, Kılınç-Karzan et al., 2009].

Results

		CPLEX-D	SB	PC	SB+PC	SB+ML
Unsolved Instances	All (523)	11	129	66	63	**52**
	Easy (255)	0	12	15	14	**13**
	Medium (120)	2	43	22	22	**17**
	Hard (148)	9	74	29	27	**22**

- An instance is "Easy" if CPLEX-D solves in $\leqslant$ 50,000 nodes, "Medium" in $\leqslant$ 500,000 nodes, "Hard" otherwise
- PC, SB+PC and SB+ML are competing, lower is better for all three criteria, and best is in bold

Results

		CPLEX-D	SB	PC	SB+PC	SB+ML
Unsolved Instances	All (523)	11	129	66	63	**52**
	Easy (255)	0	12	15	14	**13**
	Medium (120)	2	43	22	22	**17**
	Hard (148)	9	74	29	27	**22**
Num. nodes	All (523)	46,633	33,072	92,662	70,455	**59,223**
	Easy (255)	3,255	3,610	7,931	5,224	**5,124**
	Medium (120)	173,417	121,923	395,199	288,916	**234,093**
	Hard (148)	1,570,891	519,878	1,971,333	1,979,660	**1,314,263**

- An instance is "Easy" if CPLEX-D solves in $\leqslant$ 50,000 nodes, "Medium" in $\leqslant$ 500,000 nodes, "Hard" otherwise
- PC, SB+PC and SB+ML are competing, lower is better for all three criteria, and best is in bold

Results

		CPLEX-D	SB	PC	SB+PC	SB+ML
Unsolved Instances	All (523)	11	129	66	63	**52**
	Easy (255)	0	12	15	14	**13**
	Medium (120)	2	43	22	22	**17**
	Hard (148)	9	74	29	27	**22**
Num. nodes	All (523)	46,633	33,072	92,662	70,455	**59,223**
	Easy (255)	3,255	3,610	7,931	5,224	**5,124**
	Medium (120)	173,417	121,923	395,199	288,916	**234,093**
	Hard (148)	1,570,891	519,878	1,971,333	1,979,660	**1,314,263**
Total time	All (523)	499	2,263	**960**	1,093	1,059
	Easy (255)	111	602	**243**	361	382
	Medium (120)	1,123	6,169	2,493	1,892	**1,776**
	Hard (148)	3,421	9,803	4,705	4,718	**4,039**

- An instance is "Easy" if CPLEX-D solves in $\leqslant$ 50,000 nodes, "Medium" in $\leqslant$ 500,000 nodes, "Hard" otherwise
- PC, SB+PC and SB+ML are competing, lower is better for all three criteria, and best is in bold

Head-to-head Comparisons: Win/Tie/Loss

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/264/0/125/123			
PC	8/164/0/285/63	68/63/0/326/5		
SB+PC	8/227/0/225/60	72/66/7/315/6	15/320/0/125/12	
SB+ML	8/267/0/196/49	82/96/7/286/5	21/355/0/95/7	17/300/58/96/6

Table: Each cell has the numbers of: Abs. Win/Win/Tie/Loss/Abs. Loss

Compare branching strategy $\mathcal{A}$ to $\mathcal{B}$:

- *absolute win* for $\mathcal{A}$ vs. $\mathcal{B}$ on instance $\mathcal{I} \Leftrightarrow \mathcal{A}$ solves $\mathcal{I}$, $\mathcal{B}$ does not
- *win* for $\mathcal{A}$ vs. $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$, and $\mathcal{A}$ does so in fewer nodes than $\mathcal{B}$
- *tie* between $\mathcal{A}$ and $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$ in same num. of nodes

Head-to-head Comparisons: Win/Tie/Loss

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/264/0/125/123			
PC	8/164/0/285/63	68/63/0/326/5		
SB+PC	8/227/0/225/60	72/66/7/315/6	15/320/0/125/12	
SB+ML	8/267/0/196/49	82/96/7/286/5	21/355/0/95/7	17/300/58/96/6

Table: Each cell has the numbers of: Abs. Win/Win/Tie/Loss/Abs. Loss

Compare branching strategy $\mathcal{A}$ to $\mathcal{B}$:

- *absolute win* for $\mathcal{A}$ vs. $\mathcal{B}$ on instance $\mathcal{I} \Leftrightarrow \mathcal{A}$ solves $\mathcal{I}$, $\mathcal{B}$ does not
- *win* for $\mathcal{A}$ vs. $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$, and $\mathcal{A}$ does so in fewer nodes than $\mathcal{B}$
- *tie* between $\mathcal{A}$ and $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$ in same num. of nodes

Head-to-head Comparisons: Win/Tie/Loss

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/264/0/125/123			
PC	8/164/0/285/63	68/63/0/326/5		
SB+PC	8/227/0/225/60	72/66/7/315/6	15/320/0/125/12	
SB+ML	8/267/0/196/49	82/96/7/286/5	21/355/0/95/7	17/300/58/96/6

Table: Each cell has the numbers of: Abs. Win/Win/Tie/Loss/Abs. Loss

Compare branching strategy $\mathcal{A}$ to $\mathcal{B}$:

- *absolute win* for $\mathcal{A}$ vs. $\mathcal{B}$ on instance $\mathcal{I} \Leftrightarrow \mathcal{A}$ solves $\mathcal{I}$, $\mathcal{B}$ does not
- *win* for $\mathcal{A}$ vs. $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$, and $\mathcal{A}$ does so in fewer nodes than $\mathcal{B}$
- *tie* between $\mathcal{A}$ and $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$ in same num. of nodes

Head-to-head Comparisons: Win/Tie/Loss

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/264/0/125/123			
PC	8/164/0/285/63	68/63/0/326/5		
SB+PC	8/227/0/225/60	72/66/7/315/6	15/320/0/125/12	
SB+ML	8/267/0/196/49	82/96/7/286/5	21/355/0/95/7	17/300/58/96/6

Table: Each cell has the numbers of: Abs. Win/Win/Tie/Loss/Abs. Loss

Compare branching strategy $\mathcal{A}$ to $\mathcal{B}$:

- *absolute win* for $\mathcal{A}$ vs. $\mathcal{B}$ on instance $\mathcal{I} \Leftrightarrow \mathcal{A}$ solves $\mathcal{I}$, $\mathcal{B}$ does not
- *win* for $\mathcal{A}$ vs. $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$, and $\mathcal{A}$ does so in fewer nodes than $\mathcal{B}$
- *tie* between $\mathcal{A}$ and $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$ in same num. of nodes

Head-to-head Comparisons: Win/Tie/Loss

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/264/0/125/123			
PC	8/164/0/285/63	68/63/0/326/5		
SB+PC	8/227/0/225/60	72/66/7/315/6	15/320/0/125/12	
SB+ML	8/267/0/196/49	82/96/7/286/5	21/355/0/95/7	17/300/58/96/6

Table: Each cell has the numbers of: Abs. Win/Win/Tie/Loss/Abs. Loss

Compare branching strategy $\mathcal{A}$ to $\mathcal{B}$:

- *absolute win* for $\mathcal{A}$ vs. $\mathcal{B}$ on instance $\mathcal{I} \Leftrightarrow \mathcal{A}$ solves $\mathcal{I}$, $\mathcal{B}$ does not
- *win* for $\mathcal{A}$ vs. $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$, and $\mathcal{A}$ does so in fewer nodes than $\mathcal{B}$
- *tie* between $\mathcal{A}$ and $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$ in same num. of nodes

Future Directions

- Adaptive learning w.r.t. the evolution of the search

Future Directions

- Adaptive learning w.r.t. the evolution of the search
- Fully online or offline learning

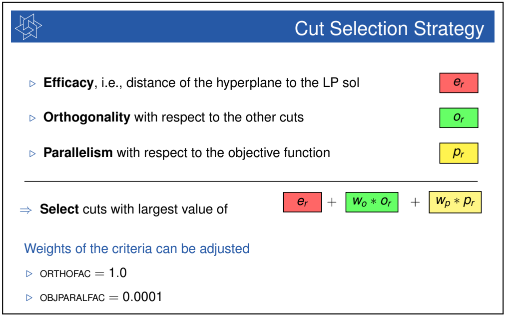
Future Directions

- Adaptive learning w.r.t. the evolution of the search
- Fully online or offline learning
- Most decisions in the MIP solver are inherently heuristic

Future Directions

- Adaptive learning w.r.t. the evolution of the search
- Fully online or offline learning
- Most decisions in the MIP solver are inherently heuristic
  - Can we use data and ML to make better informed decisions?

Figure: Cut Selection Strategy in SCIP (slide from [Wolter, 2006])



The End

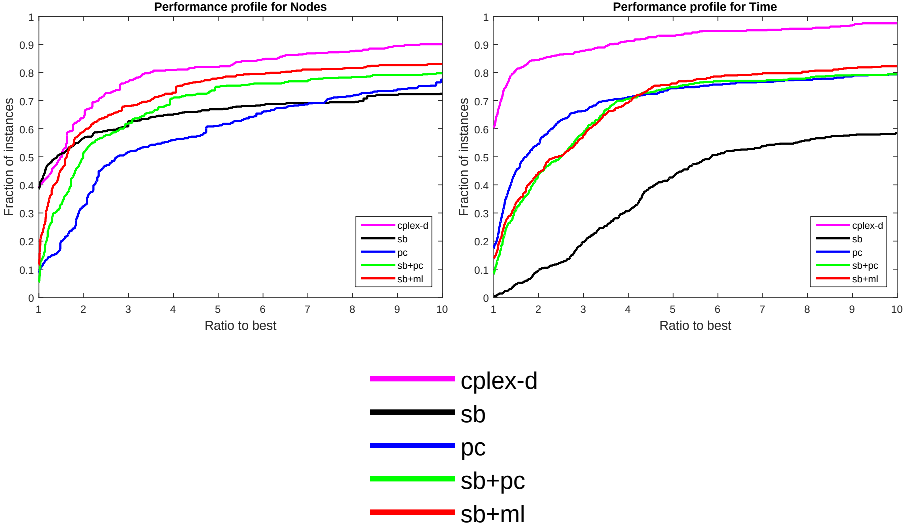
Thanks!
Questions?

Elias Khalil
elias.khalil@cc.gatech.edu

Features Description

Feature	Description	Count	Reference
Static Features (18)			
Objective function coeffs.	Value of the coefficient (raw, positive only, negative only)	3	
Num. constraints	Number of constraints that the variable participates in (with a non-zero coefficient)	1	
Stats. for constraint degrees	The *degree of a constraint* is the number of variables that participate in it. A variable may participate in multiple constraints, and statistics over those constraints' degrees are used. The constraint degree is computed on the root LP (mean, stdev., min, max)	4	
Stats. for constraint coeffs.	A variable's positive (negative) coefficients in the constraints it participates in (count, mean, stdev., min, max)	10	
Dynamic Features (54)			
Slack and ceil distances	$\min\{\breve{x}^i_j - \lfloor \breve{x}^i_j \rfloor, \lceil \breve{x}^i_j \rceil - \breve{x}^i_j\}$ and $\lceil \breve{x}^i_j \rceil - \breve{x}^i_j$	2	
Pseudocosts	Upwards and downwards values, and their corresponding ratio, sum and product, weighted by the fractionality of x_j	5	(Achterberg 2009)
Infeasibility statistics	Number and fraction of nodes for which applying SB to variable x_j led to one (two) infeasible children (during data collection)	4	
Stats. for constraint degrees	A dynamic variant of the static version above. Here, the constraint degrees are on the current node's LP. The ratios of the static mean, maximum and minimum to their dynamic counterparts are also features	7	
Min/max for ratios of constraint coeffs. to RHS	Minimum and maximum ratios across positive and negative right-hand-sides (RHS)	4	(Alvarez, Louveaux, and Wehenkel 2014)
Min/max for one-to-all coefficient ratios	The statistics are over the ratios of a variable's coefficient, to the sum over all other variables' coefficients, for a given constraint. Four versions of these ratios are considered: positive (negative) coefficient to sum of positive (negative) coefficients	8	(Alvarez, Louveaux, and Wehenkel 2014)
Stats. for active constraint coefficients	An active constraint at a node LP is one which is binding with equality at the optimum. We consider 4 weighting schemes for an active constraint: unit weight, inverse of the sum of the coefficients of all variables in constraint, inverse of the sum of the coefficients of only candidate variables in constraint, dual cost of the constraint. Given the absolute value of the coefficients of x_j in the active constraints, we compute the sum, mean, stdev., max. and min. of those values, for each of the weighting schemes. We also compute the weighted number of active constraints that x_j is in, with the same 4 weightings	24	(Patel and Chinneck 2007)

Performance Profiles



Head-to-head Comparisons: Node Ratios

	CPLEX-D	SB	PC	SB+PC	SB+ML
CPLEX-D		1.39 (389)	0.64 (449)	0.84 (452)	0.97 (463)
SB	0.72 (389)		0.47 (389)	0.61 (388)	0.76 (389)
PC	1.56 (449)	2.11 (389)		1.34 (445)	1.59 (450)
SB+PC	1.20 (452)	1.63 (388)	0.75 (445)		1.22 (454)
SB+ML	1.03 (463)	1.32 (389)	0.63 (450)	0.82 (454)	

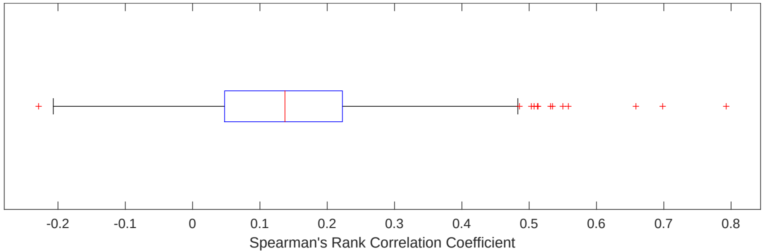
Head-to-head Comparisons: Win/Tie/Loss (averaging over seeds)

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	45/0/20			
PC	20/0/45	5/0/60		
SB+PC	28/0/37	8/0/57	51/0/14	
SB+ML	33/0/32	10/0/55	55/0/10	46/6/13

Table: Win-tie-loss counts for every pair of strategies, for the number of nodes. A triplet in a cell in row $\mathcal{A}$ and column $\mathcal{B}$ expresses the number of wins for $\mathcal{A}$ over $\mathcal{B}$, the number of ties, and the number of losses of $\mathcal{A}$ to $\mathcal{B}$, respectively, w.r.t. to the shifted geometric mean of the number of nodes. $\mathcal{A}$ wins over $\mathcal{B}$ on instance $\mathcal{I}$ iff the mean number of nodes of $\mathcal{A}$ over the random seeds on $\mathcal{I}$ is strictly less than that of $\mathcal{B}$; losses and ties are defined analogously.

Analysis of Feature Ranking Models

- Compare ranking models across instances
- For each model, consider the ranking of the features by their absolute weight in the linear model
- Similarity of two models measured by *Spearman's rank correlation coefficient* between their respective feature weight rankings



Analysis of Feature Ranking Models

- The 10 features that appear the most in the top $K = 10$ features over models, sorted by that count (corresponding column is bold in the header). The counts for other values of K are also shown. There are 2700 features in total. Static features are marked by (S); unmarked features are dynamic.

Feature	1	3	5	**10**	20	100
PC Product	1	10	17	23	30	37
PC Product x PC Product	6	10	11	17	24	38
PC Product x Mean Constraint Degree	1	2	5	13	19	33
PC Product x Max. Constraint Degree	0	0	3	13	19	34
PC Product x Min. Constraint Degree	0	3	6	13	21	34
PC Product x Max. Absolute Coefficient in Active Constraints	2	7	8	13	27	37
PC Product x Mean Absolute Coefficient in Active Constraints	2	4	4	10	25	35
PC Product x Num. Constraints (S)	0	3	5	9	17	28
PC Product x Min. Positive Constraint Coefficient (S)	1	1	2	9	18	32
PC Product x Max. Ratio of Constraint Degree (static/dynamic)	0	3	4	9	16	34

Results **with cuts and heuristics, no upper cutoff**

- "Easy" if CPLEX solves in $\leqslant$ 50,000 nodes, "Medium" in $\leqslant$ 500,000 nodes, "Hard" otherwise
- PC, SB+PC and SB+ML are competing, lower is better for all three criteria, and best is in bold

		CPLEX-D	SB	PC	SB+PC	SB+ML
Instances unsolved	All (538)	28	139	55	59	**54**
	Easy (318)	3	23	21	22	**20**
	Medium (140)	18	74	**24**	26	27
	Hard (80)	**7**	42	10	11	**7**
Num. nodes	All (538)	25,407	17,909	43,120	36,235	**32,948**
	Easy (318)	3,643	3,794	7,371	5,386	**5,358**
	Medium (140)	165,188	91,279	260,851	266,497	**217,306**
	Hard (80)	2,144,179	491,557	2,062,439	2,142,617	**1,649,930**
Total time	All (538)	681	2,273	**1,027**	1,229	1,423
	Easy (318)	203	821	**385**	498	630
	Medium (140)	2,978	9,942	**3,767**	4,077	4,210
	Hard (80)	6,223	9,815	**5,192**	5,461	5,423

Head-to-head Comparisons: Win/Tie/Loss

	CPLEX-D	SB	PC	SB+PC
CPLEX-D				
SB	5/259/0/135/116			
PC	20/183/0/280/47	98/60/0/325/14		
SB+PC	21/216/0/242/52	94/96/1/288/14	22/264/0/193/26	
SB+ML	22/231/0/231/48	98/108/1/277/13	23/313/0/148/22	24/260/55/145/19

Definitions:

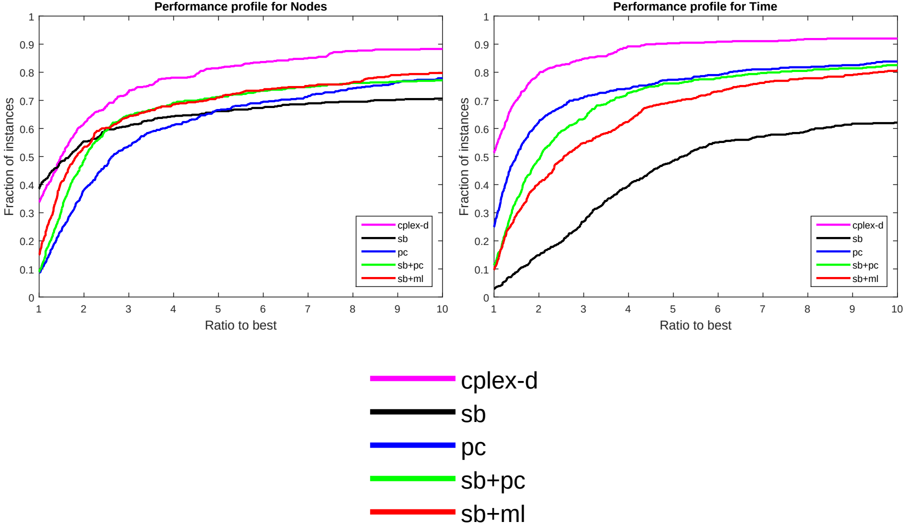
- *absolute win* for $\mathcal{A}$ v/s $\mathcal{B}$ on instance $\mathcal{I} \Leftrightarrow \mathcal{A}$ solves $\mathcal{I}$, $\mathcal{B}$ does not
- *win* for $\mathcal{A}$ v/s $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$, and $\mathcal{A}$ does so in $<$ nodes than $\mathcal{B}$
- *tie* between $\mathcal{A}$ and $\mathcal{B}$ on $\mathcal{I} \Leftrightarrow \mathcal{A}$ and $\mathcal{B}$ solve $\mathcal{I}$ in same num. of nodes

Head-to-head Comparisons: Node Ratios

	CPLEX-D	SB	PC	SB+PC	SB+ML
CPLEX-D		1.38 (394)	0.64 (463)	0.79 (458)	0.84 (462)
SB	0.72 (394)		0.48 (385)	0.65 (385)	0.67 (386)
PC	1.56 (463)	2.09 (385)		1.24 (457)	1.32 (461)
SB+PC	1.27 (458)	1.55 (385)	0.81 (457)		1.08 (460)
SB+ML	1.20 (462)	1.49 (386)	0.76 (461)	0.92 (460)	

Table: Ratios for the shifted geometric means (shift 10) over nodes on instances solved by both strategies. The first value in a cell in row $\mathcal{A}$ and column $\mathcal{B}$ is the ratio of the average number of nodes used by $\mathcal{A}$ to that of $\mathcal{B}$. The second value is the number of instances solved by both $\mathcal{A}$ and $\mathcal{B}$.

Performance Profiles



Pseudocost Branching (PC)

- Goal: Imitate SB without many simplex iterations.
- Record change in objective value over time, and use historical averages as proxy for SB scores.
- Objective increase per unit change in x_j at N_i when branching down:

$$\varsigma_{i,j}^{-} = \frac{\Delta_j^{-}}{f_j^{-}}$$

with $f_j^{-} = \check{x}_j^i - \lfloor \check{x}_j^i \rfloor$

- Take average over those events :

$$\Psi_j^{-} = \frac{\sum \varsigma_{i,j}^{-}}{\#\text{ times branched down on } x_j}$$

- PC score at some node:

$$s_j = \mathrm{score}\big(f_j^{-}\Psi_j^{-}, f_j^{+}\Psi_j^{+}\big)$$

References I

Achterberg, T. (2009).
Constraint Integer Programming.
PhD thesis, Technische Universität Berlin.

Achterberg, T. and Berthold, T. (2009).
Hybrid branching.
In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 309–311. Springer.

Alvarez, A. M., Louveaux, Q., and Wehenkel, L. (2014).
A supervised machine learning approach to variable branching in branch-and-bound.
Technical Report, Université de Liège.

Fischetti, M. and Monaci, M. (2012).
Branching on nonchimerical fractionalities.
Operations Research Letters, 40(3):159–164.

He, H., Daumé III, H., and Eisner, J. (2014).
Learning to search in branch-and-bound algorithms.
In *Advances in Neural Information Processing Systems*.

References II

Hendel, G. (2015).
Enhancing MIP branching decisions by using the sample variance of pseudo costs.
In *Integration of AI and OR Techniques in Constraint Programming*, volume 9075, pages 199 – 214.

Joachims, T. (2002).
Optimizing search engines using clickthrough data.
In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142.

Joachims, T. (2006).
Training linear SVMs in linear time.
In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226.

Kılınç-Karzan, F., Nemhauser, G. L., and Savelsbergh, M. W. (2009).
Information-based branching schemes for binary linear mixed integer problems.
Mathematical Programming Computation, 1(4):249–293.

Patel, J. and Chinneck, J. W. (2007).
Active-constraint variable ordering for faster feasibility of mixed integer linear programs.
Mathematical Programming, 110(3):445–474.

References III

Wolter, K. (2006).
Cutting plane separators in SCIP.

Related Work

- [Achterberg and Berthold, 2009],[Hendel, 2015]: recent PC-based strategies
- [Kılınç-Karzan, Savelsbergh and Nemhauser, 2009]: collect information on which variables are likely to lead to fathomed child nodes quickly, then restart solve and branch on those
- [He et al., 2014]: ML (classification) for node selection; offline, can prune optimum
- [Alvarez et al., 2014]: ML (regression) for variable selection; offline, slow and modest results