May 2026

Learning for MIP

Learning to Configure Separators and Data Augmentation with Large Language Models

Cathy Wu

Associate Professor | MIT LIDS, CEE, & IDSS

Mixed Integer Programming (MIP) Workshop

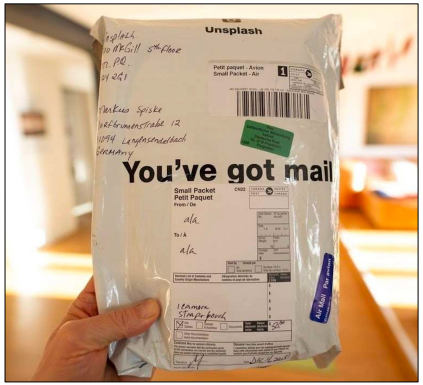
Data Science for Transportation Systems

An **AI for Engineering** Research Group

Focus: Design AI & optimization methods to make it easier to understand and improve transportation systems

Motivating example: Vehicle Routing Problems (VRP)

- One of the most widely studied in Combinatorial Optimization:
  - ≈10,000 works published in 2025 alone (Google Scholar), mostly heuristics
  - Direct application in real systems that distribute goods and provide services
  - Last-mile delivery, waste collection, field service, ride-pooling, on-demand transit



~10,000
papers in 2025

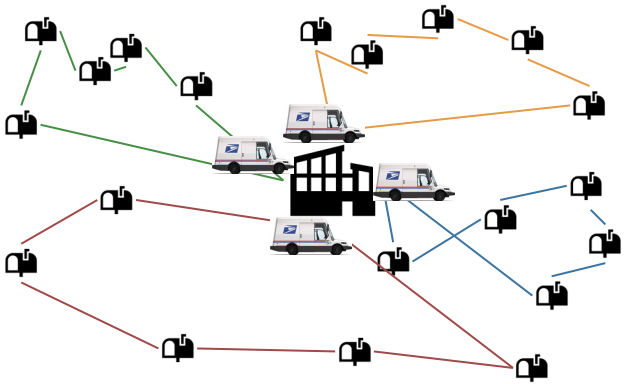
$11T+
global logistics costs, 2023

65M+
US parcel deliveries / day

Dantzig, G. B., & Ramser, J. H. "The Truck Dispatching Problem." Management Science, 1959.
Stats: Statista/Research & Markets (2024); ShipMatrix US parcel report (2024).

Long Tail of Transportation Optimization Problems

- Definition: **Vehicle Routing Problems (VRPs)**
- Given
  - Depot
  - N customers (location, demand)
  - Vehicle fleet (capacity)
  - … **Other constraints** …
- Find the lowest cost set of routes that satisfies all customers



Classical dimensions

- Vehicle capacities
- Time windows
- Heterogeneous fleets
- Multiple depots
- Pickup and delivery, backhauling
- Split delivery
- Arc routing (e.g., garbage collection)

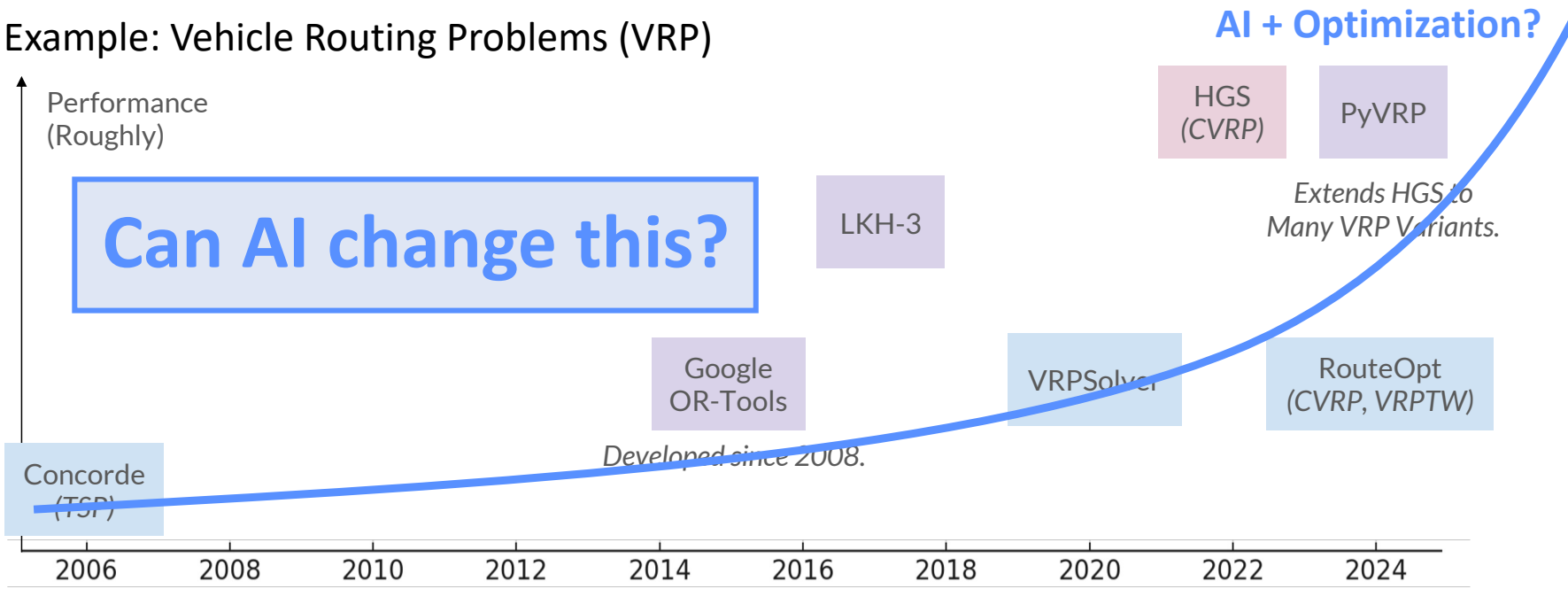
Emerging variants

- Electric vehicle routing (charging decisions)
- Drone routing (3D, endurance limits)
- Warehouse routing (pickers, sortation)
- Multi-modal logistics (truck + drone, rail + truck)

Laporte, G. “Fifty Years of Vehicle Routing.” Transportation Science, 2009.
Dantzig, G. B., & Ramser, J. H. “The Truck Dispatching Problem.” Management Science, 1959.

Solving the Long Tail is Slow and Laborious

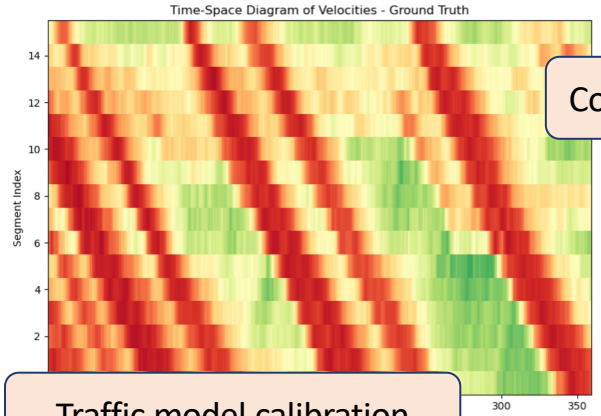
Example: Vehicle Routing Problems (VRP)



*Roughly, the y axis corresponds to the performance of the solver (e.g. HGS > LKH-3 > OR-Tools).

Laporte, G. "Fifty Years of Vehicle Routing." Transportation Science, 2009.
Dantzig, G. B., & Ramser, J. H. "The Truck Dispatching Problem." Management Science, 1959.

The Long Tail of Optimization is Cross-Cutting

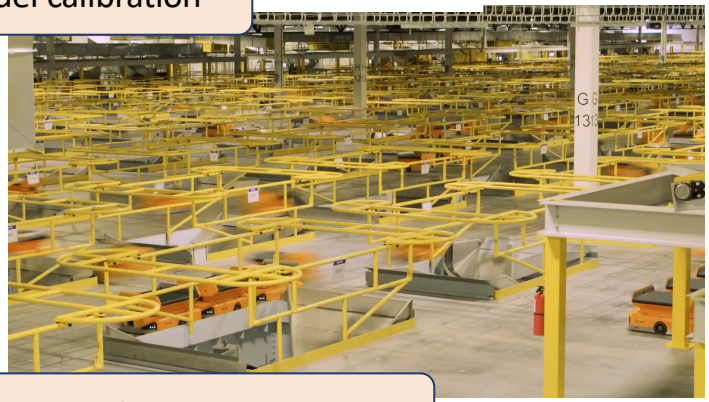


Traffic model calibration

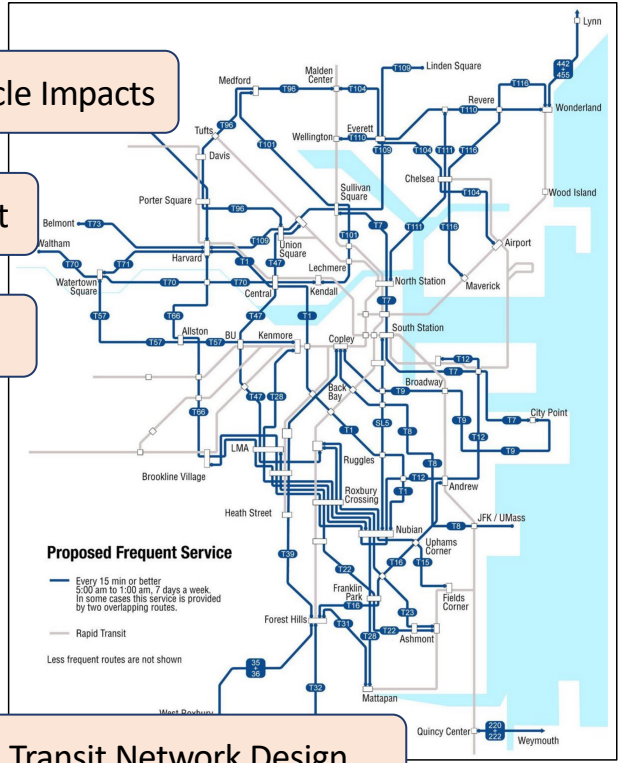
Connected and Automated Vehicle Impacts

Dynamic traffic assignment

Network optimization

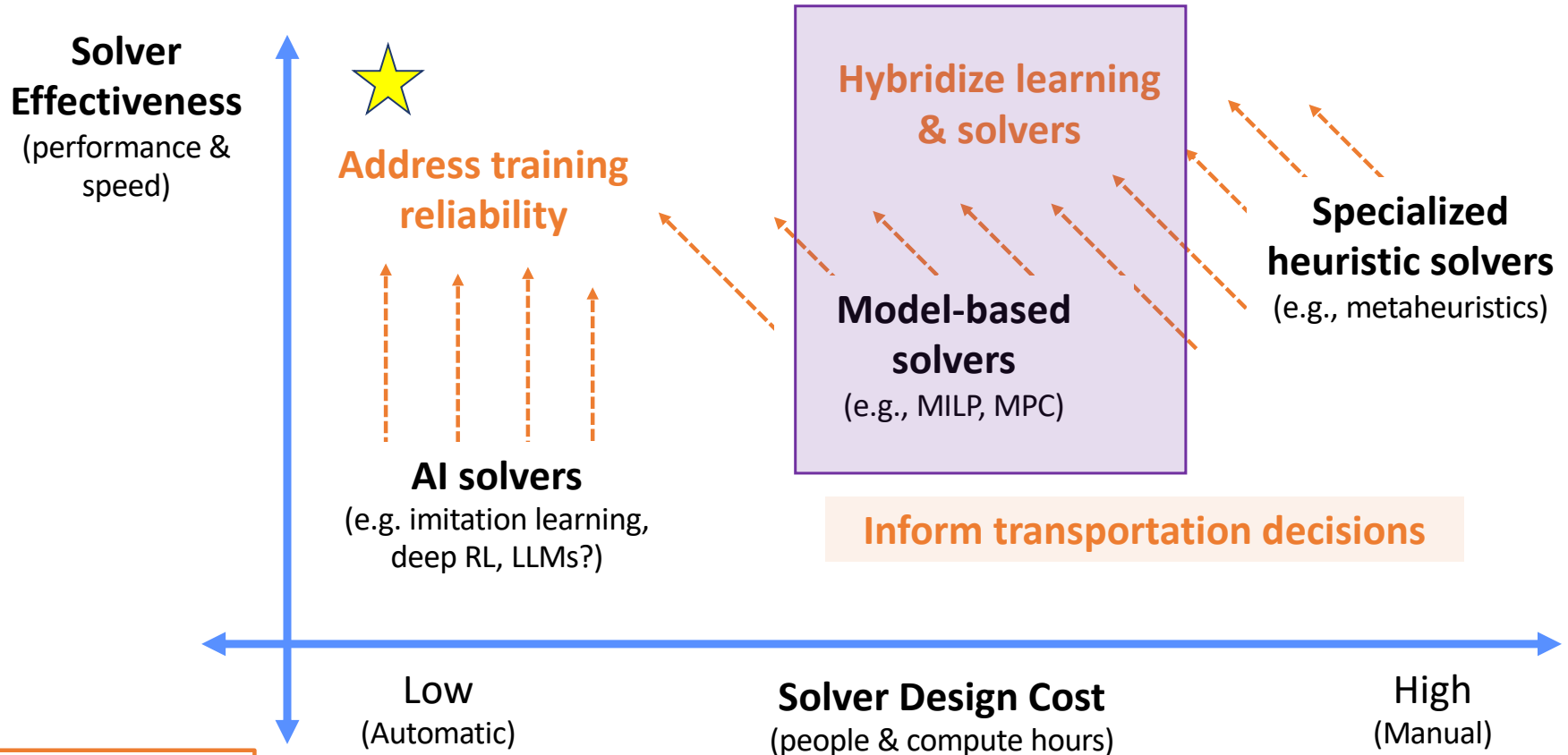


Warehouse operations



Transit Network Design

Optimization Solver Landscape

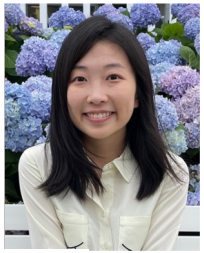


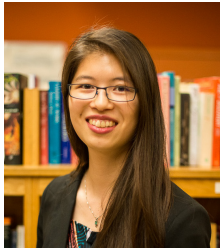
My research lab

Learning to Configure Separators in Branch-and-Cut

NeurIPS 2023

Sirui Li*, Wenbin Ouyang*, Max B. Paulus, **Cathy Wu**



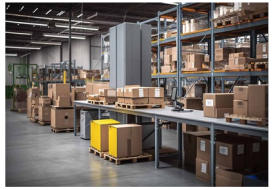
Branch-and-cut for Mixed Integer Linear Programming

Mixed Integer Programming (MILP) Solvers

GUROBI OPTIMIZATION

IBM CPLEX

Facility Location

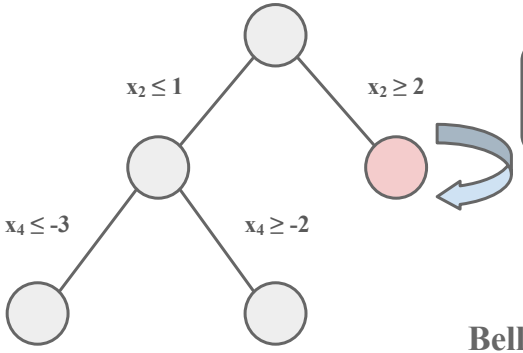


Packing

Last-mile delivery

Branch-and-Cut Algorithm

MILP $x^* = \text{argmin}\ \{c^T x: Ax \leq b;\ x_i \in \mathbb{Z}\ \ \forall i \in I\}$



Core Structure w/ Optimality Guarantees

Bells & Whistles that make the solver practical

Wu

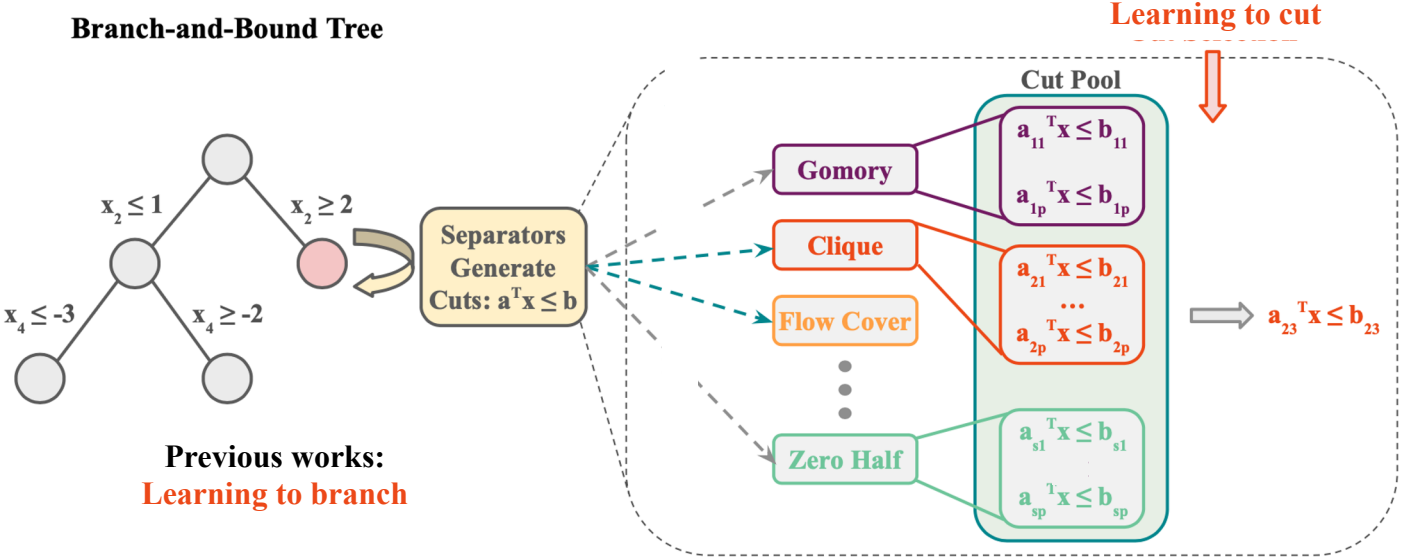
Separator Configuration in Branch-and-Cut

Mixed integer linear programs (MILPs)

$$\begin{aligned} \min_{x} \quad & c^T x && \text{objective} \\ \text{s.t.} \quad & Ax \le b && \text{constraints} \\ & x_i \in \mathbb{Z}, \forall i \in \mathcal{I} && \text{integrality} \end{aligned}$$

Previous works:
Algorithm configuration for MILP (no learning)

Previous works:
Learning to cut



Previous works:
Learning to branch

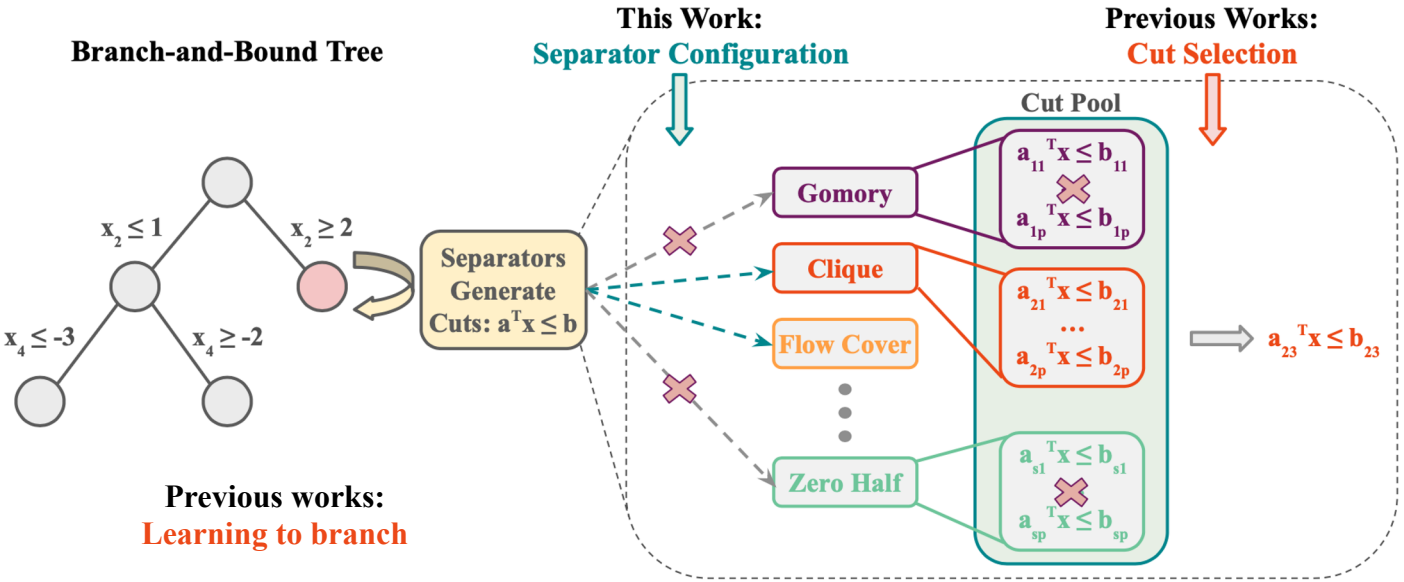
Algorithm configuration for MILP: Xu et al., AAAI 2010 Xu et al., IJCAI, 2011
Learning to branch: He et al., 2014, Gasse et al., 2019, Khalil et al., 2016, Khalil et al., 2017, Kruber et al., 2017. **Learning to cut** : Tang et al., 2020, Paulus et al., 2022

The Separator Configuration Task

Mixed integer linear programs (MILPs)

$$\min_{x} \quad c^T x \qquad \text{objective}$$
$$\text{s.t.} \quad Ax \le b \qquad \text{constraints}$$
$$x_i \in \mathbb{Z}, \forall i \in \mathcal{I} \qquad \text{integrality}$$

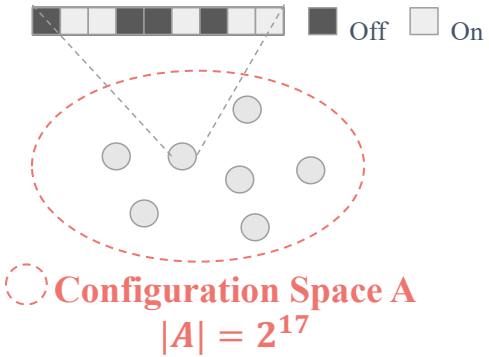
This work introduces a new machine learning task to accelerate solving MILPs.

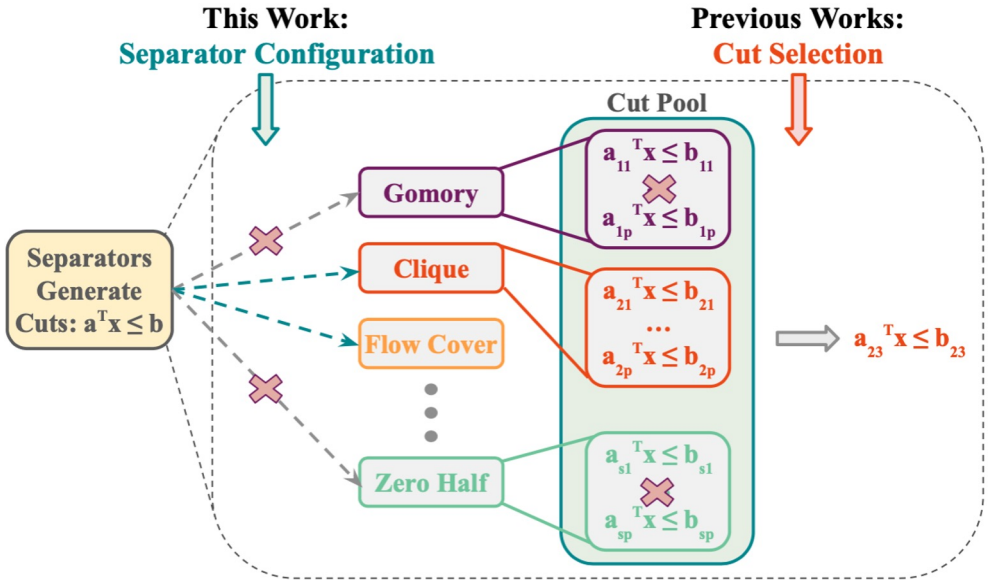


Li*, Ouyang*, Paulus, **Wu**. “Learning to Configure Separators in Branch-and-Cut.” NeurIPS, 2023.

The Separator Configuration Task

- **Problem**: Find a decision rule $\tilde{f}_\theta$ that, given a MILP instance $x = (A, b, c, \mathcal{I})$ from a fixed but unknown distribution, selects a separator configuration $s \in \{0,1\}^M$ to minimize solve time (or optimality gap) for a fixed gap (or time).
  - Let M denote the number of separators (M=17 for SCIP, M=21 for Gurobi)
- **Key challenge**: Configuration space $|A| = 2^M$ is huge! ($\approx 130K$ to $2M$)





Li*, Ouyang*, Paulus, **Wu**. "Learning to Configure Separators in Branch-and-Cut." NeurIPS, 2023.

Learning-to-Separate Method for MILPs

- **Key idea**: Diminishing marginal returns
- **Proposition**: Optimal predictor performance $f_{\overline{A}}$ is **submodular** in $\overline{A} \subseteq A$.
- **Implication**: Can greedily construct $\overline{A} \ll A$; in practice $|\overline{A}| \approx 20$.

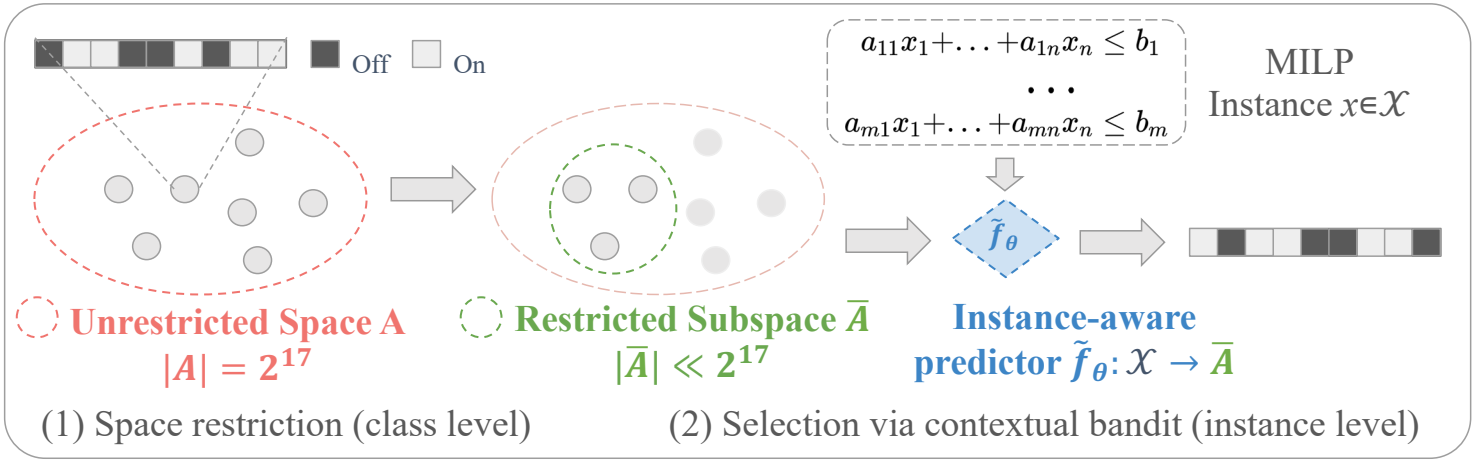
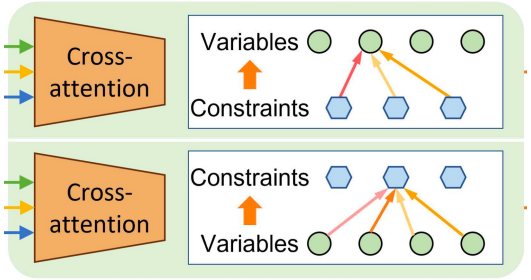


Figure: Learning-to-separate method

Li*, Ouyang*, Paulus, **Wu**. "Learning to Configure Separators in Branch-and-Cut." NeurIPS, 2023.

Context Encoding & Architecture

Representing the MILP problem: $x = (A, b, c, \mathcal{I})$

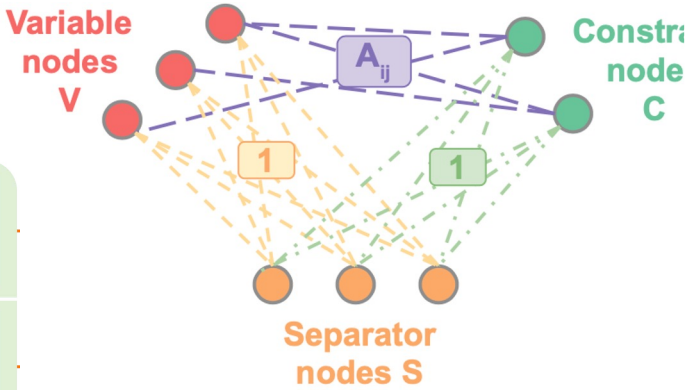


Dual attention for MIP

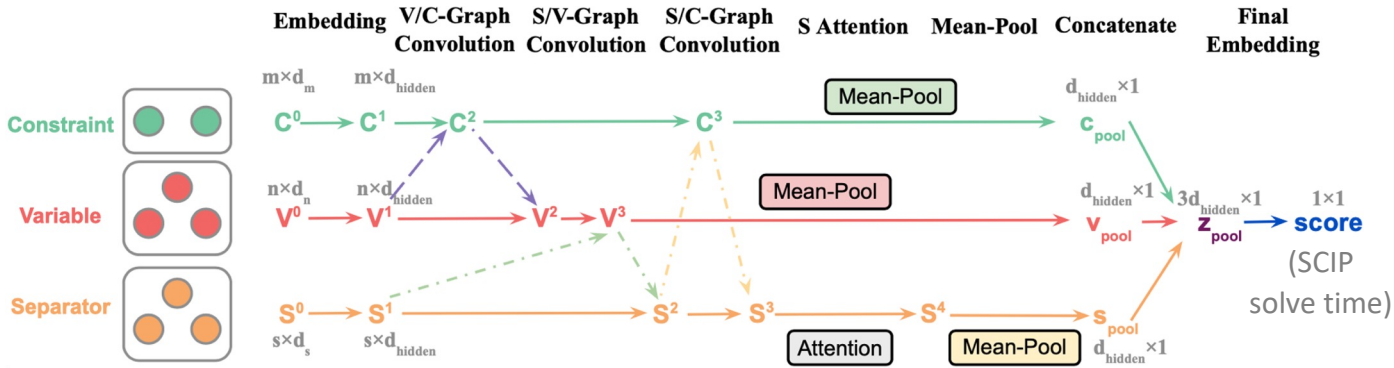
Huang, et al., ICML 2026

$\tilde{f}_\theta(x, s)$: $X \times S \to \mathbb{R}$

where X is the MILP instance class and $S = \{0, 1\}^M$ with M separators

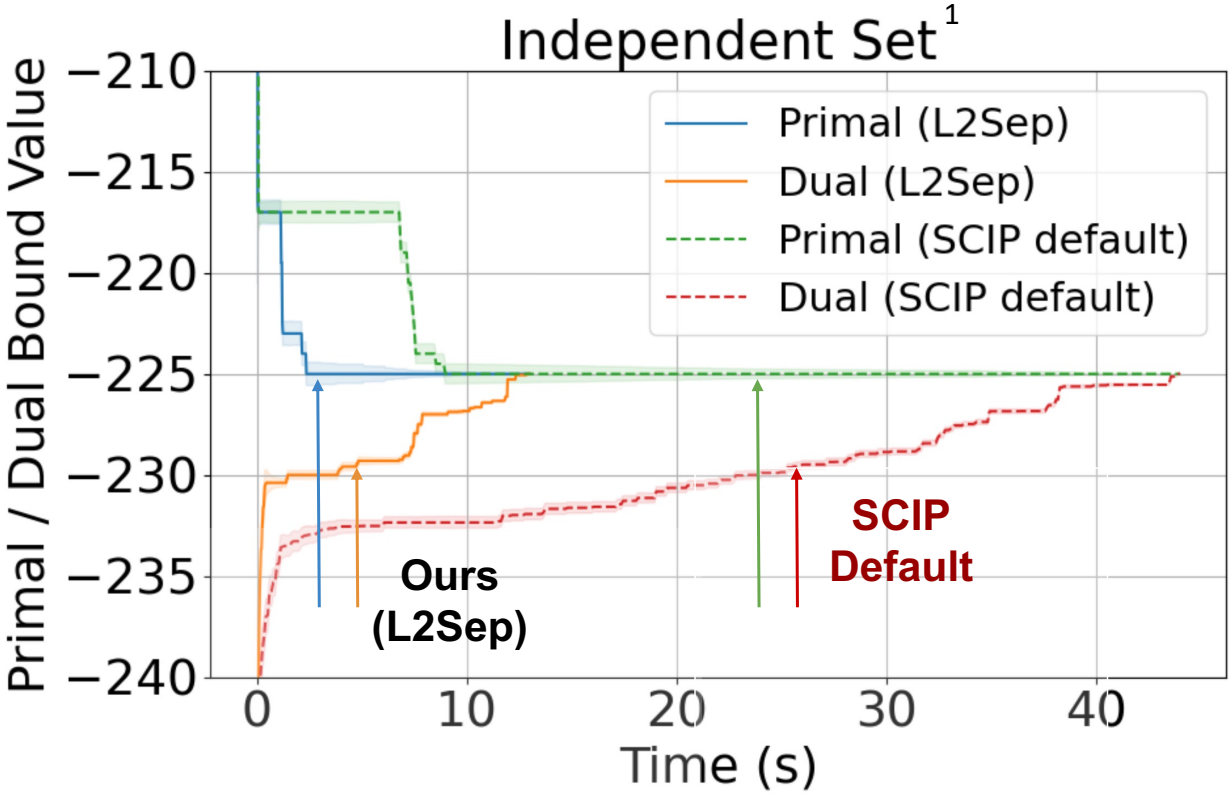


Node Type	Feature	Description
Vars	norm coef	Objective coefficient, normalized by objective norm
	type	Type (binary, integer, impl. integer, continuous) one-hot
	has lb	Lower bound indicator
	has ub	Upper bound indicator
	norm redcost	Reduced cost, normalized by objective norm
	solval	Solution value
	solfrac	Solution value fractionality
	sol_is_at_lb	Solution value equals lower bound
	sol_is_at_ub	Solution value equals upper bound
	norm_age	LP age, normalized by total number of solved LPs
	basestat	Simplex basis status (lower, basic, upper, zero) one-hot
Cons, Added Cuts	is_cut	Indicator to differentiate cut vs. constraint
	type	Separator type, one-hot
	rank	Rank of a row
	norm_nnzrs	Fraction of nonzero entries
	bias	Unshifted side normalized by row norm
	row_is_at_lhs	Row value equals left hand side
	row_is_at_rhs	Row value equals right hand side
	dualsol	Dual LP solution of a row, normalized by row and objective norm
	basestat	Basis status of a row in the LP solution, one-hot
	norm_age	Age of row, normalized by total number of solved LPs
	norm_nlp_creation	LPs since the row has been created, normalized
	norm_intcols	Fraction of integral columns in the row
	is_integral	Activity of the row is always integral in a feasible solution
	is_removable	Row is removable from the LP
	is_in_lp	Row is member of current LP
	violation	Violation score of a row
	rel_violation	Relative violation score of a row
	obj_par	Objective parallelism score of a row
	exp_improv	Expected improvement score of a row
	supp_score	Support score of a row
	int_support	Integral support score of a row
	scip_score	SCIP score of a row for cut selection



[1] Paulus et al. "Learning to cut by looking ahead: Cutting plane selection via imitation learning." ICML 2022

Results: Accelerate primal & dual bounds



Accelerates convergence of both primal and dual bound, by:

- Reducing time to generate the cut pool
- Improving cuts, which accelerate other B&C components (e.g. strong branching, pseudocost branching, dual LP)

[1] **Data distribution for the figure: Independent Set with 500 nodes**, "Ecole: A gym-like library for machine learning in combinatorial optimization solvers." In Learning Meets Combinatorial Algorithms at NeurIPS 2020.

Results: Synthetic MILP benchmarks

Performance: median (std)

Table 1: **Tang et al. and Ecole.** Absolute solve time of SCIP default and relative time improvement (median and standard deviation) of different methods (higher the better, best are bold-faced).

		Tang			Ecole		
	Method	Bin. Pack.	Max. Cut	Pack.	Comb. Auc.	Indep. Set	Fac. Loc.
	Default Time (s)	0.076s (0.131s)	1.77s (0.56s)	8.82s (25.46s)	2.73s (4.43s)	8.21s (114.15s)	61.1s (55.37s)
Heuristic Baselines	Default	0%	0%	0%	0%	0%	0%
	Random	-23.4% (153.8%)	-108.4% (168.2%)	-91% (127.8%)	-48.6% (159.0%)	-5% (161.4%)	-33.3% (157.2%)
	Prune	13.9% (27.0%)	2.7% (26.2%)	6.6% (45.0%)	12.3% (24.2%)	18.0% (24.2%)	24.7% (47.9%)
Ours Heuristic Variants	Inst. Agnostic Configuration	33.7% (36.6%)	69.8% (10.5%)	20.1% (38.0%)	60.1% (27.6%)	57.8% (29.5%)	11.5% (21.8%)
	Random within Restr. Subspace	26.9% (33.6%)	68.0% (11.0%)	18.8% (38.7%)	58.1% (28.7%)	57.4% (75.8%)	17.7% (33.0%)
Ours Learned	L2Sep	**42.3% (34.2%)**	**71.9% (11.3%)**	**28.5% (39.3%)**	**66.2% (26.2%)**	**72.4% (27.8%)**	**29.4% (39.6%)**

Similar gains over base Gurobi v10.0.1 MILP solver of **10-55%**

Subspace restriction **without** learning

with learning

Relative time improvements of **30-70%**

Li*, Ouyang*, Paulus, **Wu**. "Learning to Configure Separators in Branch-and-Cut." NeurIPS, 2023.

Results: Real-World MILP benchmarks

Performance: median (std)

Challenging heterogeneous benchmark (mixed MILP classes)

Table 3: **Real-world MILPs.** Absolute solve time of SCIP default and relative time improvement (median and standard deviation) of different methods (higher the better, best are bold-faced).

Methods	Default Times (s)	Heuristic Baselinses			Ours Heuristic Variants		Ours Learned
		Default	Random	Prune	Inst. Agnostic Configuration	Random within Restr. Subspace	L2Sep
MIPLIB	25.08s (57.05s)	0%	-149.1% (149.7%)	4.8% (107.6%)	5.5% (71.5%)	1.9% (74.9%)	**12.9% (73.1%)**
NN Verification	31.42s (22.44s)	0%	-300.0% (152.3%)	31.5% (36.3%)	31.4% (38.3%)	30.7% (34.1%)	**37.5% (33.9%)**
Load Balancing	31.86s (7.07s)	0%	-300.1% (129.5%)	21.1% (150.8%)	10.4% (8.5%)	10.0% (31.5%)	**21.2% (20.3%)**

Subspace restriction **without** learning

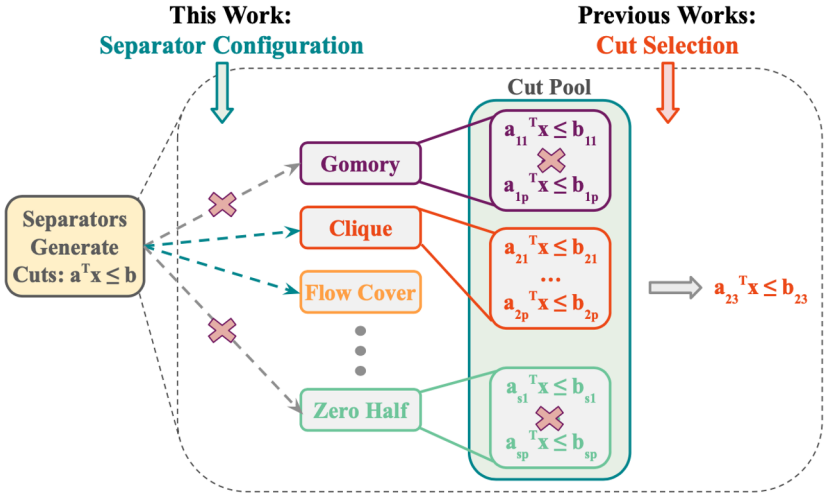
with learning

Relative time improvements of **10-40%**

Li*, Ouyang*, Paulus, **Wu**. "Learning to Configure Separators in Branch-and-Cut." NeurIPS, 2023.

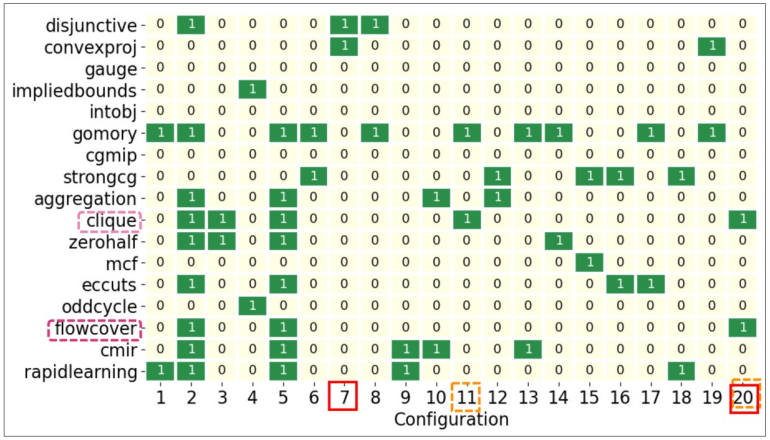
Training & evaluation setup

- Hardware
  - 48 Intel AVX512 CPUs
  - 1 NVIDIA Volta V100 GPU
- Single task training (one MILP class)
- Configuration space restriction: 1.5 days
  - Dataset: 100 instances
- Training: 2 days
  - Dataset: 800 instances
- Evaluation: 100 instances

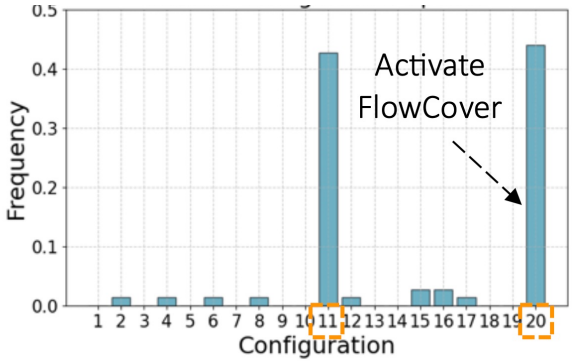


Learning-to-Separate: Interpretation analysis

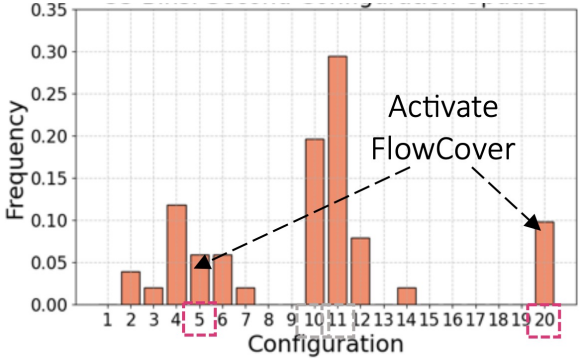
- The learned model recovers known facts from operations research about separators.
- **Example: Bin Packing**
  - Many bins ≈ Bipartite Matching problem
    → Flowcover cuts effective
  - Few bins ≈ Knapsack problem
    → Clique cuts effective



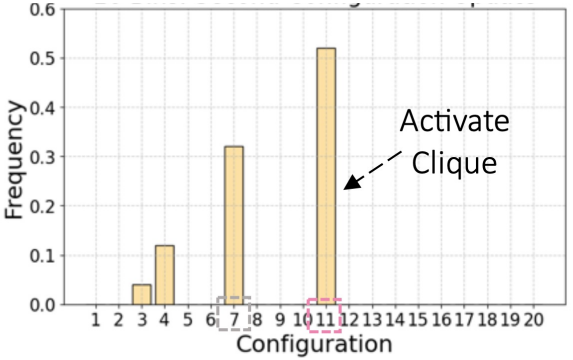
Many bins (66 bins)



Moderate bins (33 bins)



Few bins (16 bins)



Li*, Ouyang*, Paulus, **Wu**. "Learning to Configure Separators in Branch-and-Cut." NeurIPS, 2023.

Research Directions

- **Support cutting plane research**
  - Data-driven evaluation of new cutting plane algorithms
  - Interpretability: What was learned? → Contribute back to MIP literature

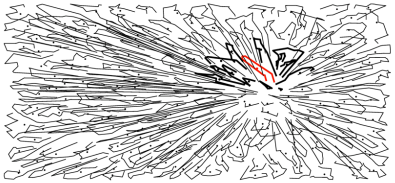
- **Applications**
  - Scaling problems that are bottlenecked on separator selection

- **Algorithmic advances**
  - RL for separator configuration (Ye, et al., NeurIPS 2025)
  - LLMs for separator configuration (Lawless, et al., CPAIOR 2025)
  - Foundation models for MIP / combinatorial optimization

Learning-guided combinatorial optimization

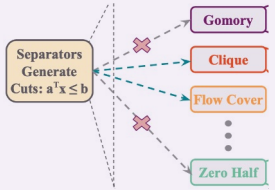
Contribution: **Deep learning can accelerate combinatorial optimization solvers by 2-10x.**

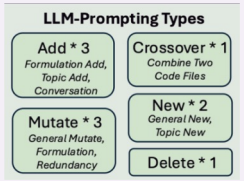
Learning to Focus and Prioritize Search



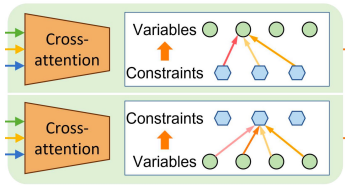
NeurIPS21 **Spotlight (<3%)** ICLR24 JAIR26

Learning for Mixed-Integer Linear Programming



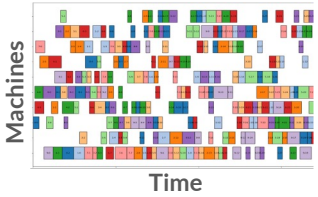


NeurIPS23 ICLR25

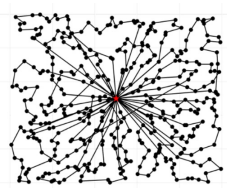


ICML26

Learning to Reuse Across Iterations

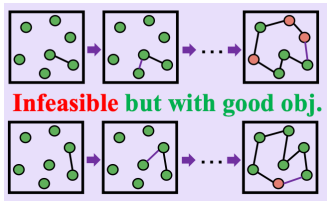


ICLR25



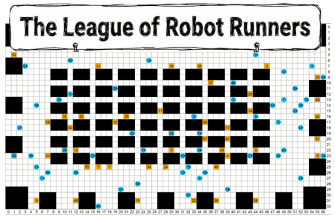
ICLR26 **Oral (1%)**

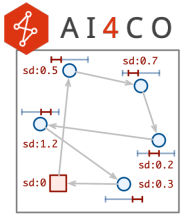
Learning to Repair Solutions



ICLR26

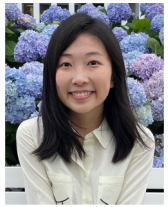
Benchmarks & Competitions

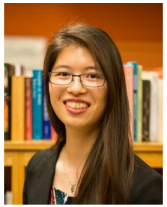




ICAPS24 KDD25 **Oral (9%)**

Modest data and compute (1,000 instances, 1 day on V100); **Fairly generalizable**.




Towards Foundation Models for Mixed Integer Linear Programming

Sirui Li[1], *Janardhan Kulkarni*[2], *Ishai Menache*[2], *Cathy Wu*[1], *Beibin Li*[2]

[1]MIT, [2]Microsoft Research

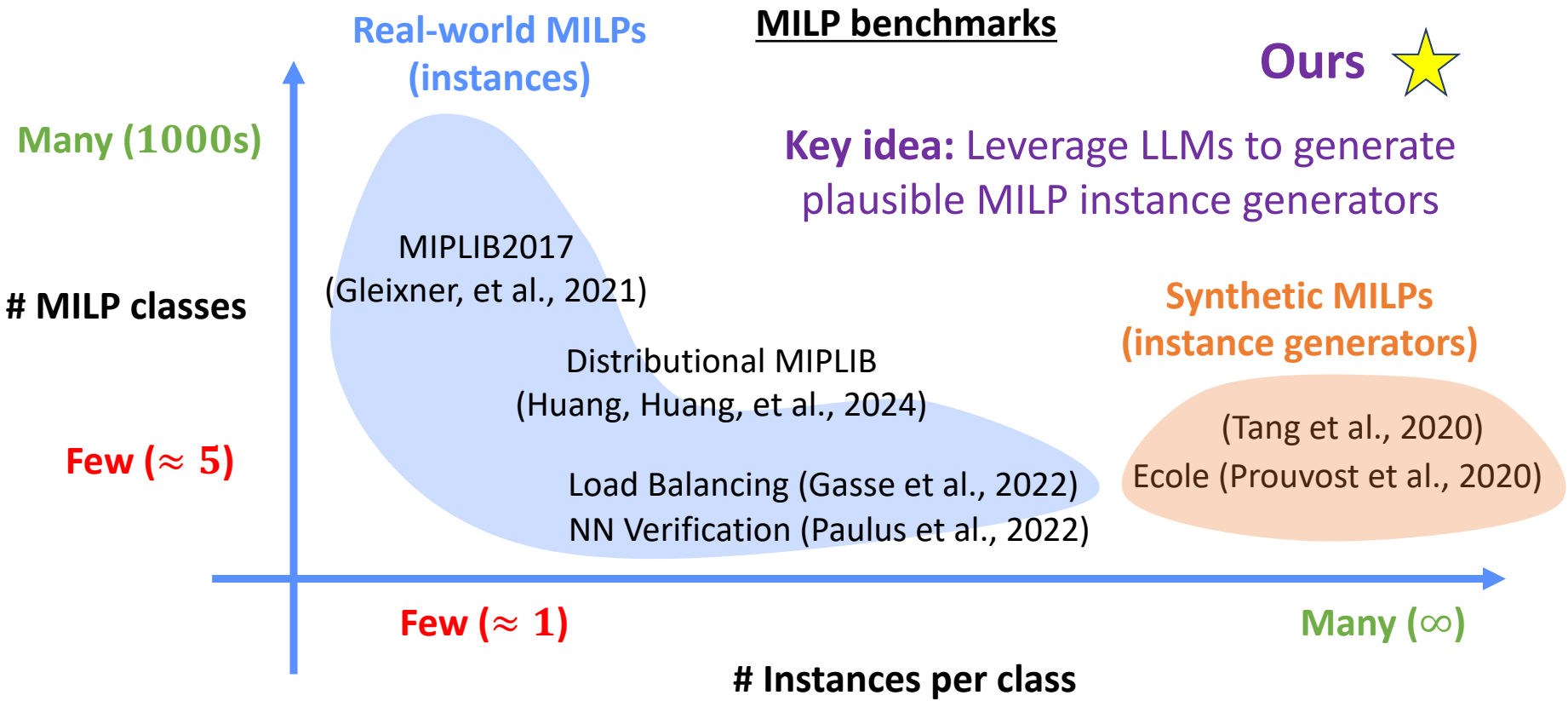
ICLR 2025

Massachusetts Institute of Technology

Microsoft

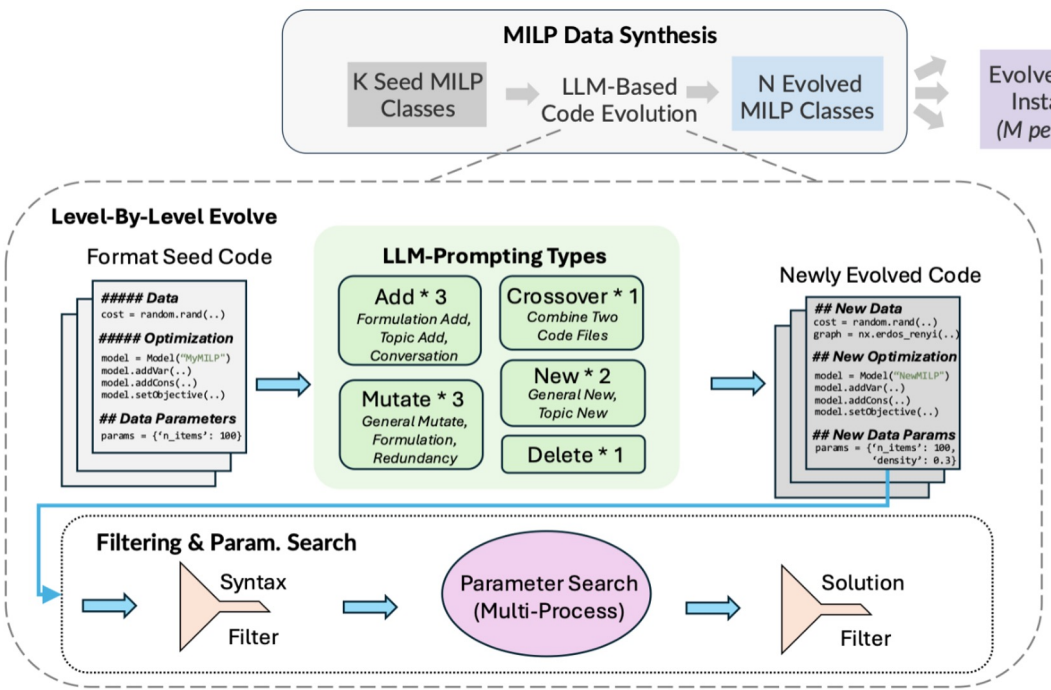
Wu

Towards foundation models for MILP



Li, Kulkarni, Menache, **Wu**, and B. Li, "Towards foundation models for mixed integer linear programming," ICLR, 2025.

MILP-Evolve: 2,000 LLM-generated MILP classes



Our Learning Pipeline (MILP-Evolve):

LLM-Based Optimization Code Synthesis

+

Graph Neural Network Representation Learning for diverse MILP tasks

Hugging Face

Dataset of instance generators is available at:

`https://huggingface.co/datasets/microsoft/MILP-Evolve`

Li, Kulkarni, Menache, **Wu**, and B. Li, "Towards foundation models for mixed integer linear programming," ICLR, 2025.

Topics & domains

```
{
    "Linear Programming": {
        "subtopics": [
            "Basic concepts of linear programming",
            "Linear inequalities and equations",
            "Graphical solution methods",
            "Simplex method"
        ]
    },
    "Integer Programming": {
        "subtopics": [
            "Binary variables and logical constraints",
            "Total unimodularity",
            "Cutting plane methods"
        ]
    },
    "Mixed Integer Linear Programming": {
        "subtopics": [
            "Formulation of MILP problems",
            "Linear vs. integer variables",
            "Applications in logistics, scheduling, and planning"
        ]
    },
    "Solution Techniques": {
        "subtopics": [
            "Branch and Bound",
            "Branch and Cut",
            "Heuristic algorithms",
            "Decomposition methods"
        ]
    },
    "Software and Tools": {
        "subtopics": [
            "Commercial solvers (e.g., CPLEX, Gurobi)",
            "Open-source solvers (e.g., CBC, SCIP)",
            "Modeling languages (e.g., AMPL, GAMS)"
        ]
    },
```

```
{
    "Logistics and Supply Chain Management": {
        "description": "Optimizing supply chain networks, routing vehicles,
        "famous_companies": [
            "DHL",
            "FedEx",
            "UPS",
            "Maersk"
        ],
        "sub_domains": [
            "Last-Mile Delivery",
            "Freight Logistics",
            "Warehouse Management"
        ],
        "key_technologies": [
            "IoT",
            "Blockchain",
            "Machine Learning"
        ],
        "regulatory_bodies": [
            "International Maritime Organization",
            "Federal Aviation Administration"
        ],
        "challenges": [
            "Carbon Footprint",
            "Labor Costs",
            "Globalization"
        ]
    },
    "Healthcare": {
        "description": "Hospital resource allocation, patient scheduling, h
        "famous_companies": [
            "Johnson & Johnson",
            "Pfizer",
            "GSK",
            "Novartis"
        ],
        "sub_domains": [
            "Telemedicine",
            "Pharmaceutical Supply Chain",
```

Seed v.s. Generated MILP Classes

✨8 Seed Classes (Canonical Classes): Independent Set, Set Cover, Capacitated Facility Location, Combinatorial Auction, Multiple Knapsack, Generalized Indep. Set, Max Satisfiability, Multicommodity Network Flow

```
model = Model("IndependentSet")
var_names = {}
for node in graph.nodes:
    var_names[node] = model.addVar(vtype="B", name=f"x_{node}")
for count, group in enumerate(inequalities):
    model.addCons(quicksum(var_names[node] for node in group) <= 1,
name=f"clique_{count}")
objective_expr = quicksum(var_names[node] for node in graph.nodes)
model.setObjective(objective_expr, "maximize")
```

🧿Example Evolved Classes: Conference Room Scheduling, Maze Explore Optimization, Health Resource Allocation, Courier Route Optimization, etc.

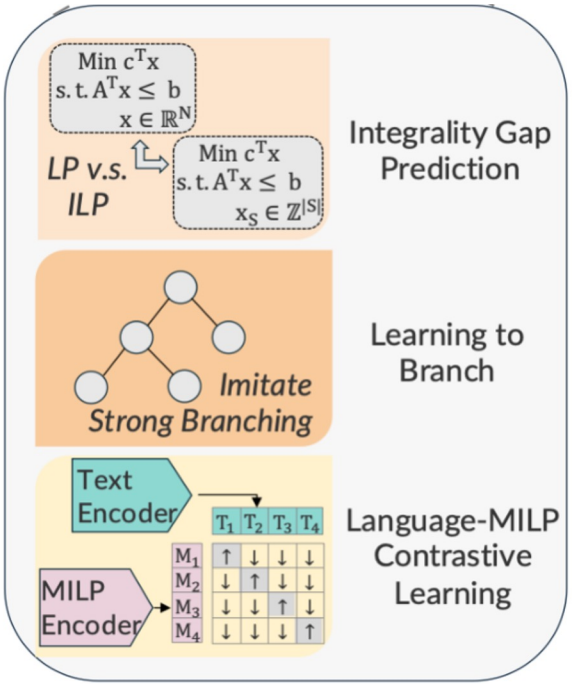
```
model = Model("MazeExplorationOptimization")
n_robots = len(exploration_costs)
n_cells = len(critical_items)

robot_vars = {r: model.addVar(vtype="B", name=f"Robot_{r}") for r in range(n_robots)}
cell_exploration_vars = {(r, c): model.addVar(vtype="C", name=f"Cell_{r}_Cell_{c}") for r in range(n_robots) for c in range(n_cells)}
unmet_detection_vars = {c: model.addVar(vtype="C", name=f"Unmet_Cell_{c}") for c in range(n_cells)}

coordinate_vars = {r: model.addVar(vtype="B", name=f"Coordinate_{r}") for r in range(n_robots)}

model.setObjective(
    quicksum(exploration_costs[r] * robot_vars[r] for r in range(n_robots)) +
    quicksum(detection_costs[r][c] * cell_exploration_vars[r, c] for r in range(n_robots) for c in range(n_cells)) +
    quicksum(1000 * unmet_detection_vars[c] for c in range(n_cells) if priority_cells[c] == 1),
    "minimize"
)
for c in range(n_cells):
    # Critical item detection satisfaction
    model.addCons(quicksum(cell_exploration_vars[r, c] for r in range(n_robots)) + unmet_detection_vars[c] == critical_items[c], f"Nei
    # Priority cell detection
    if priority_cells[c] == 1:
        model.addCons(quicksum(cell_exploration_vars[r, c] for r in range(n_robots)) + unmet_detection_vars[c] >= critical_items[c], f


for r in range(n_robots):
    # Battery limits for each robot
    model.addCons(quicksum(cell_exploration_vars[r, c] for c in range(n_cells)) <= battery_capacities[r] * robot_vars[r], f"Maze_Batte
    # Cell exploration only if robot is active
    for c in range(n_cells):
        model.addCons(cell_exploration_vars[r, c] <= critical_items[c] * robot_vars[r], f"Active_Robot_Constraint_{r}_{c}")
    # Energy consumption constraints
    model.addCons(quicksum(cell_exploration_vars[r, c] for c in range(n_cells)) <= energy_limits[r], f"Energy_Limit_{r}")
    # Sensor range for each robot
```

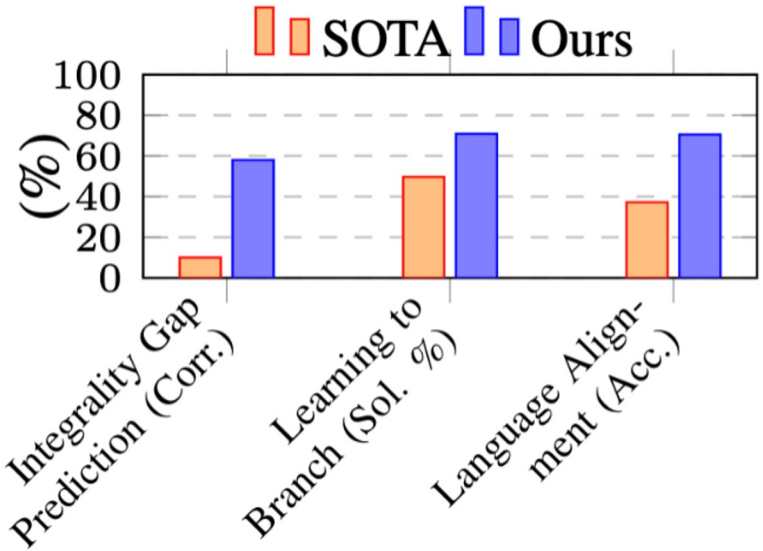
☐ Generate MILP Instances by Calling the MILP Generator (Code) with Different Seeds

Li, Kulkarni, Menache, **Wu**, and B. Li, "Towards foundation models for mixed integer linear programming," ICLR, 2025.

Results

Learning Tasks





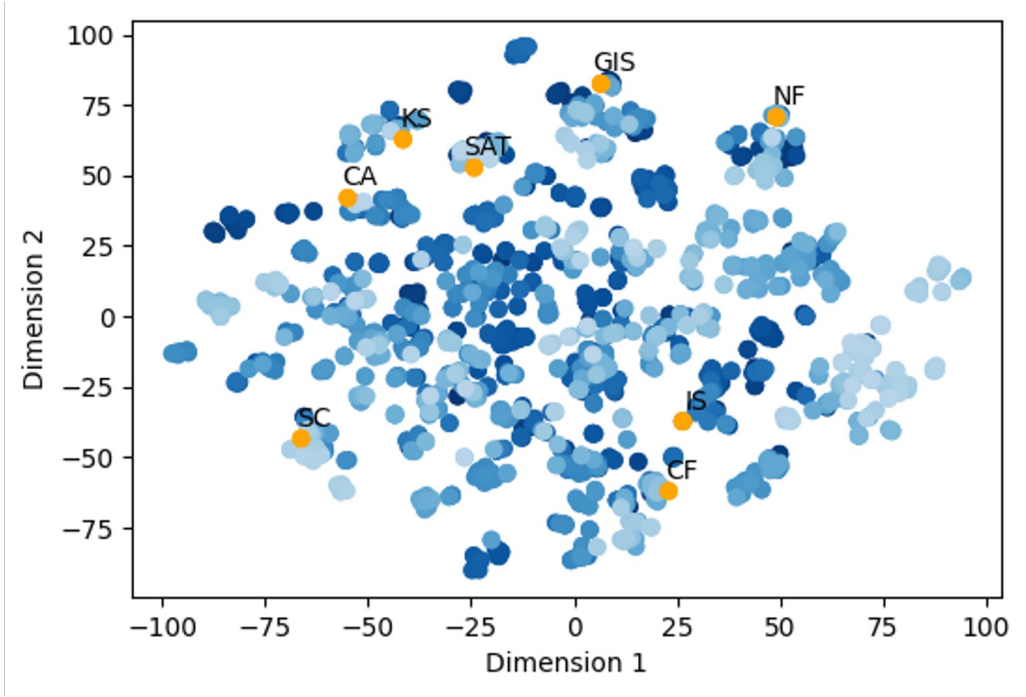
Result: *Significantly improved multi-class learning performance*

Li, Kulkarni, Menache, **Wu**, and B. Li, "Towards foundation models for mixed integer linear programming," ICLR, 2025.

Results

Class diversity

(TSNE visualization of GPT code embedding)



- 8 Seed Classes
- 1500 Evolved Classes

Li, Kulkarni, Menache, **Wu**, and B. Li, “Towards foundation models for mixed integer linear programming,” ICLR, 2025.

References

MathWorks

Microsoft

amazon science

1. Li*, Ouyang*, Paulus, **Wu**. “Learning to Configure Separators in Branch-and-Cut.” NeurIPS, 2023.

2. Li, Kulkarni, Menache, **Wu**, and B. Li, “Towards foundation models for mixed integer linear programming,” ICLR, 2025.