Combining Constraint Programming and Machine Learning

From Current Progress to Future Opportunities

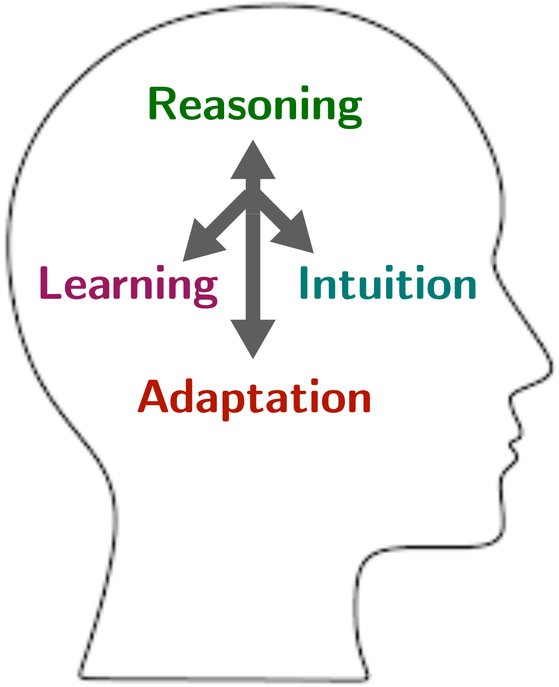
SOFT'2026 - Lisbon - July 24, 2026

UCLouvain POLYTECHNIQUE MONTRÉAL CORAIL Mila CIRRELT

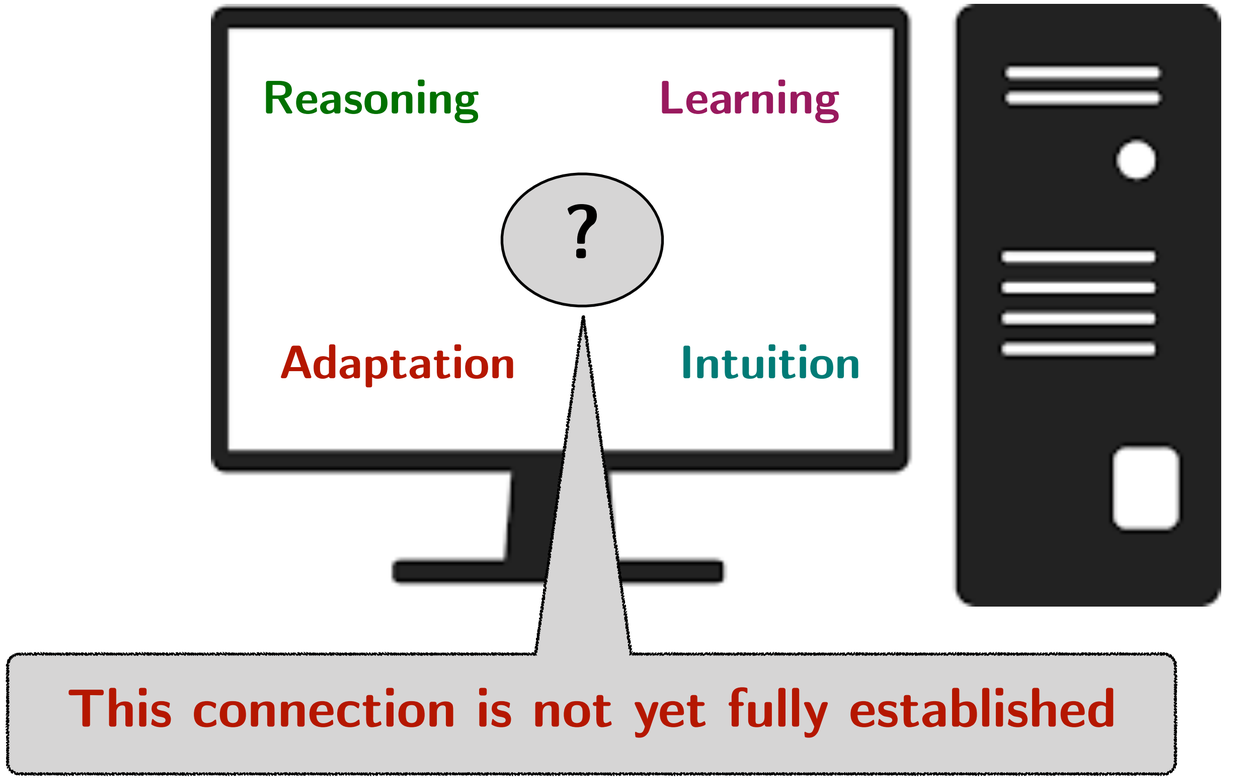
Quentin Cappart

Human intelligence versus artificial intelligence

Human intelligence



Artificial intelligence



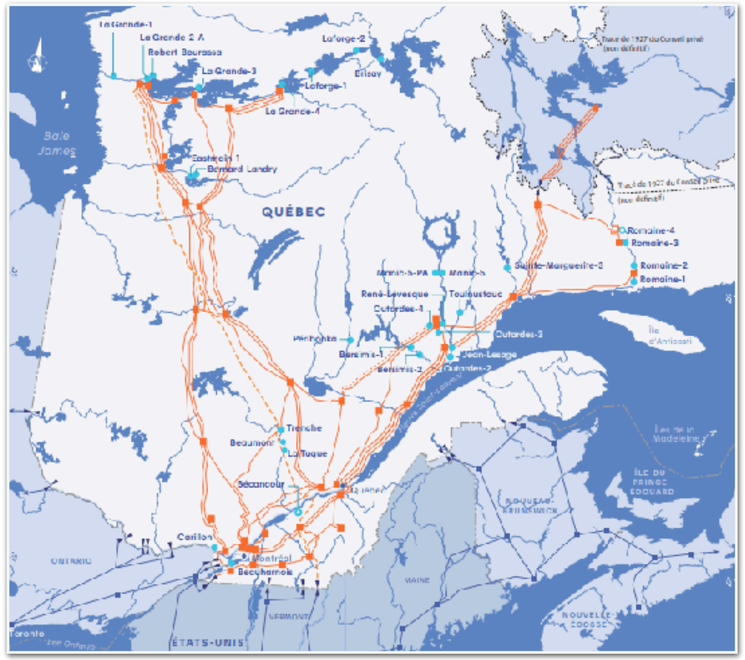
Long-term research plan: building an AI with these connections

Goal: solving efficiently interesting and computationally hard problems

Scope of my research

? But what is an interesting and computationally hard problem ?

Interesting: problems having a positive impact (e.g., sustainable development)



Hydro Québec

- Electricity generation
- Transmission
- Distribution

Problem: scheduling the maintenance of equipment

Challenge: there is no closed-form for few constraints

Contribution: learning these constraints

Impact: we solved a situation still unsolved by HQ

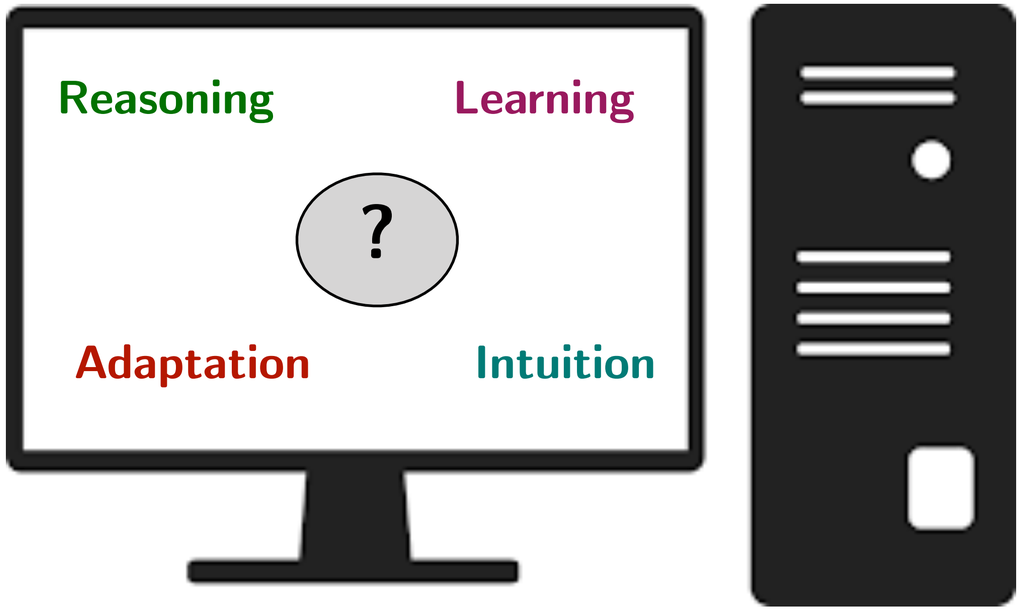
[Popovic et al., *CP 2022*], [Barral et al., *CPAIOR 2024*]

Computationally hard: the main challenge is algorithmic (e.g., NP-hard problems)

Additional goal: providing easy-to-use and open-source software

I propose to solve such problems with a hybrid artificial intelligence

Declarative solvers as a unifying framework



My research hypothesis

***Constraint Programming** can be a hosting technology for building this hybrid AI*

$$\text{CP} = \text{model} + \text{reasoning} + \text{search} + \text{learning}$$

? How can we plug learning inside a CP solver ?

Overview of a CP Solver

Hydro Québec

? But where can we plug learning in this process ?

Combinatorial Problem

```
int: n = 3;
int: tot = n * (n^2 + 1) div 2;

array[1..n,1..n] of var 1..nan: x;

constraint all_different(x);

constraint forall(k in 1..n)(sum(i in 1..n)(x[k,i]) == tot);
constraint forall(k in 1..n)(sum(i in 1..n)(x[i,k]) == tot);
constraint sum(i in 1..n) (x[i,i]) = tot;
constraint sum(i in 1..n) (x[i,n-i+1]) = tot;

solve satisfy;
```

CP Modelling

CSP and COP

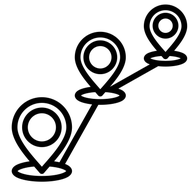
CP Model

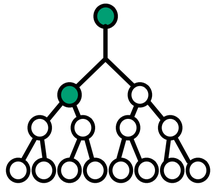
Search-based Solver

Backtracking search

Large neighborhood search

Solution



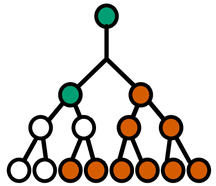


Search Node

Reasoning

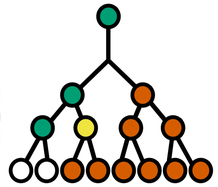
Propagators

Fix-point algorithm



Reduced Tree

Next Exploration



Branching

Variable-selection

Value-selection

Survey on the topic

We published (very recently) a survey to see how this question has been answered

JAIR Journal of Artificial Intelligence Research

Dec 30, 2025

Special Track: CP and ML

Combining Constraint Programming and Machine Learning: From Current Progress to Future Opportunities

Q. Cappart

T. Guns

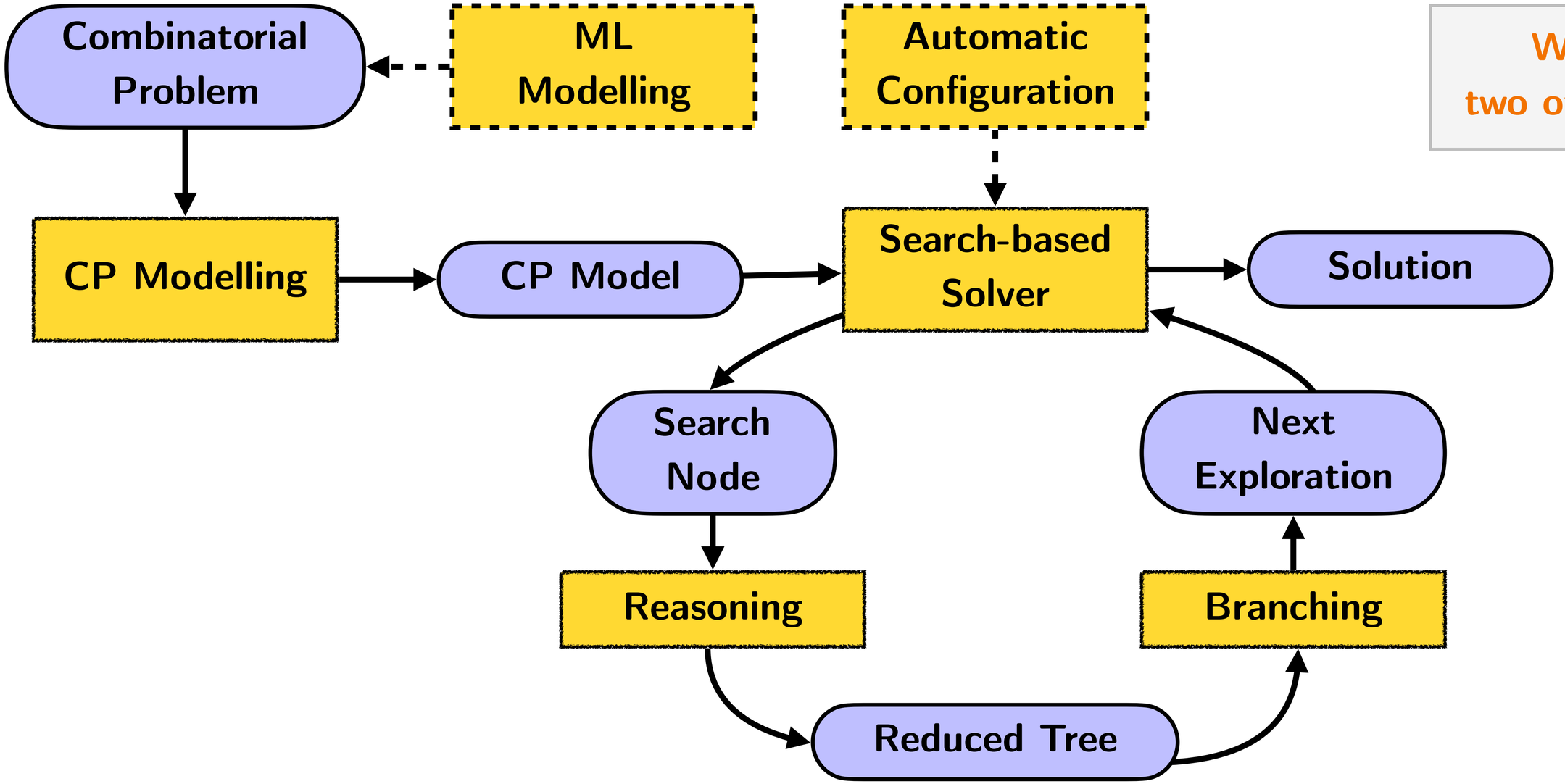
M. Lombardi

G. Pesant

D. Tsouros

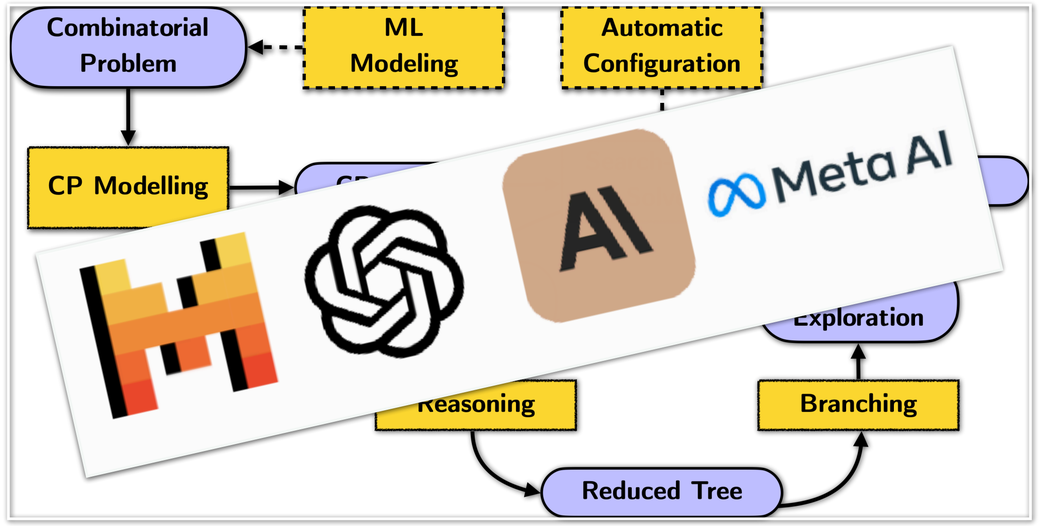
Strict scope: contributions involving CP and learning (CP for ML, and ML for CP)

Overview of a CP Solver



Each decisional step has the potential to be improved with learning

Limitation of learning alone



CP = model + reasoning + search + learning

The motivations are to

1) Improve the performance of solvers

2) Make this easy to grasp for end-users

3) Broaden the scope to new problems

? Can we use only learning for that (LLM or agentic AI) ?

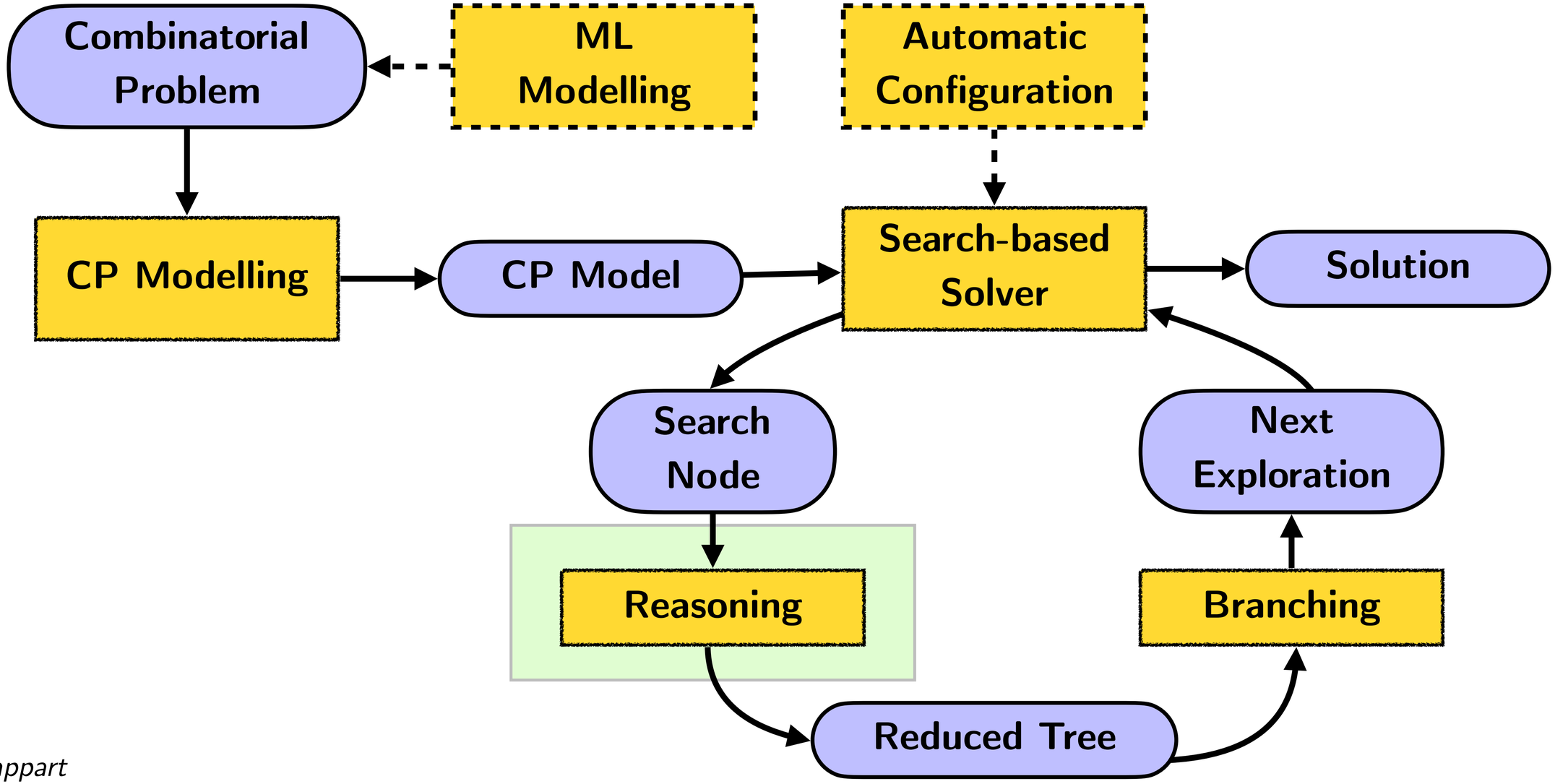
Fundamental limitation of learning: the predictions obtained can be **wrong**

Requirement for a practical use: we want to **trust our algorithms**

- **Search procedure:** solving the problem **with guarantee** (feasibility and optimality)
- **Reasoning:** accelerating the solving thanks to **true knowledge** (symbolic information)
- **Learning:** accelerating with **learned knowledge** (that can be wrong)
- **Modeling:** making everything **easy for the end-user** (declarative programming)

Agenda for today

I will show how learning can be used for pruning (with guarantees)



Typical propagators in a CP solver

? How to do that? Propagation is mainly based on certifiable methods (algorithms, logic, math)

CP = model + reasoning + search + learning

- **Fix-point**
- **Consistency (AC3, etc.)**
- **Global Constraints**
  - **Graph algorithmic (allDifferent, etc.)**
  - **Decision diagrams (Haddock, etc.)**
  - **Cost-based filtering**
  - ...
- **Other tools (noGoods, etc.)**

Some propagators rely on tricky-to-get information to be efficient

Cost-based filtering (Focacci et al., CP 1999)

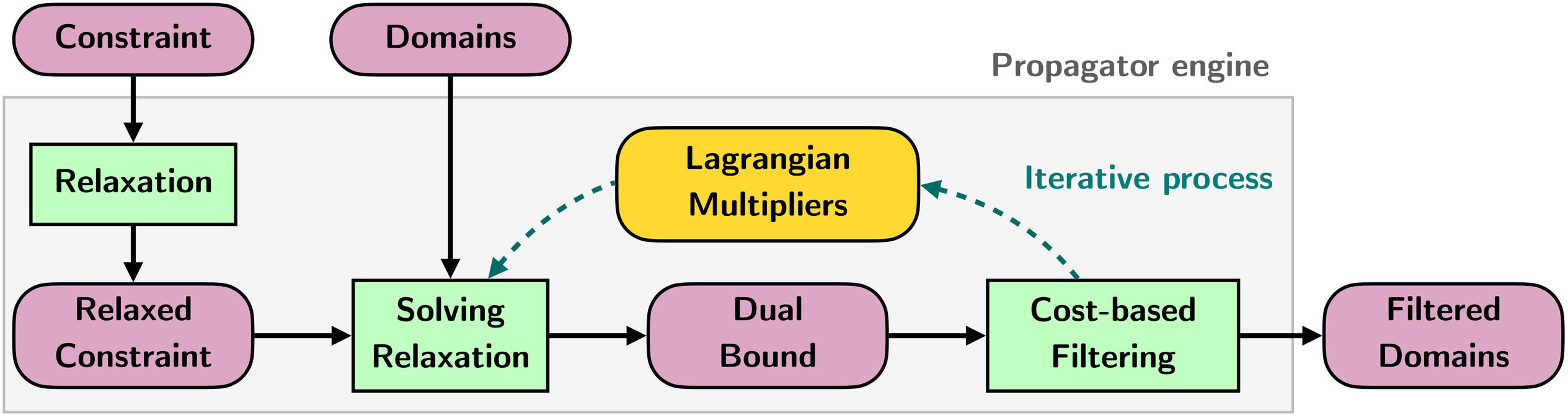
Cost-based filtering leverages relaxation to improve the filtering of a constraint

Step 1: a valid relaxation is embedded into the global constraint

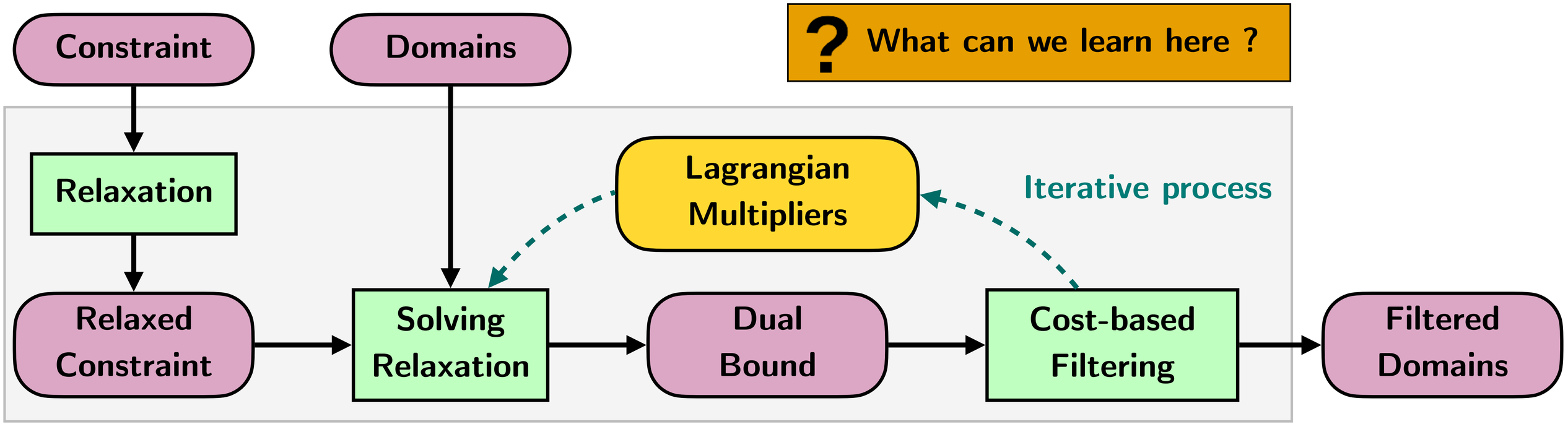
Step 2: the relaxed problem is solved to get a dual bound on the cost

Step 3: the bound is used to prune the search space via cost-based filtering

Caveat: the quality of the relaxation depends on Lagrangian multipliers with iterative improvements



Data-driven cost-based filtering



Trick: the quality of the relaxation **heavily** depends on **Lagrangian multipliers**

Bad news: **determining the best values of the multipliers is generally costly**

We propose to obtain the multipliers through **self-supervised learning**

Self-supervised learning: **method that generates itself its learning signal** (does not require label)

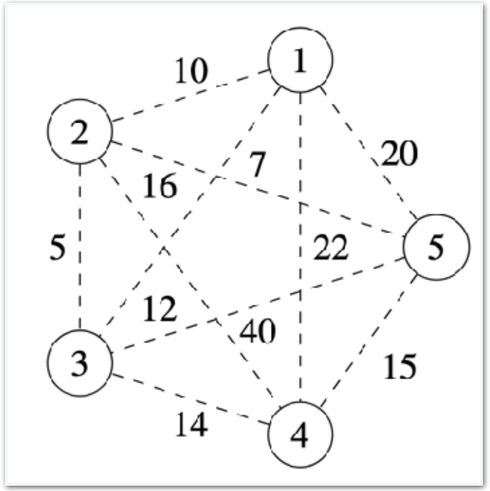
Case study on the *WeightedCircuit* constraint

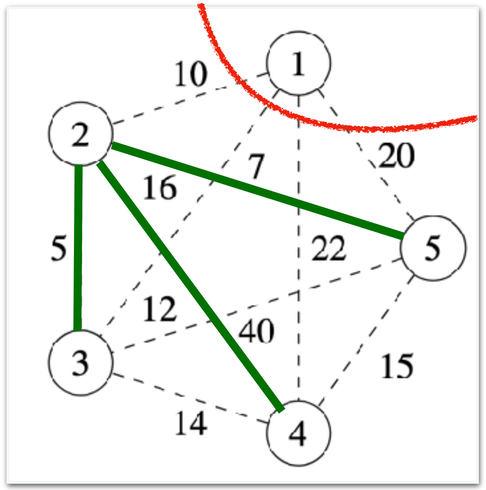
WeightedCircuit(X, G, d) : **ensure that variables** X **form a TSP tour in** G **of cost** $\leq d$

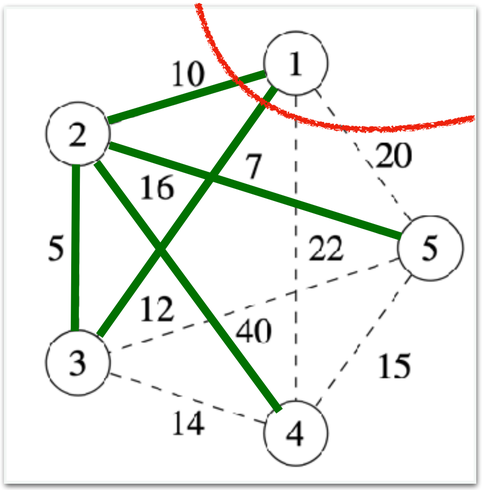
1-tree relaxation(X, G) : **allows the TSP to go twice in a specific node**

Step 1: ensure that variables X **form a minimum spanning tree in** $G \backslash \{v_1\}$

Step 2: link the remaining node v_1 **to the tree with the two cheapest edges**





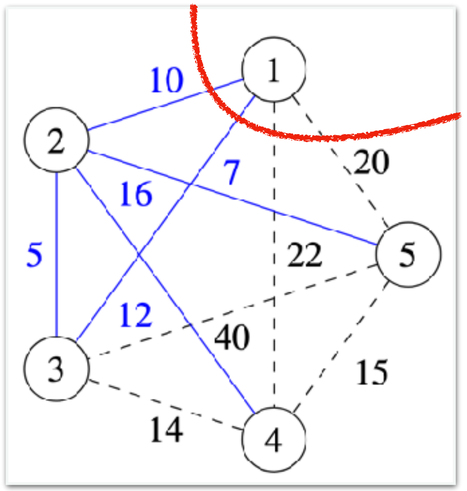


Optimal TSP cost: 62

Relaxation cost: 50

We have here an example of *how a specific constraint can be relaxed*

Improving the dual bound



Relaxation cost: 50

? The bound is not very tight, could we improve it ?

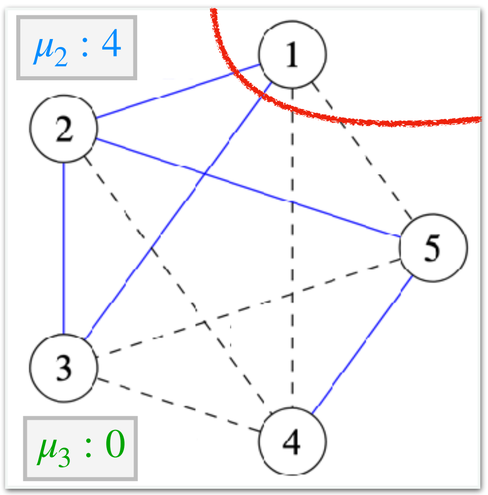
Lagrangian Multipliers

$\langle \mu_1, \mu_2, \mu_3, \mu_4, \mu_5 \rangle$

$$c'_{u,v} = c_{u,v} + \mu_u + \mu_v$$

$$c'_{2,3} = c_{2,3} + \mu_2 + \mu_3$$

$$9 = 5 + 4 + 0$$



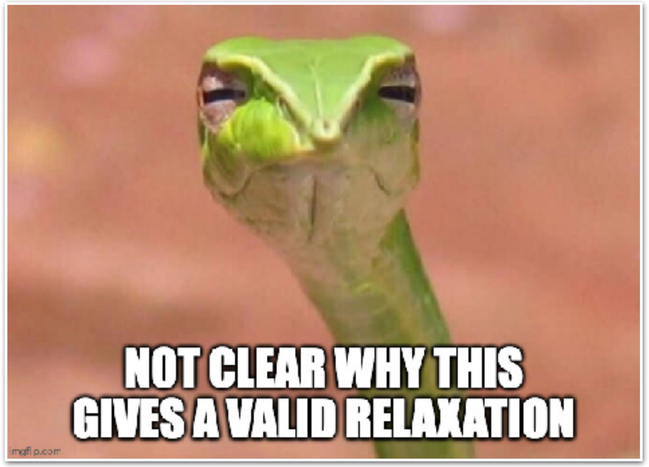
Relaxation cost: 59

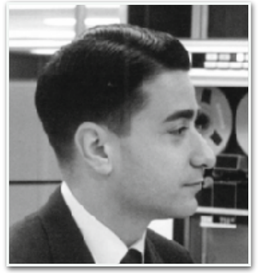
Main idea: perturbate the cost of each edge with values called Lagrangian multipliers

Lagrangian multiplier: value associated with each node

Perturbation: change the cost of each edge based on the multipliers of its nodes

Lagrangian multipliers





$$c'_{u,v} = c_{u,v} + \mu_u + \mu_v$$

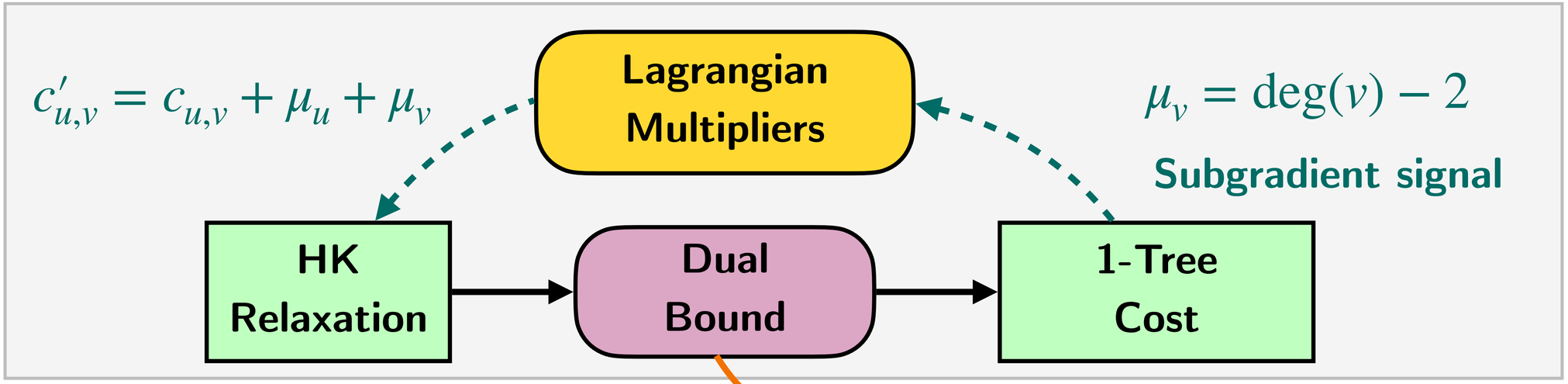
Good news: proved by **Held and Karp** in 1970

Property 1: the **optimal TSP tour is invariant** under this perturbation

Property 2: better bounds generally result in a **much better filtering**

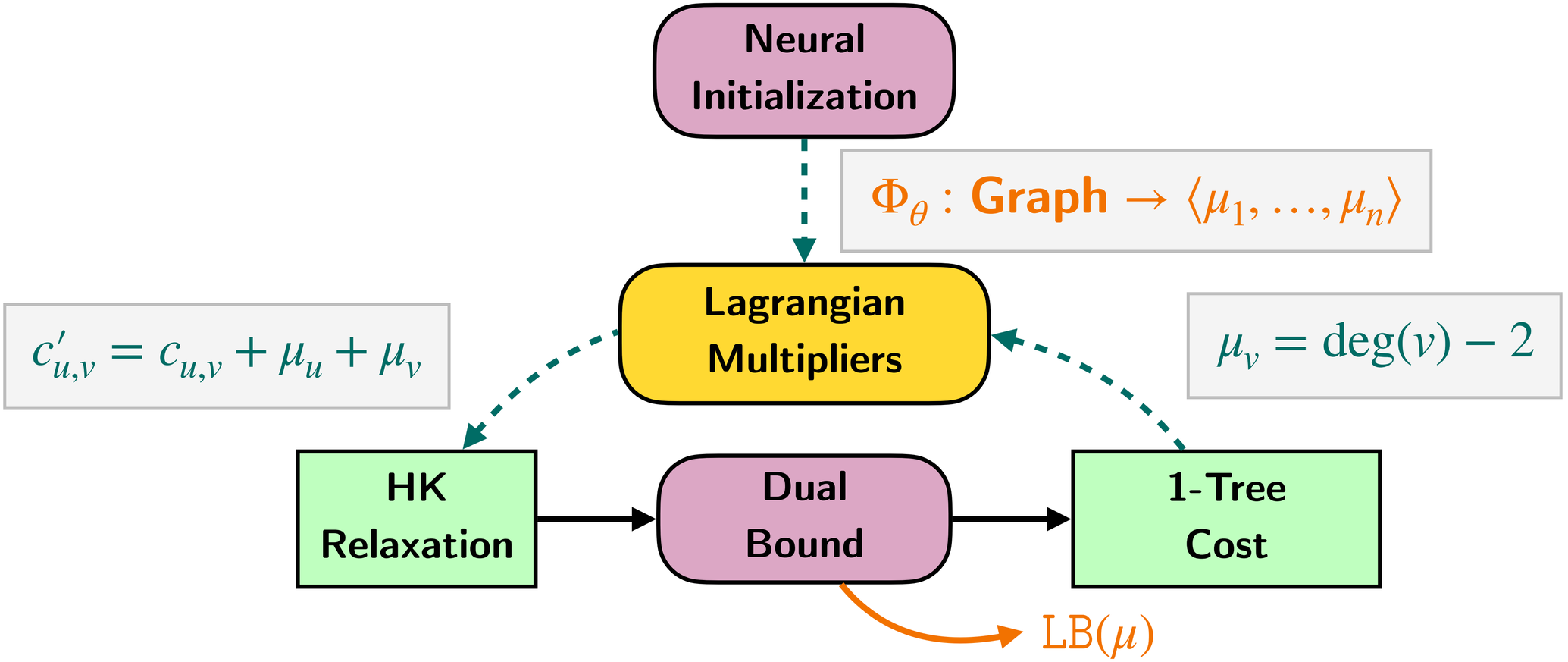
? How can we find good values for the multipliers ?

H&K also proposed an **iterative algorithm**



Learning the Lagrangian multipliers

Unfortunately, this iterative adjustment of the multipliers is computationally expensive

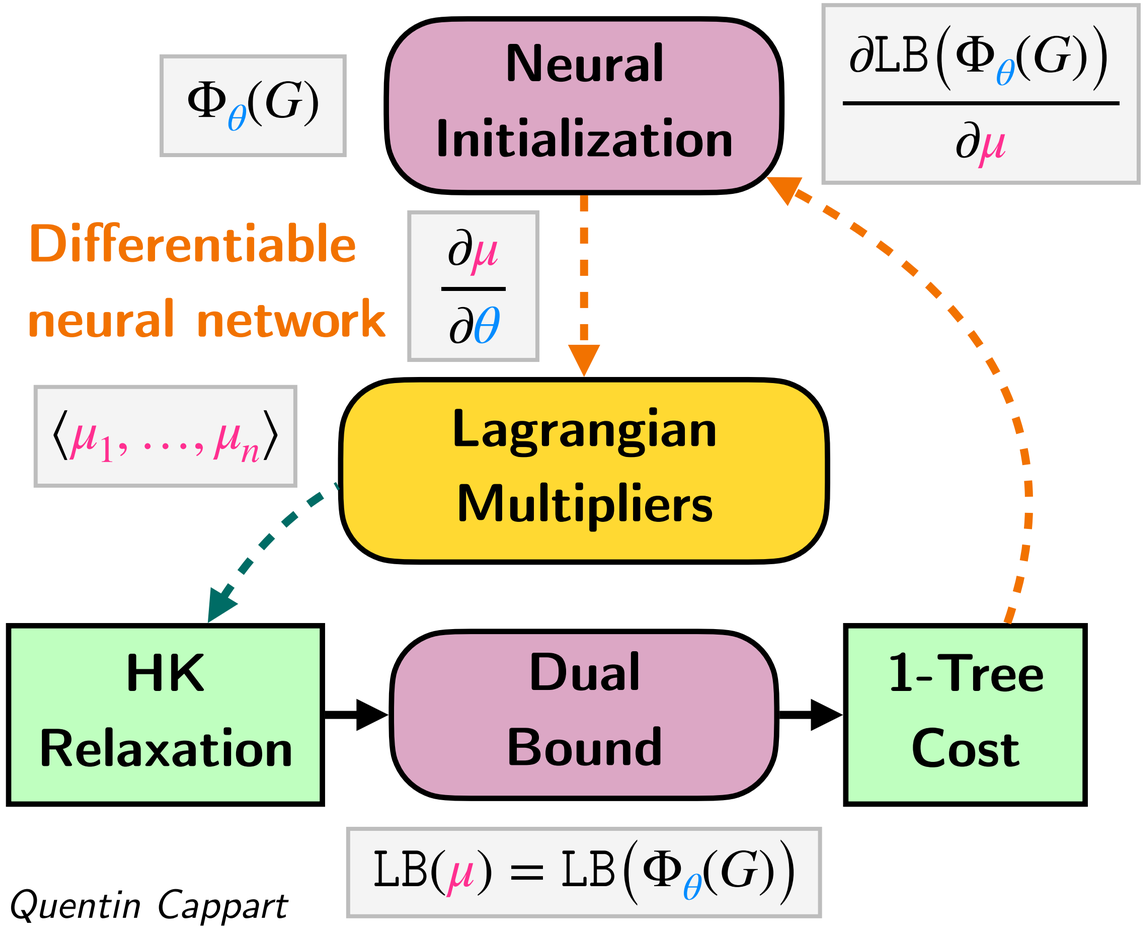


We propose to use a graph neural network to predict the optimal values of the multipliers

Learning the Lagrangian multipliers

? How to train the network ? We do not have any label !

We directly focus on maximizing the dual bound (self-supervised learning)



$$\max_{\mu} \mathrm{LB}(\mu)$$ **Our target**

$$\max_{\theta} \mathrm{LB}\big(\Phi_\theta(G)\big)$$ **Predicting the multipliers**

$$\nabla_\theta \, \mathrm{LB}\big(\Phi_\theta(G)\big)$$ **Computing the gradient**

$$\frac{\partial \mathrm{LB}\big(\Phi_\theta(G)\big)}{\partial \mu} \times \frac{\partial \mu}{\partial \theta}$$ **Application of chain rule**

$$\big(\deg(v) - 2\big) \times \frac{\partial \mu}{\partial \theta}$$ **Held and Karp update**

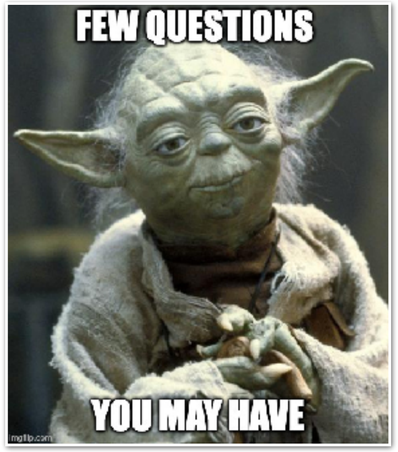
Learning the Lagrangian multipliers

We now have a gradient signal that can be used for gradient-ascent-like training

```
for each epoch :
  for each instance G :
    μ ← Φ_θ(G)
    θ ← θ + α ∇_θ LB(μ)
return θ
```

Predicting the multipliers

Gradient-ascent step



Neural architecture: graph neural network with SAGEConv [Hamilton et al., *NeurIPS 2017*]

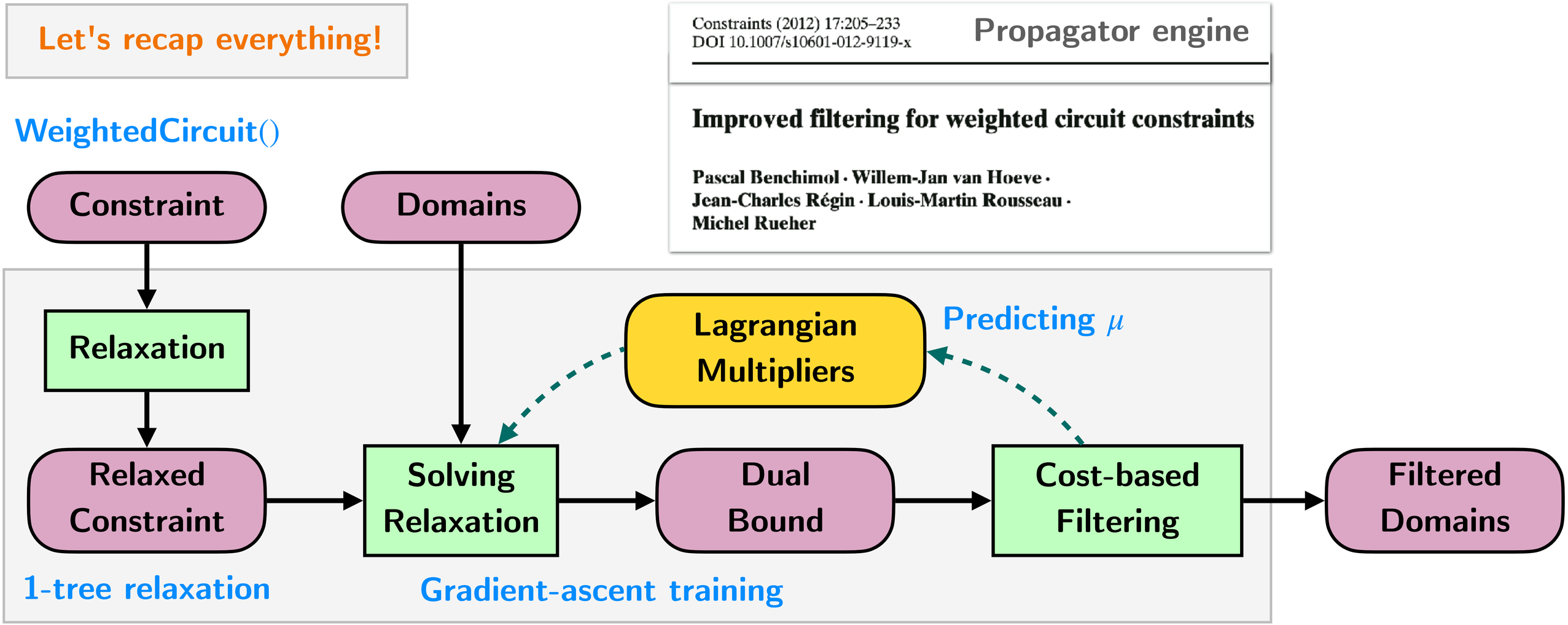
Features: city coordinates, presence on the 1-tree, cost of the edges, etc.

Training loop: improved with modern tools (e.g. Adam optimizer)

Propagator used: cost-based filtering for weightedCircuit [Benchimol et al., *Constraints 2012*]

Learning the Lagrangian multipliers

Let's recap everything!



Constraints (2012) 17:205–233
DOI 10.1007/s10601-012-9119-x

Propagator engine

Improved filtering for weighted circuit constraints

Pascal Benchimol · Willem-Jan van Hoeve · Jean-Charles Régin · Louis-Martin Rousseau · Michel Rueher

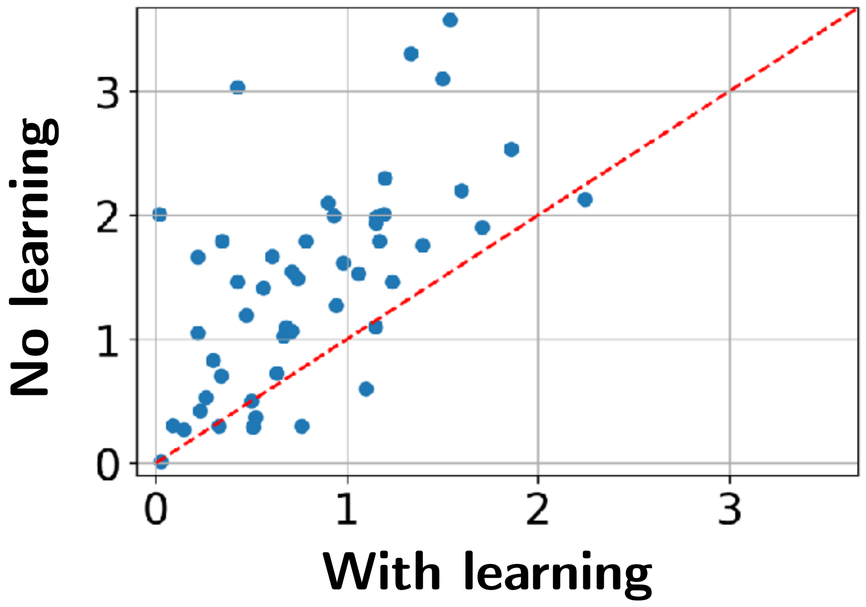
As a result, we can either improve the strength or the efficiency of the propagator

Experiments on the TSP

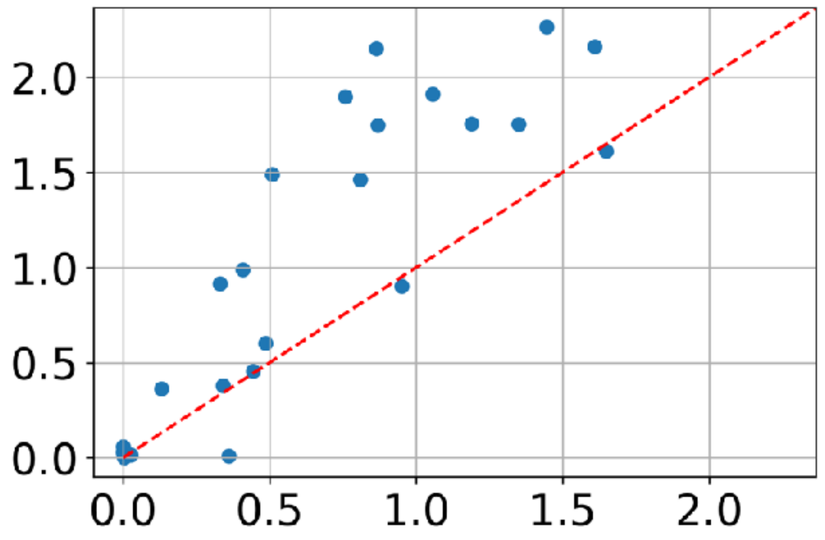
? Ok, but does it work in practice ?

Experiment: scatter plot comparing the optimality gap with and without learning (1,800 seconds)

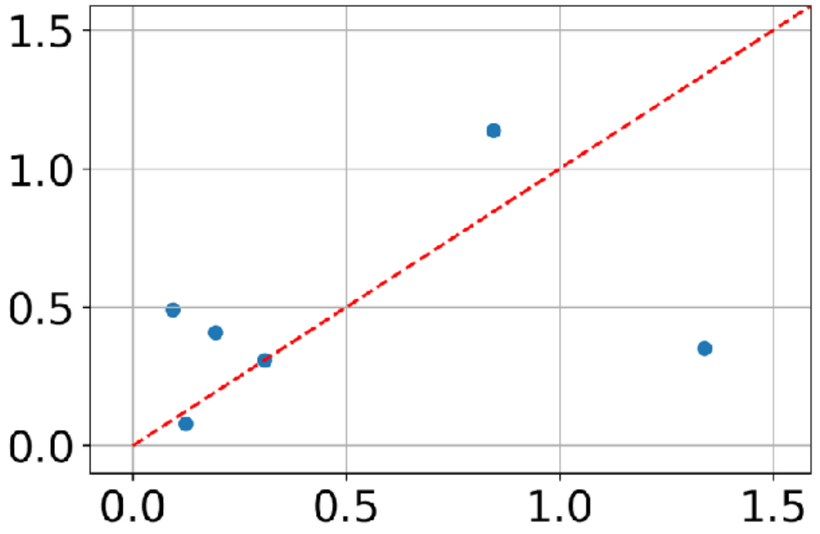
Random TSP (200 cities)



Clustered TSP (200 cities)



Hard TSP (Hougardy et al. 2021)



Take-away 1: warm-starting the multipliers with learning enabled stronger filtering

Take-away 2: replacing the HK process only with learning was not successful

Learning Lagrangian Multipliers for the TSP

Learning Lagrangian Multipliers for the Travelling Salesman Problem (CP 2024)

Augustin Parjadis
Polytechnique Montréal, Canada

Quentin Cappart
Polytechnique Montréal, Canada

Bistra Dilkina
Center for Artificial Intelligence in Society, University of Southern California, Los Angeles, CA, USA

Aaron Ferber
Center for Artificial Intelligence in Society, University of Southern California, Los Angeles, CA, USA

Louis-Martin Rousseau
Polytechnique Montréal, Canada





Current work: adapting this proof of concept to other global constraints

Questions ?

Typical propagators in a CP solver

Limitation: improvements are only at the constraint-level and not at the model-level

? Can we use a similar idea to get a more general bounding mechanism in a CP solver ?

CP = model + reasoning + search + learning

- Fix-point
- Consistency (AC3, etc.)
- Global Constraints
- Other tools (noGoods, etc.)
- Cost-based filtering
- Lagrangian decomposition

We extend the idea of learning multipliers to CP-based Lagrangian decomposition

Lagrangian decomposition (Guignard and Kim, 1987)

Lagrangian decomposition splits the problem into independent and easier subproblems

Step 1: each variable in each constraint is duplicated, except for the first constraint

Step 2: a constraint linking the values is added for each new variable

Step 3: these constraints are moved into the objective function with a penalty term

$$\begin{aligned} \min \quad & f(X_1, X_2, X_3) \\ \text{s.t.} \quad & C_1(X_1, X_2, X_3) \\ & C_2(X_2, X_3) \\ & X_1, X_2, X_3 \in \mathbb{N}^+ \end{aligned}$$

$$\begin{aligned} \min \quad & f(X_1, X_2, X_3) \\ \text{s.t.} \quad & C_1(X_1, X_2, X_3) \\ & C_2(Y_2, Y_3) \\ & Y_2 = X_2, \; Y_3 = X_3 \\ & X_1, X_2, Y_2, X_3, Y_3 \in \mathbb{N}^+ \end{aligned}$$

$$\min \quad f(X_1, X_2, X_3) + \mu_2 \cdot (X_2 - Y_2) + \mu_3 \cdot (X_3 - Y_3)$$

Again, we have Lagrangian multipliers, but now at the model-level

Lagrangian decomposition (Guignard and Kim, 1987)

$$\mathrm{LB}(\mu_2, \mu_3) = \left\{ \begin{array}{ll} \min & f(X_1, X_2, X_3) + \mu_2 \cdot (X_2 - Y_2) + \mu_3 \cdot (X_3 - Y_3) \\ \textbf{s.t.} & C_1(X_1, X_2, X_3) \\ & C_2(Y_2, Y_3) \\ & X_1, X_2, Y_2, X_3, Y_3 \in \mathbb{N}^+ \end{array} \right.$$

Solving this relaxed problem gives a dual bound (lower if minimizing)

? But, is it easy to solve ?

Observation: by construction, each constraint has its own set of variables

Consequence: each constraint can be solved independently

$$\mathrm{LB}(\mu_2, \mu_3) = \begin{array}{ll} \min & \left(f(X_1, X_2, X_3) + \mu_2 \cdot X_2 + \mu_3 \cdot X_3 \right) \\ \textbf{s.t.} & C_1(X_1, X_2, X_3) \\ & X_1, X_2, X_3 \in \mathbb{N}^+ \end{array} + \begin{array}{ll} \min & \left(-\mu_2 \cdot Y_2 - \mu_3 \cdot Y_3 \right) \\ \textbf{s.t.} & C_2(Y_2, Y_3) \\ & Y_2, Y_3 \in \mathbb{N}^+ \end{array}$$

Given the multipliers, we obtain a bound by solving several subproblems

Lagrangian decomposition in CP (Hà et al. , CP 2015)

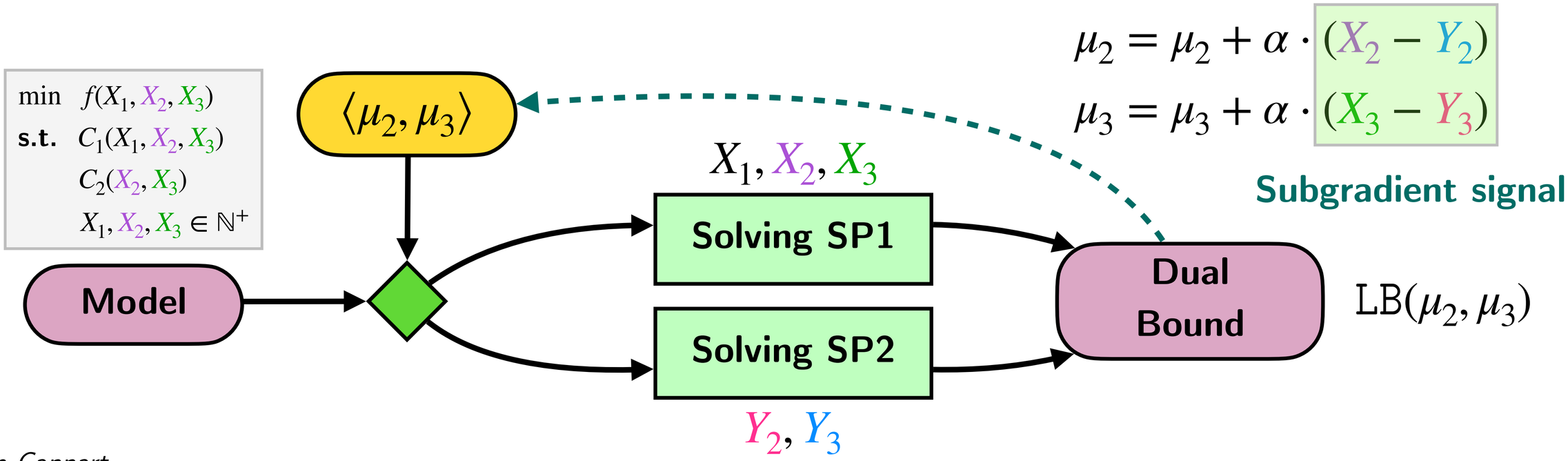
? How to set the multipliers to get a tight bound for a CP solver ?

Initialization: we set the multipliers to an arbitrary value

Step 1: we solve all subproblems with these values (we get a dual bound)

Step 2: we update the multipliers with sub-gradient (we improve the bound)

Main loop: we repeat steps 1 and 2 for *x* iterations

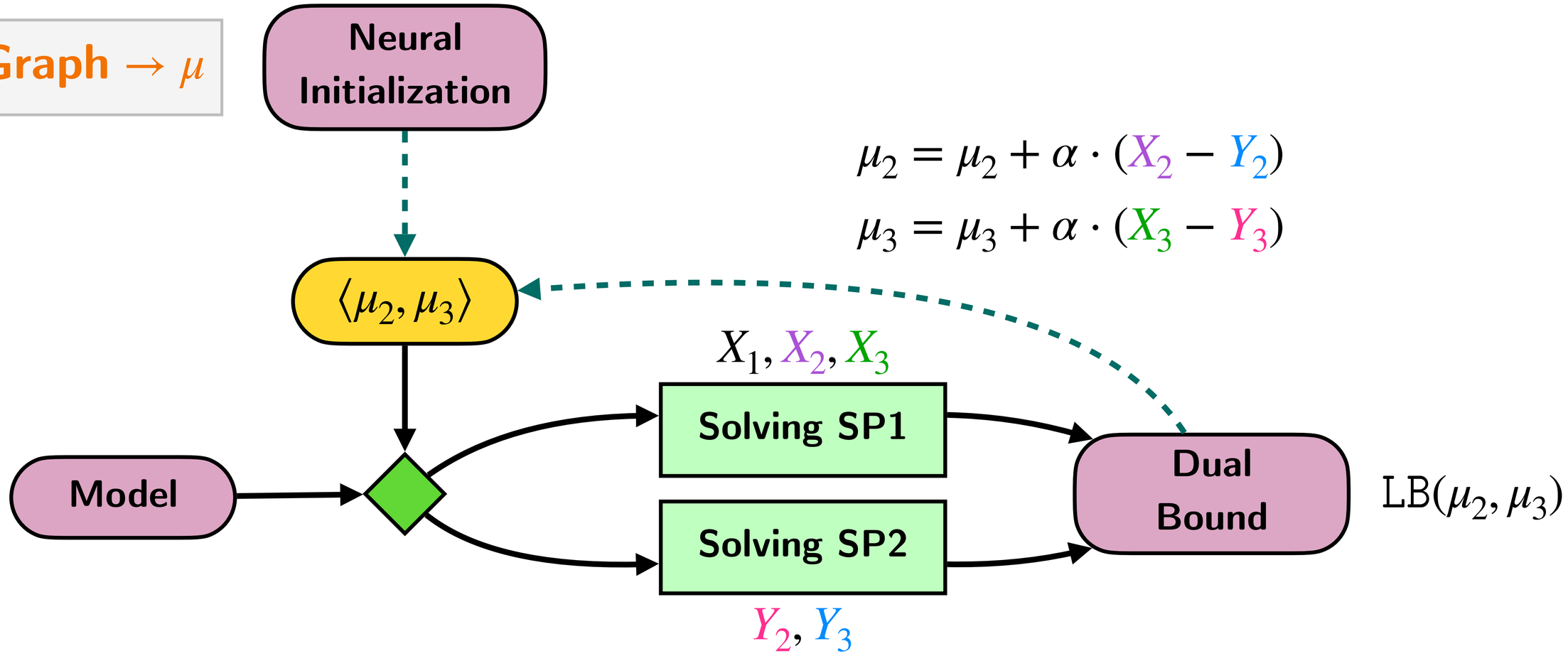


Learning multipliers for Lagrangian decomposition

Unfortunately, this process is very costly as it requires solving several subproblems at each iteration

Again, we propose to use a neural network to predict the optimal values of the multipliers

Φ_θ : **Graph** $\rightarrow \mu$



Learning multipliers for Lagrangian decomposition

We directly focus on maximizing the dual bound (self-supervised learning)

$$\max_{\mu} \mathrm{LB}(\mu)$$

Our target

$$\max_{\theta} \mathrm{LB}\big(\Phi_\theta(G)\big)$$

Predicting the multipliers

$$\nabla_\theta \, \mathrm{LB}\big(\Phi_\theta(G)\big)$$

Computing the gradient

$$\frac{\partial \mathrm{LB}\big(\Phi_\theta(G)\big)}{\partial \mu} \times \frac{\partial \mu}{\partial \theta}$$

Application of chain rule

$$(X - Y) \times \frac{\partial \mu}{\partial \theta}$$

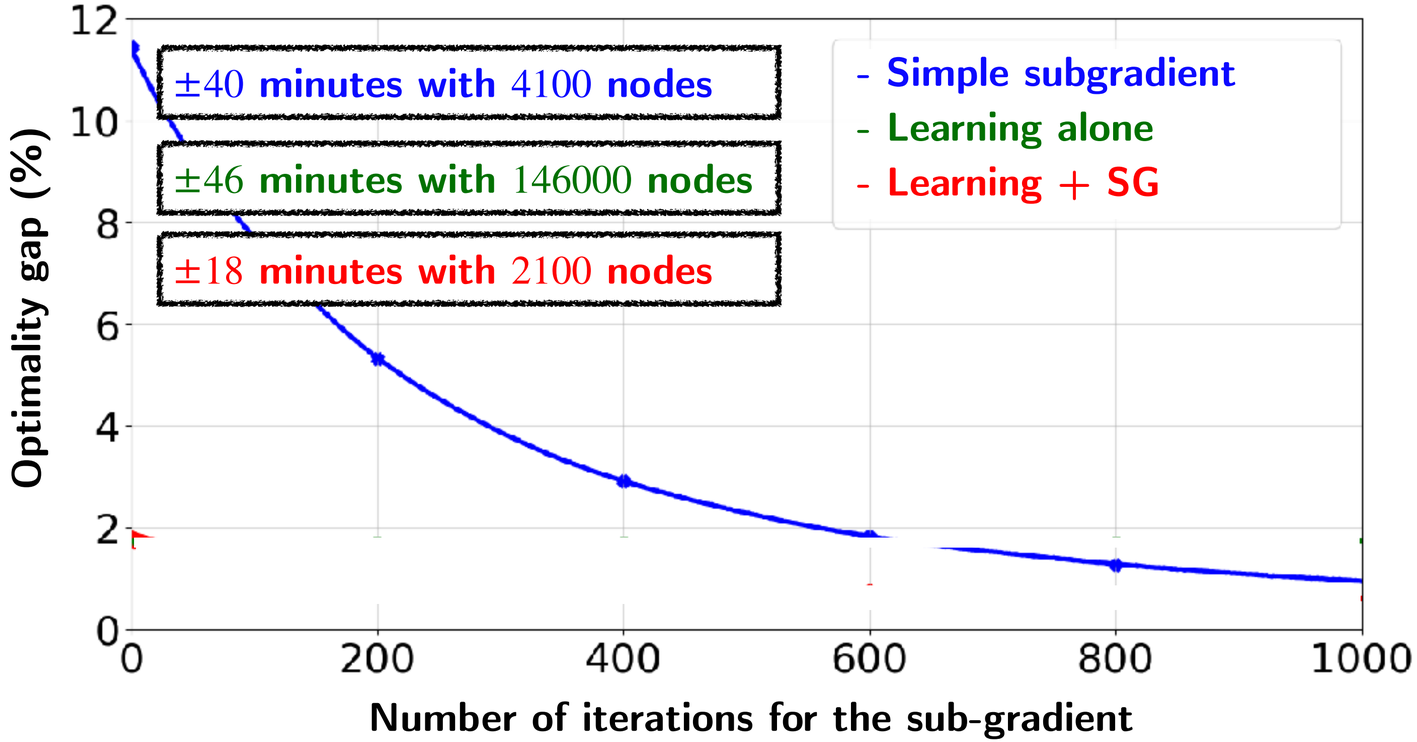
Sub-gradient update (note: it requires to solve the subproblems)

Experiments on the multidimensional knapsack

Multidimensional knapsack

Size: 100 items

Dimension: 5 constraints



Metric: quality of the dual bound obtained at the root node

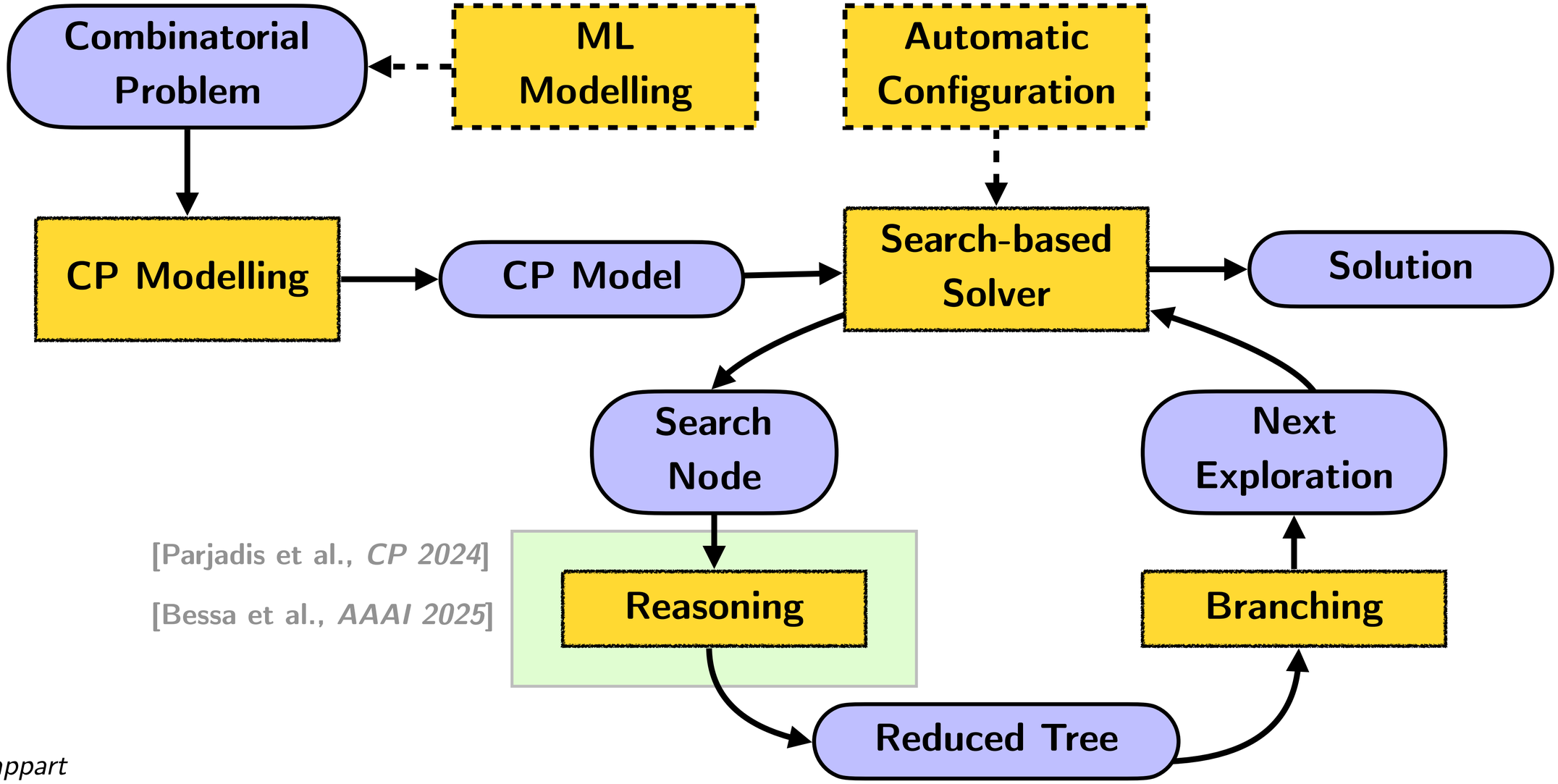
Observation 1: simple subgradient requires a lot of iterations to find good bounds

Observation 2: learning alone manages to get directly good bounds

Observation 3: learning to initialize sub-gradient quickly gives better bounds

Agenda for today

Learning - combined with other mechanisms - can be used for pruning in CP



[Parjadis et al., *CP 2024*]

[Bessa et al., *AAAI 2025*]

Survey on the topic

JAIR Journal of Artificial Intelligence Research

Dec 30, 2025

Special Track: CP and ML

Combining Constraint Programming and Machine Learning: From Current Progress to Future Opportunities

Q. Cappart

T. Guns

M. Lombardi

G. Pesant

D. Tsouros

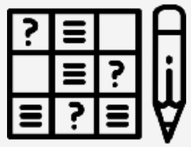
First topic: how CP has been improved with machine learning

Second topic: how machine learning has been improved with CP

Third topic: analysis of current limitations and promising opportunities

Survey on the topic: ML for improving CP

CP Modelling



Constraint acquisition

Learning objective functions

Automatic model generation

Search-based Solver

Large neighborhood search

Ant colony optimization

Belief propagation

...

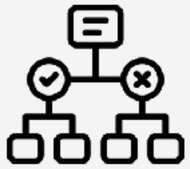
Reasoning

Deduction of nogoods

Finding Lagrangian multipliers

Not so much work!

Branching

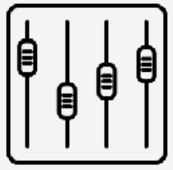


Value-selection heuristic

Variable-selection heuristic

Offline and online learning

Automatic Configuration



Problem-specific methods

Instance-specific methods

Portfolio-based selection

...

ML Modeling

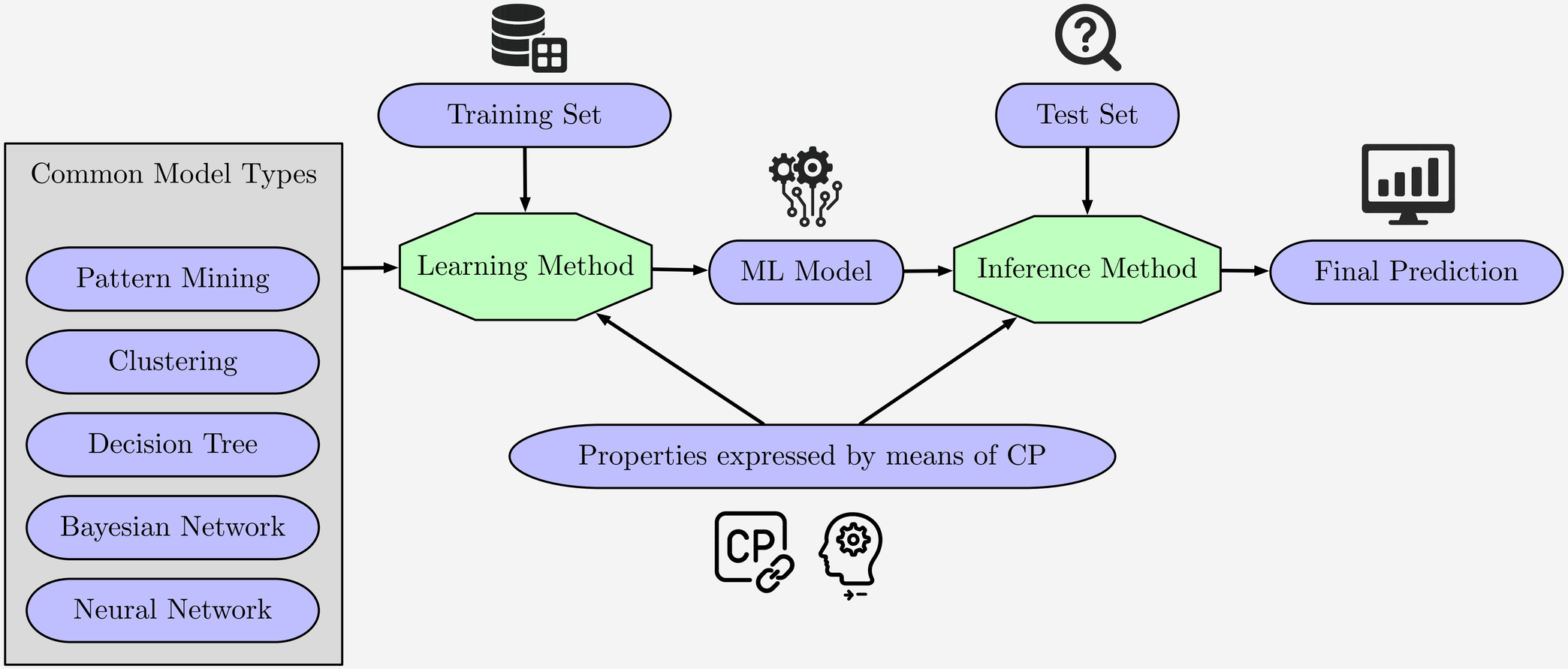
Generic representation

Problem-based representation

Focus on scheduling

Survey on the topic: CP for improving ML

Improving machine learning with CP



Survey on the topic: GitHub repository

Paper list

Improving contraint programming with machine learning

This list categorizes key research directions in the the role of ML in enhancing the CP pipeline.

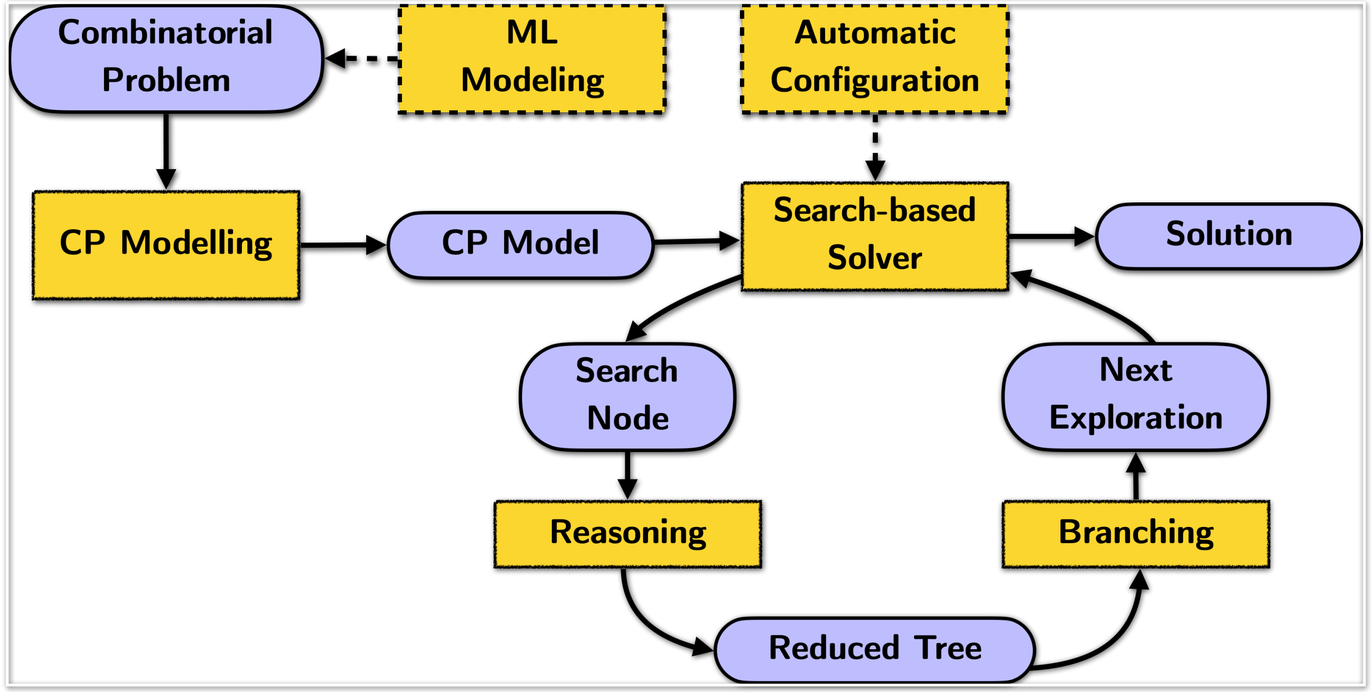
1. Learning to model - Learning constraints

- [AIJ'2017] Constraint acquisition
- [AAAI'2018] Learning constraints from examples
- [AIJ'2023] Learning constraints through partial queries
- [CPAIOR'2024] Acquiring Constraints for a Non-linear Transmission Maintenance Scheduling Problem
- [JAIR'2025] A Query-Based Constraint Acquisition Approach for Enhanced Precision in Program Precondition Inference

1.1 Passive constraint acquisition

- [ECML'2005] A SAT-based version space algorithm for acquiring constraint satisfaction problems
- [CP'2022] Constraint acquisition based on solution counting
- [IJCAI'2023] Learning constraint networks over unknown constraint languages
- [AMAA'2021] Classifier-based constraint acquisition
- [ECAI'2020] Robust constraint acquisition by sequential analysis
- [IJAR'2024] A statistical approach to learning constraints
- [AAAI'2024] What are the rules? Discovering constraints from data
- [ICTAI'2021] Unsupervised constraint acquisition
- [CP'2011] A constraint seeker: Finding and ranking global constraints from examples
- [CP'2012] A model seeker: Extracting global constraint models from positive examples
- [IJAIT'2024] Enhancing Constraint Acquisition through Hybrid Learning: An Integration of Passive and Active Learning Strategies
- [CP'2016] Learning parameters for the sequence constraint from solutions
- [CP'2017] Learning the parameters of global constraints using branch-and-bound
- [ICTAI'2010] On learning constraint problems
- [CP'2022] Learning constraint programming models from data using generate-and-aggregate
- [IJCAI'2022-Workshop] Extrapolating Constraint Networks by Symbolic Classification
- [AAAI'2025] Generalizing Constraint Models in Constraint Acquisition
- [SETN'2024] The Impact of Solution Diversity on Passive Constraint Acquisition

Living document: GitHub repository to add new references



Feel free to contribute with your own references!

Many thanks!!

UCLouvain

CIRRELT

POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Mila

CORAIL

icteam

Hydro Québec

GIRO

servicenow

E AI

SNCF

Dash

FASTERCOM

IVADO

NSERC CRSNG

Digital Research Alliance of Canada

mitacs

And my wonderful team at UCLouvain/PolyMTL :-)

Research group

corail-research.github.io

Full survey + references

(JAIR 2025)

Slides

qcappart.github.io

quentin.cappart@uclouvain.be