Enhancing constraint programming with machine learning

Current challenges and future opportunities

CP 2024 - Girona

POLYTECHNIQUE MONTRÉAL

Mila

CIRRELT

CORAIL

Combinatorial Optimization and Reasoning in Artificial Intelligence Laboratory

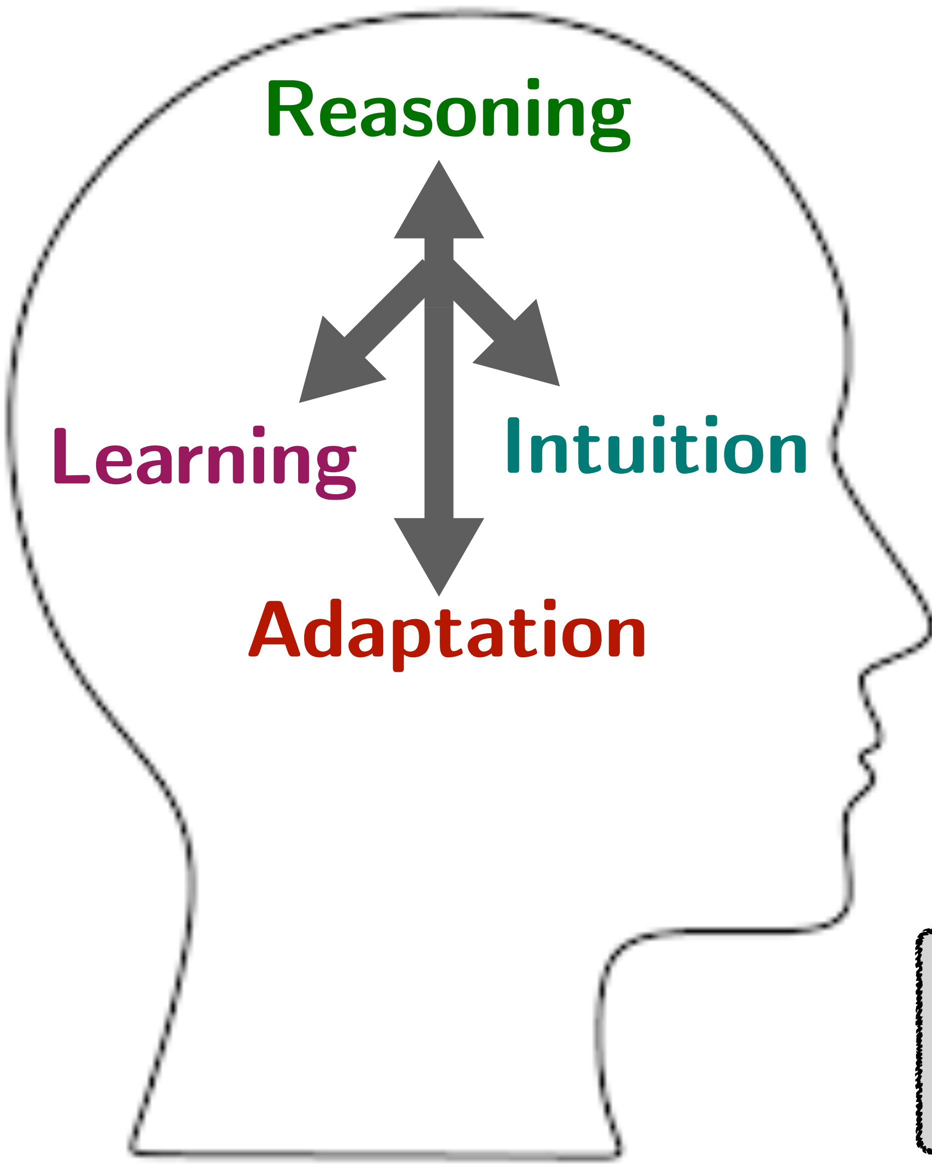
CP 2024 GIRONA

ACP

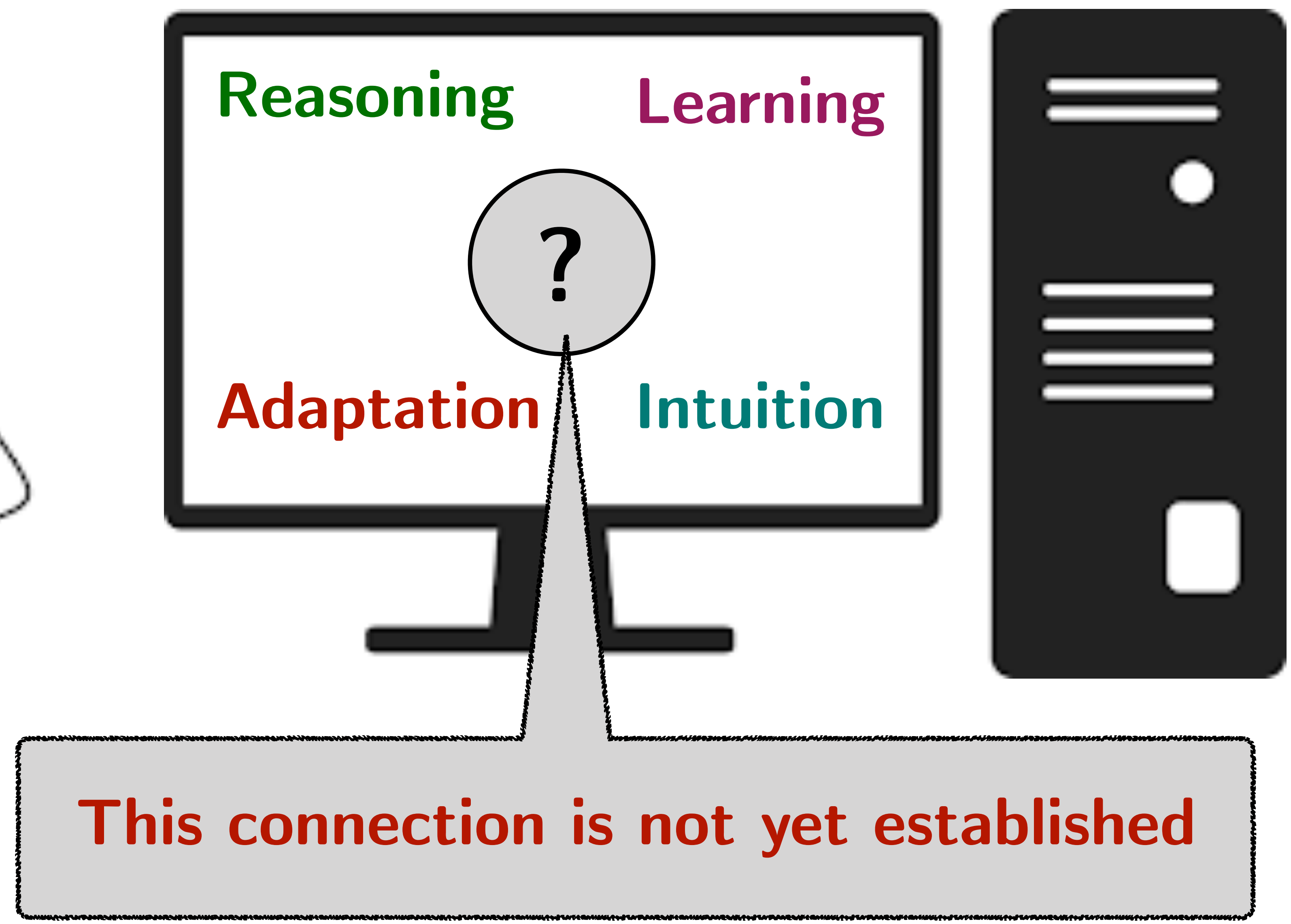
Quentin Cappart

Human intelligence versus artificial intelligence

Human intelligence



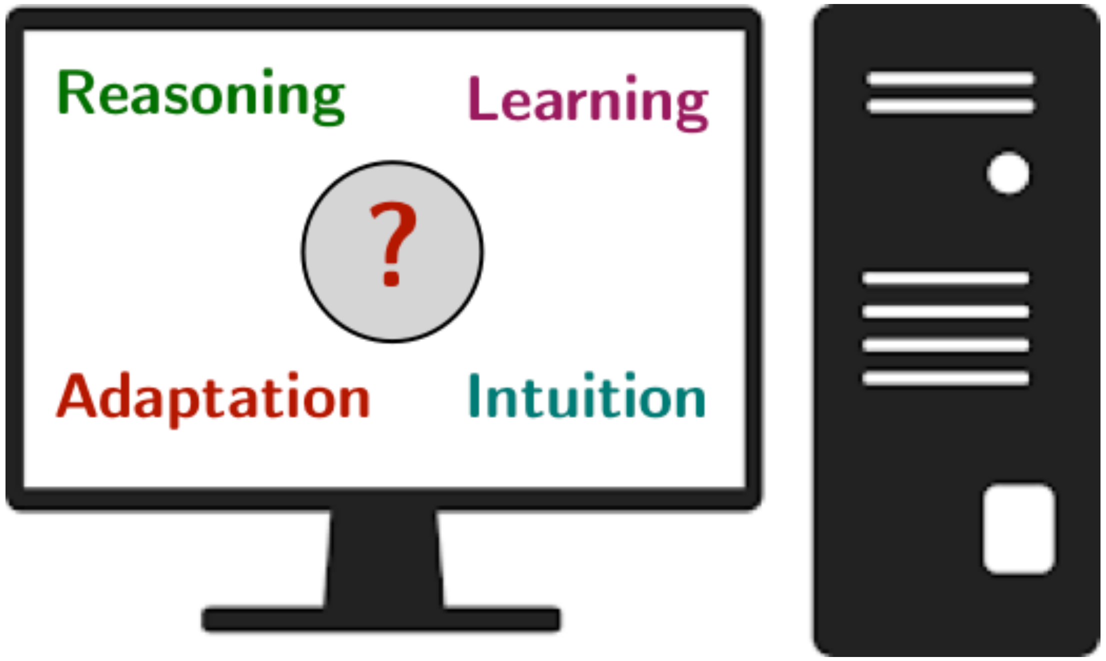
Artificial intelligence



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

Goal: **providing a better solving process for combinatorial problems**

Constraint programming as a unifying framework



My research hypothesis

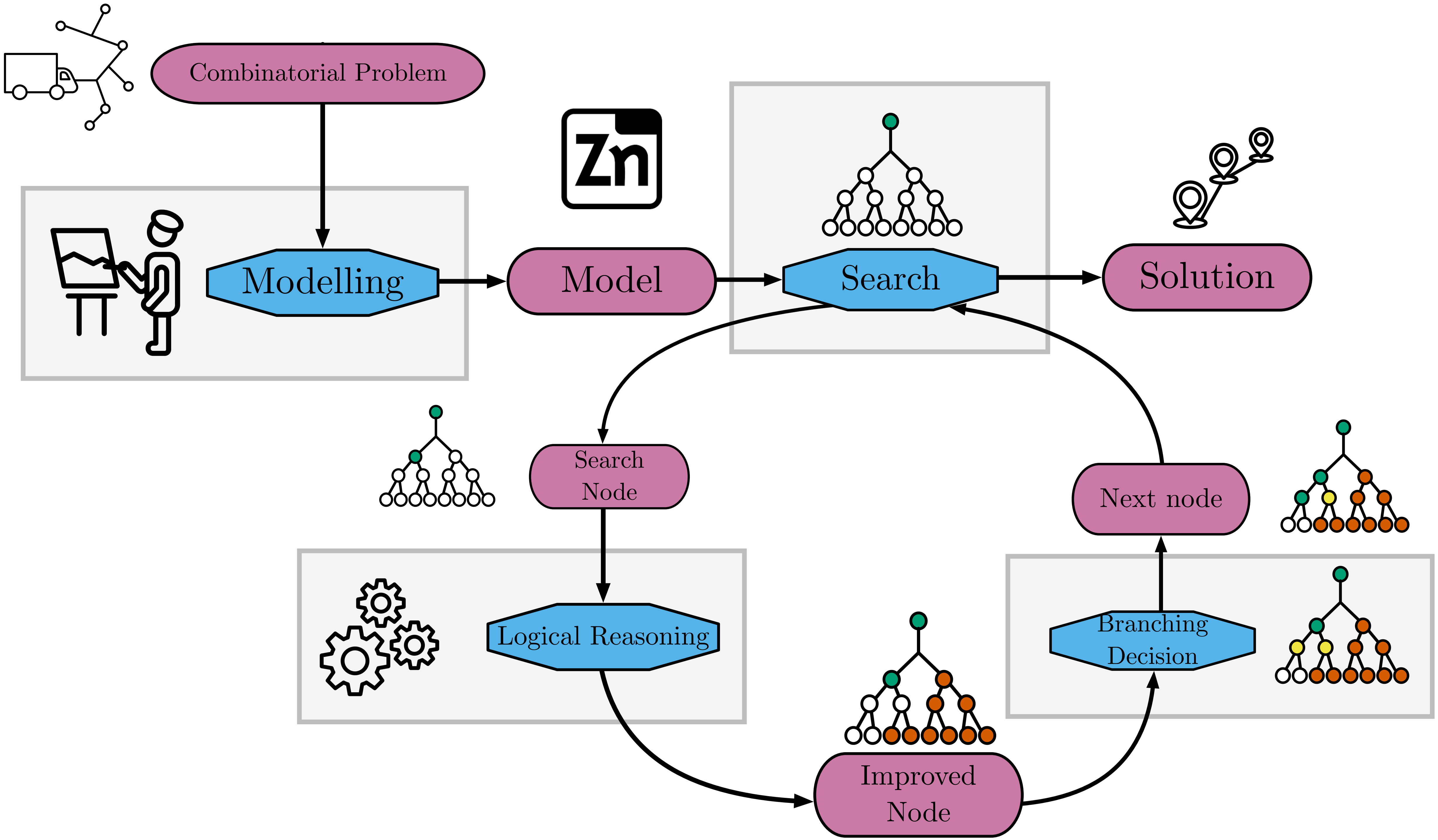
***Constraint programming* can be a hosting technology for building this hybrid AI**

CP = model + propagation + search + learning

- model: Huge flexibility
- propagation: Logical reasoning
- search: Following an intuition
- learning: Leveraging past knowledge

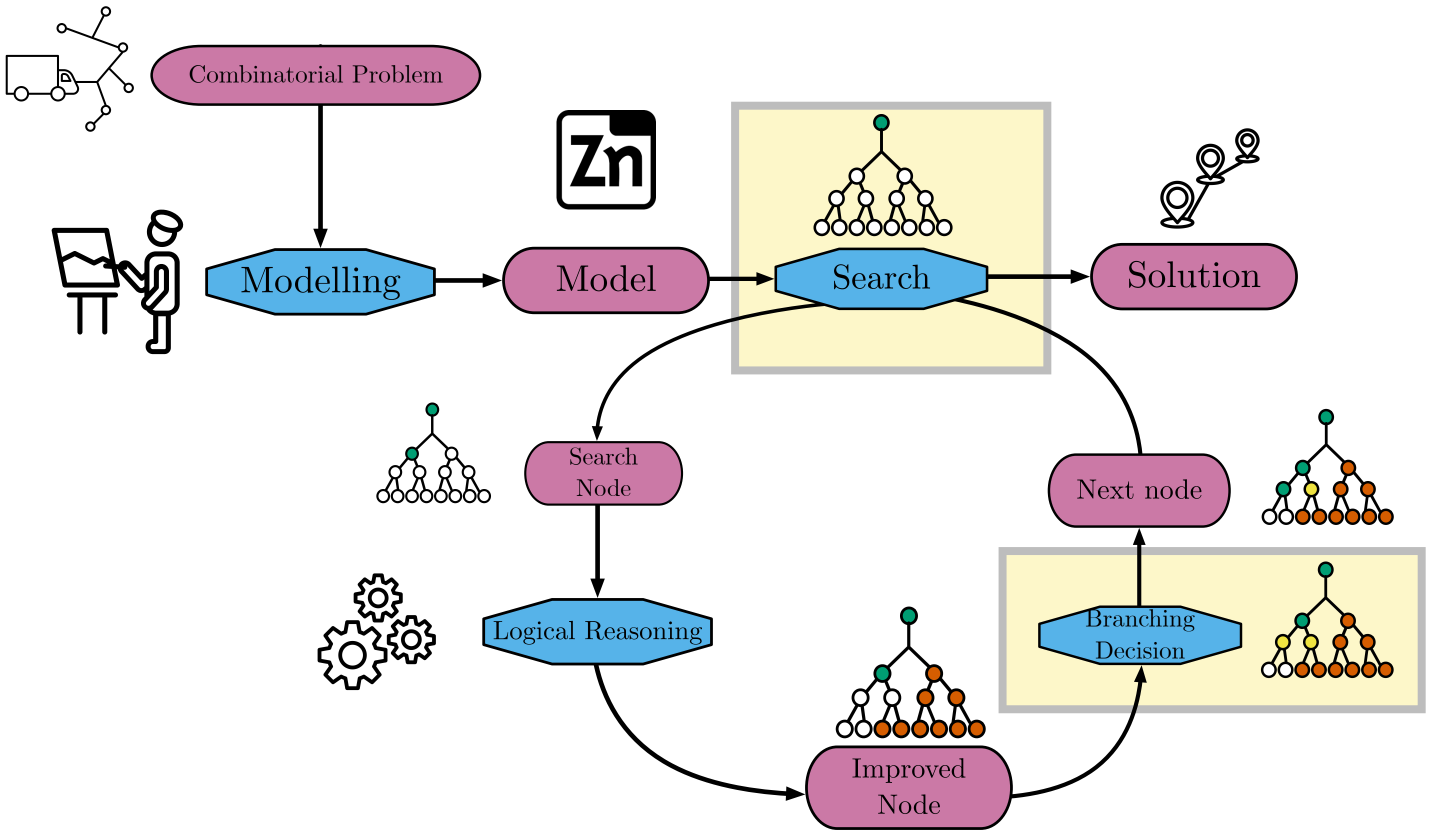
? How learning can be used to enhance the efficiency of CP solvers ?

Summary of the CP pipeline



Each decisional step has the potential to be improved with learning

Enhancing CP with learning



My first intuition: search is a good candidate as it heavily relies on imperfect heuristics

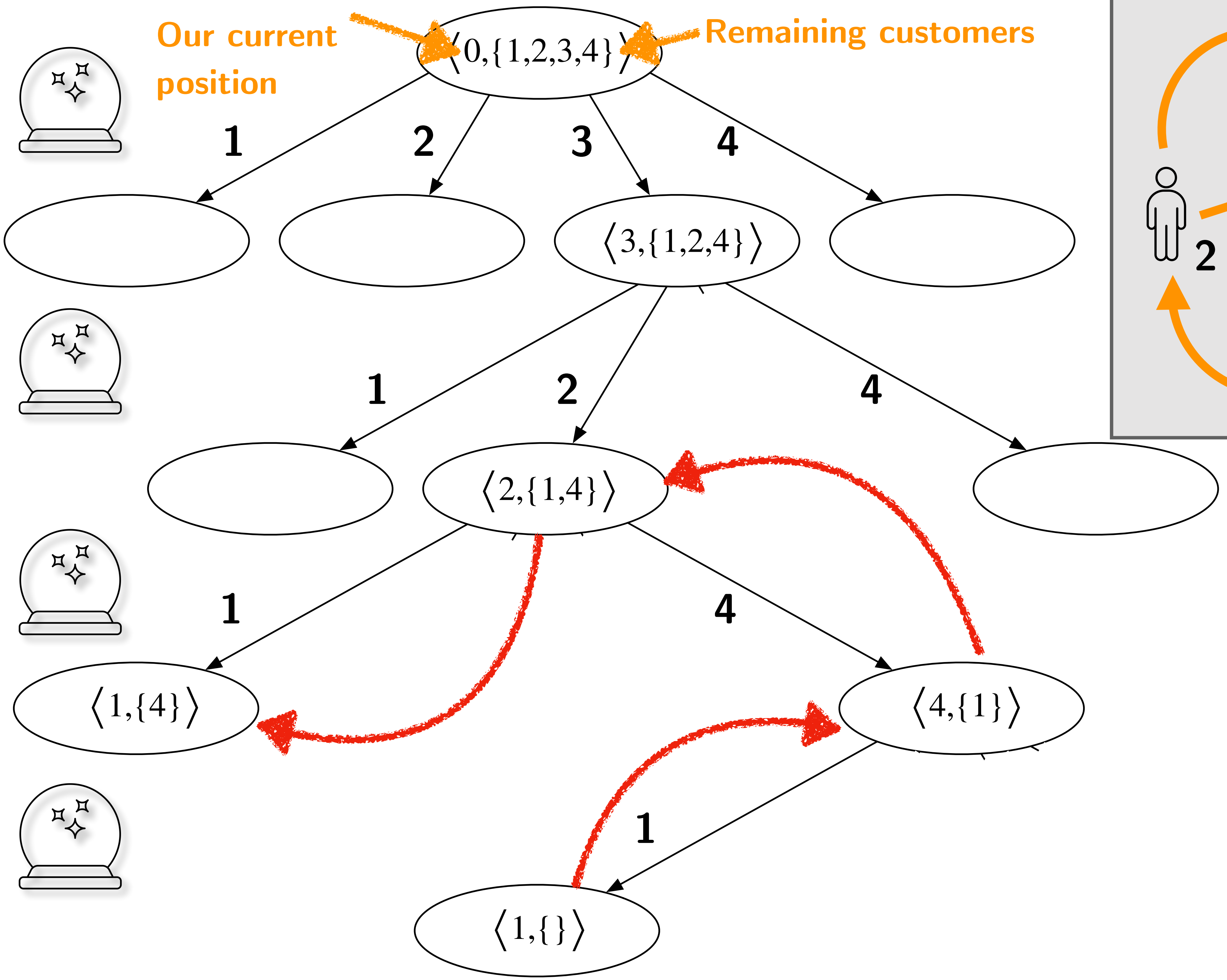
Consolidating argument: thanks to backtrack we can recover from bad predictions

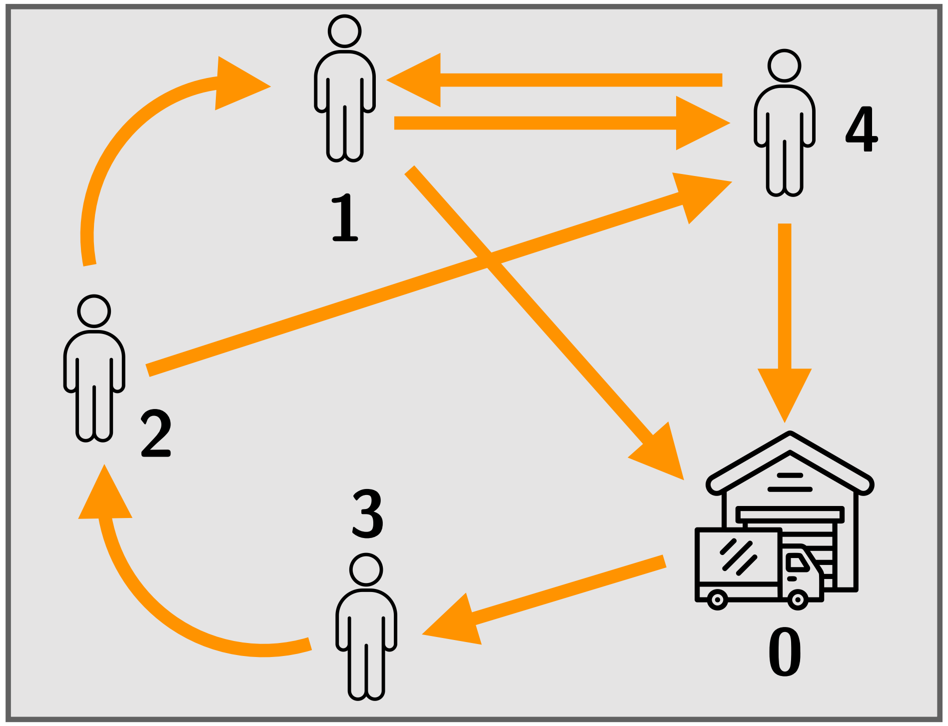
Context: learning showed promising results for MIP (Gasse et al., Neurips 2019)

My first research direction was to learn appropriate value-selection heuristic

(with the intent to go further by learning variable-selection heuristics)

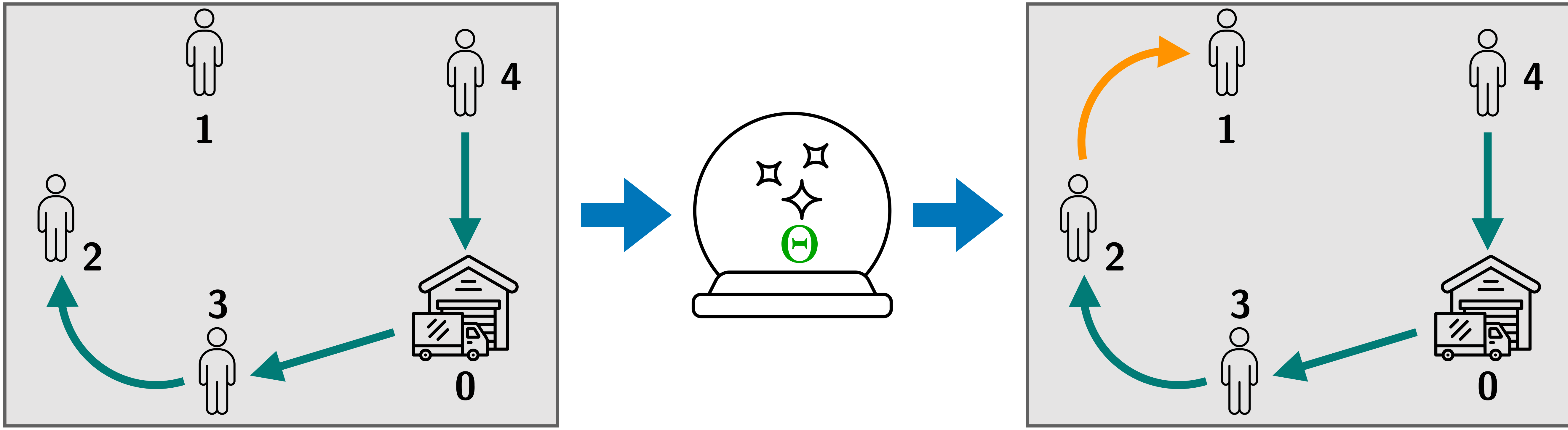
CP search with a learned heuristic on TSP





The ability to recover from bad decisions is fundamental for complete approaches

CP search with a learned heuristic on TSP



G, x | $f_\Theta(G, x)$ | $d \sim D(x)$

Instance partially solved | Predictive model | Value to branch on

Variable to branch on

? How can we build this predictive model (this magical crystal ball) ?

Challenge 1: we do not know what is the best choice (i.e., we do not have labels)

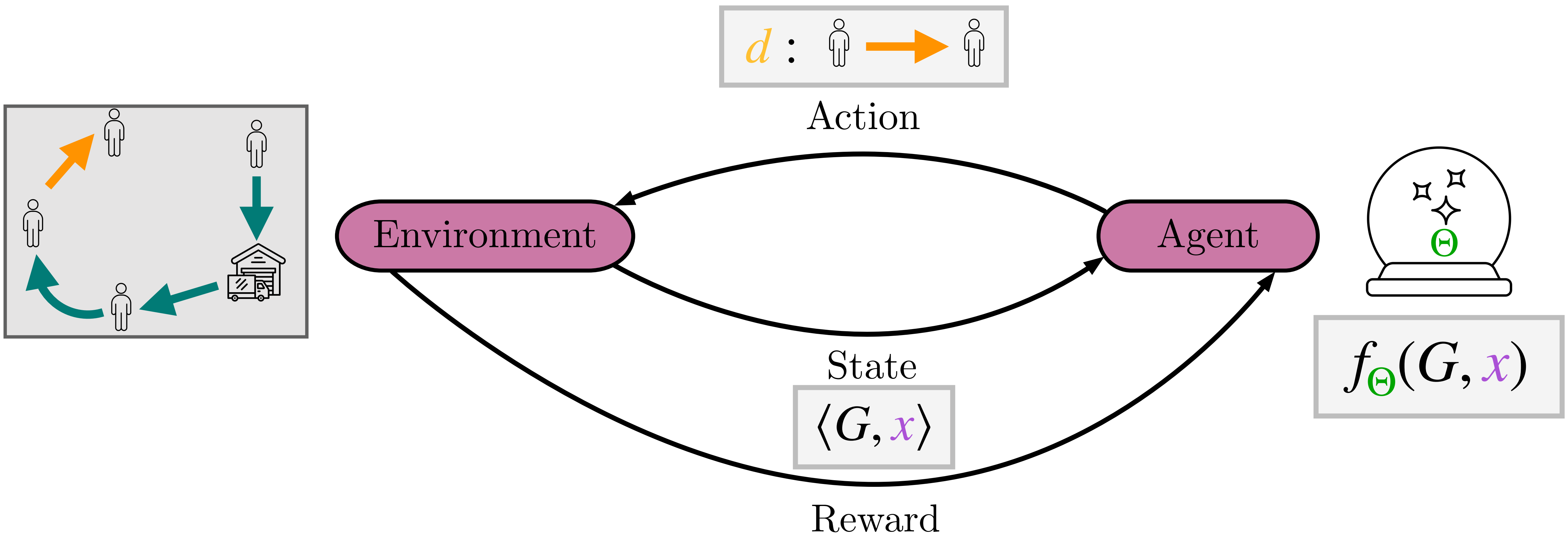
Challenge 2: difficult to represent a combinatorial problem in a proper way for learning

Tackling the first challenge (Cappart et al., AAAI 2021)

Challenge 1: we do not know what is the best choice

Consequence: tricky to rely on supervised learning

Our initial proposition: leverage reinforcement learning instead



Prioritize feasibility THEN quality

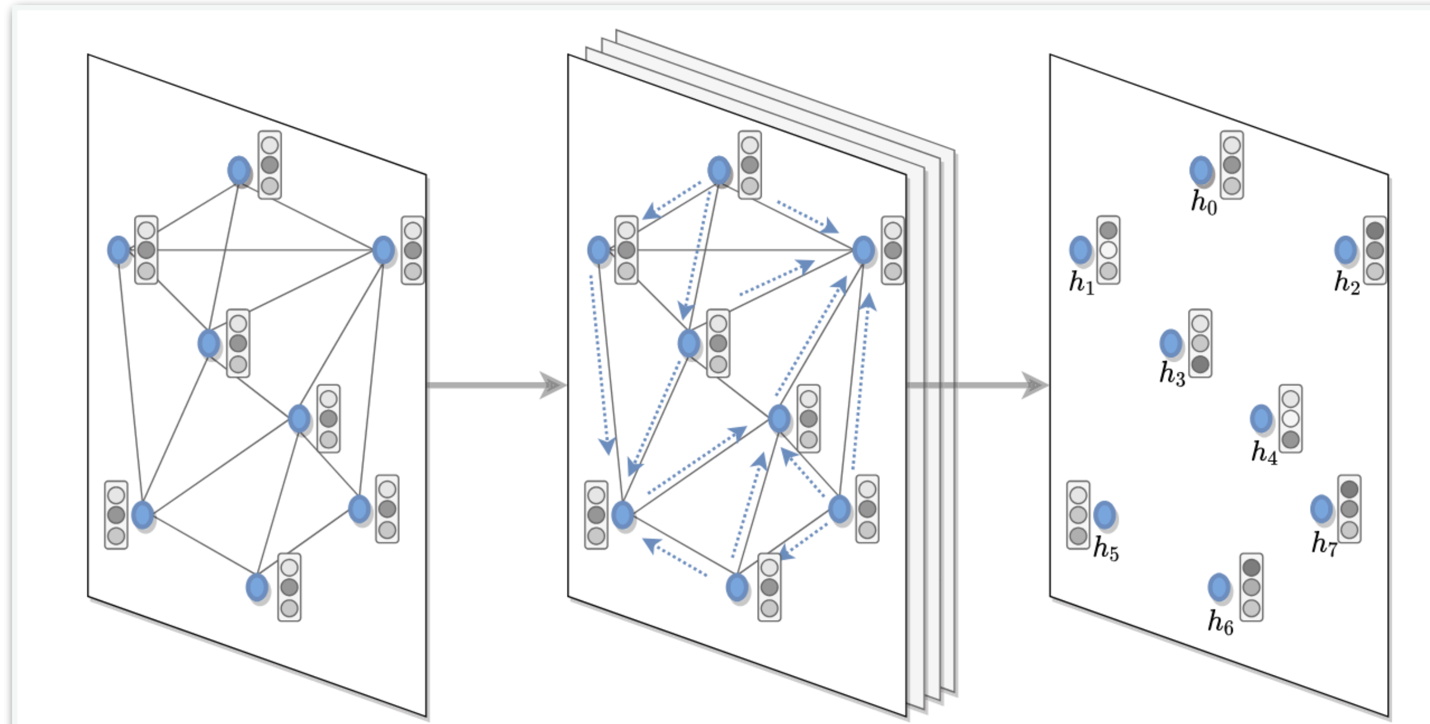
Main idea: unveiling the connections of RL with CP through dynamic programming

A first proof of concept (Cappart et al., AAAI 2021)

Algorithm 1: BaB-DQN Search Procedure.

▷ **Pre:** $\mathcal{Q}_p$ is a COP having a DP formulation.
▷ **Pre:** **w** is a trained weight vector.
$\langle \texttt{X}, \texttt{D}, \texttt{C}, \texttt{O} \rangle := \texttt{CPEncoding}(\mathcal{Q}_p)$
$\mathcal{K} = \emptyset$
$\Psi := \texttt{BaB-search}(\langle \texttt{X}, \texttt{D}, \texttt{C}, \texttt{O} \rangle)$
while Ψ **is not completed do**
  $\texttt{s} := \texttt{encodeStateRL}(\Psi)$
  $\texttt{x} := \texttt{takeFirstNonAssignedVar}(\texttt{X})$
  if $\texttt{s} \in \mathcal{K}$ **then**
    $v := \texttt{peek}(\mathcal{K}, \texttt{s})$
  else
    $v := \text{argmax}_{u \in D(\texttt{x})} \hat{Q}(\texttt{s}, u, \mathbf{w})$
  end
  $\mathcal{K} := \mathcal{K} \cup \{\langle \texttt{s}, v \rangle\}$
  $\texttt{branchAndUpdate}(\Psi, \texttt{x}, v)$
end
return $\texttt{bestSolution}(\Psi)$

Challenge 2: how to represent a COP for learning?



Representing a problem as a graph, and feeding it to a graph neural network

Experiments: results showed that relevant branching decisions can be learned

Limitation: far below state-of-the-art results and limited to relatively small instances

Technical difficulty: not-so-efficient to embed learning into existing CP solvers

These first results showed the promise in this direction and drove my research

Improvements in learning to branch

(Chalumeau, Coulon et al., CPAIOR 2021)

SeaPearl.jl

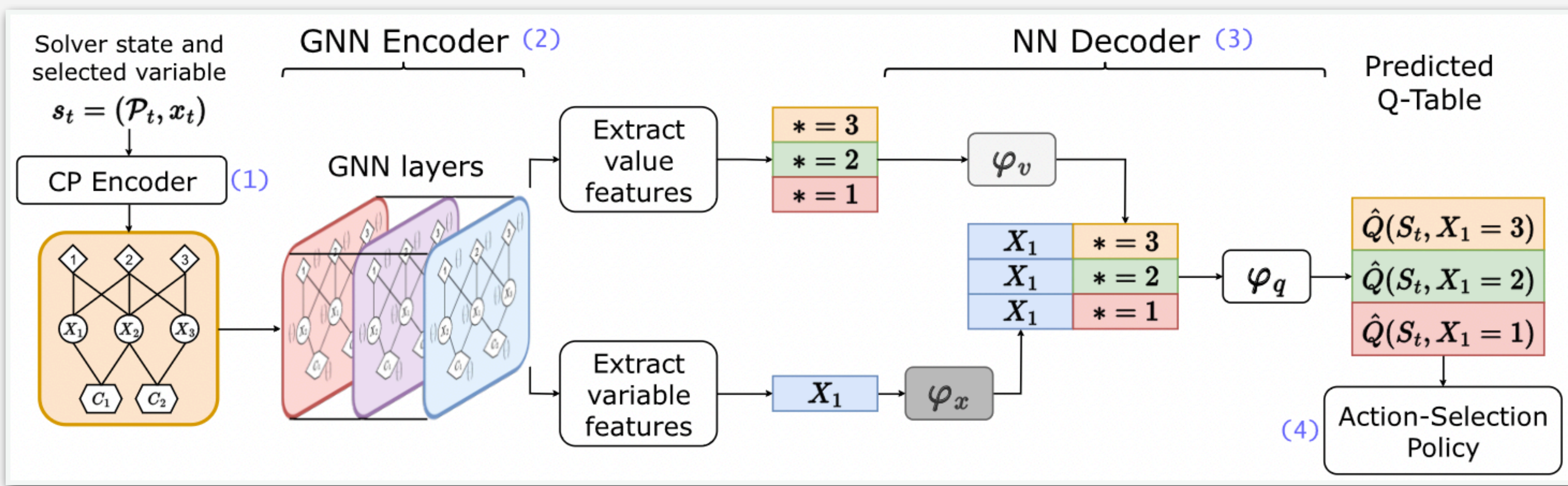
Seapearl: minimalist CP solver aiming to **ease the integration of learning**

Improvement: carrying out the **learning inside the solver** and not outside

Limitation 1: **huge overhead** in calling a GNN at each search node

Limitation 2: **challenging to train** (large resources, instability, etc.)

(Marty et al., CP 2023)



Main idea: **mainly simplifying the framework and downgrading our ambition!**

Improvement: **forget optimality proof** (too difficult with value-selection heuristics)

New goal: **finding quickly good solutions** (redefining the reward function)

Limitation: **still a huge overhead** in calling often the GNN

My current thoughts in learning to branch

Learning to branch comes with a LOT of challenges...

(1) **Overhead in calling a heavy model at each node** (compared to cheap heuristics)

(2) **Subjective choices in designing the reward** (feasibility, quality, optimality)

(3) **A model good at the root node may be poor at deeper levels** (*distributional shift*)

Here are few advices based on what worked the best for me

(1) **Do not use learning at each branching step** (too costly compared to heuristics)

(2) **Call the model only at the top of the tree** (where you likely have more samples)

(3) **Proving optimality may be out-of-range** (how to properly reward this?)

(4) **Focus on finding quickly good solutions** (in few nodes)

(5) **Prioritize hybrid approaches** (learning at the beginning, then use heuristics)

Going further with my thoughts...

Reinforcement learning may not be the best method for improving CP

Argument 1: much harder to train than supervised learning

Argument 2: indirectly require an unknown label (reward is an approximation of it)

Learning a value-selection heuristic may not be the best thing to do

Observation 1: learning is too costly and unstable to replace cheap heuristics

Observation 2: learning something does not always give practical improvements

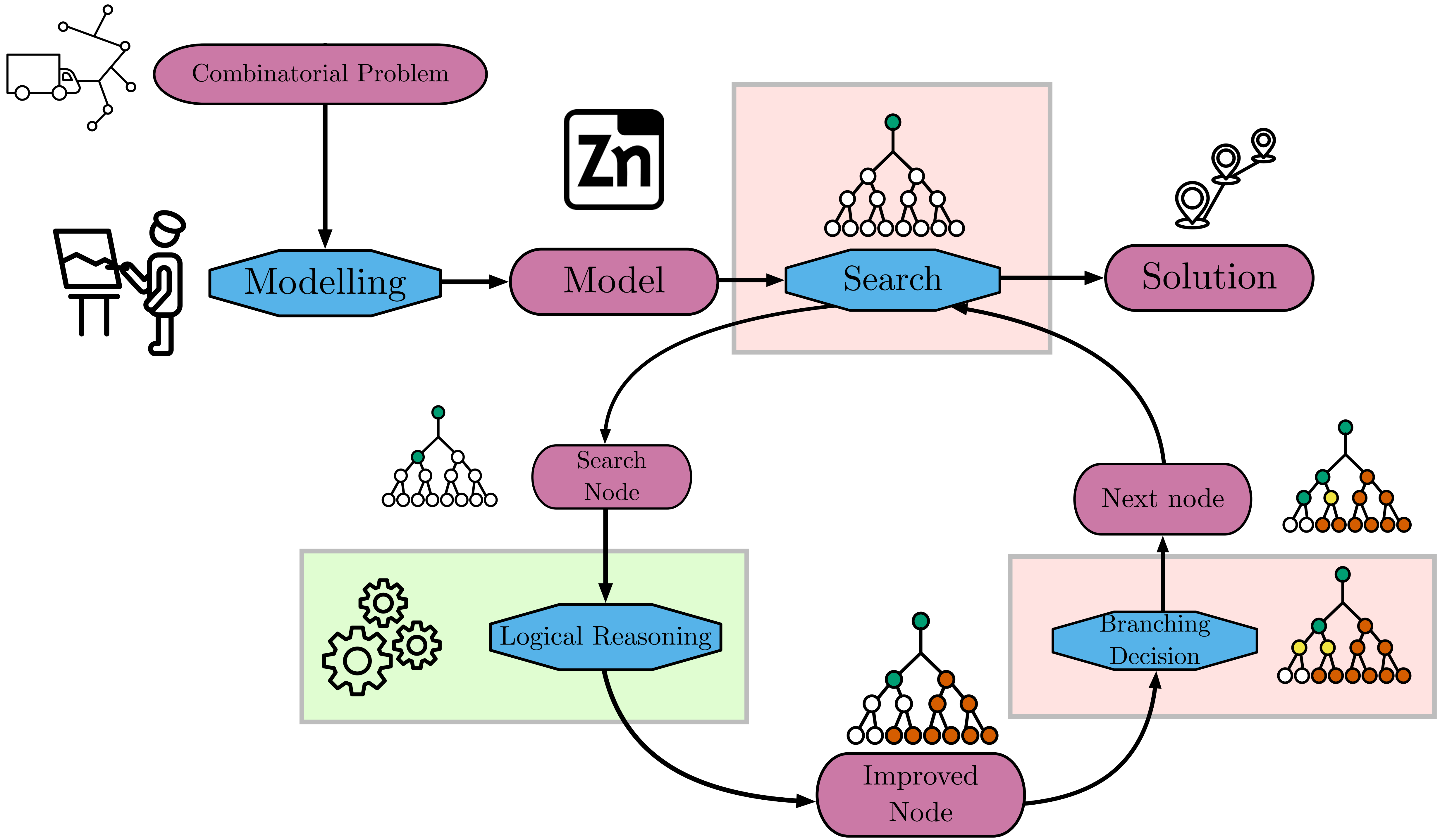
Note: please only see this as my personal opinion, and not as an irrevocable truth :-)

? What about learning a variable-selection heuristic for CP ?

My intuition: we will probably have similar challenges than for the value-selection

Ok, but what do you propose then ?

Can we learn something else?



My main current research direction is to learn how to prune the search space

(Said differently, I plan to improve the quality of filtering)

Types of propagation in a CP solver

? But how to do that ? Propagation is mainly algorithmic

CP = model + propagation + search

Fix-point

Consistency (AC3, etc.)

Global constraints

Other tools (nogoods, etc.)

Pure algorithmic

MDD

Cost-based filtering

Some propagators relies on tricky-to-get information that we propose to learn

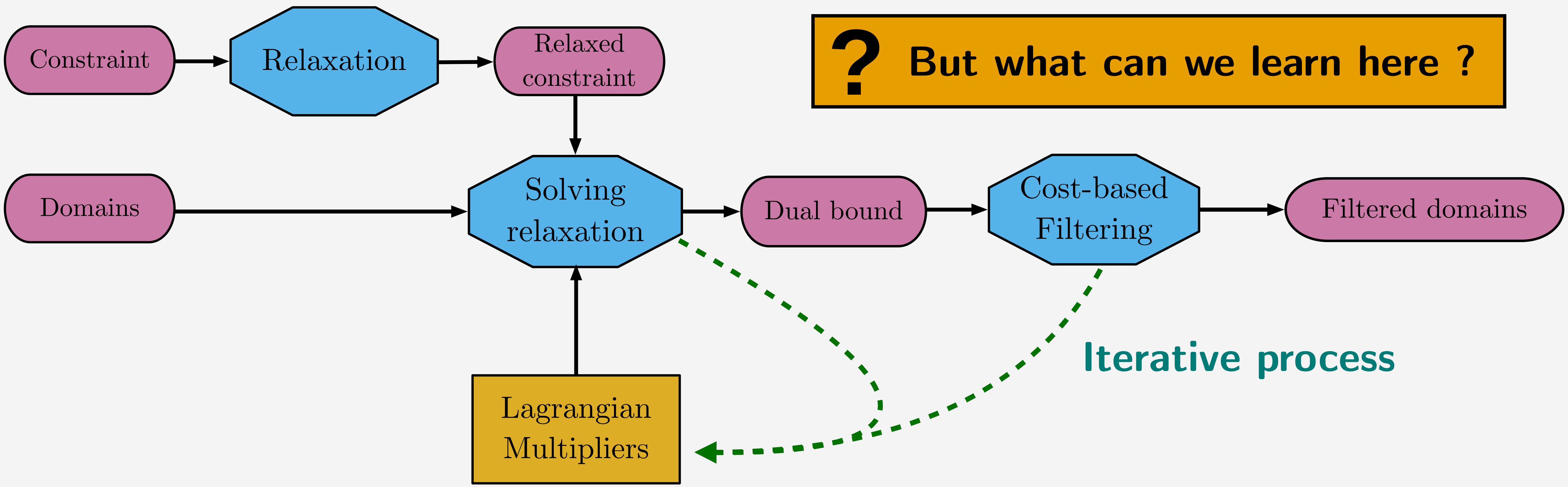
Cost-based filtering (Focacci et al., CP 1999)

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



? But what can we learn here ?

Trick: the quality of the relaxation 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

Proof of concept on *weightedCircuit* constraint

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

Proof of concept that motivates us to extend the idea to other constraints

Types of propagation in a CP solver

? Can we learn a more general bounding mechanism in CP ?

CP = model + propagation + search

- Fix-point
- Consistency (AC3, etc.)
- Global constraints
  - Pure algorithmic
  - MDD
  - Cost-based filtering
- Other tools (nogoods, etc.)
  - Lagrangian decomposition

We extend the idea to learn 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

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

$\Rightarrow$

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

$$\max \quad f(X_1, X_2, X_3) + \boxed{\mu_2} \cdot \boxed{(Y_2 - X_2)} + \boxed{\mu_3} \cdot \boxed{(Y_3 - X_3)}$$

Again, we have Lagrangian multipliers, but in a more generic way than before

Lagrangian decomposition (Guignard and Kim, 1987)

$$\mathscr{B}(\mu_2,\mu_3) = \begin{cases} \max & f(X_1, X_2, X_3) + \mu_2 \cdot (X_2 - Y_2) + \mu_3 \cdot (X_3 - Y_3) \\ \textbf{s.t.} & C_1(X_1, X_2, X_3) \\ & C_2(Y_2, Y_3) \\ & X_1, X_2, Y_2, X_3, Y_3 \in \mathbb{N}^+ \end{cases}$$

Solving this relaxed problem will give a dual bound

? But, is it easy to solve ?

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

Consequence: each constraint can be solved independently

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

Given some multipliers, we can obtain a bound by solving several subproblems

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

$$\mathcal{B}(\mu_2,\mu_3) = \boxed{\begin{array}{l} \max \Big(f(X_1, X_2, X_3) + \mu_2 \cdot X_2 + \mu_3 \cdot X_3 \Big) \\ \textbf{s.t. } C_1(X_1, X_2, X_3) \\ \qquad X_1, X_2, X_3 \in \mathbb{N}^+ \end{array}} + \boxed{\begin{array}{l} \max \big(-\mu_2 \cdot Y_2 - \mu_3 \cdot Y_3 \big) \\ \textbf{s.t. } C_2(Y_2, Y_3) \\ \qquad Y_2, Y_3 \in \mathbb{N}^+ \end{array}}$$

? How to set the values of the multipliers ?

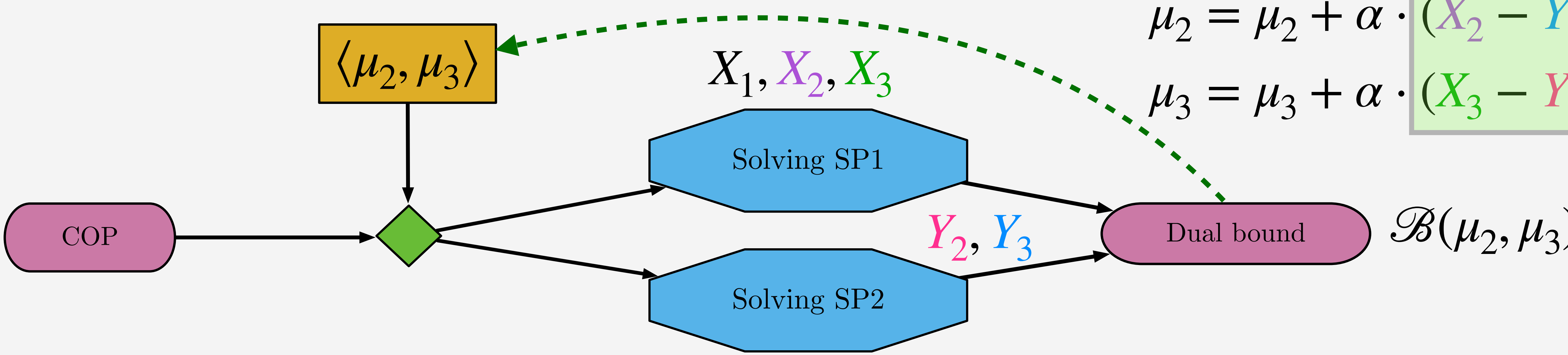
Initialization: we set the multipliers to an arbitrary value

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

$$\mu_2 = \mu_2 + \alpha \cdot (X_2 - Y_2)$$
$$\mu_3 = \mu_3 + \alpha \cdot (X_3 - Y_3)$$



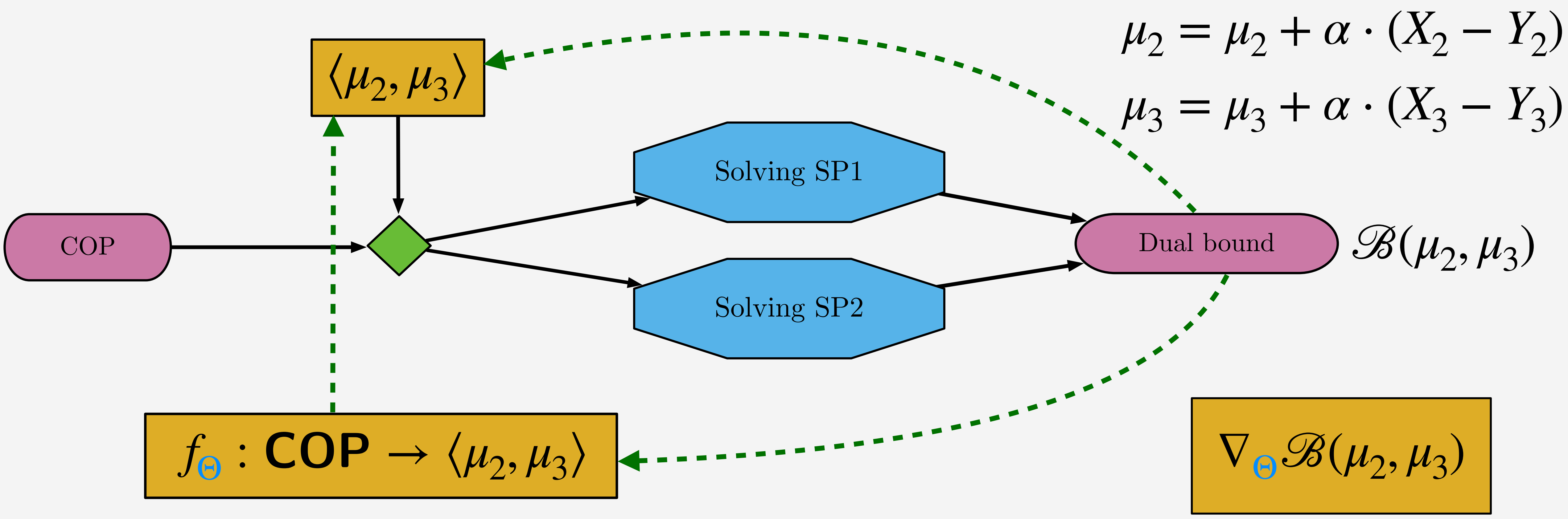
Learning multipliers for Lagrangian decomposition

This process is very costly as it requires solving few subproblems at each iteration

We propose a self-supervised learning approach to compute them

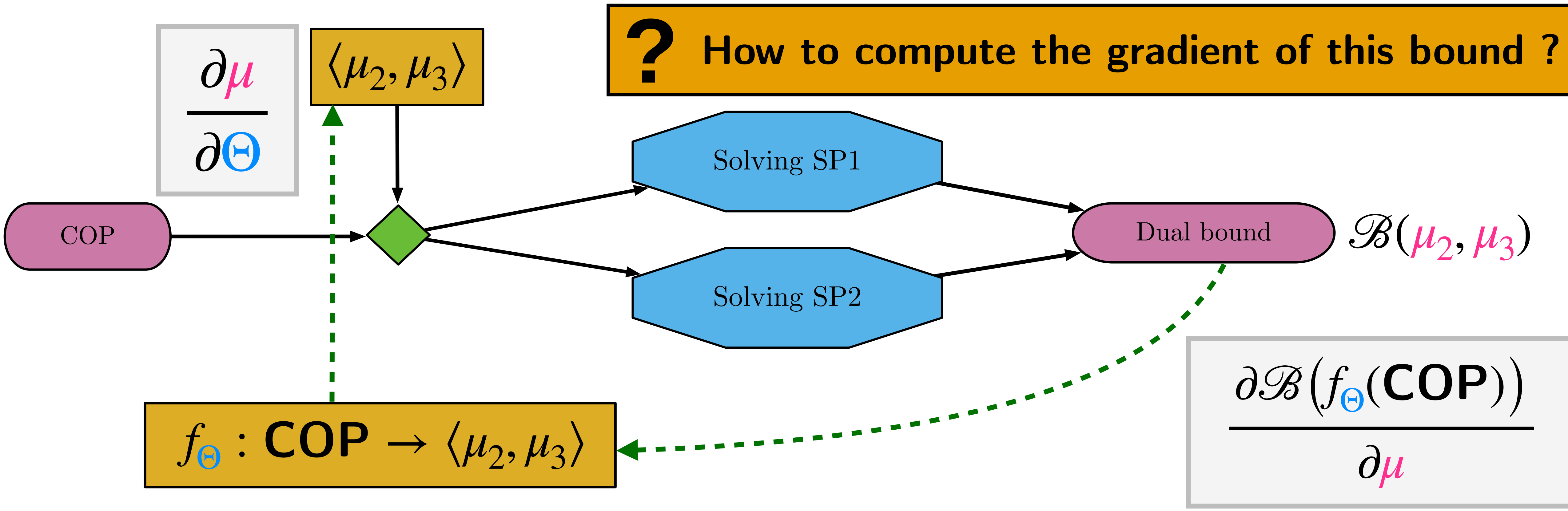
Step 1: multipliers are now obtained by a differentiable predictive model

Step 2: the model is trained end-to-end by differentiating the bound



Intuition: the optimisation process is carried out offline during a training phase

Learning multipliers for Lagrangian decomposition



$$\nabla_\Theta \mathcal{B}(\mu) = \frac{\partial \mathcal{B}\big(f_\Theta(\mathbf{COP})\big)}{\partial \mu} \times \frac{\partial \mu}{\partial \Theta} \quad \boxed{= (X - Y) \times \frac{\partial \mu}{\partial \Theta}}$$

Step 1: we use the chain-rule to uncover dependencies

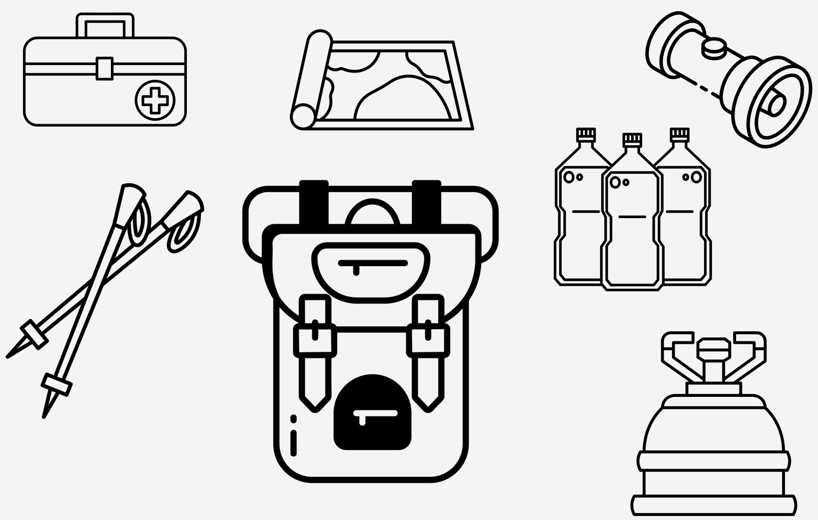
Step 2: right-term is a simple backpropagation in the predictive model

Step 3: left-term reuses the initial sub-gradient expression

Training: gradient descent on training instances (no label and no reward required)

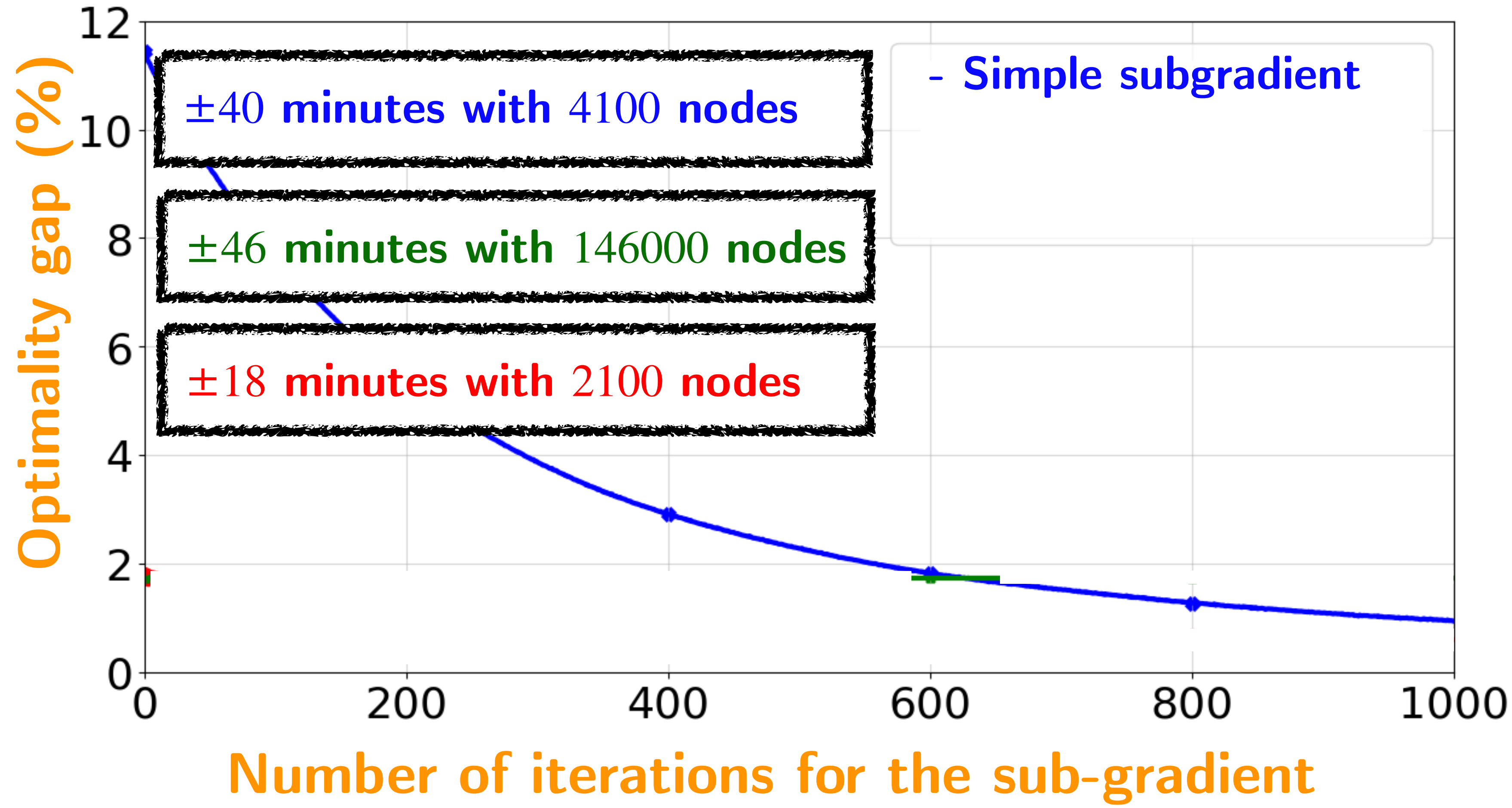
Qualitative results

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

Our results showed both the interest of learning alone or with sub-gradient

My current thoughts in learning to bound

I think that learning to bound in CP is a promising direction

(1) The lack of generic relaxation is a shortcoming of CP (compared to MIP solvers)

(2) Lagrangian relaxation (or decomposition) ensures the validity of the bound

(3) Learning can fully replace the subgradient (less filtering but cheaper)

(4) Learning can also only initialize subgradient (better filtering but more expensive)

(5) Results are encouraging in terms of execution time (not the case for branching)

But there are still questions and challenges to address...

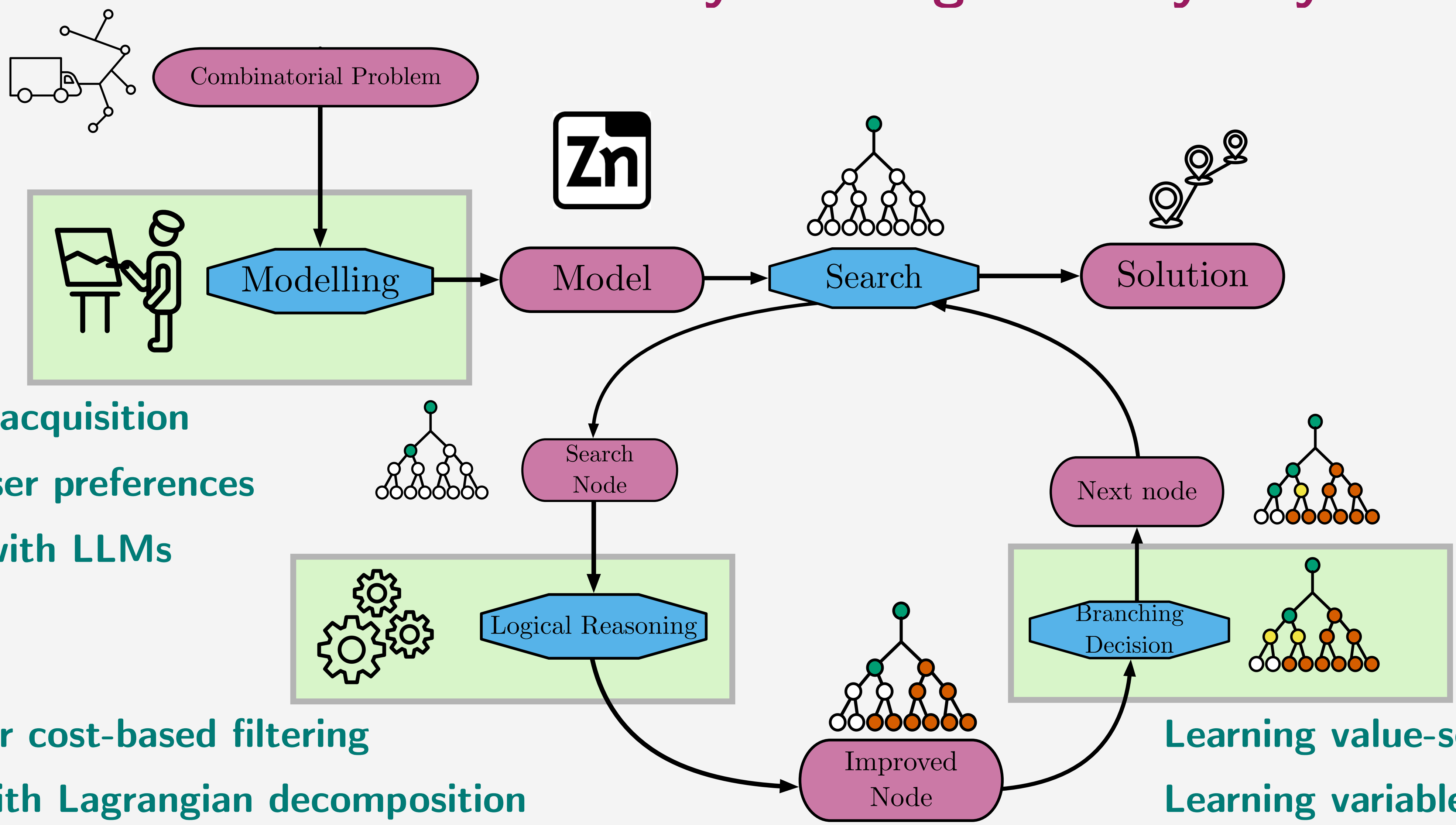
(1) Which predictive model should we use? (is GNN a right choice?)

(2) Can we have a generic representation of any COP (Boisvert et al., CPAIOR 2024)

(3) How to handle problems with only few representative instances?

Conclusion with my final notes

CP can be enhanced by learning in many ways...



Constraint acquisition

Learning user preferences

Modeling with LLMs

Learning for cost-based filtering

Learning with Lagrangian decomposition

Learning value-selection

Learning variable-selection

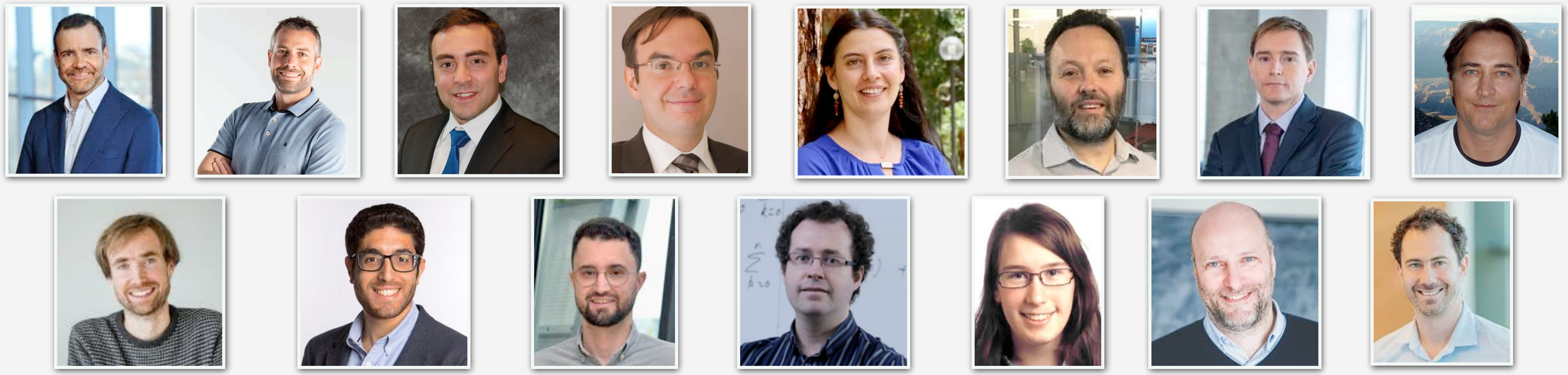
My take-home messages

(1) Learning something meaningful does not always result in improved performances

(2) Prefer to use learning for replacing costly operations instead of cheap ones

(3) Promising research direction, but a lot of challenges to handle for a practical use

Many thanks!!



Hydro Québec

GIRO

SNCF

servicenow

E AI

FASTERCOM

IVADO

NSERC CRSNG

Digital Research Alliance of Canada

mitacs

POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

CIRRELT

Mila

CORAIL

Combinatorial Optimization and Reasoning in Artificial Intelligence Laboratory

Reference list (work we carried out)

Learning to branch

- Cappart, Q., Moisan, T., Rousseau, L. M., Prémont-Schwarz, I., & Cire, A. A.
  Combining reinforcement learning and constraint programming for combinatorial optimization. [AAAI 2021]
- Chalumeau, F., Coulon, I., Cappart, Q., & Rousseau, L. M.
  Seapearl: A constraint programming solver guided by reinforcement learning. [CPAIOR 2021]
- Marty, T., François, T., Tessier, P., Gautier, L., Rousseau, L. M., & Cappart, Q.
  Learning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver. [CP 2023 - distinguished paper]

Learning to bound

- Cappart, Q., Bergman, D., Rousseau, L. M., Prémont-Schwarz, I., & Parjadis, A.
  Improving variable orderings of approximate decision diagrams using reinforcement learning. [IJOC 2022]
- Parjadis, A., Cappart, Q., Dilkina, B., Ferber, A., & Rousseau, L. M.
  Learning Lagrangian Multipliers for the Travelling Salesman Problem. [CP 2024 - Best ML paper award]
- Dabert, D., Bessa, S., Bourgeat, M., Rousseau, L.M., Cappart, Q.
  Learning Valid Dual Bounds in Constraint Programming [ArXivPreprint 2024]

Learning to model and graph neural networks

- Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., & Veličković, P.
  Combinatorial optimization and reasoning with graph neural networks. [IJCAI 2021, JMLR 2023]
- Boisvert, L., Verhaeghe, H., & Cappart, Q.
  Towards a Generic Representation of Combinatorial Problems for Learning-Based Approaches. [CPAIOR 2024]
- Barral, H., Gaha, M., Dems, A., Côté, A., Nguewouo, F., & Cappart, Q.
  Acquiring Constraints for a Non-linear Transmission Maintenance Scheduling Problem. [CPAIOR 2024]

Reference list (other related works)

Learning to branch (also in MIP)

- Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., & Dilkina, B.
  Learning to branch in mixed integer programming. [AAAI 2016]
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L., & Lodi, A.
  Exact combinatorial optimization with graph convolutional neural networks. [NeurIPS 2019]
- Song, W., Cao, Z., Zhang, J., Xu, C., & Lim, A.
  Learning variable ordering heuristics for solving constraint satisfaction problems. [EAAA 2022]

Foundations of our work on learning to prune

- Held, M., & Karp, R. The traveling-salesman problem and minimum spanning trees [Operations Research 1970]
- Guignard, M., & Kim, S. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. [Math. Prog. 1987]
- Focacci, F., Lodi, A., & Milano, M. Cost-based domain filtering [CP 1999]
- Bergman, D., Cire, A. A., & van Hoeve, W. J. Improved constraint propagation via lagrangian decomposition. [CP 2015]
- Hà, M. H., Quimper, C. G., & Rousseau, L. M. General bounding mechanism for constraint programs. [CP 2015]

Learning dual bounds (only outside CP?)

- Deng, Y., Kong, S., Liu, C., & An, B.
  Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems. [NeurIPS 2022]
- Abbas, A., & Swoboda, P. Doge-train: Discrete optimization on gpu with end-to-end training. [AAAI 2024]

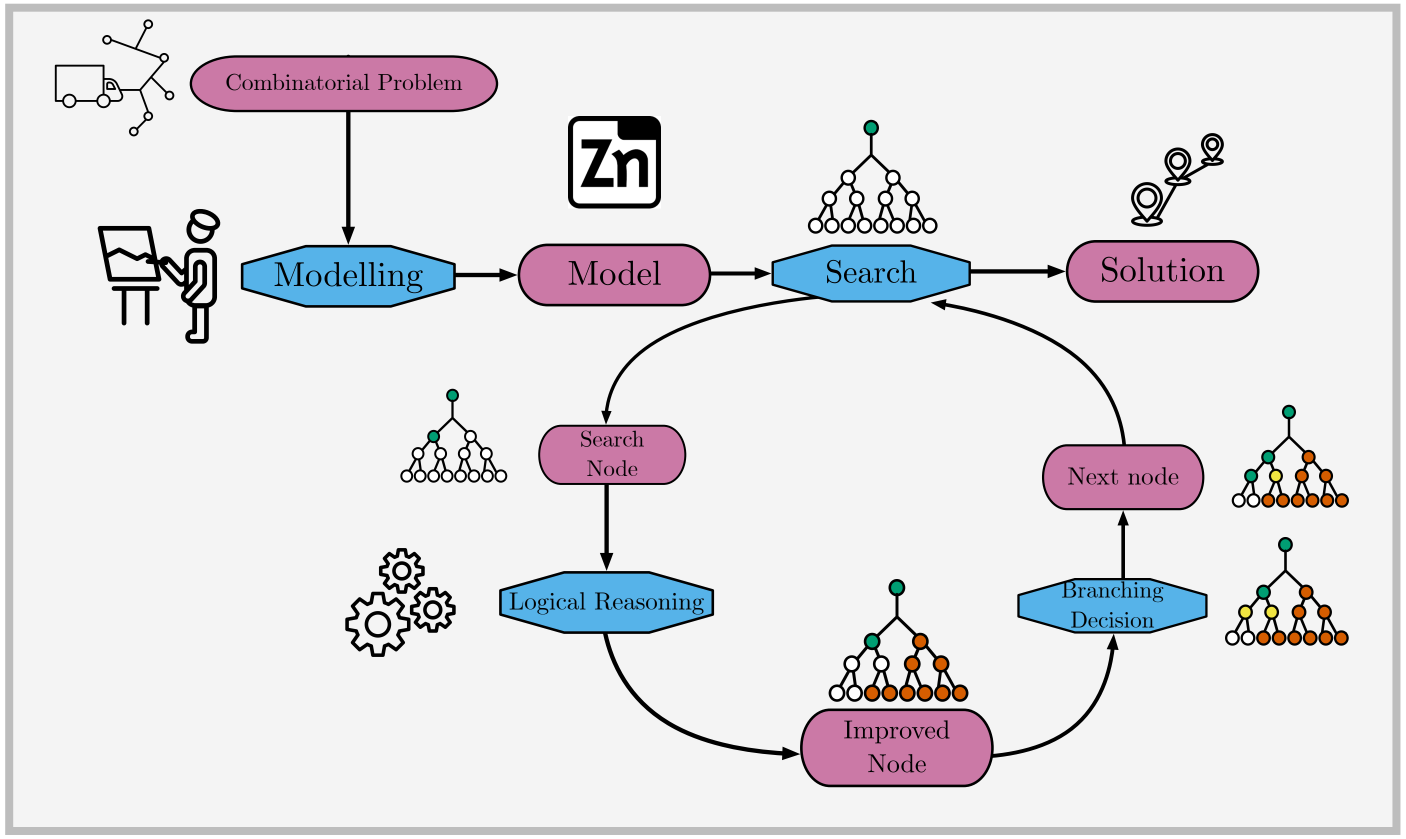
Please reach me if you know other works using learning to get dual bounds :-)

Research group: *corail-research.github.io*

Personal page: *qcappart.github.io*

Slides: *QR code (or via my personal page)*

Email: *quentin.cappart@polymtl.ca*



POLYTECHNIQUE MONTRÉAL
Mila
CIRRELT
CORAIL
Combinatorial Optimization and
Reasoning in
Artificial Intelligence
Laboratory

CP 2024 GIRONA

ACP

Quentin Cappart

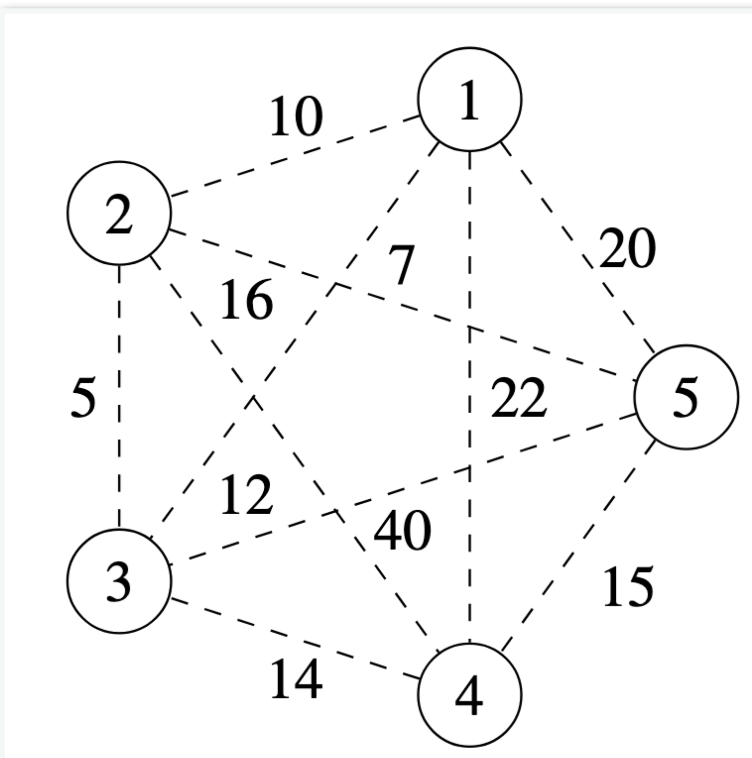
Learning Lagrangian Multipliers for the TSP

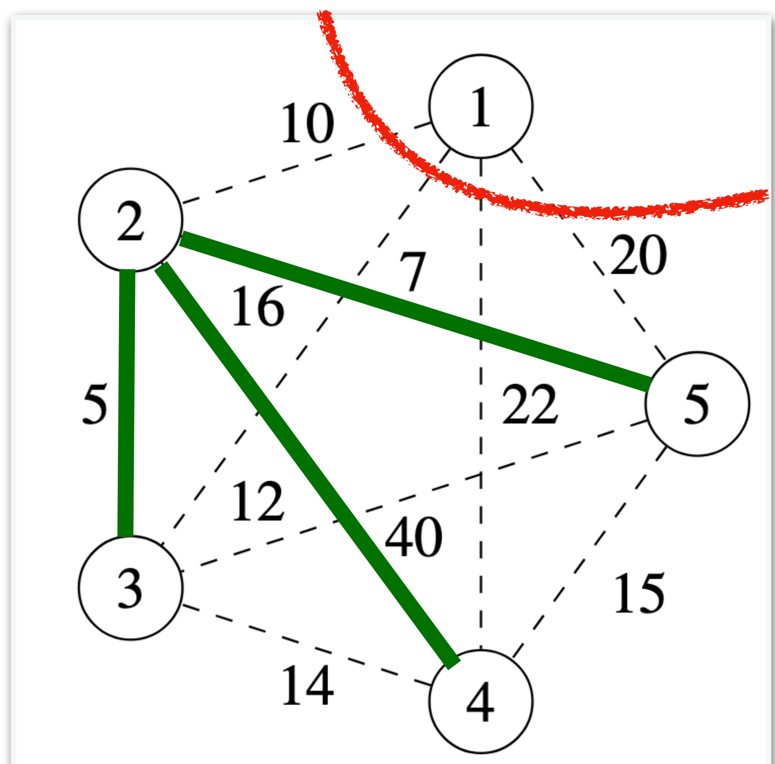
WeightedCircuit(X, G, d) : **ensure that variables X form a TSP tour in G of cost $\leq d$**

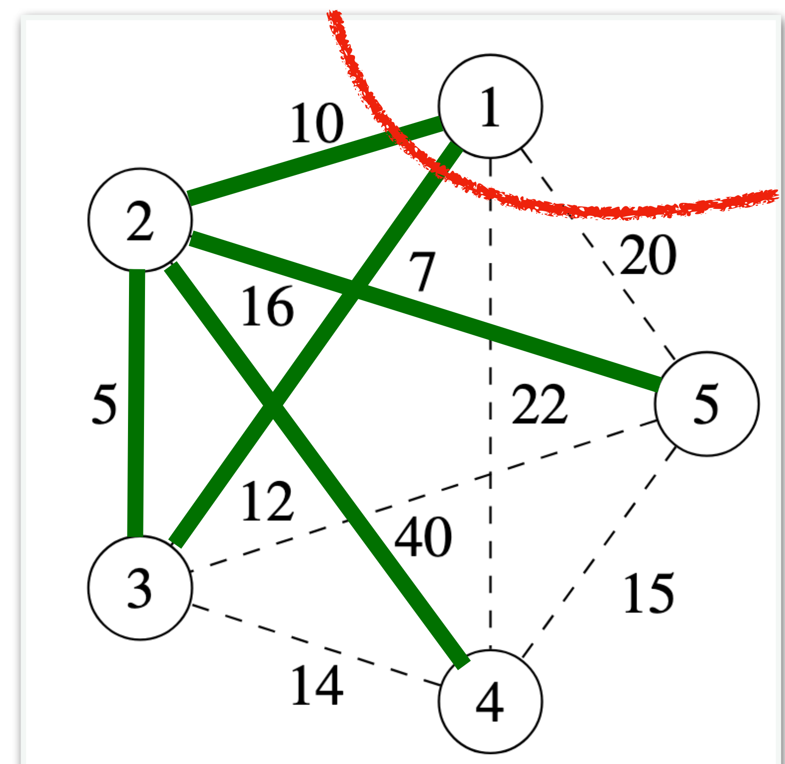
1-tree relaxation(X, G) : **allows the TSP to go twice in a specific node**

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

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







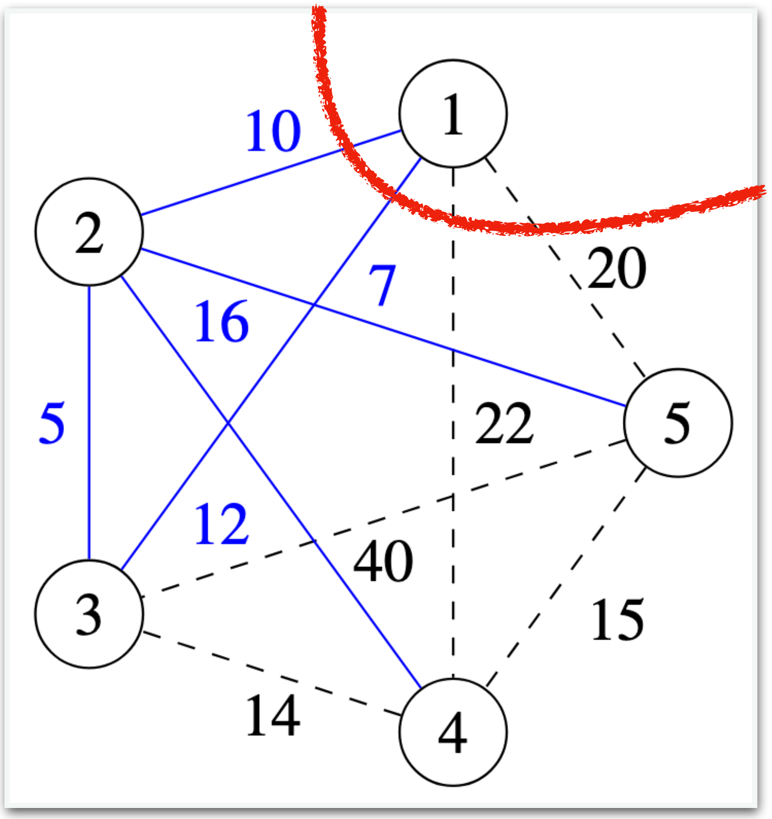
Optimal TSP cost: 62

Relaxation cost: 50

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

Learning Lagrangian Multipliers for the TSP

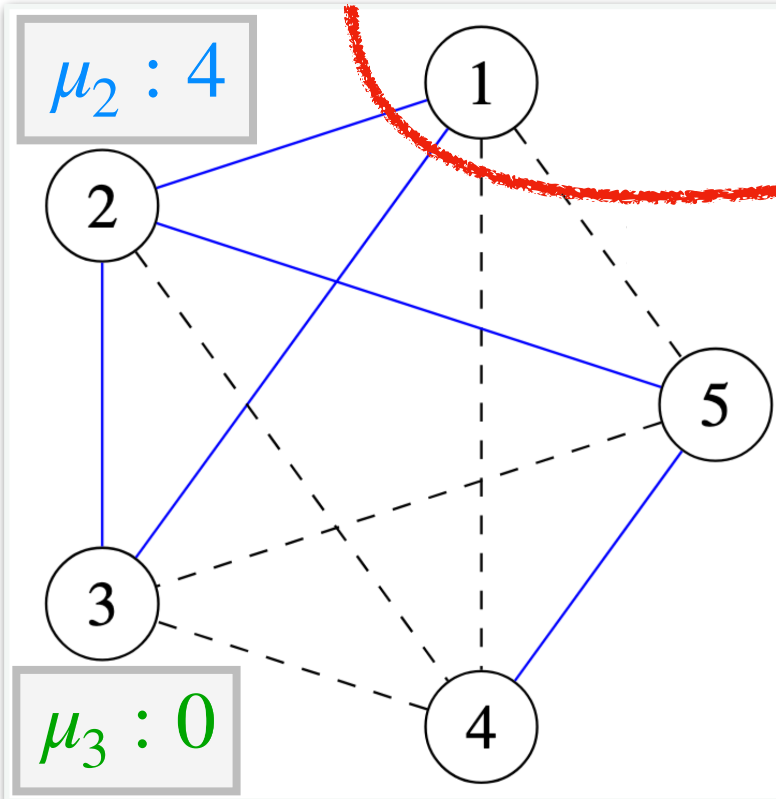
? The bound is not very tight, could we improve it ?



Relaxation cost: 50

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

$$c'_{u,v} = c_{u,v} + \mu_u + \mu_v$$
$$c'_{2,3} = c_{2,3} + \mu_2 + \mu_3$$
$$9 = 5 + 4 + 0$$



Relaxation cost: 59

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

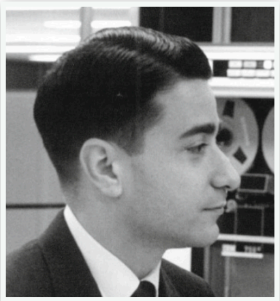
Lagrangian multiplier: value associated with each node

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

Nice property 1: the optimal TSP tour is invariant under this perturbation

Nice property 2: better bounds generally result in a much better filtering

Learning Lagrangian Multipliers for the TSP

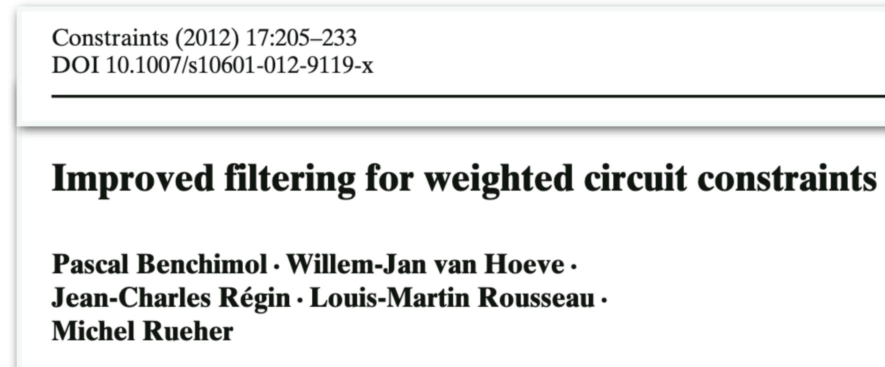


? But how can we find good values for the multipliers ?

Held and Karp, 1970: proposed an iterative algorithm for that

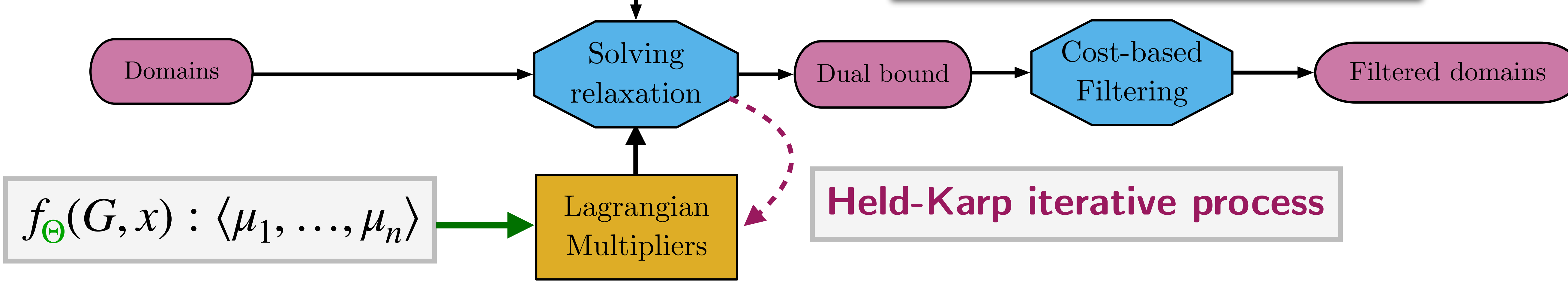
WeightedCircuit

1-tree relaxation


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

Improved filtering for weighted circuit constraints

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



Bad news: the iterative adjustment of the multipliers is computationally expensive

Parjadis et al. (CP 2024): use learning to initialize the multipliers

Idea: use a GNN to predict multipliers (one per node) from a TSP instance

Results: better filtering achieved on random and symmetric TSPs

Observation: replacing HK process with learning was not successful