

Order matters: Boosting Mathematical Solver via Machine Learning Techniques

Contact author: Xijun Li / xijun.li@huawei.com

Noah's Ark Lab | Decision Making and Reasoning Lab



*Thanks Prof. Xiaodong Luo and Prof. Akang Wang for inviting me to give this talk at CUHKSZ and SRIBD.



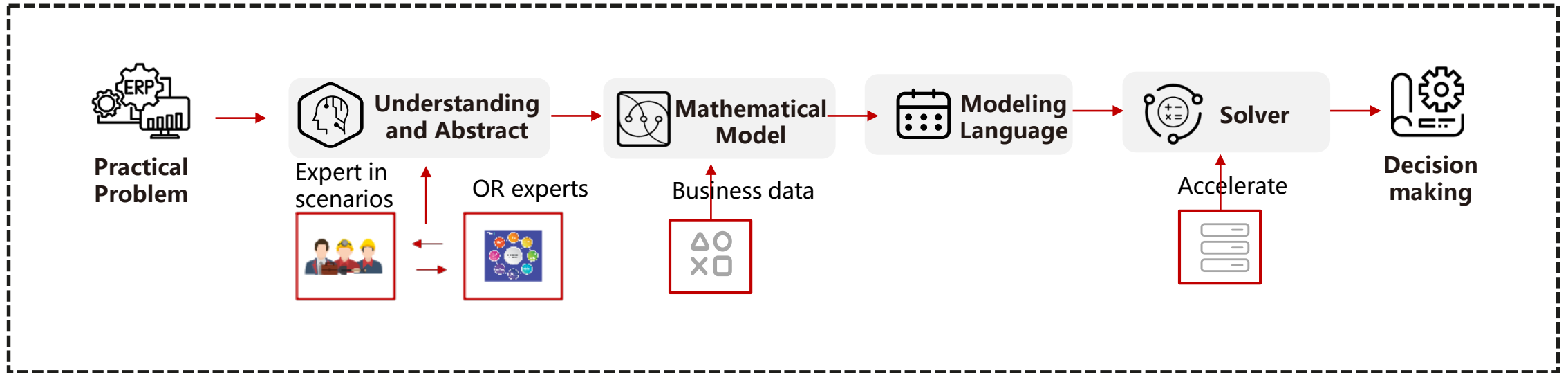
NOAH'S ARK LAB



Outline

- **Preliminary**
- **Order Matters: Why We Care**
- **Two of Our Group' s Research Work in Year 2022**
 - > **Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning**
 - > **Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model**
- **Challenges and Outlook**

Preliminary



The procedure of calling solver to solve mathematical optimization problem

Preliminary

Mixed-Integer Linear Program

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

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

Linear Program (LP) relaxation

$$\begin{aligned} & \arg \min_x \quad c^T x \\ & \text{subject to} \quad Ax \leq b, \\ & \quad \quad \quad l \leq x \leq u, \\ & \quad \quad \quad x \in \mathbb{R}^n. \end{aligned}$$

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

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

Branch and Bound

Stopping criterion:

- ▶ $L = U$ (optimality certificate)
- ▶ $L = \infty$ (infeasibility certificate)
- ▶ $L - U < \text{threshold}$ (early stopping)

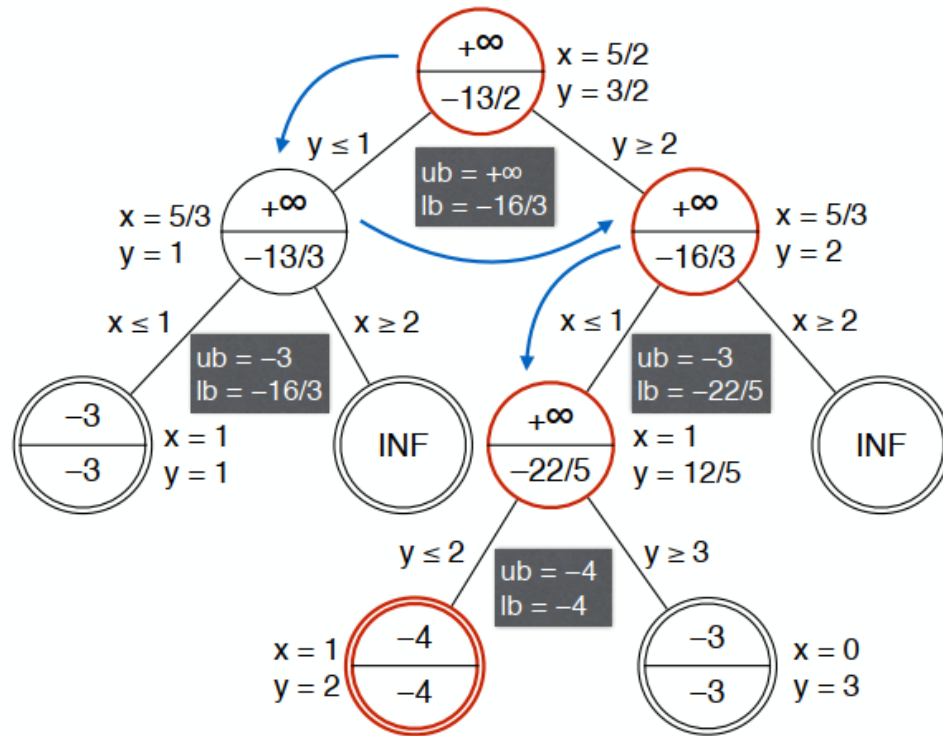
Split the LP recursively over a non-integral variable, i.e. $\exists i \leq p \mid x_i^* \notin \mathbb{Z}$

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

Lower bound (L): minimal among leaf nodes.

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

Preliminary



node expansion order

global lower and upper bound

optimal node

fathomed node

• LP relaxations

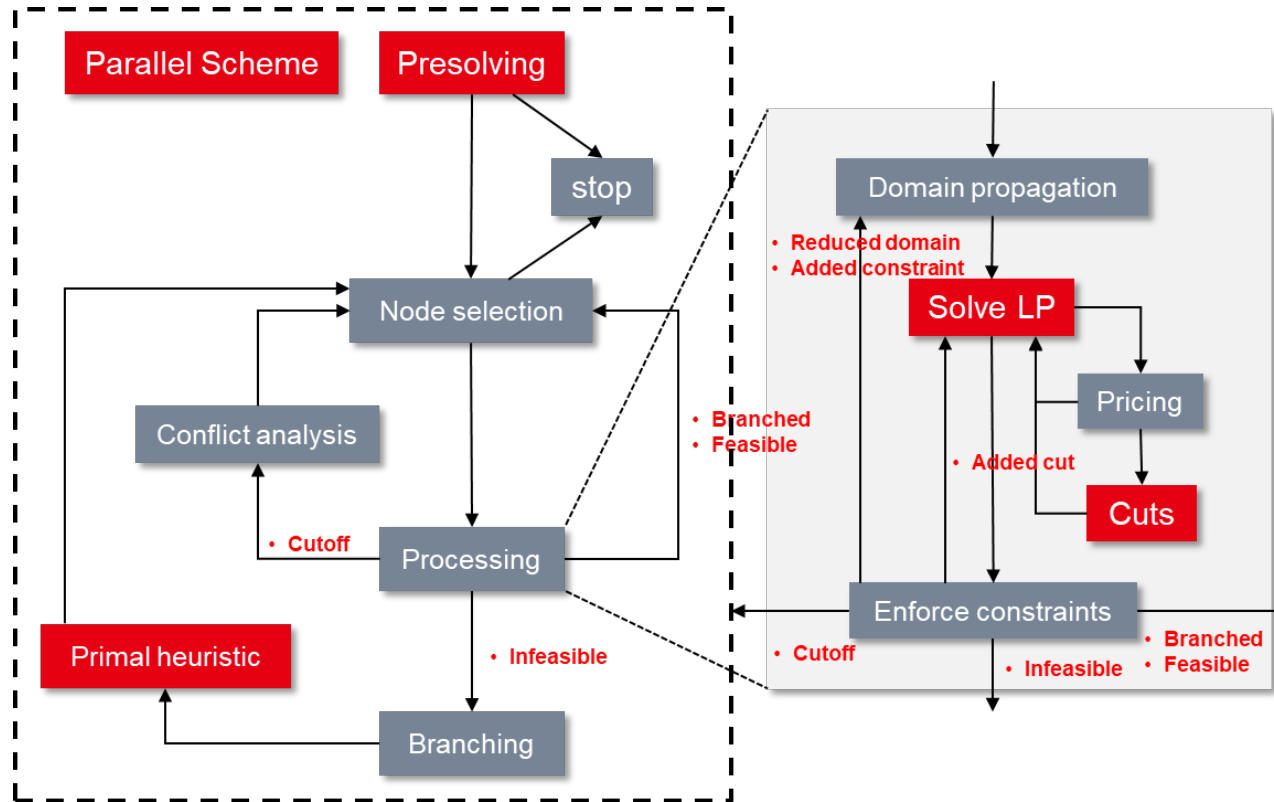
• Upper and lower bound

• Branching

$$\begin{aligned}
 \min \quad & -2x - y \\
 \text{s.t.} \quad & 3x - 5y \leq 0 \\
 & 3x + 5y \leq 15 \\
 & x \geq 0, y \geq 0 \\
 & x, y \in \mathbb{Z}
 \end{aligned}$$

The illustration of branch-and-bound framework

Preliminary



Presolve: reduces the dimension of the input problems
Node selection: determines which node to solve in the B&B tree
Branching: determines which fractional variable to solve in the B&B tree
Cut separation: generates corresponding constraints to shrink the search space
Primal heuristics: aims to find the primal feasible solution ASAP
Solve LP: calls primal and dual simplex to solve LP relaxation
Domain propagation: deduces the narrowed bound for variables and propagates these bounds

Solving procedure of MILP solver

Order matters: why we care



Prof. Andrea Lodi [1]

1. "The performance of MIP solvers is subject to some unexpected variability that appears, for example, when changing from one computing platform to another, when permuting rows and/or columns of a model, etc."
2. "This phenomenon has certainly been observed for decades and has been the topic of an extensive literature in the artificial intelligence and SATisfiability communities."
3. "One source of performance variability is rooted in the so-called imperfect tie breaking. Most of the decisions taken by an MIP solver are based on ordering candidates according to scores and selecting the candidate with the best score. This is true for cut separation and cut filtering as well as for one of the most crucial decisions of the MIP solution process, namely, the variable to branch on at each node."



Dr. Ivet Galabova [2], Developer of HiGHS

For some problems the presolve performance is very sensitive to the order in which the rules are applied. Hence, an analysis of presolve would be incomplete without an investigation of this effect for particular LP instances. Alternative orderings may be more suitable for a problem, depending on the problem structure.

[1] Lodi, Andrea, and Andrea Tramontani. "Performance variability in mixed-integer programming." Theory driven by influential applications. INFORMS, 2013. 1-12.

[2] Galabova, Ivet. "Presolve, crash and software engineering for HiGHS." (2023).

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning[1]

Introduction

An easily overlooked phenomenon: the appearing order of variables in a given LP instance indeed affects the solver's performance. The solver takes input the model constructed by human experts.

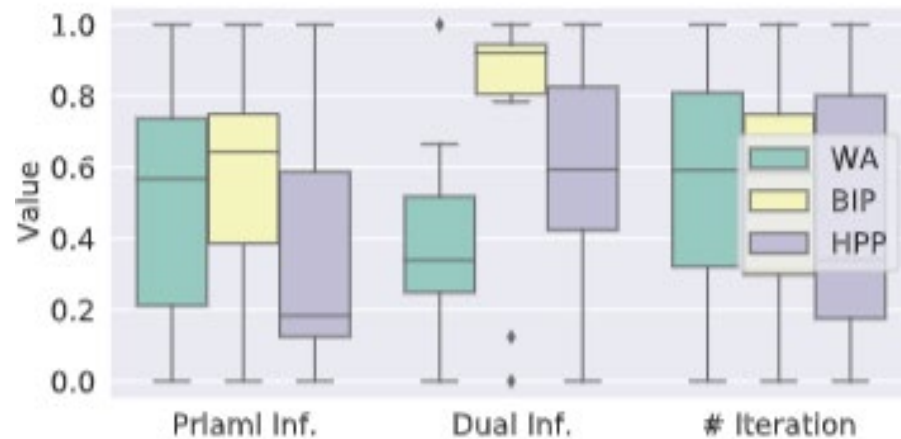


Figure: Three distinct LP instances are selected to perform the preliminary experiment. The significant variance of metrics shows that solver performance is quite sensitive to the different formulations for a given LP instance.

Idx	HPP (m=146722, n=260636, nnz=668270)			WA (m= 64282, n=61000, nnz=359428)			BIP (m=195, n=1083, nnz=7440)		
	Primal Inf.	Dual Inf.	# Iteration	Primal Inf.	Dual Inf.	# Iteration	Primal Inf.	Dual Inf.	# Iteration
1	7.721295E+09	1.884180E+11	19408.00	311.22	1.9980E+09	14083.00	9.43	1.922493E+07	539.00
2	7.720309E+09	1.882416E+11	19242.00	295.33	1.9790E+09	17020.00	9.58	1.937180E+07	486.00
3	7.725508E+09	1.869514E+11	19208.00	269.29	1.9530E+09	16495.00	9.38	1.916960E+07	557.00
4	7.764919E+09	1.876644E+11	19408.00	262.35	2.0620E+09	15149.00	9.47	1.932945E+07	453.00
5	7.726977E+09	1.878485E+11	19175.00	299.10	1.9080E+09	16351.00	9.37	1.915617E+07	586.00
6	7.746786E+09	1.877944E+11	19266.00	281.19	1.8650E+09	14971.00	9.43	1.922493E+07	528.00
7	7.746866E+09	1.879217E+11	19274.00	308.00	1.9630E+09	15778.00	9.22	1.706866E+07	591.00
8	7.729257E+09	1.873669E+11	19391.00	269.86	2.1070E+09	16586.00	9.00	1.880115E+07	519.00
9	7.745738E+09	1.875424E+11	19203.00	296.18	2.0280E+09	14984.00	9.22	1.902808E+07	453.00
10	7.727762E+09	1.882426E+11	19240.00	284.81	2.2290E+09	15861.00	9.05	1.674231E+07	554.00

Motivation: using machine learning technique to automatically find better formulation for solvers.

Three challenges to introduce ML techniques:

- 1) How to appropriately represent an LP instance
- 2) What machine learning model is suitable
- 3) How to efficiently train above inferring model

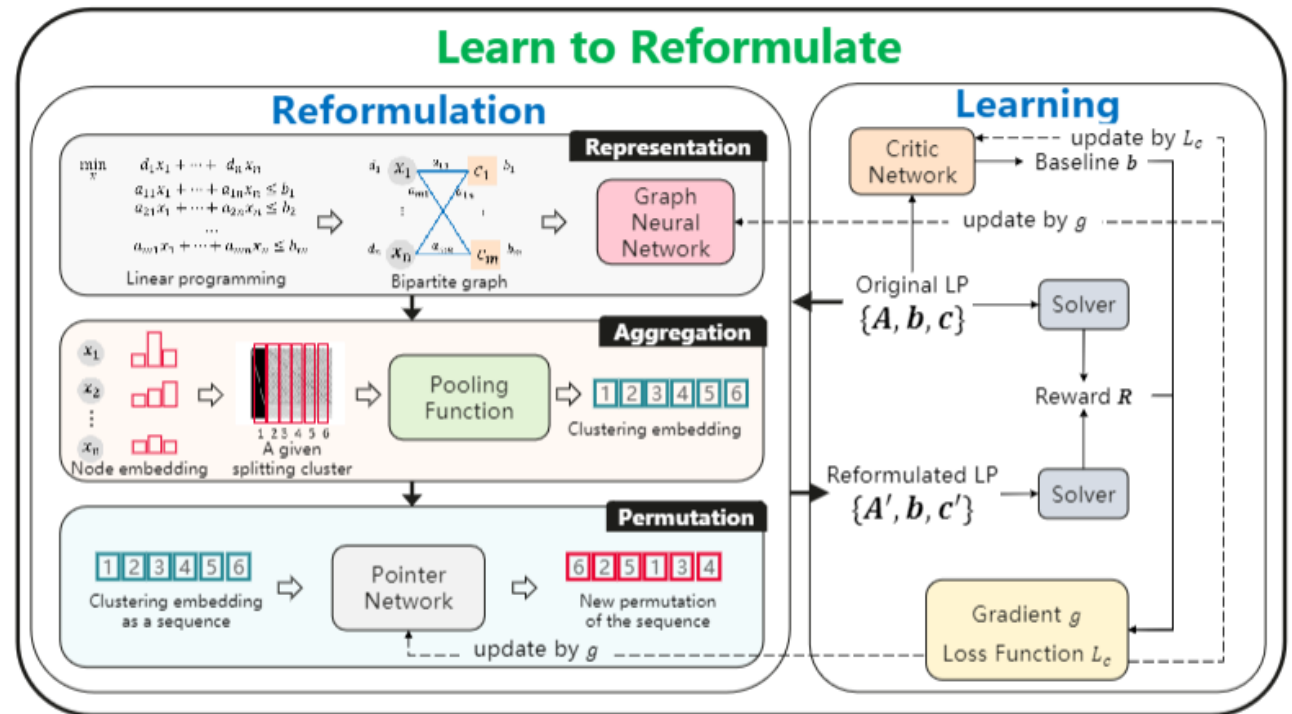
[1] Xijun Li, Qingyu Qu, Fangzhou Zhu, Jia Zeng, Mingxuan Yuan, Jie Wang: Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning. AAAI 2023 workshop on AI to Accelerate Science and Engineering

8 * This work has been filed a patent for Huawei.

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

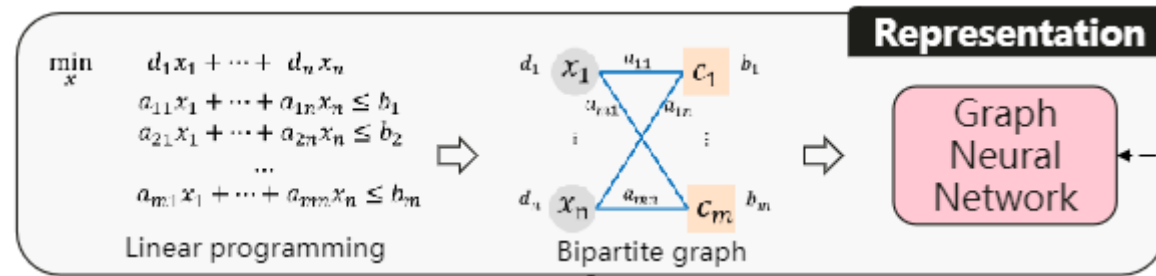
Proposed method: overview

- ❑ **Representation:** The inputting LP instance is represented by a bipartite graph, and then the embedding of variables is obtained via a GNN
- ❑ **Aggregation:** The embedding of variables will be further aggregated with a given group of variable clusters
- ❑ **Permutation:** Taking as input the previous embeddings, a pointer network (PN) is used to output a new permutation of variables
- ❑ **Learning:** The learning part interacts with the reformulation part to update the parameters of GNN and PN, trained with REINFORCE algorithm.



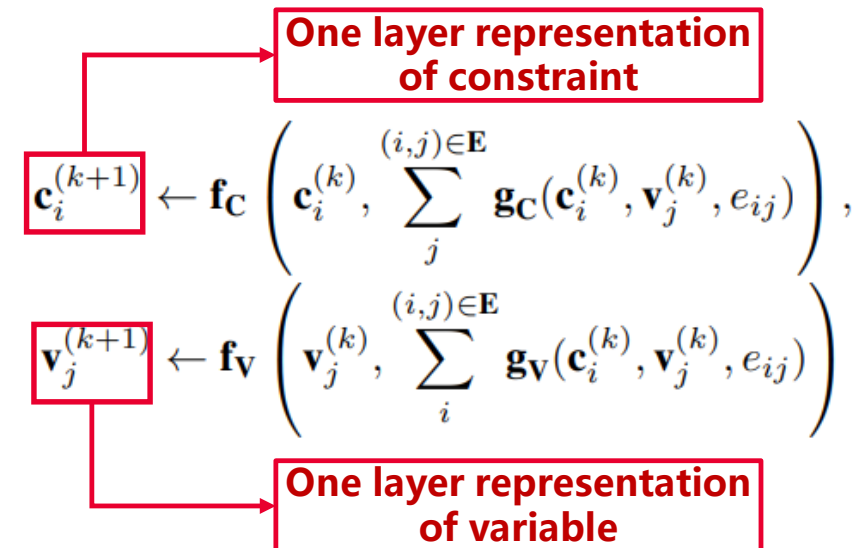
Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Proposed method: reformulation part



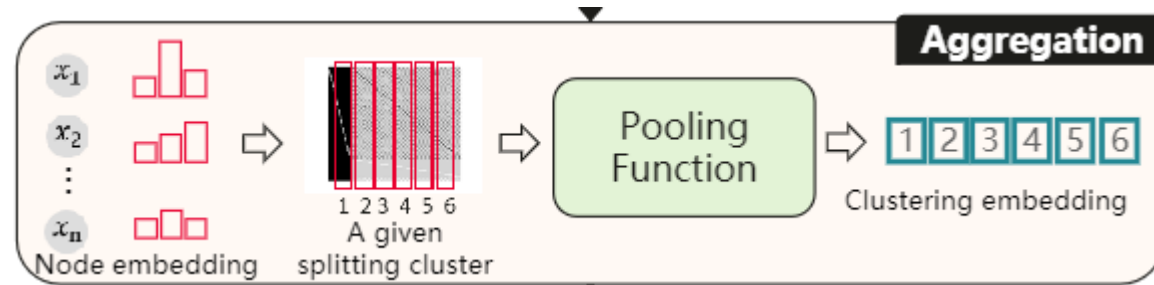
- **Representation:** The inputting LP instance is represented by a bipartite graph and get representation via a graph convolutional neural network (GCNN).

We adopt the same method as done in (Gasse et al. 2019) to represent a given linear programming as a bipartite graph $(\mathcal{G}, \mathbf{C}, \mathbf{E}, \mathbf{V})$. Specifically, in the bipartite graph, $\mathbf{C} \in \mathbb{R}^{m \times c}$ corresponds to the features of the constraints in the LP; $\mathbf{V} \in \mathbb{R}^{n \times d}$ denotes the features of the variables in the LP; and an edge $e_{ij} \in \mathbf{E}$ between a constraint node i and a variable node j if the corresponding coefficient $\mathbf{A}_{i,j} \neq 0$. For simplicity, we just attach the value of $\mathbf{A}_{i,j}$ to the corresponding edge e_{ij} . Readers can refer to the used features in Appendix.



Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Proposed method: reformulation part



□ **Aggregation:** The embedding of variables are aggregated with a given group of variable/constraint clusters **due to the combinatorial explosion of search space**

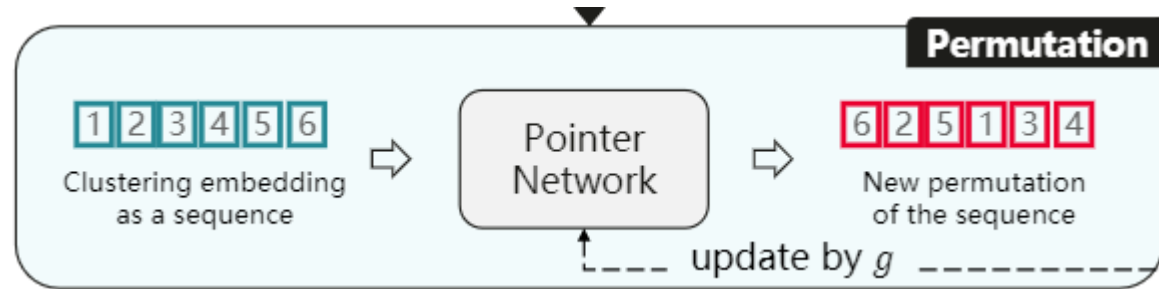
- **Splitting:** For a given LP instance, the variables/constraints are split up into many disjoint cluster. *Note that variables within one cluster are subject to the order of variables in the original LP.* The clustering method is not restricted. *It could be specified by human experts or using the hyper-graph decomposition method*
- **Pooling function:** With above splitting clusters and variable embedding, we perform the aggregation for each cluster via pooling function

$$\Sigma_j = \mathcal{P}(\{\mathbf{v}_i | x_i \in C_j\})$$

where the pooling function that could be maximum, minimum, average, or other appropriate functions. We still do not restrict the kind of pooling function here.

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Proposed method: reformulation part



- **Permutation:** Taking as input the previous embeddings, a pointer network (PN) is used to output a new permutation of variables
 - Given an LP, we reformulate it by reordering its variables/constraints
 - More specifically, we want to learn a policy that can output a better formulation for a given LP

Reformulation policy

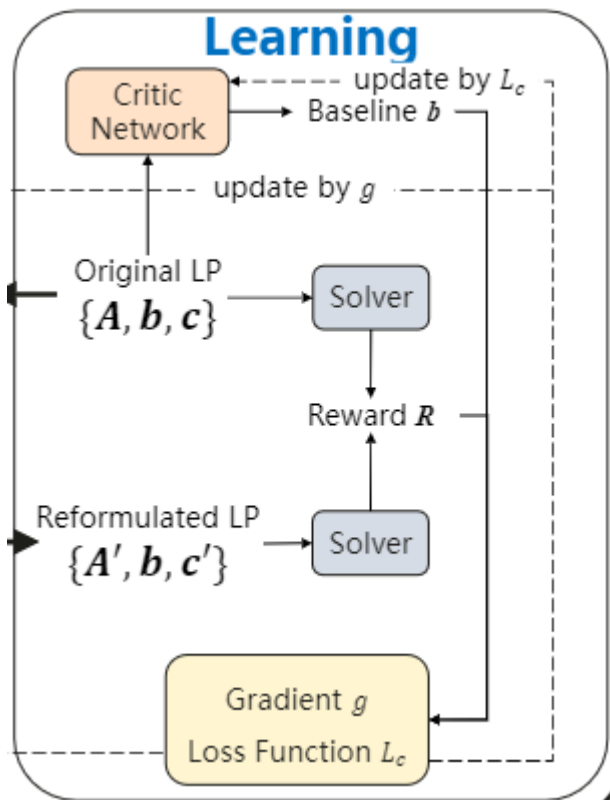
$$P(\pi | \{C_j\}_{j=1}^M) = \prod_{j=1}^M p(\pi(j) | \pi(< j), \{C_j\}_{j=1}^M)$$

a splitting clustering

the permutation of given splitting clustering

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Proposed method: learning part



Policy

$$p(\pi | \{C_j\}_{j=1}^M) = \prod_{j=1}^M p(\pi(j) | \pi(< j), \{C_j\}_{j=1}^M)$$

a splitting clustering

the permutation of given splitting clustering

Reward

$$R(\pi | l) = 1 - \frac{S_{\mathcal{M}}(l|\pi)}{S_{\mathcal{M}}(l)}$$

a given LP instance

the solving performance of the reformulated LP instance

the solving performance of the input LP instance

Optimization

$$J(\theta_G, \theta_P | l) = \mathbb{E}_{\pi \sim p_{\theta_G, \theta_P}(\cdot | l)} [R(\pi | l)]$$

the parameterized representation (i.e., GCN)

the parameterized policy (i.e., pointer network)

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Experimental evaluation: benchmark

- **Datasets:** three sets of Mixed Integer Linear Programming (MILP) problems. All datasets are scenario specific.
 - Two of them, Balanced Item Placement (BIP) and Workload Apportionment (WA), are from NeurIPS 2021 ML4CO competition
 - The third one is obtained from a real-life production planning scenario called HPP.
- **Testing solver:** Gurobi (commercial), SCIP and CLP (open-source)
- **Metrics:** we measure the improvement gained from reformulation
 - Solving time
 - Iteration number of the search process

Table 1: Statistical description of used dataset

Dataset	m	n	NNZ
BIP	195.00 ± 00.00	1083.00 ± 00.00	7440.00 ± 00.00
WA	6.43e04 ± 54.51	6.1e04 ± 00.00	3.62e05 ± 6007.41
HPP	1.25e06 ± 6.93e04	2.66e06 ± 1.83e05	6.64e06 ± 4.29e05

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Experimental evaluation: numerical results

Table 2: Improvement of the iteration number (the lower, the better) over testing instances of three datasets.

Dataset	Solver	Avg. # Iteration		Improv. (%)
		Original	Reformulated	
BIP	CLP	956.32	705.10	26.27
	SCIP	673.58	545.33	19.04
	Gurobi	525.46	428.30	18.49
WA	CLP	6943.37	5985.18	13.8
	SCIP	11354.34	9874.87	13.03
	Gurobi	17962.85	15110.35	15.88
HPP	CLP	98352.27	90277.55	8.21
	SCIP	94124.48	82933.08	11.89
	Gurobi	23526.64	21656.27	7.95

Take-away messages:

- 1) Various solvers (including commercial and open-source) can benefit from reformulation.
- 2) Gurobi is the most robust to the varying formulation.

Table 3: Improvement of solving time (the lower, the better) by our proposed method over all instances of three datasets

Dataset	Seen/Unseen	Solver	Avg. Solving Time (s)		Avg. Reordering Time (s)	Improv. (%)
			Original	Reformulated		
BIP	Seen	CLP	0.03862	0.02851	0.00011	26.18
		SCIP	0.04625	0.03453	0.00012	25.35
		Gurobi	0.01086	0.00849	0.00012	21.78
	Unseen	CLP	0.01792	0.01355	0.00019	24.39
		SCIP	0.04636	0.03472	0.00017	25.11
		Gurobi	0.01144	0.00939	0.00018	17.86
WA	Seen	CLP	6.74099	5.76085	0.00695	14.54
		SCIP	3.96057	3.39302	0.00811	14.33
		Gurobi	5.21913	4.23533	0.00723	18.85
	Unseen	CLP	7.17389	6.17385	0.00689	13.94
		SCIP	4.04960	3.29840	0.00709	18.55
		Gurobi	4.31686	3.59897	0.00685	16.63
HPP	Seen	CLP	71.0665	63.7396	0.01441	10.31
		SCIP	40.4433	34.2676	0.01719	15.27
		Gurobi	43.2238	37.7647	0.01996	12.63
	Unseen	CLP	70.9742	65.9208	0.01334	7.12
		SCIP	40.7626	34.9947	0.01755	14.15
		Gurobi	43.9299	40.0245	0.01454	8.89

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Hindsight: what better formulation the agent found

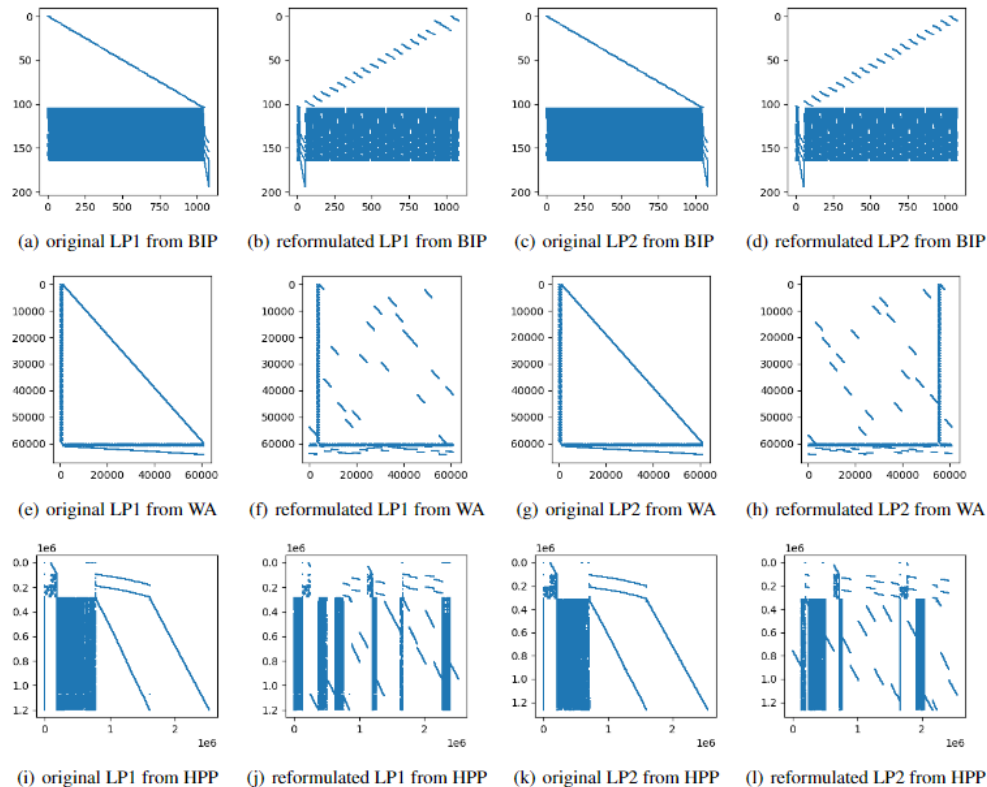


Figure 5: Visualization of reformulation process over BIP, WA and HPP datasets

- 1) **Efficacy:** our proposed reformulation method captures the characteristics of LP instances originating from different scenarios.
- 2) **Different captured patterns on various dataset:** The reformulation is relatively stable when the original LP instances are highly similar. All the original LP instances of BIP have the same number of constraints and variables, which is only different in the value of coefficients. because the corresponding original LP instances differ in not only the value of coefficients but also the number of constraints and variables.

Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning

Summary of this work

- In this paper, we propose a reinforcement learning-based reformulation method to accelerate the linear programming solving. To the best of our knowledge, this is the first work that exploits the performance variability of modern solvers via machine learning techniques to gain performance.
- The idea of using machine learning techniques to exploit or to reduce the performance variability of mathematical programming solvers can be extended in many directions.
 - One can further **learn the modeling experience or gain the modeling tricks** from the reformulation derived from the neural networks.
 - One can use the proposed method to decide the **better ordering rules in various decision-making components in solver such as pricing, variable selection, cut separation**, etc.
 - We believe that this work can inspire the future research to better exploit the performance variability of solvers to improve the solvers.

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

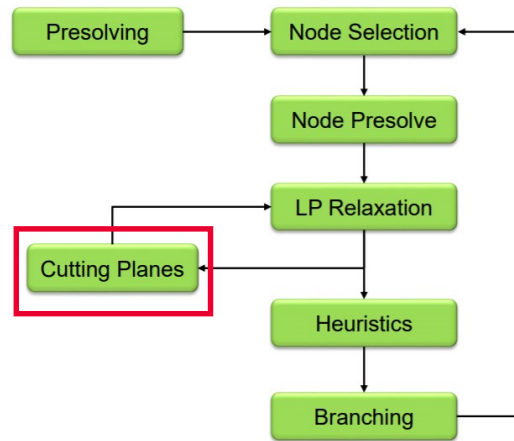


Production Planning



Port Dispatching

- Mixed Integer Programming formulates a wide range of important real-world applications



Pipeline of solving MILPs

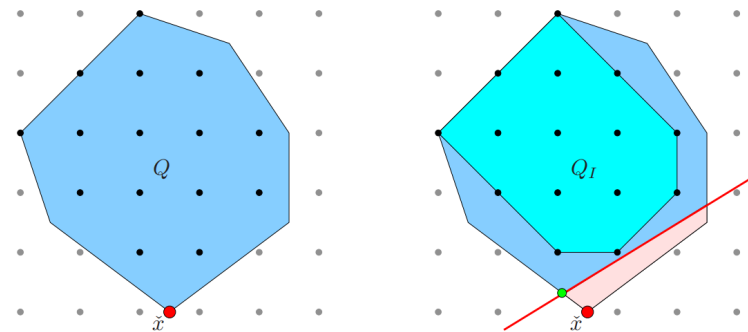


Figure 2.3. A cutting plane that separates the fractional LP solution \bar{x} from the convex hull Q_1 of integer points of Q .

Example of cutting planes

- **Cutting planes** are important for improving the efficiency of solving MILPs
- **Cut selection** aims to select a proper subset of candidate cuts

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Introduction: cut selection

- While many classes of cutting-planes are at the disposal of integer programming solvers, our scientific understanding is far from complete with regards to cutting-plane selection, i.e., **the task of selecting a portfolio of cutting-planes to be added to the LP relaxation** at a given node of the branch-and-bound tree [1].
- The large number of cutting planes generated by these separators, however, can pose a computational problem. Therefore, **a sophisticated cut management is indispensable** [2].
- Adding all of these cuts to the LP does not yield the best overall performance. Therefore, **a selection criterion is needed in order to identify a “good” subset of the generated cuts** [3].

```
Separator Stats :
Separators      :      Time      Calls      VioCuts      UniCuts      Applied      DelPool      DeLLP      AvgVio      AvgLPVio
cutpool         :      5.16         0      28488      20480      2529      16028      1638      0.454045     0.119761
IMPL_0         :      0.22         64      1221      866      335      679      214      0.086314     0.109228
Parity_0       :      0.21         64       83       49       21       33       10      0.146949     0.149105
BinPack_0      :      0.01         64       13       10       11       2        3      0.114480     0.133982
DJ_0           :      0.28         64      114       72       27       55       17      0.101793     0.156583
PATH_0         :      7.19         64     13409     9816     1008     8247     639     0.645638     0.143585
ModK_0         :      0.74         50      744      543      181      419     112     0.181118     0.250318
AGGR_0         :      4.36         43       3        3        3        1        1      4.202781     0.745435
TwoRowDJ_0     :      2.23         50      154      130      10      123       4      0.161719     0.170186
GMI_0          :      7.36         50     10595     6878     179     6469     117     0.342388     6.643384
Presolve_0     :      0.00         64     2152     2113     754       0      521     0.115196     0.144838
```

Snapshot of OptVerse Solver 's Log

The task of selecting a portfolio of cutting-planes is a NP-hard problem !

Cut selection: trade-off between tightening LP relaxations and computational burden of cuts

[1] Wesselmann, Franz, and U. Stuhl. "Implementing cutting plane management and selection techniques." Technical Report. University of Paderborn, 2012.

[2] Dey, Santanu, and Marco Molinaro. "Theoretical challenges towards cutting-plane selection." Mathematical Programming 170 (2018): 237-266.

[3] Achterberg, Tobias. Constraint integer programming. Diss. 2007.

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Limitations of existing methods: heuristics



Prof. Yoshua Bengio [1]



Prof. Andrea Lodi [1]

1. However, assuming that MILPs come from a specific distribution, there has been a **recent surge in research related to statistical approaches to learning such heuristics**.
2. From the CO point of view, **machine learning can help improve an algorithm** on a distribution of problem instances in two ways. On the one side, On the other side,
3. Indeed, learning mechanisms able to **discover structural properties of seemingly equivalent components** of an algorithm would certainly be very useful

Machine learning offers a promising approach to learn more effective heuristics from MILPs collected from specific applications

[1] Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d' horizon." *European Journal of Operational Research* 290.2 (2021): 405-421.

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Limitations of existing methods: learning-based

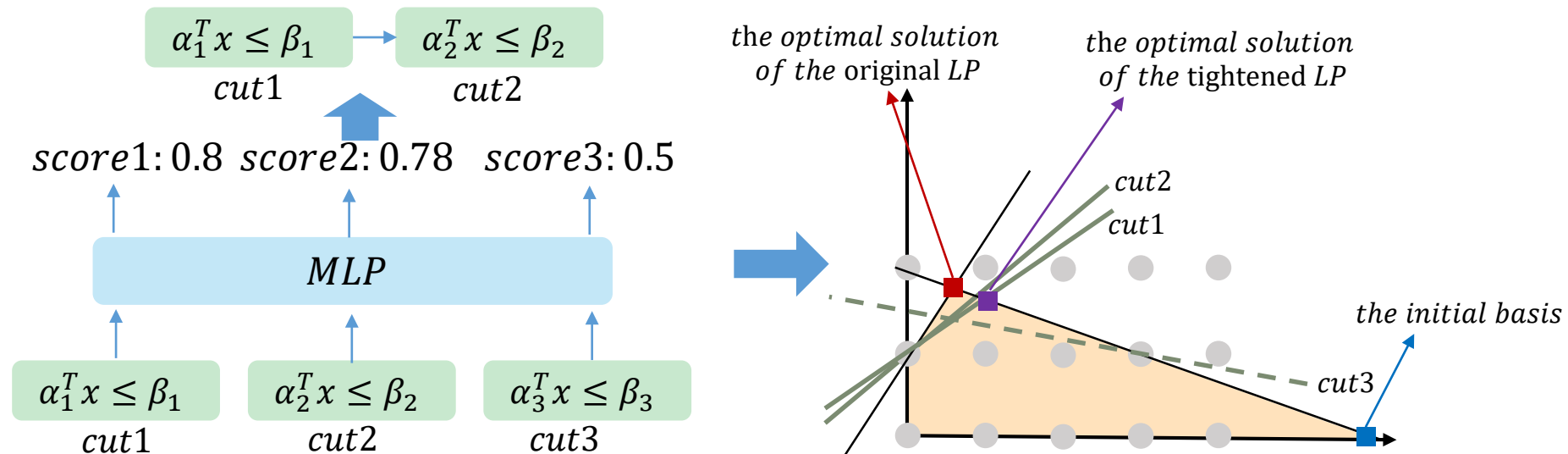
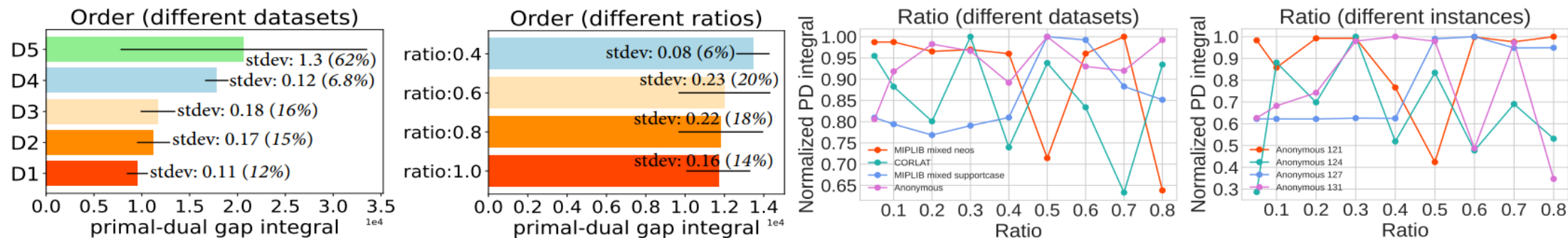


Illustration of Score-based policy

- Neglecting learning **how many cuts to select**
- **Order of selected cuts** is important for the solving efficiency
- Do not take into account **the interaction among cuts**

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Three key problems in cut selection



(a) Evaluate RandomAll on five different datasets. (b) Evaluate RandomNv on MIPLIB mixed neos. (c) Evaluate NV with different ratios on four datasets. (d) Evaluate NV with different ratios on Anonymous.

Fig. 1: (a)-(b) We design two cut selection heuristics, namely RandomAll and RandomNV (see Section 4.1 for details), which both add the same subset of cuts in random order for a given MILP. The results in (a) and (b) show that adding the same selected cuts in different order leads to variable overall solver performance. (c)-(d) We use the Normalized Violation (NV) heuristics [21] in the following experiments. The results in (c) show that the performance of NV varies widely with the given ratios across different datasets. The results in (d) show that the performance of NV varies widely with the given ratios across different instances from the Anonymous dataset.

(P1) Which cuts should be preferred (P2) How many cuts to select (P3) What **order** to prefer (**we first observe**)

Order and ratio of selected cuts significantly impact the efficiency of solving MILPs

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Proposed method: overview

□ Reinforcement Learning Formulation

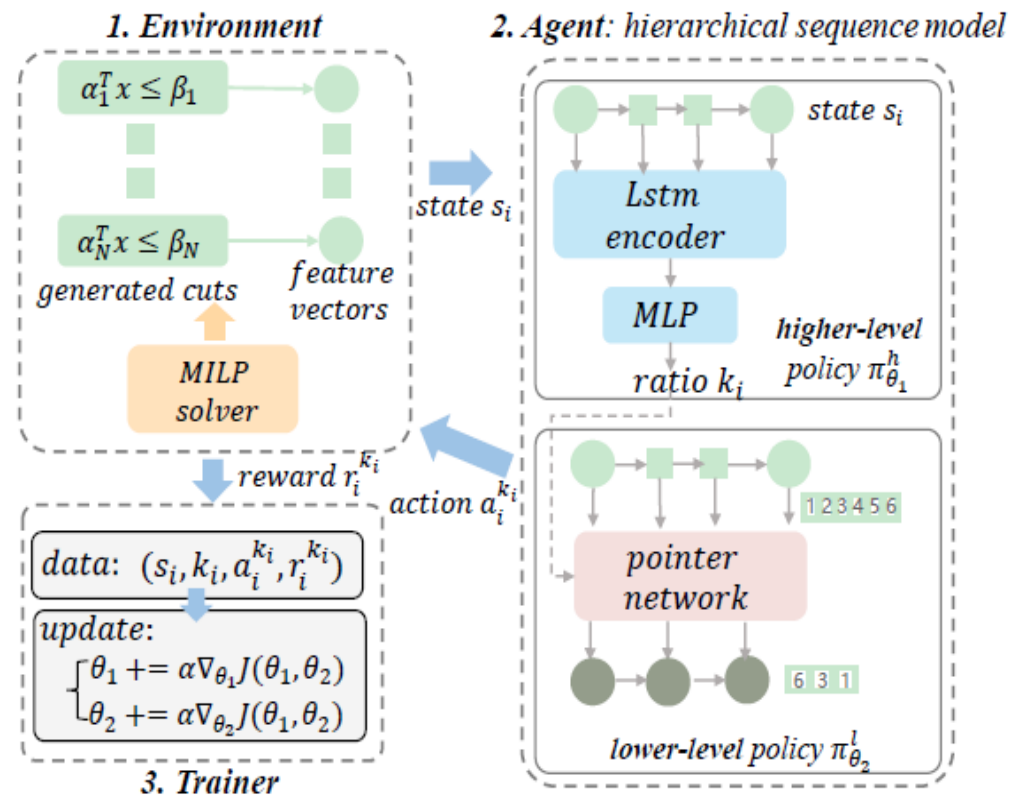
- Formulate MILP solver as the environment
- Formulate cut selection policy as the agent
- Time/Primal-dual gap integral as reward

□ Hierarchical Sequence Model

- We **first** formulate cut selection as a sequence to sequence learning problem
- Propose Hierarchical Sequence Model to model the agent

□ Training Algorithm

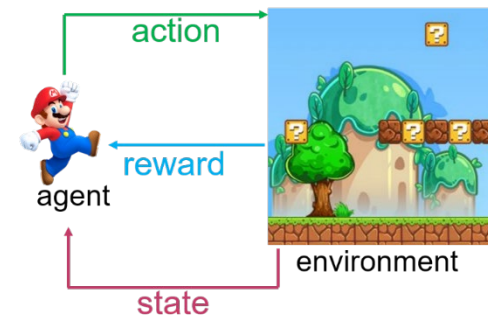
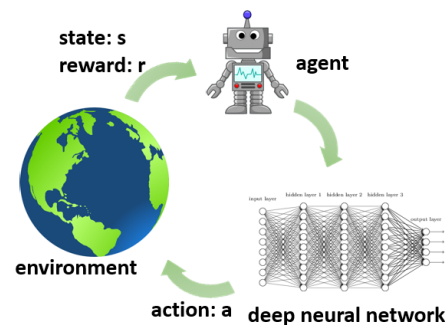
- We derive a hierarchical policy gradient



Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Proposed method: *Markov* decision process formulation

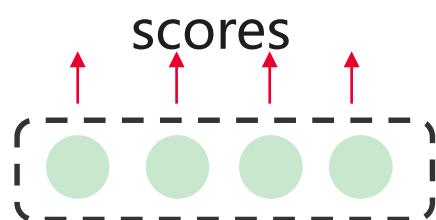
- State space S : We define each state by a **sequence** of thirteen-dimensional feature vectors (see Appendix F.1).
- Action space A : All the **ordered subsets** of the candidate cuts
- Reward function r : **Solving time/Primal-dual gap integral/Dual bound improvement**
- Transition function f : A **deterministic** mapping from the current state and action to the next state
- The terminal state. There is no standard and unified criterion to determine when to terminate the cut separation procedure.
 - The number of cut separation rounds is a **hyperparameter** in MILP solvers.
 - We set the cut separation rounds as one and add cuts at the root node in this paper.
 - As a result, the formulation is a **contextual bandit**.



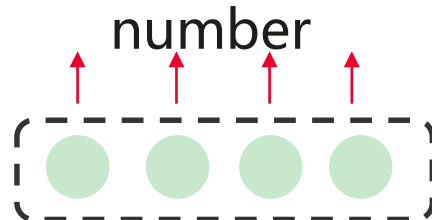
Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Proposed method: cut selection policy

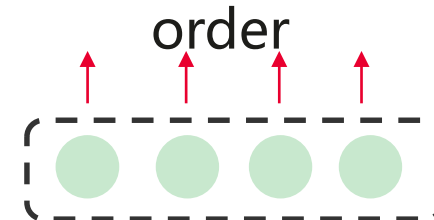
- Cut selection policy: $\pi: S \rightarrow P(A)$, where $P(A)$ denotes the **probability distribution** over the action space
 - Tackle (P1)-(P3) simultaneously
 - Exploration and differentiability
- Directly learning such policies is challenging
 - the **cardinality** of the action space can be extremely **large** due to its combinatorial structure
 - the **length** and max length of actions are **variable** across different MILPs



(P1) Which cuts to prefer



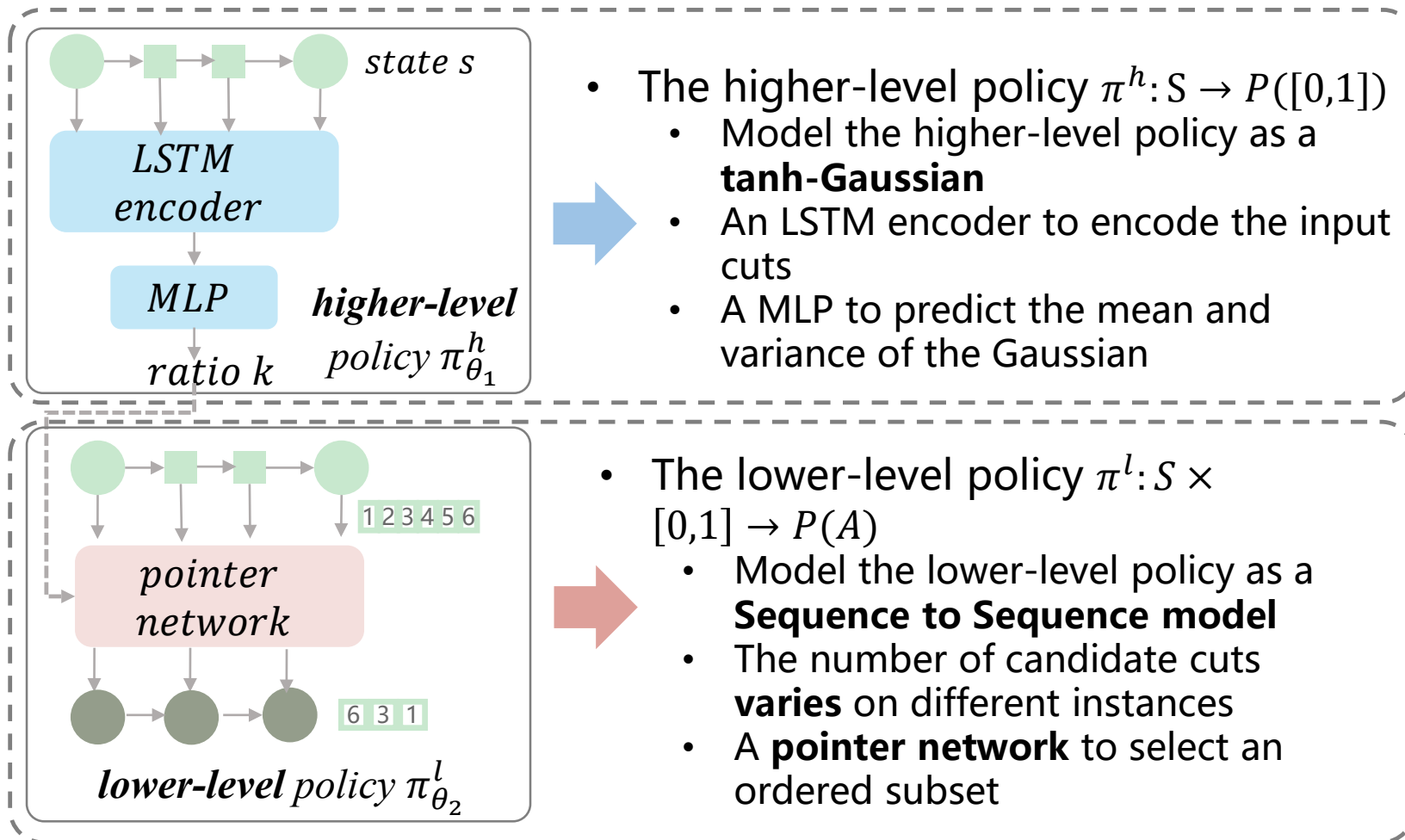
(P2) How many cuts to select



(P3) What **order** of selected cuts to prefer (**Our observation**)

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Proposed method: hierarchical sequence model



Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Proposed method: hierarchical policy gradient

$$J(\theta) = \mathbb{E}_{s \sim \mu, a_k \sim \pi_\theta(\cdot|s)} [r(s, a_k)],$$

Optimization
objective

Proposition 1. Given the cut selection policy $\pi_\theta(a_k|s) = \mathbb{E}_{k \sim \pi_{\theta_1}^h(\cdot|s)} [\pi_{\theta_2}^l(a_k|s, k)]$ and the training objective (3), the hierarchical policy gradient takes the form of

$$\nabla_{\theta_1} J([\theta_1, \theta_2]) = \mathbb{E}_{s \sim \mu, k \sim \pi_{\theta_1}^h(\cdot|s)} [\nabla_{\theta_1} \log(\pi_{\theta_1}^h(k|s)) \mathbb{E}_{a_k \sim \pi_{\theta_2}^l(\cdot|s, k)} [r(s, a_k)]],$$

$$\nabla_{\theta_2} J([\theta_1, \theta_2]) = \mathbb{E}_{s \sim \mu, k \sim \pi_{\theta_1}^h(\cdot|s), a_k \sim \pi_{\theta_2}^l(\cdot|s, k)} [\nabla_{\theta_2} \log \pi_{\theta_2}^l(a_k|s, k) r(s, a_k)].$$

Hierarchical policy gradient

- HEM leverages the hierarchical structure of the cut selection task, which is important for **efficient exploration** in complex decision-making tasks
- We train HEM via gradient-based algorithms, which is **sample efficient**

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Experimental evaluation: benchmark

- Datasets: **nine** MILP benchmarks
 - **Easy**: Set Covering, Maximum Independent Set, Knapsack (synthetic)
 - **Medium**: MIK (knapsack constraints), CORLAT (a real dataset used for the construction of a wildlife corridor)
 - **Hard**: Load balancing (**Google**), Anonymous (a large-scale industrial application), **MIPLIB 2017**
- Baselines
 - **five** widely used human-designed cut selection rules
 - NoCuts, Random, Normalized Violation (NV), Efficacy (Eff), and Default
 - a state-of-the-art (SOTA) learning-based method, score-based policy (SBP)
- Metrics
 - Time: solving time
 - PD integral: **Primal-dual gap integral**

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Experimental evaluation: significantly improves the efficiency

TABLE 2: Policy evaluation on easy, medium, and hard datasets. The best performance are marked in bold. For all datasets, the results show that HEM significantly outperforms the baselines in terms of solving time and primal-dual gap integral.

Easy: Set Covering ($n = 1000, m = 500$)			Easy: Max Independent Set ($n = 500, m = 1953$)			Easy: Multiple Knapsack ($n = 720, m = 72$)			
Method	Time(s) ↓	Improvement (time, %) ↑	PD integral ↓	Time(s) ↓	Improvement (time, %) ↑	PD integral ↓	Time(s) ↓	Improvement (time, %) ↑	PD integral ↓
NoCuts	6.31 (4.61)	NA	56.99 (38.89)	8.78 (6.66)	NA	71.31 (51.74)	9.88 (22.24)	NA	16.41 (14.16)
Default	4.41 (5.12)	29.90	55.63 (42.21)	3.88 (5.04)	55.80	29.44 (35.27)	9.90 (22.24)	-0.20	16.46 (14.25)
Random	5.74 (5.19)	8.90	67.08 (46.58)	6.50 (7.09)	26.00	52.46 (53.10)	13.10 (35.51)	-32.60	20.00 (25.14)
NV	9.86 (5.43)	-56.50	99.77 (53.12)	7.84 (5.54)	10.70	61.60 (43.95)	13.04 (36.91)	-32.00	21.75 (24.71)
Eff	9.65 (5.45)	-53.20	95.66 (51.71)	7.80 (5.11)	11.10	61.04 (41.88)	9.99 (19.02)	-1.10	20.49 (22.11)
SBP	1.91 (0.36)	69.60	38.96 (8.66)	2.43 (5.55)	72.30	21.99 (40.86)	7.74 (12.36)	21.60	16.45 (16.62)
HEM (Ours)	1.85 (0.31)	70.60	37.92 (8.46)	1.76 (3.69)	80.00	16.01 (26.21)	6.13 (9.61)	38.00	13.63 (9.63)

Medium: MIK ($n = 413, m = 346$)			Medium: Corlat ($n = 466, m = 486$)			Hard: Load Balancing ($n = 61000, m = 64304$)			
Method	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)
NoCuts	300.01 (0.009)	2355.87 (996.08)	NA	103.30 (128.14)	2818.40 (5908.31)	NA	300.00 (0.12)	14853.77 (951.42)	NA
Default	179.62 (122.36)	844.40 (924.30)	64.10	75.20 (120.30)	2412.09 (5892.88)	14.40	300.00 (0.06)	9589.19 (1012.95)	35.40
Random	289.86 (28.90)	2036.80 (933.17)	13.50	84.18 (124.34)	2501.98 (6031.43)	11.20	300.00 (0.09)	13621.20 (1162.02)	8.30
NV	299.76 (1.32)	2542.67 (529.49)	-7.90	90.26 (128.33)	3075.70 (7029.55)	-9.10	300.00 (0.05)	13933.88 (971.10)	6.20
Eff	298.48 (5.84)	2416.57 (642.41)	-2.60	104.38 (131.61)	3155.03 (7039.99)	-11.90	300.00 (0.07)	13913.07 (969.95)	6.30
SBP	286.07 (41.81)	2053.30 (740.11)	12.80	70.41 (122.17)	2023.87 (5085.96)	28.20	300.00 (0.10)	12535.30 (741.43)	15.60
HEM(Ours)	176.12 (125.18)	785.04 (790.38)	66.70	58.31 (110.51)	1079.99 (2653.14)	61.68	300.00 (0.04)	9496.42 (1018.35)	36.10

Hard: Anonymous ($n = 37881, m = 49603$)			Hard: MIPLIB mixed neos ($n = 6958, m = 5660$)			Hard: MIPLIB mixed supportcase ($n = 19766, m = 19910$)			
Method	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)
NoCuts	246.22 (94.90)	18297.30 (9769.42)	NA	253.65 (80.29)	14652.29 (12523.37)	NA	170.00 (131.60)	9927.96 (11334.07)	NA
Default	244.02 (97.72)	17407.01 (9736.19)	4.90	256.58 (76.05)	14444.05 (12347.09)	1.42	164.61 (135.82)	9672.34 (10668.24)	2.57
Random	243.49 (98.21)	16850.89 (10227.87)	7.80	255.88 (76.65)	14006.48 (12698.76)	4.41	165.88 (134.40)	10034.70 (11052.73)	-1.07
NV	242.01 (98.68)	16873.66 (9711.16)	7.80	263.81 (64.10)	14379.05 (12306.35)	1.86	161.67 (131.43)	8967.00 (9690.30)	9.68
Eff	244.94 (93.47)	17137.87 (9456.34)	6.30	260.53 (68.54)	14021.74 (12859.41)	4.30	167.35 (134.99)	9941.55 (10943.48)	-0.14
SBP	245.71 (92.46)	18188.63 (9651.85)	0.59	256.48 (78.59)	13531.00 (12898.22)	7.65	165.61 (135.25)	7408.65 (7903.47)	25.37
HEM(Ours)	241.68 (97.23)	16077.15 (9108.21)	12.10	248.66 (89.46)	8678.76 (12337.00)	40.77	162.96 (138.21)	6874.80 (6729.97)	30.75

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Experimental evaluation: ablation studies

Table 2: Comparison between HEM and HEM without the higher-level model.

Method	Easy: Maximum Independent Set ($n = 500, m = 1953$)			Medium: Corlat ($n = 466, m = 486$)			Hard: MIPLIB mixed neos ($n = 6958, m = 5660$)		
	Time(s) ↓	Improvement ↑ (Time, %)	PD integral ↓	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)
NoCuts	8.78 (6.66)	NA	71.31 (51.74)	103.30 (128.14)	2818.40 (5908.31)	NA	253.65 (80.29)	14652.29 (12523.37)	NA
Default	3.88 (5.04)	55.81	29.44 (35.27)	75.20 (120.30)	2412.09 (5892.88)	14.42	256.58 (76.05)	14444.05 (12347.09)	1.42
SBP	2.43 (5.55)	72.32	21.99 (40.86)	70.41 (122.17)	2023.87 (5085.96)	28.19	256.48 (78.59)	13531.00 (12898.22)	7.65
HEM w/o H	1.88 (4.20)	78.59	16.70 (28.15)	63.14 (115.26)	1939.08 (5484.83)	31.20	249.21 (88.09)	13614.29 (12914.76)	7.08
HEM (Ours)	1.76 (3.69)	79.95	16.01 (26.21)	58.31 (110.51)	1079.99 (2653.14)	61.68	248.66 (89.46)	8678.76 (12337.00)	40.77

Each component of HEM is important for the solving efficiency.

Table 3: Comparison between HEM, HEM-ratio, and HEM-ratio-order.

Method	Easy: Maximum Independent Set ($n = 500, m = 1953$)			Medium: Corlat ($n = 466, m = 486$)			Hard: MIPLIB mixed neos ($n = 6958, m = 5660$)		
	Time(s) ↓	Improvement ↑ (Time, %)	PD integral ↓	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)	Time(s) ↓	PD integral ↓	Improvement ↑ (PD integral, %)
NoCuts	8.78 (6.66)	NA	71.31 (51.74)	103.30 (128.14)	2818.40 (5908.31)	NA	253.65 (80.29)	14652.29 (12523.37)	NA
Default	3.88 (5.04)	55.81	29.44 (35.27)	75.20 (120.30)	2412.09 (5892.88)	14.42	256.58 (76.05)	14444.05 (12347.09)	1.42
SBP	2.43 (5.55)	72.32	21.99 (40.86)	70.41 (122.17)	2023.87 (5085.96)	28.19	256.48 (78.59)	13531.00 (12898.22)	7.65
HEM-ratio-order	2.30 (5.18)	73.80	21.19 (38.52)	70.94 (122.93)	1416.66 (3380.10)	49.74	245.99 (93.67)	14026.75 (12683.60)	4.27
HEM-ratio	2.26 (5.06)	74.26	20.82 (37.81)	67.27 (117.01)	1251.60 (2869.87)	55.59	244.87 (95.56)	13659.93 (12900.59)	6.77
HEM (Ours)	1.76 (3.69)	79.95	16.01 (26.21)	58.31 (110.51)	1079.99 (2653.14)	61.68	248.66 (89.46)	8678.76 (12337.00)	40.77

Tackling (P1)-(P3) is important for the solving efficiency

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Experimental evaluation: generalize to instances larger than training data

TABLE 8: Evaluate the generalization ability of HEM on Set Covering and Maximum Independent Set.

Set Covering ($n = 1000, m = 1000, 2\times$)				Set Covering ($n = 1000, m = 2000, 4\times$)		
Method	Time(s) ↓	Improvement ↑ (time, %)	PD integral ↓	Time(s) ↓	Improvement ↑ (time, %)	PD integral ↓
NoCuts	82.69 (78.27)	NA	609.43 (524.92)	284.44 (48.70)	NA	3215.34 (1019.47)
Default	61.01 (78.12)	26.22	494.63 (545.76)	149.69 (141.92)	47.37	1776.22 (1651.15)
Random	64.44 (73.98)	22.07	520.84 (489.52)	208.12 (131.52)	26.53	2528.36 (1678.66)
NV	92.05 (80.11)	-11.32	725.53 (541.68)	286.10 (45.47)	-0.58	3422.46 (1024.19)
Eff	92.32 (79.33)	-11.64	733.72 (538.60)	286.20 (45.04)	-0.62	3437.06 (1043.44)
SBP	3.52 (1.36)	95.74	92.89 (25.83)	7.62 (6.46)	97.32	256.79 (145.92)
HEM (Ours)	3.33 (0.47)	95.97	89.24 (14.26)	7.40 (5.03)	97.40	250.83 (131.43)

Maximum Independent Set ($n = 1000, m = 3946, 4\times$)				Maximum Independent Set ($n = 5940, m = 1500, 9\times$)		
Method	Time(s) ↓	Improvement ↑ (time, %)	PD integral ↓	Time(s) ↓	Improvement ↑ (time, %)	PD integral ↓
NoCuts	170.06 (100.61)	NA	874.45 (522.29)	300.00 (0)	NA	2019.93 (353.27)
Default	42.40 (76.00)	48.72	198.61 (331.20)	111.18 (144.13)	60.91	616.46 (798.94)
Random	118.25 (109.05)	-43.00	574.33 (516.11)	245.13 (115.80)	13.82	1562.20 (793.09)
NV	160.30 (101.41)	-93.86	784.98 (493.24)	299.97 (0.49)	-5.46	1922.52 (349.67)
Eff	158.75 (100.40)	-91.98	779.63 (493.05)	299.45 (3.77)	-5.28	1921.61 (361.26)
SBP	50.55 (89.14)	38.87	253.81 (426.94)	108.42 (143.68)	61.88	680.41 (903.88)
HEM (Ours)	35.34 (67.91)	57.26	160.56 (282.03)	108.02 (143.02)	62.02	570.48 (760.65)

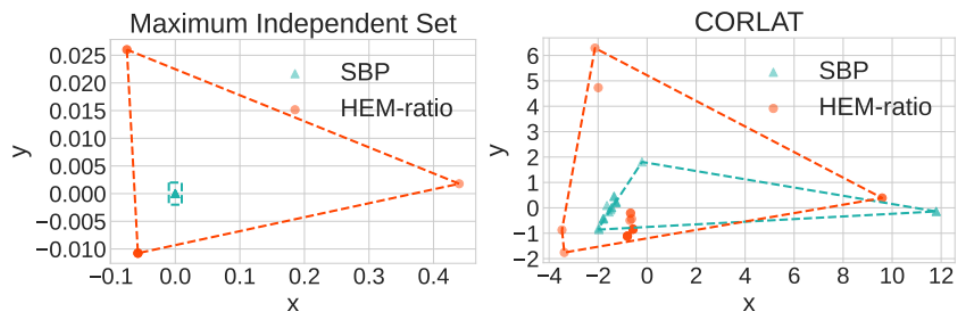
Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Evaluation on Huawei production planning and order matching problems

TABLE 9: Evaluation on Huawei production planning and order matching problems.

Method	Production planning ($n = 3582.25, m = 5040.42$)				Order matching ($n = 67839.68, m = 31364.84$)			
	Time (s) ↓	Improvement ↑ (Time, %)	PD integral ↓	Improvement ↑ (PD integral, %)	Time (s) ↓	Improvement ↑ (Time, %)	PD integral ↓	Improvement ↑ (PD integral, %)
NoCuts	278.79 (231.02)	NA	17866.01 (21309.85)	NA	248.42 (287.29)	NA	403.41 (345.51)	NA
Default	296.12 (246.25)	-6.22	17703.39 (21330.40)	0.91	129.34 (224.24)	47.93	395.80 (341.23)	1.89
Random	280.18 (237.09)	-0.50	18120.21 (21660.01)	-1.42	95.76 (202.23)	61.45	406.73 (348.30)	-0.82
NV	259.48 (227.81)	6.93	17295.18 (21860.07)	3.20	245.61 (282.04)	1.13	406.69 (347.72)	-0.81
Eff	263.60 (229.24)	5.45	16636.52 (21322.89)	6.88	243.30 (276.65)	2.06	417.71 (360.13)	-3.54
SBP	276.61 (235.84)	0.78	16952.85 (21386.07)	5.11	44.14 (148.60)	82.23	379.23 (326.86)	5.99
HEM (Ours)	241.77 (229.97)	13.28	15751.08 (20683.53)	11.84	43.88 (148.66)	82.34	368.51 (309.02)	8.65

Two real applications, around **10%** improvement on average



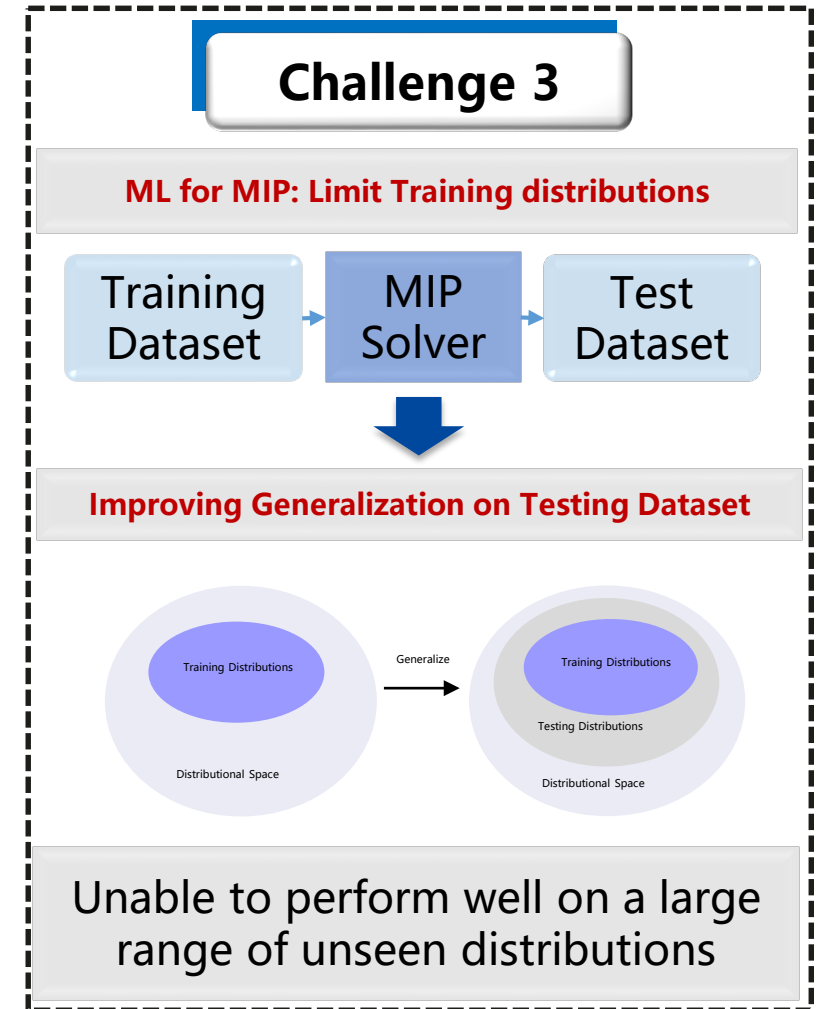
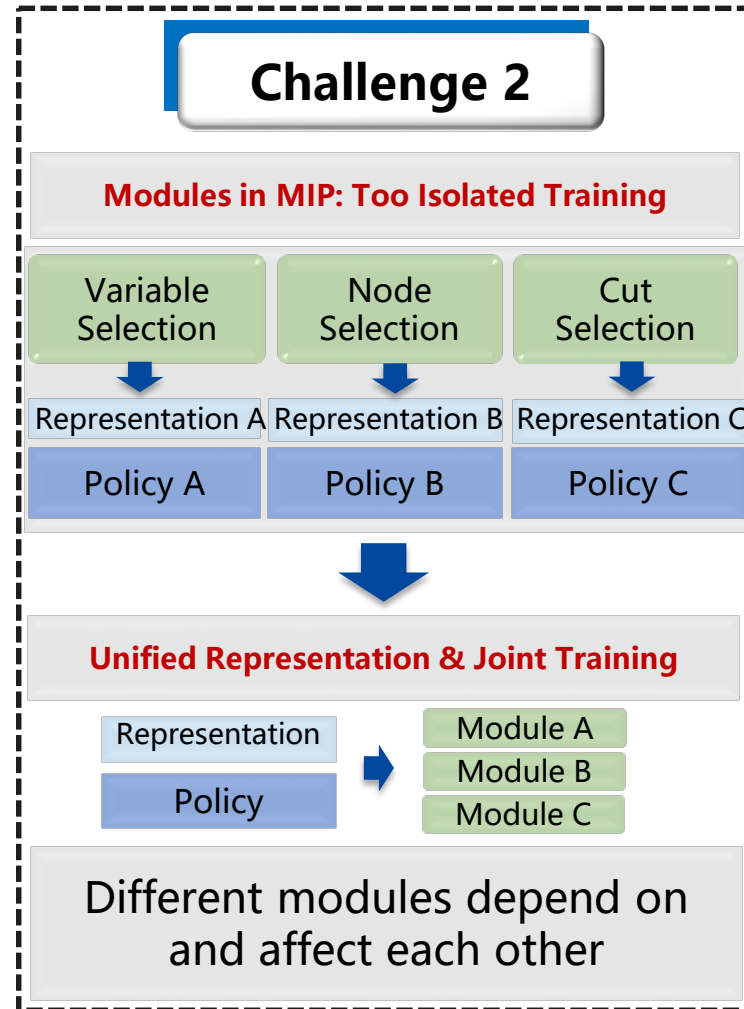
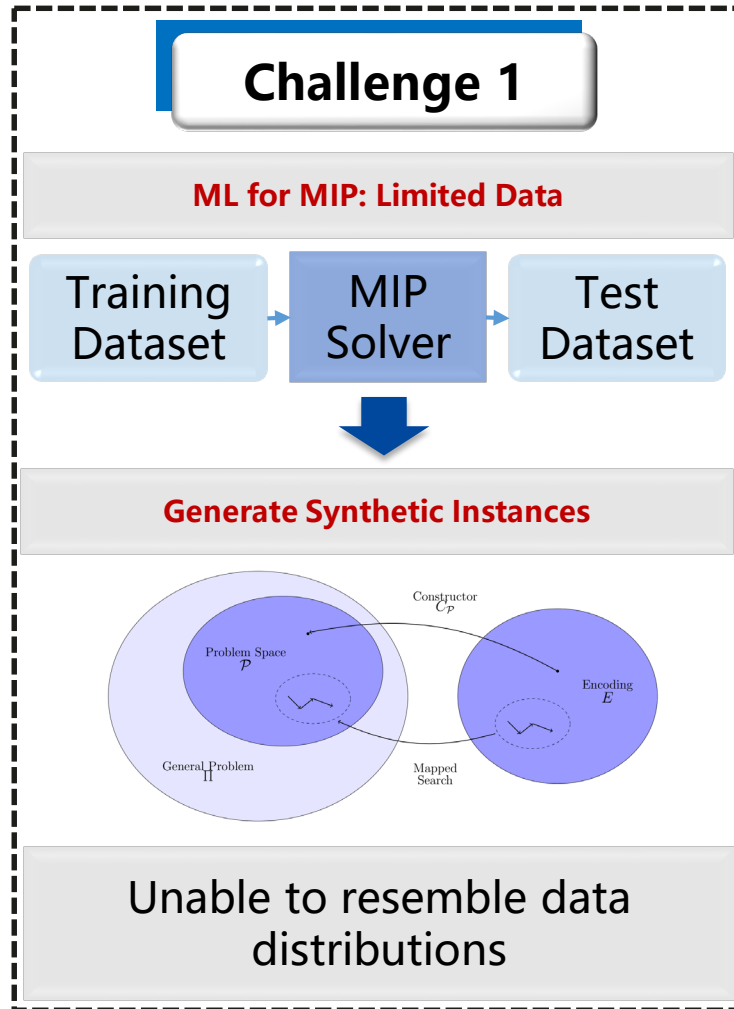
- Visualize the diversity of selected cuts
- HEM selects much more diverse cuts that can well complement each other nicely

Learning Cut Selection for Mixed Integer Programming via Hierarchical Sequence Model

Summary of this work

- We **observe** from extensive empirical results that the **order** of selected cuts has a significant impact on the efficiency of solving MILPs.
- To the best of our knowledge, our proposed HEM is **the first** method that is able to tackle (P1)-(P3) in cut selection simultaneously from a data-driven perspective.
- We propose to formulate the cut selection task as a **sequence to sequence learning problem**, which not only can capture the underlying order information, but also well captures the interaction among cuts to select cuts that complement each other nicely.
- Extensive experiments demonstrate that HEM **achieves significant improvements** over competitive baselines on challenging MILP problems, including some benchmarks from MIPLIB 2017 and large-scale real-world production planning problems.

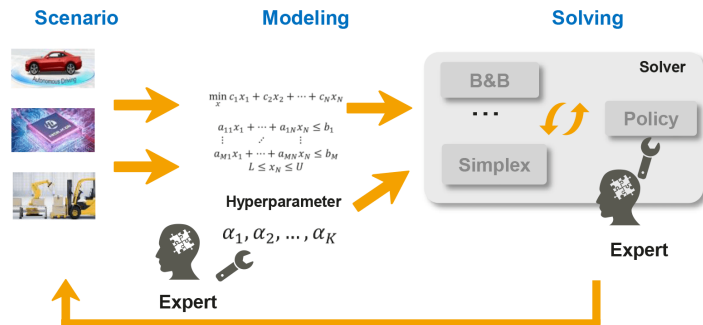
Challenges and outlook: from learning to game



[Solver] AI Solver with Automatic Algorithmic Reasoning

华为黄大年茶思屋挑战难题

<https://www.chaspark.com/#/questions/828669011084787712?sub=828669011097370624>



Mathematical programming solver, which is a very important tool for solving optimization problems, is widely used in numerous fields, such as chip design, autonomous driving, resource scheduling, and supply chain optimization. Solving optimization problems in reality usually consists of two parts: modeling and solving. Optimizing both parts requires the experience of experts and a lot of human effort. Modeling requires a good understanding of the background of the optimization problems — this is a prerequisite for building a model that the business needs. Solving requires a deep understanding of the optimization algorithm and lots of solving policies for different problem structures so that the solver can achieve fast and accurate results in many scenarios.

Generally, two layers of policies will help optimize the solver performance:

- **Outer policy [1]:** Based on the structures of domain-specific problems, experts select a group of optimal parameters out of massive hyperparameters and input them into the solver, or the solver developers set a group of good default parameters out of expert experience.
- **Inner policy:** The solver developers translate expert experience into the following work in order to improve solver stability and performance:
 - ✓ Design an algorithm to discover solver-friendly model structures (such as model rearrangement [3, 4] and massive structure mining).
 - ✓ Design and reason efficient solving policies manually [2] (e.g., branch variable selection and cutting plane selection).

Compared with the outer policy, research on the inner policy is still in its infancy. With the abundant data accumulated in enterprises, we can use AI technologies to implement automatic algorithm reasoning of the solver's inner policy, thus reducing dependency on expert experience and enhancing the solver's competitiveness.

Technical Challenges

- **Solver-friendly model structure discovery algorithms:**
 - ✓ **Low automation rate:** Structure analysis depends heavily on optimization experts' understanding of the problem and cannot be adaptively implemented by a solver.
 - ✓ **Non-solver-friendly:** Structure discovery algorithms do not perform joint optimization based on solver performance.
- **Automatic design and reasoning of solving policies:**
 - ✓ **High cost of designing new solving policies:** Designing new solving policies requires extensive theoretical knowledge, and once designed, the policies cannot be automatically applied to other problems.
 - ✓ **Joint optimization of multiple solving policies:** Considering the mutual influence between different policies in the solver, the optimal performance can only be achieved through joint optimization.
 - ✓ **Poor generalizability:** It is difficult to obtain a common solving policy by training different types of models.

Current Scenario & Achievements

- **Manual structure discovery:** In applications such as supply chain production planning and supply network optimization, manually designing structure mining methods based on the production hierarchy or period can improve the solving efficiency by 50%. However, this requires x person-months of design of a structure discovery algorithm, resulting in high costs.
- **Single-module policy learning:** In the current mixed integer programming (MIP) solver, technologies like imitation learning and reinforcement learning are applied in key performance modules (e.g., branch variable selection, cutting plane generation and selection, and pre-solving). Compared with the default solver rules, these technologies have significantly improved the performance of policy learning. When it comes to industrial problems, the average performance improvement even exceeds 20%. However, multi-module joint learning is yet to be optimized.

Technical Requirements

Design an AI solver that features automatic algorithmic reasoning. Compared with the default policies of the traditional mathematical programming solver (OptVerse AI Solver or SCIP), the solving time on the given dataset (MIP LIB 2017 or Huawei-designed) must be reduced by 30%.

MIP LIB 2017: https://miplib.zib.de/tag_benchmark.html

[1] <https://www.chaspark.net/#/questions/716705670765858816?sub=716707674204524546>

[2] Fawzi, Alhoussein, et al. "Discovering faster matrix multiplication algorithms with reinforcement learning." Nature 610.7930 (2022): 47-53.

[3] Lodi, Andrea, and Andrea Tramontani. "Performance variability in mixed-integer programming." Theory driven by influential applications. INFORMS, 2013. 1-12.

[4] Xijun Li, Qingyu Qu, Fangzhou Zhu, Jia Zeng, Mingxuan Yuan, Jie Wang: Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning. AAAI 2023 workshop on AI to Accelerate Science and Engineering

Many thanks & QA



华为诺亚方舟实验室-决策推理实验室，聚焦于**AI4求解器**、**AI4EDA**（芯片设计自动化软件）等前沿研究方向，支撑芯片设计制造、工业排产、物流调度等重要产业，撬动巨额市场价值。**欢迎志同道合的老师、同学们和我们一同探索！**

You can find more technical details on <https://xijunlee.github.io/>

Summary of our group's work in last two years

- [1] Xijun Li*, Zhihai Wang*, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, Feng Wu: Learning Cut Selection for Mixed-Integer Linear Programming via Hierarchical Sequence Model. ICLR 2023 [[pdf](#)]
- [2] Han Lu, Zenan Li, Runzhong Wang, Qibing Ren, Xijun Li, Mingxuan Yuan, Jia Zeng, Xiaokang Yang, Junchi Yan: ROCO: A General Framework for Evaluating Robustness of Combinatorial Optimization Solvers on Graphs. ICLR 2023 [[pdf](#)]
- [3] Xijun Li, Qingyu Qu, Fangzhou Zhu, Jia Zeng, Mingxuan Yuan, Jie Wang: Accelerating Linear Programming Solving by Exploiting the Performance Variability via Reinforcement Learning. AAAI 2023 workshop on AI to Accelerate Science and Engineering
- [4] Wenxuan Guo, Hui-Ling Zhen, Xijun Li#, Mingxuan Yuan, Yaohui Jin, Junchi Yan: Machine Learning Methods in Solving the Boolean Satisfiability Problem. Machine Intelligence Research Journal
- [5] Jiayi Zhang, Chang Liu, Xijun Li#, Hui-Ling Zhen, Junchi Yan, Mingxuan Yuan: A Survey for Solving Mixed Integer Programming via Machine Learning. Neurocomputing Journal [[pdf](#)]
- [6] Ji Zhang, Xijun Li#, Xiyao Zhou, Mingxuan Yuan, Zhuo Cheng, Keji Huang, Yifan Li: L-QoCo: learning to optimize cache capacity overloading in storage systems. DAC 2022: 379-384 (CORE A, CCF A) [[pdf](#)]
- [7] Xijun Li*#, Yunfan Zhou*, Jinhong Luo, Mingxuan Yuan, Jia Zeng, Jianguo Yao: Learning to Optimize DAG Scheduling in Heterogeneous Environment. MDM 2022: 137-146 (CORE A, CCF C) [[pdf](#)] [[video](#)] [[slides](#)]
- [8] Qingyu Qu, Xijun Li#, Yunfan Zhou; Yordle: An Efficient Imitation Learning for Branch and Bound; NeurIPS 2021 ML4CO Competition [[pdf](#)]

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and
organization for a fully connected,
intelligent world.

**Copyright©2018 Huawei Technologies Co., Ltd.
All Rights Reserved.**

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

