Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming

Andrea Lodi
andrea.lodi@cornell.edu
Joint work with Lara Scavuzzo, Karen Aardal and Neil Yorke-Smith

CO@Work 2024

ZIB, Berlin, September 23, 2024

Motivation

The use of Machine Learning (ML) for Combinatorial Optimization (CO) — and Mixed-Integer Linear Programming (MILP) — problems has been ubiquitous in the last 5-10 years at the very least.

This is due to the incredible success of ML, especially deep learning, in beating human capabilities in image recognition, language processing and games.

Those successes led to ask natural questions about using modern statistical learning in other disciplines, CO being one of them.

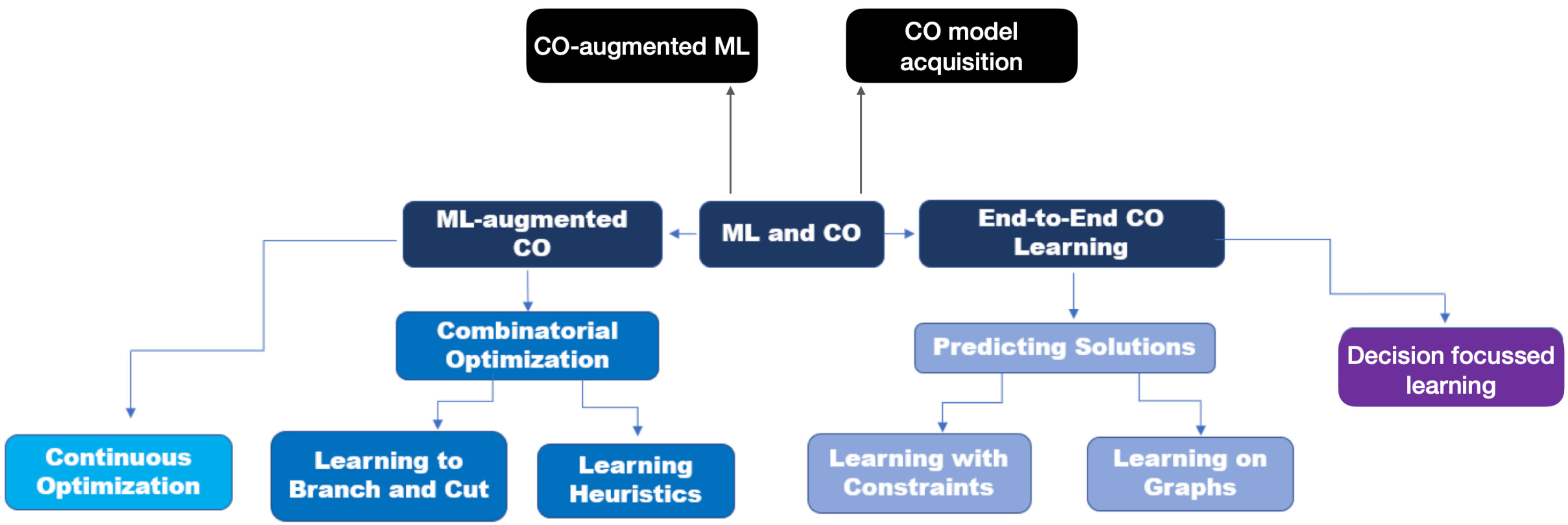
Outline

The talk

- First, **briefly** discusses MILP, a successful story of mathematics, algorithms and software development.
- Then, reviews MILP at the time of Artificial Intelligence: Methodological directions in which the use of Machine Learning is changing and will change (?) MILP.
- And, finishes with some perspectives, challenges and opportunities.

Schematic Overview

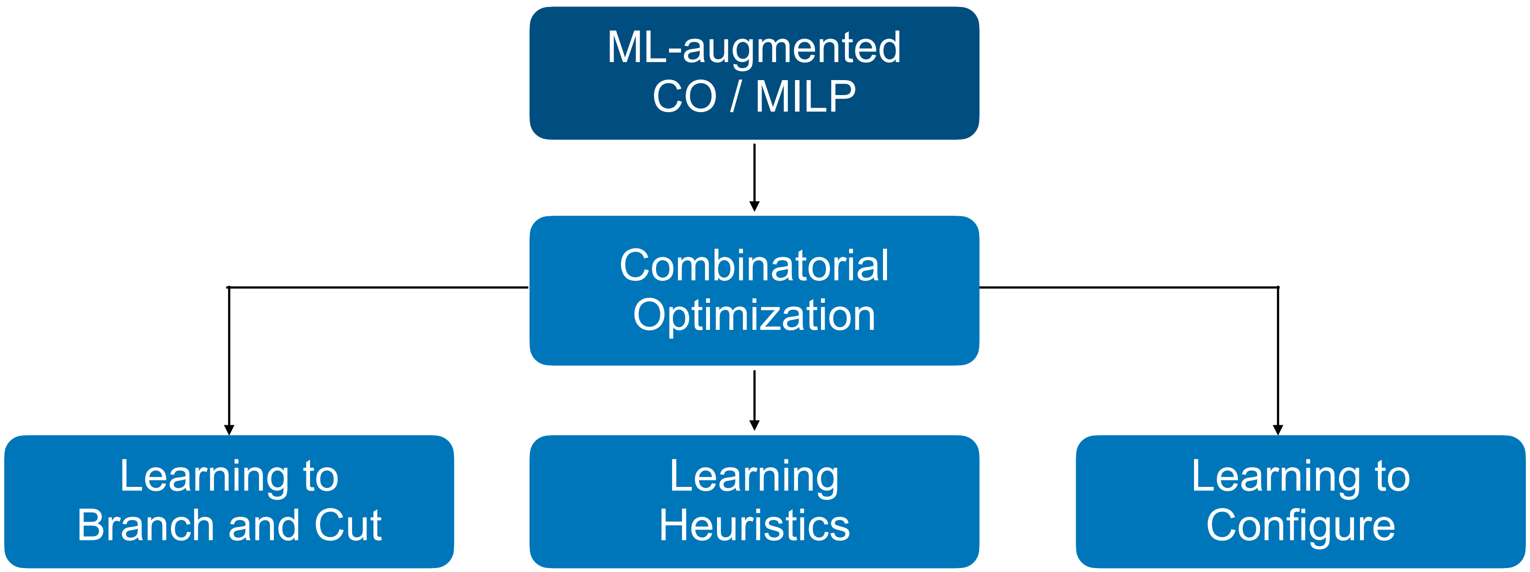
Slide courtesy of N. Yorke-Smith



Y. Bengio, A. Lodi, A. Prouvost: Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon, EJOR 2021, 405-421

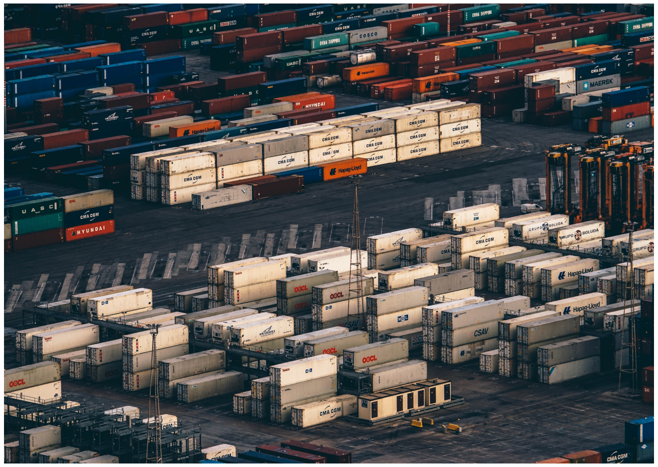
J. Kotary, F. Fioretto, P. Van Hentenryck, B. Wilder: End-to-End Constrained Optimization Learning: A Survey. IJCAI 2021: 4475-4482

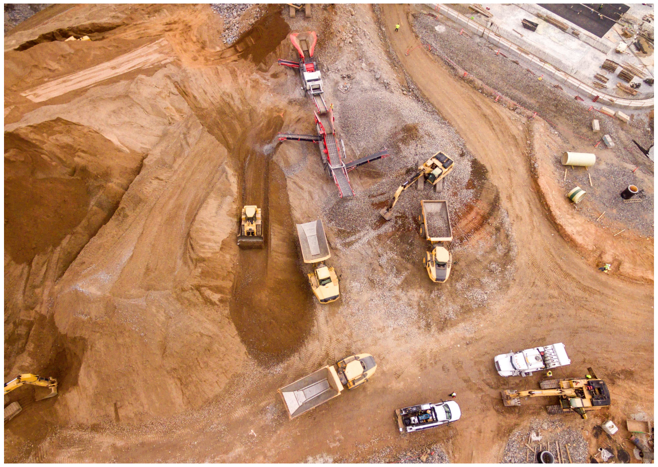
ML-augmented MILP



L. Scavuzzo, K. Aardal, A. Lodi, N. Yorke-Smith: Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming, arXiv:2402.05501, 2024,
Mathematical Programming https://doi.org/10.1007/s10107-024-02130-y

Mixed-Integer Linear Programming: Where?

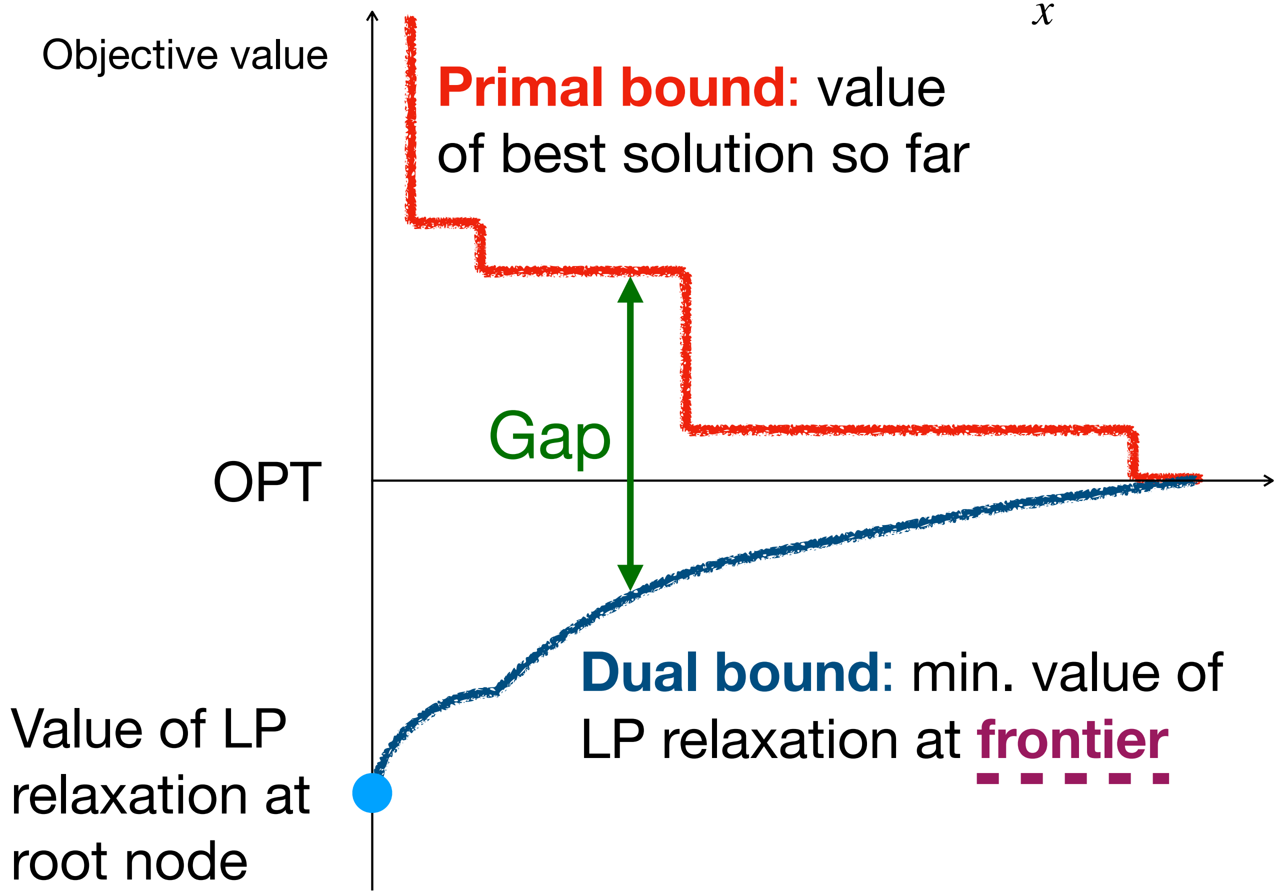

HYUNDAI
MAERSK
OOCL
CSAV

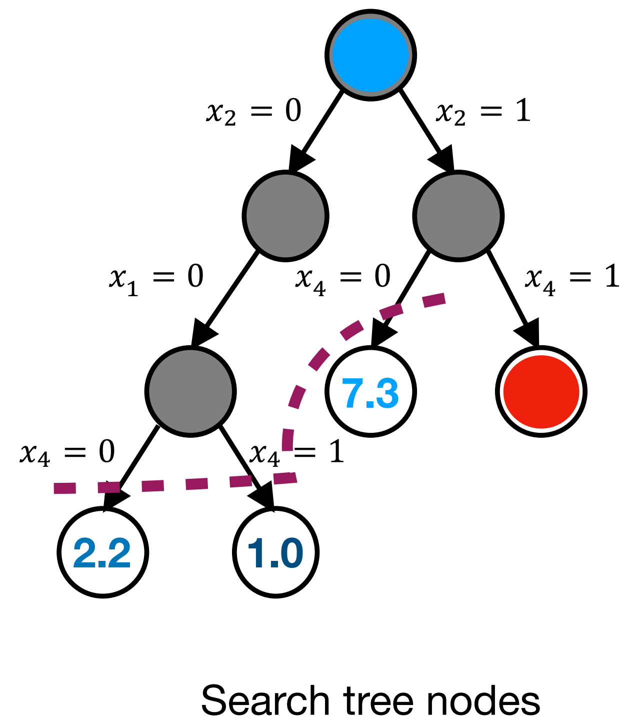


Mixed-Integer Linear Programming: How?

Branch and Bound (B&B)

$$\min_x c^T x \textbf{ s.t. } Ax \le b, x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$$





Slide courtesy of E. Khalil

MILP Key Features

The current generation of MILP solvers incorporates pretty much everything that has been developed since 1958 (Gomory's seminal work).

The algorithms can be grouped in four building blocks:

- Preprocessing / Configuration
- Cutting Plane Generation
- Sophisticated Branching Strategies
- Primal Heuristics

Preprocessing / Configuration

In the preprocessing phase a MILP solver tries to

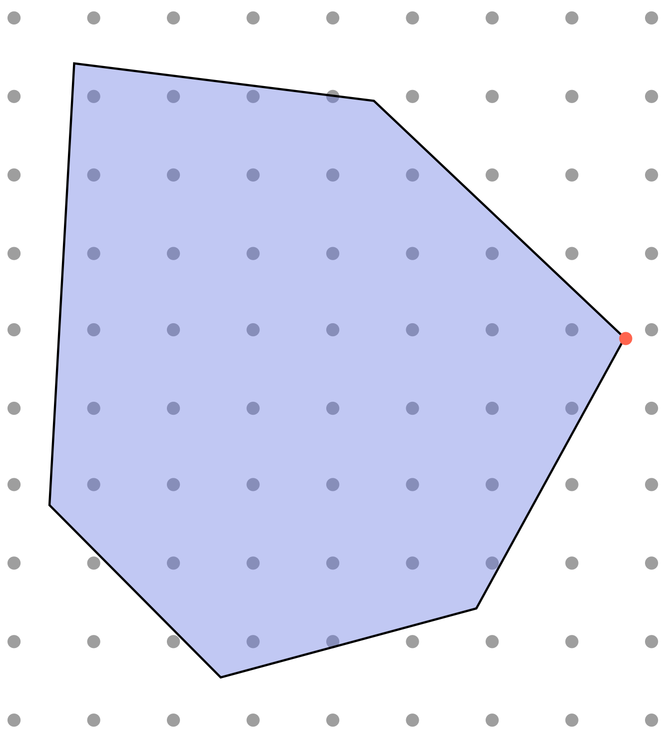
- detect certain changes in the input, and
- configure the algorithm

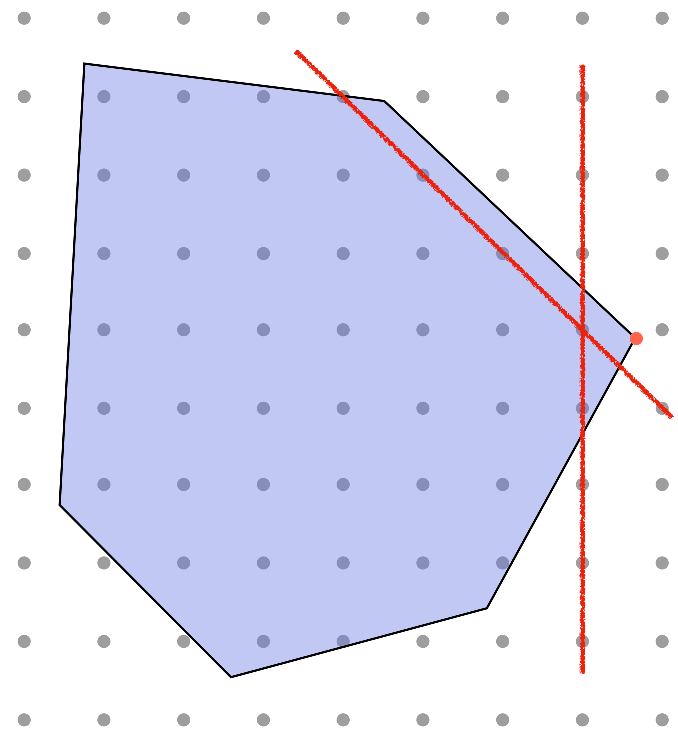
so as to likely obtain a better performance of the solution process.

In terms of detection, the MILP is cleaned and potentially strengthened by heuristically discovering implications that improve the LP relaxation.

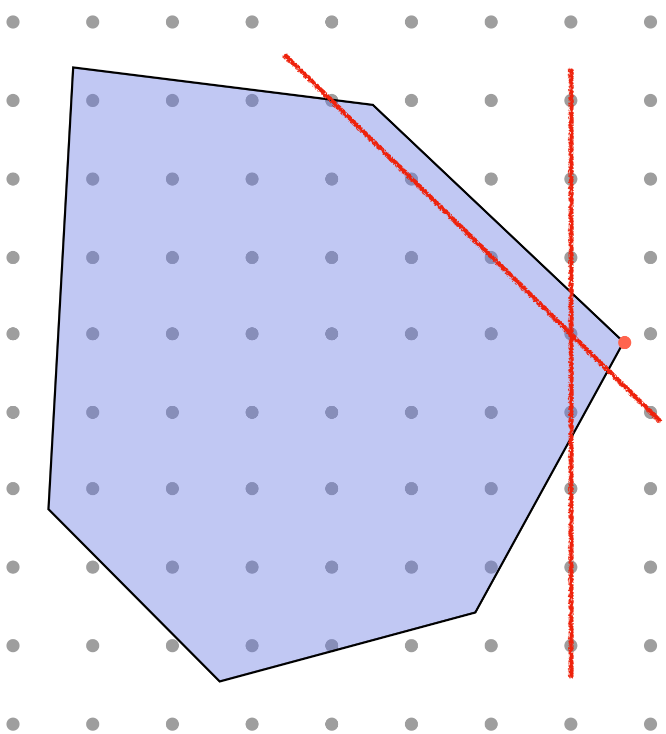
In terms of configuration, MILP solvers have a large amount of algorithmic parameters whose effective selection can lead to dramatic performance improvements.

Cutting Planes





Cutting Planes (cont.d)

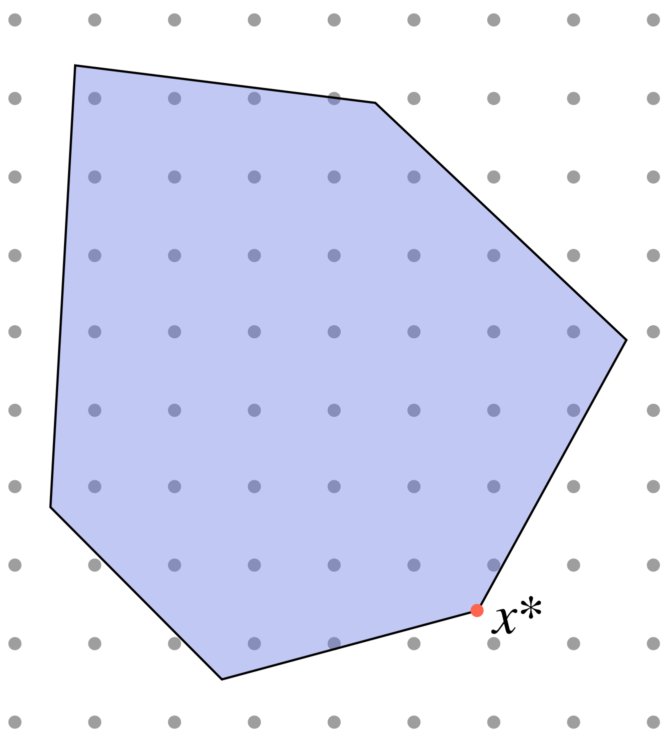


Many families of cutting planes are part of the arsenal of MILP solvers.

They differ in the way cuts are separated, which generally involves the aggregation of the original constraints into the so-called *base inequality* and a rounding step.

A good selection criterion is critical to improving the LP relaxation while avoiding an excessive number of cuts, which would slow down LP solving as well as lead to numerical instability.

Branching



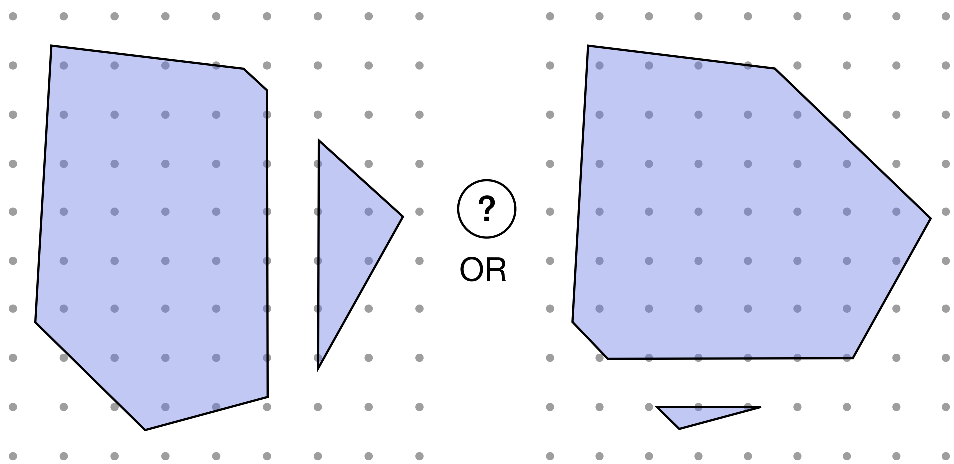
When the process of strengthening the LP relaxation (by either preprocessing or cuts) is no longer effective, the MILP (associated with any node) is split into sub-MILPs by branching.

This is a crucial step in MILP technology with dramatic effects on the effectiveness of the process.

In principle, any variable $x_j^* \in \mathbb{Z}^p$ whose value is not integer could be used to branch by imposing

$$x_j \leq \lfloor x_j^* \rfloor \vee x_j \geq \lfloor x_j^* \rfloor + 1$$

Branching: Variable Selection

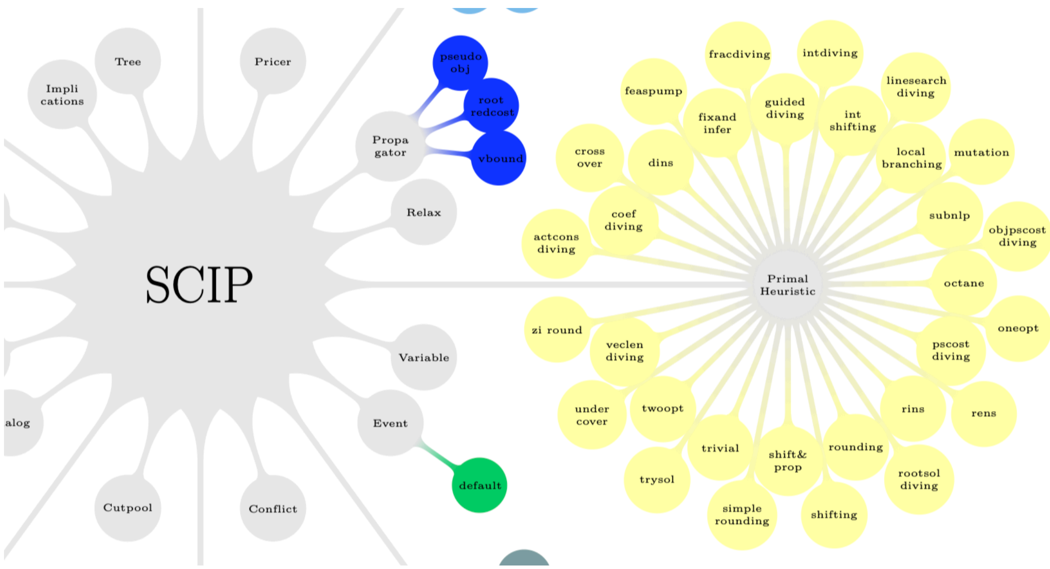


However, selecting an ineffective variable, i.e., one that does not produce any effect in the bound, leads to exponential-size B&B trees.

Currently, the best method we know is called *strong branching* and simulates branching on any variable, then selecting the most effective one.

Of course, this is too expensive, and clever simplified versions are used (reliability branching).

Primal Heuristics



Once the MILP solvers have started to be reliable, practitioners recognized the need of producing good feasible solutions early in the process.

Primal heuristics are those algorithms that are run during B&B to either producing feasible solutions or improving them.

ML-augmented MILP: The Opportunity

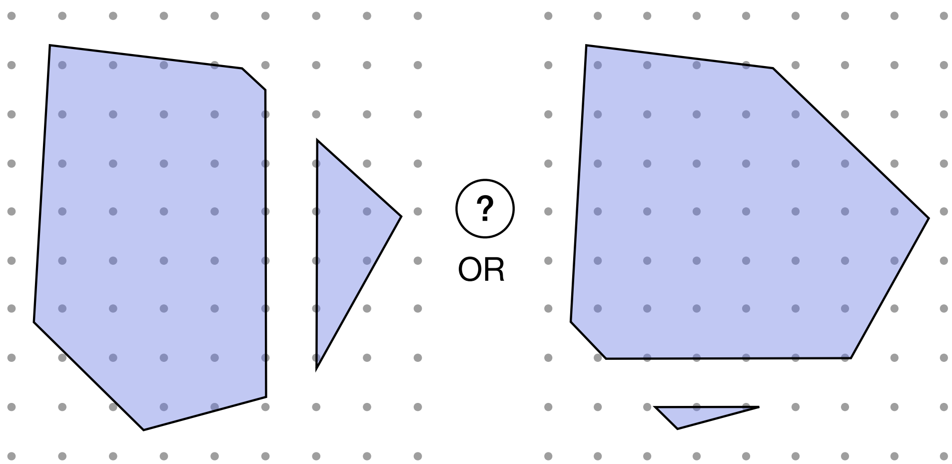
Too long

- Expert knowledge of how to make decisions
- Too expensive to compute
- Need for fast approximation

Too heuristic

- No idea which strategy will perform better
- Need a well performing policy
- Need to discover policies

Variable Selection (reprise)



Interestingly, variable selection falls in both categories:

- The best method we know for it (strong branching) is expensive (too slow) and, in any case,
- It is a heuristic, i.e., we do not have mathematical understanding of what is best (too heuristic).

Question

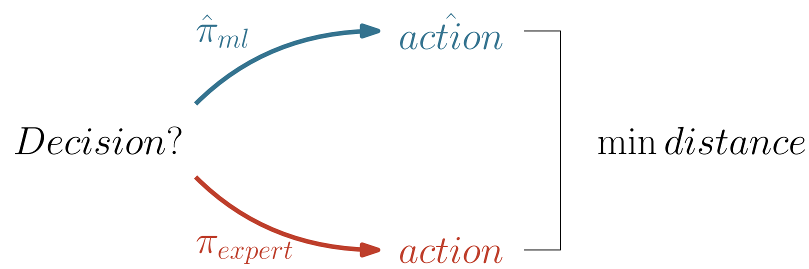
- Can **Machine Learning** methods as Imitation Learning, Reinforcement Learning and all the recent powerful techniques (e.g., Deep Learning) and architectures (e.g., Graph Neural Networks) help **Combinatorial Optimization** — particularly **MILP** — algorithms by dealing with the issues above ("too slow" and / or "too heuristic")?

Requirement

- We want to keep the **guarantees** provided by (exact) CO/MILP algorithms, namely,
  - feasibility, and
  - sometimes optimality.

Learning Methods

Demonstration



The idea is that there is an expert (an algorithm instead of a human like is common in ML) that we want to imitate. Thus, the data is labeled.

The distance of the picture is intended as the loss of not following accurately the expert label through the prediction.

Several ML models can be applied, with a significant use of Neural Networks (NNs).

Learning Methods (cont.d)

Experience

$$Decision? \xrightarrow{\hat{\pi}_{ml}} \hat{action} \xrightarrow{score} reward \quad \uparrow \quad \max return$$

By demonstration (or imitation), we are restricted to the quality of the expert that we cannot improve.

Learning by experience, often combined with a initial imitation phase, allows to potentially discover new policies.

It is generally more complex to train.

Learning Process

The learning process is itself one of (inexact) optimization, called training.

Its ingredients are

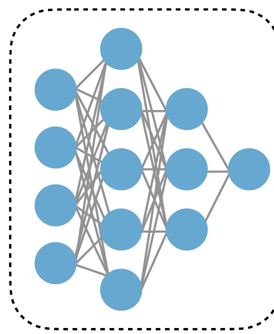
- Optimization algorithms
- Hyper-parameters
- Train, validation and test datasets
- Data collection
- Overfitting
- Online vs offline learning

ML-augmented MILP: Representation

MILP Representation

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}$$

?



The one above is the MILP standard definition.

It is not necessarily sufficient or fit for the ML task at hand, so the question is which characteristics of the MILP to represent and how.

MILP Representation (cont.d)

There are four desirable properties for such representation:

- Permutation invariance: permuting the order of the variables and/or constraints should leave the representation unchanged.
- Scale invariance: it is preferred to keep values within controlled ranges, which helps the learning process.
- Size invariance: the size of the representation should not depend on the size of the instance.
- Low computational cost: low cost of extracting, storing and processing data.

Representing Variables

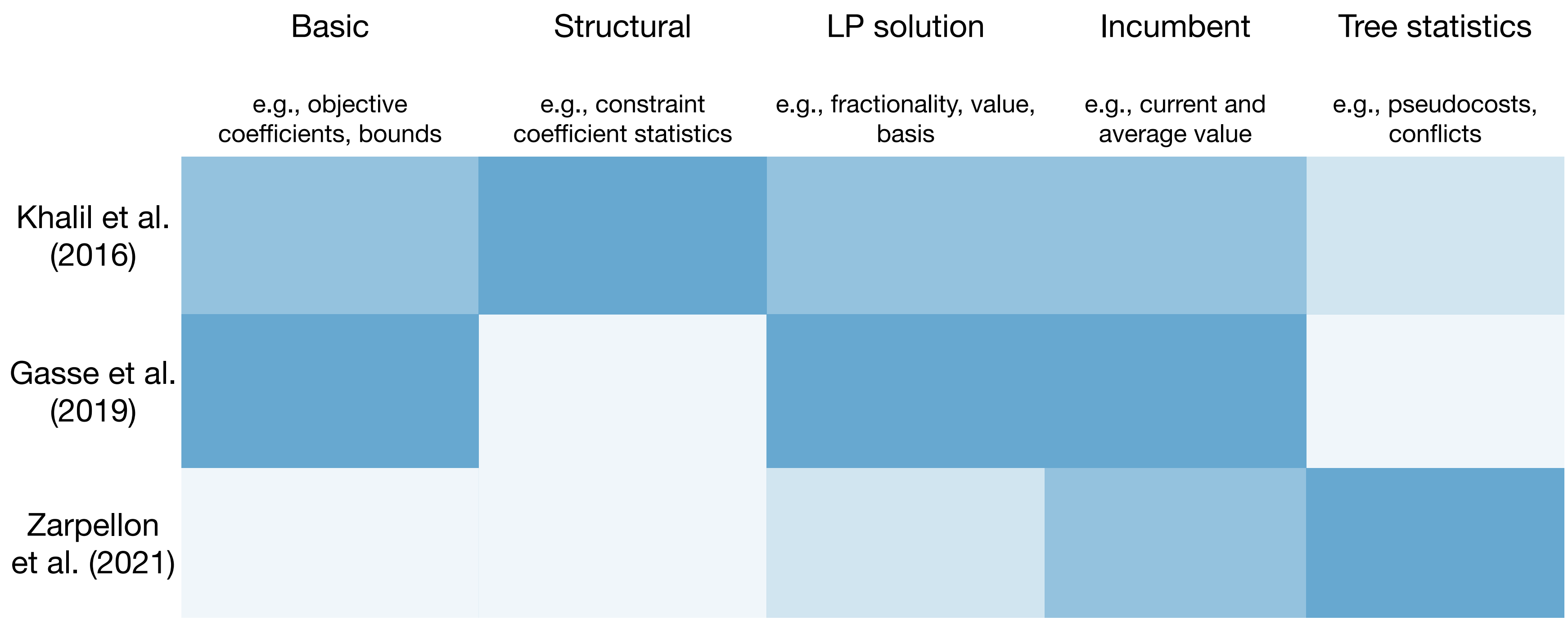
We recognized three main ways of representing variables. The tradeoff that needs to be found is associated with the desirable properties just discussed.

Khalil et al. (2016) use descriptors gathered in a vector of fixed size. Those descriptors aggregate information whose length would otherwise depend on the problem size.

Gasse et al. (2019) use a bipartite graph representation whose advantage is associated with the mapping into the so-called Graph Neural Networks (GNNs).

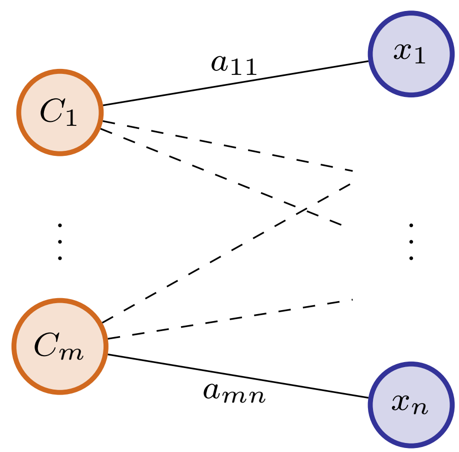
Zarpellon et al. (2021) take a different approach, stressing the importance of historical information collected in the B&B tree (during execution).

Representing Variables (cont.d)



Bipartite Graph Representation

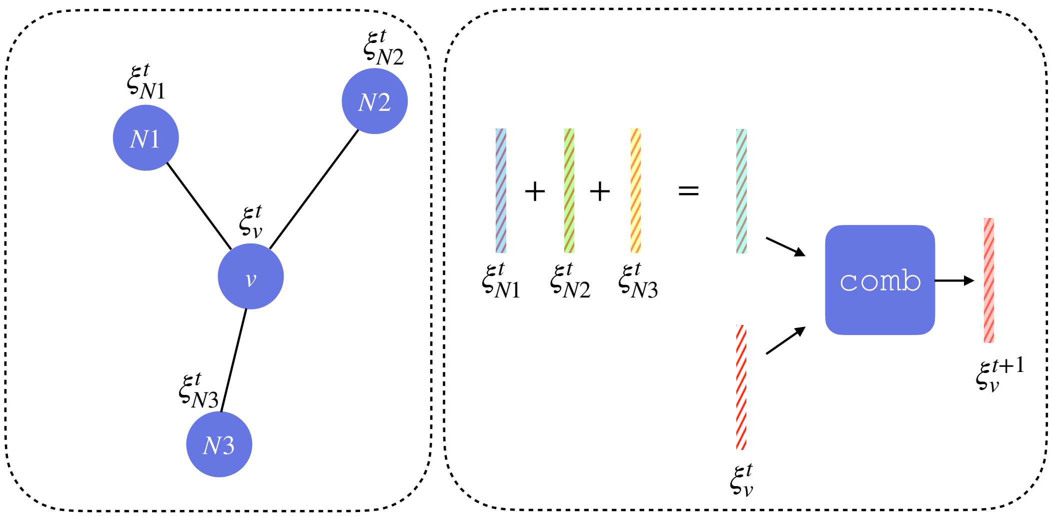
$$
\begin{array}{lccccccc}
\min_{\boldsymbol{x}} & c_1x_1 & + & \cdots & + & c_nx_n & & \\
 & a_{11}x_1 & + & \cdots & + & a_{1n}x_n & \leq & b_1 \\
 & & & \vdots & & & & \\
 & a_{m1}x_1 & + & \cdots & + & a_{mn}x_n & \leq & b_m
\end{array}
$$



From (Bipartite) Graphs to GNNs

- A (d-dimensional) graph embedding ξ is a function that takes in a graph $G = (V, E)$ and a node $v \in V$ and returns an element $\xi(G, v) \in \mathbb{R}^d$.

- A Graph Neural Network is a function that takes as input a graph $G = (V, E)$ and an initial embedding ξ^0 and defines a recursive embedding ξ^t over the vertices of G.

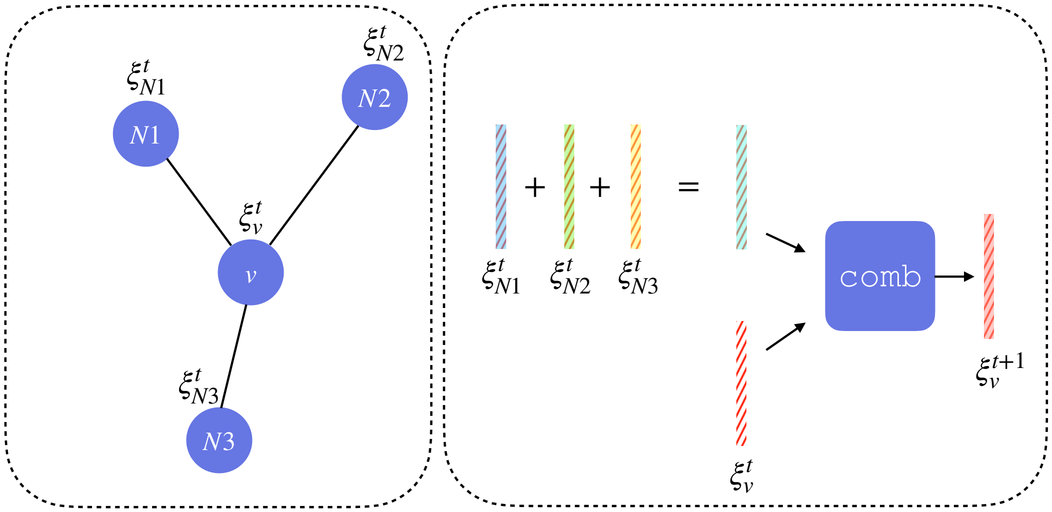


From (Bipartite) Graphs to GNNs (cont.d)

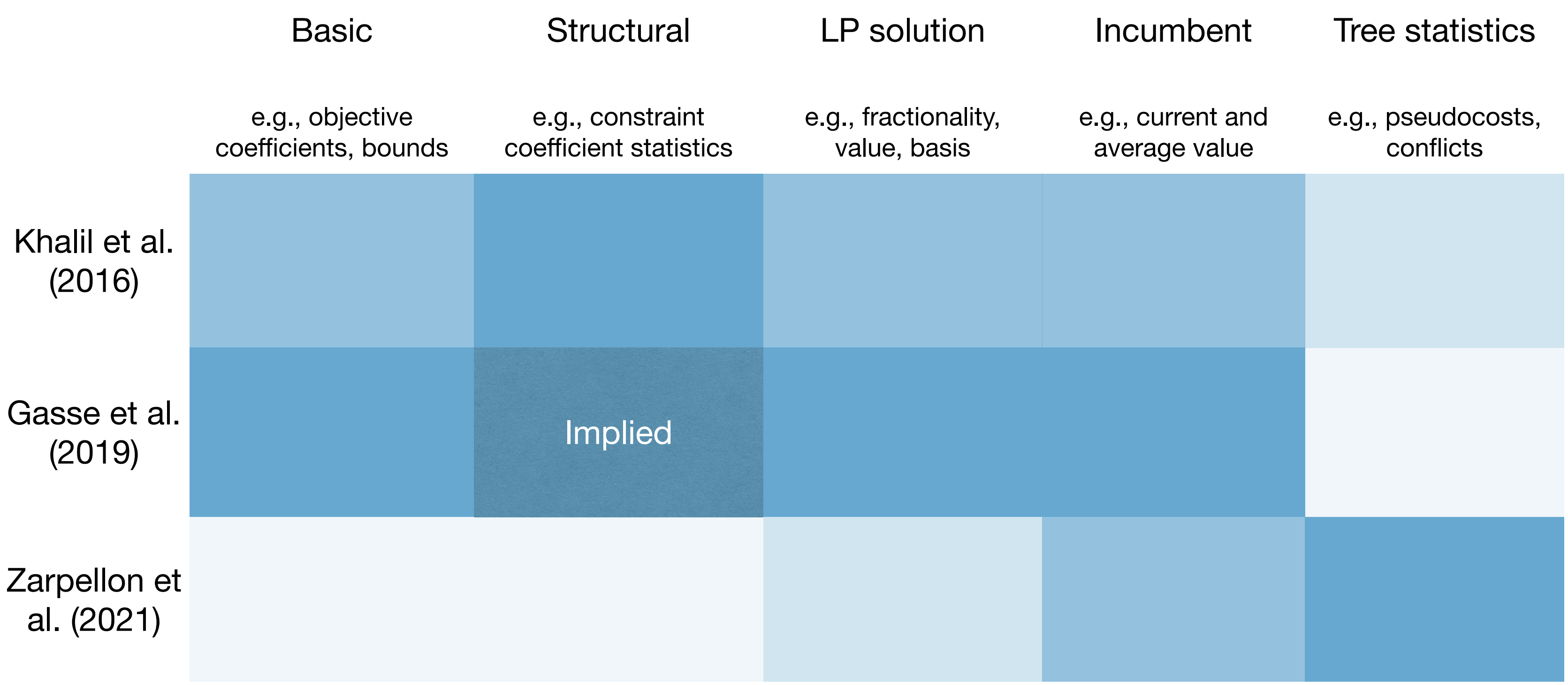
The recursion is obtained by

- First, *aggregating* the embeddings of the neighbors of a node, and
- Second, *combining* the aggregated embeddings of the neighbors with the original embedding.

The combination can be obtained, for example, through a feed-forward NN and one iteration of the process is called *message passing*.



Representing Variables (reprise)



ML-augmented MILP: Learning Tasks

Learning Tasks: Primal Heuristics

A number of methodologies have been proposed for this purpose.

Conceptually, they can be split into three main approaches:

(a) *guiding a heuristic search* with a starting predicted solution,

(b) *solution improvement* via a learned neighborhood selection criterion, and

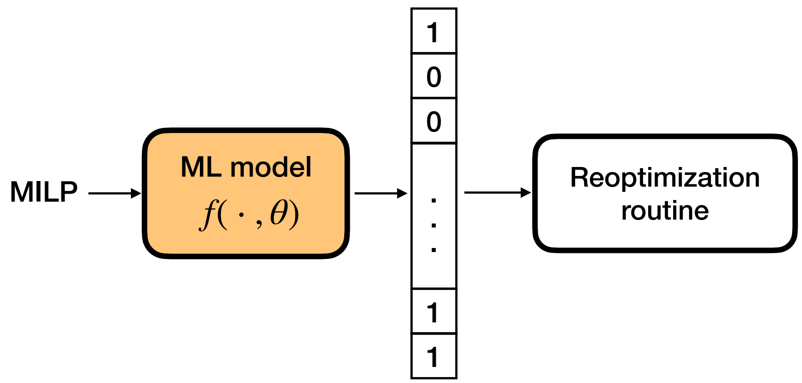
(c) *learning a schedule* to pre-existing heuristic routines.

For (a) and (b), the important concept is that of Large Neighborhood Search.

The idea is to optimize an auxiliary MILP of smaller size, constructed by reducing the feasible region of the original MILP.

Typically done by fixing the value of some of the variables and optimizing the rest.

Primal Heuristics

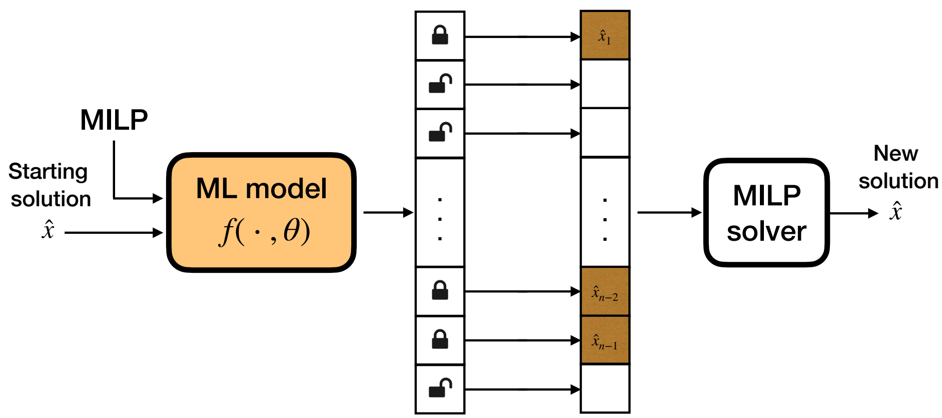


Guiding a heuristic search with a starting predicted solution:

- The goal is to produce a (partial) assignment of the binary variables in a binary or mixed binary MILP, that can then be used to guide the search.
- Often, this is obtained by starting from a set of collected solutions.

Examples in Ding et al. (2020), Nair et al. (2020) and Khalil et al. (2022).

Primal Heuristics (cont.d)



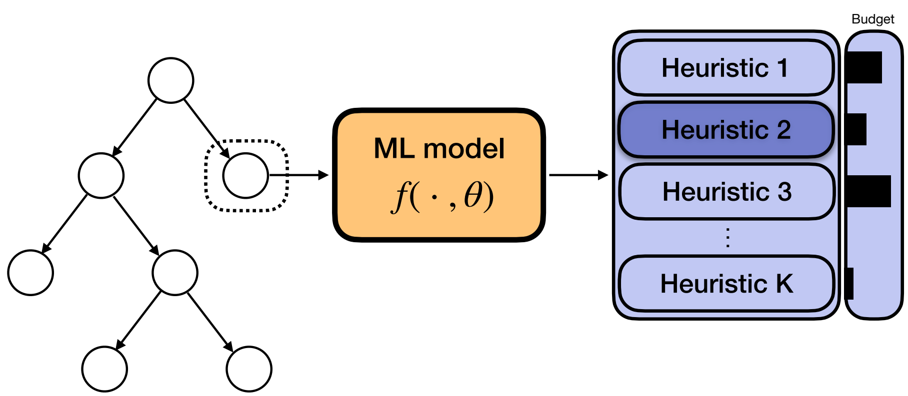
Neighborhood selection:

- which and/or
- how many variables to unfix and re-optimize.

The goal is to identify substructures of the problem that can be used to decompose it into smaller, more manageable sub-problems.

Examples in Song et al. (2020), Wu et al. (2021) and Liu et al. (2022).

Primal Heuristics (cont.d)

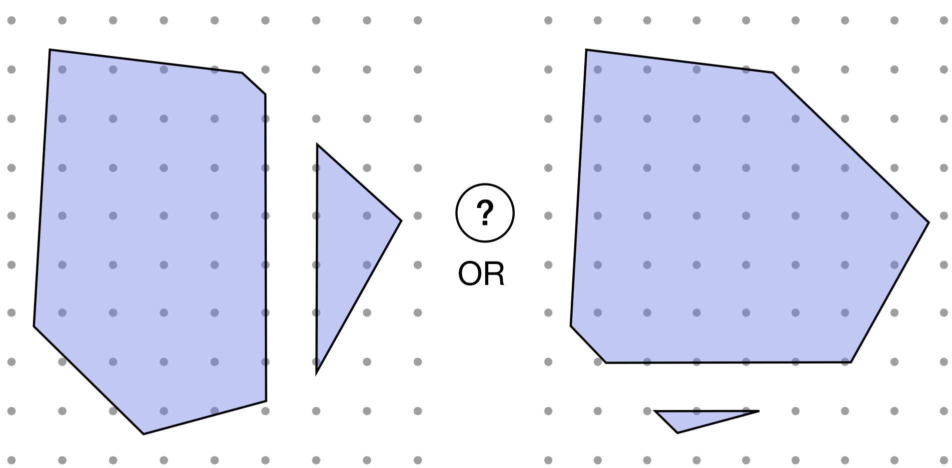


Scheduling of primal heuristics:

- which heuristics to run and/or
- for how long.

Examples in Hendel et al. (2019), Hendel (2022) and Chmiela et al. (2021, 2023).

Learning Tasks: Variable Selection



Learning to branch has been by far the most active area of integration of ML into MILP.

Initially, most of the effort has been concentrated on approximating strong branching, i.e., using strong branching as the expert to imitate.

Variable Selection: Learning Strong Branching

	ML paradigm	Online/offline	Model
Alvarez et al. (2017)	SL, regression	Offline	ExtraTrees
Alvarez et al. (2016)	SL, regression	Online	Linear
Khalil et al. (2016)	SL, ranking	Online	Linear
Gasse et al. (2019)	SL, imitation	Offline	GNN
Etheve et al. (2020)	RL, Q-learning	Offline	NN
Scavuzzo et al. (2022)	RL, policy gradient	Offline	GNN
Zarpellon et al. (2021)	SL, imitation	Offline	NN
Lin et al. (2022)	SL, imitation	Offline	Transformer

The table gives a summary of different learning approaches for branching.

We use the acronym SL for supervised (demonstration) learning and RL for reinforcement learning (experience).

Gasse et al. (2019) propose training a GNN to imitate strong branching via behavioral cloning.

Essentially, this means that the actual variable scores are disregard and the focus is on learning relative magnitudes among them.

Through this approach the authors were able to outperform reliability branching, marking a breakthrough in the learning to branch literature.

Variable Selection: Towards a General Branching Rule

The SL approaches for strong branching specialize to combinatorial structures, i.e., they are trained on specific distributions (if not on single instances as for online learning).

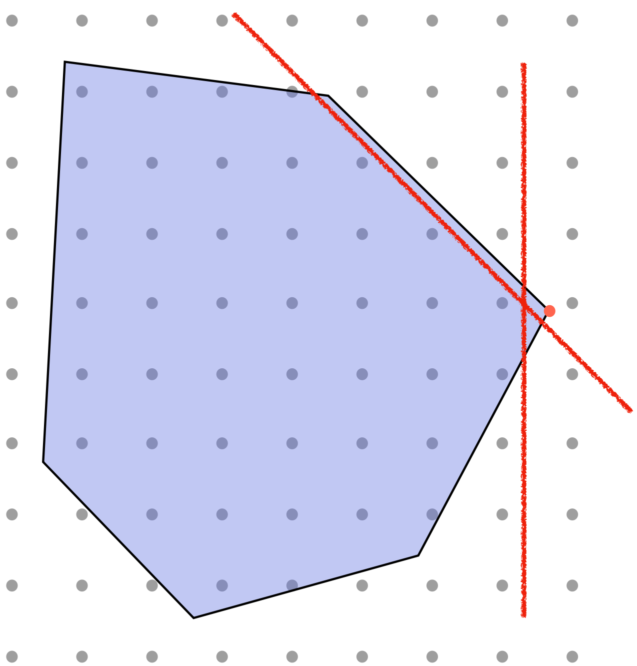
They fail to generalize.

The approaches by Zarpellon et al. (2021) and Lin et al. (2022) make a significant step in overcoming this limitation by doing SL but using information of the B&B tree evolution.

The subsequent step is to go beyond demonstration and learning by experience.

The RL attempts in Etheve et al. (2020) and Scavuzzo et al. (2022) follow this path.

Learning Tasks: Cut Selection



As anticipated, cut selection is a fundamental MILP component.

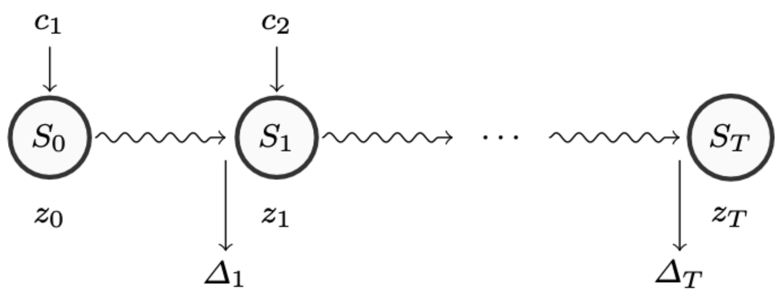
Several metrics proposed for the purpose of scoring cuts.

For example, the objective parallelism, measured as the cosine of the angle between the objective function and the cut, or the cutoff distance, measured as the distance between the cut and the LP-relaxation solution.

More recently, the question of cut selection has been addressed with ML-driven predictions.

Deza and Khalil (2023) recently surveyed the topic in details.

Single-cut Selection



Action: choose a cut c_k from the cut pool $\mathcal{C}_k$
Transition: apply cut, resolve LP
Objective: $\Delta_k = z_k - z_{k-1}$

The idea is to frame the cut selection problem (in the simplified version of one cut at a time) as a Markov Decision Process.

Paulus et al. (2022) use *imitation learning* and essentially their expert is the extension of the strong branching idea to cuts: the effect of each cut is simulated by solving an LP.

Tang et al. (2020) use instead *RL*, which allows to potentially go beyond the greedy look-ahead of one step at the price of more complex convergence.

Learning Tasks: Preprocessing / Configuration

As anticipated, the impact of good configuration or preprocessing has an extremely high potential because MILP solvers are highly configurable software tools.

Indeed, this is the area in which the success stories have already been incorporated in the commercial solvers.

The first of these success stories has been Bonami et al. (2018, 2022), where the authors prescribe for each mixed integer quadratic programming instance if the quadratic objective function should be linearized or not.

The resulting ML predictor runs in CPLEX default since version 12.10.

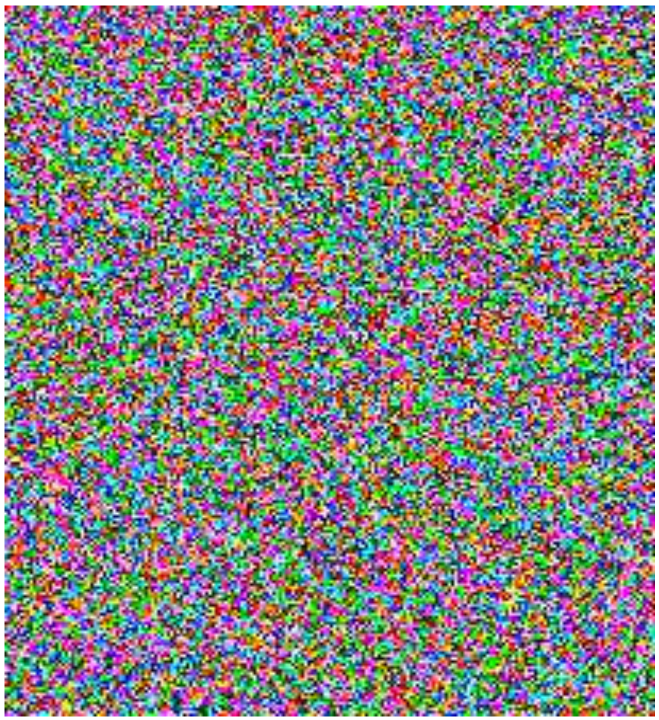
Preprocessing / Configuration: Examples

	Component	Question	Options
Kruber et al. (2017)	General	Should the Dantzig-Wolfe decomposition be used?	yes / no
Hendel et al. (2019)	LP solver	Which simplex pricing rule to use?	devex / steepest / quick-start steepest
Berthold and Hendel (2021)	Presolve	Which scaling method to apply?	Standard / Curtis-Reid
Berthold et al. (2022)	Cutting	Should cuts be applied outside the root node?	yes / no
Turner et al. (2023)	Cutting	How should cut scores be weighted?	$\mu \in \mathbb{R}^4_{\geq 0}$

Notably, the method presented in Berthold and Hendel (2021) is used by default in FICO Xpress since version 8.9 to decide scaling.

ML-augmented MILP: Perspectives and Challenges

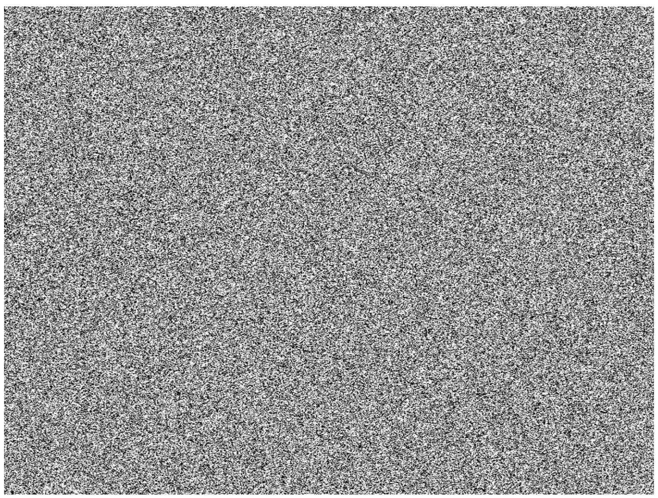
Generalization: Random Images



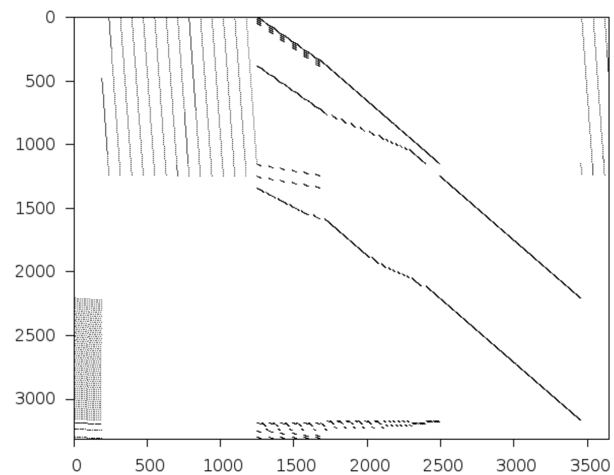
Random iid pixels

Random face (GAN)
thispersondoesnotexist.com

Generalization: Random Instances



Random iid coefficients



a1c1s1 from MipLib 2017

A Business Perspective

- Many businesses care about solving **similar** problems **repeatedly**
- Solvers do not make any use of this aspect
- Power systems and market Xavier et al. (2019)
  - Schedule 3.8 kWh ($400 billion) market annually in the US
  - Solved multiple times a day
  - 12x speed up combining ML and MILP

Software: An Example

https://www.scipopt.org/

- One of the fastest non-commercial solvers for MIP
- ~800k lines of code; many advanced features and extensions
- **Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers**

Antoine Prouvost Mila, Polytechnique Montréal
Justin Dumouchelle Polytechnique Montréal
Lara Scavuzzo Technische Universiteit Delft
Maxime Gasse Mila, Polytechnique Montréal
Didier Chételat Polytechnique Montréal
Andrea Lodi Mila, Polytechnique Montréal

ECOLE Extensible Combinatorial Optimization Learning Environments

https://www.ecole.ai/

A Challenge and an Opportunity

Ch: The use of NNs and especially GNNs has proven effective for the considered learning tasks. However, computation on NNs is especially effective by using GPUs, while classical MILP computation is on CPUs.

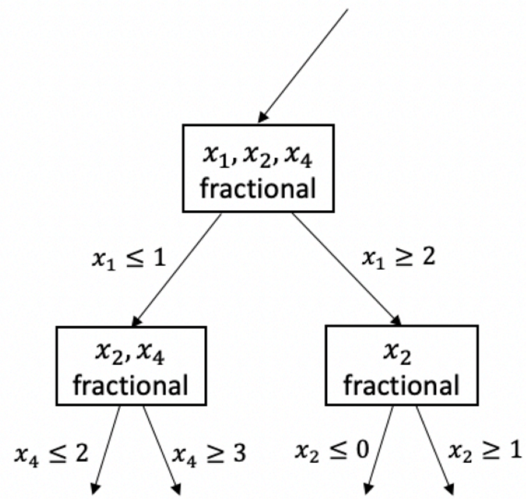
Op: The use of restart mechanisms in MILP is increasing and the chance of using the information collected by running the solvers for a limited amount of time seems to favor the use of ML even more.

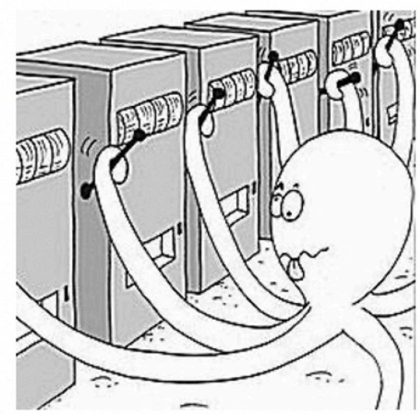
ML-augmented MILP: Summary

Summary

Three Tasks

Finding feasible solutions of MILP





Primal Task

Dual Task

Configuration Task

NeurIPS 2021 competition: ML4CO

Machine Learning for Combinatorial Optimization
COMPETITION 2021

Mixed Integer Linear Programming (MILP)

$$\begin{aligned} \arg\min_{\mathbf{x}} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}^\top \mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned}$$

Machine Learning for Combinatorial Optimization (ML4CO)
NeurIPS 2021 competition (Submission deadline: Oct 31 2021)
https://www.ecole.ai/2021/ml4co-competition/