Mixed-Integer Programming: 75 years of history and the Artificial Intelligence challenge

Andrea Lodi
andrea.lodi@cornell.edu

CERMICS Colloquium

Paris, February 9, 2024 @

École des Ponts, ParisTech

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CANADA
EXCELLENCE
RESEARCH
CHAIR

DATA SCIENCE
FOR REAL-TIME
DECISION-MAKING

CORNELL
TECH

HOME OF THE **JACOBS**
TECHNION-CORNELL
INSTITUTE

Outline

The talk is developed in two parts:

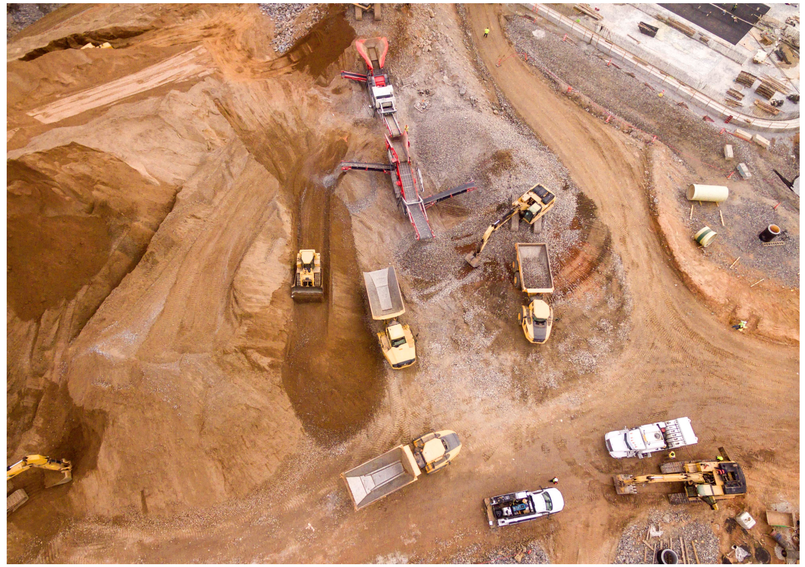
- Part 1: Mixed-Integer Linear Programming (MILP), a successful story:
  - 75 years of mathematics, algorithms and software development.

- Part 2: MILP at the time of Artificial Intelligence:
  - Methodological directions in which the use of Machine Learning is changing and will change (?) MILP.

We will finish with some perspectives.

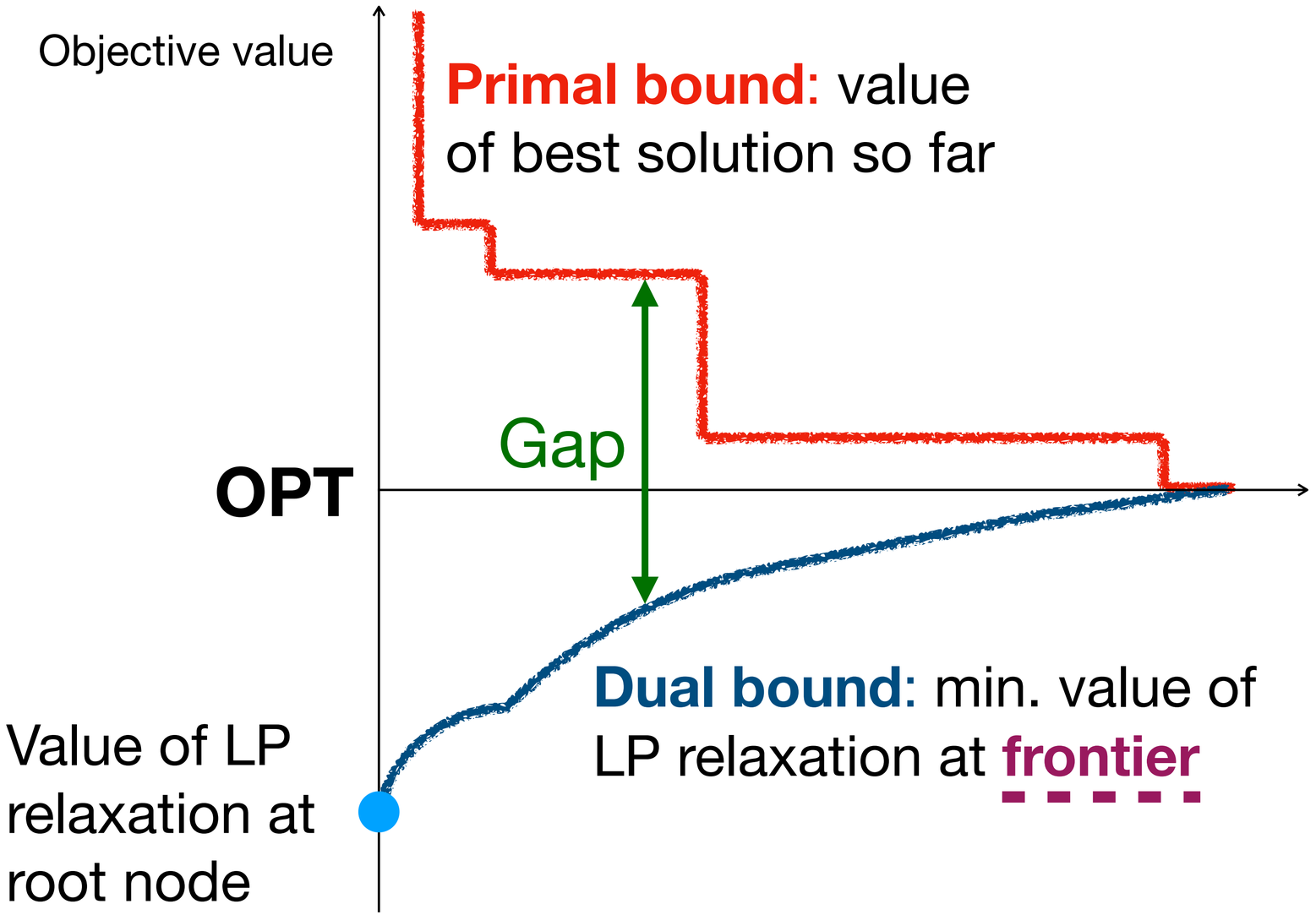
Mixed-Integer Linear Programming: Where?

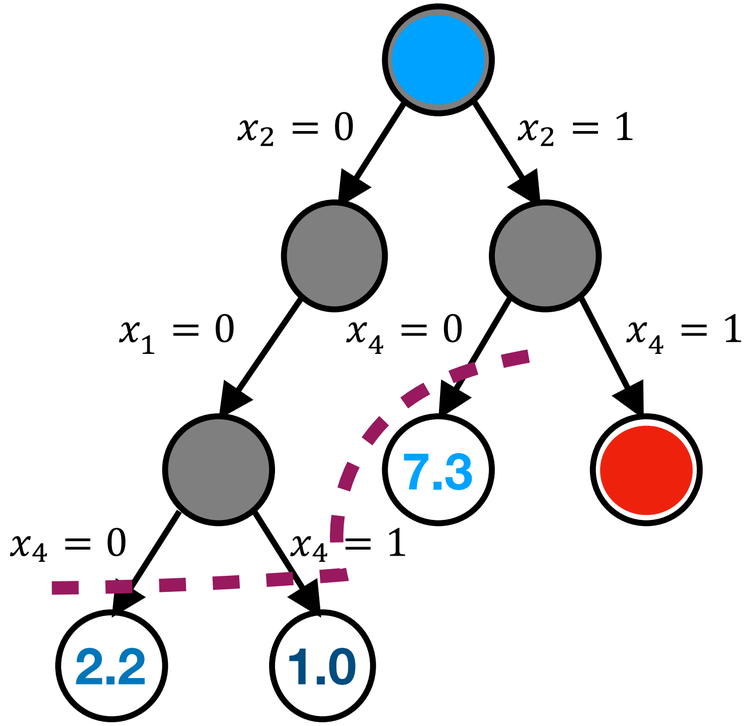

U A S C
HYUNDAI
COSCO
MAERSK
OOCL
CMA CGM
Hapag-Lloyd
CSAV



Mixed-Integer Linear Programming: How?

$$\min_x c^T x \textbf{ s.t. } Ax \leq b, x \in \{0,1\}^n$$





Search tree nodes

Slide courtesy of E. Khalil

MILP evolution, early days

- Despite quite some work on basically all aspects of IP and in particular on cutting planes, the early days of general-purpose MILP solvers were mainly devoted to develop fast and reliable LP solvers used within good branch-and-bound schemes.

- Remarkable exceptions are:
  - 1983 Crowder, Johnson & Padberg: PIPX, pure 0/1 MILPs
  - 1987 Van Roy & Wolsey: MPSARX, mixed 0/1 MILPs

- When do the early days end?
  Or equivalently, when does the current generation of MILP solvers appear?

- It looks like a major (crucial) step to get to nowadays MILP solvers has been the ultimate proof that cutting plane generation in conjunction with branching could work in general, i.e., after the success in the TSP context:
  - 1994 Balas, Ceria & Cornuéjols: lift-and-project
  - 1996 Balas, Ceria, Cornuéjols & Natraj: gomory cuts revisited

MILP evolution, an interesting experiment

- Bob Bixby & Tobias Achterberg performed the following interesting experiment comparing Cplex versions from Cplex 1.2 (the first one with MILP capability) up to Cplex 11.0.
- 1,734 MILP instances, time limit of 30,000 CPU seconds, computing times as geometric means normalized wrt Cplex 11.0 (equivalent if within 10%).

Cplex versions	year	better	worse	time
11.0	2007	0	0	1.00
10.0	2005	201	650	1.91
9.0	2003	142	793	2.73
8.0	2002	117	856	3.56
7.1	2001	63	930	4.59
6.5	1999	71	997	7.47
6.0	1998	55	1060	21.30
5.0	1997	45	1069	22.57
4.0	1995	37	1089	26.29
3.0	1994	34	1107	34.63
2.1	1993	13	1137	56.16
1.2	1991	17	1132	67.90

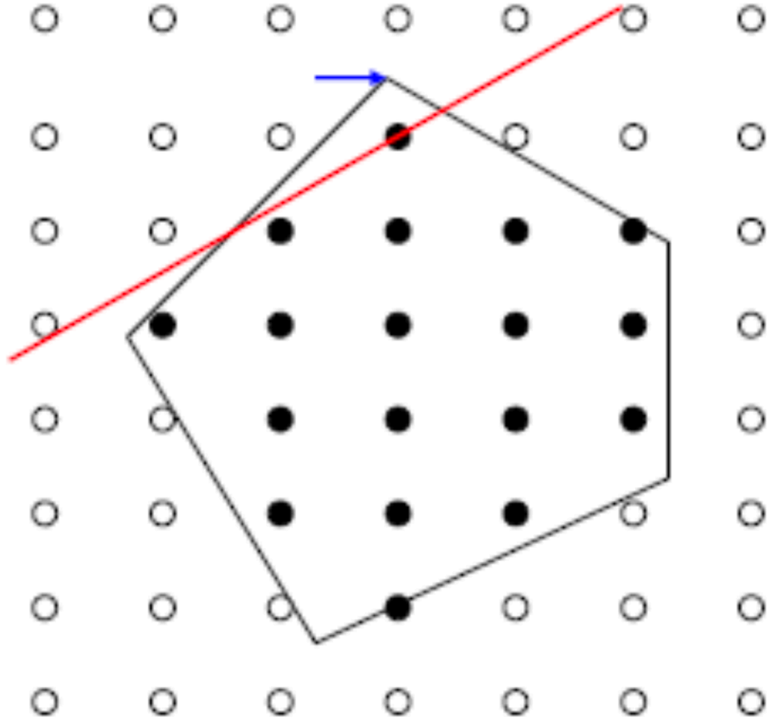


Figure 1: Strengthening the LP relaxation by cutting planes.

MILP evolution, nowadays key features

- The current generation of MILP solvers incorporates key ideas developed continuously during the first 50 years of Integer Programming (often in the context of location problems):
  - Preprocessing:
    probing, bound strengthening, propagation
  - Cutting plane generation:
    Gomory Mixed Integer cuts, Mixed Integer Rounding, cover cuts, flow covers, . . .
  - Sophisticated branching strategies:
    strong branching, pseudo-cost branching, diving and hybrids
  - Primal heuristics:
    rounding heuristics (from easy to complex), local search, . . .
- Moreover, MILP computation has reached such an effective and stable quality to allow the solution of sub-MILPs in the algorithmic process, the MIPping approach [Fischetti & Lodi 2002].

MILP building blocks: preprocessing / presolving

- In the preprocessing phase a MILP solver tries to detect certain changes in the input that will probably lead to a better performance of the solution process.
- Generally without “changing” the set of optimal solutions, a notable exception being symmetry breaking reductions.
- There are two different venues for preprocessing.
  1. Model preprocessing:
     MILP models often contain some “garbage”, i.e., redundant or weak information slowing down the solution process by forcing the solver to perform useless operations. Thus, modern MILP solvers have the capability of cleaning up and strengthen a model so as to create a presolved instance on which the MILP technology is then applied.
  2. Algorithmic preprocessing:
     More sophisticated presolve mechanisms are also able to discover important implications and sub-structures that might be of fundamental importance later on in the computation for both branching purposes and cutting plane generation.
- Finally, the lower and upper bounds on the objective function and the solution of LPs can be used to perform even stronger reduction (known as probing) with the aim of fixing variables.

MILP building blocks: cutting planes

- From what has been discussed before, it is clear that cutting planes are a crucial components of MILP solvers.

- Given the MILP (1), we are mainly interested in the two sets

$$S := \{Ax \leq b, x \geq 0,\ x_j \in \mathbb{Z}, \forall j \in I\} \tag{3}$$

and

$$P := \{Ax \leq b, x \geq 0\}. \tag{4}$$

- Generality: We are interested in general-purpose cutting planes, those that can be derived without assuming any special structure for the polyhedron P.
- Validity: An inequality $\alpha x \leq \beta$ is said to be valid for S if it is satisfied by all $x \in S$.
- Obtaining a valid inequality for a continuous set: Given P, any valid inequality for it is obtained as $uAx \leq \beta$, where $u \in \mathbb{R}^m_+$ and $\beta \geq ub$. (Farkas Lemma)

MILP building blocks: cutting planes (cont.d)

- Separation:
  Given a family of valid inequalities $\mathcal{F}$ and a solution $x^* \in P \setminus S$, the **Separation problem for** $\mathcal{F}$ is defined as

  Find an inequality $\alpha x \leq \beta$ of $\mathcal{F}$ valid for S such that $\alpha x^* > \beta$ or show that none exists.

- Iterative strengthening
  1. solve the problem $\{\max c^T x \ : \ x \in P\}$ and get x^*
  2. if $x^* \in S$ then **STOP**
  3. solve the separation problem, add $\alpha x \leq \beta$ to P and go to 1.

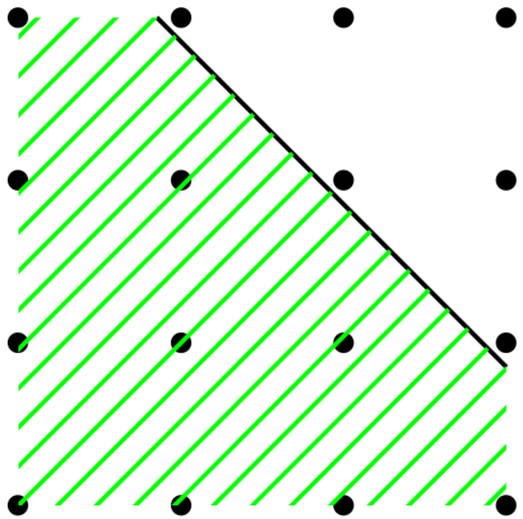
- (Almost) all cutting plane classes that belong to the arsenal of nowadays MILP solvers belong to the family of split cuts, i.e., they are separated by exploiting in some way (from easy to complex) a disjunction on the integer variables.

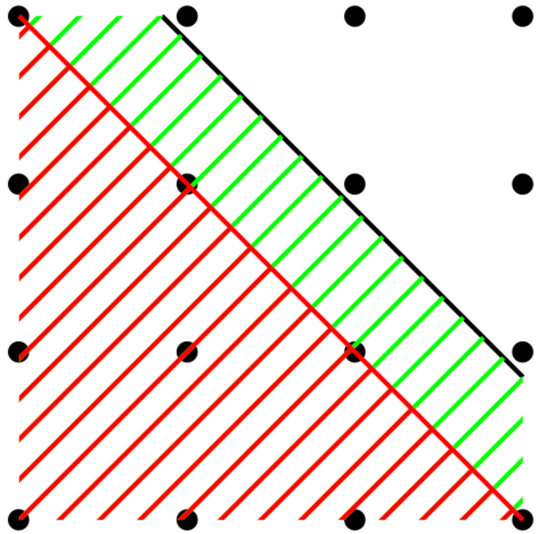
MILP building blocks: cutting planes (cont.d)

- A basic rounding argument:
  If $x \in \mathbb{Z}$ and $x \leq f$ $f \notin \mathbb{Z}$, then $x \leq \lfloor f \rfloor$.

- Using rounding:
  Consider an inequality $\alpha x \leq \beta$ such that $\alpha_j \in \mathbb{Z},\ j = 1, \ldots, n$ in the pure integer case $I = \{1, \ldots, n\}$. If $\alpha x \leq \beta$, then $\alpha x \leq \lfloor \beta \rfloor$ is valid as well.

- Example:
  $x \in \mathbb{Z}^2$ such that $x_1 + x_2 \leq 1.9$ $\Rightarrow\ x_1 + x_2 \leq \lfloor 1.9 \rfloor = 1$





MILP building blocks: cutting planes (cont.d)

- Theorem [Gomory 1958, Chvátal 1973]:

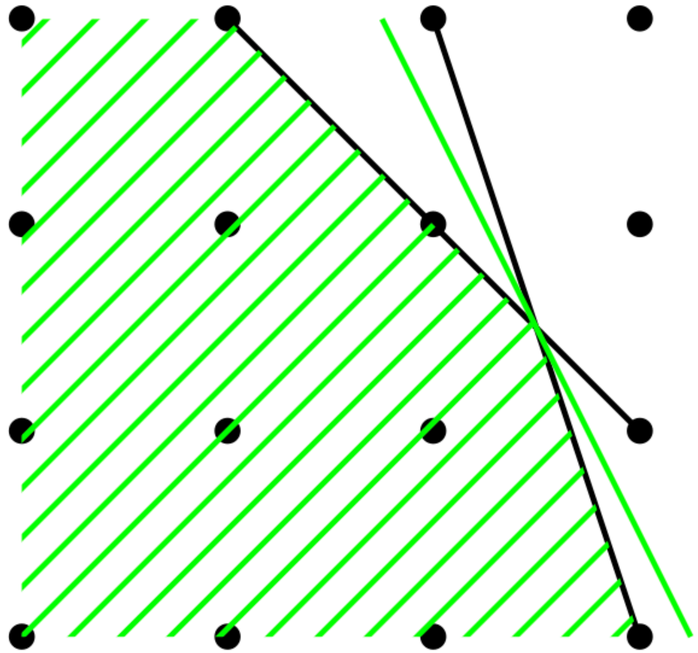
  If $x \in \mathbb{Z}^n$ satisfies $Ax \leq b$, then the inequality $uAx \leq \lfloor ub \rfloor$ is valid for S for all $u \geq 0$ such that $uA \in \mathbb{Z}^m$.

Example:

Consider the polyhedron given by the two inequalities

$$x_1 + x_2 \leq 2$$
$$3x_1 + x_2 \leq 5$$

Let $u_1 = u_2 = \frac{1}{2}$, $\Rightarrow 2x_1 + x_2 \leq 3.5$



MILP building blocks: cutting planes (cont.d)

- Theorem [Gomory 1958, Chvátal 1973]:

  If $x \in \mathbb{Z}^n$ satisfies $Ax \leq b$, then the inequality $uAx \leq \lfloor ub \rfloor$ is valid for S for all $u \geq 0$ such that $uA \in \mathbb{Z}^m$.

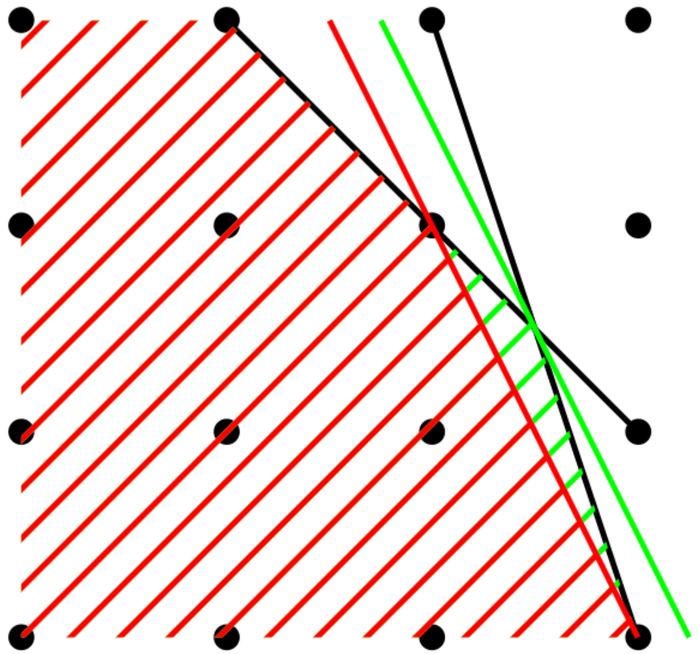
Example:

Consider the polyhedron given by the two inequalities

$$x_1 + x_2 \leq 2$$
$$3x_1 + x_2 \leq 5$$

Let $u_1 = u_2 = \frac{1}{2}$, $\Rightarrow 2x_1 + x_2 \leq 3.5$

and rounding we obtain $2x_1 + x_2 \leq 3$



MILP building blocks: branching

- In its basic version the branch-and-bound algorithm [Land & Doig 1960] iteratively partitions the solution space into sub-MILPs (the children nodes) that have the same theoretical complexity of the originating MILP (the father node, or the root node if it is the initial MILP).

- Usually, for MILP solvers the branching creates two children by using the rounding of the solution of the LP relaxation value of a fractional variable, say x_j, constrained to be integral

$$x_j \leq \lfloor x_j^* \rfloor \quad \textbf{OR} \quad x_j \geq \lfloor x_j^* \rfloor + 1. \tag{5}$$

- On the two children, left (or "down") branch and right (or "up") branch, the integrality requirement on the variables $x_j, \forall j \in I$ is relaxed and the LP relaxation is solved (again).

- Sub-MILPs become smaller and smaller due to the partition mechanism (basically some of the decisions are taken) and eventually the LP relaxation is directly integral (or infeasible).

- In addition, the LP relaxation is solved at every node to decide if the node itself is worthwhile to be further partitioned: if the LP relaxation value is already not better (bigger) than the incumbent, the node can be safely fathomed.

MILP building blocks: branching (cont.d)

- Of course, the basic idea of the splitting a node does not require that branching is performed as in (5): i.e., more than two children could be created, and one can branch on more general hyperplanes, or, in general, on any other disjunctive condition.

- The reason why variable branching (5) is the most popular (and this situation is not likely to change anytime soon, at least for MILP solvers) is that it takes full advantage of the ability of the Simplex algorithm to recompute the optimal solution of the LP relaxation if only variable bounds (possibly one) have changed.

- In fact, on average, for a single LP solution Interior Point algorithms performs better than the Simplex algorithm [Rothberg 2010], which is in turn (currently) unbeatable in the iterative context.

- The described branch-and-bound framework requires two independent and important decisions at any step: Node and Variable selection.

MILP building blocks: branching (cont.d)

1. Node selection:
   This is very classical: one extreme is the so called best-bound first strategy in which one always considers the most promising node, i.e., the one with the highest LP value, while the other extreme is depth first where one goes deeper and deeper in the tree and starts backtracking only once a node is fathomed.
   All other techniques, more or less sophisticated, are basically hybrids around these two ideas.

2. Variable selection:
   The variable selection problem is the one of deciding how to partition the current node, i.e., on which variable to branch on in order to create the two children.

   For a long time, a classical choice has been branching on the most fractional variable, i.e., in the 0-1 case the closest to 0.5.

   That rule has been computationally shown to be worse than a complete random choice [Achterberg et al. 2005]. However, it is of course very easy to evaluate.

   In order to devise stronger criteria one has to do much more work.

MILP building blocks: branching (cont.d)

2. Variable selection (cont.d):
   The extreme is the so called strong branching technique [Applegate et al. 2007; Linderoth & Savelsbergh 1999].

   In its full version, at any node one has to simulate branch on each candidate fractional variable and select the one on which the improvement (decrease) in the bound on the left branch times the one on the right branch is the maximum.

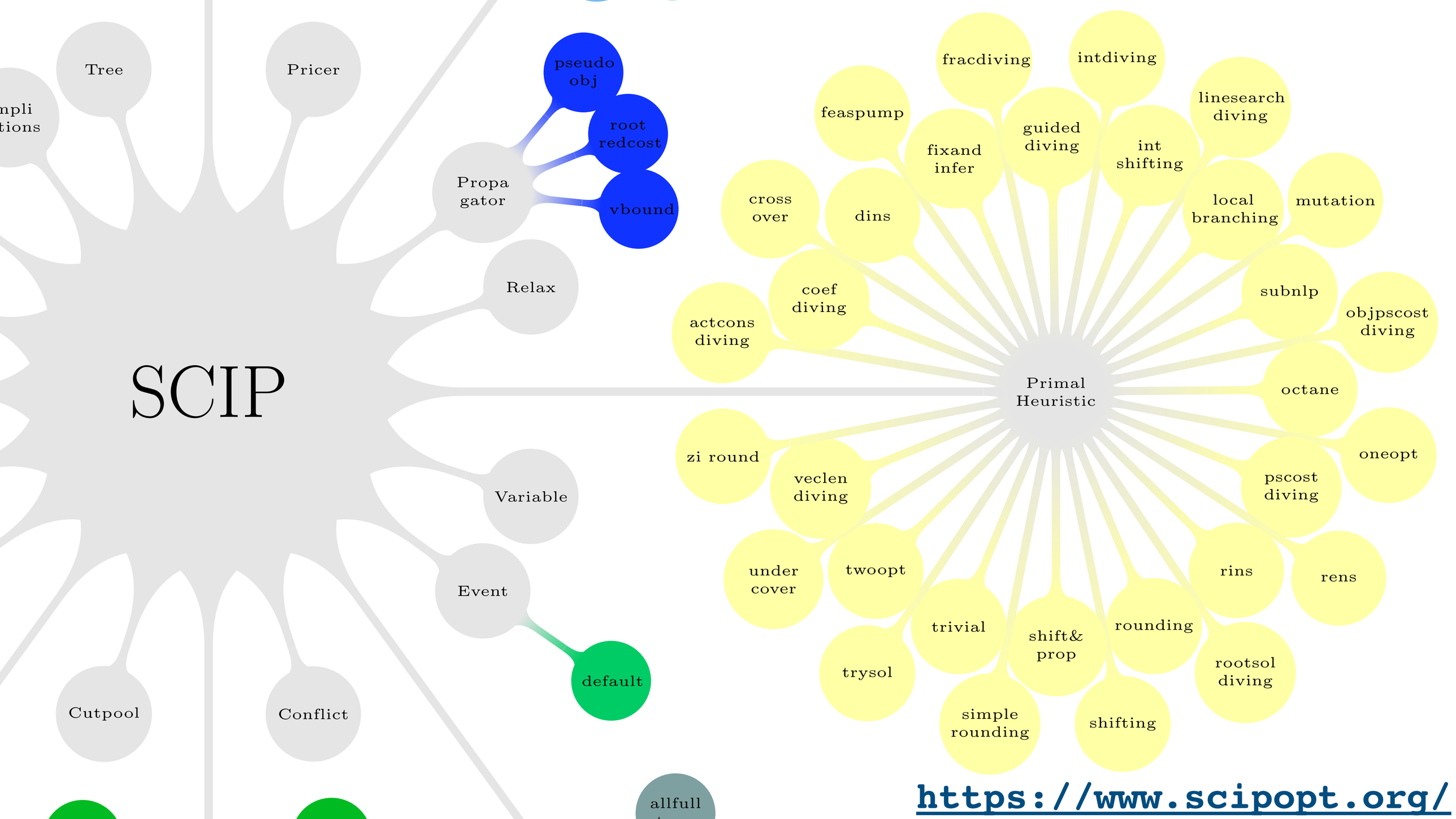
   Of course, this is in general computationally unpractical (discussed later) but all MILP solvers implement lighter versions of this scheme.

   Another sophisticated technique is pseudocost branching [Benichouet al. 1971] that keeps a history of the success (in terms of the change in the LP relaxation value) of the branchings already performed on each variable as an indication of the quality of the variable itself.

   The most recent effective and sophisticated method, called reliability branching [Achterberg et al. 2005], integrates strong and pseudocost branchings by defining a reliability threshold, i.e., a level below which the information of the pseudocosts is not considered accurate enough and some strong branching is performed.

MILP building blocks: primal heuristics

- The last 20 years have seen a tremendous improvement in the capability of primal heuristics to almost optimal solutions early in the tree.
- However, a very meaningful experiment [Danna 2006] has shown that even the knowledge of the optimal solution from the beginning of the search only improves on average the MILP running time by a factor of 2.
- In other words, heuristics largely impact on the user perception of the quality of a solver, and are fundamental in the real-world context.
- The primal heuristics implemented in the solvers go from very light and easy, as variations of the classical rounding of the LP solution, to much more heavy and complex, like local search and metaheuristics.
- As mentioned earlier, a relevant step has been solving MILPs within the MILP technology [Fischetti & Lodi 2002].


SCIP
Tree
Pricer
Propa gator
pseudo obj
root redcost
vbound
Relax
Variable
Event
default
Cutpool
Conflict
Primal Heuristic
fracdiving
intdiving
feaspump
fixand infer
guided diving
int shifting
linesearch diving
cross over
dins
local branching
mutation
actcons diving
coef diving
subnlp
objpscost diving
octane
zi round
veclen diving
pscost diving
oneopt
under cover
twoopt
rins
rens
trivial
shift& prop
rounding
trysol
rootsol diving
simple rounding
shifting
allfull
https://www.scipopt.org/

Mixed-Integer Linear Programming: The AI Challenge and Opportunity

Preamble and Disclaimer

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

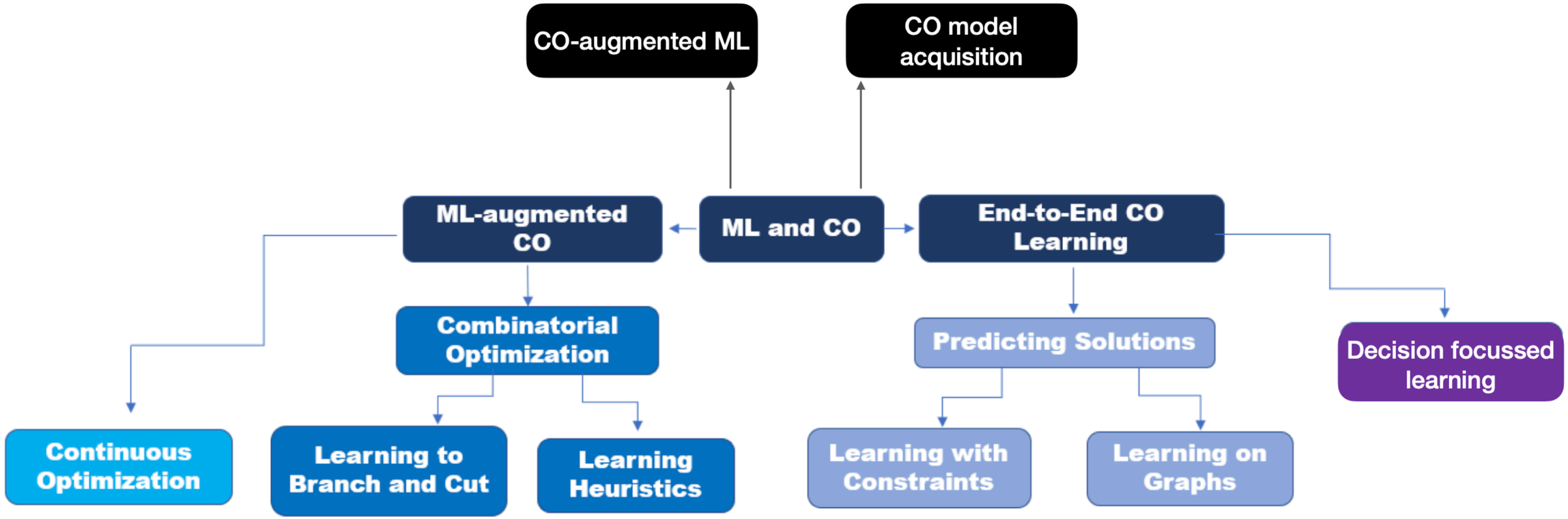
This is due to the incredible success of ML, especially deep learning, in beating human capabilities in image recognition, language processing and sequential games.

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

The new, recent frontier of Generative Artificial Intelligence is yet another story and is **not** covered by this talk.

Schematic Overview

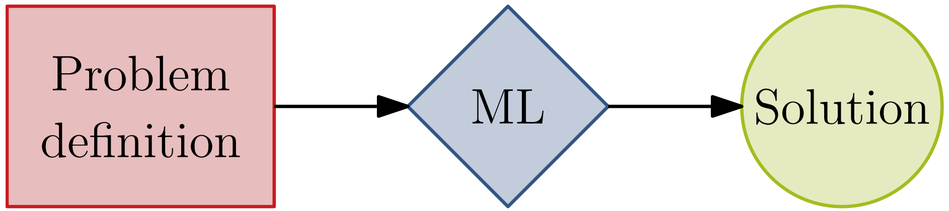
Slide courtesy of N. Yorke-Smith



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

End to End Learning

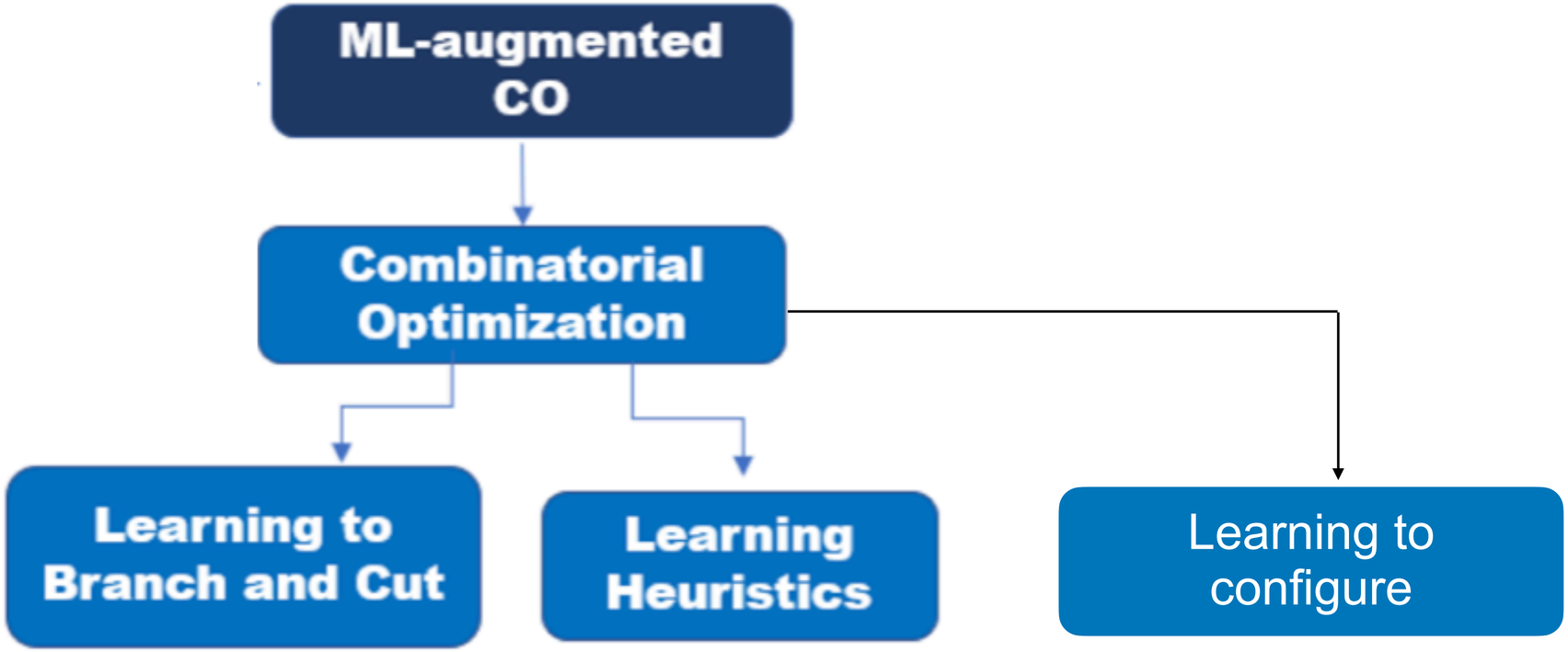


- Learning TSP solutions
  [*Bello et al. 2017*][*Kool and Welling 2018*][*Emami and Ranka 2018*]
  [*Vinyals et al. 2015*][*Nowak et al. 2017*]

- Predict aggregated solutions to MILP under partial information
  [*Larsen et al. 2018*]

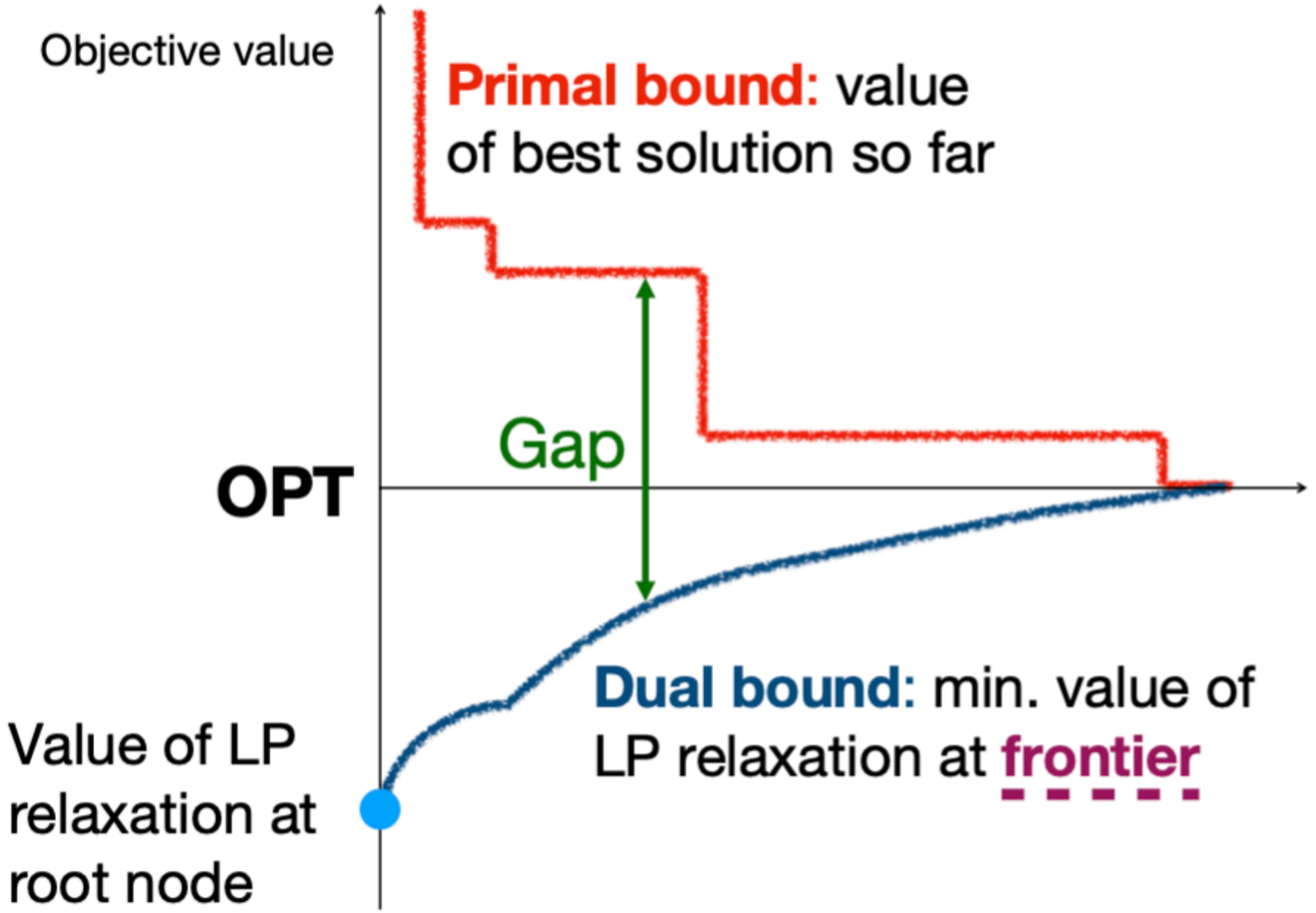
- Approximate obj value to SDP (for cut selection)
  [*Baltean-Lugojan et al. 2018*]

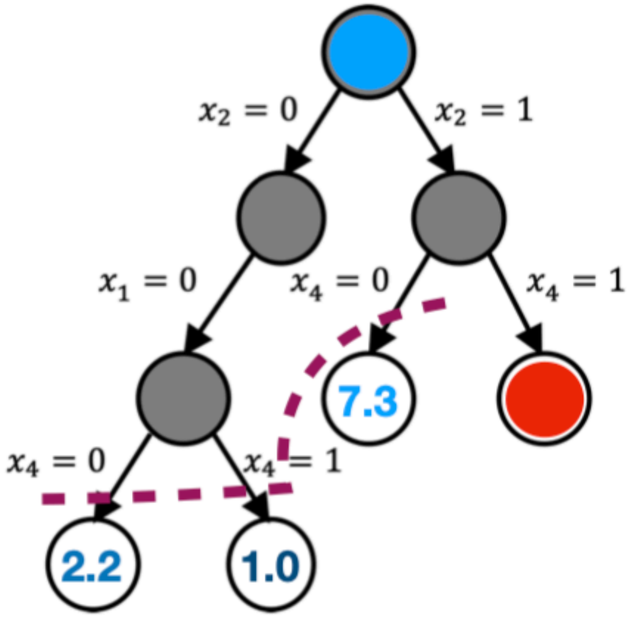
ML-augmented CO (or MILP)



L. Scavuzzo, K. Aardal, A. Lodi, N. Yorke-Smith: Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming, arXiv:2402.05501, 2024.

$$\min_x c^T x : Ax \leq b, x \in \{0,1\}^n$$





Search tree nodes

Slide courtesy of E. Khalil

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

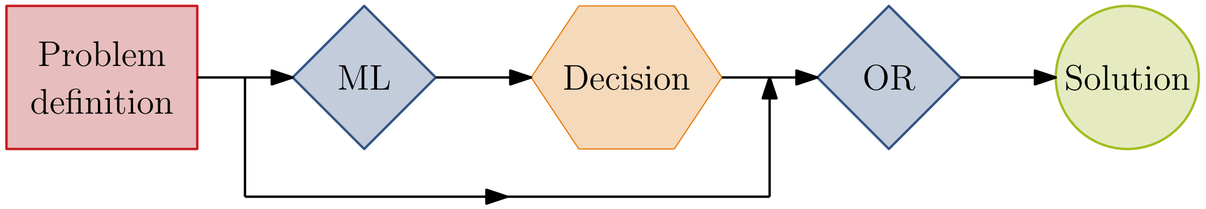
Question

- 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 Properties



- Use a decomposition method
  [*Kruber et al. 2017*]

- Linearize an MIQP
  [*Bonami et al. 2018*]

- Provide initial cancer treatment plans to inverse optimization
  [*Mahmood et al. 2018*]

Learning Properties, an example

- A classifier to decide on the linearization of MIQPs in CPLEX
  [Bonami, Lodi & Zarpellon (2018, 2021)]

Solving MIQPs

We consider Mixed-Integer Quadratic Programming problems

$$min_x \left\{ \tfrac{1}{2} x^T Q x + c^T x : Ax = b, l \leq x \leq u, x_j \in \mathbb{Z}\ \forall\, j \in \mathcal{I} \right\},\ Q \in \mathbb{R}^{n \times n} \text{ symmetric}$$

"**convex**", i.e., $Q \succcurlyeq 0$ or easily made so by

- Linearization of products $x_i x_j,\ x_j \in \{0,1\},\ l_i \leq x_i \leq u_i$ (via McCormick inequalities)
- Perturbation of Q_B diagonal $x^T Q x + \sum_{j \in B} \rho_j \left(x_j^2 - x_j\right) = x^T Q x$

at preprocessing, then solved by CPLEX using QP simplex-based B&B.

Both operations can be applied to $Q \succcurlyeq 0$

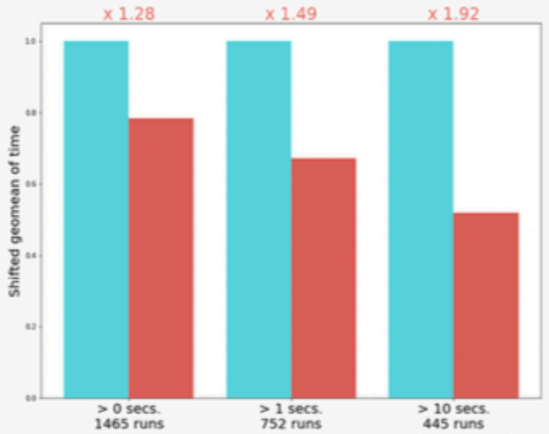
↪ linearization potentially **reformulates a MIQP into a MILP** (depending on Q nnz)

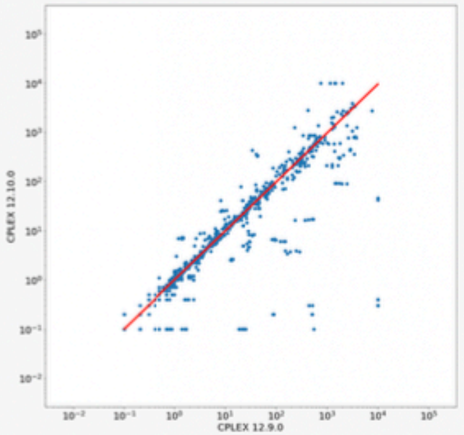
Learning Properties, an example (cont.d)

A classifier in CPLEX

Fine-tuning and final training of SVM in the solver,

- 28% better runtimes on MIQP test instances (92% on > 10 secs.)
- few degradations, compensated by overall improvements



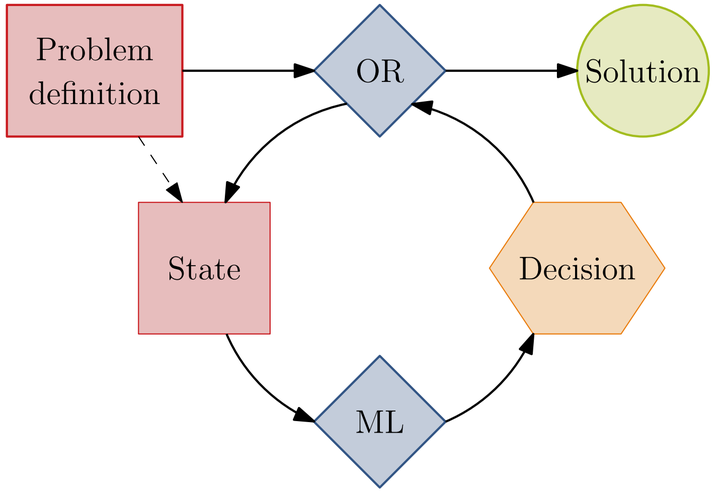


↪ CPLEX 12.10.0 deploys a classifier to decide on MIQPs linearization

↪ Framework and methodology for end-to-end integration of ML into MIP technology

Learning Repeated Decisions

- Learning where to run heuristics in B&B
  [*Khalil et al. 2017b*]
- Learning to branch
  [*Lodi and Zarpellon 2017*] (survey)
- Learning gradient descent
  e.g. [*Andrychowicz et al. 2016*]
- Predicting booking decisions under imperfect information
  [*Larsen et al. 2018*]
- Learning cutting plane selection
  [*Baltean-Lugojan et al. 2018*]



just a matter of viewpoint

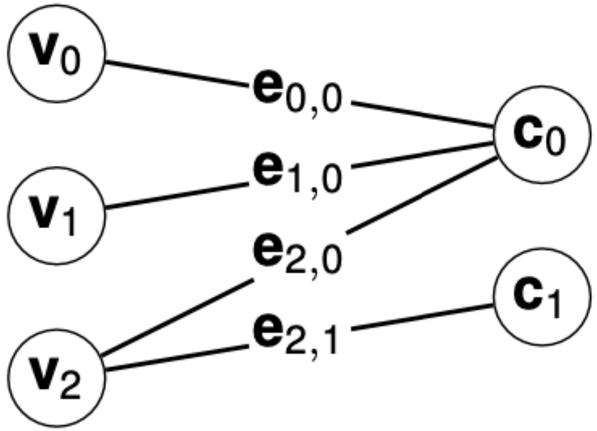
Learning Repeated Decisions, an example

Exact combinatorial optimization with graph convolutional neural networks

[Gasse, Chételat, Ferroni, Charlin & Lodi (2019)]

Natural representation : variable / constraint bipartite graph

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



- $\mathbf{v}_i$: variable features (type, coef., bounds, LP solution. . .)
- $\mathbf{c}_j$: constraint features (right-hand-side, LP slack. . .)
- $\mathbf{e}_{i,j}$: non-zero coefficients in $\mathbf{A}$

Learning Repeated Decisions, an example (cont.d)

Full Strong Branching (FSB): good branching rule, but expensive.
Can we learn a fast, good-enough approximation?

Behavioral cloning

- collect $\mathcal{D} = \{(\mathbf{s}, a^\star), \dots\}$ from the expert agent (FSB)
- estimate $\pi^\star(a\mathbf{s})$ from $\mathcal{D}$

\+ no reward function, supervised learning, well-behaved

– will never surpass the expert...

Minimum set covering [Balas & Hu (1980)]

	Easy			Medium			Hard		
Model	Time	Wins	Nodes	Time	Wins	Nodes	Time	Wins	Nodes
FSB	17.30	0 / 100	17	411.34	0 / 90	171	3600.00	0 / 0	n/a
RPB	8.98	0 / 100	**54**	60.07	0 / 100	1741	1677.02	4 / 65	47 299
XTrees	9.28	0 / 100	187	92.47	0 / 100	2187	2869.21	0 / 35	59 013
SVMrank	8.10	1 / 100	165	73.58	0 / 100	1915	2389.92	0 / 47	42 120
λ-MART	7.19	14 / 100	167	59.98	0 / 100	1925	2165.96	0 / 54	45 319
GCNN	**6.59**	**85** / 100	134	**42.48**	**100** / 100	**1450**	**1489.91**	**66** / 70	**29 981**

3 problem sizes

- 500 rows, 1000 cols (easy), training distribution
- 1000 rows, 1000 cols (medium)
- 2000 rows, 1000 cols (hard)

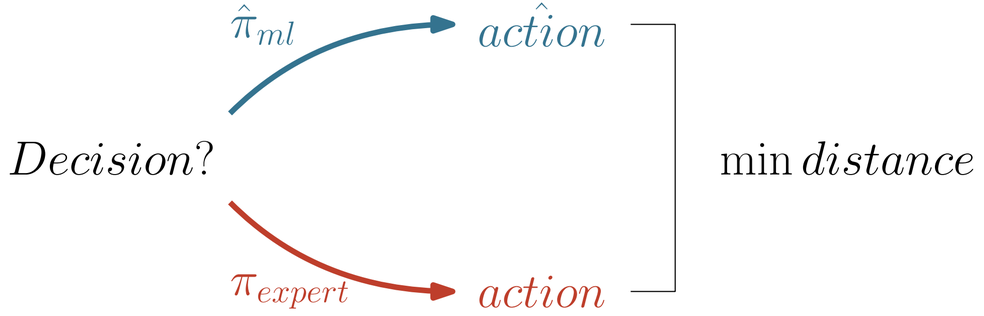
Pays off: better than SCIP's default in terms of solving time.
Generalizes to harder problems !

Similar results on: *combinatorial auctions*, *capacitated facility location*, *maximum independent set*.

Code online: `https://github.com/ds4dm/learn2branch`

Learning Methods

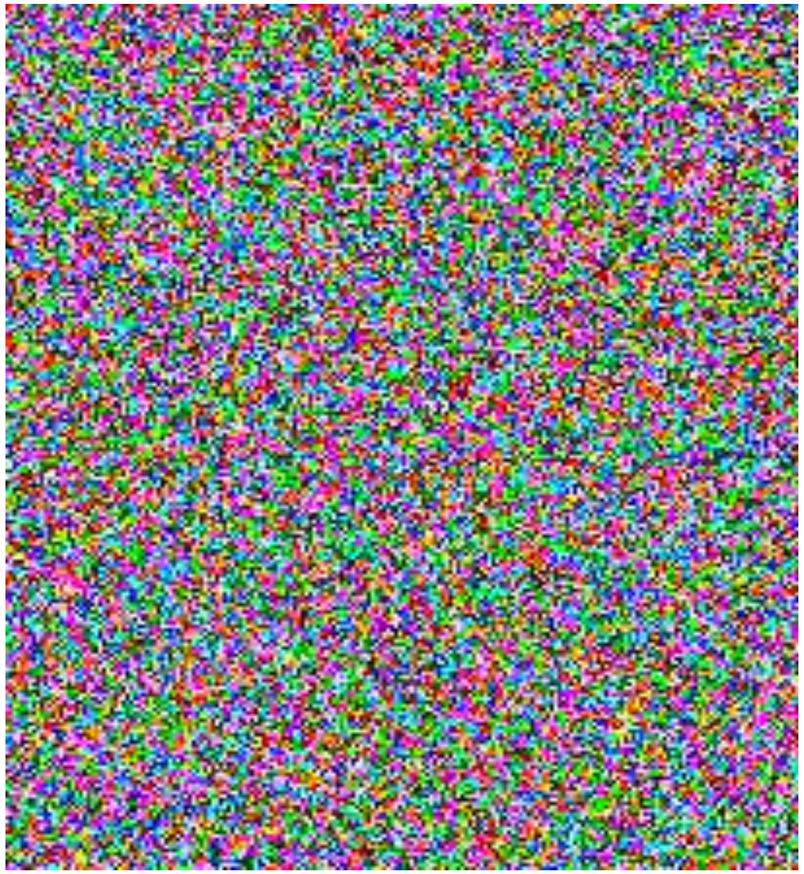
Demonstration



Experience

ML-augmented MILP: Perspectives

Random Images

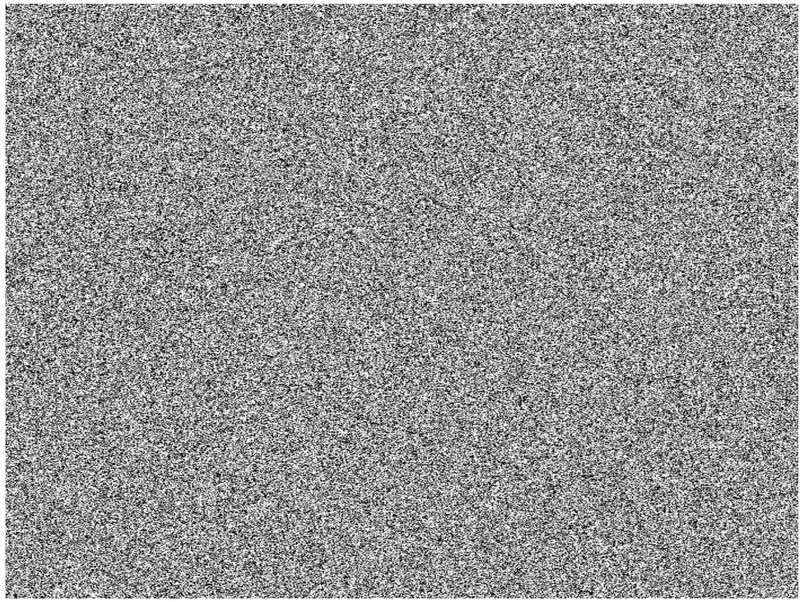


Random iid pixels

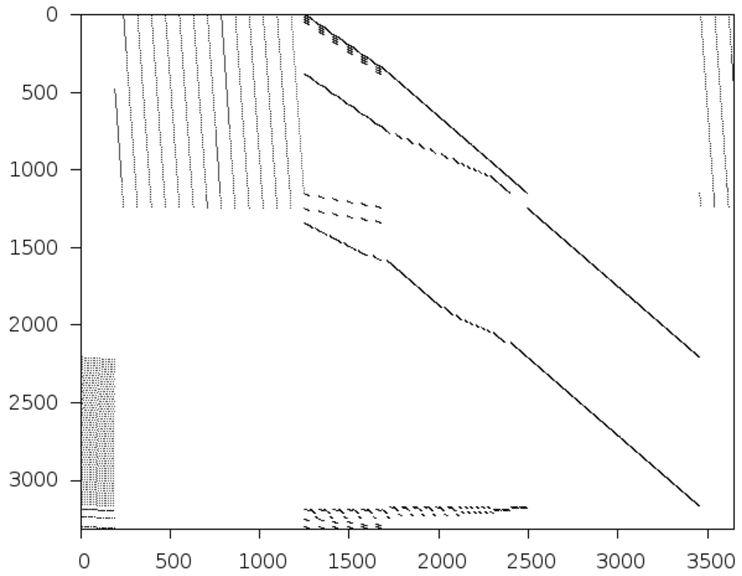
Random face (GAN)

thispersondoesnotexist.com

Random Instances

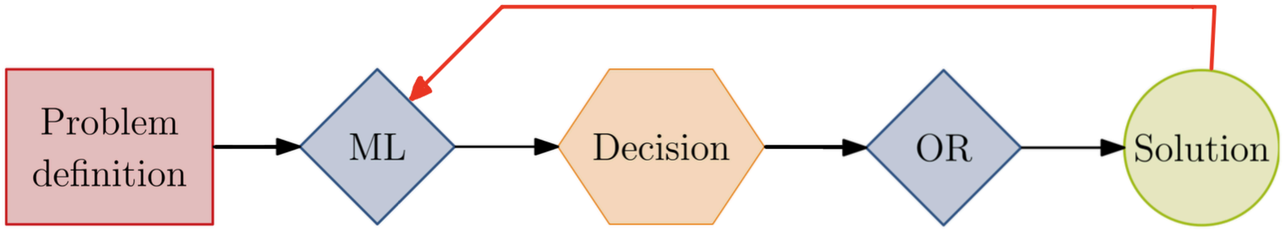


Random iid coefficients



a1c1s1 from MipLib 2017

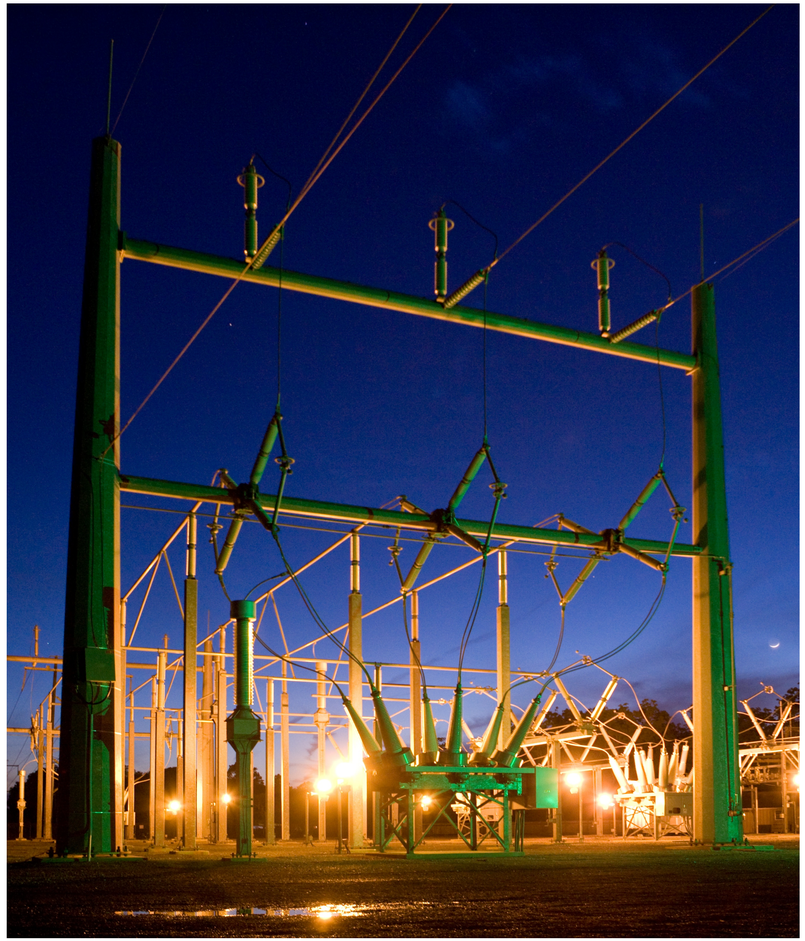
Decision-aware learning



- Quadratic Programming
  [Amos and Kolter 2017]
- Novel loss functions development
  [Elmachtoub and Grigas 2022] [Mandi et al. 2022]
- Perturbation or penalization methods
  [Mandi and Guns 2020] [Blondel et al. 2020] [Wilder et al. 2019] [Parmentier 2021]

Business Applications

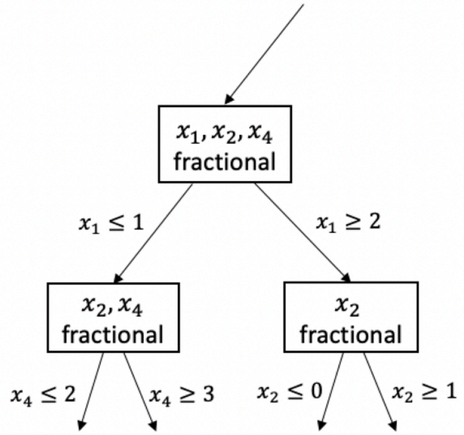
- Many businesses care about solving **similar** problems **repeatedly**
- Solvers do not make any use of this aspect
- Power systems and market [*Xavier et al. 2019*]
  - Schedule 3.8 kWh ($400 billion) market annually in the US
  - Solved multiple times a day
  - 12x speed up combining ML and MILP

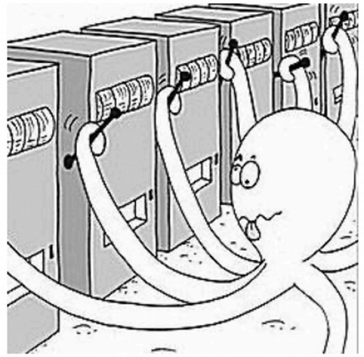


ML-augmented MILP: Summary

Three Tasks

Finding feasible solutions of MILP





Primal Task | **Dual Task** | **Configuration Task**

Summary

- Mixed-Integer Programming is one of the tools of choice for many applications, and still an active area of research in mathematics.
- The needs for making MILP competitive in many applied contexts have in common the exploitation of data, for example
  - data uncertain,
  - “learn” to repeatedly solve the “same instance,”
  - pattern discovery.
- Overall, we start to have (solid) evidence that ML can play a significant role to deal with those needs and challenges.