AI/ML Approaches to Mixed-Integer Programming

Selin Bayramoglu[1], George Nemhauser[1], **Nick Sahinidis**[1,2]

[1] H. Milton School of Industrial and Systems Engineering, Georgia Institute of Technology

[2] School of Chemical & Biomolecular Engineering, Georgia Institute of Technology

AI/ML AND OPTIMIZATION

- **Optimization for AIML**
  - Steepest descent
  - Cyclic coordinate search

- **AI/ML for optimization**
  - Use AI/ML to accelerate optimization algorithms
  - Systematize heuristics for tuning, customizing, adapting optimization algorithms

AI/ML FOR (INTEGER) OPTIMIZATION

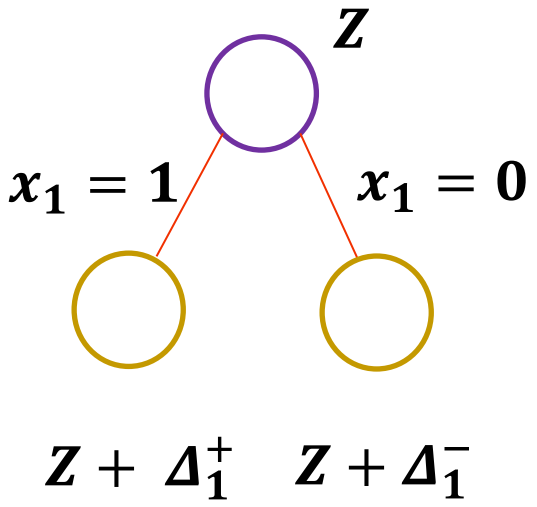
- **Algorithm tuning**
  - Decision to linearize MIQPs for CPLEX (Bonami et al., 2018)
  - Partitioning variable domains in solving QCQPs (Kannan et al., 2023)

- **Instance-specific learning**
  - First perform target (expensive) branching strategy and collect data, build a model and continue solving with the learned strategy (Khalil et al., 2016)

- **Offline learning**
  - Predicting good initial feasible solutions and redundant constraints **for a family of problems** (Xavier et al., 2021)

BRANCHING IN INTEGER PROGRAMMING

- **Pseudocost branching (Benichou et al., 1971)**
- **Strong branching (SB) (Applegate et al., 1995)**
  - Solves two LPs for each fractional binary at a node!
- **Reliability branching (Achterberg et al., 2005)**
  - Reliable pseudocosts
- **Hybrid branching (RPB) (Achterberg and Berthold, 2009)**
  - Single score that combines pseudocost scores, inference values, number of cutoffs, etc.



$$\text{SB score}_1 = f(\Delta_1^+, \Delta_1^-)$$

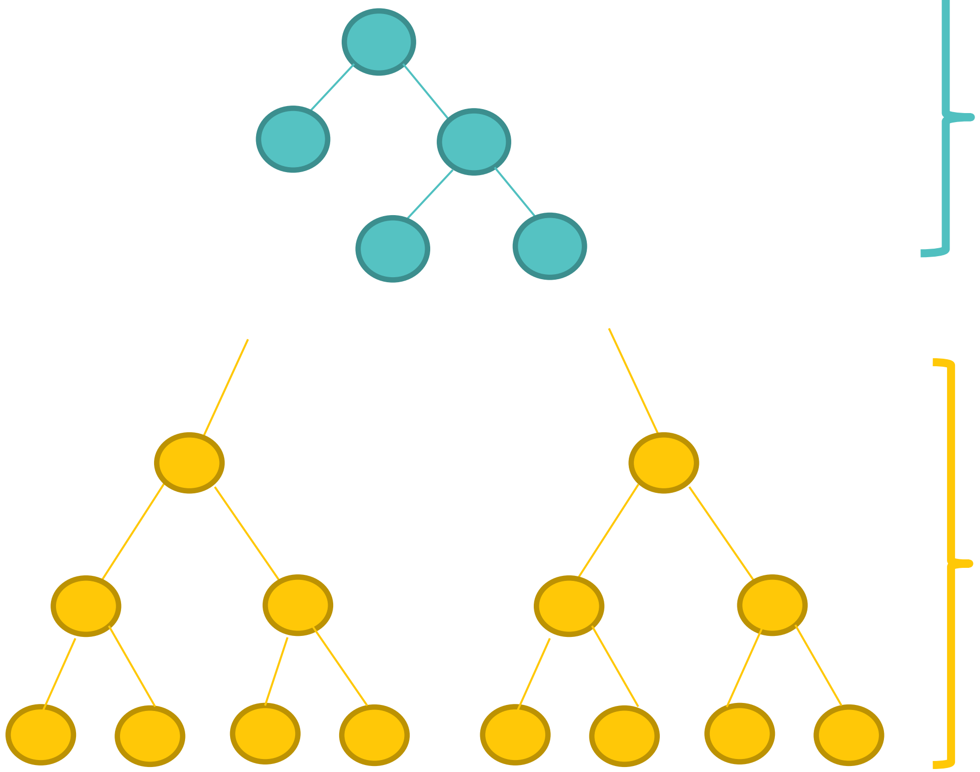
ML FOR BRANCHING

- **Studies on learning to branch**
  - Ranking by SB scores, **SVM^{rank}** (Khalil et al. 2016)
  - SB scores, **ExtraTrees** (Alvarez et al. 2014, 2017)
  - Selecting the best SB candidate, **graph neural network** (Gasse et al. 2019, Nair et al. 2020, Gupta et al. 2022)
  - Ranking by default rule (RPB), **deep neural network** (Zarpellon et al. 2021)

- **Theoretical results**
  - Balcan et al. (2017) use ML to find an optimal weighting of branching scores given an input problem distribution

INSTANCE SPECIFIC LEARNING

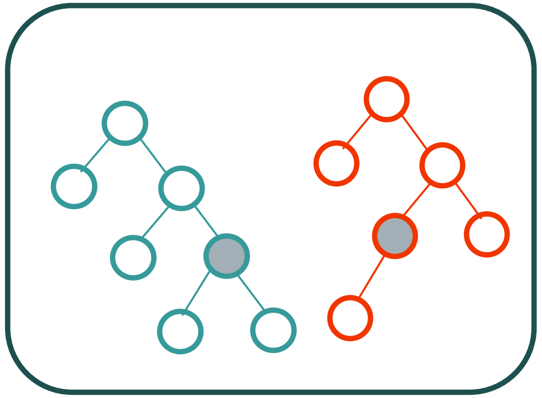
Khalil et al. (2016)

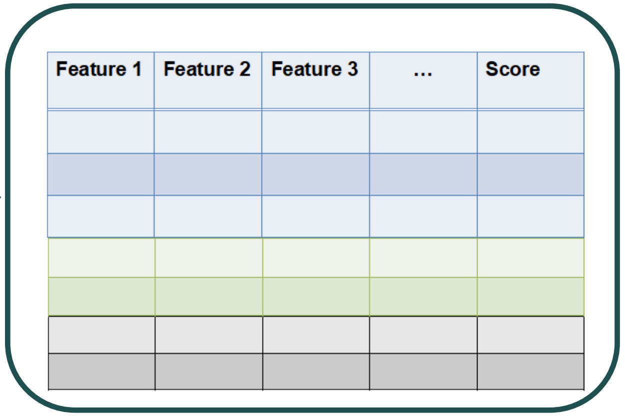


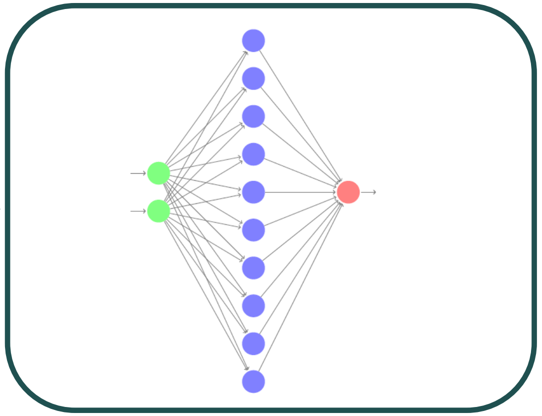
Learn branching rule

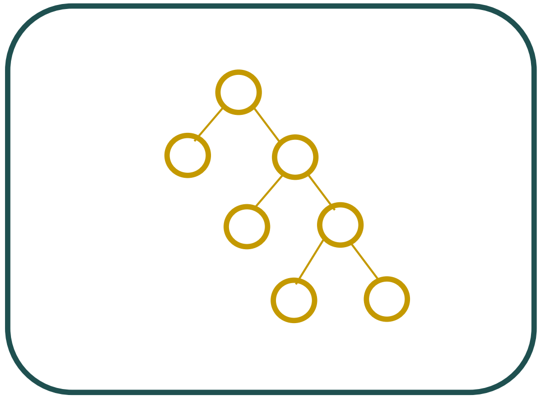
Apply learned rule

OFFLINE LEARNING









Collect data by solving many **similar problems** with strong branching

Create datasets

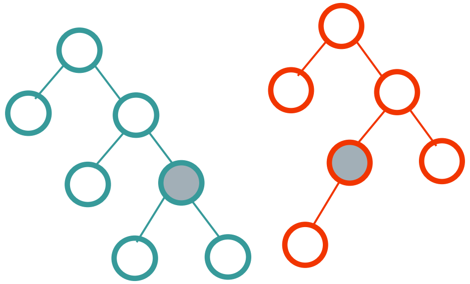
Build a model of strong branching scores

$$\text{Score} \approx f(\text{Feature}_1, \text{Feature}_2, \text{Feature}_3, \ldots)$$

Solve problems from the **same family** with the new rule

PROPOSED APPROACH

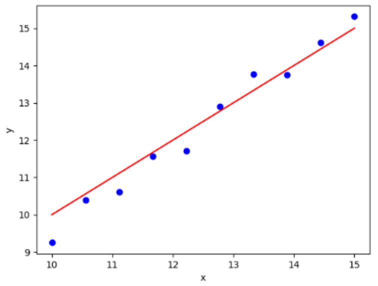
Sparse machine learning models based on the LASSO, L0L1 and L0L2



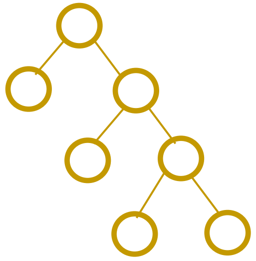
Collect data by solving many similar problems with strong branching

Feature 1	Feature 2	Feature 3	...	Score

Create datasets



Build a model of strong branching scores
Lasso, L0-regularized models



Solve problems from the same family with the new rule

Score $\approx$ $\boldsymbol{\beta_1}$Solution value
+ $\boldsymbol{\beta_2}$Objective coefficient
+ $\boldsymbol{\beta_3}$Number of rows the variable is in + …

MAIN RESULTS

- Regularized linear regression-based branching rules speed up SCIP
- Training advantages in comparison to neural networks
  - Short training times
  - Perform well even when a fraction of the data is used for training
- No need for a GPU for training or deployment

FEATURES

Features from Khalil et al. (2016) and Gasse et al. (2019)

- Static features
  - *Objective function coefficient of a candidate*
  - *Number of constraints the candidate is in*
- Dynamic features
  - *Solution point of the current node's LP relaxation*
  - *Solution infeasibility (most infeasible branching)*
  - *Mean, minimum and maximum of the dual values for each constraint the candidate is in*
  - *Up/down pseudocosts of the candidate, their weighted sum and product (hybrid branching, pseudocost branching)*

- **Feature engineering**
  - Quadratic transformations

SPARSE REGRESSION

- Sparse models are solutions to

$$\hat{\beta} \in \underset{\beta \in \mathbb{R}^p}{\arg\min} \quad \frac{1}{2}||y - X\beta||_2^2 + \lambda_0||\beta||_0 + \lambda_q||\beta||_q^q$$

(y is the score vector, X is the training dataset)

- Penalizing number of nonzero coefficients and the norm of the solution vector

- The LASSO $\lambda_0 = 0, \lambda_1 > 0$ *glmnet* (Friedman et al., 2010)
- L0L1 model $\lambda_0 > 0, \lambda_1 > 0$ *l0learn* (Hazimeh et al., 2022)
- L0L2 model $\lambda_0 > 0, \lambda_2 > 0$

Tibshirani (1995), Hazimeh and Mazumder (2020)

COMPUTATIONAL SETTING

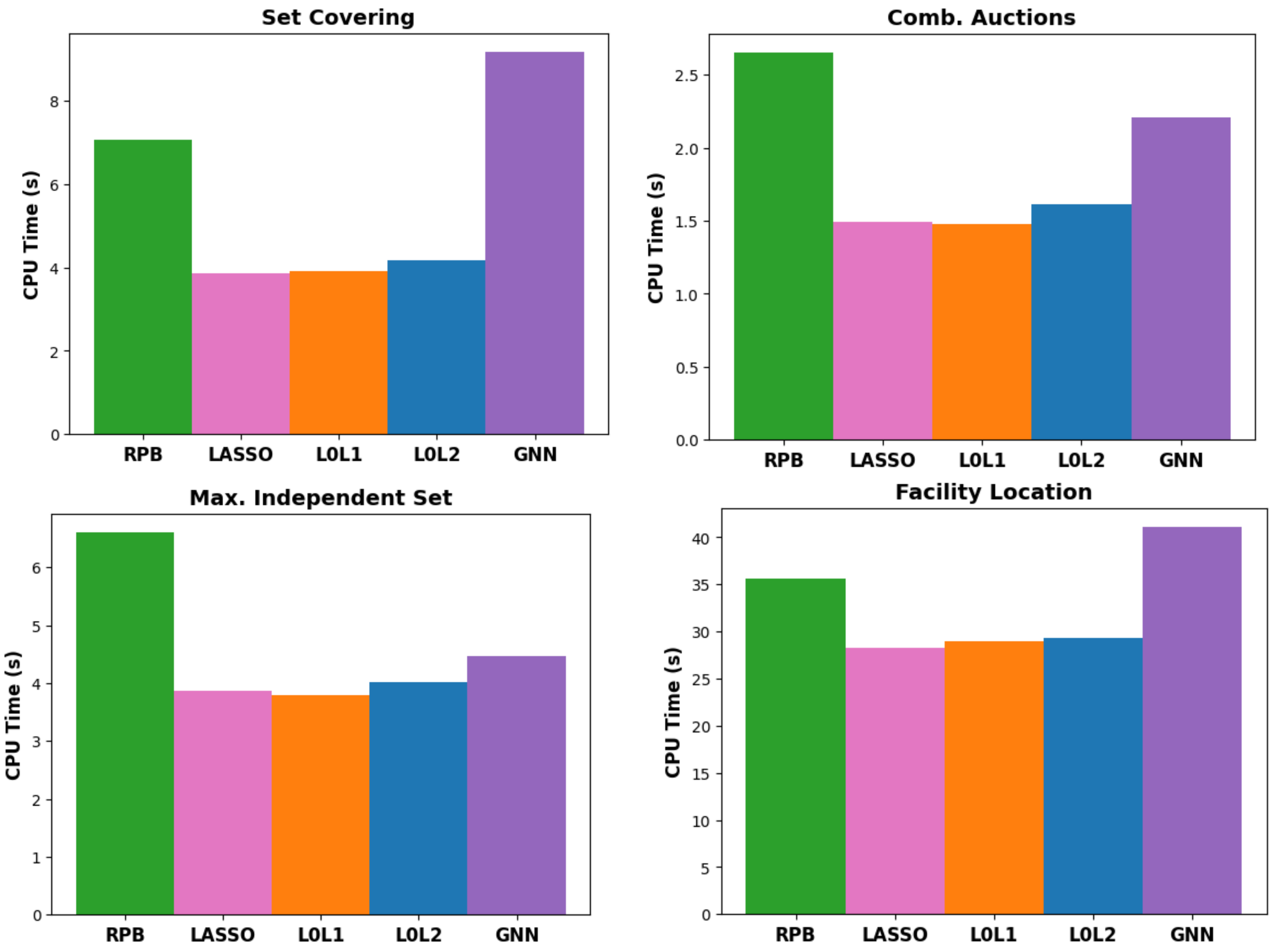
Problem	Small	Medium	Large
Set Covering (# rows, # cols)	500, 1000	1000, 1000	2000, 1000
Comb. Auctions (# items, # bids)	100, 500	200, 1000	300, 1500
Max. Independent Set (# nodes, affinity)	500, 4	1000, 4	1500, 4
Facility Location (# customers, # facilities)	100, 100	200, 100	400, 100

Gasse et al., 2019

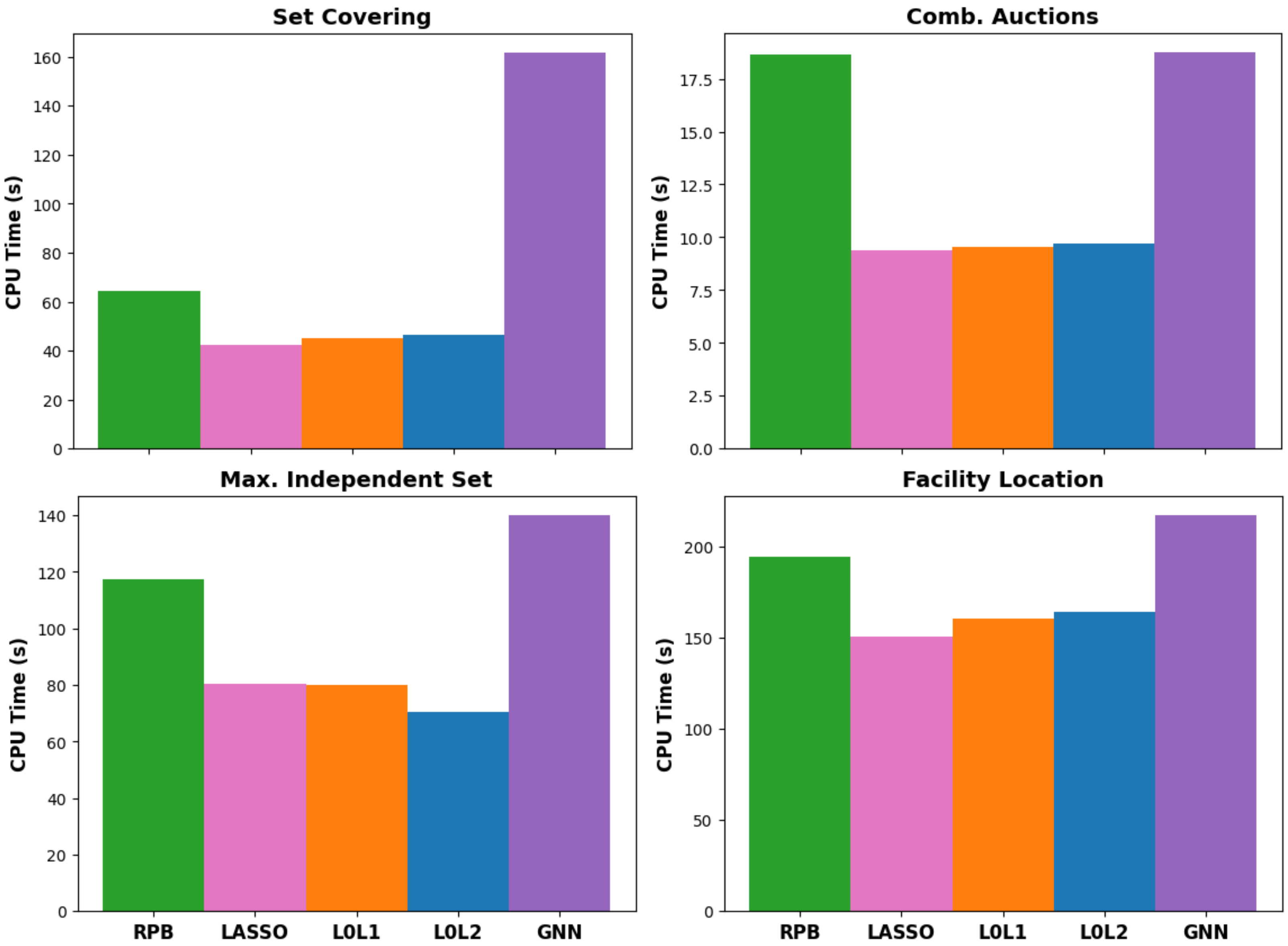
ML MODEL SIZES

Model	Number of parameters across all models
LASSO	< 2000
L0L1	$\leq$ 50
L0L2	$\leq$ 50
GNN	64,000

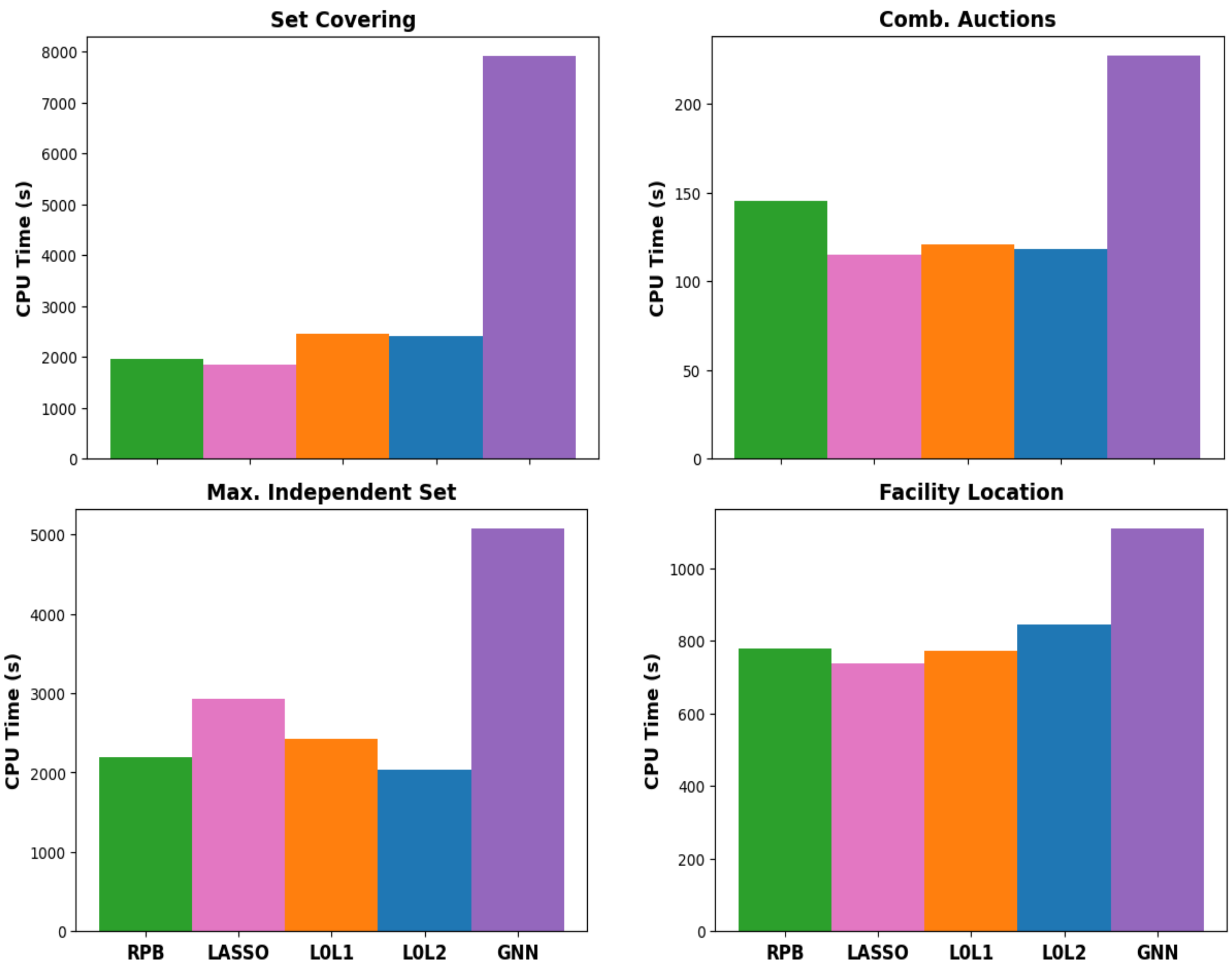
DEPLOYMENT TIMES FOR SMALL INSTANCES



DEPLOYMENT TIMES FOR MEDIUM INSTANCES



DEPLOYMENT TIMES FOR LARGE INSTANCES



EFFECTIVE SAMPLING

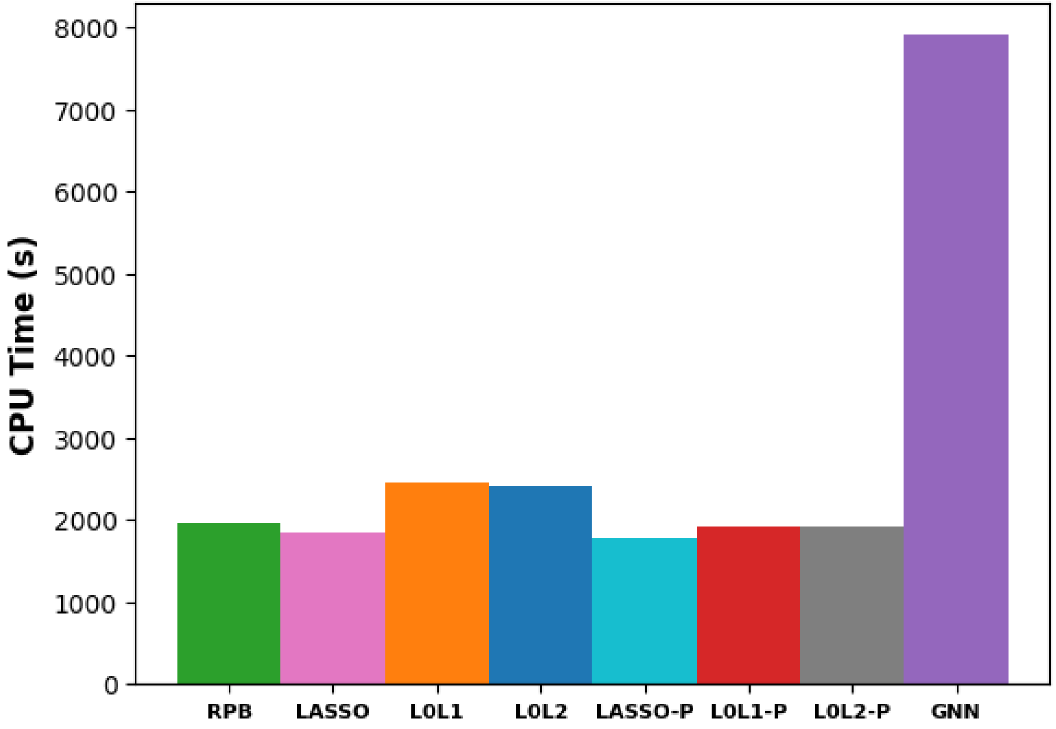
- **Models with fewer parameters can be trained with a smaller sample size**
  - Solve instances and collect candidate data until we accumulate **25K** observations in the training and validation datasets
  - GNN literature utilized **120K** observations

- **Training on the relevant input size can be more effective**
  - Train and test on instances of the **same size**

- **Models trained with this scheme**
  - LASSO-P, L0L1-P and L0L2-P

LARGE SET COVERING PROBLEMS

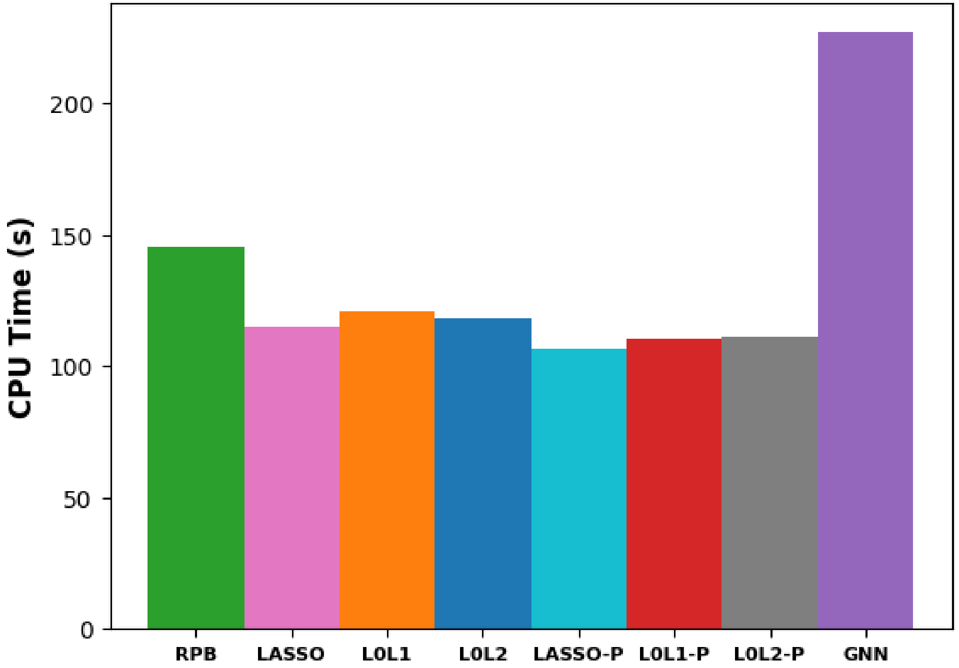
LASSO-P performs the best in terms of solving time (9% faster than RPB)



GNN trained on small problems

LARGE COMBINATORIAL AUCTIONS PROBLEMS

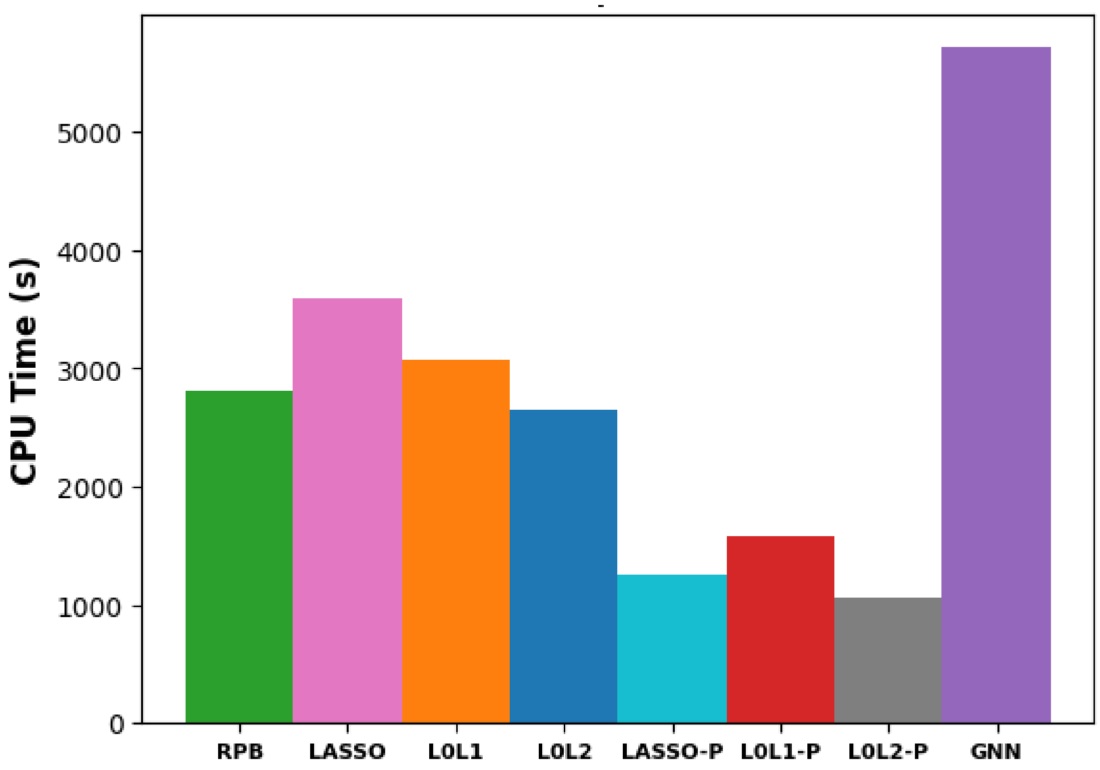
LASSO-P solves instances 27% faster than RPB



GNN trained on small problems

LARGE MAXIMUM INDEPENDENT SET PROBLEMS

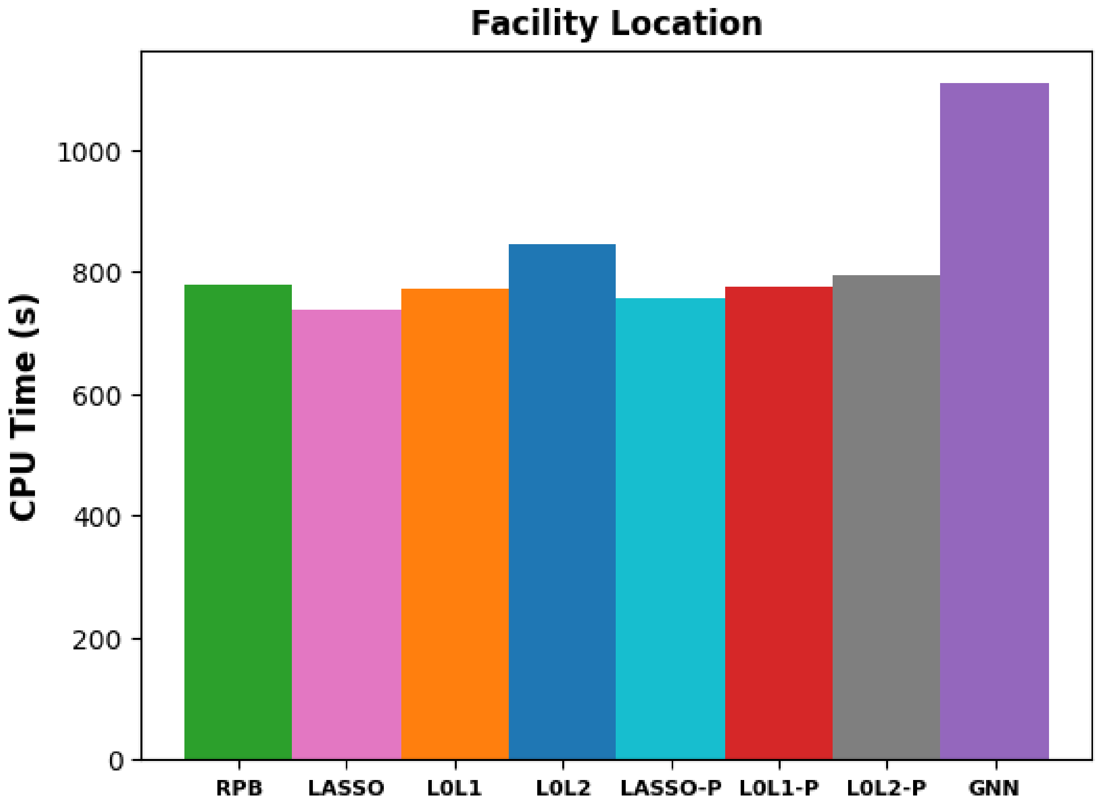
L0L2-P reduces solving time by 81% compared to RPB



GNN trained on small problems

LARGE FACILITY LOCATION PROBLEMS

LASSO solves instances on average 5% faster than RPB



GNN trained on small problems

TRAINING TIMES

Average training time of the GNN and the sparse models in hours

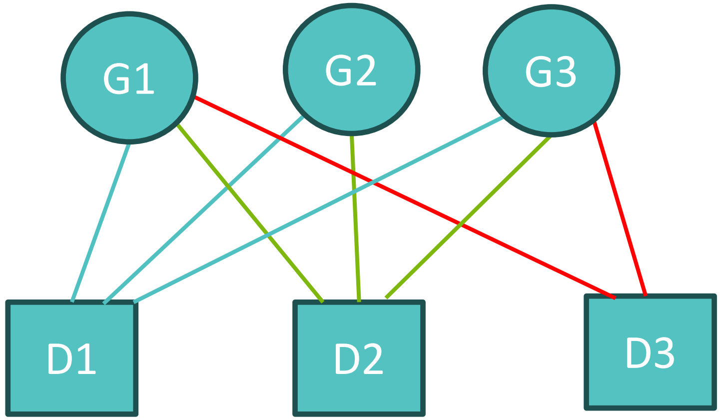
	Small-instance sampling			Effective Sampling			
	LASSO	L0L1	L0L2	LASSO-P	L0L1-P	L0L2-P	GNN
Set Covering	0.20	1.09	0.76	0.02	0.10	0.07	6.75
Combinatorial Auctions	0.21	1.14	0.72	0.04	0.11	0.07	1.37
Facility Location	0.18	1.01	0.59	0.03	0.10	0.09	8.73
Max. Independent Set	0.27	0.58	0.35	0.03	0.07	0.04	1.23

MINLP FOR AC-NETWORK CONSTRAINED UC

- Base instance from minlp.org, contributed by Anjos and Conejo (2020)
- Six-node network with three generator nodes and three demand nodes

800	500	250	Start-up cost ($)
5	15	30	Variable production cost ($/MWh)
[80, 300]	[50, 200]	[30, 100]	Active power output limits (MW)
[-150, 150]	[-100, 100]	[-50, 50]	Reactive power output limits (VAr)

Generators



Demands

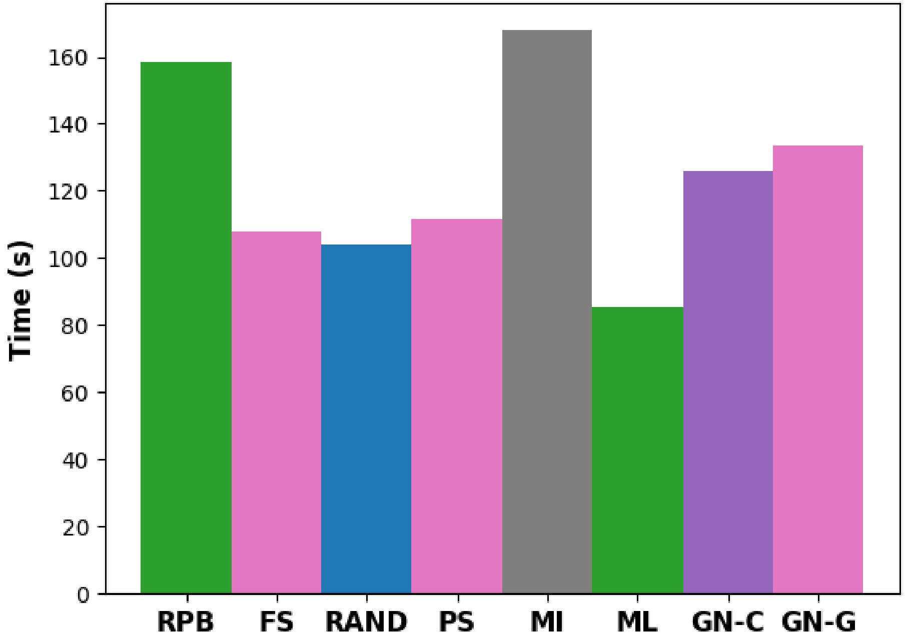
Apparent power capacity of lines (VA)
AC power flow equations
Voltage limits at every node

100	90	50	Demand for active power at t = 1 (MW)
75	67.5	37.5	Demand for reactive power at t = 1 (VAr)

- Generate instances by varying the startup cost of G1 in [720, 880]

EVALUATION

Optimality gap limit of 5% and time limit of 1 hour of CPU time



RPB:	**Default SCIP**
FS:	**Full strong branching**
RAND:	**Random**
PS:	**Pseudocost branching**
MI:	**Most infeasible**
ML:	**Lasso-based branching**
GN-C:	**CPU-based graph neural network**
GN-G:	**GPU-based graph neural network**

CONCLUSIONS

- **Sparse ML models**
  - Speed up SCIP
  - Faster than a state-of-the-art ML rule, the GNN, on a CPU-only machine
  - Do not require GPUs
  - Work with small sets of measurements
  - Rapid training

- **Understand why certain features are selected in the models**