Machine learning for optimization (5/5)

Clément W. Royer

M2 MODO - 2025/2026

January 28, 2026 (updated)

Today's session

- One topic left: ML for MILP.
- Issue: Find an easy idea to replicate in a lab session.
- Many interesting ideas, not enough time/computational capacity!

Fix Go back to a lecture mode (for the first part of the session).

Course project

- Finish the three lab sessions (if not done yet).
- Study the suggested papers.

Roadmap

1. Branch-and-Bound for MILP
2. Learning tasks for MILP

Roadmap

1. Branch-and-Bound for MILP
2. Learning tasks for MILP

Mixed-integer linear programs

$$
\begin{array}{lll}
\text{minimize}_{\boldsymbol{x}\in\mathbb{R}^n} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & \\
\text{s.t.} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} & \\
& x_j \in \mathbb{Z}_{\geq 0} & \forall j \in \mathcal{A} \\
& x_j \in \{0,1\} & \forall j \in \mathcal{B} \\
& x_j \geq 0 & \forall j \in \mathcal{C}
\end{array}
$$

- $\boldsymbol{A} \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^m$, $\boldsymbol{c} \in \mathbb{Q}^n$.
- $\mathcal{A} \cup \mathcal{B} = \mathcal{I}$: Discrete variables (integer+binary).
- $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ partition of $\{1, \ldots, n\}$.

Mixed-integer linear programs

$$
\begin{array}{lll}
\text{minimize}_{\boldsymbol{x}\in\mathbb{R}^n} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} & \\
\text{s.t.} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} & \\
& x_j \in \mathbb{Z}_{\geq 0} & \forall j \in \mathcal{A} \\
& x_j \in \{0,1\} & \forall j \in \mathcal{B} \\
& x_j \geq 0 & \forall j \in \mathcal{C}
\end{array}
$$

- $\boldsymbol{A} \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^m$, $\boldsymbol{c} \in \mathbb{Q}^n$.
- $\mathcal{A} \cup \mathcal{B} = \mathcal{I}$: Discrete variables (integer+binary).
- $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ partition of $\{1, \ldots, n\}$.

LP relaxation

Replace integer constraints by

$$
\begin{array}{ll}
x_j \geq 0 & \forall j \in \mathcal{A} \\
x_j \in [0,1] & \forall j \in \mathcal{B}.
\end{array}
$$

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities
  - $\bar{z}$: Upper bound on the optimal value (from an integer feasible solution.

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities
  - $\bar{z}$: Upper bound on the optimal value (from an integer feasible solution.
  - $\underline{z}$: Lower bound from the best optimal value obtained by LP relaxations.

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities
  - $\bar{z}$: Upper bound on the optimal value (from an integer feasible solution.
  - $\underline{z}$: Lower bound from the best optimal value obtained by LP relaxations.
- Stops when

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities
  - $\bar{z}$: Upper bound on the optimal value (from an integer feasible solution.
  - $\underline{z}$: Lower bound from the best optimal value obtained by LP relaxations.
- Stops when
  1. $\underline{z} = \bar{z}$, or

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities
  - $\bar{z}$: Upper bound on the optimal value (from an integer feasible solution.
  - $\underline{z}$: Lower bound from the best optimal value obtained by LP relaxations.
- Stops when
  1. $\underline{z} = \bar{z}$, or
  2. all nodes from the tree have been explored, or

Solvers for MILP

Most state-of-the-art solvers use variants of the **branch-and-bound** paradigm.

- Build a tree of LP relaxations based on adding continuous bounds on variables one at a time.
- At every iteration, use the quantities
  - $\bar{z}$: Upper bound on the optimal value (from an integer feasible solution.
  - $\underline{z}$: Lower bound from the best optimal value obtained by LP relaxations.
- Stops when
  1. $\underline{z} = \bar{z}$, or
  2. all nodes from the tree have been explored, or
  3. time out.

Example (can do subproblems with Python!)

$$
\begin{array}{lll}
\text{minimize}_{\boldsymbol{x}\in\mathbb{R}^2} & -(3x_1 + x_2) & \\
\text{s.t.} & -x_1 + x_2 & \leq \quad 2 \\
& 8x_1 + x_2 & \leq \quad 19 \\
& x_1, x_2 & \geq \quad 0 \\
& x_1, x_2 & \in \quad \mathbb{Z}.
\end{array}
$$

Inside an MILP solver

- **Preprocessing** Reduce the number of constraints and/or variables.
  $\rightarrow$ Fundamental in practice, empirical in nature.

Inside an MILP solver

- **Preprocessing** Reduce the number of constraints and/or variables.
  $\rightarrow$ Fundamental in practice, empirical in nature.
- **Branching selection** Given an LP relaxation solution $\boldsymbol{x}^{LP}$ and several variables $x_j^{LP} \in \mathbb{Q} \setminus \mathbb{Z}$, decide which one to branch on$\rightarrow$ No generic rule.

Inside an MILP solver

- **Preprocessing** Reduce the number of constraints and/or variables.
  $\rightarrow$ Fundamental in practice, empirical in nature.
- **Branching selection** Given an LP relaxation solution $\boldsymbol{x}^{LP}$ and several variables $x_j^{LP} \in \mathbb{Q} \setminus \mathbb{Z}$, decide which one to branch on$\rightarrow$ No generic rule.
- **Node selection** Given several LP relaxations/tree nodes, which one to solve first?

Inside an MILP solver

- **Preprocessing** Reduce the number of constraints and/or variables.
  $\rightarrow$ Fundamental in practice, empirical in nature.
- **Branching selection** Given an LP relaxation solution $\boldsymbol{x}^{LP}$ and several variables $x_j^{LP} \in \mathbb{Q} \setminus \mathbb{Z}$, decide which one to branch on$\rightarrow$ No generic rule.
- **Node selection** Given several LP relaxations/tree nodes, which one to solve first?
- **Cutting planes** Strengthen LP relaxations by adding inequality constraints (cuts)
  $\rightarrow$ Cuts at root node or at every node (Branch-and-cut).
  $\rightarrow$ Solvers have a cut management strategy (too many cuts slow down a solve).

Inside an MILP solver

- **Preprocessing** Reduce the number of constraints and/or variables.
  $\rightarrow$ Fundamental in practice, empirical in nature.
- **Branching selection** Given an LP relaxation solution $\boldsymbol{x}^{LP}$ and several variables $x_j^{LP} \in \mathbb{Q} \setminus \mathbb{Z}$, decide which one to branch on$\rightarrow$ No generic rule.
- **Node selection** Given several LP relaxations/tree nodes, which one to solve first?
- **Cutting planes** Strengthen LP relaxations by adding inequality constraints (cuts)
  $\rightarrow$ Cuts at root node or at every node (Branch-and-cut).
  $\rightarrow$ Solvers have a cut management strategy (too many cuts slow down a solve).
- **Primal heuristics** Used to find integer feasible solutions quickly.

Inside an MILP solver

- **Preprocessing** Reduce the number of constraints and/or variables.
  $\rightarrow$ Fundamental in practice, empirical in nature.
- **Branching selection** Given an LP relaxation solution $\boldsymbol{x}^{LP}$ and several variables $x_j^{LP} \in \mathbb{Q} \setminus \mathbb{Z}$, decide which one to branch on$\rightarrow$ No generic rule.
- **Node selection** Given several LP relaxations/tree nodes, which one to solve first?
- **Cutting planes** Strengthen LP relaxations by adding inequality constraints (cuts)
  $\rightarrow$ Cuts at root node or at every node (Branch-and-cut).
  $\rightarrow$ Solvers have a cut management strategy (too many cuts slow down a solve).
- **Primal heuristics** Used to find integer feasible solutions quickly.

SCIP has $\geq$ 2000 parameters for configuration!

ML for MILP

- General goal: Learn one parameter/heuristic at a time!

ML for MILP

- General goal: Learn one parameter/heuristic at a time!
- Many possibilities≡Many research papers!

ML for MILP

- General goal: Learn one parameter/heuristic at a time!
- Many possibilities≡Many research papers!
- Most notable publications: Surveys!

ML for MILP

- General goal: Learn one parameter/heuristic at a time!
- Many possibilities≡Many research papers!
- Most notable publications: Surveys!

Applying ML to learn an MILP solver component

- Requires a learning task.
- Requires data.
- Requires to design a model to learn from data.

Roadmap

1. Branch-and-Bound for MILP
2. Learning tasks for MILP

Learning paradigms

Supervised learning Given data $\{(X_i, Y_i)\}_{i=1,\ldots,n}$, learn a model $f(\cdot; \boldsymbol{\theta})$ by solving

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(f(X_i; \boldsymbol{\theta}), Y_i\right)$$

where ℓ is a loss function.

Learning paradigms

Supervised learning Given data $\{(X_i, Y_i)\}_{i=1,\ldots,n}$, learn a model $f(\cdot; \boldsymbol{\theta})$ by solving

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(f(X_i; \boldsymbol{\theta}), Y_i\right)$$

where ℓ is a loss function.

Reinforcement learning Sequential decision making

- At each time t, given state s_t, an agent performs action a_t and receives reward R_t.

Learning paradigms

Supervised learning Given data $\{(X_i, Y_i)\}_{i=1,\dots,n}$, learn a model $f(\cdot; \boldsymbol{\theta})$ by solving

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(f(X_i; \boldsymbol{\theta}), Y_i\right)$$

where ℓ is a loss function.

Reinforcement learning Sequential decision making

- At each time t, given state s_t, an agent performs action a_t and receives reward R_t.
- Goal: Learn a policy π for the agent that maps states to actions by maximizing reward:

$$\underset{\pi}{\text{maximize}}\, \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} R_t(\pi)\right],$$

where $\gamma \in [0, 1]$.

Learning paradigms ('ed)

Imitation learning

- Reinforcement learning where the best action for a state is provided by an expert.
- Quite popular in the literature on ML for MILP.

Learning paradigms ('ed)

Imitation learning

- Reinforcement learning where the best action for a state is provided by an expert.
- Quite popular in the literature on ML for MILP.

Online/offline learning

- Offline Learn once and for all by training, then run the solver with what you learned (inference).
- Online Do training as the solver proceeds
  More adaptive, but expensive to run and implement.

Examples of learning tasks

Primal heuristics

- Given a family of heuristics, learn how to schedule them for a specific instance (offline).

Examples of learning tasks

Primal heuristics

- Given a family of heuristics, learn how to schedule them for a specific instance (offline).
- Predict a partial assignment of integer variable by
  - supervised learning on known feasible sets.
  - supervised learning on known optimal values.

Examples of learning tasks

Primal heuristics

- Given a family of heuristics, learn how to schedule them for a specific instance (offline).
- Predict a partial assignment of integer variable by
  - supervised learning on known feasible sets.
  - supervised learning on known optimal values.

Cutting planes

- Practice: Given an LP relaxation, generate various cuts, select some of them to remove fractional solutions.
- Idea: Learn cut selection from imitation/reinforcement learning.

Examples of learning tasks

Primal heuristics

- Given a family of heuristics, learn how to schedule them for a specific instance (offline).
- Predict a partial assignment of integer variable by
  - supervised learning on known feasible sets.
  - supervised learning on known optimal values.

Cutting planes

- Practice: Given an LP relaxation, generate various cuts, select some of them to remove fractional solutions.
- Idea: Learn cut selection from imitation/reinforcement learning.

Node selection

- Classical approach: Use best known lower bound (best first search), no recent advances.
- Idea: Learn offline by supervised learning.

Learning branching rules

Strong branching (classical)

- At node i: z^{LP_i} is the optimal LP relaxation value, and $z^{LP_i,k+}/z^{LP_i,k-}$ those obtained by branching on variable x_k.
  Score of x_k: Computed from $z^{LP_i,k+} - z^{LP_i}$ and $z^{LP_i,k-} - z^{LP_i}$.
- Deciding with strong branching can reduce the size of the branch and bound tree.
- Downside: Must solve extra LPs.

Learning branching rules

Strong branching (classical)

- At node i: z^{LP_i} is the optimal LP relaxation value, and $z^{LP_i,k+}/z^{LP_i,k-}$ those obtained by branching on variable x_k.
  Score of x_k: Computed from $z^{LP_i,k+} - z^{LP_i}$ and $z^{LP_i,k-} - z^{LP_i}$.
- Deciding with strong branching can reduce the size of the branch and bound tree.
- Downside: Must solve extra LPs.

Learning branching strategies (examples)

- Learn to predict strong branching scores offline or online.
- Learn variable rankings according to their scores.
- Learn branching rules from scratch.

Branching and Graph Neural Networks

From Gasse et al '19 Learn strong branching rules from an expert (imitation).

- State (of solver): Current node and LP relaxation, upper/lower bounds $\bar{z}/\underline{z}$.

Key aspects

- One benchmark: 10000 random instances for training, solved with SCIP to get best action.
- Re-implementation of strong branching from SCIP.
- Training takes >3 hours.

Branching and Graph Neural Networks

From Gasse et al '19 Learn strong branching rules from an expert (imitation).

- State (of solver): Current node and LP relaxation, upper/lower bounds $\bar{z}/\underline{z}$.
- Actions: Strong branching.

Key aspects

- One benchmark: 10000 random instances for training, solved with SCIP to get best action.
- Re-implementation of strong branching from SCIP.
- Training takes >3 hours.

Branching and Graph Neural Networks

From Gasse et al '19 Learn strong branching rules from an expert (imitation).

- State (of solver): Current node and LP relaxation, upper/lower bounds $\bar{z}/\underline{z}$.
- Actions: Strong branching.
- Policy: Probability vector of branching on variable k.

Key aspects

- One benchmark: 10000 random instances for training, solved with SCIP to get best action.
- Re-implementation of strong branching from SCIP.
- Training takes >3 hours.

Branching and Graph Neural Networks

From Gasse et al '19 Learn strong branching rules from an expert (imitation).

- State (of solver): Current node and LP relaxation, upper/lower bounds $\bar{z}/\underline{z}$.
- Actions: Strong branching.
- Policy: Probability vector of branching on variable k.
- GNN model: 3 layers, convolutions+fully connected+ReLU (not so different from our lab session).

Key aspects

- One benchmark: 10000 random instances for training, solved with SCIP to get best action.
- Re-implementation of strong branching from SCIP.
- Training takes >3 hours.

Additional tasks (non-exhaustive list)

- MIP 2023 challenge: Learn efficient solver configurations from past instances.
- CPLEX: Predict whether an MI**Q**P should be linearized before solving with CPLEX or not (2018).
- Predicting Lagrange multipliers for Lagrangian relaxation of MILPs (2024).

Data for learning and MILP

Challenges

- Highly task-dependent
  Ex) To learn branching rules, need instances for which optimality is hard to certify.
- Instance distributions/generators needed (not standard).

$\rightarrow$ Surveys are valuable but not all datasets are easy to find.

Data for learning and MILP

Challenges

- Highly task-dependent
  Ex) To learn branching rules, need instances for which optimality is hard to certify.
- Instance distributions/generators needed (not standard).

$\rightarrow$ Surveys are valuable but not all datasets are easy to find.

Recent developments: Platforms/packages

- Ecole (for reinforcement learning+SCIP, especially).
- MIPlearn (+ CPLEX/Gurobi/XPRESS), currently version 0.4.
- ...

References

- Y. Bengio, A. Lodi, A. Prouvost, *Machine learning for combinatorial optimization: A methodological tour d'horizon*, European Journal on Operational Research, 2021.
- P. Bonami, A. Lodi, G. Zarpellon, *Learning a classification of mixed-integer quadratic programming problems*, CPAIOR 2018.
- F. Demelas, J. Le Roux, M. Lacroix, A. Parmentier, *Predicting Lagrangian Multipliers for Mixed Integer Linear Programs*, ICML 2024.
- M. Gasse, D. Chételat, N. Ferroni, L. Charlin, A. Lodi, *Exact combinatorial optimization with graph convolutional neural networks*, NeurIPS 2019.
- L. Scavuzzo, K. Aardal, A. Lodi, N. Yorke-Smith, *Machine learning augmented branch-and-bound for mixed integer linear programming*, Mathematical Programming, 2024.

Typical venues for such research

- **Machine Learning conferences**:
  - NeurIPS, ICML, ICLR.
  - AAAI, AISTATS,...
- **Optimization journals**:
  - Mathematical Programming, European Journal on Operational Research.
  - INFORMS Journal on Computing, Open Journal on Mathematical Optimization, ...

Back to the course (project)

Our to-do list

- Make sure you understand how branch-and-bound works.
- Pick two papers (look at them all, possibly).
- Digest their methodology.
- Compare these approaches as part of the project.