Machine Learning for Combinatorial and Global Optimization

Can Li

Davidson School of Chemical Engineering, Purdue University

PSE seminar

Collaborators

A. Prouvost

M. Gasse

D. Chételat

J. Dumouchelle

L. Scavuzzo

A. Lodi

POLYTECHNIQUE MONTRÉAL

DATA SCIENCE FOR REAL-TIME DECISION-MAKING

Mila

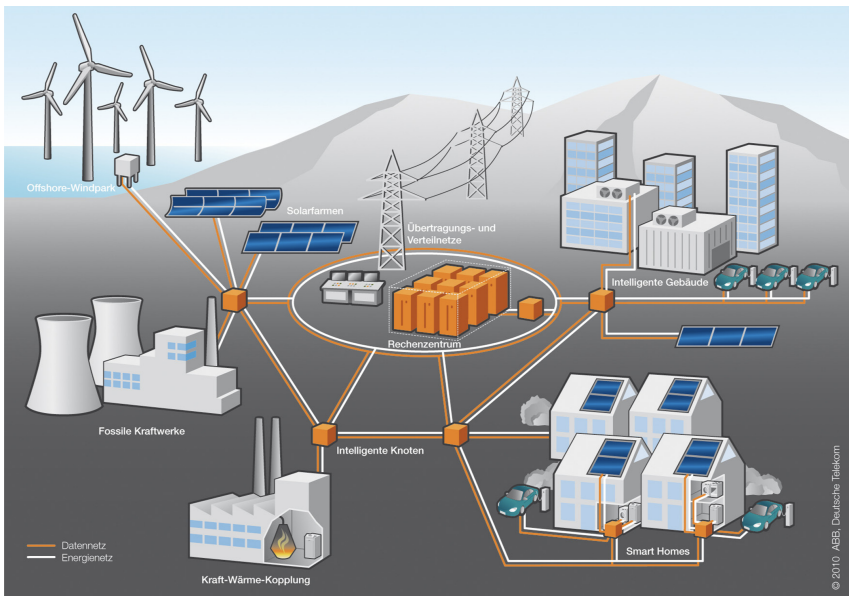
Motivation

It is common to repeatedly solve similar combinatorial optimization problems in practice.

Motivation

It is common to repeatedly solve similar combinatorial optimization problems in practice.

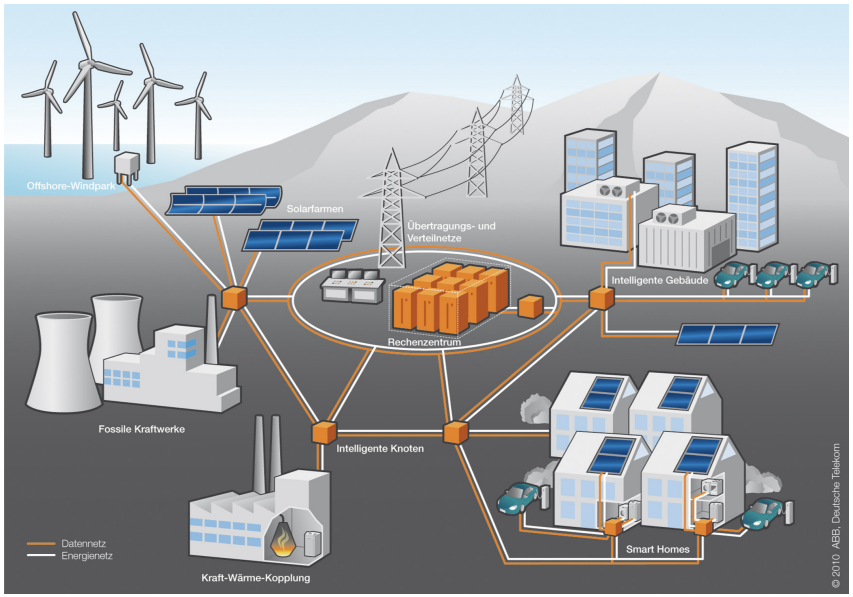
- Power producers repeatedly solve unit commitment (UC) problems to meet power demand at minimum market cost. Xavier et al., 2021



Motivation

It is common to repeatedly solve similar combinatorial optimization problems in practice.

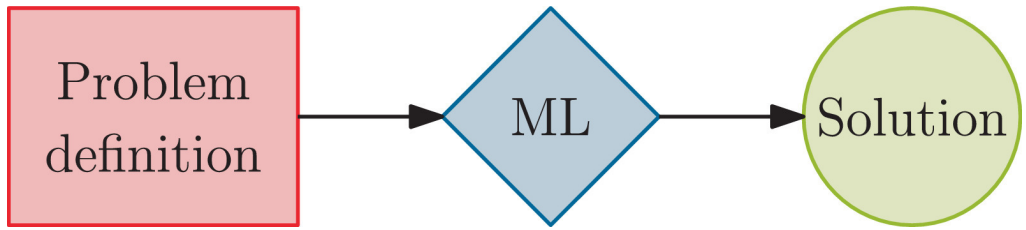
- Power producers repeatedly solve unit commitment (UC) problems to meet power demand at minimum market cost. Xavier et al., 2021



Use machine learning to make decisions when solving similar optimization problems.

End to end learning[1]

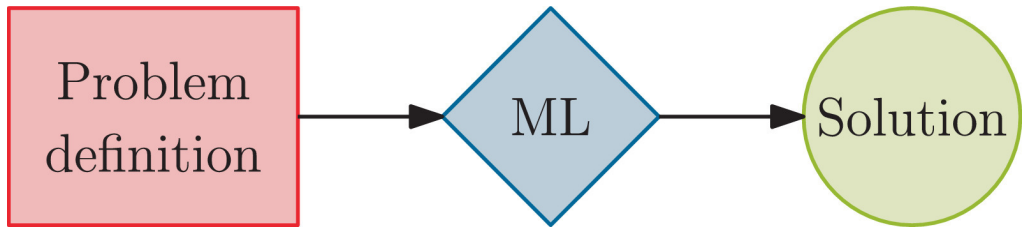
Train the machine learning model to output solutions directly from the input instance



[1] Y. Bengio et al. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon.

End to end learning[1]

Train the machine learning model to output solutions directly from the input instance

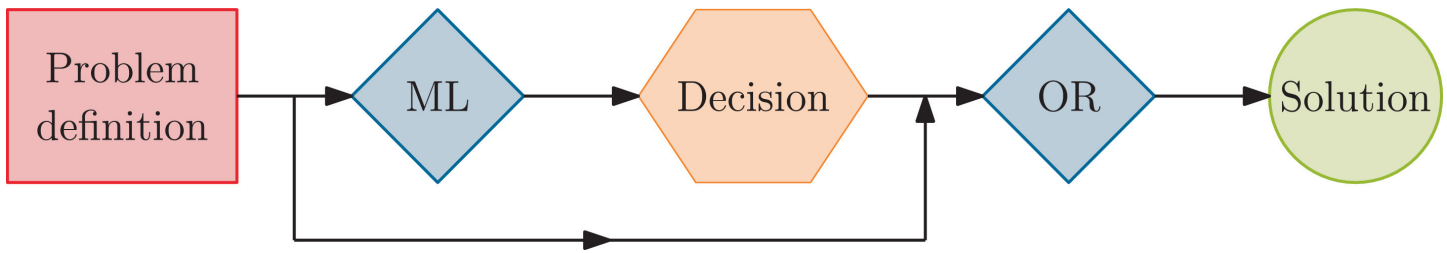


- W. Chen et al. (2022). Learning optimization proxies for large-scale security-constrained economic dispatch.
- E. Khalil et al. (2017). Learning combinatorial optimization algorithms over graphs.

[1] Y. Bengio et al. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon.

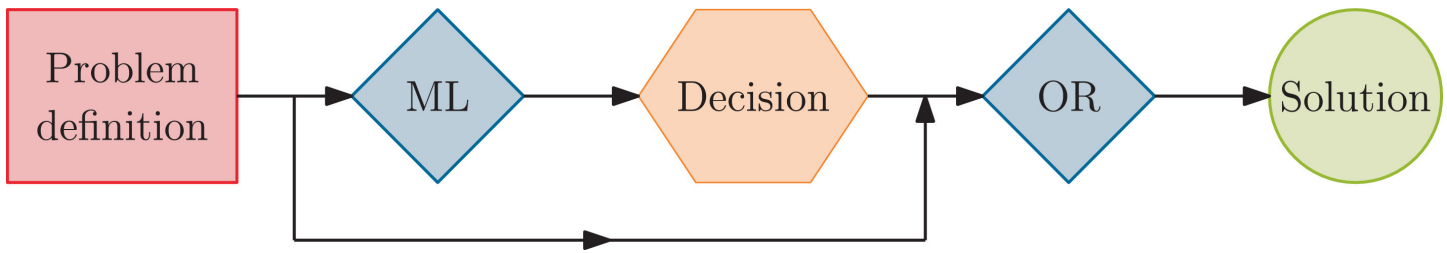
Learning to configure algorithms

Machine learning can provide a parametrization of the algorithm



Learning to configure algorithms

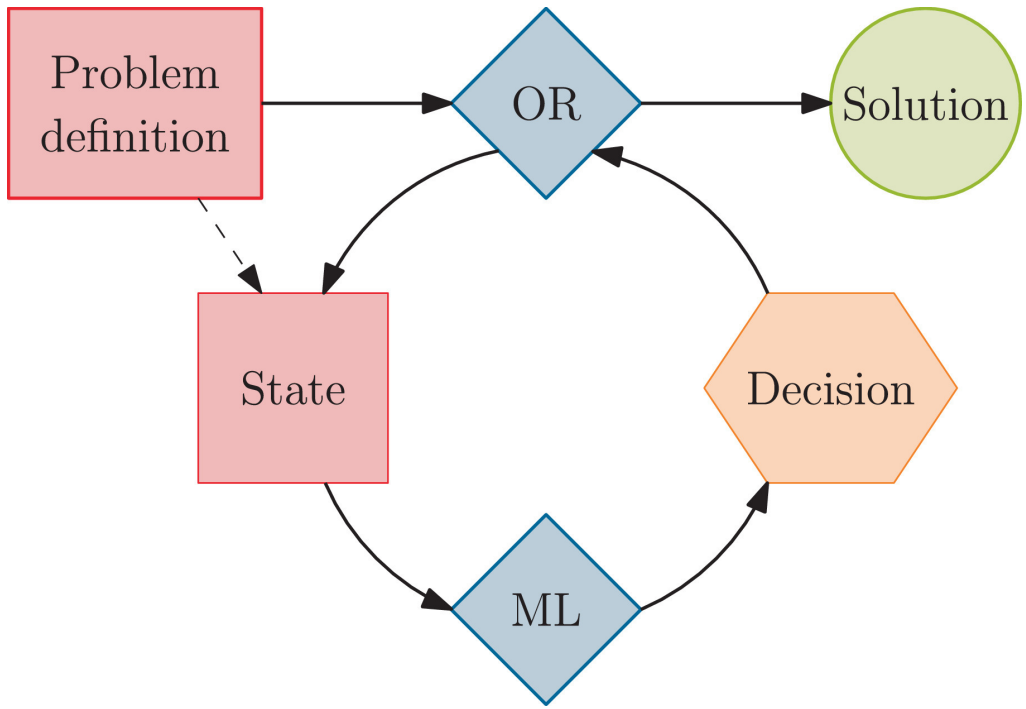
Machine learning can provide a parametrization of the algorithm



- I. Mitrai et al. (2022). Learning to Initialize Generalized Benders Decomposition.
- P. Bonami et al. (2022). A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX.
- M. Kruber et al. (2017). Learning when to use a decomposition.

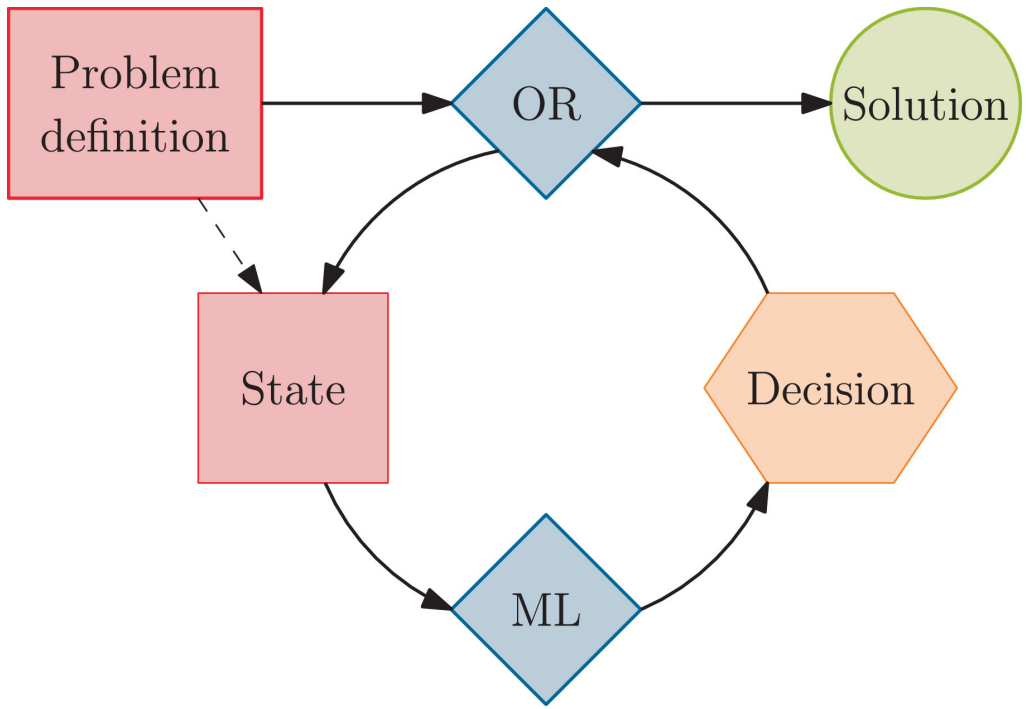
Machine learning alongside optimization algorithms

Calling an ML method to make decisions within an optimization algorithm



Machine learning alongside optimization algorithms

Calling an ML method to make decisions within an optimization algorithm



- S. Zeng et al. (2022). A reinforcement learning approach to parameter selection for distributed optimal power flow.
- M. Gasse et al. (2019b). Exact combinatorial optimization with graph convolutional neural networks.

Machine Learning for Combinatorial Optimization

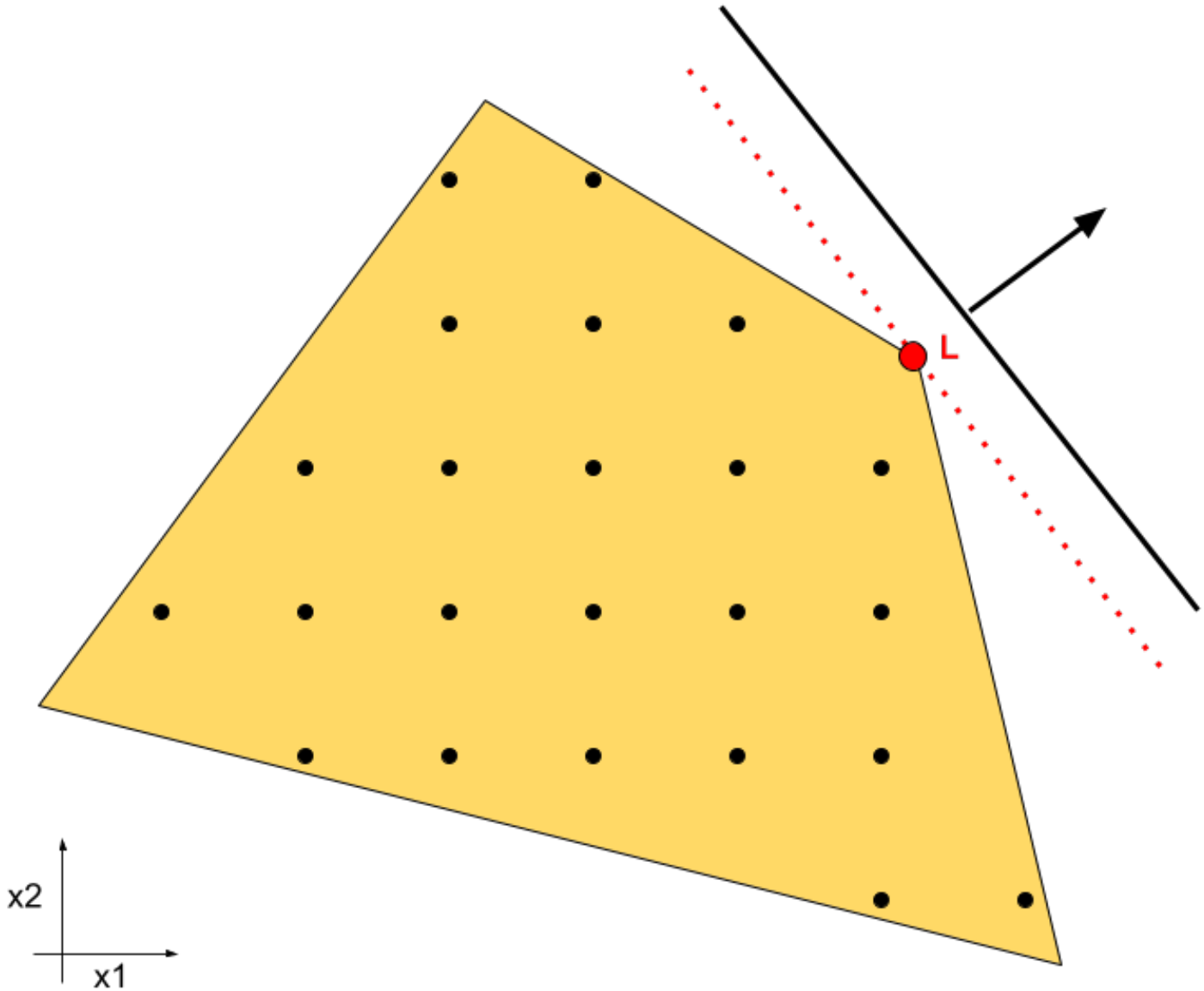
Mixed-Integer Linear Program (MILP)

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

- $c \in \mathbb{R}^n$ the objective coefficients
- $A \in \mathbb{R}^{m \times n}$ the constraint coefficient matrix
- $b \in \mathbb{R}^m$ the constraint right-hand-sides
- $l, u \in \mathbb{R}^n$ the lower and upper variable bounds
- $p \leq n$ integer variables

NP-hard problem.

Linear Program (LP) relaxation



Branch-and-Bound

Split the LP recursively over a non-integral variable, i.e. $\exists i \leq p \mid x_i^\star \notin \mathbb{Z}$

$$x_i \leq \lfloor x_i^\star \rfloor \quad \vee \quad x_i \geq \lceil x_i^\star \rceil.$$

Lower bound (L): minimal among leaf nodes.
Upper bound (U): minimal among leaf nodes with integral solution.

Branch-and-Bound

Split the LP recursively over a non-integral variable, i.e. $\exists i \leq p \mid x_i^\star \notin \mathbb{Z}$

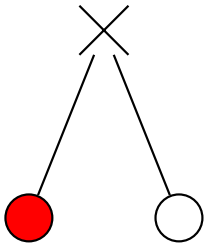
$$x_i \leq \lfloor x_i^\star \rfloor \quad \vee \quad x_i \geq \lceil x_i^\star \rceil.$$

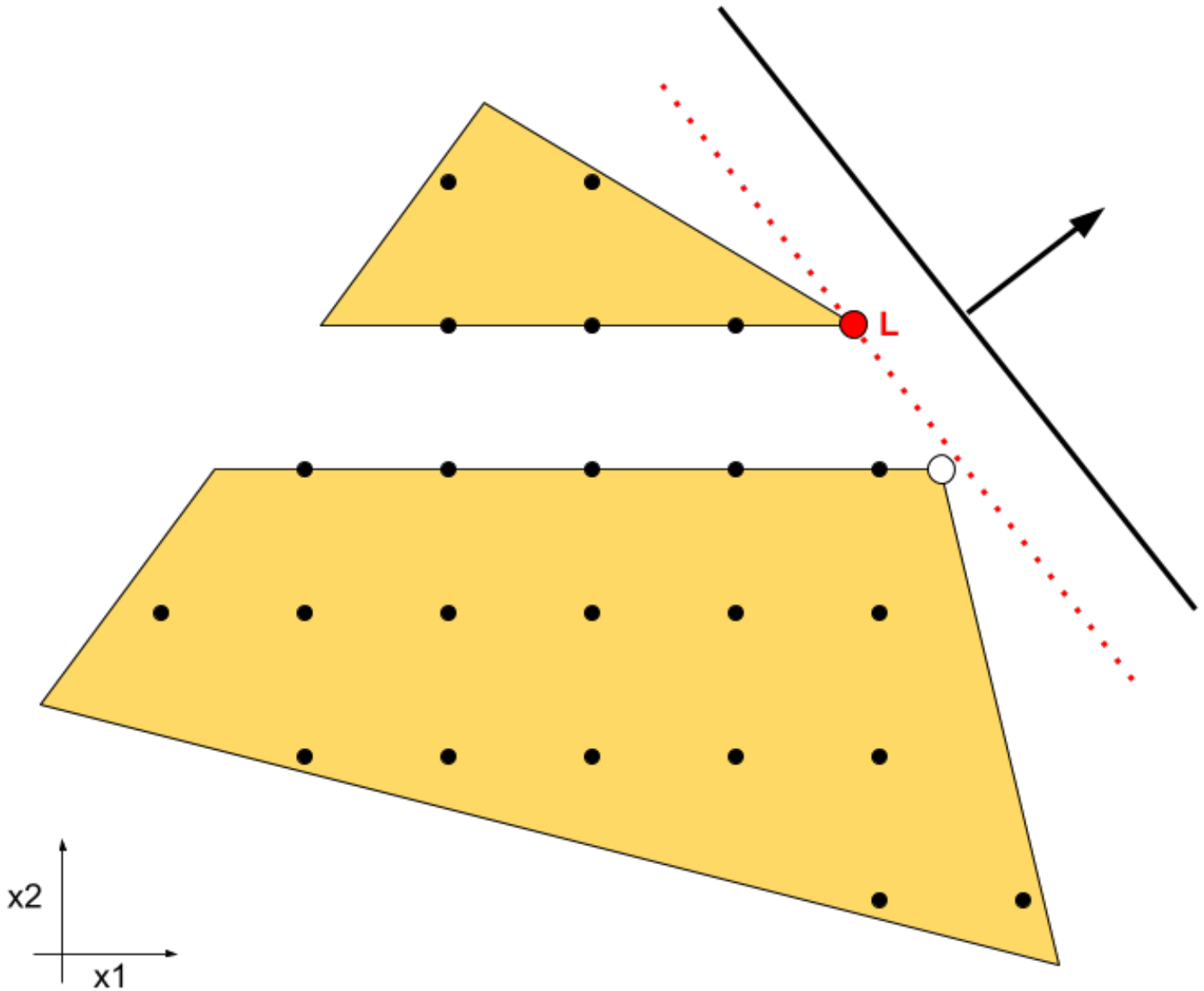
Lower bound (L): minimal among leaf nodes.
Upper bound (U): minimal among leaf nodes with integral solution.

Stopping criterion:

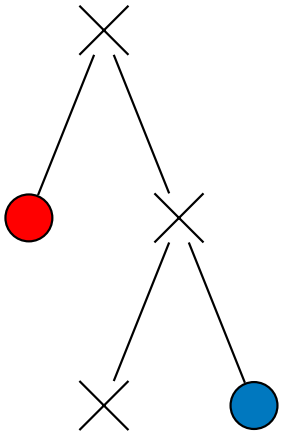
- L = U (optimality certificate)
- L = ∞ (infeasibility certificate)
- L - U < threshold (early stopping)

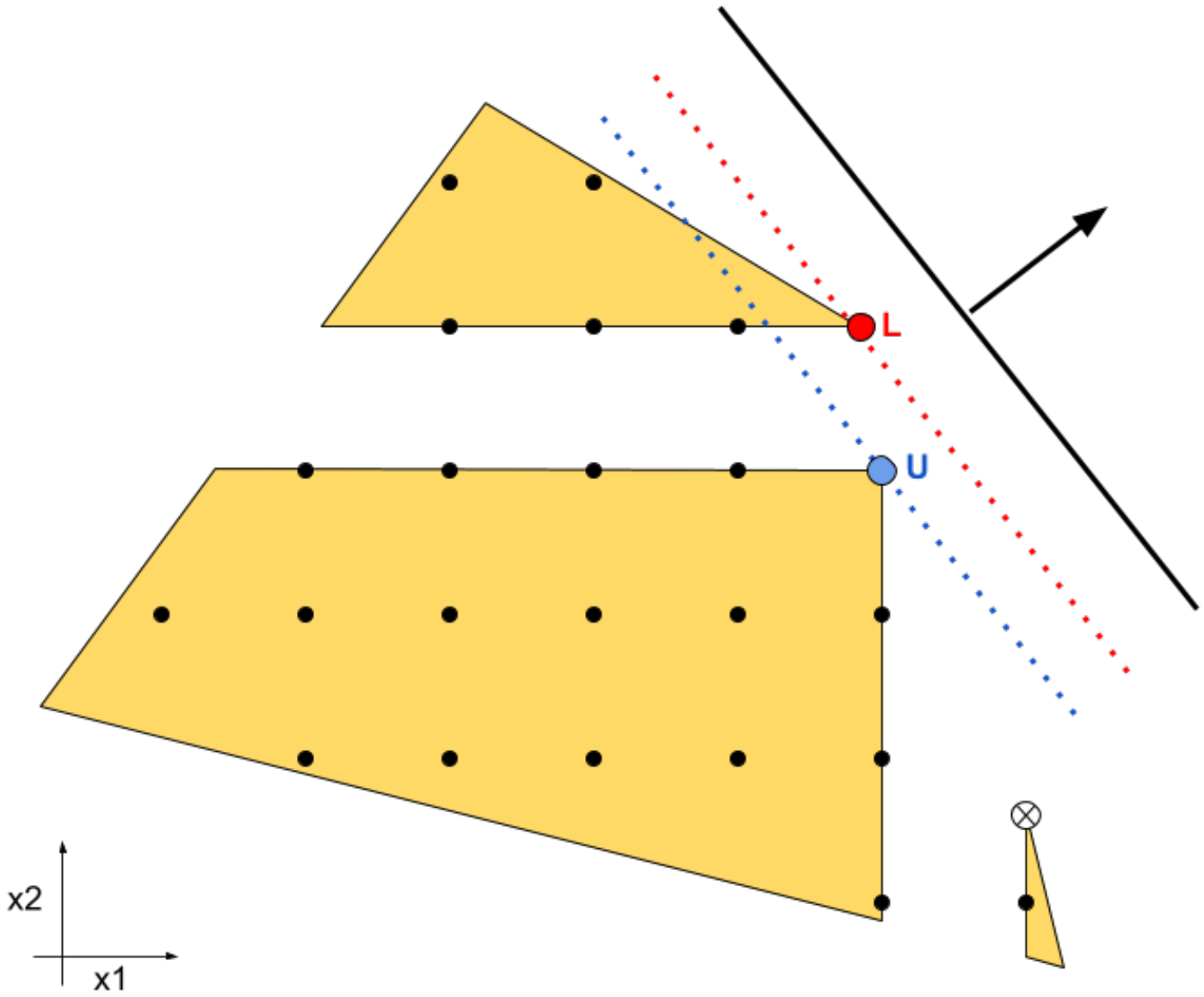
Branch-and-Bound



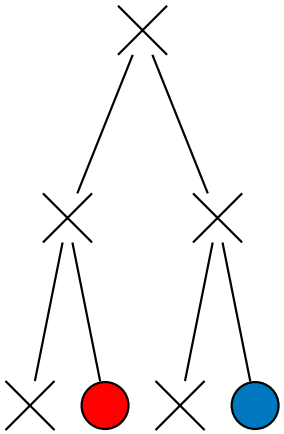


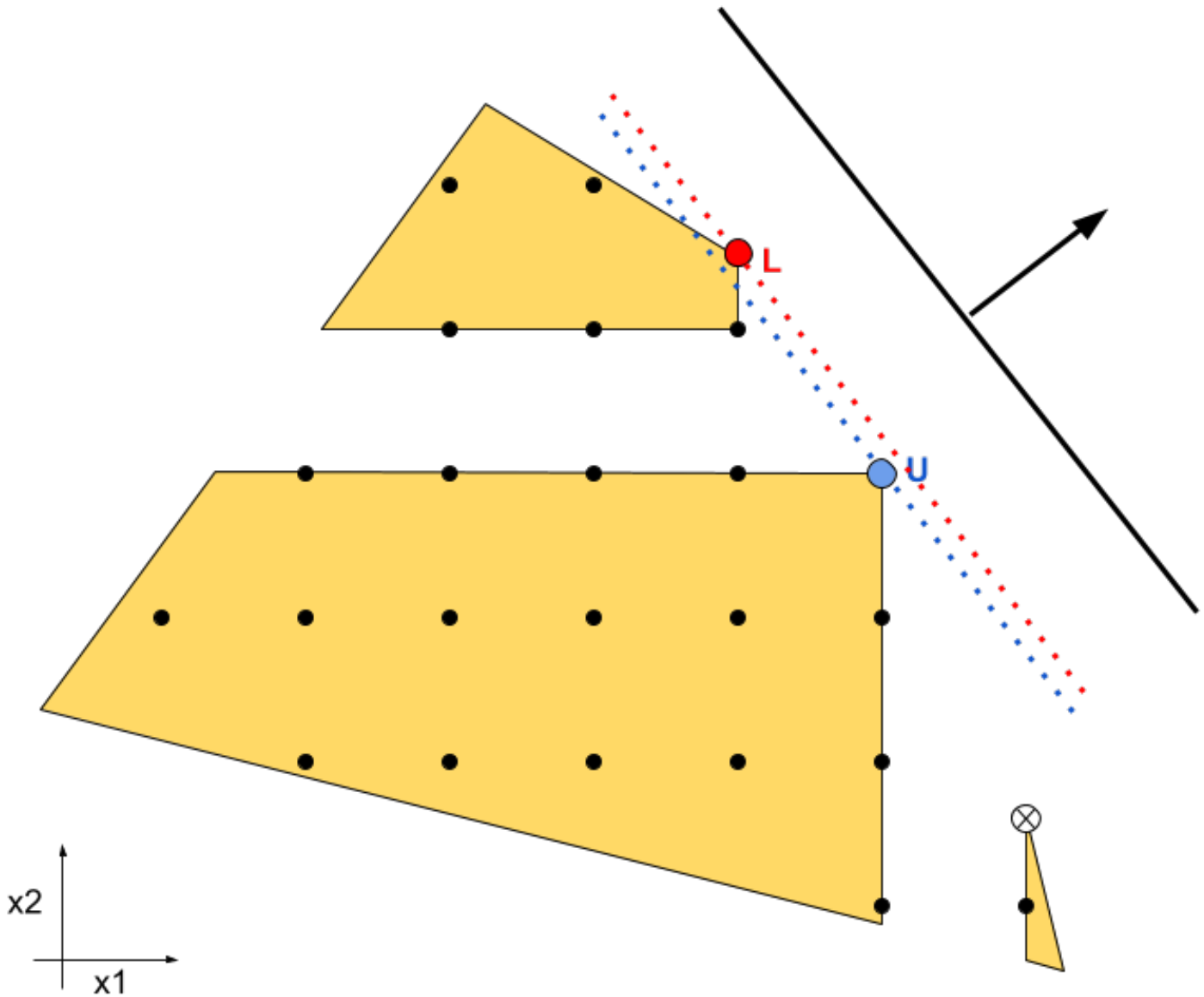
Branch-and-Bound



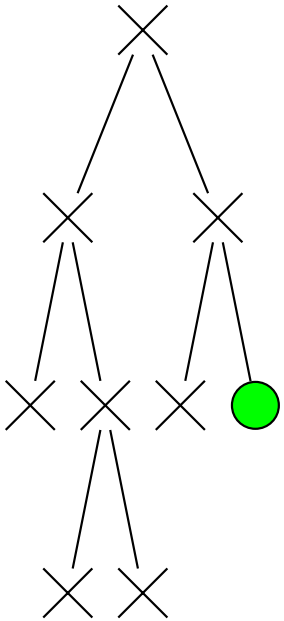


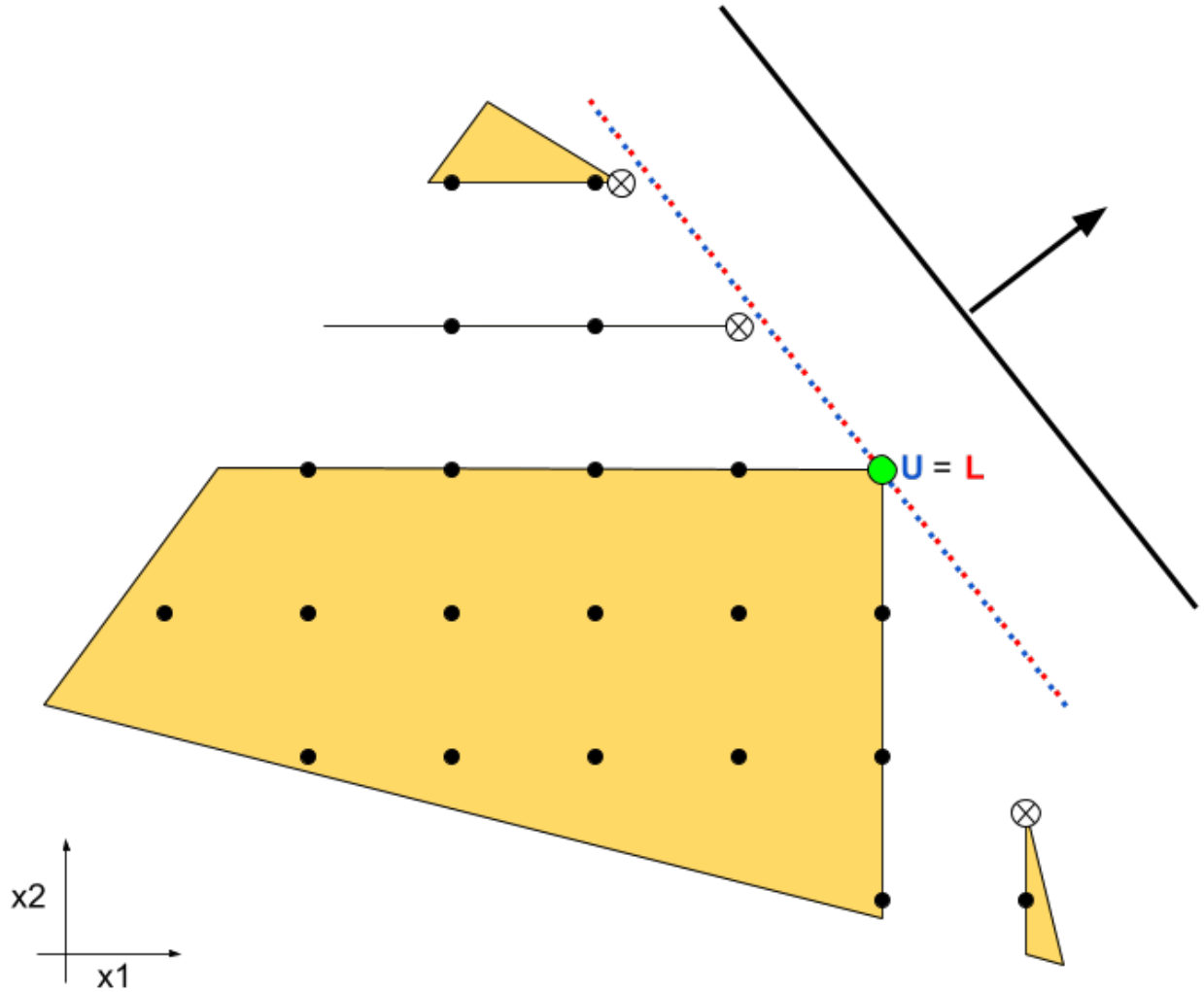
Branch-and-Bound



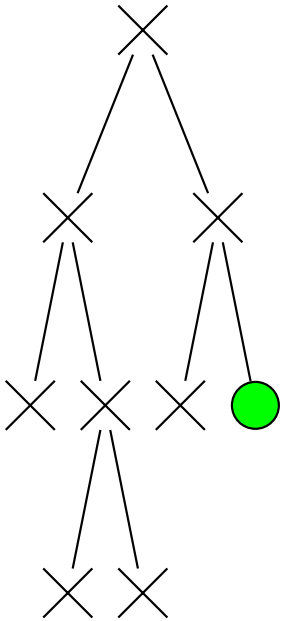


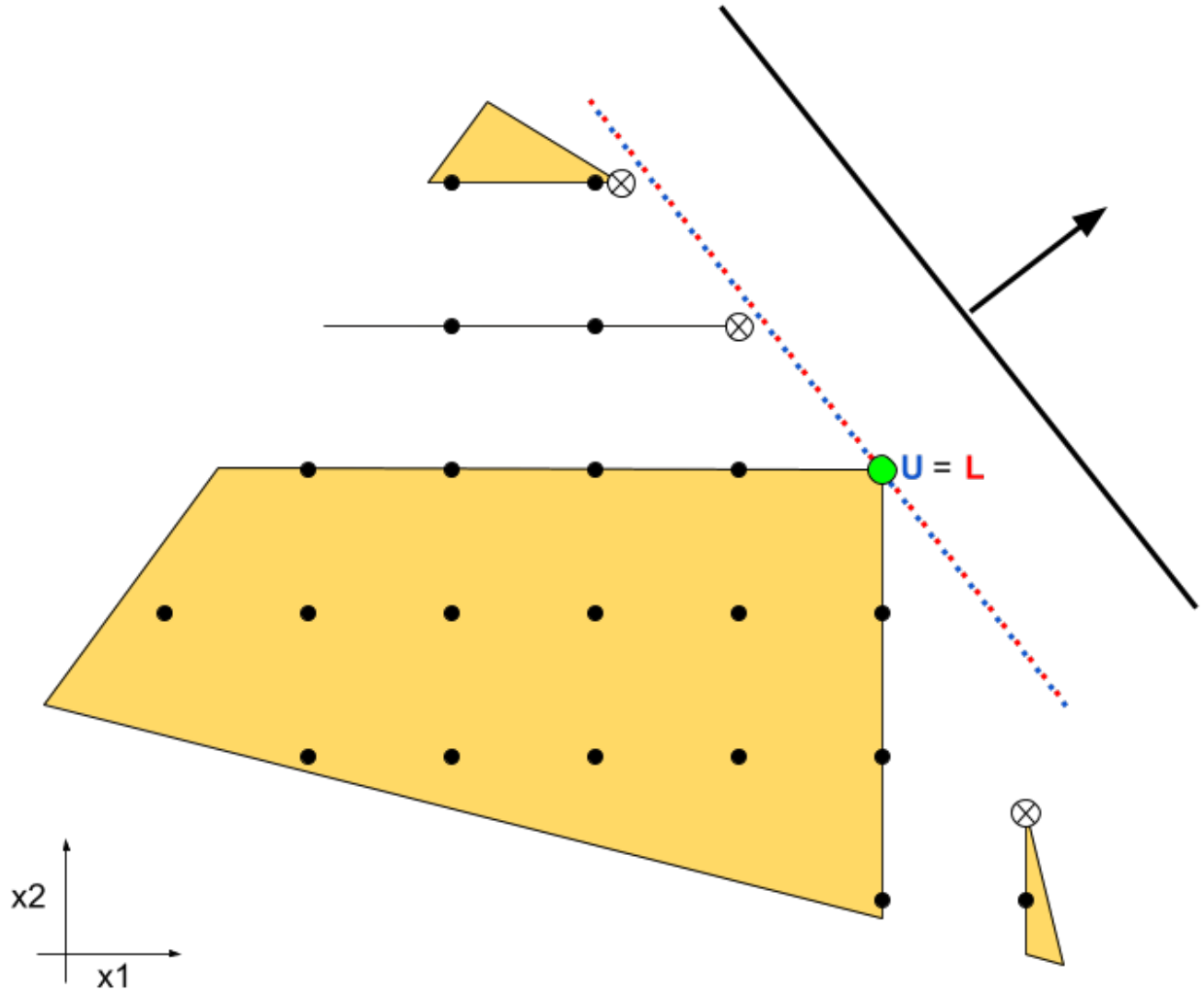
Branch-and-Bound





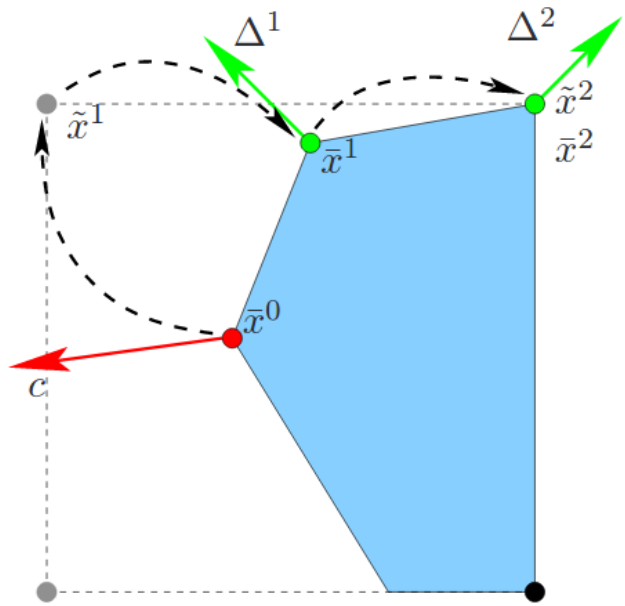
Branch-and-Bound





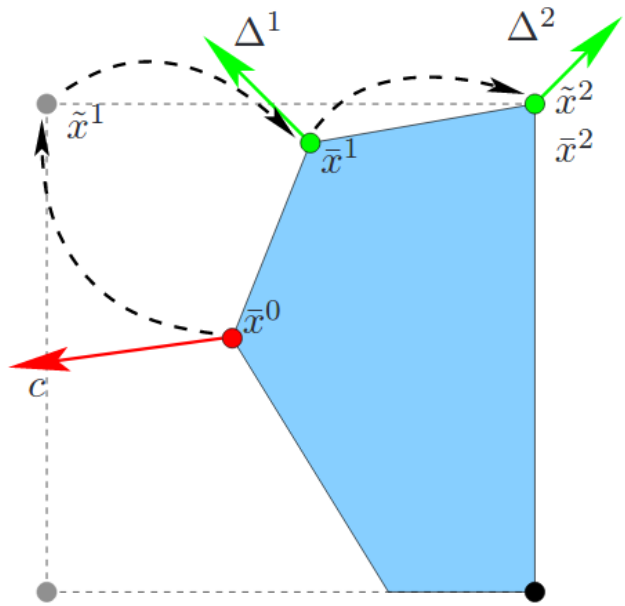
Decision task: which node and variable to select for branching?

Primal heuristics (generic search routines) might run at the each node.



T. Berthold (2006). Primal heuristics for mixed integer programs.

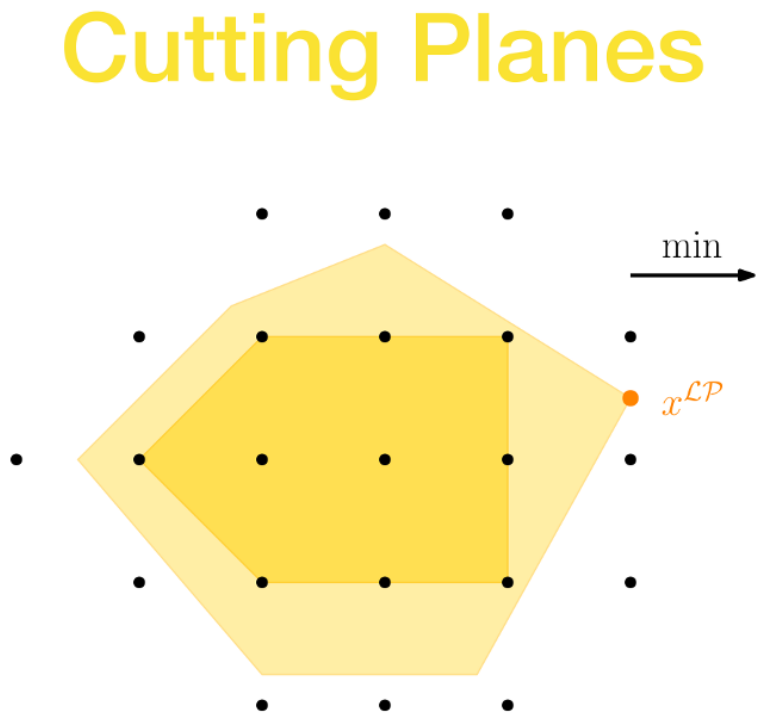
Primal heuristics (generic search routines) might run at the each node.



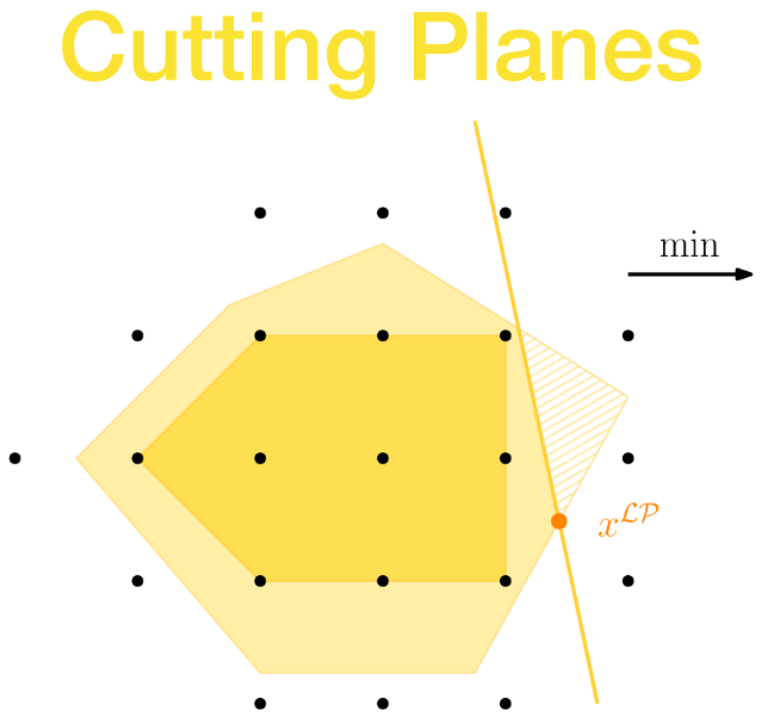
Decision task: which heuristics to run? When? (heuristics are costly)

T. Berthold (2006). Primal heuristics for mixed integer programs.

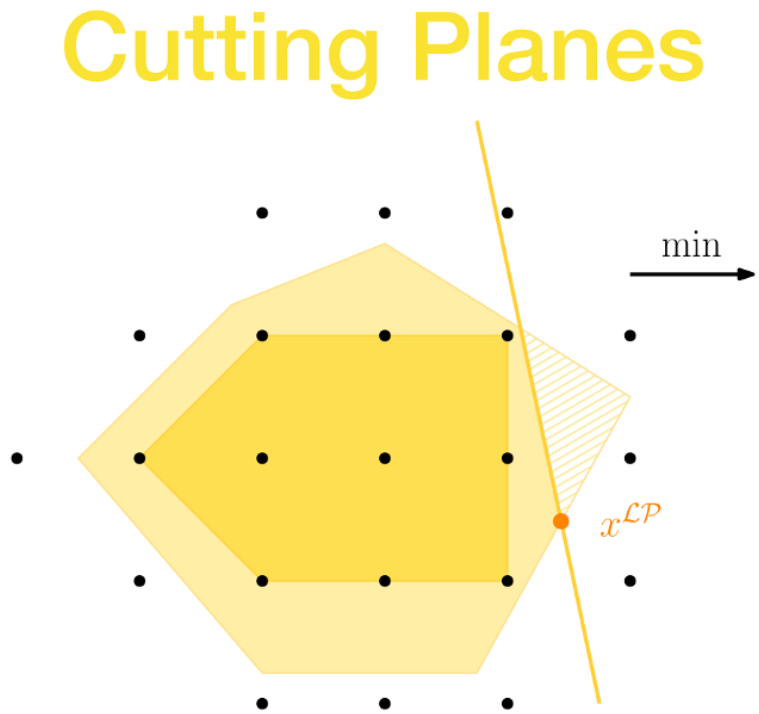
Cuts can be added to the sub-MILPs to tighten the bounds.
(Branch-and-cut)



Cuts can be added to the sub-MILPs to tighten the bounds.
(Branch-and-cut)

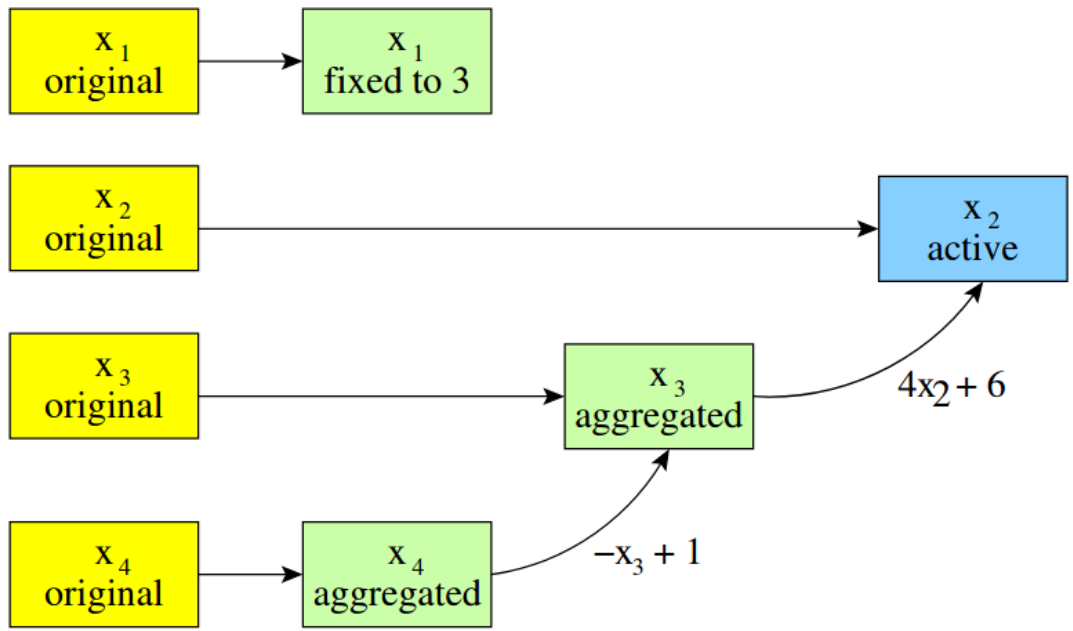


Cuts can be added to the sub-MILPs to tighten the bounds. (Branch-and-cut)



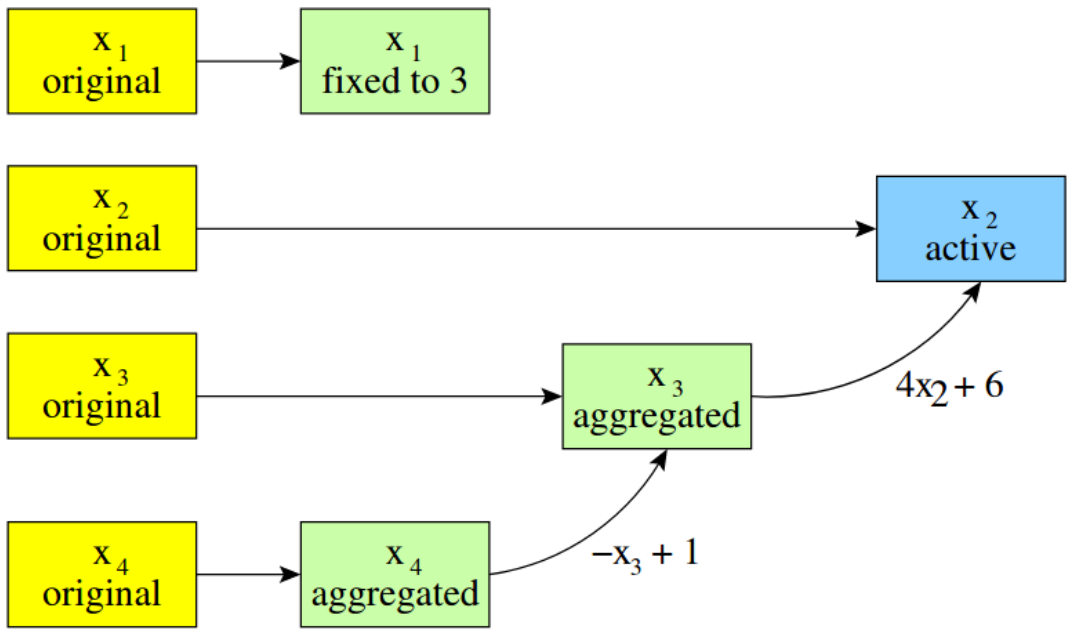
Decision task: which cuts to add to the LP ? Not all cuts are good, some are redundant. Adding too many cuts can lead numerical instabilities.

Preprocessing routines can be run before the solving starts (usually several, sequentially), to simplify and / or tighten the problem formulation.



T. Achterberg (2004). SCIP - A Framework to Integrate Constraint and Mixed Integer Programming.

Preprocessing routines can be run before the solving starts (usually several, sequentially), to simplify and / or tighten the problem formulation.



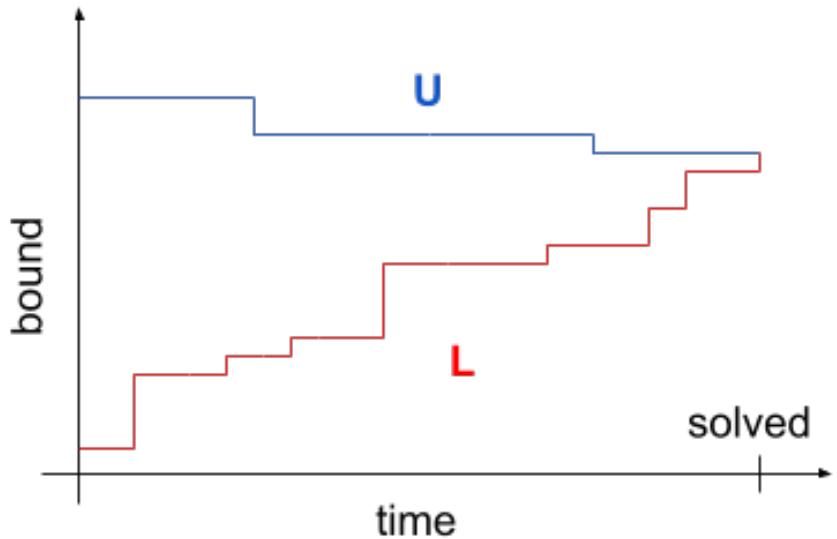
Decision task: which routines to run? How many times?

T. Achterberg (2004). SCIP - A Framework to Integrate Constraint and Mixed Integer Programming.

Solver Design: a Complex Control Problem

Many intertwined decisions:

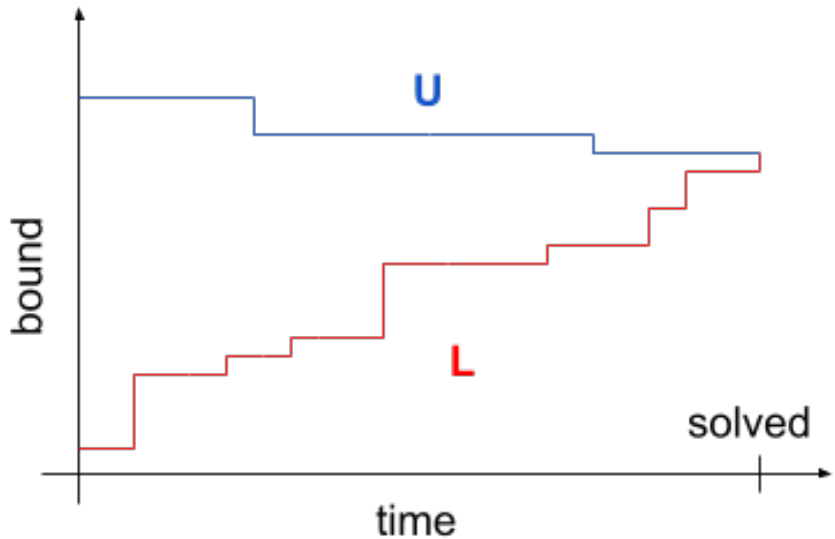
- node selection
- variable selection
- cutting planes
- primal heuristics
- preprocessing
- simplex initialization
- ...



Solver Design: a Complex Control Problem

Many intertwined decisions:

- node selection
- variable selection
- cutting planes
- primal heuristics
- preprocessing
- simplex initialization
- ...



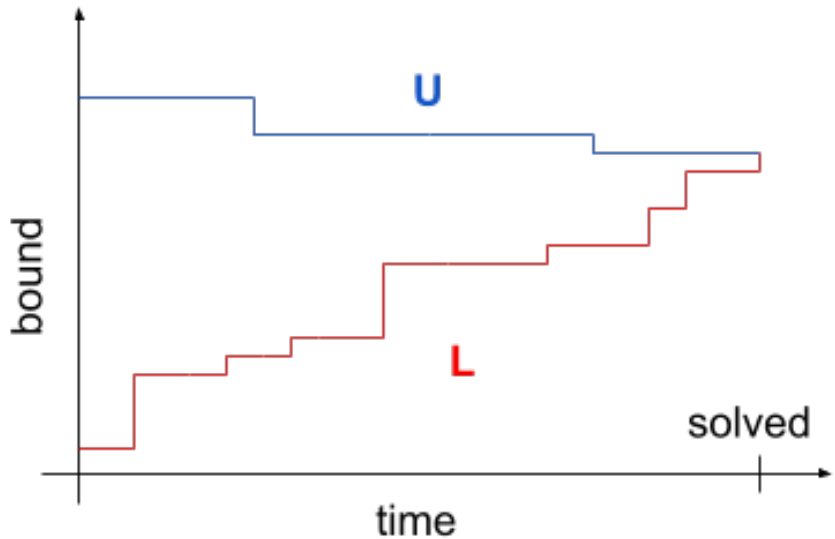
Many evaluation metrics:

- B&B tree size
- solving time: reach U=L fast
- primal-dual integral: U - L ↘ fast
- dual integral: L ↗ fast
- primal integral: U ↘ fast

Solver Design: a Complex Control Problem

Many intertwined decisions:

- node selection
- variable selection
- cutting planes
- primal heuristics
- preprocessing
- simplex initialization
- ...



Many evaluation metrics:

- B&B tree size
- solving time: reach U=L fast
- primal-dual integral: U - L ↘ fast
- dual integral: L ↗ fast
- primal integral: U ↘ fast

State-of-the-art solvers: expert rules + benchmarks.

Learning to branch ?

Not a new idea, early attempts in the 2000's [Achterberg, 2007].

Learning to branch ?

Not a new idea, early attempts in the 2000's [Achterberg, 2007].

Increased interest in recent years.

Imitation Learning (IL), approximate of strong branching, fast

- [E. B. Khalil, Le Bodic, et al., 2016]
- [Hansknecht et al., 2018]
- [Balcan et al., 2018]
- [Gasse et al., 2019a]
- [Gupta et al., 2020]
- [Nair et al., 2020]

Learning to branch ?

Not a new idea, early attempts in the 2000's [Achterberg, 2007].

Increased interest in recent years.

Imitation Learning (IL), approximate of strong branching, fast

- [E. B. Khalil, Le Bodic, et al., 2016]
- [Hansknecht et al., 2018]
- [Balcan et al., 2018]
- [Gasse et al., 2019a]
- [Gupta et al., 2020]
- [Nair et al., 2020]

Reinforcement learning (RL), learn new rules from scratch

- [Sun et al., 2020]
- [Etheve et al., 2020]

Imitation learning

Full Strong Branching (FSB): testing which of the candidate variable gives the best improvement to the objective function before actually branching on them. good branching rule, but expensive.

[2] A. Gleixner et al. (July 2018). The SCIP Optimization Suite 6.

Imitation learning

Full Strong Branching (FSB): testing which of the candidate variable gives the best improvement to the objective function before actually branching on them. good branching rule, but expensive. Can we learn a fast, good-enough approximation ?

[2] A. Gleixner et al. (July 2018). The SCIP Optimization Suite 6.

Imitation learning

Full Strong Branching (FSB): testing which of the candidate variable gives the best improvement to the objective function before actually branching on them. good branching rule, but expensive. Can we learn a fast, good-enough approximation ?

Behavioural cloning

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

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

− will never surpass the expert...

Implementation with the open-source solver SCIP[2]

[2] A. Gleixner et al. (July 2018). The SCIP Optimization Suite 6.

Imitation learning

Full Strong Branching (FSB): testing which of the candidate variable gives the best improvement to the objective function before actually branching on them. good branching rule, but expensive. Can we learn a fast, good-enough approximation ?

Behavioural cloning

- collect $\mathcal{D} = \{(s, a^\star), \dots\}$ from the expert agent (FSB)
- estimate $\pi^\star(a \mid s)$ from $\mathcal{D}$
- \+ no reward function, supervised learning, well-behaved
- − will never surpass the expert...

Implementation with the open-source solver SCIP[2]

Not a new idea

- [Alvarez et al., 2017] predict SB scores, XTrees model
- [E. B. Khalil, Le Bodic, et al., 2016] predict SB rankings, SVMrank model
- [Hansknecht et al., 2018] do the same, λ-MART model

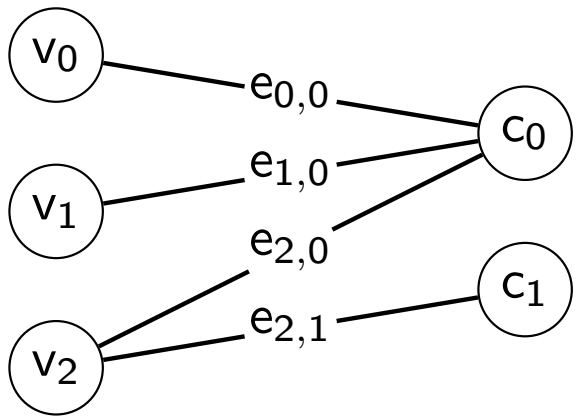
[2] A. Gleixner et al. (July 2018). The SCIP Optimization Suite 6.

Node state encoding

Graph Convolutional Neural Network

Natural representation : variable / constraint bipartite graph

$$\begin{aligned} \underset{x}{\arg\min} \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b, \\ & l \leq x \leq u, \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned}$$



- v_i: variable features (type, coef., bounds, LP solution. . .)
- c_j: constraint features (right-hand-side, LP slack. . .)
- $e_{i,j}$: non-zero coefficients in A

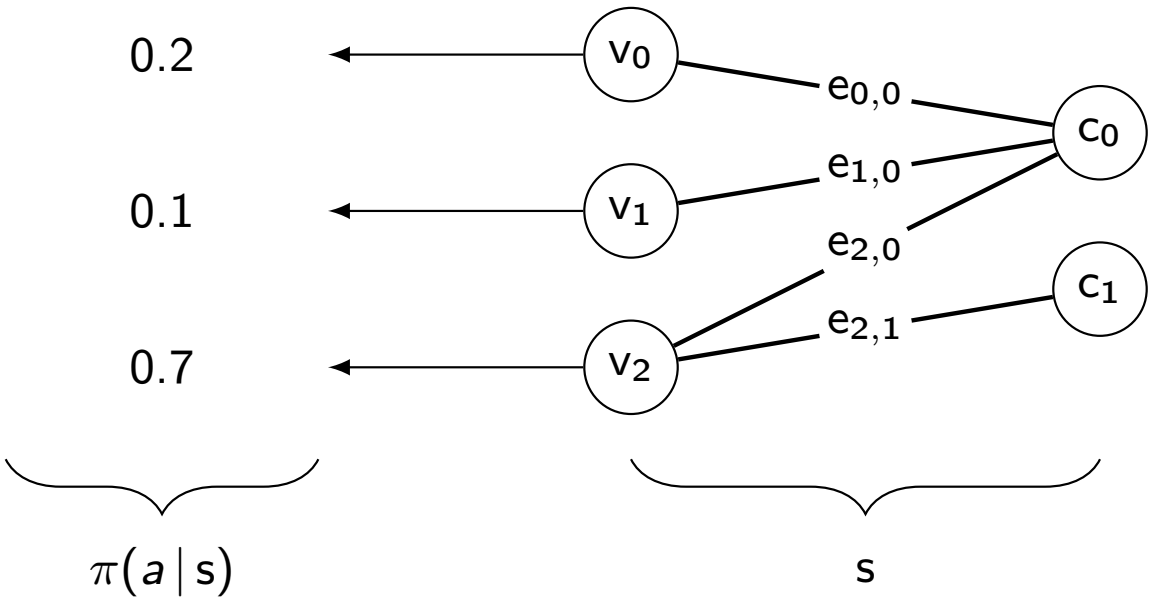
D. Selsam et al. (2019). Learning a SAT Solver from Single-Bit Supervision.

Branching Policy as a GCNN Model

Neighbourhood-based updates:

$\mathcal{C}$-side convolution $c_i \leftarrow f_{\mathcal{C}}\left(c_i, \sum_{j \in \mathcal{N}_i} g_{\mathcal{C}}(c_i, v_j, e_{i,j})\right)$

$\mathcal{V}$-side convolution $v_j \leftarrow f_{\mathcal{V}}\left(v_j, \sum_{i \in \mathcal{N}_j} g_{\mathcal{V}}(c_i, v_j, e_{i,j})\right)$



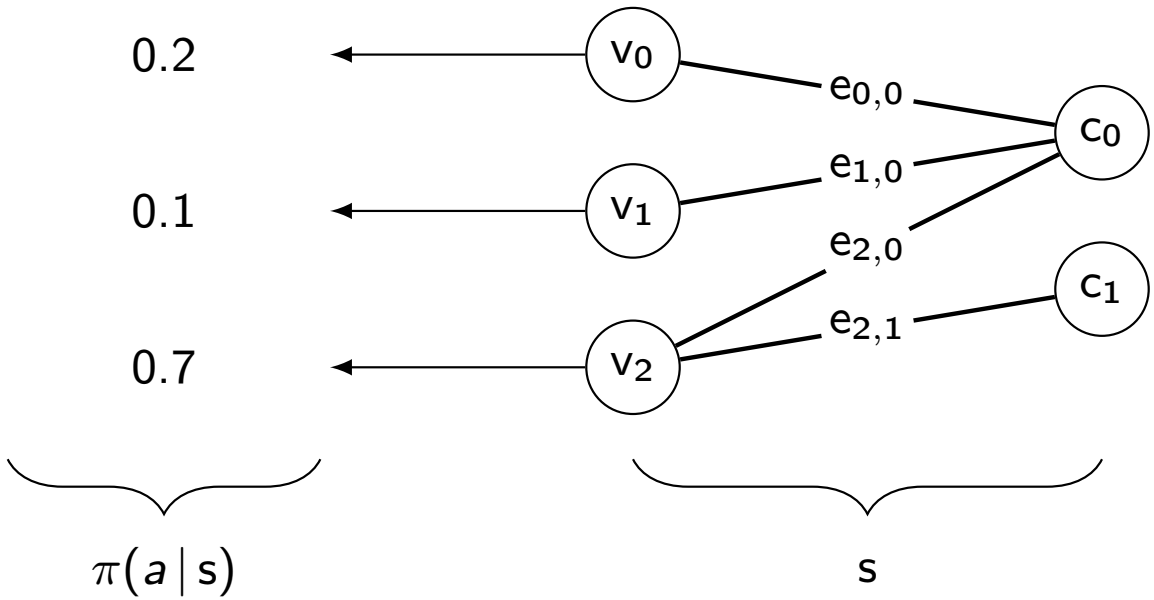
T. N. Kipf et al. (2016). Semi-Supervised Classification with Graph Convolutional Networks.

Branching Policy as a GCNN Model

Neighbourhood-based updates:

$\mathcal{C}$-side convolution $c_i \leftarrow f_{\mathcal{C}}\left(c_i, \sum_{j \in \mathcal{N}_i} g_{\mathcal{C}}(c_i, v_j, e_{i,j})\right)$

$\mathcal{V}$-side convolution $v_j \leftarrow f_{\mathcal{V}}\left(v_j, \sum_{i \in \mathcal{N}_j} g_{\mathcal{V}}(c_i, v_j, e_{i,j})\right)$



Natural model choice for graph-structured data

- permutation-invariance
- benefits from sparsity

T. N. Kipf et al. (2016). Semi-Supervised Classification with Graph Convolutional Networks.

Minimum set covering[3]

	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)

[3] E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.

Minimum set covering[3]

Model	Easy Time	Easy Wins	Easy Nodes	Medium Time	Medium Wins	Medium Nodes	Hard Time	Hard Wins	Hard 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.

[3] E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.

Minimum set covering[3]

	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 !

[3] E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.

Minimum set covering[3]

	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*.

[3] E. Balas et al. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study.

ML4CO: a growing field

Node selection

- [He et al., 2014]
- [Song, Lanka, Zhao, et al., 2018]

Variable selection

- [E. B. Khalil, Le Bodic, et al., 2016]
- [Hansknecht et al., 2018]
- [Balcan et al., 2018]
- [Gasse et al., 2019a]
- [Gupta et al., 2020]
- [Nair et al., 2020]

Cutting planes selection

- [Baltean-Lugojan et al., 2018]
- [Tang et al., 2019]

Primal heuristic selection

- [E. B. Khalil, Dilkina, et al., 2017]
- [Hendel et al., 2018]

Formulation selection

- [Bonami et al., 2018]

Neighborhood search heuristics

- [Ding et al., 2019]
- [Song, Lanka, Yue, et al., 2020]
- [Addanki et al., 2020]

Diving heuristics

- [Song, Lanka, Zhao, et al., 2018]
- [Yilmaz et al., 2020]
- [Nair et al., 2020]

Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers

Why Ecole ?

Poor reproducibility in the field

- closed-source solvers
- problem benchmarks
- evaluation metrics

A. Prouvost et al. (2020). Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers.

Why Ecole ?

Poor reproducibility in the field

- closed-source solvers
- problem benchmarks
- evaluation metrics

High bar of entry for newcomers

- low-level C/C++ code
- highly technical APIs even for OR experts

A. Prouvost et al. (2020). Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers.

Why Ecole ?

Poor reproducibility in the field

- closed-source solvers
- problem benchmarks
- evaluation metrics

High bar of entry for newcomers

- low-level C/C++ code
- highly technical APIs even for OR experts

Gap between the ML and OR communities

- amputated solvers raise criticism in the OR community
- OR experts employ basic ML models

$\Longrightarrow$ need for a *standard*, *open* platform based on a *state-of-the-art* solver

---

A. Prouvost et al. (2020). Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers.

Why Ecole ?

Poor reproducibility in the field

- closed-source solvers
- problem benchmarks
- evaluation metrics

Remove technical obstacles, so that we can focus on the interesting challenges !

High bar of entry for newcomers

- low-level C/C++ code
- highly technical APIs even for OR experts

Gap between the ML and OR communities

- amputated solvers raise criticism in the OR community
- OR experts employ basic ML models

$\Longrightarrow$ need for a *standard*, *open* platform based on a *state-of-the-art* solver

---

A. Prouvost et al. (2020). Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers.

The PO-MDP Formulation

Sequential control problem = **Markov decision process**

State = state of the branch-and-bound process (solver)

The PO-MDP Formulation

Sequential control problem = **Markov decision process**

State = state of the branch-and-bound process (solver)
Actions = variables, nodes, primal heuristics, cuts, preprocessing routines to select
Episode = solving an instance to completion

The PO-MDP Formulation

Sequential control problem = **Markov decision process**

State = state of the branch-and-bound process (solver)
Actions = variables, nodes, primal heuristics, cuts, preprocessing routines to select
Episode = solving an instance to completion
Probability of a trajectory $\tau \in (s_0, \ldots, s_T)$

$$\tau \sim \underbrace{p_{init}(s_0)}_{\text{initial state}} \prod_{t=0}^{\infty} \underbrace{\pi(a_t|s_t)}_{\text{next action}} \underbrace{p_{trans}(s_{t+1}|a_t, s_t)}_{\text{next state}}$$

The PO-MDP Formulation

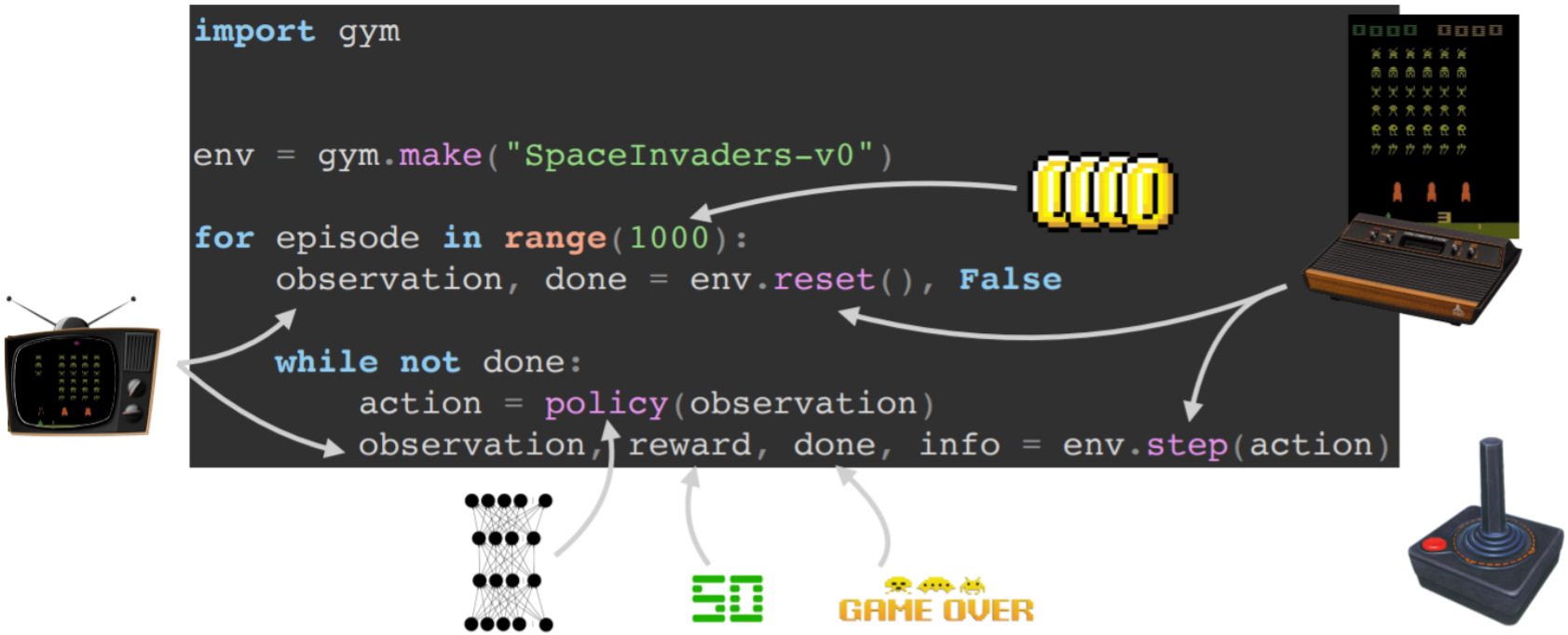
Sequential control problem = **Markov decision process**

State = state of the branch-and-bound process (solver)
Actions = variables, nodes, primal heuristics, cuts, preprocessing routines to select
Episode = solving an instance to completion
Probability of a trajectory $\tau \in (s_0, \ldots, s_T)$

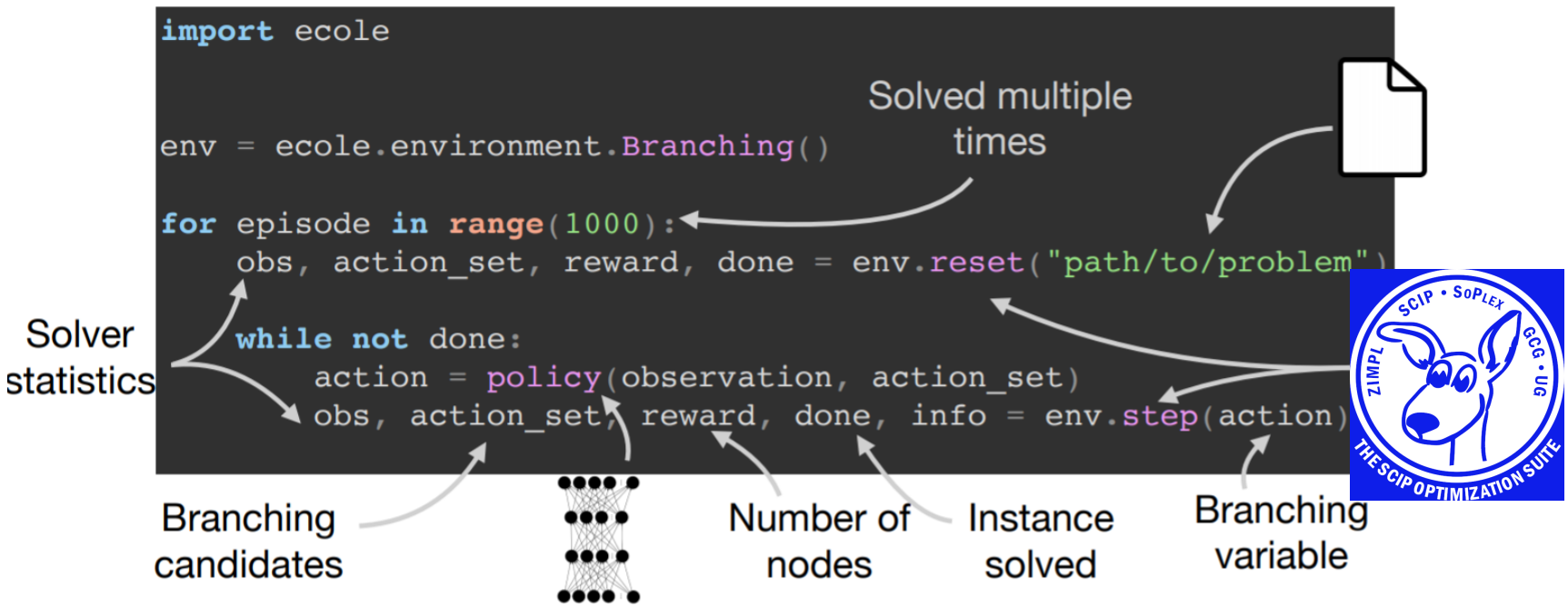
$$\tau \sim \underbrace{p_{init}(s_0)}_{\text{initial state}} \prod_{t=0}^{\infty} \underbrace{\pi(a_t|s_t)}_{\text{next action}} \underbrace{p_{trans}(s_{t+1}|a_t, s_t)}_{\text{next state}}$$

Partially-obervable-MDP: state $s \in \mathcal{S} \rightarrow$ observation $o \in \mathcal{O}$

OpenAI Gym API



Ecole API


import ecole
env = ecole.environment.Branching()
for episode in range(1000):
obs, action_set, reward, done = env.reset("path/to/problem")
while not done:
action = policy(observation, action_set)
obs, action_set, reward, done, info = env.step(action)
Solved multiple times
Solver statistics
Branching candidates
Number of nodes
Instance solved
Branching variable
SCIP • SoPLEX
ZIMPL
GCG • UG
THE SCIP OPTIMIZATION SUITE

What's in Ecole now ?

`https://doc.ecole.ai`

Environments:

- Solver configuration
- Branching (variable selection)
- Primal Search (feas. solutions)

Observations:

- Hutter2011 [Hutter et al., 2011]
- Khalil2016 [E. B. Khalil, Le Bodic, et al., 2016]
- Gasse2019 [Gasse et al., 2019a]
- Branching Scores (SB, Pseudocost)

Rewards:

- Solving Time
- NNodes (B&B tree size)
- LP Iterations
- Primal and dual integral

Instance Generators:

- Set Covering [Balas et al., 1980]
- Comb. Auction [Leyton-Brown et al., 2000]
- Facility Location [Cornuejols et al., 1991]
- Independent Set [Bergman et al., 2016]

Machine Learning for Global Optimization

Mixed-Integer Nonlinear Program (MINLP)

$$\begin{aligned} \arg\min_{x} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0, \\ & l \leq x \leq u, \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}. \end{aligned}$$

- $f : \mathbb{R}^n \to \mathbb{R}$ the objective function
- $g : \mathbb{R}^n \to \mathbb{R}^m$ the constraint coefficient matrix
- f, g can be nonconvex
- $l, u \in \mathbb{R}^n$ the lower and upper variable bounds
- $p \leq n$ integer variables

Wide applications of global optimization

Energy systems optimization

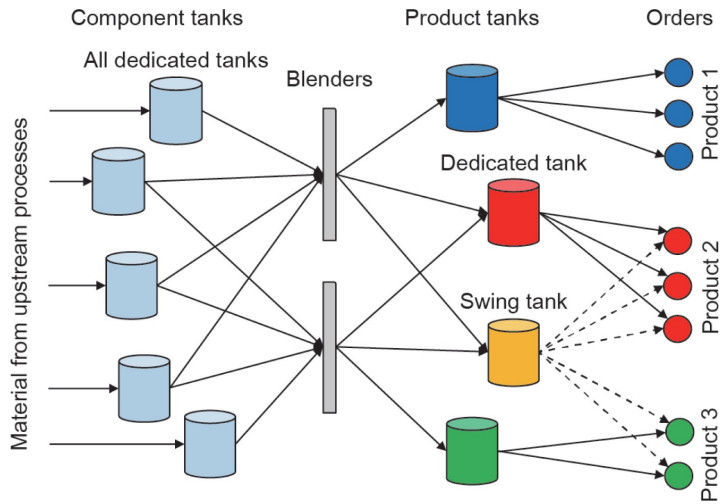
- AC optimal power flow
- Pooling problem

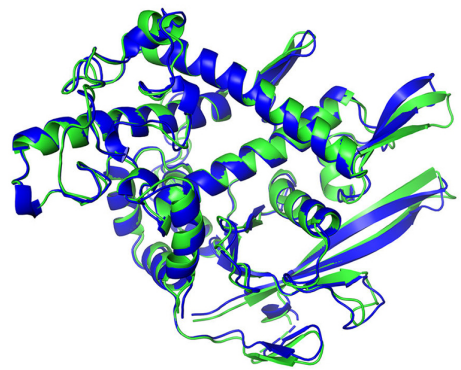
Chemical engineering

- Heat exchanger design
- Design of molecules

Systems biology

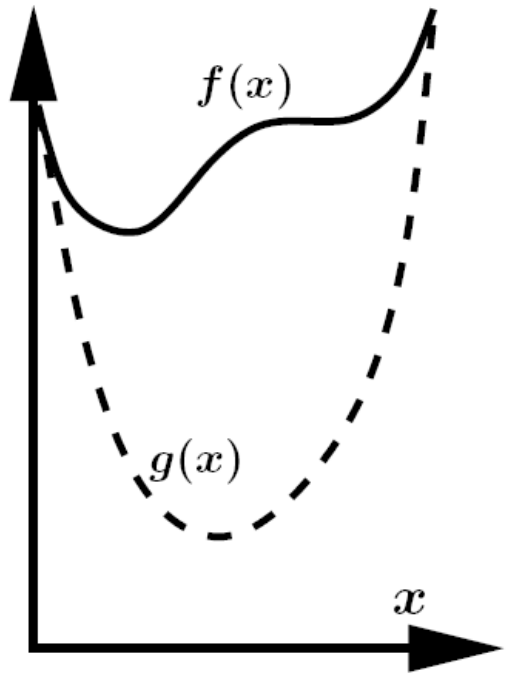
- Protein structure prediction
- Metabolic engineering





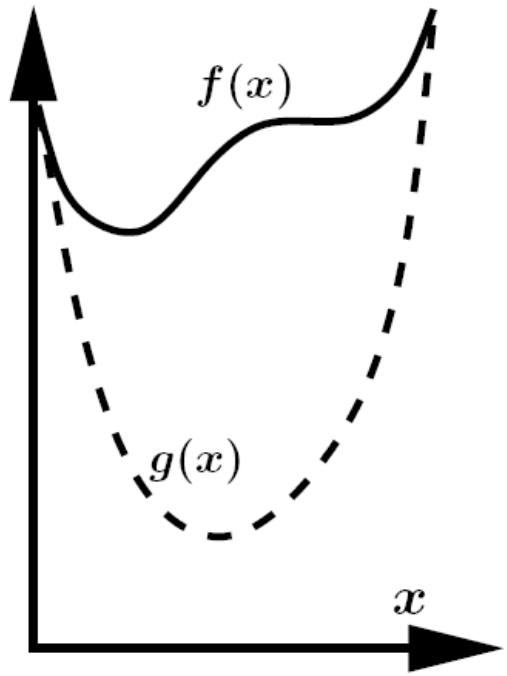
Convex relaxations for nonconvex functions

Convex function $g(x)$ underestimate nonconvex function $f(x)$



Convex relaxations for nonconvex functions

Convex function $g(x)$ underestimate nonconvex function $f(x)$

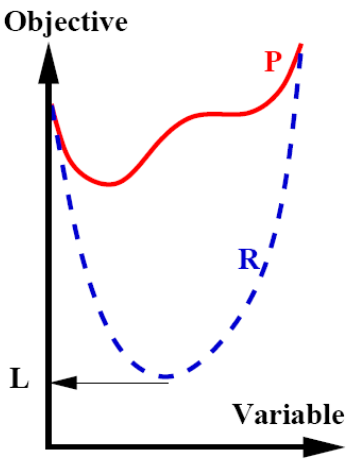


Example: quadratic functions $x^\top Qx$, Q not p.s.d

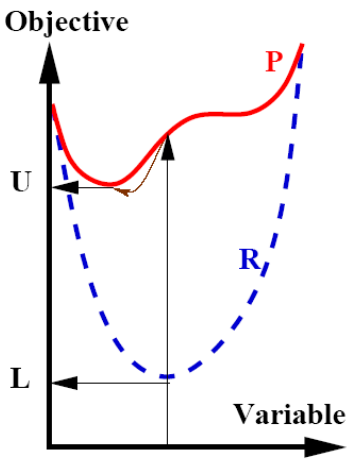
- McCormick inequalities
- Reformulation linearization technique (RLT)
- Semidefinite programming (SDP)

Spatial branch-and-bound

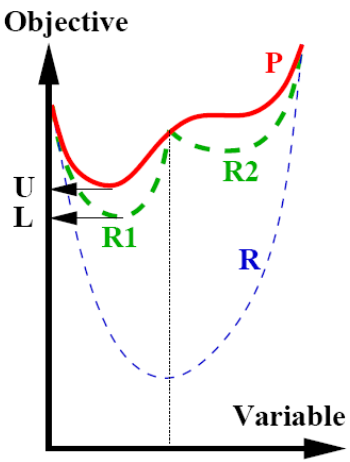
Need to perform spatial branching on the continuous variable to obtain global optimality



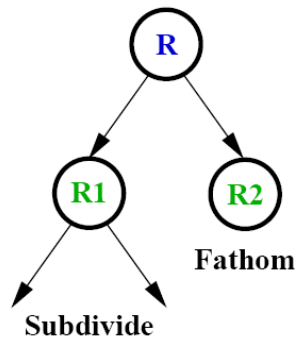
a. Lower Bounding



b. Upper Bounding



c. Domain Subdivision



d. Search Tree

Machine learning for global optimization

Spatial branching

- improve the standard spatial branching rules [Ghaddar et al., 2022]

Machine learning for global optimization

Spatial branching

- improve the standard spatial branching rules [Ghaddar et al., 2022]

Cut selection

- select sparse SDP cuts [Baltean-Lugojan et al., 2019]

Machine learning for global optimization

Spatial branching

- improve the standard spatial branching rules [Ghaddar et al., 2022]

Cut selection

- select sparse SDP cuts [Baltean-Lugojan et al., 2019]

Initial formulation

- whether to linearize an MIQP or not [Bonami et al., 2022]

Machine learning for global optimization

Spatial branching

- improve the standard spatial branching rules [Ghaddar et al., 2022]

Cut selection

- select sparse SDP cuts [Baltean-Lugojan et al., 2019]

Initial formulation

- whether to linearize an MIQP or not [Bonami et al., 2022]

Learning to select relaxations in a branch and bound algorithm

- This talk.

Quadratic Unconstrained Binary Optimization(QUBO)

$$\min x^\top Q x$$

$$x \in \{0,1\}^n$$

A number of applications can be represented as QUBOs.

- Maxcut
- Quadratic Stable Set Problem
- Graph Coloring
- Partition Problem

Quadratic Unconstrained Binary Optimization(QUBO)

$$\min x^\top Q x$$

$$x \in \{0,1\}^n$$

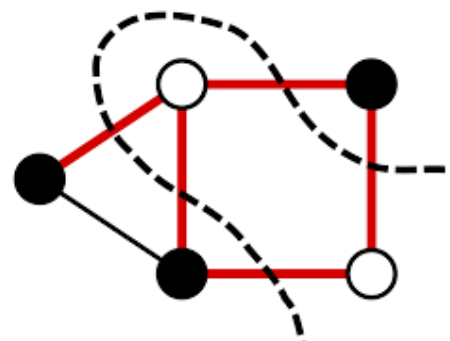
A number of applications can be represented as QUBOs.

- Maxcut
- Quadratic Stable Set Problem
- Graph Coloring
- Partition Problem

Maxcut formulation

Given graph $G = (V, E)$ & matrix $W \in S^n$ with $w_{i,j} \neq 0$ if $(i,j) \in E$ & 0 otherwise

$$f^* = \max_{x \in \{-1,+1\}^n} \sum_{(i,j) \in E} w_{i,j} \left(\frac{1 - x_i x_j}{2} \right)$$



Branch-and-bound Algorithm for QUBO

- In order to solve QUBO to global optimality, a branch-and-bound algorithm is needed.

Branch-and-bound Algorithm for QUBO

- In order to solve QUBO to global optimality, a branch-and-bound algorithm is needed.
- **Lower bound**
  Different relaxations can be used in the branch-and-bound algorithm
  1. Linear Programming (LP)
  2. Semidefinite programming (SDP)
  3. Hybrid LP & SDP

Branch-and-bound Algorithm for QUBO

- In order to solve QUBO to global optimality, a branch-and-bound algorithm is needed.
- **Lower bound**
  Different relaxations can be used in the branch-and-bound algorithm
  1. Linear Programming (LP)
  2. Semidefinite programming (SDP)
  3. Hybrid LP & SDP
- **Upper bound**
  Heuristics like Goemans and Williamson (1995)

Branch-and-bound Algorithm for QUBO

- In order to solve QUBO to global optimality, a branch-and-bound algorithm is needed.
- **Lower bound**
  Different relaxations can be used in the branch-and-bound algorithm
  1. Linear Programming (LP)
  2. Semidefinite programming (SDP)
  3. Hybrid LP & SDP
- **Upper bound**
  Heuristics like Goemans and Williamson (1995)
- We will focus on how to effectively use lower bound in this talk

LP relaxations

Let $X \in S^n$ represents $xx^\top$. The QUBO can be relaxed as

$$\min_{X, x \in [0,1]} \quad < Q, X >$$

$$\texttt{s.t.} \quad \texttt{valid linear inequalities}$$

LP relaxations

Let $X \in S^n$ represents $xx^\top$. The QUBO can be relaxed as

$$\min_{X, x \in [0,1]} \quad < Q, X >$$

$$\texttt{s.t.} \quad \texttt{valid linear inequalities}$$

- McCormick inequalities

$$X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_i \quad \forall i, j$$

LP relaxations

Let $X \in S^n$ represents $xx^\top$. The QUBO can be relaxed as

$$\min_{X, x \in [0,1]} \quad < Q, X >$$
$$\texttt{s.t.} \quad \texttt{valid linear inequalities}$$

- McCormick inequalities

$$X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_i \quad \forall i, j$$

- Odd cycle inequalities derived from the CUT polytope, e.g., triangle inequalities.

$$\left.\begin{aligned} \chi_{ij} + \chi_{ik} + \chi_{jk} &\leq 2 \\ \chi_{ij} - \chi_{ik} - \chi_{jk} &\leq 0 \\ -\chi_{ij} + \chi_{ik} - \chi_{jk} &\leq 0 \\ -\chi_{ij} - \chi_{ik} + \chi_{jk} &\leq 0 \end{aligned}\right\} \text{ for all distinct } i, j, k \in V$$

where $\chi_e \in \{0,1\}^{|E|}$ whether edge e is in the cut. $\chi_{ij} = x_i + x_j - X_{ij} - X_{ji}$

SDP relaxations

$$X = xx^\top \overset{\text{relax}}{\Longrightarrow} X \succeq xx^\top \overset{\text{Schur complement}}{\Longleftrightarrow} \begin{bmatrix} 1 & x^\top \\ x & X \end{bmatrix} \succeq 0$$

SDP relaxations

$$X = xx^\top \overset{\text{relax}}{\Longrightarrow} X \succeq xx^\top \overset{\text{Schur complement}}{\Longleftrightarrow} \begin{bmatrix} 1 & x^\top \\ x & X \end{bmatrix} \succeq 0$$

- Basic SDP relaxation of QUBO can be solved by interior point solvers efficiently
- For large problems, basic SDP relaxation bounds improve slowly in the branch and bound algorithm.

SDP relaxations

$$X = xx^\top \overset{\text{relax}}{\Longrightarrow} X \succeq xx^\top \overset{\text{Schur complement}}{\Longleftrightarrow} \begin{bmatrix} 1 & x^\top \\ x & X \end{bmatrix} \succeq 0$$

- Basic SDP relaxation of QUBO can be solved by interior point solvers efficiently
- For large problems, basic SDP relaxation bounds improve slowly in the branch and bound algorithm.

SDP + linear relaxations

- Stronger than SDP or linear relaxations alone. Reduce the number of nodes in branch-and-bound.
- State-of-the-art QUBO solvers, e.g., BiqMac, BiqCrunch, use SDP + polyhedral relaxations.
- Stronger but also more expensive to solve.

Motivations of our work

Trade-offs in applying different relaxations

- LP relaxation
  - easy to solve; weak bounds
  - Enumerate more nodes in the same amount of time
- SDP + LP relaxations
  - stronger bounds; expensive to solve
  - Higher chance of fathoming a node.

Motivations of our work

Trade-offs in applying different relaxations

- LP relaxation
  - easy to solve; weak bounds
  - Enumerate more nodes in the same amount of time
- SDP + LP relaxations
  - stronger bounds; expensive to solve
  - Higher chance of fathoming a node.

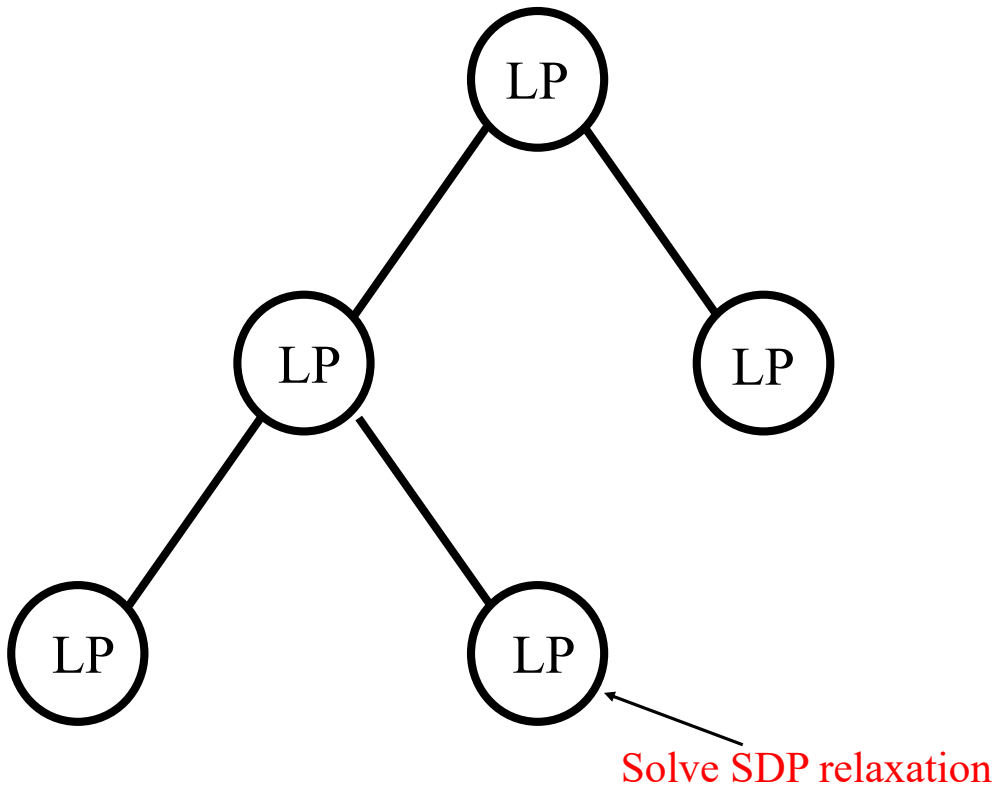
Questions we aim to address

- Which relaxation should we use at each node of the branch-and-bound algorithm?
- Is it worth solving a more expensive relaxation (SDP-based relaxation) at each node?

Hybrid bounding scheme

We adopt the framework of Furini and Traversi (2013).

- By default, we solve the LP relaxation of the problem at each node.
- The SDP relaxation is only solved if we believe a node is likely to be pruned by SDP bound.



Simple setting

McCormick relaxation v.s. McCormick relaxation + SDP

- At each node, we solve the LP relaxation with McCormick inequalities.

$$(LP) \quad \min_{X, x \in [0,1]} \quad < Q, X >$$
$$\texttt{s.t.} \quad X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_i \quad \forall i, j$$

- The SDP relaxation is solved conditionally.

$$(SDP) \quad \min_{X, x \in [0,1]} \quad < Q, X >$$
$$\texttt{s.t.} \quad X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_i \quad \forall i, j$$
$$\begin{bmatrix} 1 & x^\top \\ x & X \end{bmatrix} \succeq 0$$

Predicting SDP bound using Heuristic/Machine learning

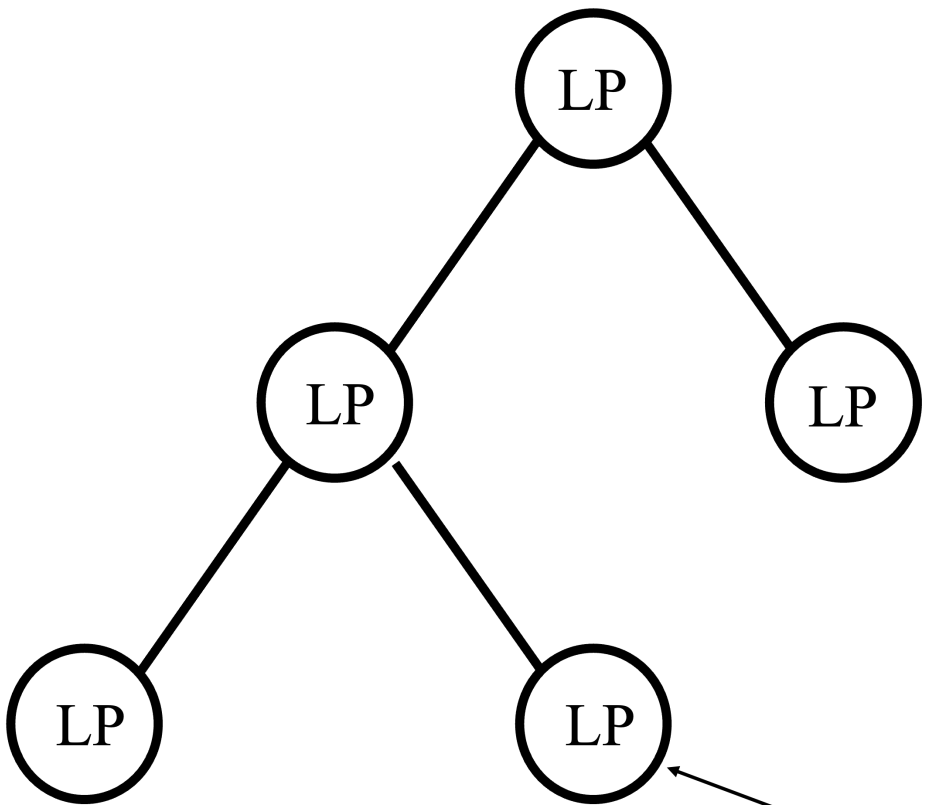
- If we can predict whether the SDP bound can fathom a node without solving an SDP using a heuristic/machine learning, the computational time can be saved for the nodes that cannot be fathomed.

Predicting SDP bound using Heuristic/Machine learning

- If we can predict whether the SDP bound can fathom a node without solving an SDP using a heuristic/machine learning, the computational time can be saved for the nodes that cannot be fathomed.
- This work: *learning to predict bounds*

A simple observation

- If a feasible solution of the SDP relaxation can be found that is smaller than the incumbent solution, the SDP bound cannot fathom and node. We can skip solving the SDP.
- How to quickly find a near optimal solution to the SDP?

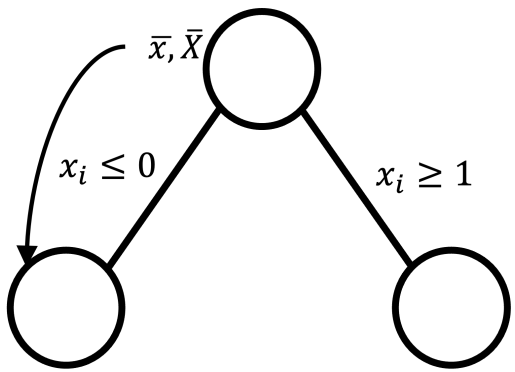


$\overline{LB} \geq LB(SDP), \overline{LB} \leq UB$

The SDP bound cannot fathom the node

QCP-based approach

- Suppose SDP is solved at a node with optimal solution $\bar{x}, \bar{X}$. We solve the child nodes with x, X fixed to $\bar{x}, \bar{X}$ except the ith row and column.



$$\min < Q, X >$$

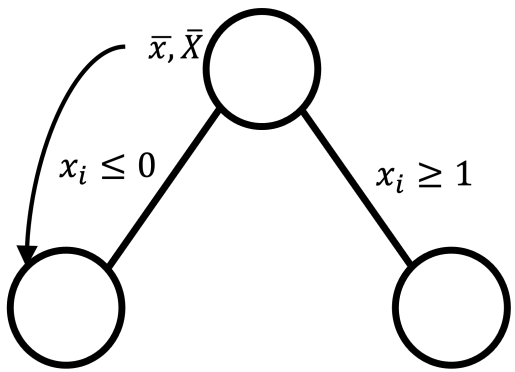
$$X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_j \quad \forall j$$

$$0 \leq x_i \leq 1$$

$$\begin{bmatrix} X_{i,i} & X_{[i,1:i-1]} & x_i & X_{[i,i+1:]} \\ X_{[1:i-1,i]} & \bar{X}_{1:i-1,1:i-1} & \bar{x}_{1:i-1} & \bar{X}_{[1:i-1,i+1:]} \\ x_i & \bar{x}_{1:i-1}^\top & 1 & \bar{x}_{i+1:}^\top \\ X_{[i+1:,i]} & \bar{X}_{[i+1:,1:i-1]} & \bar{x}_{i+1:} & \bar{X}_{[i+1:,i+1:]} \end{bmatrix} \succeq 0$$

QCP-based approach

- Suppose SDP is solved at a node with optimal solution $\bar{x}, \bar{X}$. We solve the child nodes with x, X fixed to $\bar{x}, \bar{X}$ except the ith row and column.



$$\min < Q, X >$$

$$X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_j \quad \forall j$$

$$0 \leq x_i \leq 1$$

$$\begin{bmatrix} X_{i,i} & X_{[i,1:i-1]} & x_i & X_{[i,i+1:]} \\ X_{[1:i-1,i]} & \bar{X}_{1:i-1,1:i-1} & \bar{x}_{1:i-1} & \bar{X}_{[1:i-1,i+1:]} \\ x_i & \bar{x}_{1:i-1}^\top & 1 & \bar{x}_{i+1:}^\top \\ X_{[i+1:,i]} & \bar{X}_{[i+1:,1:i-1]} & \bar{x}_{i+1:} & \bar{X}_{[i+1:,i+1:]} \end{bmatrix} \succeq 0$$

- The SDP can be reformulated as a convex QCP by taking the Schur complement.

QCP-based approach

$$\min < Q, X >$$

$$X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_j \quad \forall j$$

$$0 \leq x_i \leq 1$$

$$\begin{bmatrix} X_{i,i} & X_{[i,1:i-1]} & x_i & X_{[i,i+1:]} \\ X_{[1:i-1,i]} & \bar{X}_{1:i-1,1:i-1} & \bar{x}_{1:i-1} & \bar{X}_{[1:i-1,i+1:]} \\ x_i & \bar{x}_{1:i-1}^\top & 1 & \bar{x}_{i+1:}^\top \\ X_{[i+1:,i]} & \bar{X}_{[i+1:,1:i-1]} & \bar{x}_{i+1:} & \bar{X}_{[i+1:,i+1:]} \end{bmatrix} \succeq 0$$

QCP-based approach

$$\min < Q, X >$$

$$X_{ij} \geq 0, \quad X_{ij} \geq x_i + x_j - 1, \quad X_{ij} \leq x_j \quad \forall j$$

$$0 \leq x_i \leq 1$$

$$\begin{bmatrix} X_{i,i} & X_{[i,1:i-1]} & x_i & X_{[i,i+1:]} \\ X_{[1:i-1,i]} & \bar{X}_{1:i-1,1:i-1} & \bar{x}_{1:i-1} & \bar{X}_{[1:i-1,i+1:]} \\ x_i & \bar{x}_{1:i-1}^\top & 1 & \bar{x}_{i+1:}^\top \\ X_{[i+1:,i]} & \bar{X}_{[i+1:,1:i-1]} & \bar{x}_{i+1:} & \bar{X}_{[i+1:,i+1:]} \end{bmatrix} \succeq 0$$

Equivalent to a convex QCP with $\mathcal{O}(n)$ variables and constraints

$$X_{i,i} - \begin{bmatrix} X_{[1:i-1,i]} \\ x_i \\ X_{[i+1:,i]} \end{bmatrix}^\top \bar{X}^{-1} \begin{bmatrix} X_{[1:i-1,i]} \\ x_i \\ X_{[i+1:,i]} \end{bmatrix} \geq 0$$

Combine QCP with Machine learning

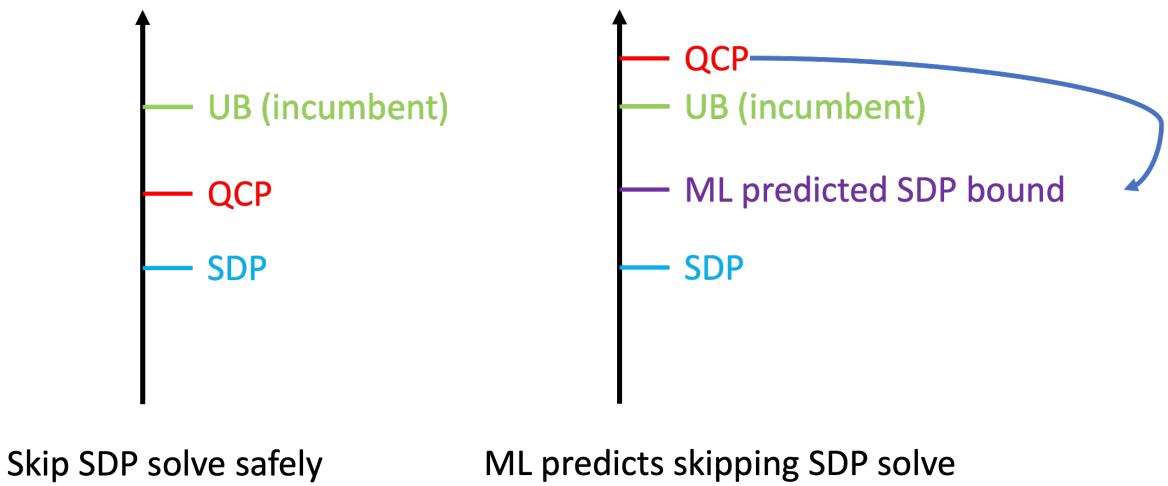
- If the QCP value is smaller than the incumbent solution, we can safely skip the SDP solve.
- Can we be more aggressive in skipping the nodes?

Combine QCP with Machine learning

- If the QCP value is smaller than the incumbent solution, we can safely skip the SDP solve.
- Can we be more aggressive in skipping the nodes?

Estimating the "true" SDP bounds using ML

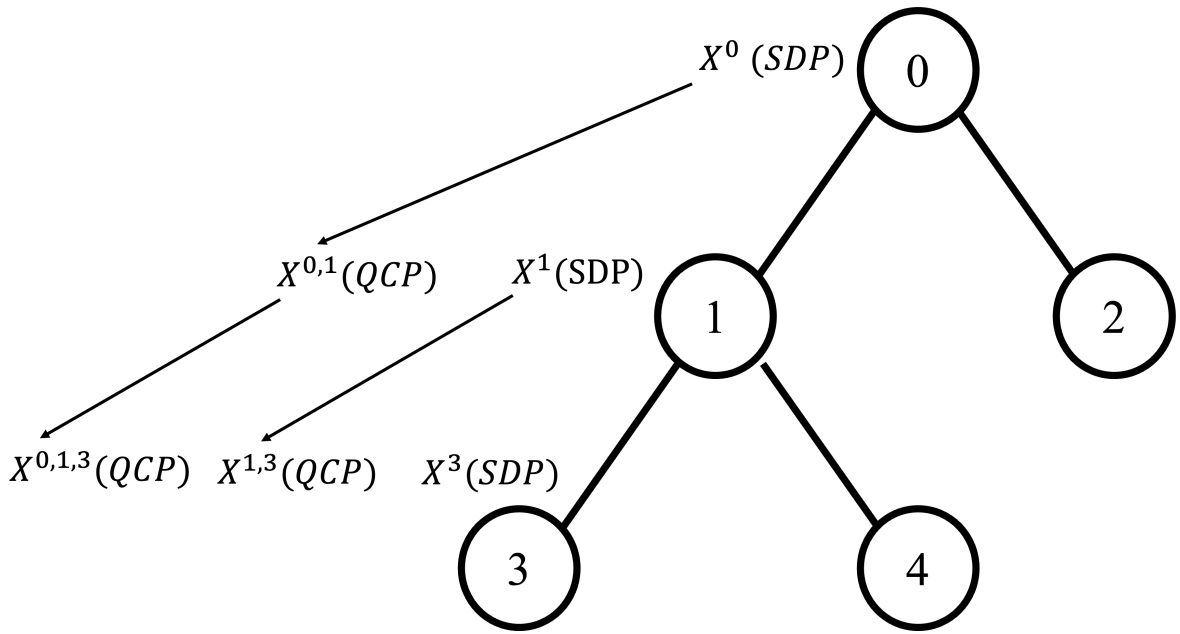
- Idea: we can estimate the "suboptimality" of QCP solve from data
- If we have an accurate estimation of the suboptimality, we might skip the SDP solve even if the QCP value is greater than the incumbent.



Skip SDP solve safely

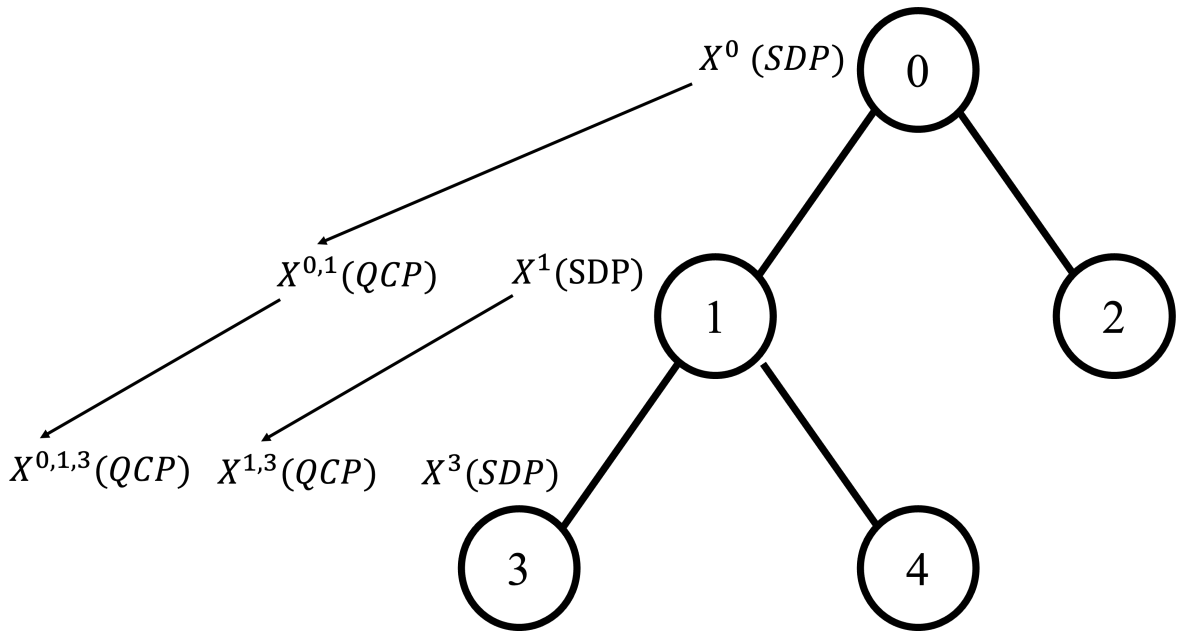
ML predicts skipping SDP solve

Data collection



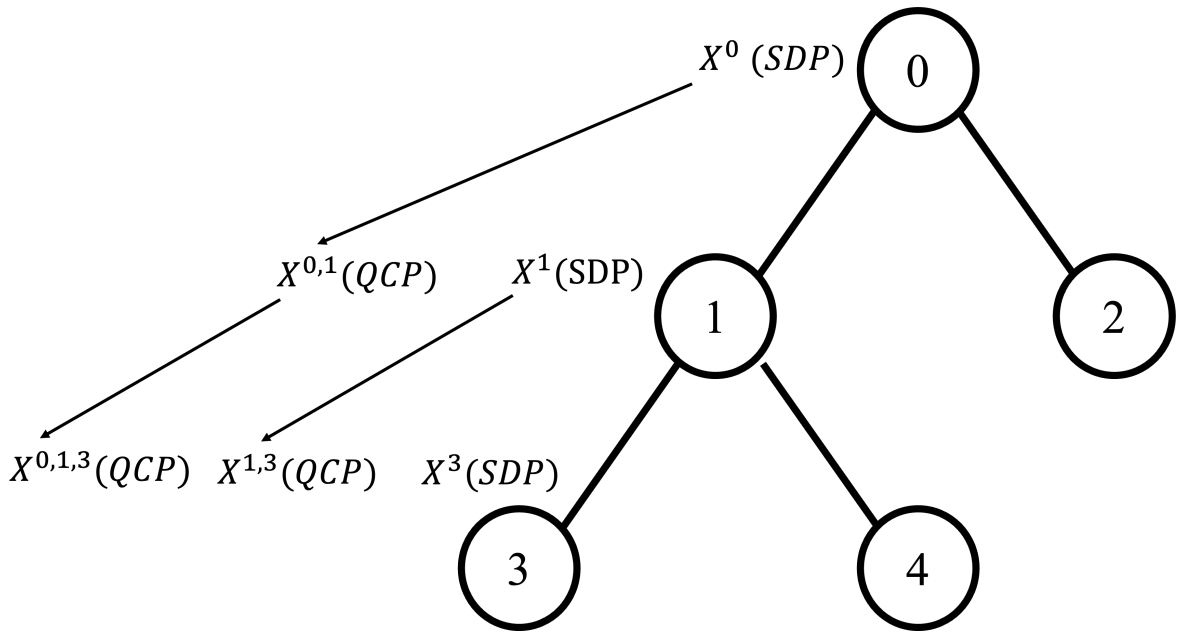
- SDP is solved at each node

Data collection



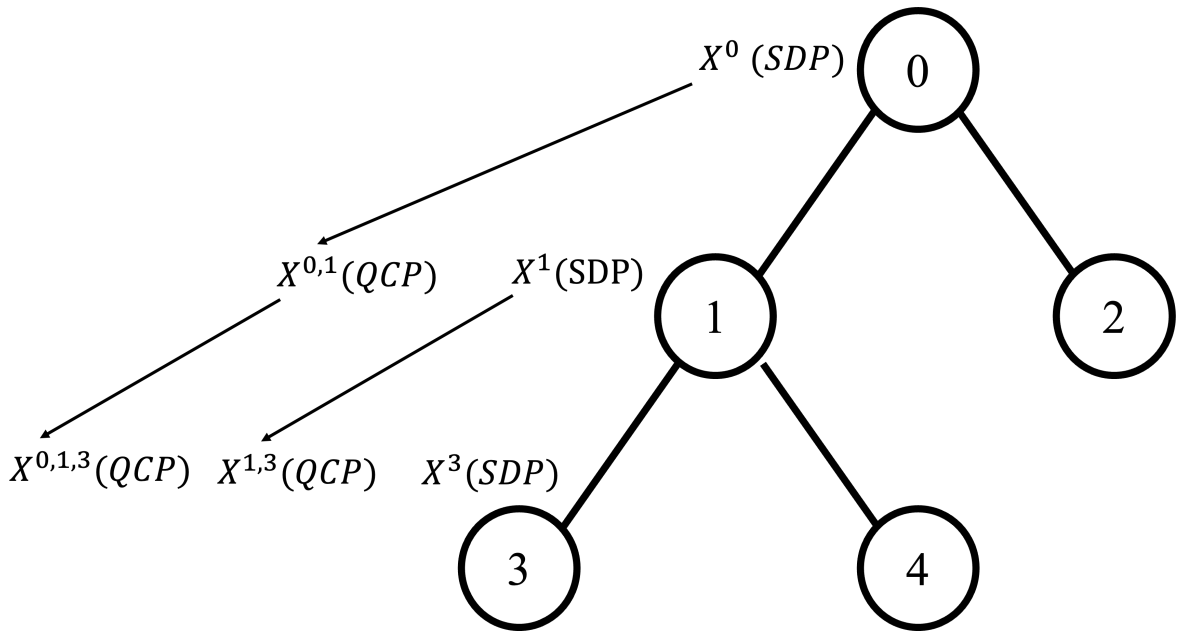
- SDP is solved at each node
- QCP is solved based on the SDP/QCP solves of its parent node

Data collection



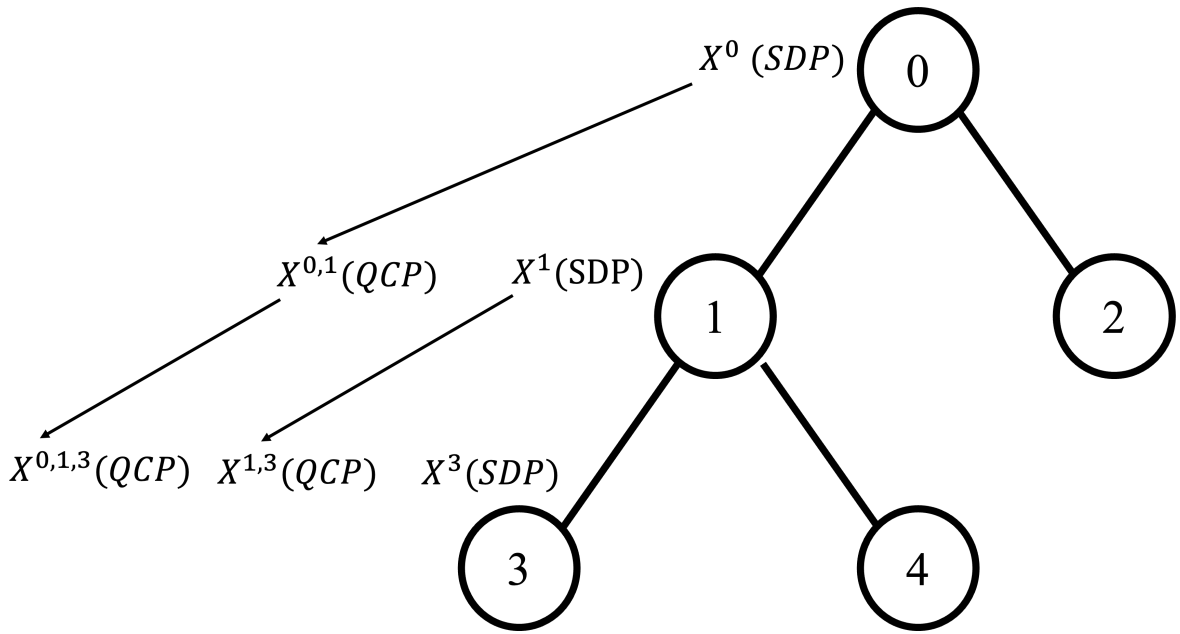
- SDP is solved at each node
- QCP is solved based on the SDP/QCP solves of its parent node
- `napprox` denotes the number of QCP approximations made

Data collection



- SDP is solved at each node
- QCP is solved based on the SDP/QCP solves of its parent node
- `napprox` denotes the number of QCP approximations made
- For $X^{0,1,3}$, `napprox = 2`. For $X^{1,3}$ and $X^{0,1}$, `napprox = 1`

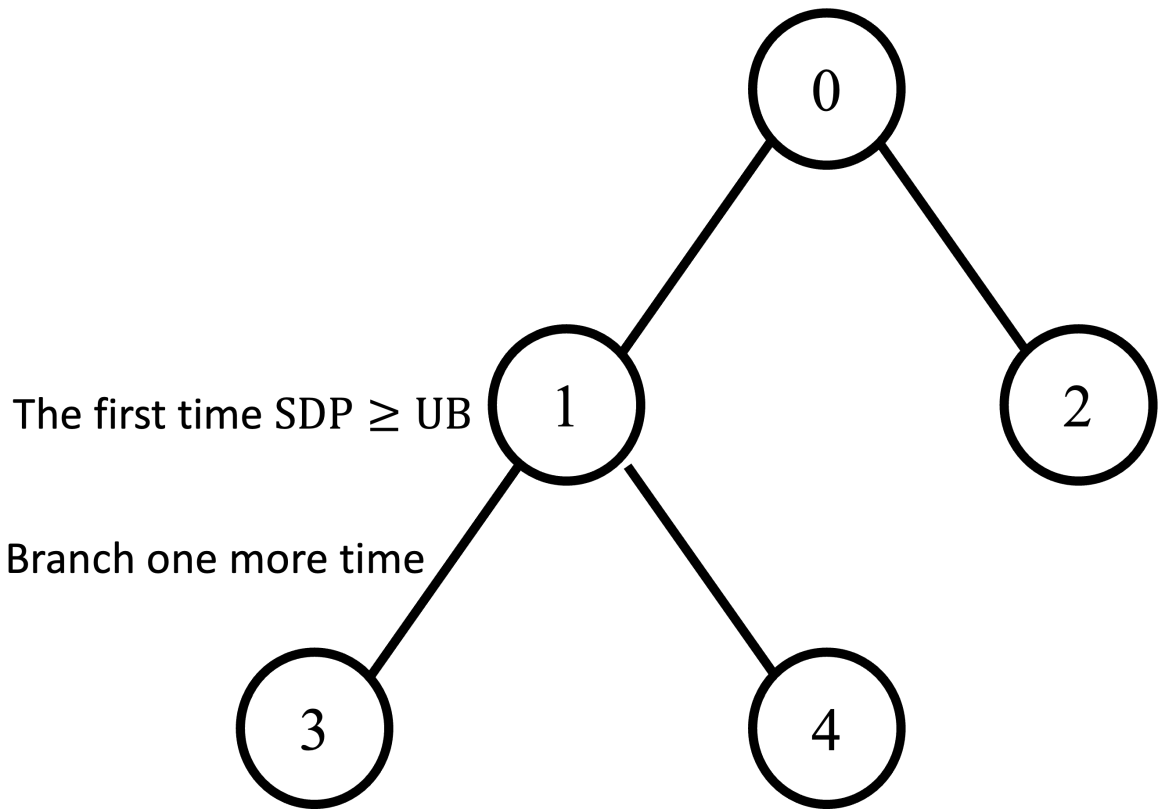
Data collection



- SDP is solved at each node
- QCP is solved based on the SDP/QCP solves of its parent node
- `napprox` denotes the number of QCP approximations made
- For $X^{0,1,3}$, `napprox = 2`. For $X^{1,3}$ and $X^{0,1}$, `napprox = 1`
- we only consider `napprox`≤ 3

Data collection continued

- Since we can make false predictions, We branch one more time after the node can be fathomed by SDP bound during data collection.
- The training data is collected on 36 maxcut instances of size 60 with densities ranging from 10% to 90%.



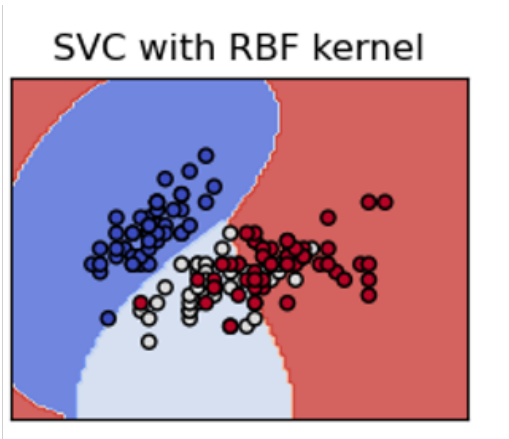
Feature engineering

Table: Features for each QCP solved

obj_qcp - obj_fea	QCP and incumbent solution difference
obj_fea - obj_sdp_root	incumbent solution and root node SDP difference
depth	node depth
napprox	number of QCP approximations
geo_mean	geometric mean of number of branches on 1 and 0
density	density of the maxcut adjacency matrix

SVM classifier

- With the features, we predict whether a node can be fathomed by SDP bound or not. (0/1 labels)
- We use SVC from scikit-learn that implements a Support Vector Machine (SVM) classifier with radial basis function (rbf) kernel.



SVM classifier

- With the features, we predict whether a node can be fathomed by SDP bound or not. (0/1 labels)
- We use SVC from scikit-learn that implements a Support Vector Machine (SVM) classifier with radial basis function (rbf) kernel.

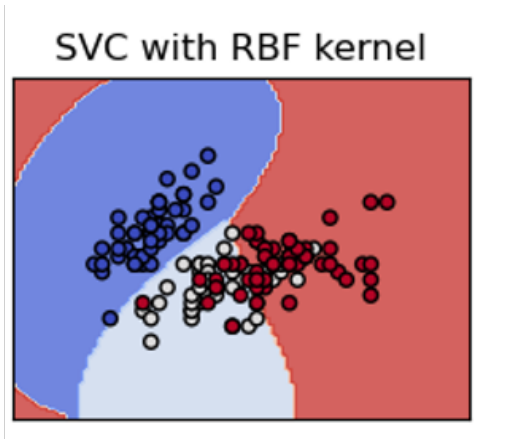
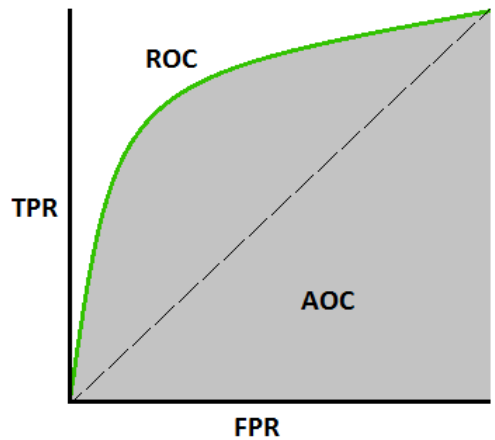


Table: Prediction statistics

	Accuracy	Precision	Recall
SVM	93.58%	94.93%	95.15%

TPR v.s. FPR

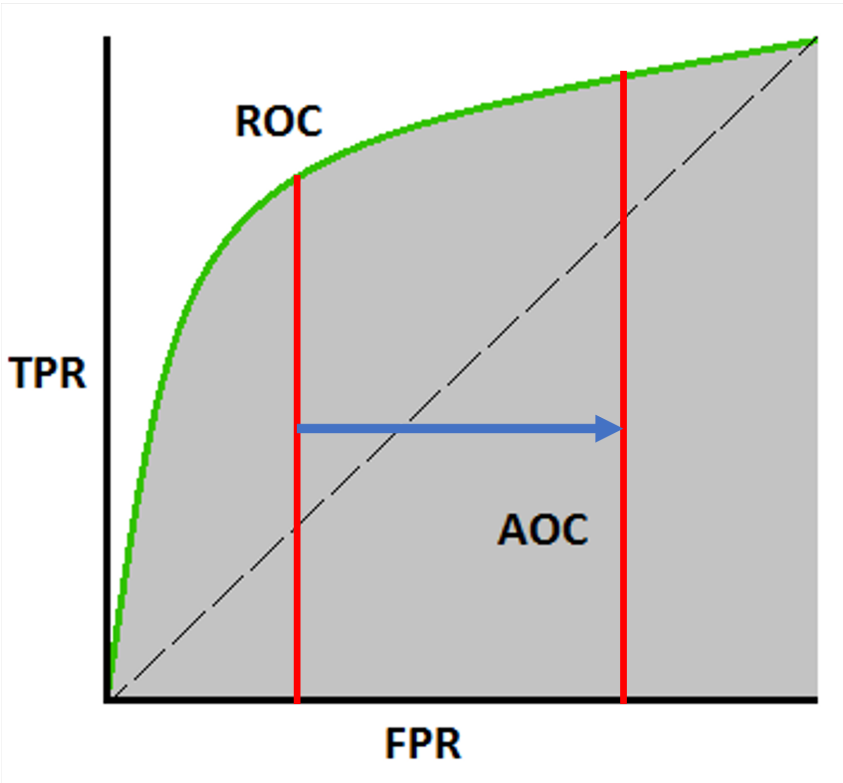
- TPR (recall): true positive rate
  $$\frac{\#\text{ nodes predicted to be fathomed by SDP that is true}}{\#\text{node that can be fathomed by SDP}}$$
- FPR: false positive rate
- We want to increase recall such that we do not miss many nodes that can be fathomed by SDP bounds. Otherwise, more nodes will be created and have to be solved.



TPR v.s. FPR

move the decision boundary to have higher recall

- Set the decision boundary threshold such that $8TPR - FPR$ is maximized



Testing results

- Increasing the recall also decreases accuracy and precision

Table: Prediction statistics

	Accuracy	Precision	Recall
SVM	93.58%	94.93%	95.15%
SVM with high recall	92.40%	90.30%	98.86%

Numerical results

- Average over 9 maxcut instances with densities 10%-90%
- The reduction in nodes and time can be different
- QCP SVM with high recall performs the best

Table: Node and time reduction compared with solving SDP at each node

	QCP	QCP SVM	QCP SVM high recall
# SDP solve reduction	19%	26%	28%
walltime reduction	16%	8%	20%

Convex Approximation for AC-OPF Using a Feasibility-Enhanced Neural Network

AC Optimal Power Flow (AC-OPF)

- The AC-OPF (or some approximation thereof) is the cornerstone of the operation and planning of power systems and is generally solved multiple times a day by power system operators worldwide.
- Its goal is to determine the most economical production levels of generating units to supply the demand while satisfying physical and engineering constraints.
  - **Physical constraints** model the nonconvex governing physical laws, Ohm's law and Kirchhoff law, known as *power flow equations*.
  - **Engineering constraints** model voltage, angle difference, transmission, and generation limits.

AC Optimal Power Flow (AC-OPF)

- The generating units and demands are distributed throughout a network, which is composed of
  - **Nodes**: Known as buses
  - **Edges**: Representing mainly transmission lines
  - The matrix encoding the network information is known as *nodal admittance matrix*.
- Every node is characterized by a complex voltage and a complex net power injection.
- Each generating unit and demand is connected to a certain node.
- In this talk, we consider that complex voltages are represented in rectangular coordinates, which allows us to formulate the problem as a nonconvex QCQP.

AC Optimal Power Flow Formulation

The AC-OPF can be formulated as a nonconvex quadratically constrained optimization problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & Ax = b, \\ & g_i(x) \leq 0,\ i = 1, \ldots, m_2 \\ & x \in \mathbb{R}^n. \end{aligned}$$

- $f : \mathbb{R}^n \to \mathbb{R}$ the convex objective function
- $A \in \mathbb{R}^{n \times m_1}$ the constraint coefficient matrix
- $b \in \mathbb{R}^{m_1}$ the constraint coefficient vector
- $g : \mathbb{R}^n \to \mathbb{R}$ the quadratic (convex and nonconvex) constraints

The AC-OPF is known to be NP-hard [Bienstock et al., 2019].

Relaxations and approximations for AC-OPF

Substantial efforts have been devoted to finding tractable surrogates

Relaxations

- Provide lower bounds.
- Infeasibility certificates.
- Linear: Copper plate and Network flow [Coffrin, H. Hijazi, et al., 2016]
- Second-order conic: [Jabr, 2006; Kocuk et al., 2016]
- Quadratic convex: [Coffrin, H. L. Hijazi, et al., 2016]
- Semidefinite: [Bai et al., 2008]

Approximations

- Based on two ideas:
  1. Engineering assumptions (line parameters, voltage magnitudes, angle differences)
  2. Linearization/convexification points.
- Linear: LPAC [Coffrin and Hentenryck, 2014], IV-Flow [O'Neill et al., 2012; Castillo et al., 2016]
- Convex: SOC [Jabr, 2007], QPAC [Coffrin, H. Hijazi, et al., 2015], Our work!

For a comprehensive review, [Molzahn et al., 2019].

AC-OPF Reformulation

Reformulating the AC-OPF as a Difference-of-Convex-Functions Program

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & Ax = b, \\ & g_i(x) \leq 0,\ i = 1, \ldots, m_2 \\ & x \in \mathbb{R}^n \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & Ax = b, \\ & \hat{g}_i(x) - \check{g}_i(x) \leq 0,\ i = 1, \ldots, m_2, \\ & x \in \mathbb{R}^n, \end{aligned}$$

where $\hat{g}(x)$, and $\check{g}(x)$ are convex functions.

- The reformulated problem is still nonconvex due to $\check{g}(x)$.
- Convexify using a first-order Taylor series approximation [Yuille et al., 2003; Lipp et al., 2016].

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & Ax = b, \\ & \hat{g}(x) - \check{g}(\tilde{x}) - \nabla \check{g}(\tilde{x})^{\top}(x - \tilde{x}) \leq 0, \\ & x \in \mathbb{R}^n. \end{aligned}$$

The QCAC Approximation

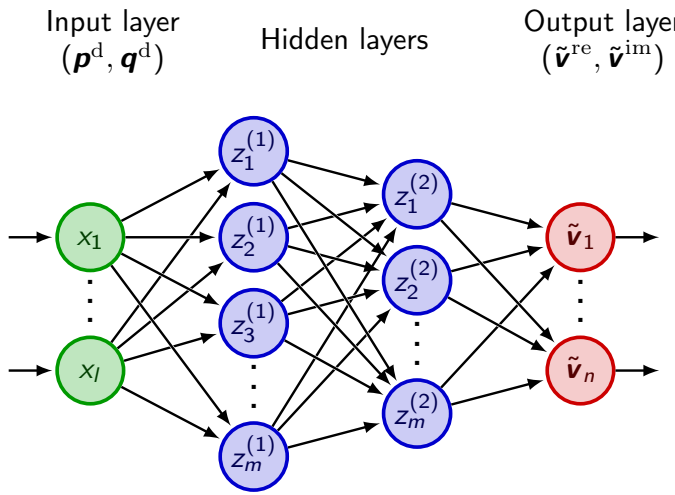
- A feasible convexification point, $\tilde{x}$, renders an inner convex approximation of the original problem.
- What if the convexification point, $\tilde{x}$, is not feasible?

$$
\begin{aligned}
\min_{x,s} \quad & f(x) + \lambda s \\
\text{s.t.} \quad & Ax = b, \\
& \hat{g}(x) - \check{g}(\tilde{x}) - \nabla \check{g}(\tilde{x})^{\top}(x - \tilde{x}) \leq s, \\
& x \in \mathbb{R}^n,\ s \in \mathbb{R}_{\geq 0},
\end{aligned}
$$

  where λ is a penalty term and s is a nonnegative slack variable.
- Note that no further assumptions are made to convexify the problem!
- Next, how can we predict good convexification points?
  We can leverage solutions of historical instances using *End-to-end learning*.

End-to-end learning

- Learning the mapping from the input parameters of an optimization problem to its solution.
- Given dataset $\{(\boldsymbol{x}_\ell, \boldsymbol{y}_\ell^*)\}_{\ell \in \mathcal{L}}$, where $\boldsymbol{y}_\ell^*$ denotes a solution to the optimization problem for the input $\boldsymbol{x}_\ell$.
- In the context of the AC-OPF: the input is the nodal demand vector and the output corresponds to the rectangular coordinates of the nodal voltages at the solution.
- **Main challenge**: Enforce constraints on the predictions!



Standard feed-forward NN

Enforcing constraints in neural networks

Soft methods

- Penalizing constraint violations
  - Augmented loss function
  - PINNs [Nellikkath et al., 2022]
  - Sensitivity-informed [Singh et al., 2022]
- Augmented Lagrangian methods
  - ALM [Fioretto et al., 2020]

Hard methods

- Implicit layers [Amos et al., 2017]
- Postprocessing [Zamzam et al., 2020; Li et al., 2022; Pan et al., 2023]
- Self-supervised [Donti et al., 2021]

What constraints do we want to enforce?

The relationship between current injections and nodal voltages, known as *Ohm's law*, is linear:

$$\begin{bmatrix} \boldsymbol{i}^{\mathrm{re}} \\ \boldsymbol{i}^{\mathrm{im}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{G} & -\boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{G} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^{\mathrm{re}} \\ \boldsymbol{v}^{\mathrm{im}} \end{bmatrix}.$$

- However, some of the current injections, the ones with generation and/or demand, are unknown before solving the problem.
- There is a subset of nodes whose current injections are known a priori and equal to zero. Such nodes are called *zero-injection nodes*.
- This talk: A hard method to enforce Ohm's law of zero-injection nodes.

Enforcing hard linear equality constraints

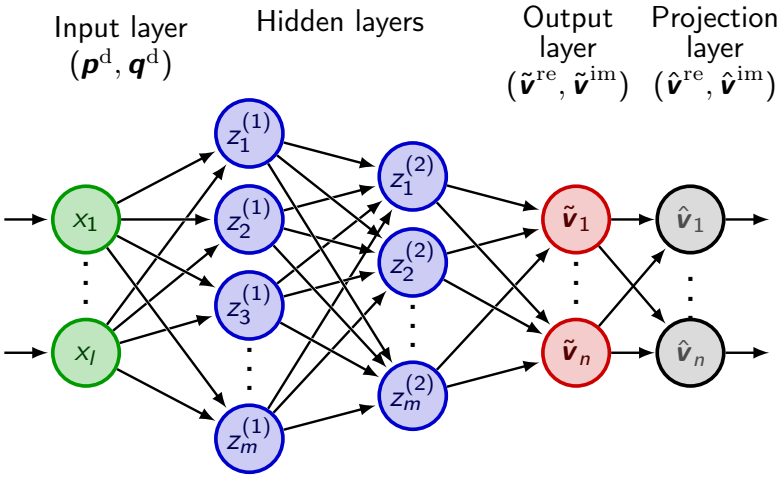
Can we enforce hard linear equality constraints using only an explicit layer?

- Explicit layers in feedforward NNs can be expressed as

$$\begin{bmatrix} \hat{\boldsymbol{v}}^{\mathrm{re}} \\ \hat{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} = \sigma\left(\boldsymbol{W} \begin{bmatrix} \tilde{\boldsymbol{v}}^{\mathrm{re}} \\ \tilde{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} + \boldsymbol{b} \right),$$

  where $\sigma(\cdot)$ denotes the nonlinear activation function.

- **Goal:** Find $\sigma(\cdot)$, $\boldsymbol{W}$, and $\boldsymbol{b}$ of the *projection layer* such that a set of linear equalities is satisfied during training and <u>inference</u>.



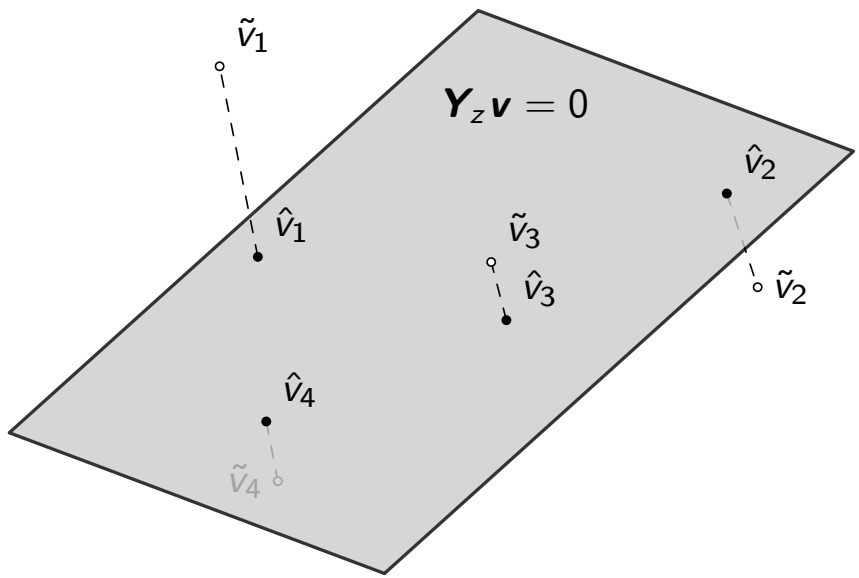
Proposed NN with projection layer

Projecting predicted variables

Orthogonal projection of the predicted variables $(\hat{\boldsymbol{v}}^{\mathrm{re}}, \hat{\boldsymbol{v}}^{\mathrm{im}})$ onto

$$\boldsymbol{Y}_z \begin{bmatrix} \boldsymbol{v}^{\mathrm{re}} \\ \boldsymbol{v}^{\mathrm{im}} \end{bmatrix} = 0,$$

where $\boldsymbol{Y}_z = \begin{bmatrix} \boldsymbol{G}_z & -\boldsymbol{B}_z \\ \boldsymbol{B}_z & \boldsymbol{G}_z \end{bmatrix}$.



Orthogonal projection of predicted voltages, $\tilde{\boldsymbol{v}}$, onto the nullspace of $\boldsymbol{Y}_z$.

Determining weights and biases

The orthogonal projection onto a linear set of equalities can be formulated as a quadratic problem

$$(\hat{\boldsymbol{v}}^{\mathrm{re}}, \hat{\boldsymbol{v}}^{\mathrm{im}}) \in \underset{\boldsymbol{v}^{\mathrm{re}}, \boldsymbol{v}^{\mathrm{im}}}{\arg\min} \quad \|\tilde{\boldsymbol{v}}^{\mathrm{re}} - \boldsymbol{v}^{\mathrm{re}}\|_2^2 + \|\tilde{\boldsymbol{v}}^{\mathrm{im}} - \boldsymbol{v}^{\mathrm{im}}\|_2^2$$

$$\text{s.t.} \quad \boldsymbol{Y}_z \begin{bmatrix} \boldsymbol{v}^{\mathrm{re}} \\ \boldsymbol{v}^{\mathrm{im}} \end{bmatrix} = 0.$$

Its closed-form solution is given by

$$\begin{bmatrix} \hat{\boldsymbol{v}}^{\mathrm{re}} \\ \hat{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} = \boldsymbol{A}^* \begin{bmatrix} \tilde{\boldsymbol{v}}^{\mathrm{re}} \\ \tilde{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix},$$

where $\boldsymbol{A}^* = \mathbb{I} - \boldsymbol{Y}_z^\top \left(\boldsymbol{Y}_z \boldsymbol{Y}_z^\top \right)^{-1} \boldsymbol{Y}_z$.

Determining weights and biases

- The closed-form solution can be represented as an explicit layer

$$\begin{bmatrix} \hat{\boldsymbol{v}}^{\mathrm{re}} \\ \hat{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} = \boldsymbol{A}^* \begin{bmatrix} \tilde{\boldsymbol{v}}^{\mathrm{re}} \\ \tilde{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} \Longleftrightarrow \begin{bmatrix} \hat{\boldsymbol{v}}^{\mathrm{re}} \\ \hat{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} = \sigma \left(\boldsymbol{W} \begin{bmatrix} \tilde{\boldsymbol{v}}^{\mathrm{re}} \\ \tilde{\boldsymbol{v}}^{\mathrm{im}} \end{bmatrix} + \boldsymbol{b} \right),$$

  where $\sigma(\cdot)$ is a linear activation function, $\boldsymbol{W} = \boldsymbol{A}^*$, and $\boldsymbol{b} = 0$.

- The matrix $\boldsymbol{A}^* = \mathbb{I} - \boldsymbol{Y}_z^\top \left(\boldsymbol{Y}_z \boldsymbol{Y}_z^\top \right)^{-1} \boldsymbol{Y}_z$, which corresponds to the weights of the projection layer, *only depends on the topology of the network* and is independent of the operating conditions. Hence, $\boldsymbol{A}^*$ is only computed once for training and inference.

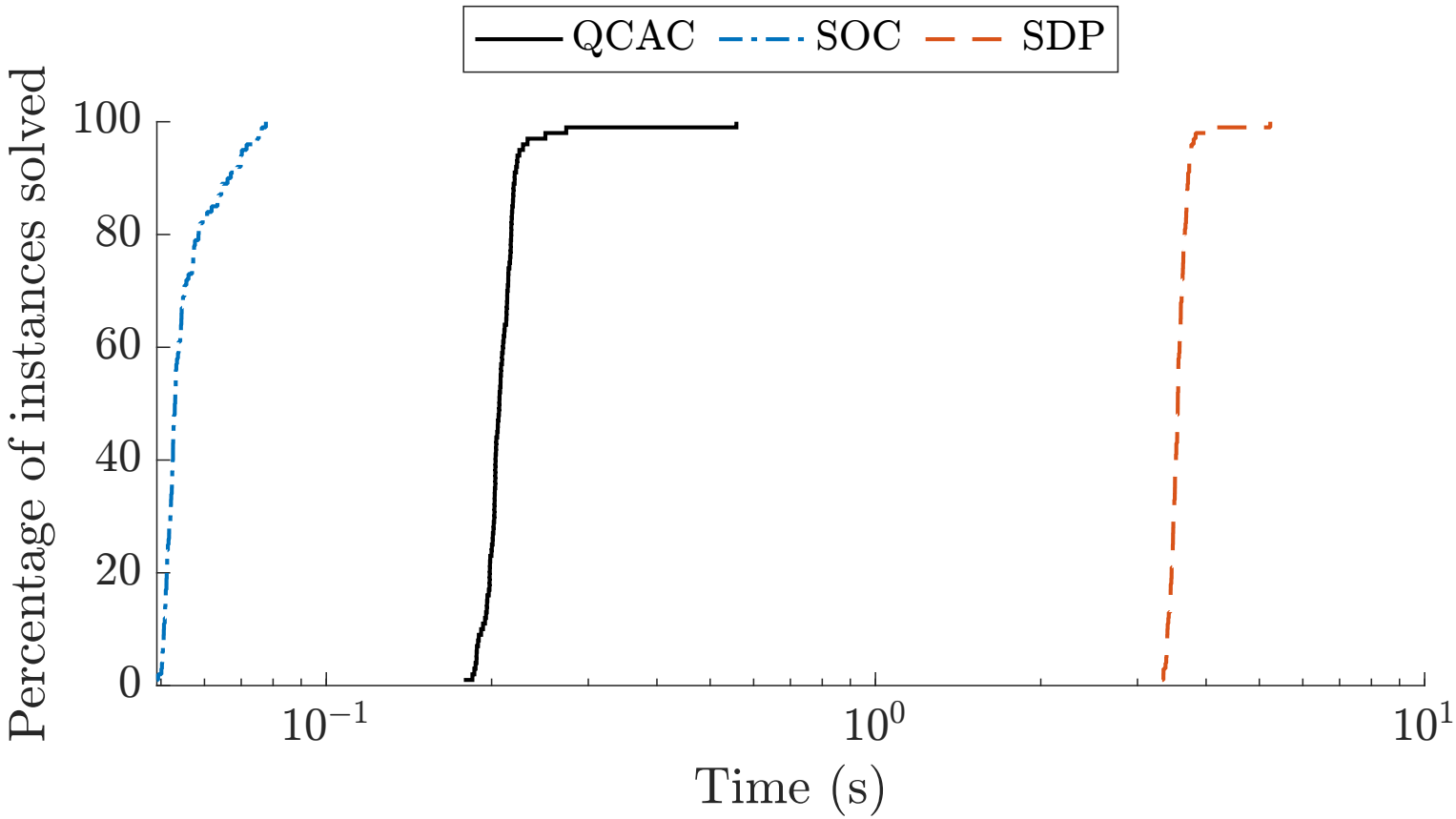
Numerical results

- Congested condition of the IEEE 118-bus system from the PGLib [Babaeinejadsarookolaee et al., 2021].
- 100 samples for random active and reactive power demands, $\pm 40\%$ and $\pm 15\%$ from the base case, respectively.

Table: Optimality gap comparison

Model	Optimality gap (%)		
	Median	Min	Max
QCAC approximation	**1.4824**	**0.006**	**12.0822**
SOC relaxation	25.2628	10.5189	32.574
SDP relaxation	10.0873	3.9027	15.1559

What about solution time?



One order of magnitude faster and more accurate than the SDP relaxation!

Even more important, what about distance to feasibility?

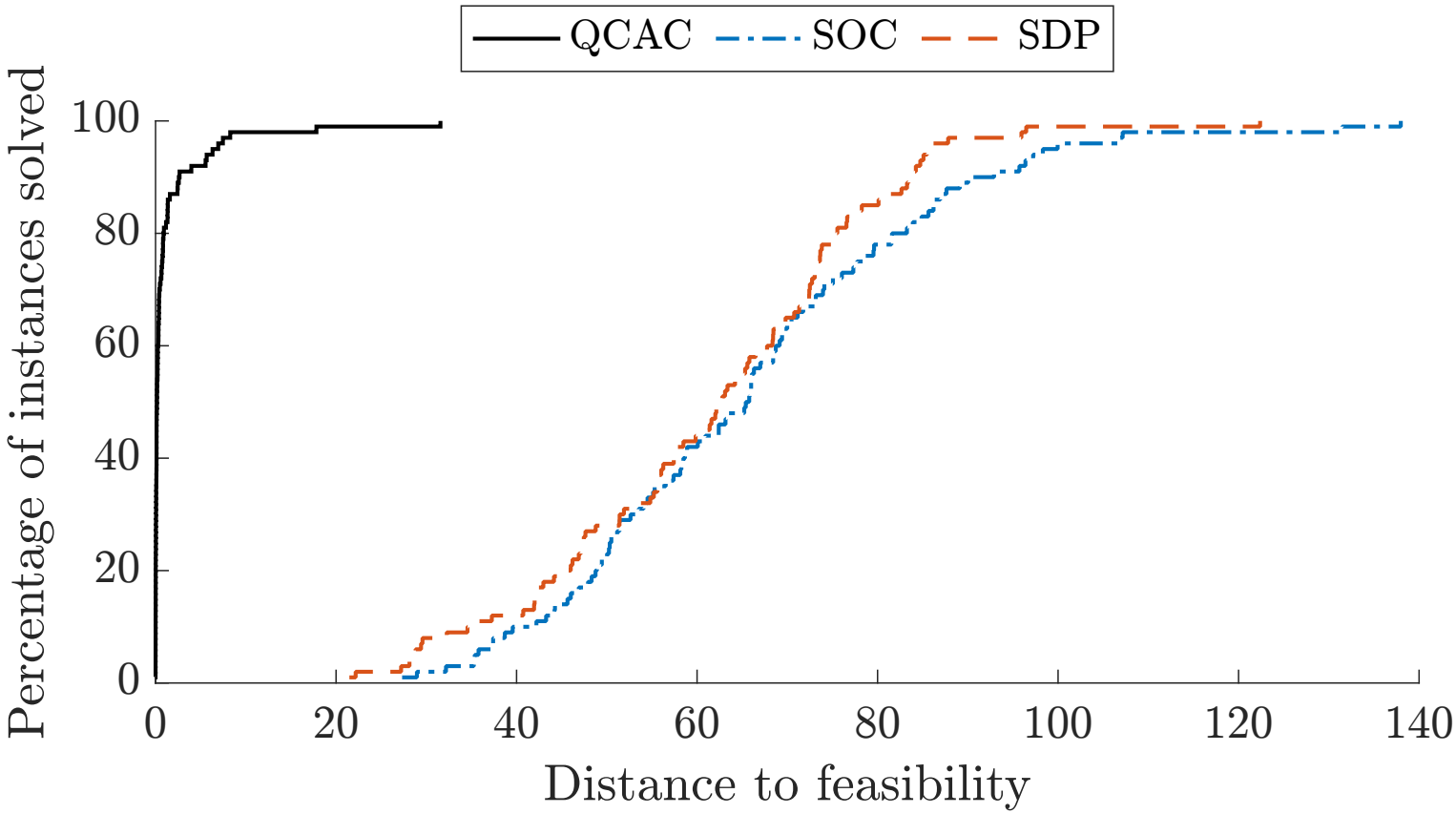


Figure: Feasibility comparison of studied models

And significantly closer to being feasible!

Generation dispatch correlation

Table: Correlation coefficient comparison

Model	Correlation coefficient	
	Active power	Reactive power
QCAC approximation	**0.9949**	**0.8943**
SOC relaxation	0.8605	0.6101
SDP relaxation	0.9607	0.5601

- Better correlation in generation dispatch makes the model more suitable for applications sensitive to active and reactive power generation.
- For instance, unit commitment and optimal reactive power dispatch

Conclusions

- ML for combinatorial and global optimization is a fast-evolving field.
- A number of primal and dual tasks can potentially be accelerated using ML.