Model solving

Ellen Vitercik

ML to speed up MILP solvers

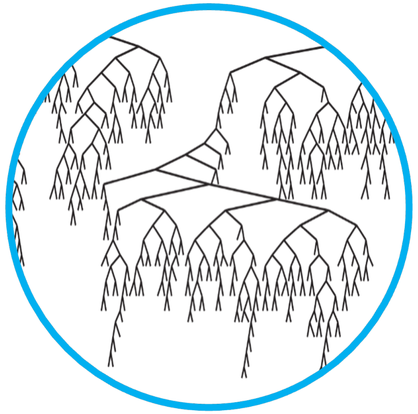
- Modern MILP solvers expose **hundreds** of parameters
- Defaults are rarely optimal: instance families vary widely in structure
- Small changes can yield **large runtime** differences
- **Many** ways to integrate ML into solvers
  - Potential for ML to provide **significant** speedups over defaults

Outline

1. Background
   i. **Branch-and-cut (B&C) algorithm**
   ii. Brief overview of ML for B&C: non-GenAI
2. LLMs for speeding up solvers
3. Takeaways

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search



- Explore **search tree** of restricted MILP subproblems
  - Each node adds bounds on integer variables
- Solve **LP relaxation** to upper bound subproblem's integer-feasible solution
  - If integer-feasible: incumbent solution
- **Branch** by choosing a variable and splitting its domain ← **Variable selection policy**
- **Prune** nodes whose bound can't beat incumbent
- Terminate when all nodes **pruned** or **proven optimal**

Background: Branch-and-cut algorithm

Uses guidance from LP relaxations to guide search

MILP

maximize $\boldsymbol{c}^T\boldsymbol{x}$

subject to $A\boldsymbol{x} \leq \boldsymbol{b}$

$x_i \in \mathbb{Z}$ for all $i \in I$

Linear programming (LP) relaxation

maximize $\boldsymbol{c}^T\boldsymbol{x}$

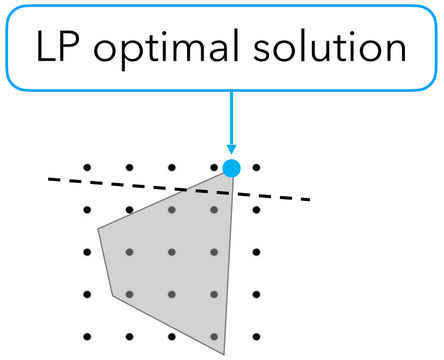
subject to $A\boldsymbol{x} \leq \boldsymbol{b}$

~~$x_i \in \mathbb{Z}$ for all $i \in I$~~

Cutting planes (CPs) are additional constraints that:

- Separate LP optimal solution
- Don't separate any integer point

Many different families of CPs; which to use when?



Outline

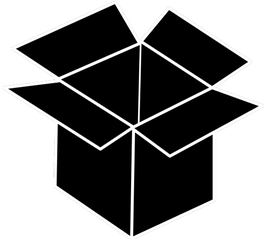
1. Background
   i. Branch-and-cut (B&C) algorithm
   ii. Brief overview of ML for B&C: non-GenAI
2. LLMs for speeding up solvers
3. Takeaways

General setup

ML for MILP solvers

- Define a parameterized solver $A(\theta)$. E.g.:
  - θ are parameters exposed by Gurobi
  - θ are parameters of a neural network embedded in solver
- Specify distribution D over MILPs z (models **application domain**)
- Choose a **performance metric** $c(z, \theta)$; e.g., runtime
- **Ultimate goal:** minimize $\mathbb{E}_{z \sim D}\left[c(z, \theta)\right]$ (proxy of *future* cost on unseen MILPs)
  - Can learn *offline* θ or *instance-aware* $\theta(z)$ configuration

Blackbox algorithm configuration



- Early work: treat solver (largely) as a black-box; learn from evaluations
- Small subset of examples:
  - **ParamILS**: iterated local search over parameter settings
    Hutter, Hoos, Leyton-Brown, Stützle, JAIR'09
  - **SMAC**: model-based search with surrogate predictions
    Hutter, Hoos, Leyton-Brown, LION'11
  - **Portfolio-based algorithm selection**
    Lobjois, Lemaître, AAAI'98; Gomes, Selman, AI'01; Xu, Hoos, Leyton-Brown, AAAI'10; Kadioglu et al., ECAI'10, Sandholm, Handbook of Market Design'13

Blackbox algorithm configuration

Example: Portfolio-based algorithm configuration

- One configuration rarely dominates across diverse MILP instances
- **Portfolios** combine multiple strong configurations
- **Hydra** iteratively grows portfolio via targeted reconfiguration
  - Tune new member against instances current portfolio **solves poorly**
- Instance features enable **per-instance** selection (static & quick probing runs)
  - Choose configuration before solving each instance

Xu, Hoos, Leyton-Brown, AAAI'10

Configuration of solver components

Next gen: don’t treat solver as blackbox; adapt to solver components. E.g.:

Cut selection	Variable selection	Node selection
Tang et al., ICML’20	Khalil et al., AAAI’16	He et al., NeurIPS’14
Balcan et al., NeurIPS’21	Alvarez et al., INFORMS JoC’17	Labassi et al., NeurIPS’22
Balcan et al., NeurIPS’22	Balcan et al., ICML’18	Zhang et al., ICLR’25
Paulus et al., ICML’22	Gasse et al., NeurIPS’19	…
Wang et al., ICLR’23	Gupta et al., NeurIPS’20	
Li et al., NeurIPS’23	Zarpellon et al., AAAI’21	
Deza, Khalil, IJCAI’23	Scavuzzo et al., NeurIPS’22	
Ling et al., AAAI’24	…	
Cheng, Basu, NeurIPS’24		
Cheng et al., NeurIPS’24		
…		

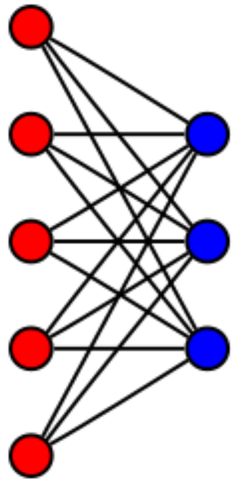
Configuration of solver components

Next gen: don’t treat solver as blackbox; adapt to solver components. E.g.:

Cut selection	Variable selection	Node selection
Tang et al., ICML’20	Khalil et al., AAAI’16	He et al., NeurIPS’14
Balcan et al., NeurIPS’21	Alvarez et al., INFORMS JoC’17	Labassi et al., NeurIPS’22
Balcan et al., NeurIPS’22	Balcan et al., ICML’18	Zhang et al., ICLR’25
Paulus et al., ICML’22	**Gasse et al., NeurIPS’19**	…
Wang et al., ICLR’23	Gupta et al., NeurIPS’20	
Li et al., NeurIPS’23	Zarpellon et al., AAAI’21	
Deza, Khalil, IJCAI’23	Scavuzzo et al., NeurIPS’22	
Ling et al., AAAI’24	…	
Cheng, Basu, NeurIPS’24		
Cheng et al., NeurIPS’24		
…		

Graph neural networks for variable selection

- **Key idea:** encode B&B state as variable-constraint **bipartite graph**
  - Use bipartite **graph neural network** as a variable selection policy
- Training: behavioral cloning of strong branching (expensive **gold standard**)
- Integrated in SCIP; four NP-hard benchmarks
- Results:
  - Best imitation accuracy among ML baselines
  - Generally faster than SCIP default; good size generalization



Gasse, Chételat, Ferroni, Charlin, Lodi, NeurIPS'19

Benchmarks

- **MIPLIB 2017:** widely used “real-world” MILP benchmark library
  - Benchmark set: 240 instances, solvable, numerically stable
  - Collection set: 1065 diverse instances, less filtered
- **Distributional MIPLIB:** library of MILP **distributions** [Huang et al., arXiv’24]
  - More than 35 distributions across 13 domains
  - Includes synthetic and real-world domains, multiple hardness levels
- **MILP-Evolve** [Li et al. ICLR’25]
  - “Evolving” pipeline for generating new MILP families
  - Designed to be highly diverse to mimic real-world optimization scenarios

Outline

1. Background
2. LLMs for speeding up solvers
   i. **Solver configuration**
   ii. Evolutionary search for MILP heuristics
3. Takeaways

Lawless, Li, Wikum, Udell, **V**, CPAIOR'25

Key challenge

Conventional data-driven approaches require a lot of compute

Conventional data-driven configuration pipeline:

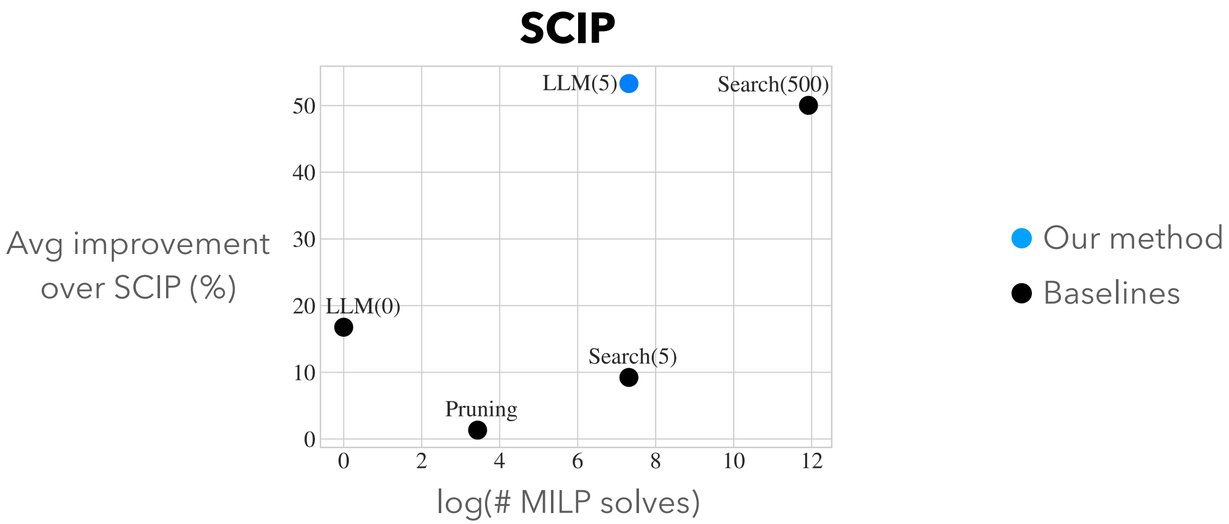
1. Gather a **training set** of MILPs
2. Find **configuration(s) with good average runtime** over training set
3. Hope for a good runtime from the same application (or distribution)

Key challenge: Evaluating one configuration's average runtime requires **solving** every MILP in the training set using that configuration

Key question: Can we generate problem-specific cutting plane configurations with **little to no historical data and compute**?

Our contributions

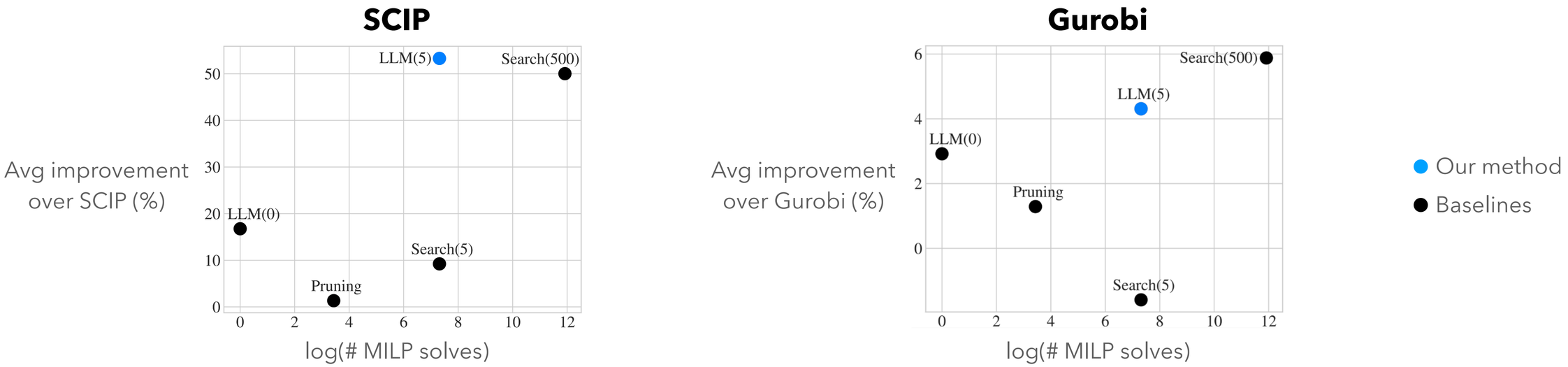
- **First LLM-based framework** to configure MILP solvers
- **Consistent improvement** over solver default (SCIP and Gurobi)
  - Pareto-optimal compared to baseline methods



Lawless, Li, Wikum, Udell, **V**, CPAIOR'25

Our contributions

- **First LLM-based framework** to configure MILP solvers
- **Consistent improvement** over solver default (SCIP and Gurobi)
  - Pareto-optimal compared to baseline methods



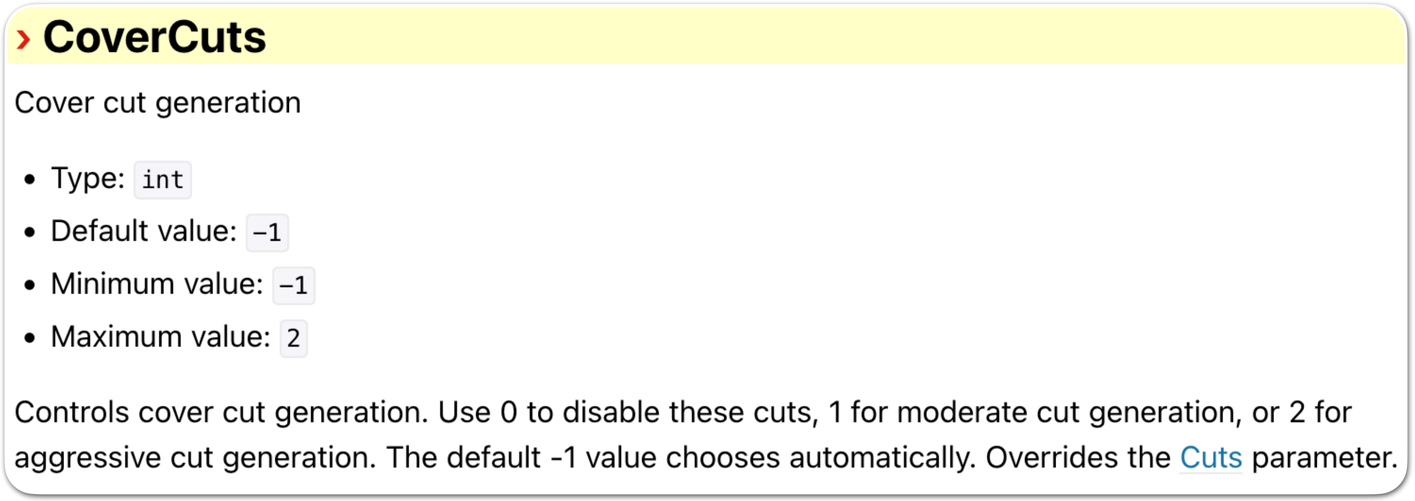
Lawless, Li, Wikum, Udell, **V**, CPAIOR'25

Why LLMs

- LLMs are powerful, but they can’t do everything
- They are good at **information retrieval**
- There’s a **rich literature** on cutting planes

Challenges to using LLMs

- LLM output can be highly **unstable**
- Cutting plane separators are **solver-specific**
  - Details of solver separators are not always available



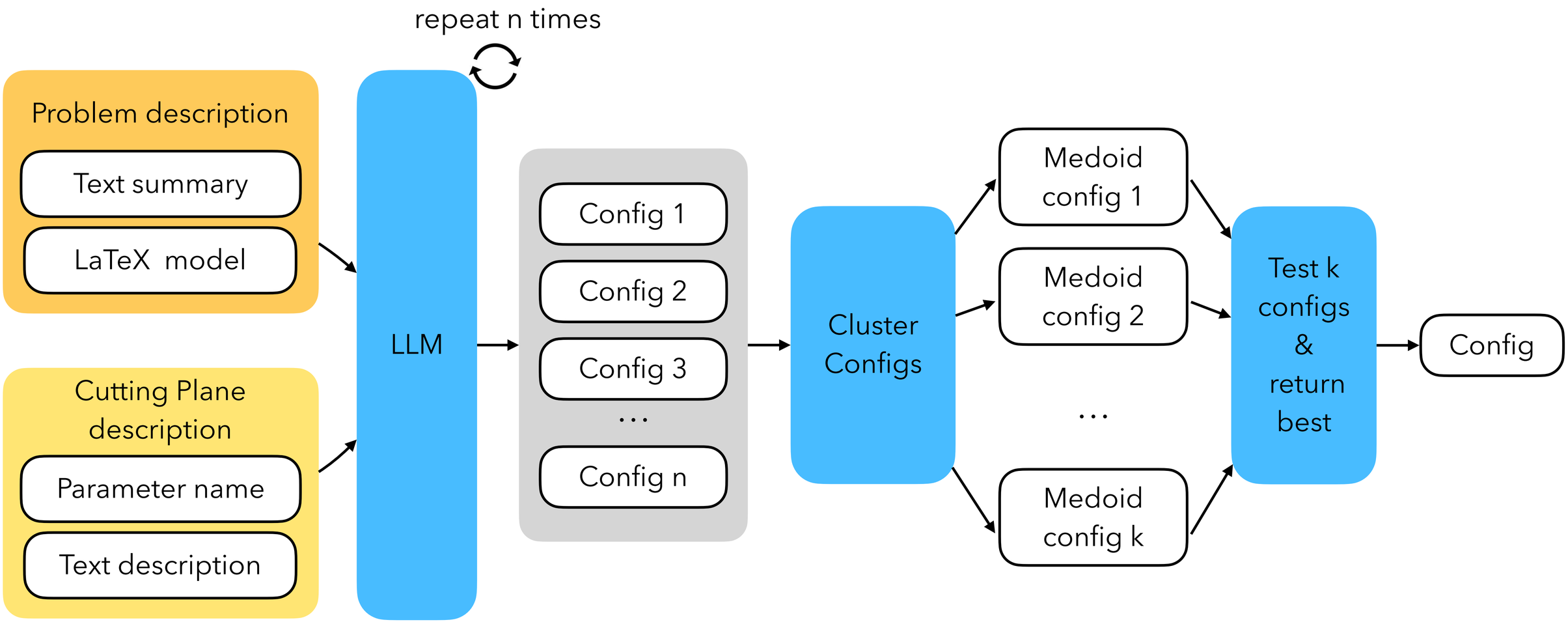
› CoverCuts

Cover cut generation

- Type: `int`
- Default value: `-1`
- Minimum value: `-1`
- Maximum value: `2`

Controls cover cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

LLM for cold-start configuration pipeline



Experimental set-up

Baselines and our method

- **Pruning**: turns off all CPs not used while solving validation set
  - Use the default setting for other CPs
- **Search**(d): sample d candidate configurations uniformly at random
  - Use the one with best median performance on validation set
- **Zero-shot**: use medoid of the largest cluster
- **Cold-start**(k):
  1. Run k medoids clustering
  2. Select the best performing medoid on the validation set

Experimental set-up

Datasets, model

	Dataset	# vars	# constrs
Classic MILP families	Binary packing	300	300
	Capacitated facility location	100	100
	Combinatorial auction	100	500
	Maximum independent set	500	1088
	Max cut	54	134
	Packing	60	60
	Set cover	500	250
Complex real-world MILP families	Load balancing	64340	61000
	Middle-mile consolidation network design (MM)	569	248

Model: GPT-4o

Training set size: 100
Val set size: 30

Evaluation metric:
% improvement over default solve time

Empirical results

Cold-start(5) yields 6–71% faster runtimes than SCIP's default

Problem	Pruning	Search(5)	Search(500)	Zero-shot	Cold-start(5)
Bin. pack.	1.33	9.23	39.3	16.76	38.35
Cap. fac.	-0.64	9.57	2.72	7.61	26.12
Comb. auc.	1.96	58.1	64.01	21.06	63.59
Ind. set	2.07	26.95	67.01	21.6	71.95
Max. cut	-2.18	17.72	69.63	71.43	71.01
Pack.	15.87	-13.81	24.49	15.09	25.51
Set cov.	6.62	-10.04	61.08	61.72	61.74
Load bal.	0.08	-150.01	-50.02	0.0	6.37
MM	-0.12	-8.83	50.03	-6.52	53.3

Empirical results

Cold-start(5) yields 6–71% faster runtimes than SCIP's default

Problem	Pruning	Search(5)	Search(500)	Zero-shot	Cold-start(5)
Bin. pack.	1.33	9.23	39.3	16.76	38.35
Cap. fac.	-0.64	9.57	2.72	7.61	26.12
Comb. auc.	1.96	58.1	64.01	21.06	63.59
Ind. set	2.07	26.95	67.01	21.6	71.95
Max. cut	-2.18	17.72	69.63	71.43	71.01
Pack.	15.87	-13.81	24.49	15.09	25.51
Set cov.	6.62	-10.04	61.08	61.72	61.74
Load bal.	0.08	-150.01	-50.02	0.0	6.37
MM	-0.12	-8.83	50.03	-6.52	53.3

By testing only 5 configs, we match/beat Search(500) on **all instances**

Empirical results

Cold-start(5) yields 6–71% faster runtimes than SCIP's default

Problem	Pruning	Search(5)	Search(500)	Zero-shot	Cold-start(5)
Bin. pack.	1.33	9.23	39.3	16.76	38.35
Cap. fac.	-0.64	9.57	2.72	7.61	26.12
Comb. auc.	1.96	58.1	64.01	21.06	63.59
Ind. set	2.07	26.95	67.01	21.6	71.95
Max. cut	-2.18	17.72	69.63	71.43	71.01
Pack.	15.87	-13.81	24.49	15.09	25.51
Set cov.	6.62	-10.04	61.08	61.72	61.74
Load bal.	0.08	-150.01	-50.02	0.0	6.37
MM	-0.12	-8.83	50.03	-6.52	53.3

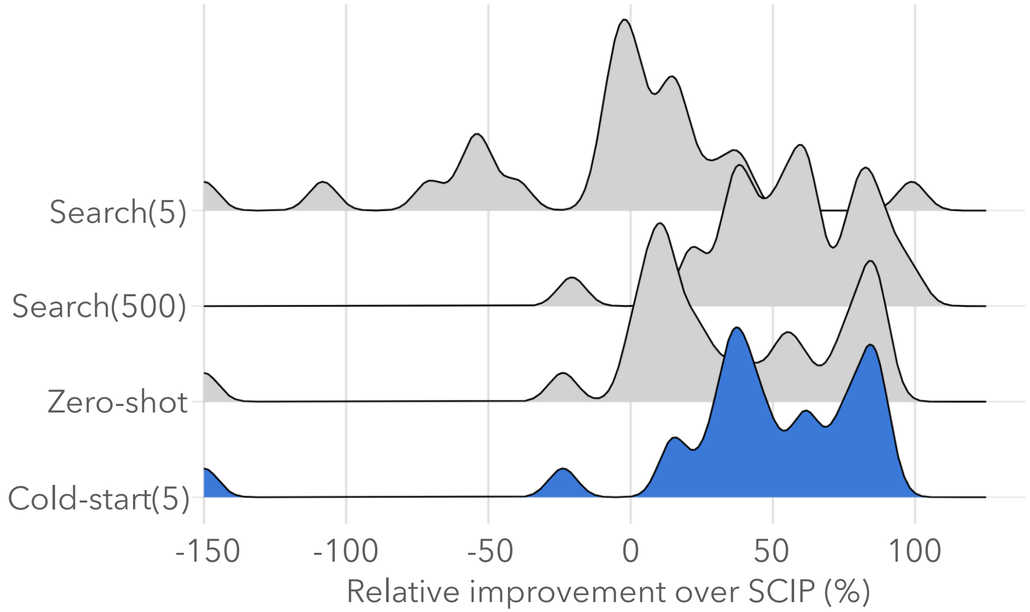
By testing only 5 configs, we match/beat Search(500) on **all instances**

Without solving **any** MILPs, we match/beat Search(500) on **4/9 instances**

On out-of-distribution instances

25 families of problem from MILP-Evolve dataset [Li et al. ICLR'25]

- New dataset, "evolving" pipeline for generating new MILP families
- Designed to be highly diverse to mimic real-world optimization scenarios



Ablations

Our design choices are robust

Setting	Ind. set	Max cut	Bin. pack.	MM
Ours	71.95	71.01	38.35	53.3
Disable cutting planes	-14.96	71.25	30.43	-150
No CP text descr.	72.27	71.49	16.85	9.29
Ensembling strategies				
Average configuration	20.65	71.24	17.52	-11.08
Mode configuration	21.08	71.44	18.11	-12.63
Smallest configuration	20.83	70.91	17.42	-4.74

Disabling CPs can reduce performance

Our **CP descriptions** boost performance

k-medoids outperforms simpler heuristics

Lawless, Li, Wikum, Udell, **V**, CPAIOR'25

Recap

Can we use LLMs to configure MILP solvers with minimal training data?

- New **LLM-based framework** to configure cutting plane separators
- Finds high-performing configuration by **solving only a few MILPs**
- **Ensembling strategy** to build portfolio of high-performing configurations
- Requires **no custom solver interface**
- Competitive with existing configuration approaches
  but only requires a **fraction of the training data and computation time**

Outline

1. Background
2. LLMs for speeding up solvers
    i. Solver configuration
    ii. Evolutionary search for MILP heuristics
        a. Background
        b. Cut selection
        c. Large neighborhood search
        d. Diving heuristics
3. Takeaways

LLMs for automated heuristic design

- Heuristic design is central to NP-hard optimization
  - But manual heuristic design is **slow**, relying on human ingenuity
- Automated heuristic design/scheduling goes back to the 1960s
  - Fisher, Thompson, ’63; Crowston et al., ’63; survey by Burke et al., JORS’13
- Automated heuristic design with LLMs:
  - LLMs generate heuristic code
  - Heuristic **fitness** scored automatically
  - **Evolutionary selection** keeps improved heuristics

E.g., FunSearch [Romera-Paredes, Nature’24], ReEvo [Ye et al., NeurIPS’24], AlphaEvolve [Novikov et al., ’25], …

Evolution of Heuristics [Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24]

Example: Online bin packing

- Task: Pack items of varying sizes into fewest bins
- Items arrive sequentially
  - Must be packed into a bin immediately
  - No knowledge of future arrivals
- Each bin has fixed capacity (experiments: $C = 100$)

Figure by Fawzi, Romera-Paredes

Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Example: Online bin packing

Heuristic representation

Heuristic

Code

```
def heuristic(item, bins):
    """
    item: scalar item size
    bins: 1D np.array of remaining capacities
    returns: per-bin scores (higher is better)
    ...
```

Natural language description

Incorporates a weighted average of the utilization ratio, dynamic adjustment, and an exponentially decaying factor, with different parameter settings to minimize the number of used bins.

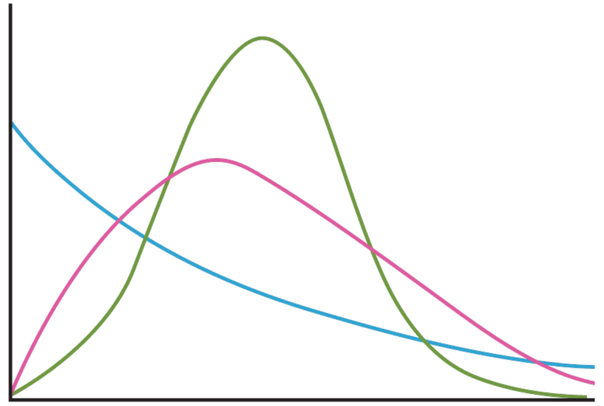
Fitness: 0.0196

Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Example: Online bin packing

Fitness metric

- **Test instances** [Romera-Paredes et al., Nature'24]**:**
  - Five Weibull test instances, each with 5000 items
- ℓb = lower bound on opt bin count [Martello & Toth '90]
- n = number of bins used by heuristic
- Fitness = avg $\left(\frac{\ell b}{n}\right)$ across the test instances



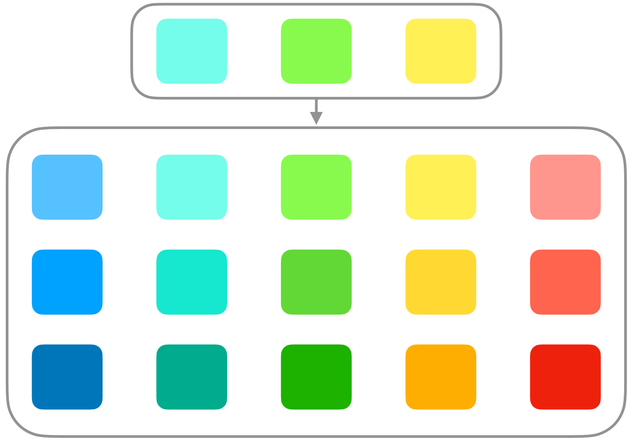
Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*
2. **Heuristic generation:**

   Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)

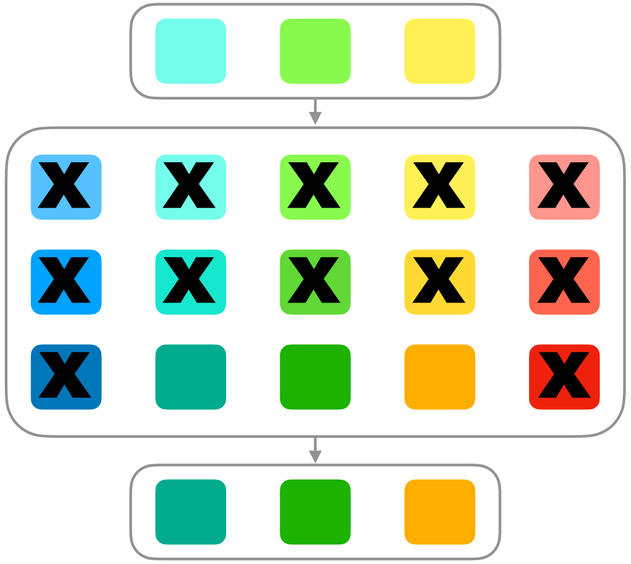
   i. Select parent heuristic(s) to form prompt
   ii. LLM generates new thought and code
   iii. Evaluate fitness on test instances
   iv. Add feasible heuristics to population



Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Algorithm overview

1. **Initialization:** generate N initial heuristics using *Initialization Prompt*
2. **Heuristic generation:**
   Apply 5 *Evolution Prompts* in parallel ($5N$ new heuristics)
   i. Select parent heuristic(s) to form prompt
   ii. LLM generates new thought and code
   iii. Evaluate fitness on test instances
   iv. Add feasible heuristics to population
3. **Retain** top N heuristics by fitness; return to Step 1



Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Example: Online bin packing

Initialization prompt

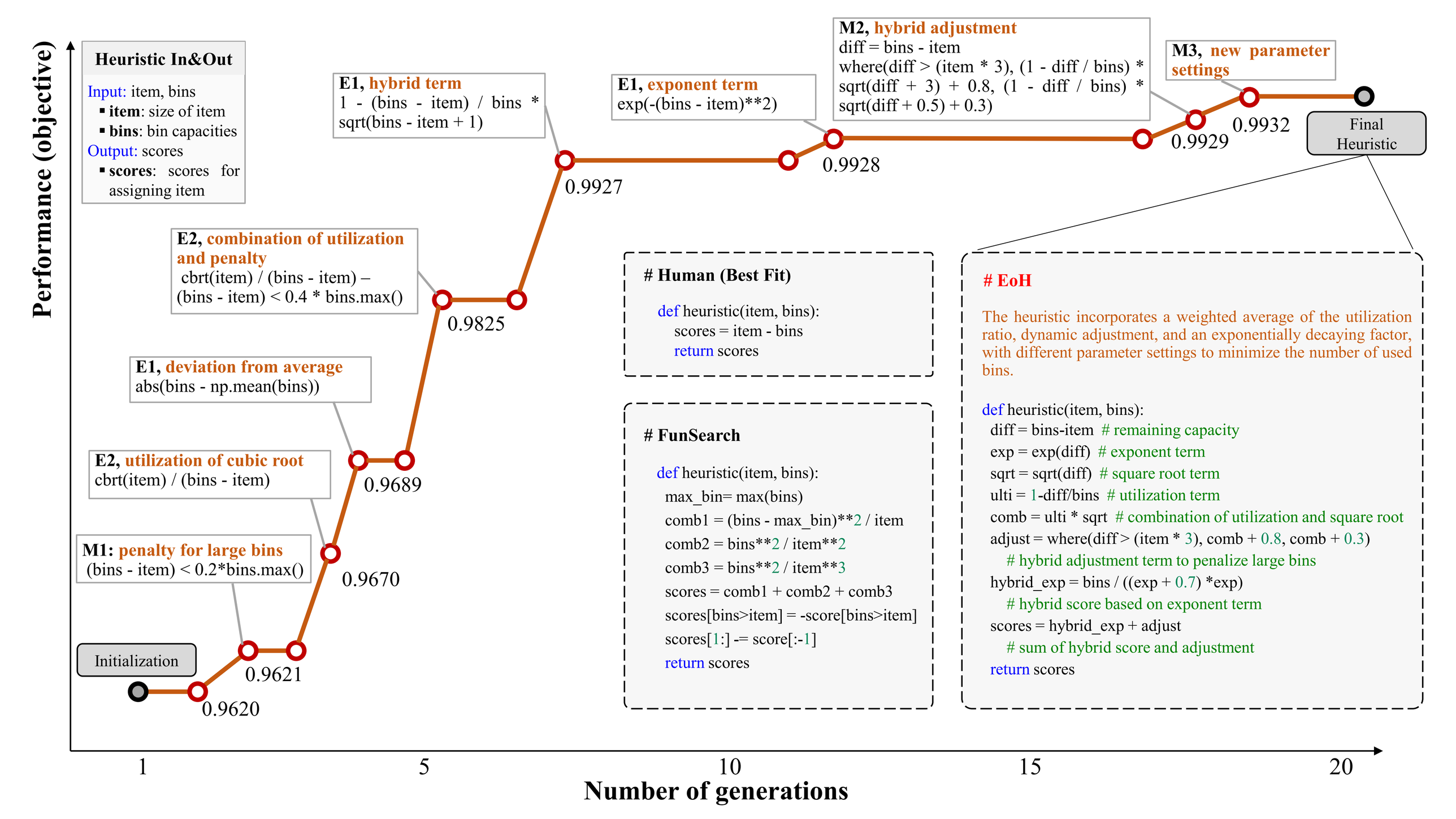
- Help design a new heuristic that scores a set of bins to assign an item
- In each step, the item will be assigned to the bin with the maximum score
- If a bin is full, it will not be used
- The final goal is to minimize the number of used bins
- Firstly, describe your new heuristic and main steps in one sentence

Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Evolution prompts

- **E1 – Diverse exploration:** generate entirely new heuristic ideas from scratch
- **E2 – Shared-Idea variants:** generate based on high-performing “themes”
- **M1 – Edit:** modify an existing heuristic
- **M2 – Parameter tuning:** fine-tune numeric settings or thresholds in code
- **M3 – simplification:** prune unnecessary components or redundant logic

Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML’24


Performance (objective)
Number of generations
1
5
10
15
20
Heuristic In&Out
Input: item, bins
item: size of item
bins: bin capacities
Output: scores
scores: scores for assigning item
Initialization
0.9620
0.9621
M1: penalty for large bins
(bins - item) < 0.2*bins.max()
0.9670
E2, utilization of cubic root
cbrt(item) / (bins - item)
0.9689
E1, deviation from average
abs(bins - np.mean(bins))
0.9825
E2, combination of utilization and penalty
cbrt(item) / (bins - item) – (bins - item) < 0.4 * bins.max()
0.9927
E1, hybrid term
1 - (bins - item) / bins * sqrt(bins - item + 1)
0.9928
E1, exponent term
exp(-(bins - item)**2)
0.9929
M2, hybrid adjustment
diff = bins - item
where(diff > (item * 3), (1 - diff / bins) * sqrt(diff + 3) + 0.8, (1 - diff / bins) * sqrt(diff + 0.5) + 0.3)
0.9932
M3, new parameter settings
Final Heuristic
# Human (Best Fit)
def heuristic(item, bins):
scores = item - bins
return scores
# FunSearch
def heuristic(item, bins):
max_bin= max(bins)
comb1 = (bins - max_bin)**2 / item
comb2 = bins**2 / item**2
comb3 = bins**2 / item**3
scores = comb1 + comb2 + comb3
scores[bins>item] = -score[bins>item]
scores[1:] -= score[:-1]
return scores
EoH
The heuristic incorporates a weighted average of the utilization ratio, dynamic adjustment, and an exponentially decaying factor, with different parameter settings to minimize the number of used bins.
def heuristic(item, bins):
diff = bins-item # remaining capacity
exp = exp(diff) # exponent term
sqrt = sqrt(diff) # square root term
ulti = 1-diff/bins # utilization term
comb = ulti * sqrt # combination of utilization and square root
adjust = where(diff > (item * 3), comb + 0.8, comb + 0.3)
# hybrid adjustment term to penalize large bins
hybrid_exp = bins / ((exp + 0.7) *exp)
# hybrid score based on exponent term
scores = hybrid_exp + adjust
# sum of hybrid score and adjustment
return scores

Example: Online bin packing

Experimental setup

- **Baselines:**
  - First Fit: place item in first bin that fits
  - Best Fit: place item in bin w/ least available space
  - Published FunSearch heuristic as-is
- **Problem sizes:** 1000–10,000 items
- **Capacities:** C = 100 and C = 500
- **Each setting:** 5 randomly generated instances

Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Bin packing results

	C=100			C=500		
Method	**1k items**	**5k items**	**10k items**	**1k items**	**5k items**	**10k items**
First Fit	5.32%	4.40%	4.44%	4.97%	4.27%	4.28%
Best Fit	4.87%	4.08%	4.09%	4.50%	3.91%	3.95%
FunSearch	3.78%	**0.80%**	**0.33%**	6.75%	1.47%	0.74%
EoH	**2.24%**	**0.80%**	0.61%	**2.13%**	**0.78%**	**0.61%**

Metric: average gap to lower bound [Martello & Toth '90]

Liu, Tong, Yuan, Lin, Luo, Wang, Lu, Zhang, ICML'24

Beyond EoH

Recent examples

- **Reflective Evolution** [Ye et al., NeurIPS'24]
  - Uses verbal gradients and reflective critiques to guide evolution
- **Multi-objective Evolution** of Heuristic [Yao et al., AAAI'25]
  - Evolves Pareto-optimal heuristics for multi-objective optimization
- **AlphaEvolve** [Novikov et al., arXiv'25]
  - Scales to file-level evolution
  - Open-source version: OpenEvolve [Sharma, GitHub'25]
  - Integrate with Deep Research methods: DeepEvolve [Liu et al., arXiv'25]

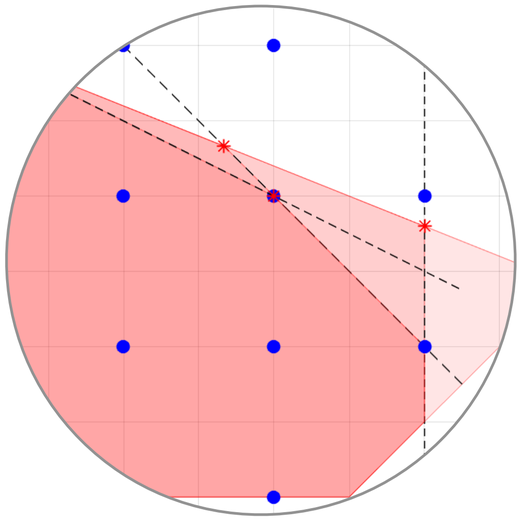
Outline

1. Background
2. LLMs for speeding up solvers
   i. Solver configuration
   ii. Evolutionary search for MILP heuristics
      a. Background
      b. Cut selection
      c. Large neighborhood search
      d. Diving heuristics
3. Takeaways

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Overview of approach

- **EvoCut** automates cut discovery using LLMs plus evolution
  - Initializes cut population; evolves via crossover and mutation
- Empirically checks **optimal-solution preservation** and fractional **separation**
- Scores cuts by relative **optimality-gap reduction**

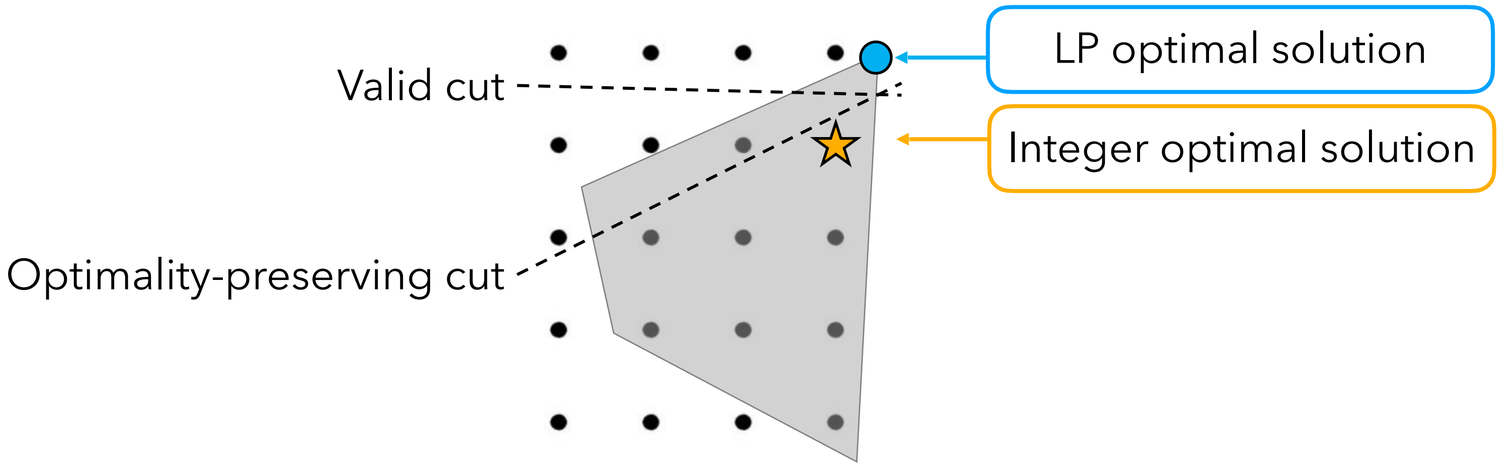


Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Acceleration cuts

Inequalities added to speed up MILP solving

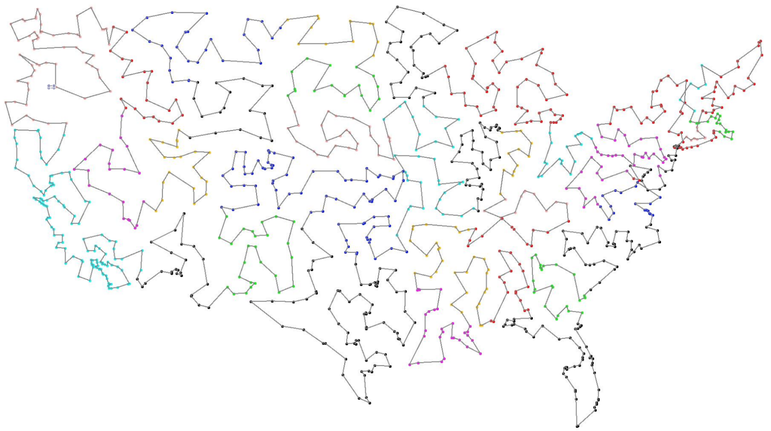
- **Valid cut:** doesn't separate any integer-feasible point
- **Optimality-preserving cut:** doesn't separate the opt integer-feasible point
- EvoCut cuts aren't proven optimality-preserving; empirically checked



Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities



Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \to j$)

Order variables: $u_i \in \{1,\ldots,n\}$(visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \to j$

Objective: $\min \sum_{i \neq j} c_{ij} x_{ij}$ (c_{ij} is the cost to travel $i \to j$)

Degree constraints (enter/leave exactly once)**:** $\sum_{j \neq i} x_{ij} = 1, \quad \sum_{j \neq i} x_{ji} = 1, \quad \forall i$

Subtour-elimination: $u_i - u_j + n x_{ij} \leq n - 1, \quad \forall i \neq j, \quad i,j \in \{2,\ldots,n\}$

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

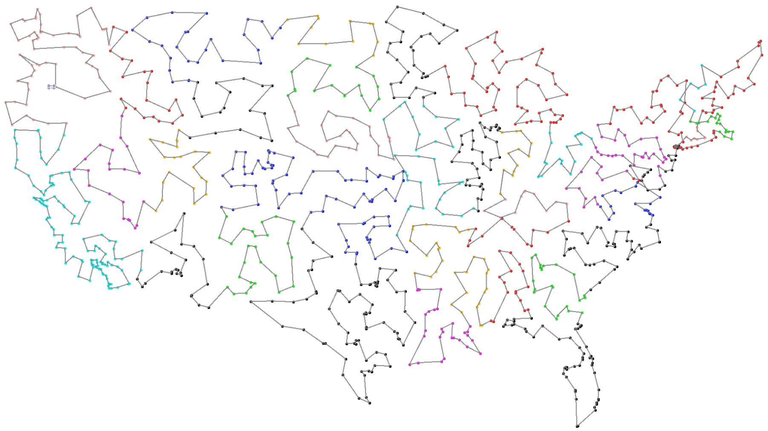
Example: Traveling salesman problem

Goal: find a minimum-cost Hamiltonian tour over n cities

Edge variables: $x_{ij} \in \{0,1\}, \forall i \neq j$ (travel $i \to j$)

Order variables: $u_i \in \{1,\ldots,n\}$(visit position)

Fix $u_1 = 1$ (start city); $u_j = 2$ means travel $1 \to j$



(Very simple) example of a CP found by EvoCut:

$$u_j \leq 2 + (n-2)(1 - x_{1j}), \forall j \in \{2,\ldots,n\}$$

- If $x_{1j} = 1, \quad u_j \leq 2$ (travel $1 \to j$)
- If $x_{1j} = 0, \quad u_j \leq n$

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Method overview

Data preprocessing

- Construct **evaluation** set D_e and **verification** set D_v of MILPs
- On D_e: run baseline solver $\forall z \in D_e$ under fixed computational budget
  - Record terminal **optimality gap** $\text{gap}_{\text{ref}}(z)$
- On D_v: run baseline solver to optimality and store **optimal solution**
  - Also store **solution to LP relaxation**

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Method overview

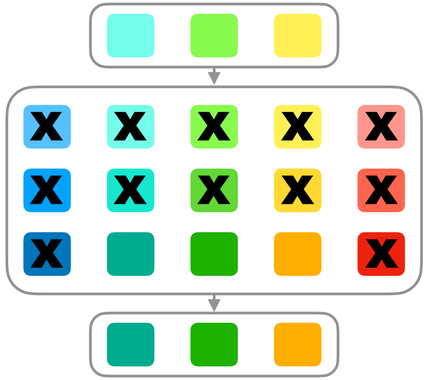
How to verify and evaluate a candidate cut

- Verification on D_v:
  1. Code must **compile**; errors trigger diagnostic prompt, retry
  2. **OSP**: maintains feasibility of optimal solutions across D_v
  3. **Usefulness**: separate LP-optimal solution on *some* D_v instance
- Run baseline solver $\forall z \in D_e$ instances under fixed computational budget
  - Record terminal optimality gap $\text{gap}_{\text{cut}}(z)$ **with cut**
- **Fitness**: average relative gap change over D_e: $\dfrac{\text{gap}_{\text{ref}}(z) - \text{gap}_{\text{cut}}(z)}{\text{gap}_{\text{ref}}(z)}$

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Method overview

Candidate cut generation



1. **Initializer LLM** seeds candidate cuts
   - Uses model code and natural language description

2. Population **evolved** for T generations
   - **Elitism**: carry top cuts into next generation
   - **Selection**: pick parents with probability proportional to fitness
   - **Reproduce**: crossover (merge two parents) or mutation

3. **Verify and evaluate** offspring before including in population
   - Failed offspring triggers feedback; retry up to max

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Experimental setup

- **Solver**: Gurobi 10.0.0; LLM: DeepSeek-R1
- **Benchmarks**:
  - TSP
  - Multi-Commodity Network Design (MCND)
  - Capacitated Warehouse Location Problem (CWLP)
  - Job Shop Scheduling Problem (JSSP)
- $|D_e| = 10, |D_v| = 2$; drawn from synthetic generators
- **Test set** D_t: public datasets; 40 hard medium/large instances each

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Snapshot of results

Checkpoints	5 s	10 s	50 s	150 s	300 s	OSP rate (%)
TSP	16.3 ± 24.9	15.4 ± 27.3	27.7 ± 31.1	44.4 ± 27.7	57.4 ± 26.3	100
MCND	9.4 ± 21.1	6.3 ± 22.0	11.7 ± 19.1	10.4 ± 18.4	17.1 ± 20.2	100
CWLP	-6.9 ± 17.0	-8.3 ± 15.1	24.0 ± 24.9	42.5 ± 21.3	46.2 ± 41.1	100
JSSP	22.8 ± 18.3	28.8 ± 19.7	39.1 ± 22.8	34.5 ± 22.1	37.3 ± 22.0	100

Relative mean gap improvement over D_t

- **Evolution helps:** initializer-only cuts have lower success

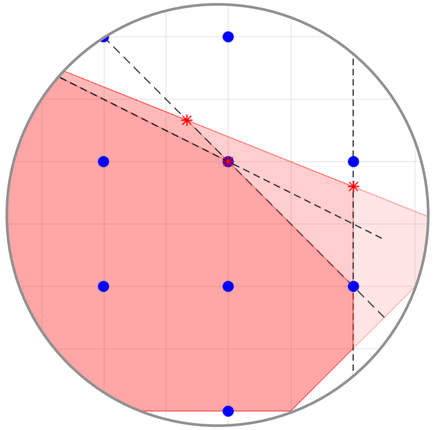
  Check (i) code rejections, (ii) OSP rejections, (iii) usefulness rejections, and (iv) fitness

  - Mutation/crossover succeed 63–82% attempts; initializer 25.4%

Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Takeaways

- **EvoCut**: Evolutionary search automates cut discovery
- Optimal-solution preservation and fractional separation verified **empirically**
- Cuts improve **optimality-gap reduction** throughout solve trajectory



Yazdani, Mostajabdaveh, Aref, Zhou, arXiv'25

Outline

1. Background
2. LLMs for speeding up solvers
   i. Solver configuration
   ii. Evolutionary search for MILP heuristics
      a. Background
      b. Cut selection
      c. Large neighborhood search
      d. Diving heuristics
3. Takeaways

Ye, Xu, Yan, Cheng, ICML'25

LLMs for large neighborhood search (LNS)

- Solving large-scale MILPs need strong **primal heuristics**
- LNS is effective; neighborhood choice dominates performance
  - Without ML: Fischetti, Lodi, MP'03; Danna et al., MP'05; …
  - With ML: Song et al., NeurIPS'20; Wu et al., NeurIPS'21; Huang et al., ICML'23; …
- Prior neighborhood selection methods need **expertise** or **heavy training**
- Propose **LLM-LNS**: LLM-guided neighborhood scoring for MILPs

Ye, Xu, Yan, Cheng, ICML'25

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
  - **Destroy**: fix most variables; free a selected subset
  - **Repair**: solve resulting MILP subproblem
- Neighborhood choice controls search power and runtime tradeoff
  - **Small neighborhoods:** fast but limited improvement
  - **Large neighborhoods:** powerful but expensive

Ye, Xu, Yan, Cheng, ICML'25

Large neighborhood search (LNS) for MILPs

- LNS improves feasible solutions by partial variable re-optimization
- Iteration alternates between **destroy** and **repair** phases
  - **Destroy**: fix most variables; free a selected subset
  - **Repair**: solve resulting MILP subproblem

For each integer variable x_i:

- Fix all other integer variables to current values
- Relax x_i; solve resulting LP
- Measure deviation from integer solution value
- Free/destroy variables with largest deviation

Ye, Xu, Yan, Cheng, ICML'25

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
  - Select top-k scores; free (i.e., destroy) those variables in subproblem
    - **Example prompt:**

- Given feasible MILP solution, plus bounds and objective coefficients for variables
- Goal: improve the current solution using LNS
- Description of LNS
- You must score all variables for neighborhood selection decisions
- Add some randomness in subset choice to avoid local optima and stagnation
- m prior scoring rules: description, average fitness (objective value) over training set, code
- Create a totally different new scoring rule

Ye, Xu, Yan, Cheng, ICML'25

LLM-LNS framework: Method overview

Inner-outer agent

- **Inner agent:** prompted to score variables to define neighborhoods
  - Select top-k scores; free (i.e., destroy) those variables in subproblem
- **Outer agent** evolves prompts to improve inner strategies
  - **Example prompt:**

> - Goal: solve minimization problem by using LLM-generated LNS heuristics
> - We already designed several initial prompts and collected the LLM's resulting heuristics
> - Analyze each prompt and the objective function values achieved by LNS heuristic
> - Create new prompt w/ different structure, motivated by insights from prompt–value pairs

Ye, Xu, Yan, Cheng, ICML'25

Experimental setup

Benchmarks

- **Benchmarks**:
  - Set cover (SC)
  - Min vertex cover (MVC)
  - Max independent set (MIS)
  - See paper for mixed integer knapsack set (MIKS)
- Tested on instances with (on the order of) 10^6 variables/constraints
- Train on **small-scale instances** for scalable heuristic evolution
  - Training instances are roughly **100x smaller** than the test instances

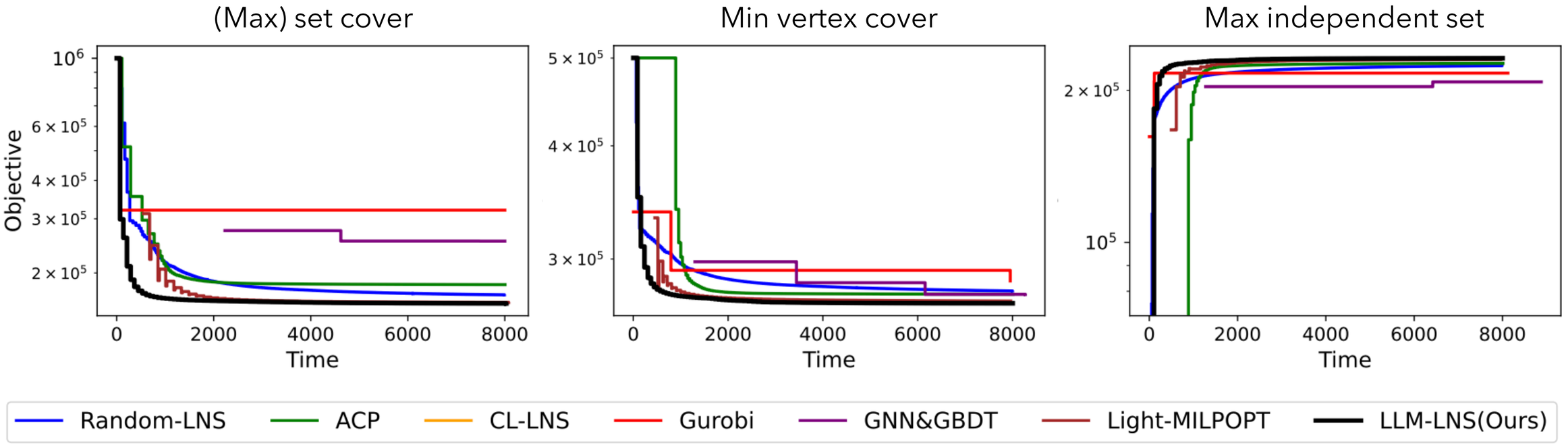
Experimental setup

Baselines

- **Random-LNS:** random variable neighborhoods in LNS
- **ACP**: adaptive constraint propagation guides neighborhood choice
  Ye et al., AAAI'23
- **CL-LNS**: contrastive learning selects LNS neighborhoods
  Huang et al., ICML'23
- **GNN&GBDT**: ML pipeline combining graphs and boosted trees
  Ye et al., ICML'23
- **Light-MILPopt**: lightweight ML framework for MILP optimization
  Ye et al., ICLR'23

Experiments

Snapshot of results



- LLM-LNS is mostly **Pareto optimal** (see paper for many more experiments)

Ye, Xu, Yan, Cheng, ICML'25

Takeaways

- Prior neighborhood selection methods need **expertise** or **heavy training**
- **LLM-LNS**: LLM-guided neighborhood scoring for MILPs
- Inner-outer agent evolves neighborhood selection heuristics **and** prompts

Ye, Xu, Yan, Cheng, ICML'25

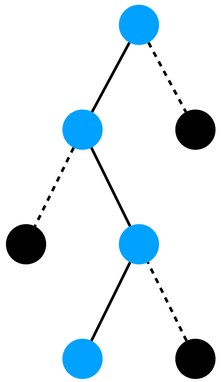
Outline

1. Background
2. LLMs for speeding up solvers
   i. Solver configuration
   ii. Evolutionary search for MILP heuristics
      a. Background
      b. Cut selection
      c. Large neighborhood search
      d. Diving heuristics
3. Takeaways

Zhang, Li, Li, Liu, Chen, Li, Zhong, An, Liu, arXiv'25

Evolutionary search for diving heuristics

- **Diving**: quickly explores a single path (a "dive") in the search tree
  - Iteratively fixing fractional variables to integer values, solving LP relaxation
  - Goal is to find a feasible integer solution
- Policy choices: **variable order** and **rounding direction**
- **DHEvo** aims to evolve hard instances with improving heuristics
  - Multi-agent LLM generates MILP–heuristic pairs jointly



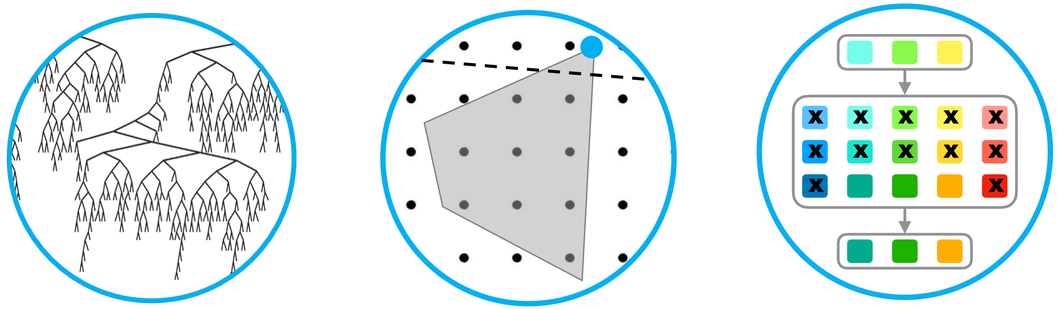
Zhang, Li, Li, Liu, Chen, Li, Zhong, An, Liu, arXiv'25

Outline

1. Background
2. LLMs for speeding up solvers
3. **Takeaways**

Takeaways

- Solver defaults are rarely optimal: instance families vary widely in structure
- **Many** ways to integrate ML into solvers
- LLMs can be leveraged in many ways to **speed up solvers**
  - **Solver configuration:** LLMs as excellent information retrieval systems
  - **Evolutionary search** for MILP heuristics: LLMs to explore semantic search space
    - Cut selection
    - Large neighborhood search
    - Diving heuristics



Thank you! Questions?