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

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

JOPT 2025

POLYTECHNIQUE MONTRÉAL

TECHNOLOGICAL UNIVERSITY

Constraint Programming (CP)

A constraint programming problem is :

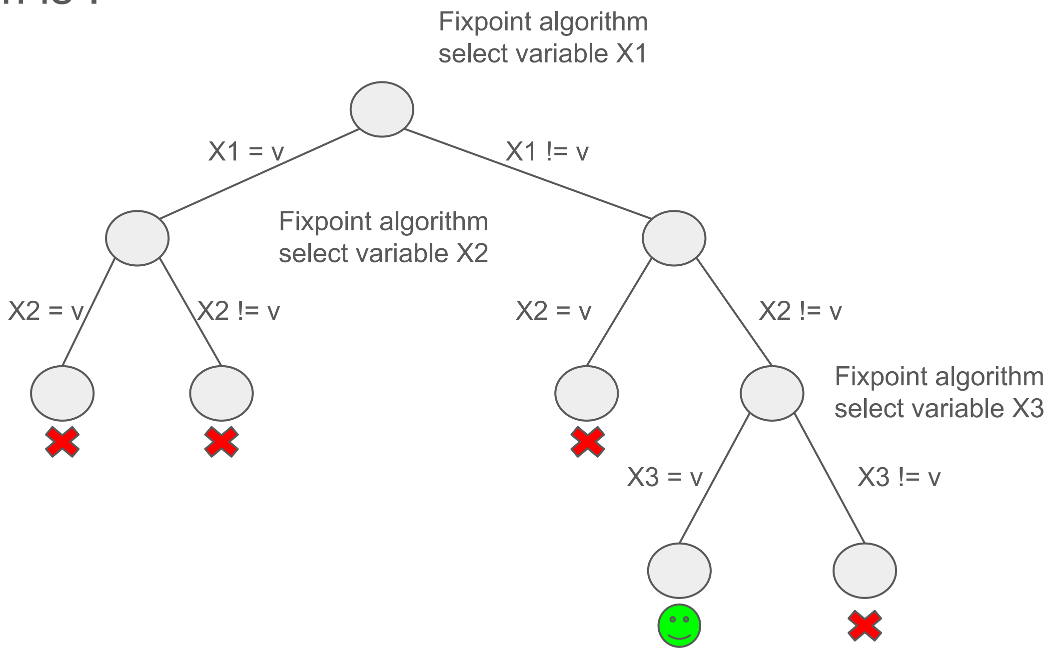
1) Variables
2) Variables' domains
3) Constraints

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

A constraint programming problem is :

1) Variables
2) Variables' domains
3) Constraints

$$
\begin{array}{ll}
\max & f(X_1, {\color{blue}X_2}, {\color{red}X_3}) \\
\text{s.t.} & C_1(X_1, {\color{blue}X_2}, {\color{red}X_3}) \\
& C_2({\color{blue}X_2}, {\color{red}X_3}) \\
& X_1, {\color{blue}X_2}, {\color{red}X_3} \in \mathbb{N}
\end{array}
$$



Fixpoint algorithm iteratively applies constraint propagation until no further domain reductions are possible

Enable solving combinatorial problems !

Lagrangian Decomposition (LD)

Lagrangian decomposition splits the problem into independent and easier subproblems

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

Guignard et al Lagrangean decomposition for interger programming : theory and applications

Lagrangian decomposition splits the problem into independent and easier subproblems

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

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

$\Rightarrow$

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

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

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

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

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

$\rightarrow$

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

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

Guignard et al Lagrangean decomposition for interger programming : theory and applications

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

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

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

Guignard et al Lagrangean decomposition for interger programming : theory and applications

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

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

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

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

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

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

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

Consequence : each constraint can be solved independently

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

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

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

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

Consequence : each constraint can be solved independently

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

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

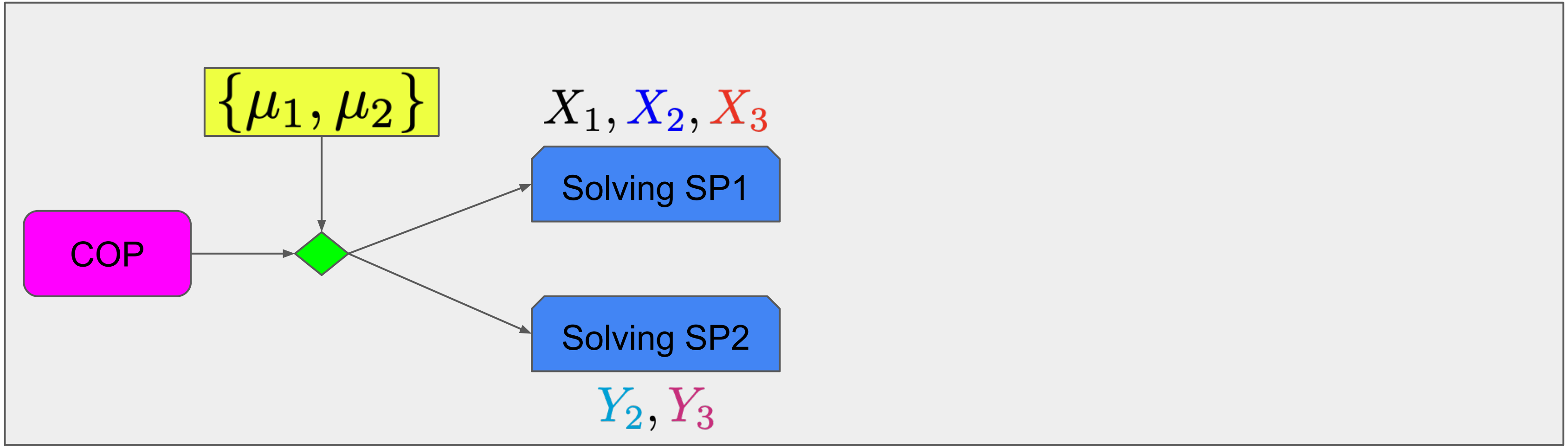
How to set the values of the multipliers ?

Lagrangian Decomposition in CP

Initialization : we set the multipliers to an arbitrary value

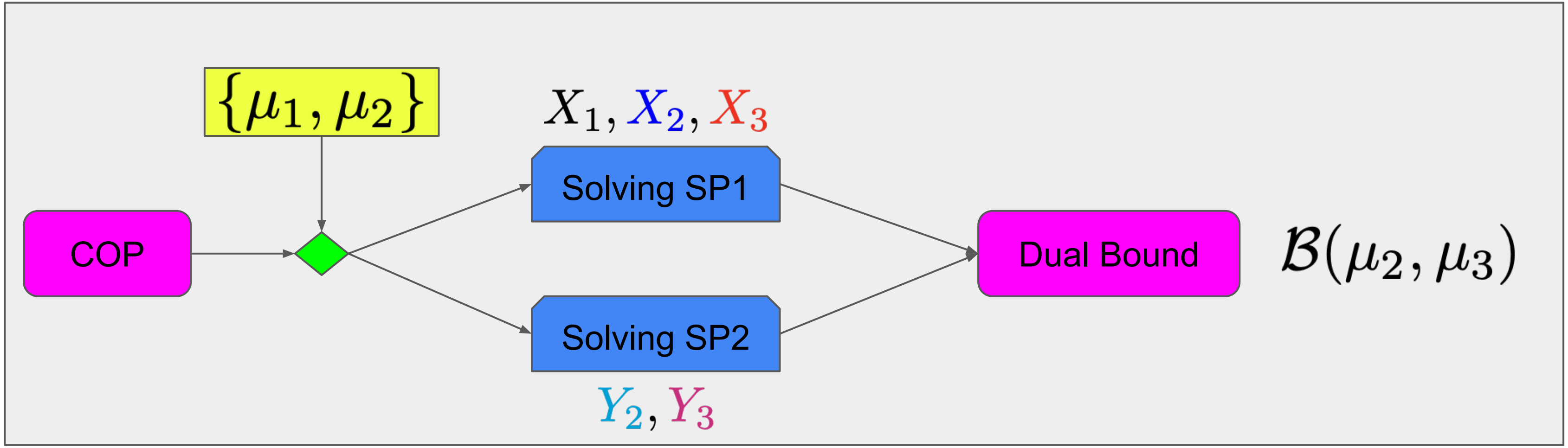
Initialization : we set the multipliers to an arbitrary value

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



Initialization : we set the multipliers to an arbitrary value

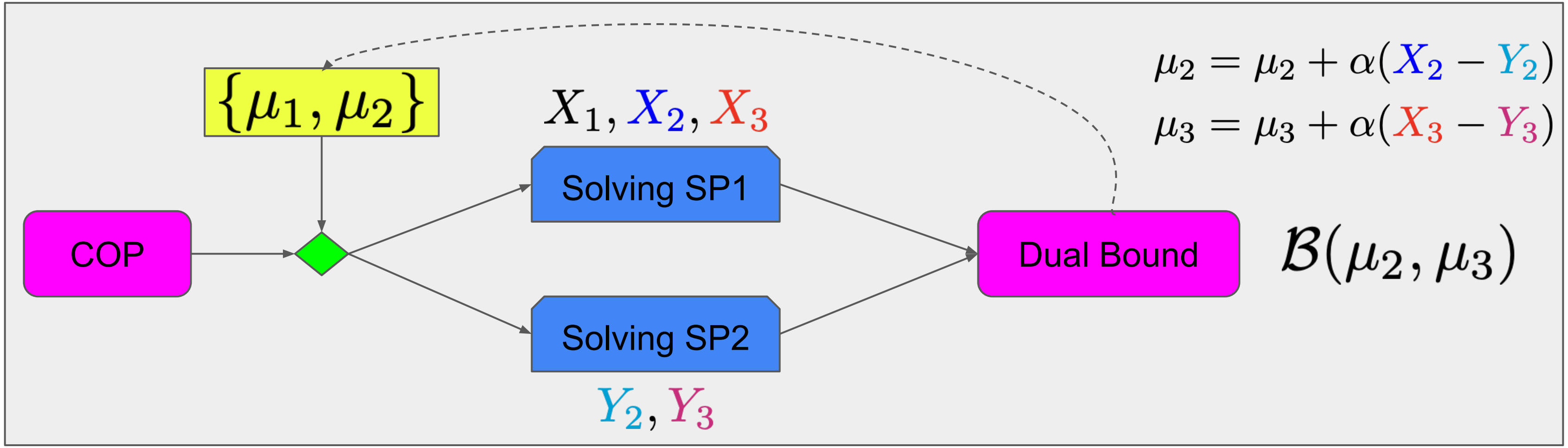
Step 1 : we solve all subproblems with these values and we get a dual bound



Initialization : we set the multipliers to an arbitrary value

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

Step 2 : we update the multipliers with sub-gradient

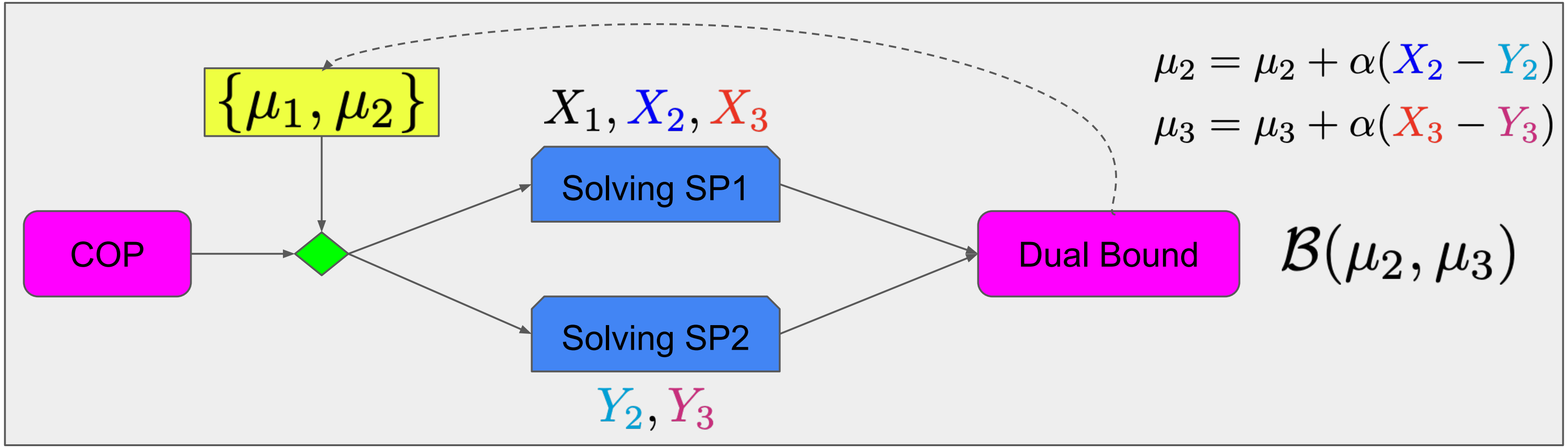


Initialization : we set the multipliers to an arbitrary value

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

Step 2 : we update the multipliers with sub-gradient

Step 3 : we repeat steps 1 and 2 for x iterations

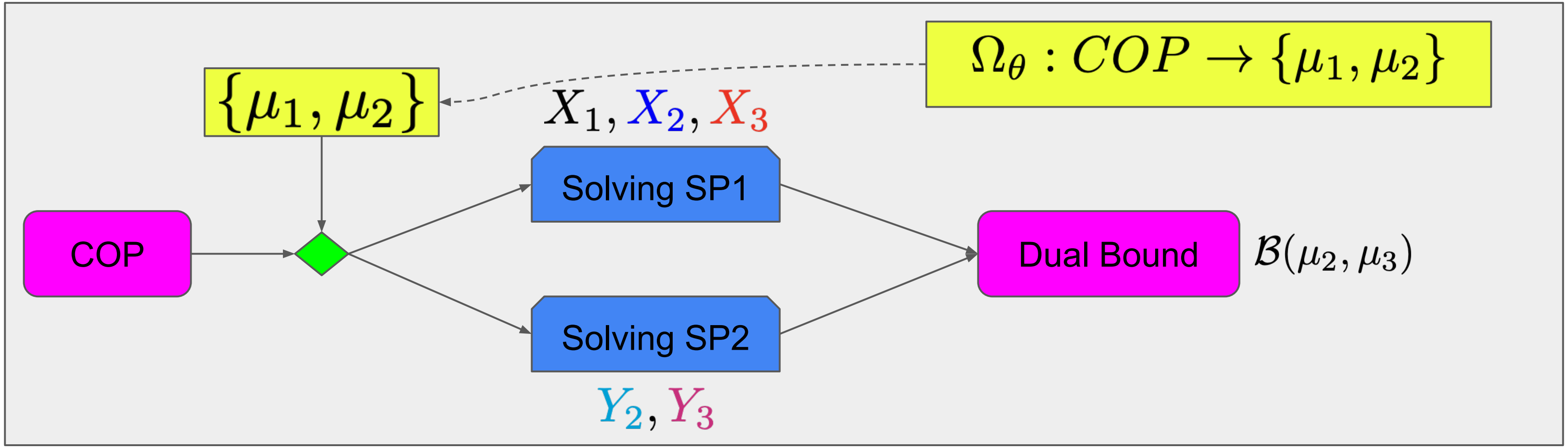


This process is very costly as it requires solving few subproblems at each iterations

Our Approach

We propose a **self-supervised** approach to compute the multipliers

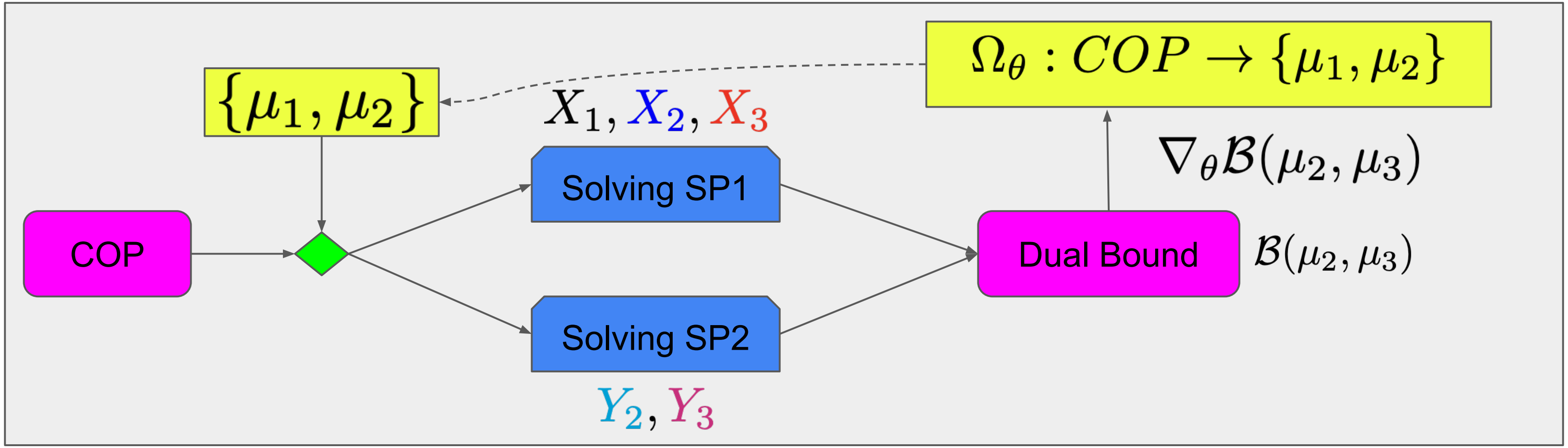
Step 1 : multipliers are now obtained by a differentiable predictive model



We propose a self-supervised approach to compute the multipliers

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

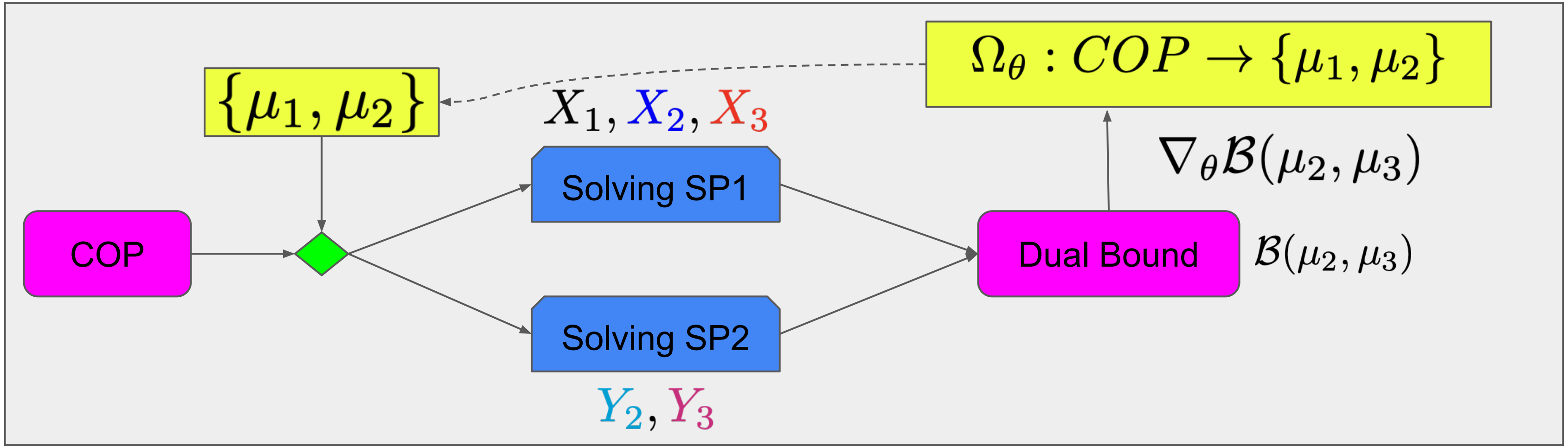
Step 2 : the model is trained end-to-end by differentiating the bound



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

How to compute the gradient of this bound ?

Ω_θ outputs $\{\mu_1, \mu_2\}$ but we want to update it regarding the quality of the bound and not the $\{\mu_1, \mu_2\}$



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

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)

Results

Experiences

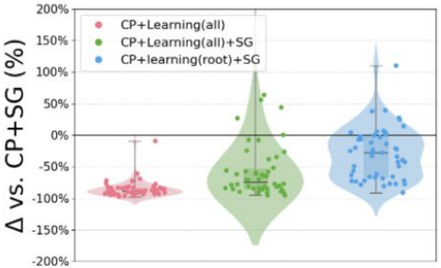
Case studies: Multi-dimensional Knapsack Problem & Shift Scheduling Problem

Competitors:

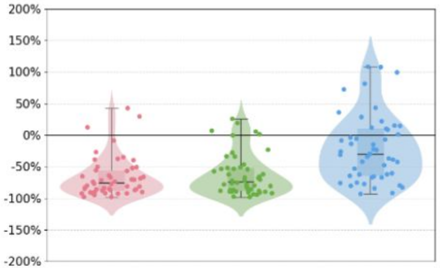
- CP : pure constraint programming approach,
- CP + SG : CP + LD and multipliers updated iteratively,
- CP + Learning(*All*) : CP + LD with bound learned and applied to every nodes
- CP + Learning(*All*) + SG : same but boud is further improved with SG,
- CP + Learning(*root*) + SG : bound learned applied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

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables

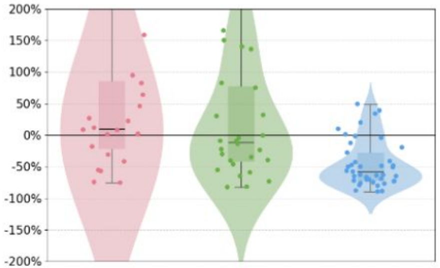
Results



(a) MKP with 30 items.



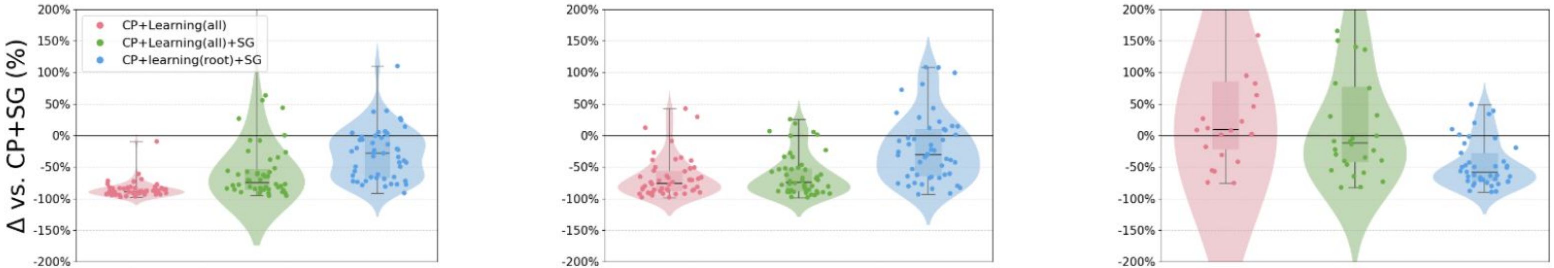
(b) MKP with 50 items.



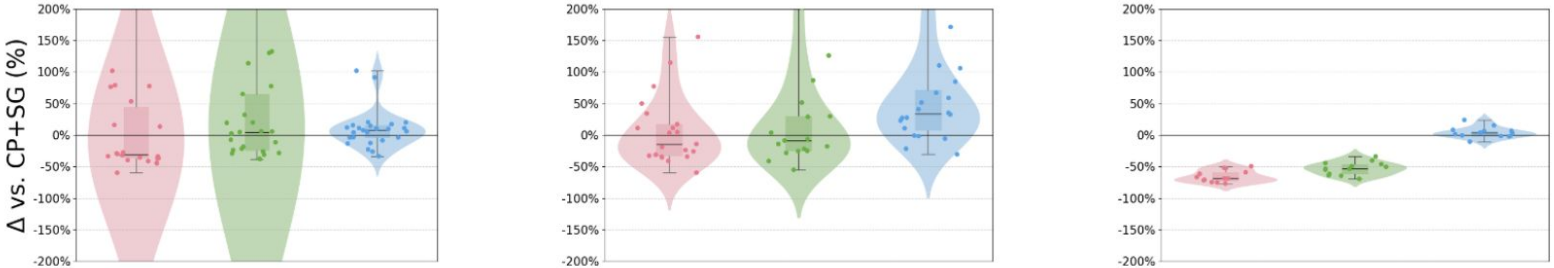
(c) MKP with 100 items.

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

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 time with our method

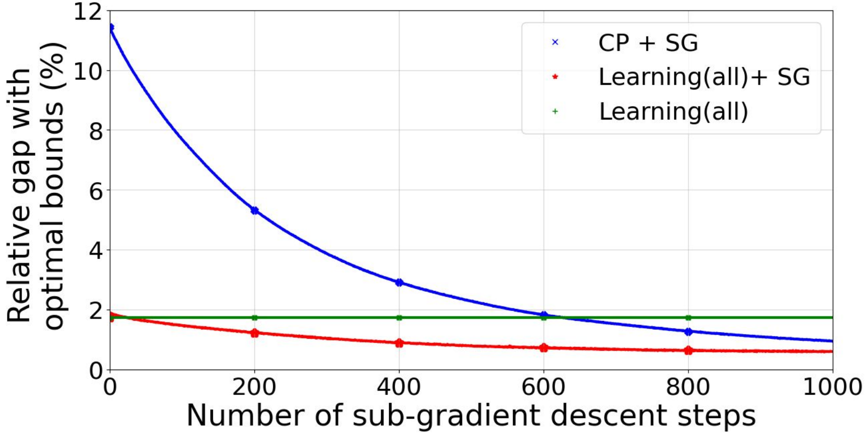
Learning has significantly improved the application of LD in CP

Results

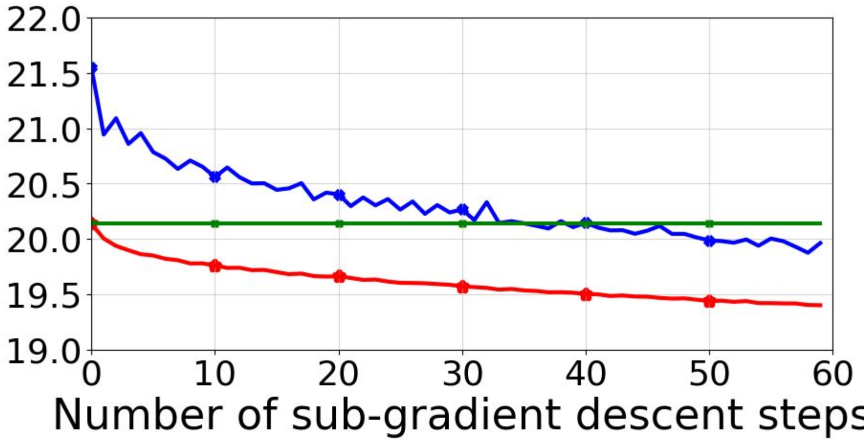
Multi-dimensional Knapsack Problem (MKP)												
Approaches	30 items				50 items				100 items			
	No. solved	Time	No. nodes	Gap	No. solved	Time	No. nodes	Gap	No. solved	Time	No. nodes	Gap
CP	50/50	**0.5**	22K	-	30/50	1,100	40M	-	0/50	-	-	-
CP+SG	50/50	19.2	116	2.0%	50/50	158	436	2.0%	41/50	2.4K	4.1K	1.3%
CP+Learning(all)	50/50	2.0	235	6.0%	50/50	**36**	2.6K	3.0%	25/50	2.8K	146K	2.0%
CP+Learning(all)+SG	50/50	6.7	170	2.0%	50/50	40	1.7K	2.0%	29/50	1.6K	33K	1.0%
CP+Learning($root$)+SG	50/50	10.6	**81**	2.0%	50/50	83	**340**	2.0%	**49/50**	**1.1K**	**2.1K**	1.0%

Shift Scheduling Problem (SSP)												
Approaches	10 symbols and 20 states				20 symbols and 20 states				10 symbols and 80 states			
	No. solved	Time	No. nodes	Gap	No. solved	Time	No. nodes	Gap	No. solved	Time	No. nodes	Gap
CP+SG	**28/50**	473	2.0K	9.3%	**27/50**	750	**2.9K**	9.8%	13/50	3.3K	**2.5K**	15.8%
CP+Learning(all)	24/50	852	8.8K	10.1%	24/50	**660**	4.7K	10.3%	**20/50**	**1.3K**	3.3K	15.8%
CP+Learning(all)+SG	24/50	880	6.3K	8.2%	22/50	770	4.5K	8.4%	17/50	1.6K	3.1K	15.1%
CP+Learning($root$)+SG	**28/50**	**453**	**1.8K**	8.2%	22/50	1,100	4.5K	8.4%	13/50	3.4K	**2.5K**	15.1%

Results



MKP 100 items



SSP 10 vars. 80 states

Evolution of the bounds for the two most challenging configurations (gap with the optimal solution)

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)

AAAI 2025 paper

Graph encoding

each node = lagrangian multiplier
MKP : one node per pair (variable, constraint)
n items, d dimension = nxd nodes
edges if share variable or constraint
6 features:

- h_v^1: the index of the related variable.
- h_v^2: the index of the related constraint.
- h_v^3: the profit of the item.
- h_v^4: the weight of the item on the related dimension.
- h_v^5: the ratio of profit to weight.
- h_v^6: the ratio of weight to capacity.

SSP : one node per triplet (variable, constraint, value)
n variables, m constraints, d values = nxmxd nodes
edge if same (variable, value) pair or same constraint (s.t. valid transition)
4 features:

- h_v^1: the index of the related variable.
- h_v^2: the index of the related constraint.
- h_v^3: the index of the related value.
- h_v^4: the profit associated with the triplet.

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.

Guignard et al Lagrangean decomposition for interger programming : theory and applications

Parjadis et al, Learning Lagrangian Multipliers for the Travelling Salesman Problem

Hà et al, General bounding mechanism for constraint programs