# Learning Valid Dual Bounds in Constraint Programming: Boosted Lagrangian Decomposition with Self-Supervised Learning

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

*Swann Bessa, Darius Dabert, Max Bourgeat, Louis-Martin Rousseau, Quentin Cappart*

Lab Article

## Lagrangian Decomposition

**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

$$\begin{array}{ll} \max & f(X_1, X_2, X_3) \\ \text{s.t.} & C_1(X_1, X_2, X_3) \\ & C_2(X_2, X_3) \\ & X_1, X_2, X_3 \in \mathbb{N} \end{array} \Rightarrow \begin{array}{ll} \max & f(X_1, X_2, X_3) \\ \text{s.t.} & C_1(X_1, X_2, X_3) \\ & C_2(Y_2, Y_3) \\ & Y_2 = X_2,\ Y_3 = X_3 \\ & X_1, X_2, X_3, Y_2, Y_3 \in \mathbb{N} \end{array}$$

$$f(X_1, X_2, X_3) + \mu_2.(Y_2 - X_2) + \mu_3.(Y_3 - X_3)$$

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

Solving this relaxed problem will give a **dual bound**

**Consequence :** each constraint can be solved independently

$$\mathcal{B}(\mu_2, \mu_3) = \begin{array}{ll} \max & (f(X_1, X_2, X_3) + \mu_2.X_2 + \mu_3.X_3 \\ \text{s.t.} & C_1(X_1, X_2, X_3) \\ & X_1, X_2, X_3 \in \mathbb{N} \end{array} + \begin{array}{ll} \max & (-\mu_2.Y_2 - \mu_3.Y_3) \\ \text{s.t.} & C_2(Y_2, Y_3) \\ & Y_2, Y_3 \in \mathbb{N} \end{array}$$

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

**How to set the values of the multipliers ?**

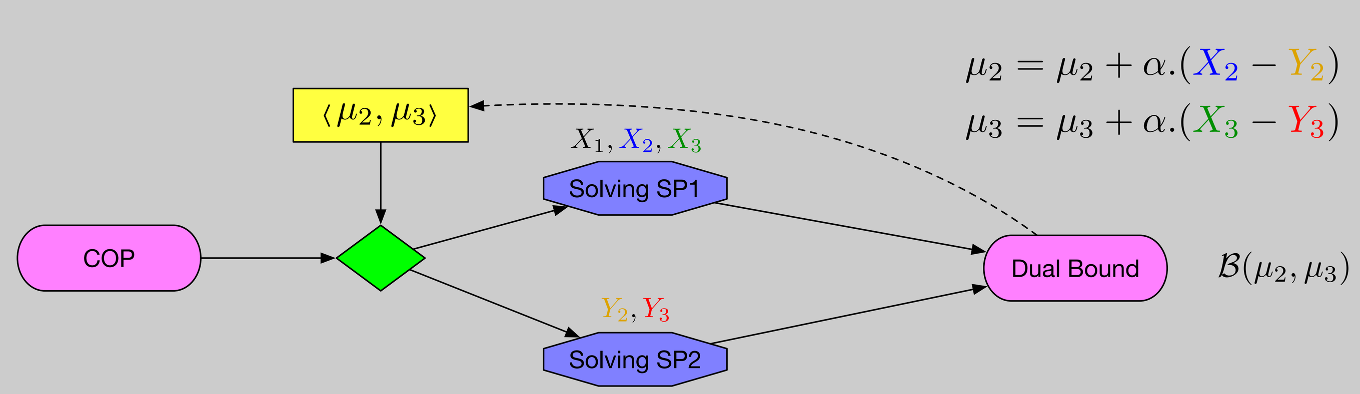
## Lagrangian Decomposition in CP

**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



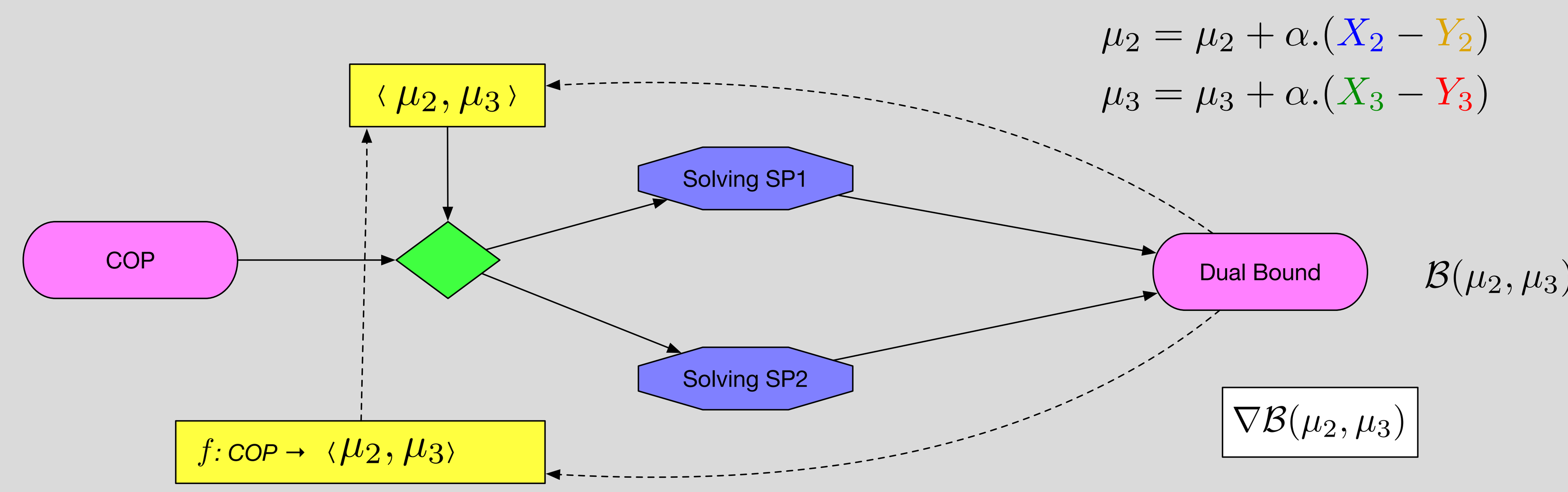

**This process is very costly as it requires solving few subproblems at each iteration**

## Our approach

**We propose a self-supervised learning approach to compute them**

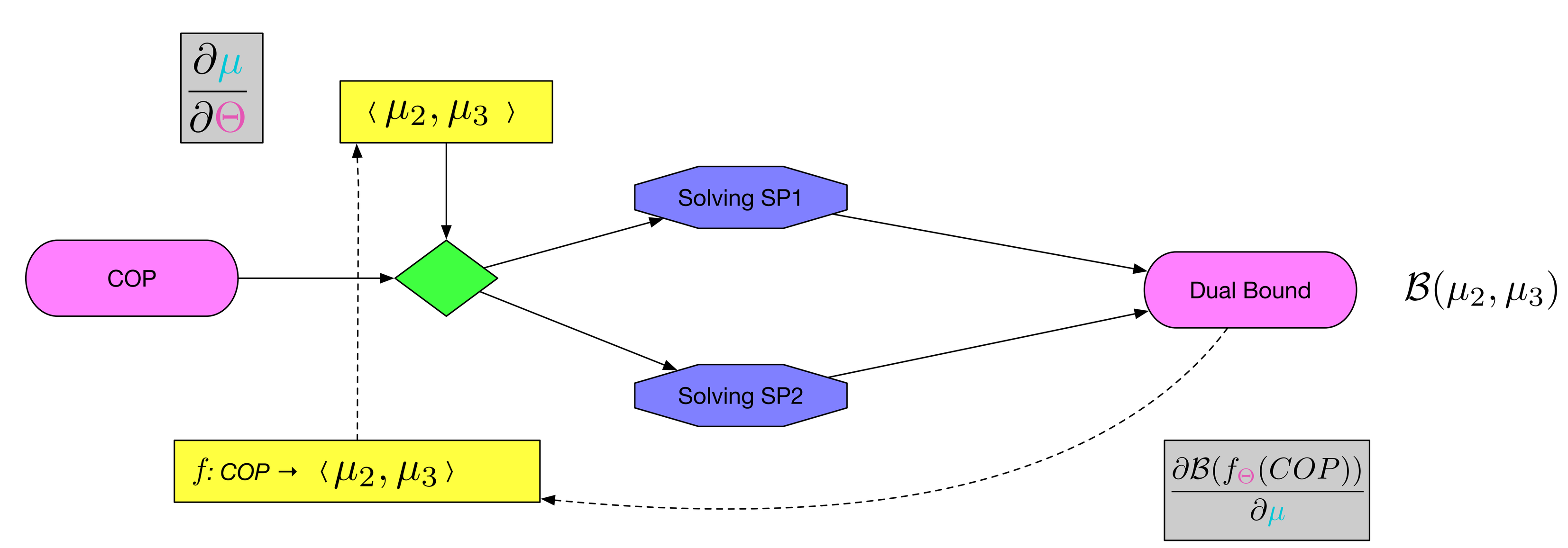
**Step 1**: multipliers are now obtained by a differentiable predictive model

**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

**How to compute the gradient of this bound ?**




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

**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)

## Take-home message

- **Lagrangian decomposition** → automatic bounding (CP)
- **Limitation** → high cost (sub-gradient optimization)
- **Novel approach** → learns Lagrangian multipliers → tight dual bounds
- **Self-supervised learning** → GNN models structure → no labeled bounds
- **First generic method** → learning valid dual bounds (CP)

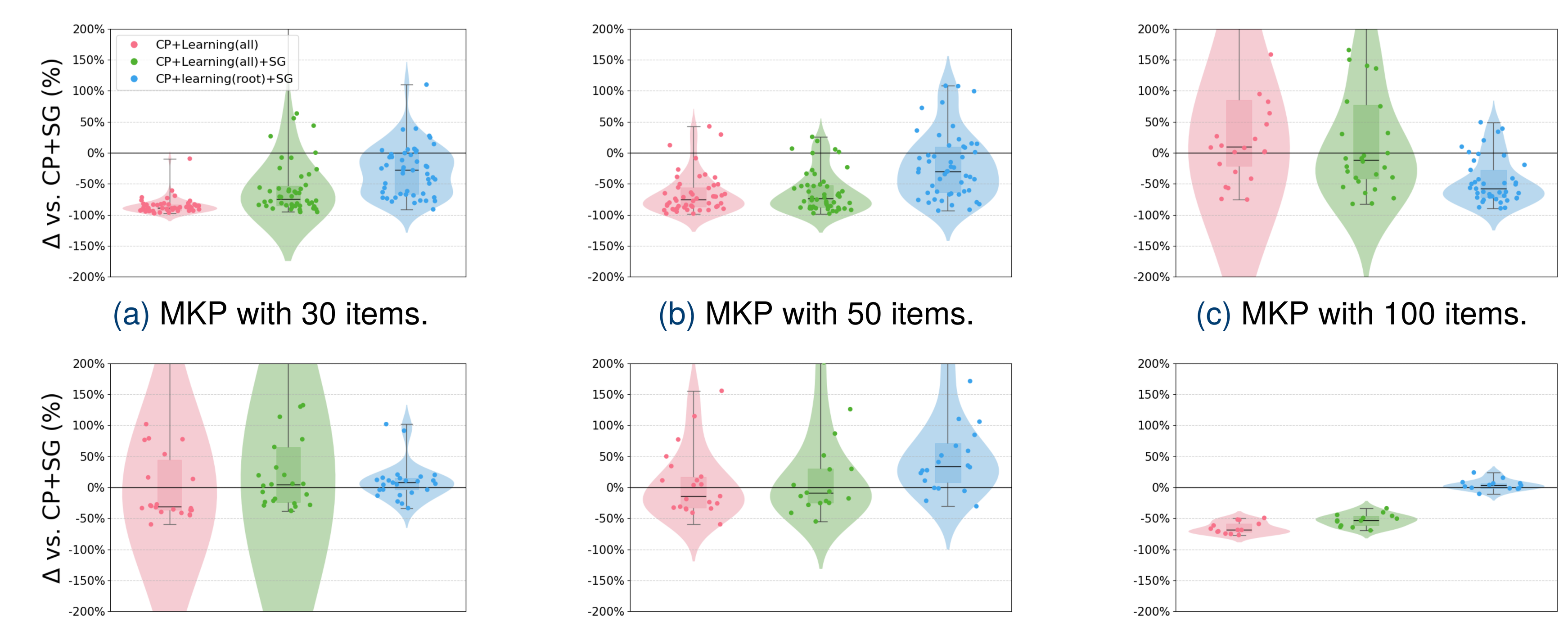
## Experiences

**Case studies:** Multi-dimensional Knapsack Problem & Shift Scheduling Problem

**Competitors:**

- CP : pure constraint programming approach,
- CP + SG : CP + LD and multipliers updated iteratively,
- CP + Learning(*All*) : CP + LD with bound learned and applied to every nodes
- CP + Learning(*All*) + SG : same but boud is further improved with SG,
- CP + Learning(*root*) + SG : bound learned applied only at the root node and used for bootstrapping sub gradients for other nodes.

## Results




(a) MKP with 30 items. (b) MKP with 50 items. (c) MKP with 100 items.

(d) SSP with 10 symbols and 20 states. (e) SSP with 20 symbols and 20 states. (f) SSP with 10 symbols and 80 states.

Each dot below 0% indicates a reduction in execution time with our method.

**Learning has significantly improved the application of LD in CP**

## References

[1] Monique Guignard and Siwhan Kim.
Lagrangean decomposition for integer programming : theory and applications.
*RAIRO. Recherche opérationnelle, tome 21*, 1987.

[2] Minh Hoàng Hà, Claude-Guy Quimper, and Louis-Martin Rousseau.
General bounding mechanism for constraint programs.
In *Principles and Practice of Constraint Programming: 21st International Conference, CP 2015, Cork, Ireland, August 31--September 4, 2015, Proceedings 21*, pages 158--172. Springer, 2015.

[3] Augustin Parjadis, Quentin Cappart, Bistra Dilkina, Aaron Ferber, and Louis-Martin Rousseau.
Learning Lagrangian Multipliers for the Travelling Salesman Problem.
In Paul Shaw, editor, *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, volume 307 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1--22:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl -- Leibniz-Zentrum für Informatik.