Mixed-Integer Linear Programs

Academy of Mathematics and Systems Sciences, CAS

May 16, 2023

Contents

Mixed-Integer Linear Programs (MIP)

- A mixed-integer linear program (MIP) has the form:

$$\text{Minimize} \quad c^T x$$
$$\text{Subject to } Ax \le b$$
$$x_i \in \mathbb{Z}\ \forall i \in I$$

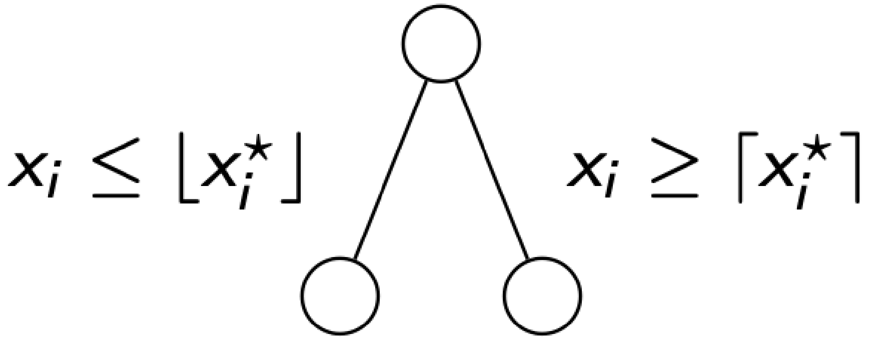
  over variables $x \in \mathbb{R}^n$, where $A \in \mathbb{R}^{m \times n}$,$b \in \mathbb{R}^m$,$c \in \mathbb{R}^n$, and $I \subset \{1, \cdots, n\}$ refers to the index set of integer variables.

- Many solvers are available to get the exact result.
- However, these methods usually rely on some heuristic process with high computational costs.
- If we remove the integer constraints in the problem then it becomes a linear program (LP), which is convex and can be solved efficiently:

$$\text{Minimize} \quad c^T x$$
$$\text{Subject to } Ax \le b$$

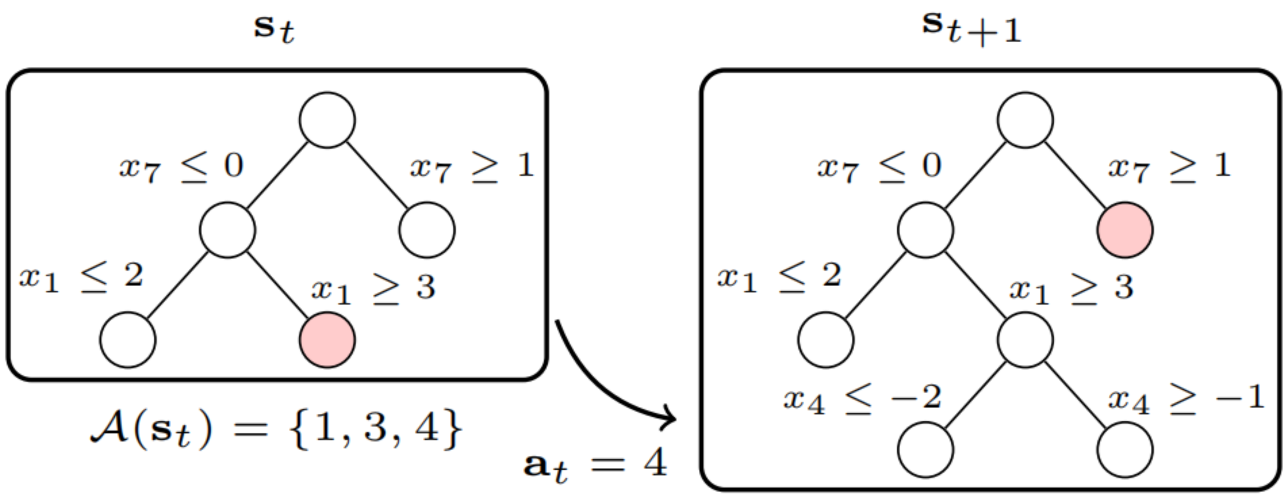
Branch-and-Bound (B&B)

- **Key idea:** Split the LP recursively over a non-integral variable.
- **Step 0:** get a feasible solution x and denote the $c^T x$ as the upper bound.
  - Denote $-\infty$ as the lower bound.
- **Step 1:** solve the relaxation LP (suppose that we get x^*). Denote $c^T x^*$ as the lower bound.
  - If x^* is feasible for MIP: renew upper bound by $c^T x^*$.
  - Else $\exists i \in I, x_i^* \notin \mathbb{Z}$, then split the LP by adding an additional constraint.



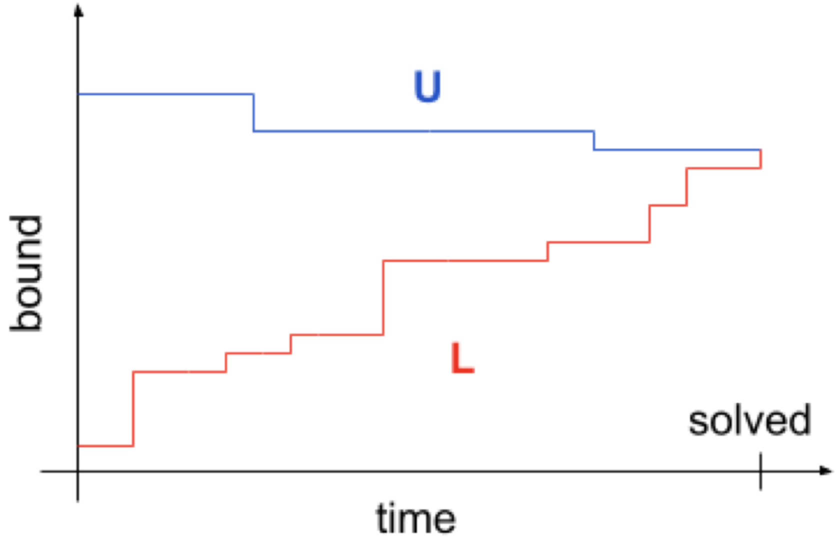
Branch-and-Bound (B&B)

- **Step 2:** Choose an unreached leaf node and solve the LP problem of this node. Renew lower bound and upper bound as in Step1. Add new leaf nodes if
  - 1. The feasible region of LP is not empty.
  - 2. The value of the target at the optimal point is not bigger than the upper bound.
  - 3. The optimal point is not feasible for MIP.
- Iteratively do Step 2 till reach the stopping criterion.



Branch-and-Bound (B&B)

- **Stopping criterion:**
  - lower bound=upper bound (optimality certificate)
  - lower bound=∞ (infeasibility certificate)
  - lower bound-upper bound<threshould (early stopping)



- **Question:**
  - How to get a feasible solution in Step 0?
    - primal heuristics
  - How to select a node and a branch in Step 2?
    - heuristic branch strategies——state of art: greedy strategy (expensive)

Primal Heuristics

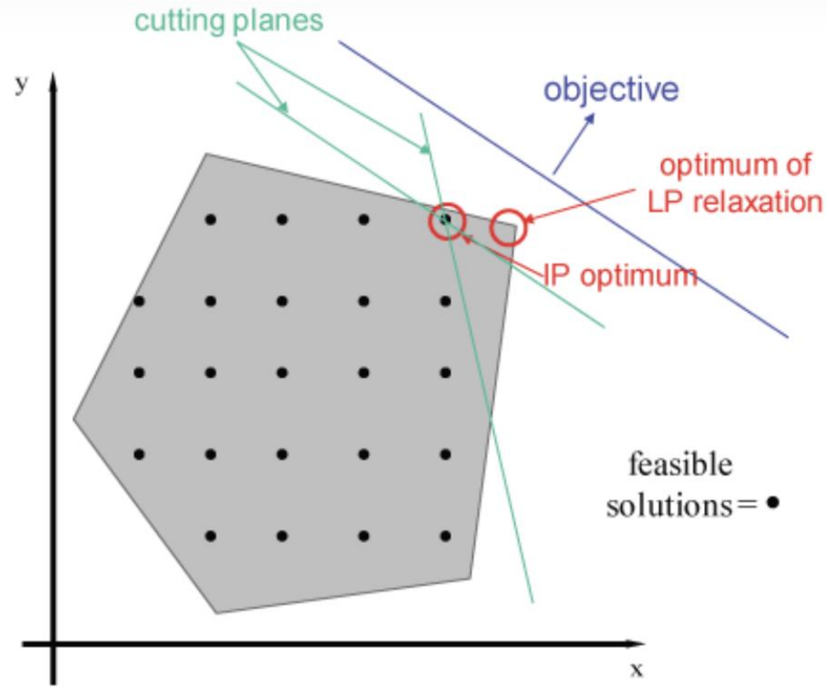
- A primal heuristic is a method that attempts to find a feasible, but not necessarily optimal, variable assignment [Berthold (2006)].
- Examples:
  - Simple rounding
  - Diving (depth-first-search in B&B)
- **Note:**
  - Any such feasible assignment provides a guaranteed upper bound on the optimal value of the MIP.
  - Any such bound found at any point during a MIP solve is called a primal bound (upper bound).

Cutting Plane Method

- **Observation:** relaxation adds infeasible region.
- **Key idea:** Gradually adding new constraints into the relaxation to kill this redundancy.

- Typical cut types:
  - Cover Cut
  - Gomory Cut
  - Clique Cut

- How to find those “good” cuts?
  - Greedy strategy (expensive)



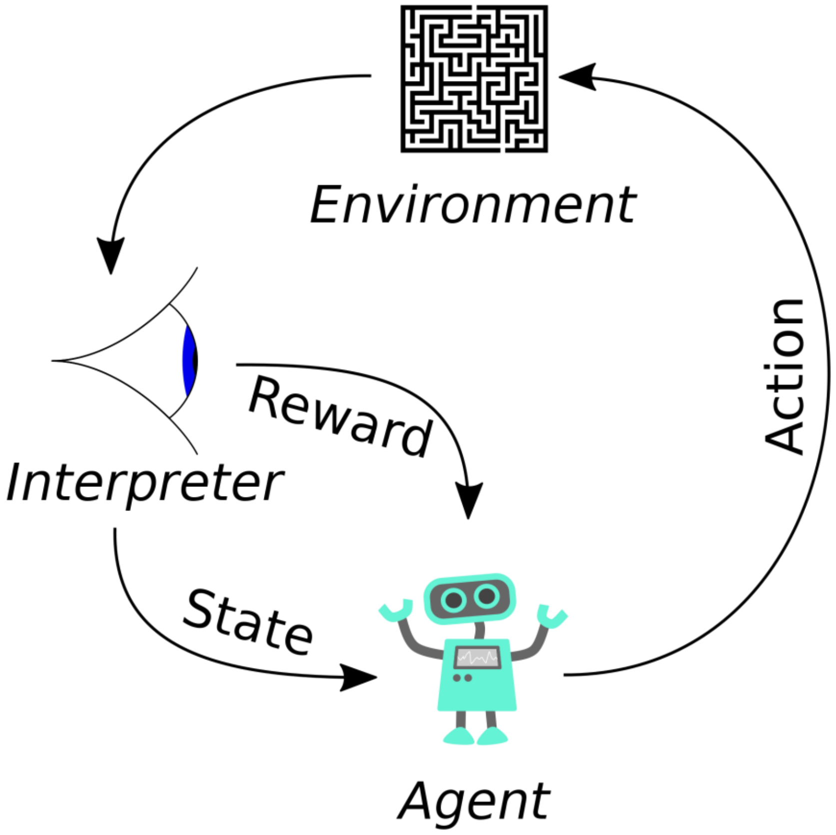
Cutting Planes

Reinforcement Learning

- **Reinforcement learning (RL)**: how intelligent agents ought to take actions in an environment to maximize the notion of cumulative **reward**.

- State space $x_t \in S$
- Action space $a_t \in A$
- Reward $r_t = r(x_t, a_t)$
- Transition $x_{t+1} \sim p(\cdot \mid x_{1:\mathrm{t}}, a_t)$
- Policy $\pi: X \rightarrow P(A)$
- A stream of experience
  $x_0 \rightarrow a_0 \rightarrow r_0 \rightarrow x_1 \rightarrow a_1 \rightarrow r_2 \rightarrow \cdots$
- Objective with $\gamma \in (0,1]$

$$J(\pi) = \max_{\pi} E_\pi[\Sigma_{t=0}^{\infty} \gamma^t r_t]$$



Markov Decision Process (MDP)

- A **Markov Decision Process (MDP)** has the form $M = (S, A, p_{init}, p_{trans}, r)$, where $S, A, p_{init}, p_{trans}, r$ are the collection of states, actions, distribution at the beginning, state transition distribution and reward function.
- Given a decision plan π, the probability of episode $\tau = (s_0, a_0, s_1, \cdots, s_T)$ is

$$p_\pi(\tau) = p_{init}(s_0) \prod_{t=0}^{|\tau|-1} \pi(a_t|s_t)\ p(s_{t+1}|s_t, a_t)$$

The MDP control problem is to find a policy that maximize the following

$$v^\pi := \mathbb{E}_{\tau\sim p_\pi}[\sum_{t=0}^{|\tau|} \gamma^t r_t]$$

By policy gradient theorem [Sutton et al. (1999)],

$$\nabla_\theta\, v^{\pi_\theta} = \mathbb{E}_{\tau\sim p_{\pi_\theta}}[\sum_{t=0}^{|\tau|-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t+1}^{|\tau|} r_{t'}]$$

Evolution Strategies for Reinforcement Learning

- Evolution Strategies (ES) is a class of black box optimization algorithms that are heuristic search procedures inspired by natural evolution
  - At every iteration (“generation”), a population of parameter vectors (“genotypes”) is perturbed (“mutated”) and their objective function value (“fitness”) is evaluated.
  - The highest-scoring parameter vectors are then recombined to form the population for the next generation, and this procedure is iterated until the objective is fully optimized [Salimans et al. (2017)].

- For example, $\mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma\epsilon)$ is the “fitness” of this generation, then $\nabla_\theta \mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma\epsilon)$ gives the direction of evolution.

Evolution Strategies for Reinforcement Learning

$$\nabla_\theta F(\theta) \approx \mathbb{E}_{\epsilon \sim N(0,I)} \left[\frac{F(\theta+\sigma\epsilon) - F(\theta)}{\sigma} \epsilon \right]$$

$$= \mathbb{E}_{\epsilon \sim N(0,I)} \left[\frac{F(\theta+\sigma\epsilon)}{\sigma} \epsilon \right]$$

$$\approx \frac{1}{N\sigma} \sum_{i=1}^{N} F(\theta + \sigma\epsilon_i)\epsilon_i \text{ (zero order optimization)}$$

Algorithm 1 Evolution Strategies

```
Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for t = 0, 1, 2, ... do
    Sample ε_1, ... ε_n ~ N(0, I)
    Compute returns F_i = F(θ_t + σε_i) for i = 1, ..., n
    Set θ_{t+1} ← θ_t + α (1/(nσ)) Σ_{i=1}^{n} F_i ε_i
end for
```

- This optimization strategy doesn't need gradient and transition information

Contents

- Mixed-Integer Linear Programs (MIP): Brief Introduction
  - Branch & bound
  - Primal heuristics
  - Cutting plane
- Reinforcement Learning: Brief Introduction
  - Markov decision process (MDP)
  - An evolution strategy for reinforcement learning
- AI4MIP
  - Learning primal heuristics
  - Learning branching policies
  - Learning to cut
  - Data augmentation

A Sketch for AI4MIP

- State-of-the-art MIP solvers get reliable outcomes but adopt some heuristic and time-consuming strategies for branching and cut.
- One gets benefits from good primal solutions when using B&B.

- People attempted to use neural networks to accelerate these expensive strategies or find a better heuristic strategy.
  - Learning primal heuristics [Nair et al. (2020)]
  - Learning branching policies [Gasse et al. (2019); Gupta et al. (2020); Ding et al. (2020); Scavuzzo et al. (2022)]
  - Learning to cut [Huang et al. (2022); Paulus et al. (2022)]
  - Data augmentation [Duan et al. (2022)]

Learning Branching Policies [Gasse et al. (2019)]

- The state-of-art branching policy (greedy) is expensive to get a good one.
- Note that one has a lot of examples of good branching given by traditional criteria, we can use neural networks to learn from those strategies.
- If NN succeeds to learn what is behind these data, we can get good branching by NN, which is much faster than using greedy criteria.
- The key is to do supervised learning from past state-of-art branching strategies with GNN.

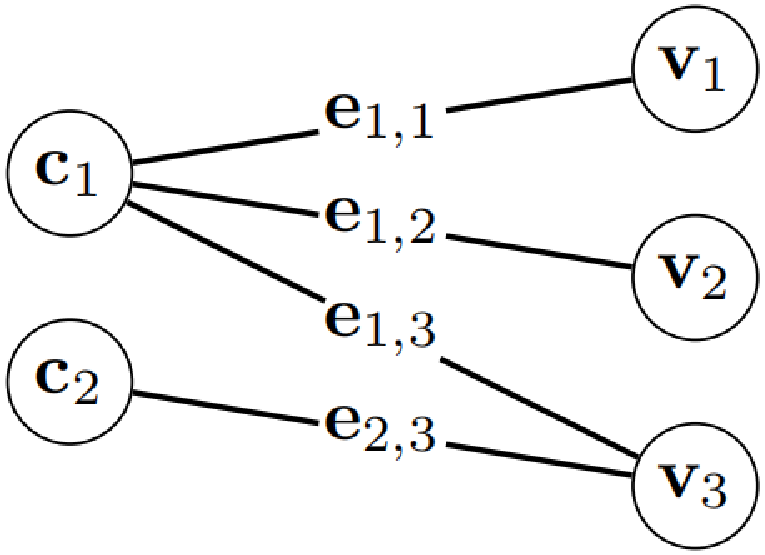
Learning Branching Policies [Gasse et al. (2019)]

- Construct a bipartite graph **G** from a MIP:
  - One side: are nodes corresponding to the constraints in the MIP
    Other side: are nodes corresponding to the variables in the MIP
  - An edge (i, j) connects a constraint node i and a variable node j if the latter is involved in the former, that is if $a_{ij} \neq 0$, where a_{ij} is the coefficient of x_j in constraint i.

- For example:
  - Constraints

    $3x + 4y + z \leq 1$

    $0x + 0y + z \leq 2$



Learning Branching Policies [Gasse et al. (2019)]

- We encode the state s_t of the B&B process at time *t* as a bipartite graph with node and edge features (G, C, E, V):
  - G is a bipartite graph and C, E, V are the features designed to describe constraints, egdes in the bipartite graph and variables in a certain B&B solver state.

Tensor	Feature	Description
C	obj_cos_sim	Cosine similarity with objective.
	bias	Bias value, normalized with constraint coefficients.
	is_tight	Tightness indicator in LP solution.
	dualsol_val	Dual solution value, normalized.
	age	LP age, normalized with total number of LPs.
E	coef	Constraint coefficient, normalized per constraint.
V	type	Type (binary, integer, impl. integer, continuous) as a one-hot encoding.
	coef	Objective coefficient, normalized.
	has_lb	Lower bound indicator.
	has_ub	Upper bound indicator.
	sol_is_at_lb	Solution value equals lower bound.
	sol_is_at_ub	Solution value equals upper bound.
	sol_frac	Solution value fractionality.
	basis_status	Simplex basis status (lower, basic, upper, zero) as a one-hot encoding.
	reduced_cost	Reduced cost, normalized.
	age	LP age, normalized.
	sol_val	Solution value.
	inc_val	Value in incumbent.
	avg_inc_val	Average value in incumbents.

- Denote training dataset by $\mathcal{D}_{train} = \{(s_i, a_i)\}$, where $\{s_i\}$ are MIP problems and s_i is a set of state and a_i is the index of the variable to branch at state s_i.
- This is obtained by running a greedy strategy on s_i.

Learning Branching Policies [Gasse et al. (2019)]

- Use 2-layer perceptrons (f_C, f_V, g_C, g_V) with ReLU activation function and graph structure to aggregate information.

$$c_i \leftarrow f_C\left(c_i, \sum_{j}^{(i,j)\in\mathcal{E}} g_C\left(c_i, v_j, e_{i,j}\right)\right), \; v_i \leftarrow f_v\left(v_j, \sum_{i}^{(i,j)\in\mathcal{E}} g_v\left(c_i, v_j, e_{i,j}\right)\right)$$

- The outcome of the GCNN+softmax gives the probability of branch on each node.

- Then train these networks by supervised learning.

$$L(\theta) = \frac{-1}{N} \sum_{(s,a^*)\in\mathcal{D}} \log \pi_\theta(a^*|s)$$

Learning Branching Policies [Gasse et al. (2019)]

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	17.30 ± 6.1%	0 / 100	17 ±13.7%	411.34 ± 4.3%	0 / 90	171 ± 6.4%	3600.00 ± 0.0%	0 / 0	n/a ± n/a %
RPB	8.98 ± 4.8%	0 / 100	**54** ±20.8%	60.07 ± 3.7%	0 / 100	1741 ± 7.9%	1677.02 ± 3.0%	4 / 65	47 299 ± 4.9%
TREES	9.28 ± 4.9%	0 / 100	187 ± 9.4%	92.47 ± 5.9%	0 / 100	2187 ± 7.9%	2869.21 ± 3.2%	0 / 35	59 013 ± 9.3%
SVMRANK	8.10 ± 3.8%	1 / 100	165 ± 8.2%	73.58 ± 3.1%	0 / 100	1915 ± 3.8%	2389.92 ± 2.3%	0 / 47	42 120 ± 5.4%
LMART	7.19 ± 4.2%	14 / 100	167 ± 9.0%	59.98 ± 3.9%	0 / 100	1925 ± 4.9%	2165.96 ± 2.0%	0 / 54	45 319 ± 3.4%
GCNN	**6.59** ± 3.1%	**85** / 100	134 ± 7.6%	**42.48** ± 2.7%	**100** / 100	**1450** ± 3.3%	**1489.91** ± 3.3%	**66** / 70	**29 981** ± 4.9%

Set Covering

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	4.11 ±12.1%	0 / 100	6 ±30.3%	86.90 ± 12.9%	0 / 100	72 ±19.4%	1813.33 ± 5.1%	0 / 68	400 ± 7.5%
RPB	2.74 ± 7.8%	0 / 100	**10** ±32.1%	17.41 ± 6.6%	0 / 100	689 ±21.2%	136.17 ± 7.9%	13 / 100	5511 ±11.7%
TREES	2.47 ± 7.3%	0 / 100	86 ±15.9%	23.70 ± 11.2%	0 / 100	976 ±14.4%	451.39 ±14.6%	0 / 95	10 290 ±16.2%
SVMRANK	2.31 ± 6.8%	0 / 100	77 ±15.0%	23.10 ± 9.8%	0 / 100	867 ±13.4%	364.48 ± 7.7%	0 / 98	6329 ± 7.7%
LMART	**1.79** ± 6.0%	**75** / 100	77 ±14.9%	14.42 ± 9.5%	1 / 100	873 ±14.3%	222.54 ± 8.6%	0 / 100	7006 ± 6.9%
GCNN	1.85 ± 5.0%	25 / 100	70 ±12.0%	**10.29** ± 7.1%	**99** / 100	**657** ±12.2%	**114.16** ±10.3%	**87** / 100	**5169** ±14.9%

Combinatorial Auction

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	30.36 ±19.6%	4 / 100	14 ±34.5%	214.25 ± 15.2%	1 / 100	76 ±15.8%	742.91 ± 9.1%	15 / 90	55 ± 7.2%
RPB	26.55 ±16.2%	9 / 100	**22** ±31.9%	156.12 ± 11.5%	8 / 100	**142** ±20.6%	631.50 ± 8.1%	14 / 96	**110** ±15.5%
TREES	28.96 ±14.7%	3 / 100	135 ±20.0%	159.86 ± 15.3%	3 / 100	401 ±11.6%	671.01 ±11.1%	1 / 95	381 ±11.1%
SVMRANK	23.58 ±14.1%	11 / 100	117 ±20.5%	130.86 ± 13.6%	13 / 100	348 ±11.4%	586.13 ±10.0%	21 / 95	321 ± 8.8%
LMART	23.34 ±13.6%	16 / 100	117 ±20.7%	128.48 ± 15.4%	23 / 100	349 ±12.9%	582.38 ±10.5%	15 / 95	314 ± 7.0%
GCNN	**22.10** ±15.8%	**57** / 100	107 ±21.4%	**120.94** ± 14.2%	**52** / 100	339 ±11.8%	**563.36** ±10.7%	**30** / 95	338 ±10.9%

Capacitated Facility Location

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard Nodes
FSB	23.58 ±29.9%	9 / 100	7 ±35.9%	1503.55 ± 20.9%	0 / 74	38 ±28.2%	3600.00 ± 0.0%	0 / 0	n/a ± n/a %
RPB	8.77 ±11.8%	7 / 100	**20** ±36.1%	**110.99** ± 24.4%	41 / 100	**729** ±37.3%	2045.61 ±18.3%	22 / 42	**2675** ±24.0%
TREES	10.75 ±22.1%	1 / 100	76 ±44.2%	1183.37 ± 34.2%	1 / 47	4664 ±45.8%	3565.12 ± 1.2%	0 / 3	38 296 ± 4.1%
SVMRANK	8.83 ±14.9%	2 / 100	46 ±32.2%	242.91 ± 29.3%	1 / 96	546 ±26.0%	2902.94 ± 9.6%	1 / 18	6256 ±15.1%
LMART	7.31 ±12.7%	30 / 100	52 ±38.1%	219.22 ± 36.0%	15 / 91	747 ±35.1%	3044.94 ± 7.0%	0 / 12	8893 ± 3.5%
GCNN	**6.43** ±11.6%	**51** / 100	43 ±40.2%	192.91 ±110.2%	**42** / 82	1841 ±88.0%	**2024.37** ±30.6%	**25** / 29	2997 ±26.3%

Maximum Independent Set

Learning to Cut [Paulus et al. (2022)]

- Although there are several ways to construct cuts, only “few” cuts are effective for solving MIP.

- However, it is time-consuming to figure out which cut is a good one.

- We have a lot of examples of good cuts given by traditional criteria. This makes it possible to learn the cutting strategy with NN.

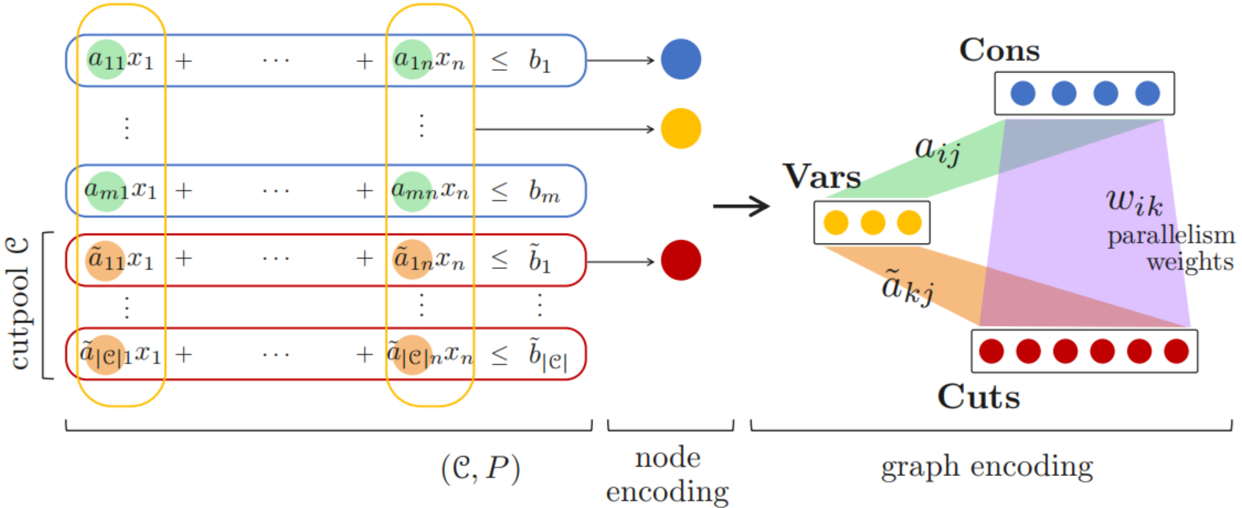
- The key is to do supervised learning from past state-of-art cut strategies with GNN.

Learning to Cut [Paulus et al. (2022)]

- Construct a tripartite graph G from a MIP and some available cuts:
  - Nodes in the graph correspond to constraints, variables and available cuts.

  - An edge (i, j) connects a constraint (cut) node i and a variable node j if the latter is involved in the former, that is if $a_{ij}(\widetilde{a_{ij}}) \neq 0$, where $a_{ij}(\widetilde{a_{ij}})$ is the coefficient of x_j in constraint (cut) i.

  - Link Cons and Cuts with a complete set of weighted edges. If (a_i, b_i) is a Cons and $(\widetilde{a_k}, \widetilde{b_k})$ is a Cut, then the weight $w_{ik} = \frac{\widetilde{a_k} \cdot a_k}{\|\widetilde{a_k}\| \|a_k\|}$.

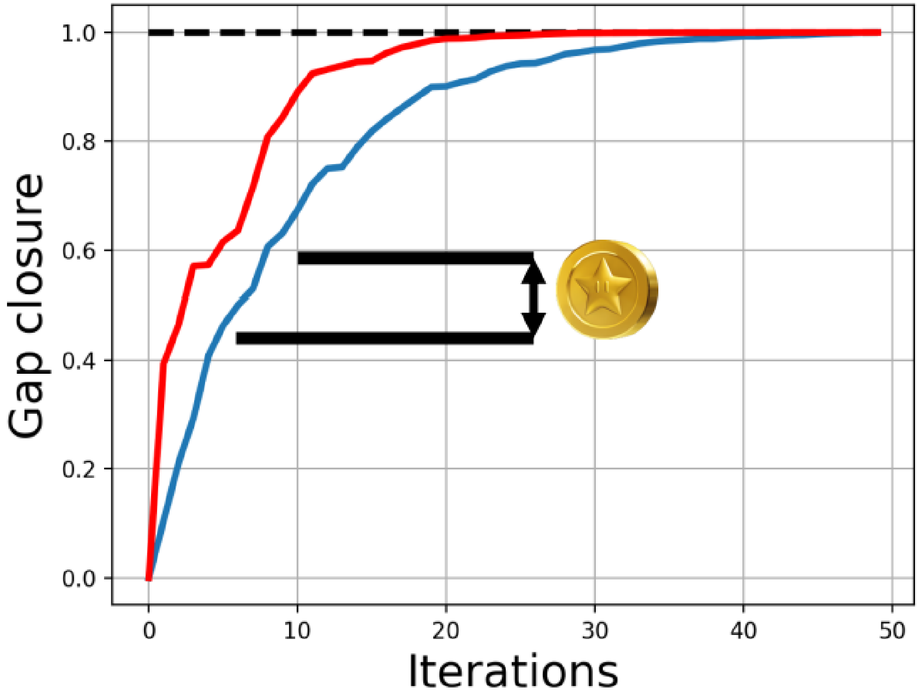


Learning to Cut [Paulus et al. (2022)]

- Denote the minimum point of linear programming P by x^*, $c^T x^*$ by z^*, the minimum point of LP after adding the cut C_j by x^j, $c^T x^j$ by z^j.
- The lookahead score (LA) of cut C_j for LP P is defined as:

$$s_{LA}(C_j, P) := z^j - z^* \geq 0$$

which reflects the "benefit" of cut C_j .



Learning to Cut [Paulus et al. (2022)]

- The training dataset is $\mathcal{D}_{train} = \left\{ (\mathcal{C}, P, \{s_{LA}(C_j, P)\}_{C_j \in \mathcal{C}}) \right\}$, where $\mathcal{C}$ is the collection of available cuts, P is a linear programming problem and $\{s_{LA}(C_j, P)\}_{C_j \in \mathcal{C}}$ is the lookahead score of each cut for problem P.

- Given a set of candidates of cuts (denoted by $\mathcal{C}$), we can learn a cut selection policy $\tilde{s}$ that imitates the lookahead expert by

$$L(\tilde{s}) := \frac{-1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} q_C \log \tilde{s}(C) + (1 - q_C) \log(1 - \tilde{s}(C))$$

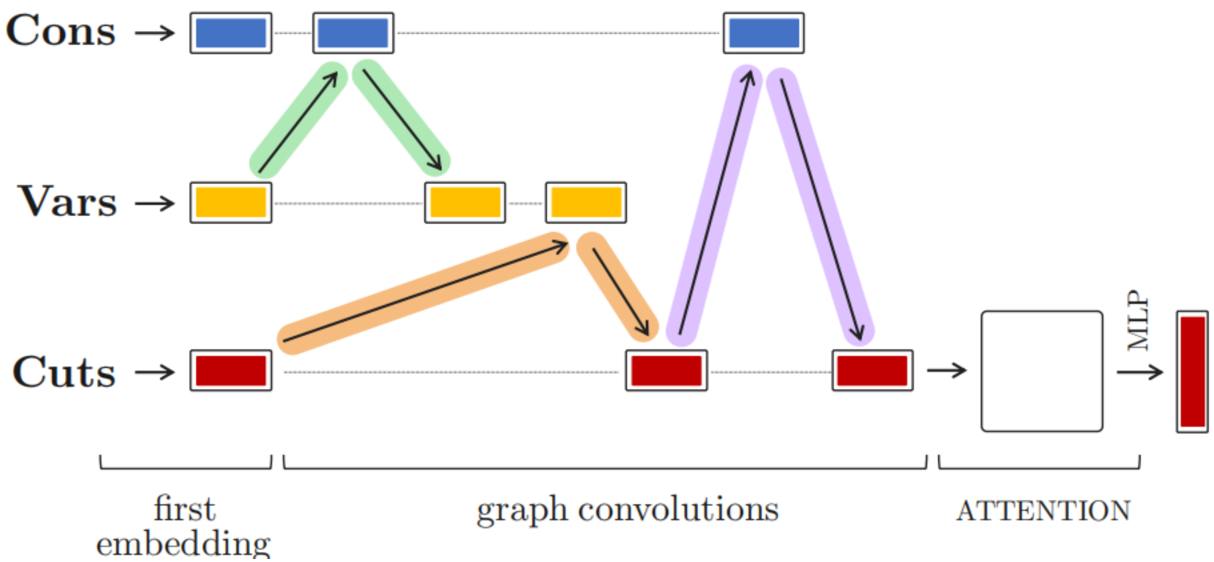
$q_C = \frac{s_{LA}(C)}{s_{LA}(C^*_{LA})}$, $C^*_{LA} = \arg\max_{C \in \mathcal{C}} s_{LA}(C)$, $\tilde{s}(\mathrm{C})$ is the predicted lookahead score.

Learning to Cut [Paulus et al. (2022)]

- The policy given by Neural network is as follows: After message passing by and extracting features by attention network and MLP layer with sigmoid activation, the predicted s_{LA} score for each cut is obtained.
- Then by optimizing

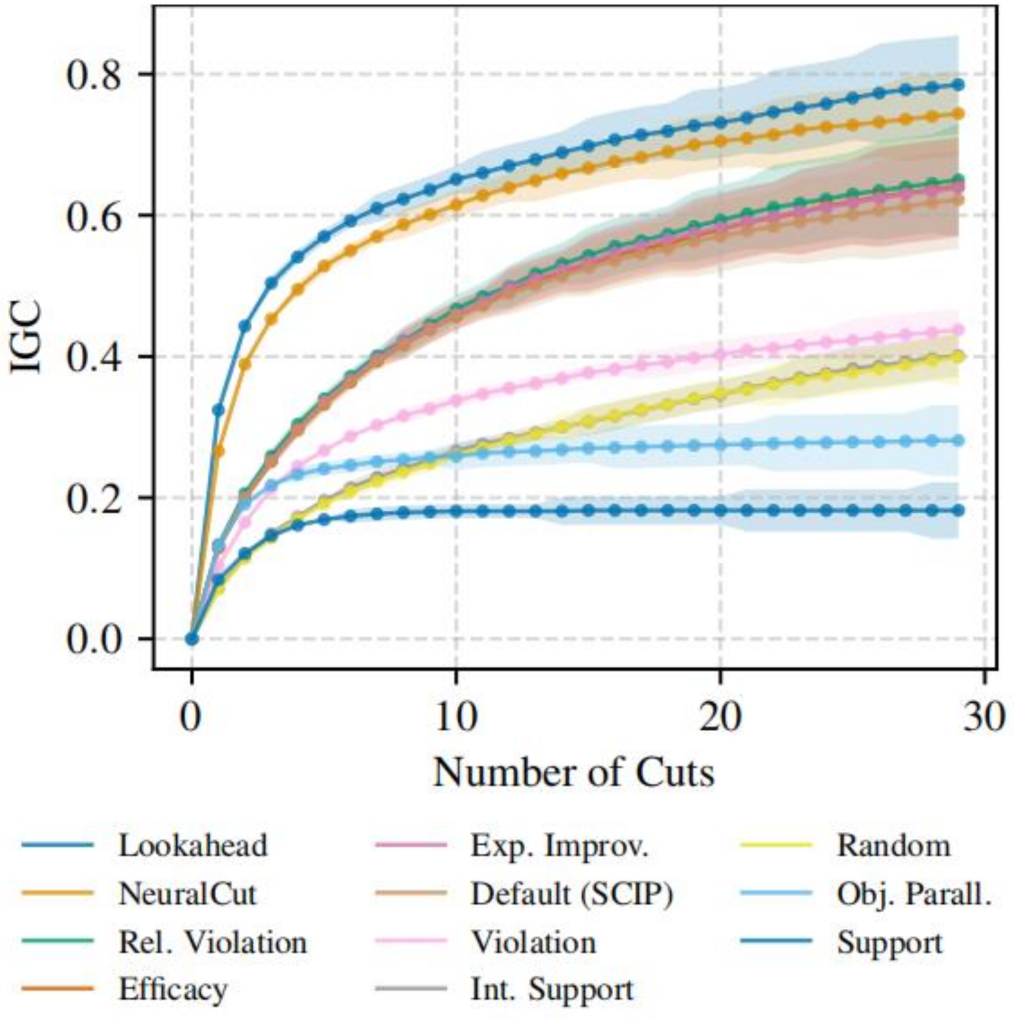
$$L(\tilde{s}_\theta) := \frac{-1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} q_C \log \tilde{s}_\theta(C) + (1 - q_C) \log(1 - \tilde{s}_\theta(C))$$

by gradient-based method, the network gets trained.



Learning to Cut [Paulus et al. (2022)]

	Bound fulfillment (↑) on test samples, mean			
	MAX. CUT	PACKING	BIN. PACKING	PLANNING
Lookahead	*1.0*	*1.0*	*1.0*	*1.0*
NeuralCut	**0.96**	**0.61**	**0.78**	***1.0***
Tang et al. (2020)	0.58	0.27	0.22	0.49
Default (SCIP)	0.71	0.60	0.33	0.64
Exp. Improv.	0.69	0.60	0.32	0.85
Efficacy	0.65	0.60	0.32	0.46
Obj. Parall.	0.47	0.34	0.27	0.44
Rel. Violation	0.50	0.60	0.33	0.48
Violation	0.64	0.35	0.21	0.26
Support	0.57	0.18	0.13	0.29
Int. Support	0.62	0.18	0.21	0.34
Random	0.41	0.15	0.16	0.25



Learning to Cut (RL) [Huang et al. (2022)]

- At each iteration, the LP relaxation gets different results since the new added cuts. Denote the LP relaxation has optimal x_t^* at iteration t.

$$x_0^* \to x_1^* \to x_2^* \to \cdots \to x_T^* \approx x_{MIP}^*$$

- The integrality gap closure: $c^T(x_t^* - x_0^*)$ is monotonic in *t*, upper bounded by $c^T(x_{MIP}^* - x_0^*)$.

- Define the reward of adding the cut by $r_t = C^T|x_t^* - x_{t+1}^*|$, which is the same as lookahead score in [Paulus et al. (2022)].

- Maximize $E_{\pi_\theta}[\Sigma_{t=0}^{\infty}\gamma^t r_t]$

Learning to Cut (RL) [Huang et al. (2022)]

- At iteration t, the constraint set of LP is

$$A_t = \{a_i^T x \le b_i\}_{i=1}^{N_t}$$

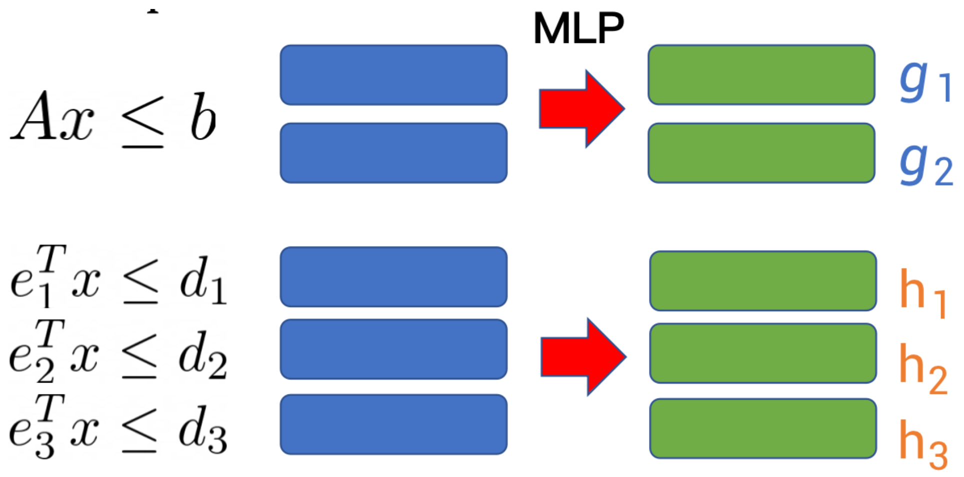
- Solving this LP produces, we get an optimal solution x_t^* along with the set of candidate cuts $\mathcal{C}_\mathrm{t}$. We set the state to be $s_t = \{A_t, c, x_t^*, \mathcal{C}_t\}$.
- Note that $\mathcal{C}_\mathrm{t}$ is also the action set of this iteration.

- When taking an action $\mathcal{C}$ in $\mathcal{C}_\mathrm{t}$, the state transit to $s_t = \{A_{t+1}, c, x_{t+1}^*, \mathcal{C}_{t+1}\}$,

- Optimization method——Evolutionary Strategies (ES)
  - Easy for practice
  - Easy to parallel

Learning to Cut (RL) [Huang et al. (2022)]

- The policy to choose a cut is by learning the embedding of linear constraints and cuts.
- A simple example is as follows:



$$
\begin{aligned}
s_1 &= g_1^{\mathrm{T}} h_1 + g_2^{\mathrm{T}} h_1 \\
s_2 &= g_1^{\mathrm{T}} h_2 + g_2^{\mathrm{T}} h_2 \\
s_3 &= g_1^{\mathrm{T}} h_3 + g_2^{\mathrm{T}} h_3 \\
\pi &= \mathrm{softmax}(s)
\end{aligned}
$$

- Huang et al. (2022) used an attention network or LSTM to learn the embedding of constraints or cuts.

Learning to Cut (RL) [Huang et al. (2022)]

Table 1: Number of cuts it takes to reach optimality. We show mean $\pm$ std across all test instances.

Tasks	**Packing**	**Planning**	**Binary**	**Max Cut**
Size	10×5	13×20	10×20	10×22
RANDOM	48 ± 36	44 ± 37	81 ± 32	69 ± 34
MV	62 ± 40	48 ± 29	87 ± 27	64 ± 36
MNV	53 ± 39	60 ± 34	85 ± 29	47 ± 34
LE	34 ± 17	310 ± 60	89 ± 26	59 ± 35
RL	$\mathbf{14 \pm 11}$	$\mathbf{10 \pm 12}$	$\mathbf{22 \pm 27}$	$\mathbf{13 \pm 4}$

Table 2: IGC for test instances of size roughly 1000. We show mean $\pm$ std of IGC achieved on adding $T = 50$ cuts.

Tasks	**Packing**	**Planning**	**Binary**	**Max Cut**
Size	30×30	61×84	33×66	27×67
RAND	0.18 ± 0.17	0.56 ± 0.16	0.39 ± 0.21	0.56 ± 0.09
MV	0.14 ± 0.08	0.18 ± 0.08	0.32 ± 0.18	0.55 ± 0.10
MNV	0.19 ± 0.23	0.31 ± 0.09	0.32 ± 0.24	0.62 ± 0.12
LE	0.20 ± 0.22	0.01 ± 0.01	0.41 ± 0.27	0.54 ± 0.15
RL	$\mathbf{0.55 \pm 0.32}$	$\mathbf{0.88 \pm 0.12}$	$\mathbf{0.95 \pm 0.14}$	$\mathbf{0.86 \pm 0.14}$

Table 3: IGC for test instances of size roughly 5000. We show mean $\pm$ std of IGC achieved on adding $T = 250$ cuts.

Tasks	**Packing**	**Planning**	**Binary**	**Max Cut**
Size	60×60	121×168	66×132	54×134
RANDOM	0.05 ± 0.03	0.38 ± 0.08	0.17 ± 0.12	0.50 ± 0.10
MV	0.04 ± 0.02	0.07 ± 0.03	0.19 ± 0.18	0.50 ± 0.06
MNV	0.05 ± 0.03	0.17 ± 0.10	0.19 ± 0.18	0.56 ± 0.11
LE	0.04 ± 0.02	0.01 ± 0.01	0.23 ± 0.20	0.45 ± 0.08
RL	$\mathbf{0.11 \pm 0.05}$	$\mathbf{0.68 \pm 0.10}$	$\mathbf{0.61 \pm 0.35}$	$\mathbf{0.57 \pm 0.10}$

Learning Primal Heuristics [Nair et al. (2020)]

- NN can be used to find good feasible points.
- Define an energy function by

$$E(x;M) = \begin{cases} c^T x, & x \in R \\ \infty \;, & x \notin R \end{cases}$$

where R is the feasible region. By energy-based generative model, we have

$$p(x|M) = \frac{\exp(-E(x;M))}{Z(M)}$$

where $Z(M)$ is to normalize the distribution to sum to 1.

- By sampling from this distribution, it is likely to get good feasible points.
- **Key:** learn this distribution with GNN by optimizing the cross-entropy.

Learning Primal Heuristics [Nair et al. (2020)]

- Denote training dataset by $\mathcal{D}_{train} = \{(X_i, M_i)\}$, $\{M_i\}$ are MIP problems and $X_i = \{x^{i,j}\}$ is a set of assignments for the instance M_i.

- X_i is obtained by running SCIP (a traditional MIP solver) on M_i and collecting feasible assignments it finds during the solve.
- Then, the loss function (cross-entropy) can be written as:

$$L(\theta) = -\sum_{i=1}^{N} \sum_{j=1}^{N_i} w_{ij} \log p_\theta(x^{i,j} | M_j), \quad \text{where } w_{ij} = \frac{\exp(-c_i^T x^{i,j})}{\sum_{k=1}^{N_i} \exp(-c_i^T x^{i,k})}$$

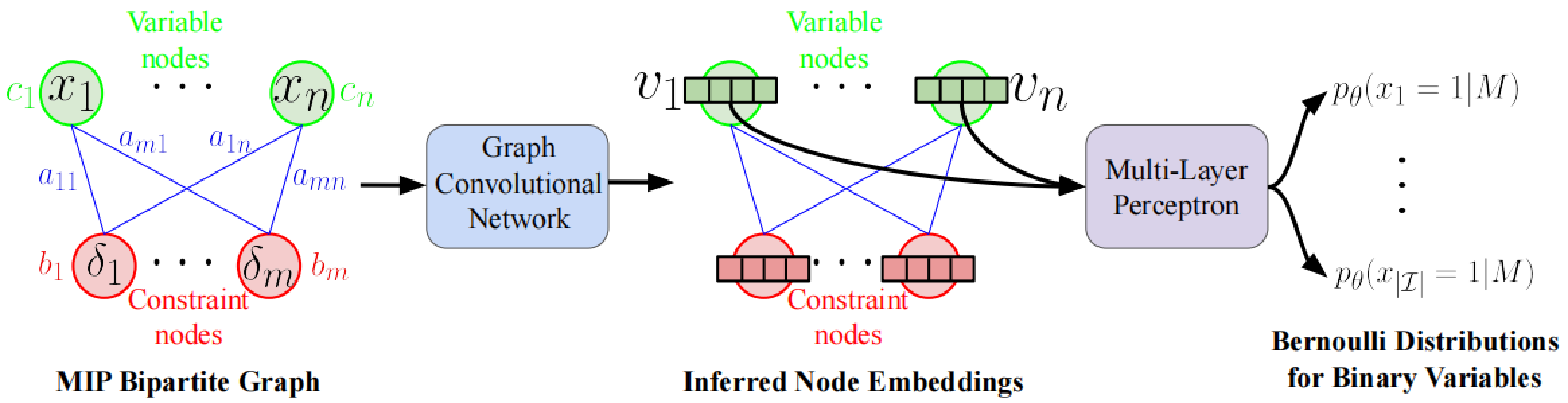
- Let I be the set of dimensions of *x* corresponding to the integer variables. Let x_d denote the *d*th dimension of *x*. We use a conditionally-independent model:

$$p_\theta(x|M) = \prod_{d \in I} p_\theta(x_d|M)$$

- For simplicity, we first assume that each x_d is binary with possible values $\{0, 1\}$ and use the Bernoulli distribution for each such variable.

Learning Primal Heuristics [Nair et al. (2020)]

- To learn Bernoulli distribution, the feature of a MIP problem first goes through a GCN module to get inferred embeddings.
- Then use an MLP layer to predict the parameters of those Bernoulli distributions.



To be specific, we get embeddings of nodes v_d after GCN, then

$$t_d = MLP(v_d; \theta)$$

and

$$\mu_d = \mathrm{p}_\theta\left(x_d = 1|M\right) = \frac{1}{1+\exp(-t_d)}$$

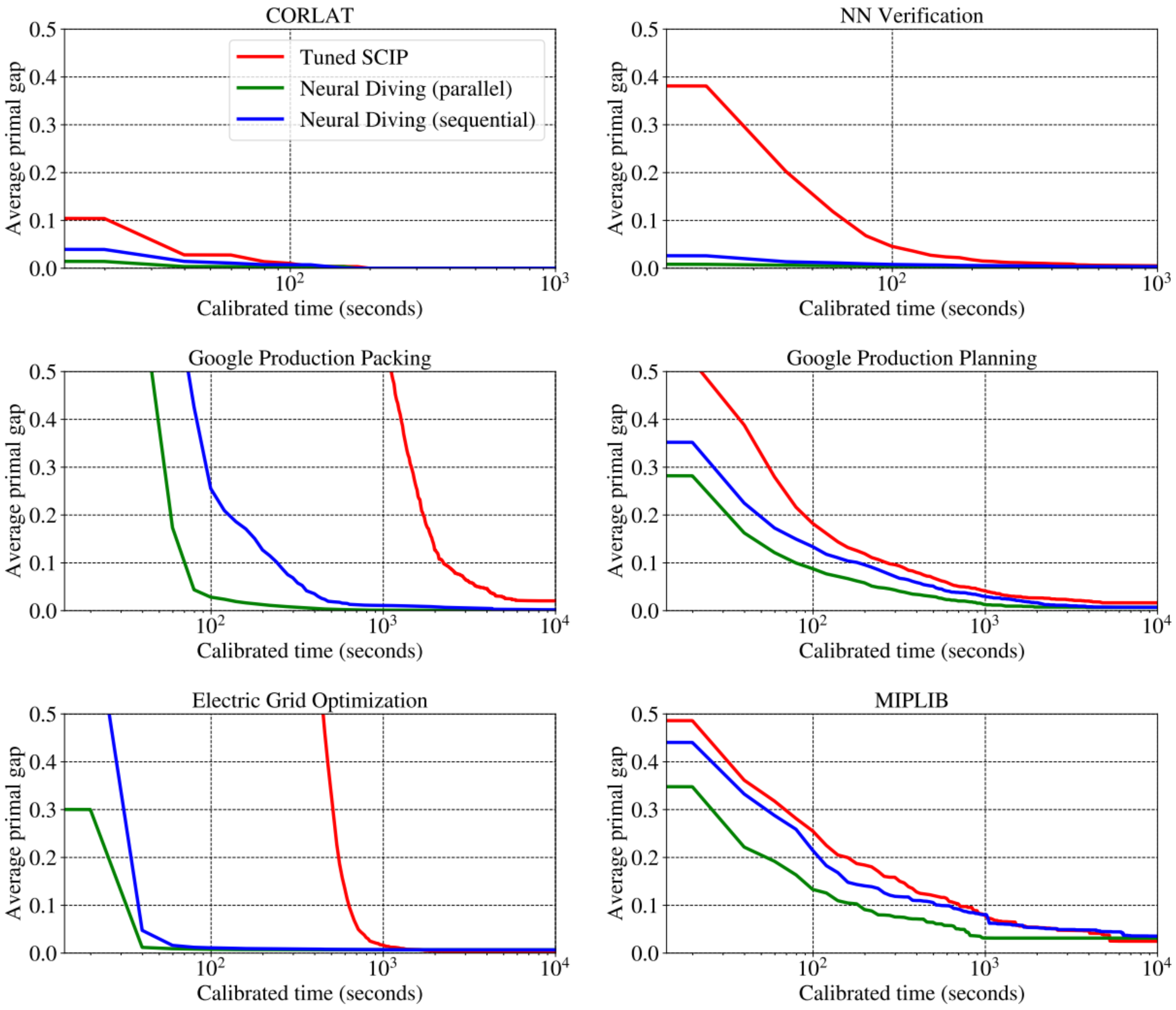
Learning Primal Heuristics [Nair et al. (2020)]

- For general integers, one reframs the prediction task for general integer variables as a sequence of binary prediction tasks, based on the binary representation of the target integer value.

- For an integer variable *z* that can be assigned values from a finite set with cardinality $card(z)$, any target value can be represented as a sequence of $\lceil \log_2(card(z)) \rceil$ bits.

- We train our model to predict these bits in sequence, from most significant to least significant bit.

Learning Primal Heuristics [Nair et al. (2020)]



Reference

- Berthold, T. (2006). Primal heuristics for mixed integer programs (Doctoral dissertation, Zuse Institute Berlin (ZIB)).
- Wikipedia contributors. (2023, April 21). Reinforcement learning. In *Wikipedia, The Free Encyclopedia*. Retrieved 13:58, April 25, 2023, from https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1151000318
- Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.
- Huang, Z., Wang, K., Liu, F., Zhen, H. L., Zhang, W., Yuan, M., ... & Wang, J. (2022). Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, *123*, 108353.
- Nair, V., Bartunov, S., Gimeno, F., Von Glehn, I., Lichocki, P., Lobov, I., ... & Zwols, Y. (2020). Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*.
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L., & Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, *32*.
- Paulus, M. B., Zarpellon, G., Krause, A., Charlin, L., & Maddison, C. (2022, June). Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning* (pp. 17584-17600). PMLR.
- Gupta, P., Gasse, M., Khalil, E., Mudigonda, P., Lodi, A., & Bengio, Y. (2020). Hybrid models for learning to branch. *Advances in neural information processing systems*, *33*, 18087-18097.

Reference

- Ding, J. Y., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., & Song, L. (2020, April). Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the aaai conference on artificial intelligence*(Vol. 34, No. 02, pp. 1452-1459).
- Scavuzzo, L., Chen, F., Chételat, D., Gasse, M., Lodi, A., Yorke-Smith, N., & Aardal, K. (2022). Learning to branch with tree mdps. *Advances in Neural Information Processing Systems*, *35*, 18514-18526.
- Duan, H., Vaezipoor, P., Paulus, M. B., Ruan, Y., & Maddison, C. (2022, June). Augment with care: Contrastive learning for combinatorial problems. In *International Conference on Machine Learning* (pp. 5627-5642). PMLR.
- Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, *12*.