

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

Swann Bessa,^{1,2} Darius Dabert,^{1,2} Max Bourgeat,¹

Louis-Martin Rousseau,¹ Quentin Cappart¹

¹ Polytechnique Montréal, Montreal, Canada

² Ecole Polytechnique, Palaiseau, France

{swann.bessa, darius.dabert}@polytechnique.edu

{max.bourgeat, louis-martin.rousseau, quentin.cappart}@polymtl.ca

Abstract

Lagrangian decomposition (LD) is a relaxation method that provides a dual bound for constrained optimization problems by decomposing them into more manageable sub-problems. This bound can be used in branch-and-bound algorithms to prune the search space effectively. In brief, a vector of Lagrangian multipliers is associated with each sub-problem, and an iterative procedure (e.g., a sub-gradient optimization) adjusts these multipliers to find the tightest bound. Initially applied to integer programming, Lagrangian decomposition also had success in constraint programming due to its versatility and the fact that global constraints provide natural sub-problems. However, the non-linear and combinatorial nature of sub-problems in constraint programming makes it computationally intensive to optimize the Lagrangian multipliers with sub-gradient methods at each node of the tree search. This currently limits the practicality of LD as a general bounding mechanism for constraint programming. To address this challenge, we propose a self-supervised learning approach that leverages neural networks to generate multipliers directly, yielding tight bounds. This approach significantly reduces the number of sub-gradient optimization steps required, enhancing the pruning efficiency and reducing the execution time of constraint programming solvers. This contribution is one of the few that leverage learning to enhance bounding mechanisms on the dual side, a critical element in the design of combinatorial solvers. To our knowledge, this work presents the first generic method for learning valid dual bounds in constraint programming. We validate our approach on two challenging combinatorial problems: The multi-dimensional knapsack problem and the shift scheduling problem. The results show that our approach can solve more instances than the standard application of LD to constraint programming, reduce execution time by more than half, and has promising generalization ability through fine-tuning.

Introduction

Combinatorial optimization has applications across diverse fields, such as aerospace, transportation planning, scheduling, and economics. The objective is to identify an optimal solution from a finite set of possibilities. A significant challenge in combinatorial optimization is the state-space explosion problem: the number of potential solutions grows exponentially with the size of the problem, rendering large-scale problem-solving intractable. *Constraint programming* (CP) is a versatile tool for addressing combinatorial opti-

mization problems (Rossi, Van Beek, and Walsh 2006) as it is not confined to linear models, unlike mixed-integer programming (MIP). It can theoretically address any combinatorial problem, including those with non-linear constraints or objectives. Standard CP solvers rely on a search procedure that exhaustively yet efficiently enumerates all possible assignments of values to variables until the optimal solution is found.

The search is commonly conducted depth-first, combined with *branch-and-bound* techniques. The search backtracks to the previous decision point when an infeasible solution is encountered due to an empty domain. Through this procedure, and given that the entire search space is explored, the final solution found is proven to be optimal. Unfortunately, this tree search grows exponentially with the number of variables, making a complete enumeration of solutions still intractable for large problems. To address this challenge, CP search is first enhanced through a mechanism called *propagation*, which reduces the number of possible combinations and, consequently, the size of the tree search. Given the current domains of the variables and a constraint, propagation eliminates values from the domains that cannot be part of the final solution due to constraint violations. This process is repeated with each domain change and for each constraint until no more values can be removed from the domains.

Given its reliance on branch-and-bound, a first design choice in CP is the *branching* strategy, which dictates how the search space is explored. Well-designed heuristics are more likely to discover promising solutions, whereas poor heuristics may lead the search into unproductive regions of the solution space. The selection of an appropriate branching strategy is generally non-trivial, and its automatic design with learning methods has been extensively studied by the research community (Michel and Van Hentenryck 2012; Song et al. 2022; Marty et al. 2023). A second critical choice, though much less explored through the lens of learning, is the *bounding* strategy. Most existing CP solvers rely on a straightforward mechanism. When a feasible solution is found, a new constraint is added to ensure the following solution is better than the current one. This new constraint can enable more propagation. Unfortunately, the filtering provided by this approach is often relatively poor. Furthermore, this mechanism only allows the computation of a primal bound and does not offer dual bounds acting as a cer-

tificate of optimality. A significant shortcoming of current CP technologies is the lack of a general and efficient dual-bounding mechanism for constraint programs, unlike MIP solvers, which rely on the linear relaxation. This limitation is one of the main reasons why CP solvers are often less competitive than MIP solvers for many combinatorial problems. This paper proposes to tackle this limitation.

Related Work

Cost-Based Domain Filtering. To our knowledge, Focacci, Lodi, and Milano (1999) were the first to leverage bounding methods from mathematical programming to improve CP solvers generically. They propose to embed in constraints an optimization component consisting of a linear program representing a linear relaxation of the constraint itself. This is commonly referred to as *cost-based domain filtering* and directly enables the computation of a dual bound when solving the relaxed problem. However, a drawback of cost-based domain filtering is that the bound is local to each constraint for which a custom relaxation module needs to be developed. At the time of writing, the global constraints catalog reports more than 400 different existing global constraints (Beldiceanu et al. 2007), but only two of them have been considered in this work: ALLDIFFERENT and PATH. This naturally led researchers to implement this idea for other global constraints (Fahle and Sellmann 2002; Sellmann 2003; Houndji et al. 2017), or to leverage different types of relaxations. For instance, Benchimol et al. (2012) proposed using the Held and Karp relaxation to improve the filtering of the WEIGHTEDCIRCUIT constraint.

CP-based Lagrangian Relaxation. Cost-based domain filtering has a second drawback: examining only one constraint at a time risks significantly underestimating the actual objective cost. This occurs because the mechanism often assumes it is possible to improve the objective, which can be impossible due to other constraints. This approach acts locally and lacks a global view of the problem. One solution is to soften all constraints and incorporate them into the objective function with a penalty term, referred to as *Lagrangian multipliers*. This approach simplifies the problem. In CP, this procedure was introduced by Sellmann and Fahle (2001) and is commonly known as *CP-based Lagrangian relaxation*. Sellmann (2004) later proved that suboptimal Lagrangian multipliers can have stronger filtering abilities than optimal ones, which is advantageous as it allows for better filtering with less computational power. This idea has subsequently been used to solve various problems (Sellmann and Fahle 2003; Cambazard, O’Mahony, and O’Sullivan 2010; Isoart and Régim 2020; Boudreault and Quimper 2021), involving more and more constraints (Menana and Demasse 2009; Cambazard and Fages 2015; Berthiaume and Quimper 2024), and has shown promise in improving the performance of CP solvers (Fontaine, Michel, and Van Hentenryck 2014).

A drawback of CP-based Lagrangian relaxation is that the initial problem loses part of its original structure as the combinatorial constraints are removed from the constraint store and moved into the objective function. Additionally, the penalized constraints must be either linear or easily lineariz-

able, which is not often true for most problems.

Lagrangian Decomposition in CP. This limitation has progressively led researchers to consider *Lagrangian decomposition* (Guignard and Kim 1987) as an alternative bounding scheme, with the advantage of applying it to various constraints. Lagrangian decomposition breaks down the initial problem into a master problem and several independent sub-problems, each of which is more manageable to solve. A vector of Lagrangian multipliers is associated with each sub-problem, and the master problem iteratively adjusts these multipliers to identify the optimal penalties, thereby guiding the solution process toward optimality. This is typically achieved through a sub-gradient procedure (Shor 2012), wherein each sub-problem must be solved exactly once per sub-gradient iteration. Cronholm and Ajili (2004) were the first to apply this concept within the context of constraint programming, proposing a specific algorithm for the multicast network design problem. A decade later, Hà, Quimper, and Rousseau (2015) advanced this idea to develop an automatic bounding mechanism for constraint programming. Concurrently, Bergman, Cire, and van Hoes (2015) suggested utilizing Lagrangian decomposition for constraints that can be represented by decision diagrams (Andersen et al. 2007). Both studies underscored the potential of Lagrangian decomposition as an automatic dual-bounding mechanism for constraint programming.

However, two significant challenges arise with Lagrangian decomposition. First, a specialized and efficient algorithm is required to solve the sub-problems. Second, it incurs substantial computational overhead as each sub-problem must be solved at every sub-gradient iteration. While the former challenge has been addressed by Chu, Gange, and Stuckey (2016), who proposed solving the sub-problems via a search procedure rather than a specialized propagator, the latter remains unresolved. To our knowledge, the efficient utilization of Lagrangian decomposition for CP has not been further investigated and remains an open research question for the practical application of this approach.

Learning Valid Dual Bounds. In another context, machine learning has been considered to enhance the solving process of branch-and-bound solvers, both for integer programming (Khalil et al. 2016) and constraint programming (Cappart et al. 2021). We refer to the surveys of Bengio, Lodi, and Prouvost (2021) and of Cappart et al. (2023) for an extended literature review. From these surveys, one can observe that most attempts to use learning inside branch-and-bound operate on the branching decisions (i.e., directing the search) or on the primal side (i.e., acting as a heuristic). However, learning to improve the quality of relaxations using better dual bounds has been much less considered. This has only been addressed for the restricted use case of solvers based on decision diagrams (Cappart et al. 2022), for combinatorial optimization over factor graphs (Deng et al. 2022), integer linear programs (Abbas and Swoboda 2024), and the specific WEIGHTEDCIRCUIT constraint (Parjadis et al. 2023). To our knowledge, learning has never been used for improving CP solvers through Lagrangian decomposition.

Statement of Contributions. Building on this context, this paper contributes a novel learning approach dedicated to generating tight and valid dual bounds in CP. We propose to achieve this by learning appropriate multipliers for a Lagrangian decomposition, significantly reducing the number of sub-gradient iterations required. The learning process is self-supervised, utilizing a graph neural network to represent the problem structure. Unlike most related work, which focuses on learning primal heuristics, our approach develops a learning-based strategy to derive valid and tight dual bounds. To our knowledge, this is the first approach to offer a generic method for learning valid dual bounds for any type of combinatorial problem. Experiments are conducted on two challenging case studies: the multi-dimensional knapsack problem and the shift-scheduling problem.

Lagrangian Decomposition in CP

Formally, a *constrained optimization problem* (COP) is defined as a tuple $\langle X, D(X), C, f \rangle$, where X is a set of discrete variables, $D(X)$ is the set of domains for these variables, C is the set of constraints that restrict assignments of values to the variables, and $f(\cdot) : X \rightarrow \mathbb{R}$ is an objective function. A feasible solution is an assignment of values from $D(X)$ to X that satisfies all the constraints in C . An optimal solution is feasible and also maximizes the objective function. The canonical expression of a COP involving m constraints is formalized below.

$$\max_{X \in D(X)} \left\{ f(X) \mid \bigwedge_{i=1}^m C_i(X) \right\} \quad (1)$$

The notation $C_i(X)$ indicates that constraint C_i has the variables X in its scope. Lagrangian decomposition (LD) involves relaxing this problem to get a valid, ideally tight, upper bound on the solution. To do so, variables involved in each constraint are duplicated, except for the first constraint.

$$\max \left\{ f(X_1) \mid \left(\bigwedge_{i=1}^m C_i(X_i) \right) \wedge \left(\bigwedge_{i=2}^m X_i = X_1 \right) \right\} \quad (2)$$

Here, X_1 refers to the initial variables, and X_2 to X_m refers to newly introduced variables. Next, a penalty term is introduced for each duplicated variable and is added to the objective function with a cost $\mu_i \in \mathbb{R}^{|X_i|} \geq 0$.

$$\max \left\{ f(X_1) + \sum_{i=2}^m \mu_i \cdot (X_1 - X_i) \mid \left(\bigwedge_{i=1}^m C_i(X_i) \right) \wedge \left(\bigwedge_{i=2}^m X_i = X_1 \right) \right\} \quad (3)$$

These costs are the *Lagrangian multipliers*. We note that the resulting problem retains the same optimal solutions as the initial problem in Equation (1). We now *relax* the problem by removing all constraints linking the variables.

$$\mathcal{B}(\mu) = \max \left\{ f(X_1) + \sum_{i=2}^m \mu_i \cdot (X_1 - X_i) \mid \bigwedge_{i=1}^m C_i(X_i) \right\} \quad (4)$$

The resulting COP, formalized below, is a relaxation of Equation (1) and solving it will provide a dual bound $\mathcal{B}(\mu)$. This corresponds to an upper bound as we are maximizing. Interestingly, each constraint C_i has a different set of variables and could then be solved independently. We can consequently reorganize the expression as follows.

$$\mathcal{B}(\mu) = \Phi(X_1) + \sum_{i=2}^m \Psi_i(X_i) \quad (5)$$

$$\Phi(X_1) = \max \left\{ f(X_1) + \left(\sum_{i=2}^m \mu_i \right) \cdot X_1 \mid C_1(X_1) \right\}$$

$$\Psi_i(X_i) = \max \left\{ -\mu_i \cdot X_i \mid C_i(X_i) \right\} \quad \forall i \in \{2, \dots, m\}$$

A valid dual bound can be obtained from a given set of multipliers by solving m independent sub-problems, which are generally easier to solve than the original problem as they involve only a single constraint. However, the quality of the bound (i.e., how close it is to the optimal solution) heavily depends on the values of the multipliers. Finding the optimal multipliers is the main challenge in LD as Equation (5) is not differentiable. The most common way is to proceed with a subgradient optimization (Shor 2012). Starting from an arbitrary value, multipliers are iteratively updated.

$$\mu_i^{t+1} = \mu_i^t + \alpha(X_1^t - X_i^t) \quad \forall i \in \{2, \dots, m\} \quad (6)$$

Here, μ_i^t refers to the set of multipliers i at iteration t , X_i^t refers to the optimal values of variables X_i at iteration t , and α is a learning rate. The process is illustrated in Figure 1 (black and red elements).

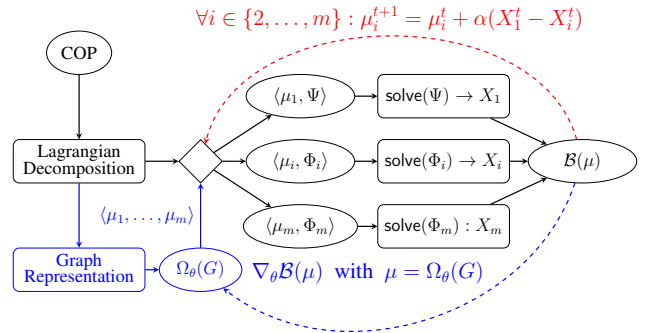


Figure 1: Main steps of LD in CP. Red elements illustrate the standard sub-gradient optimization, while the blue elements highlight our contribution based on self-supervised learning.

Hà, Quimper, and Rousseau (2015) proposed integrating this procedure into the search phase of a CP solver, involving LD and sub-gradient optimization at each node of the tree search. However, this approach is highly time-consuming, as each sub-gradient iteration requires solving each sub-problem, which can be NP-hard. We propose to significantly accelerate this process through the use of learning.

Learning Lagrangian Multipliers

The bound computed in Equation (5) has two notable properties: (1) it can be parameterized using the Lagrangian mul-

multipliers μ , and (2) it is always valid, meaning it will never underestimate the actual profit. Both properties present an opportunity to employ a learning-based approach to compute the bound. Inspired by Parjadis et al. (2023), we propose building a model $\Omega_\theta : G(V, E) \rightarrow \mathbb{R}^{|V|}$ capable of directly predicting all the μ multipliers for a COP instance given as input and represented by a graph G . The graph comprises a set of V nodes (one per multiplier) with their relationships expressed by edges in E . The model Ω_θ is parameterized by a set of p parameters $\theta = \{\theta_1, \dots, \theta_p\}$ and must be differentiable. The goal is to eliminate sub-gradient iterations by directly learning the multipliers that produce a tight bound, thereby saving execution time and enhancing filtering. The requirement for the model to be differentiable is essential for our methodology, which involves computing the bound using gradient-based optimization.

The goal is to find model parameters θ that yield the lowest possible bound. Thanks to the second property, the obtained bound is provably valid, regardless of the model's accuracy. To our knowledge, this paper presents the first generic method for learning valid dual bounds for any type of discrete COP. This is a notable strength, considering the difficulties in getting guarantees with learning in combinatorial optimization (Kotary et al. 2021). Given the multipliers $\mu = \langle \mu_2, \dots, \mu_m \rangle$, the bound minimization problem for an instance G and its related gradient are as follows.

$$\min_{\theta} \mathcal{B}(\mu) \mapsto \nabla_{\theta} \mathcal{B}(\mu) \text{ with } \mu = \Omega_{\theta}(G) \quad (7)$$

Figure 1 illustrates how our procedure (in blue) replaces the standard sub-gradient optimization (in red). However, computing the gradient of this expression is not trivial, as the bound is obtained through the solutions of combinatorial sub-problems. The chain rule can be applied since the multipliers are parameterized by θ .

$$\nabla_{\theta} \mathcal{B}(\mu) = \frac{\partial \mathcal{B}(\mu)}{\partial \mu} \cdot \frac{\partial \mu}{\partial \theta} \quad (8)$$

The right term, corresponding to the differentiation of the model Ω_θ (i.e., a graph neural network), is easily obtained via backpropagation. However, the left term requires differentiating the expression in Equation (5) for all μ .

$$\frac{\partial \mathcal{B}(\mu)}{\partial \mu} = \frac{\partial \Phi(X_1)}{\partial \mu} + \sum_{i=2}^m \frac{\partial \Psi_i(X_i)}{\partial \mu} \quad (9)$$

The analytic partial derivative cannot be obtained as it requires solving combinatorial sub-problems. Instead, we propose using a locally valid derivative by considering the optimal solution of each sub-problem, allowing us to set all variables X_i for all $i \in \{1, \dots, m\}$. Considering a specific multiplier vector μ_i , we expand the functions $\Phi(\cdot)$ and $\Psi(\cdot)$, which involve only additions of μ_i and constant multiplications. The partial derivatives of μ_i are then as follows.

$$\frac{\partial \mathcal{B}(\mu)}{\partial \mu_i} = (X_1 - X_i) \quad \forall i \in \{2, \dots, m\} \quad (10)$$

This yields the gradient in Equation (5), used to minimize a dual bound obtained through multipliers.

$$\nabla_{\theta} \mathcal{B}(\mu) = \left\langle (X_1 - X_i) \cdot \frac{\partial \mu_i}{\partial \theta}, \dots \right\rangle \quad \forall i \in \{2, \dots, m\} \quad (11)$$

We propose leveraging a self-supervised learning approach to parameterize the model Ω_θ . The training procedure is formalized in Algorithm 1. It takes as input a dataset \mathcal{D} consisting of a set of graphs $G(V, E)$ serving as instances of a specific COP. These instances can be obtained from historical data or synthetic generation. The features of the graph are problem-dependent. At each iteration, LD is carried out, and all the sub-problems are solved through a dedicated procedure to obtain the optimal values of X_i for current multipliers μ . The bound and its gradient are then computed, and a gradient descent step is executed. This will change the values of the multipliers for the next iteration and, consequently, the optimal values of variables X_i . Finally the parameters θ of the trained neural network Ω_θ are returned.

Algorithm 1: Training phase of Ω_θ from a dataset \mathcal{D} .

```

1: ▷ Pre:  $\mathcal{D}$  is the set of instances used for training.
2: ▷ Pre:  $\Omega_\theta$  is the differentiable model to train.
3: ▷ Pre:  $\theta$  are randomly initialized parameters.
4: ▷ Pre:  $K$  is the number of training epochs.

5: for  $k$  from 1 to  $K$  do
6:    $\langle G, m \rangle := \text{sampleFromTrainingSet}(\mathcal{D})$ 
7:    $\langle \Phi, \Psi_2, \dots, \Psi_m \rangle := \text{LD}(G)$ 
8:    $X_1 := \text{solve}(\Phi)$ 
9:   for  $i$  from 2 to  $m$  do
10:     $X_i := \text{solve}(\Psi_i)$ 
11:   end for
12:    $\mu := \Omega_\theta(G)$ 
13:    $\mathcal{B}(\mu) := \Phi(X_1) + \sum_{i=2}^m \Psi(X_i)$ 
14:    $\theta := \theta - \nabla_{\theta} \mathcal{B}(\mu)$ 
15: end for
16: return  $\theta$ 
```

Graph Neural Network Architecture

Graph neural networks (GNNs) (Scarselli et al. 2008) are a specific neural architecture designed to compute a vector representation, also referred to as an *embedding*, for each node of an input graph. This architecture has been widely used in related works on learning for combinatorial optimization (Cappart et al. 2023). Learning is carried out by aggregating information from neighboring nodes multiple times. Each aggregation operation is referred to as a *layer* of the GNN and involves weights that must be learned.

Let $G(V, E)$ be a simple graph used as input of Ω_θ in Algorithm 1, let $h_i \in \mathbb{R}^p$ be a p -dimensional vector representation of the input features of a node $v \in V$, and let $k_{u,v} \in \mathbb{R}^q$ be a similar q -dimensional representation of the input features of an edge $(u, v) \in E$. Finally, let L be the number of layers of the GNN. Our implementation is based on RES-GATEDCONV architecture (Bresson and Laurent 2017). The inference process involves computing the next representation (h_v^{l+1}) from the previous one (h_v^l) for each node v . This process is formalized in Equation (12), where $\langle \theta_1^l, \dots, \theta_4^l \rangle$ are the weights in layer l , $\mathcal{N}(v)$ are the neighboring nodes of v , \odot is the element-wise product, $\eta_{v,u}$ is an edge gate

between nodes v and u , and σ is a sigmoid activation.

$$h_v^{l+1} = \text{RELU}\left(\theta_1^l h_v^l + \sum_{u \in \mathcal{N}(v)} (\eta_{v,u} \odot \theta_2^l) h_u^l\right) \quad (12)$$

$$\eta_{v,u} = \sigma\left(\theta_3^l h_v^l + \theta_4^l h_u^l\right) \quad (13)$$

The edge features $k_{u,v}$ are concatenated to each message. Applying the inference on the L layers results in an embedding h_v^L for each node $v \in V$. Similarly, we also compute a *graph embedding* $\Gamma \in \mathbb{R}^r$ by averaging information from each node, as shown in Equation (14).

$$\Gamma = \frac{1}{|V|} \sum_{v \in V} \sigma(h_v^L) \quad (14)$$

Considering both a graph and node embedding allows each node to combine a global and a local representation of the problem. Finally, both embeddings are concatenated ($\cdot \parallel \cdot$) and given as input of a fully-connected neural network (FCNN) to obtain the Lagrangian multipliers, as shown in Equation (15). We recall that there is one node per multiplier in our representation. The architecture has two messages passing steps with a size of 64 each and the fully-connected neural network has two layers with 256 and 128 neurons.

$$\mu_v = \text{FCNN}(h_v^L \parallel \Gamma) \quad \forall v \in \{1, \dots, V\} \quad (15)$$

Case studies

This paper addresses two challenging COPs: the multi-dimensional knapsack and the shift scheduling problem, chosen for direct comparison with H , Quimper, and Rousseau (2015). Further details are in the appendix.

Multi-dimensional Knapsack Problem (MKP). It is an extension of the classic knapsack problem. In the MKP, we are given a set of items, each with a specific profit and multiple weights (one per dimension). Additionally, there are multiple knapsack constraints, each representing a capacity limit for a different dimension. The objective is to select a subset of items that maximizes the total profit while ensuring that the total weight in each dimension does not exceed the respective capacity constraints. In our CP formalization, for an instance of dimension d , we have d specific knapsack constraints, which result in d sub-problems. We solve each sub-problem with a standard dynamic programming algorithm for one-dimensional knapsack, which runs in $\mathcal{O}(nW)$, with n the number of items and W the capacity. For training and evaluation, a synthetic dataset is generated following the protocol of Pisinger (Kellerer, Pferschy, and Pisinger 2004). Instances of 30, 50, and 100 items and up to 5 dimensions are considered. Profits follow a uniform distribution between 0 and 500, weights between 0 and 100, and the capacity is a percentage of the total weight per dimension.

Shift Scheduling Problem (SSP). The problem involves assigning employees to shifts in a way that meets work regulations while maximizing profit. We consider a challenging version where the work regulations are so complex that modeling them for a single employee requires numerous REGULAR constraints (Pesant 2004). The goal is to assign one

activity per shift such that the transitions between activities over the periods satisfy the constraints and maximize the total profit. We consider instances with two REGULAR constraints (yielding two sub-problems) and 50 periods, as they are already challenging to solve and cannot all be solved to optimality within a 2-hour timeout. The sub-problems are solved using the procedure introduced by Pesant (2004), which runs in $\mathcal{O}(nmQ)$, where n is the number of variables, m is the number of symbols, and Q is the number of states in the REGULAR automaton. A synthetic dataset is generated for training and evaluation, following the procedure of Pesant (2004). Instances with two constraints, 50 items, and three configurations of symbols and states (10/20, 10/80, 20/20) are considered, where x/y denotes a configuration with x symbols and y states. Profits follow a uniform distribution between 0 and 100, with 0.3 as the proportion of undefined transitions and 0.5 as the proportion of final states.

Experimental Protocol

This section outlines the protocol used to evaluate the efficiency and reliability of our approach, detailing the datasets and configurations used across various testing scenarios.

Competitors. The methods tested in the experiments are:

- **CP.** A pure constraint programming approach without Lagrangian decomposition or learning.
- **CP+SG.** The same CP model is enhanced with LD and sub-gradient optimization as proposed by H , Quimper, and Rousseau (2015). We re-implemented this approach, and our results for this baseline are much better than those reported in the initial paper, likely due to the improved branching heuristics and the choice of CP solver.
- **CP+Learning(*all*).** The CP model with LD, using the bounds learned in Alg. 1 instead of sub-gradient, with learning applied at every node of the search tree.
- **CP+Learning(*all*)+SG.** Similar to the previous approach, the learned bounds are further improved by sub-gradient optimization. The trained model Ω_θ is called at every search tree node to obtain the bounds.
- **CP+Learning(*root*)+SG.** Model Ω_θ is used only at the root node, with the resulting bound serving as the initial value for bootstrapping sub-gradient in the other nodes.

Implementation. The graph neural network has been implemented in *Python* using *Pytorch Geometric* (Fey and Lenssen 2019). The CP models are solved with *Gecode* (Schulte, Lagerkvist, and Tack 2006), a CP solver implemented in C++. The interface between both languages has been implemented using *libtorch*. The implementation and the datasets used are released on GitHub¹.

Training and Validation. The training phase follows Algorithm 1. For each configuration, a specific model is trained using synthetic instances. A training set of 900 instances is used for the MKP and 200 for the SSP, with data augmentation from partially solved instances up to a depth of 5. The self-supervised learning process eliminates the need

¹Hidden while the work is under review. Available on demand.

Multi-dimensional Knapsack Problem (MKP)									
Approaches	30 items			50 items			100 items		
	No. solved	Time (sec.)	No. nodes	No. solved	Time (sec.)	No. nodes	No. solved	Time (sec.)	No. nodes
CP	50/50	0.5	22,000	30/50	1,100	40,000,000	0/50	-	-
CP+SG	50/50	19.2	116	50/50	158	436	41/50	2.4K	4.1K
CP+Learning(<i>all</i>)	50/50	2.0	235	50/50	36	2,600	25/50	2.8K	146K
CP+Learning(<i>all</i>)+SG	50/50	6.7	170	50/50	40	1,700	29/50	1.6K	33K
CP+Learning(<i>root</i>)+SG	50/50	10.6	81	50/50	83	340	49/50	1.1K	2.1K

Shift Scheduling Problem (SSP)									
Approaches	10 symbols and 20 states			20 symbols and 20 states			10 symbols and 80 states		
	No. solved	Time (sec.)	No. nodes	No. solved	Time (sec.)	No. nodes	No. solved	Time (sec.)	No. nodes
CP+SG	28/50	473	2.0K	27/50	750	2.9K	13/50	3.3K	2.5K
CP+Learning(<i>all</i>)	24/50	852	8.8K	24/50	660	4.7K	20/50	1.3K	3.3K
CP+Learning(<i>all</i>)+SG	24/50	880	6.3K	22/50	770	4.5K	17/50	1.6K	3.1K
CP+Learning(<i>root</i>)+SG	28/50	453	1.8K	22/50	1,100	4.5K	13/50	3.4K	2.5K

Table 1: Comparison of our approaches with standard baselines for the MKP and SSP on 50 instances per configuration. Results include the number of instances solved within a 2-hour timeout, average execution time, and average nodes explored to close the search. Averages are calculated only on instances solved by each method (excluding CP for MKP with 100 items). The CP baseline is also omitted for SSP due to its significantly poorer performance.

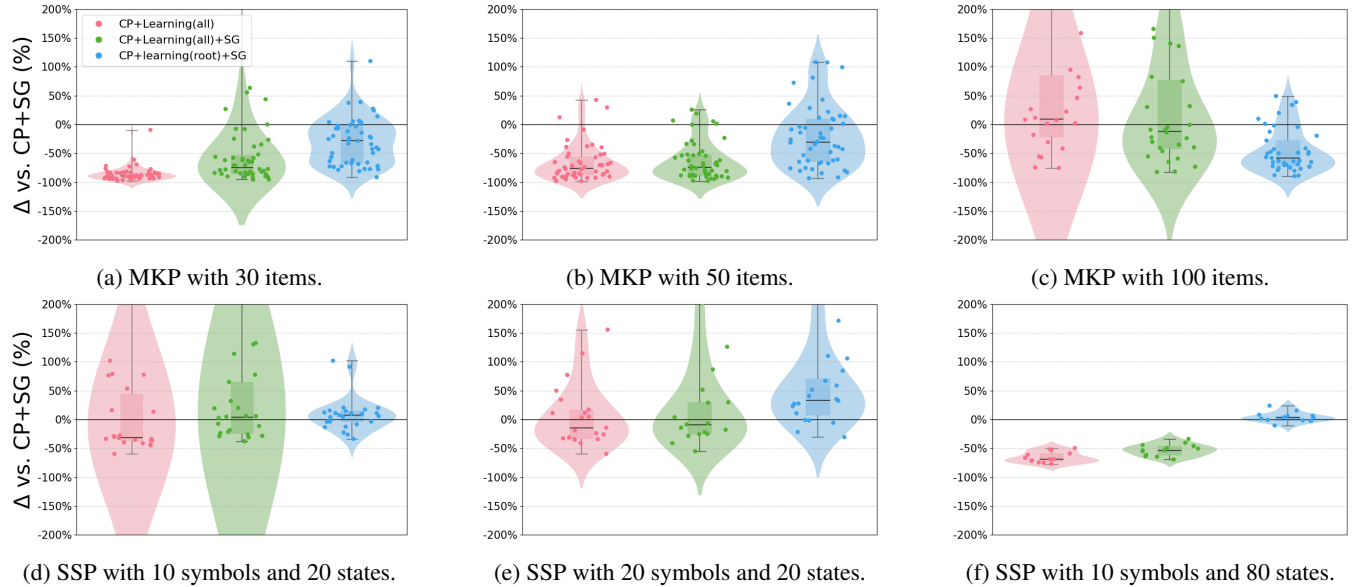


Figure 2: Relative performance of our approaches compared to the CP+SG baseline in execution time. Each dot below 0% indicates a reduction in execution time with our method for a specific instance.

for labeled multipliers. Training was conducted on a Tesla V100-SXM2-32GB GPU for 24 hours, with no extensive hyperparameter tuning (details in the attached files). Models were trained and tested in a single run, with convergence observed on a validation set of 100 MKP and 30 SSP instances. Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.001 has been used.

Empirical Results

This section presents the results of our experimental analysis, offering a detailed quantitative evaluation of our approach through a series of experiments.

Main Result: Quality of the Learned Bounds. Table 1 shows the performance of all approaches on a test set of 50 instances for both problems. We first observe that CP

without LD, although being the fastest in the easiest configuration (MKP with 30 items), has a very large number of explored nodes, which quickly explodes for larger instances. Interestingly, we observe that learning alone (CP+Learning(*all*)) consistently results in a larger search tree (i.e., more nodes) compared to sub-gradient optimization (CP+SG), indicating weaker bounds. However, this approach allows the solver to skip all sub-gradient iterations, often leading to significant execution time savings (e.g., from 158 to 36 seconds for MKP with 50 items, or from 3,300 to 1,300 seconds for SSP with 80 states). Unfortunately, these time savings are insufficient for the most challenging configurations, where the number of nodes is much higher. In such cases, using learning to initialize the sub-gradient procedure (CP+Learning(*root*)+SG) offers the best performance (e.g., reducing nodes explored from 4,100 to 2,100 for MKP with 100 items, decreasing average execution time from 2,400 to 1,100 seconds, and solving 8 additional instances). In conclusion, learning has significantly improved the application of LD in CP, either by replacing sub-gradient steps or by bootstrapping them. Performance on individual instances (Figure 2) supports this conclusion. The significance of each improvement in time was validated using a one-sided Wilcoxon signed-rank test. For each configuration, at least one of our approaches offers a significant improvement (p -value < 0.001), excepting SSP-20-20 (p -value < 0.1) and SSP-10-20 (no significant improvement).

Analysis: Fine-Tuning to OOD Instances. This experiment evaluates our approach’s ability to generalize to instances from an unseen distribution when fine-tuning is applied. We used the standard benchmark of Shih (1979), consisting of 30 MKP instances with 30 to 90 items. Fine-tuning was limited to 2 hours (10 epochs) with 100 instances from a similar distribution. Results in Table 2 show that we outperform the baseline for most instances, thanks to fine-tuning. Although CP+Learning(*all*) offers the best overall performance, it falls short in a few cases due to the intrinsic lack of guarantees in learning-based methods. Combining learning with sub-gradient (CP+Learning(*root*)+SG) mitigates this risk and provides more stable results. We also report that performance without fine-tuning is worse than the baseline.

Analysis: Evolution of the Bounds. This experiment analyzes the tightness of bounds obtained with different approaches and how they improve with each sub-gradient step. Figure 3 shows the average bounds on 50 instances for the two hardest configurations, measured by the gap with the optimal solution. In both cases, learning alone provides much better multipliers than random ones (LD before any sub-gradient step). After a number of iterations (600 for MKP and 40 for SSP), sub-gradient alone manages to improve the bounds but is still outperformed by our approach, which uses learning to initialize the sub-gradient procedure.

Conclusion

Previous works have identified Lagrangian decomposition as a promising approach for developing automatic bounding mechanisms in constraint programming. However, the practicality of LD is limited by the prohibitive execution time re-

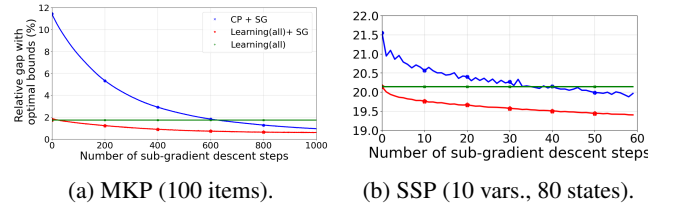


Figure 3: Evolution of the bounds for the two most challenging configurations (gap with the optimal solution).

quired for sub-gradient optimization at each node of the tree search. This paper addresses this challenge with an innovative approach that learns Lagrangian multipliers to produce tight dual bounds. The self-supervised learning process, which eliminates the need for labeled bounds, leverages a graph neural network to represent the problem structure. Experiments demonstrate the potential of this approach, either by replacing sub-gradient iterations entirely or by bootstrapping them. Fine-tuning has shown promise for generalizing to out-of-distribution instances. To our knowledge, this is the first contribution proposing a generic method for learning valid dual bounds in constraint programming.

Instances		CP+SG		CP+Learning(<i>all</i>)		CP+Learning(<i>root</i>)+SG	
Name	Vars	Nodes	Time	Nodes	Time	Nodes	Time (seconds)
Weish1	30	29	6	341	3	29	4
Weish2	30	41	7	159	1	41	6
Weish3	30	51	6	67	< 0	43	5
Weish4	30	29	4	35	< 0	29	3
Weish5	30	25	4	31	< 0	25	3
Weish6	30	51	14	649	6	51	12
Weish7	30	43	13	1197	10	43	11
Weish8	30	51	15	669	6	51	13
Weish9	30	43	11	61	1	41	9
Weish10	50	117	72	291	4	101	36
Weish11	50	69	30	73	1	45	13
Weish12	50	85	38	323	4	65	23
Weish13	50	139	56	379	5	107	33
Weish14	60	153	84	623	10	127	53
Weish15	60	111	42	581	6	111	37
Weish16	60	97	40	393	5	87	31
Weish17	60	337	223	9127	126	337	177
Weish18	70	427	210	70564	1307	429	186
Weish20	70	177	94	1159	15	171	75
Weish21	70	237	115	991	16	233	97
Weish22	80	167	174	1207	33	147	119
Weish23	80	373	376	2729	77	299	231
Weish24	80	501	309	t.o.	t.o.	511	281
Weish25	80	307	186	7444	140	319	170
Weish26	90	233	259	t.o.	t.o.	197	158
Weish27	90	269	216	t.o.	t.o.	261	175
Weish28	90	297	311	1465	49	253	193
Weish29	90	347	353	1599	56	303	205
Weish30	90	1015	1062	t.o.	t.o.	1085	997
Arithmetic mean		203	153	-	-	191	117
Geometric mean		131	60	-	-	120	44
No. best		5/30	0/30	0/30	25/30	17/30	5/30

Table 2: Fine-tuning on MKP instances (Shih 1979). *No. best* is how often each approach achieved the best results.

References

- Abbas, A.; and Swoboda, P. 2024. Doge-train: Discrete optimization on GPU with end-to-end training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20623–20631.
- Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming—CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23–27, 2007. Proceedings 13*, 118–132. Springer.
- Beldiceanu, N.; Carlsson, M.; Demassey, S.; and Petit, T. 2007. Global constraint catalogue: Past, present and future. *Constraints*, 12: 21–62.
- Benchimol, P.; Hoeve, W.-J. v.; Régim, J.-C.; Rousseau, L.-M.; and Rueher, M. 2012. Improved filtering for weighted circuit constraints. *Constraints*, 17: 205–233.
- Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: a methodological tour d’horizon. *European Journal of Operational Research*, 290(2): 405–421.
- Bergman, D.; Cire, A. A.; and van Hoeve, W.-J. 2015. Improved constraint propagation via lagrangian decomposition. In *International Conference on Principles and Practice of Constraint Programming*, 30–38. Springer.
- Berthiaume, F.; and Quimper, C.-G. 2024. Local Alterations of the Lagrange Multipliers for Enhancing the Filtering of the AtMostNValue Constraint. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 68–83. Springer.
- Boudreault, R.; and Quimper, C.-G. 2021. Improved CP-Based Lagrangian Relaxation Approach with an Application to the TSP. In *IJCAI*, 1374–1380.
- Bresson, X.; and Laurent, T. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*.
- Cambazard, H.; and Fages, J.-G. 2015. New filtering for atmostnvalue and its weighted variant: A lagrangian approach. *Constraints*, 20(3): 362–380.
- Cambazard, H.; O’Mahony, E.; and O’Sullivan, B. 2010. Hybrid methods for the multileaf collimator sequencing problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14–18, 2010. Proceedings 7*, 56–70. Springer.
- Cappart, Q.; Bergman, D.; Rousseau, L.-M.; Prémont-Schwarz, I.; and Parjadis, A. 2022. Improving variable orderings of approximate decision diagrams using reinforcement learning. *INFORMS Journal on Computing*, 34(5): 2552–2570.
- Cappart, Q.; Chételat, D.; Khalil, E. B.; Lodi, A.; Morris, C.; and Veličković, P. 2023. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130): 1–61.
- Cappart, Q.; Moisan, T.; Rousseau, L.-M.; Prémont-Schwarz, I.; and Cire, A. A. 2021. Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 3677–3687.
- Chu, G.; Gange, G.; and Stuckey, P. J. 2016. Lagrangian Decomposition via sub-problem Search. In *Integration of AI and OR Techniques in Constraint Programming: 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29–June 1, 2016, Proceedings 13*, 65–80. Springer.
- Cronholm, W.; and Ajili, F. 2004. Strong cost-based filtering for Lagrange decomposition applied to network design. In *International Conference on Principles and Practice of Constraint Programming*, 726–730. Springer.
- Deng, Y.; Kong, S.; Liu, C.; and An, B. 2022. Deep attentive belief propagation: Integrating reasoning and learning for solving constraint optimization problems. *Advances in Neural Information Processing Systems*, 35: 25436–25449.
- Fahle, T.; and Sellmann, M. 2002. Cost based filtering for the constrained knapsack problem. *Annals of Operations Research*, 115: 73–93.
- Fey, M.; and Lenssen, J. E. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*.
- Focacci, F.; Lodi, A.; and Milano, M. 1999. Cost-Based Domain Filtering. *Principles and Practice of Constraint Programming—CP’99: 5th International Conference, CP’99, Alexandria, VA, USA, October 11–14, 1999. Proceedings 5*.
- Fontaine, D.; Michel, L.; and Van Hentenryck, P. 2014. Constraint-based lagrangian relaxation. In *Principles and Practice of Constraint Programming: 20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings 20*, 324–339. Springer.
- Guignard, M.; and Kim, S. 1987. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical programming*, 39(2): 215–228.
- Hà, M. H.; Quimper, C.-G.; and Rousseau, L.-M. 2015. 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*, 158–172. Springer.
- Houndji, V. R.; Schaus, P.; Hounkonnou, M. N.; and Wolsey, L. 2017. The weighted arborescence constraint. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 185–201. Springer.
- Isoart, N.; and Régim, J.-C. 2020. Adaptive CP-based Lagrangian relaxation for TSP solving. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21–24, 2020, Proceedings 17*, 300–316. Springer.
- Kellerer, H.; Pferschy, U.; and Pisinger, D. 2004. *Knapsack Problems*. Springer.
- Kellerer, H.; Pferschy, U.; Pisinger, D.; Kellerer, H.; Pferschy, U.; and Pisinger, D. 2004. Multidimensional knapsack problems.

- Khalil, E.; Le Bodic, P.; Song, L.; Nemhauser, G.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.
- Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kotary, J.; Fioretto, F.; van Hentenryck, P.; and Wilder, B. 2021. End-to-End Constrained Optimization Learning: A Survey. In *30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, 4475–4482. International Joint Conferences on Artificial Intelligence.
- Marty, T.; François, T.; Tessier, P.; Gautier, L.; Rousseau, L.-M.; and Cappart, Q. 2023. Learning a Generic Value-Selection Heuristic Inside a Constraint Programming Solver. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- Menana, J.; and Demasse, S. 2009. Sequencing and counting with the multicost-regular constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27-31, 2009 Proceedings* 6, 178–192. Springer.
- Michel, L.; and Van Hentenryck, P. 2012. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 9th International Conference, CPAIOR 2012, Nantes, France, May 28-June 1, 2012. Proceedings* 9, 228–243. Springer.
- Parjadis, A.; Cappart, Q.; Dilkina, B.; Ferber, A.; and Rousseau, L.-M. 2023. Learning Lagrangian Multipliers for the Travelling Salesman Problem. *arXiv preprint arXiv:2312.14836*.
- Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In *International conference on principles and practice of constraint programming*, 482–495. Springer.
- Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.
- Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80.
- Schulte, C.; Lagerkvist, M.; and Tack, G. 2006. Gecode. *Software download and online material at the website: <http://www.gecode.org>*, 11–13.
- Sellmann, M. 2003. Cost-based filtering for shorter path constraints. In *International Conference on Principles and Practice of Constraint Programming*, 694–708. Springer.
- Sellmann, M. 2004. Theoretical foundations of CP-based lagrangian relaxation. In *International Conference on Principles and Practice of Constraint Programming*, 634–647. Springer.
- Sellmann, M.; and Fahle, T. 2001. CP-based lagrangian relaxation for a multimedia application. In *Third International Workshop on the Integration of AI and OR Techniques (CPAIOR 2001)*.
- Sellmann, M.; and Fahle, T. 2003. Constraint programming based lagrangian relaxation for the automatic recording problem. *Annals of Operations Research*, 118: 17–33.
- Shih, W. 1979. A branch and bound method for the multi-constraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30(4): 369–378.
- Shor, N. Z. 2012. *Minimization methods for non-differentiable functions*, volume 3. Springer Science & Business Media.
- Song, W.; Cao, Z.; Zhang, J.; Xu, C.; and Lim, A. 2022. Learning variable ordering heuristics for solving constraint satisfaction problems. *Engineering Applications of Artificial Intelligence*, 109: 104603.

Multi-Dimensional Knapsack Problem

The *Multi-Dimensional Knapsack Problem* (MKP) is an extension of the classic knapsack problem (Kellerer et al. 2004). In the MDPK, there are multiple constraints (or dimensions) rather than just one. Each item i has a value v_i and multiple weights $\{w_i^1, \dots, w_i^D\}$, each corresponding to a different dimension d . The goal is to select a subset of items that maximizes the total value while ensuring that the sum of the weights in each dimension does not exceed the capacity W^d of that dimension. This problem has numerous practical applications, including resource allocation, budgeting, and portfolio optimization, where multiple constraints must be simultaneously satisfied.

Constraint Programming Model

Let x_i be a binary variable indicating if the item i is inserted into the knapsack. This problem can be easily formalized as a constrained optimization problem.

$$\begin{aligned} \max_x \quad & \sum_{i=1}^n v_i x_i \\ \text{s. t.} \quad & \sum_{i=1}^n w_i^d x_i \leq W^d \quad \forall d \in \{1, \dots, D\} \\ & x_i \in \{0, 1\} \quad \forall i \in \{1, \dots, n\} \end{aligned} \quad (16)$$

Graph Encoding

We propose the following encoding to represent a MKP instance into a graph $G(V, E)$. We recall that in our representation, each node $v \in V$ corresponds to a specific Lagrangian multiplier. A node v is created for each pair of variable and constraint present in the problem instance. Then, an instance of n items and d dimensions yields a graph with $n \times d$ nodes. Nodes are linked by edges if they share the same variable or constraint. Each node v is also decorated with six 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.

Shift Scheduling Problem

The *Shift-Scheduling Problem* (SSP) involves scheduling employees' activities (e.g., work, break, lunch, rest) while adhering to labor regulations. Constraint programming is well-suited for solving this problem, as complex work regulations can be naturally formalized using REGULAR constraints (Pesant 2004). These constraints are typically specified using a non-deterministic finite automaton \mathcal{A} , defined by the 5-tuple $\mathcal{A} = (Q, A, \delta, q_0, F)$ where:

- Q is a finite set of states;
- A is the alphabet;
- $\delta : Q \times A \rightarrow Q$ is the transition function;

- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final states.

The constraint $\text{REGULAR}(\langle x_1, \dots, x_n \rangle, \mathcal{A})$ is satisfied if the sequence of variable values $\langle x_1, \dots, x_n \rangle$ belongs to the regular language recognized by a deterministic finite automaton \mathcal{A} . Intuitively, each variable x_i represents an employee's activity at time i . The constraint is satisfied if the sequence of states complies with the transition function and leads to a final state in the automaton. We consider in this paper a simple yet challenging version of the SSP with one employee, complex labor regulations, and multiple regular constraints.

Constraint Programming Model

Let W be the set of work activities, T the set of periods, \mathcal{A} the set of deterministic finite automata representing labor regulations, and p_{ij} (from a table P) the profit from assigning activity $i \in W$ to period $j \in T$. Each period must have an assigned activity that respects the automata transitions and maximizes total profit. Let x_i be a decision variable representing the activity assigned to period j , and let $\text{ELEMENT}(i, T, v)$ be a constraint that holds if $v = T[i]$. The SSP model is formulated as proposed by Hà, Quimper, and Rousseau (2015). Specifically, $\text{ELEMENT}(x_j, P, c_j)$ ensures that the profit c_j equals $p_{i,j}$ if and only if $x_j = i$.

$$\begin{aligned} \max_x \quad & \sum_{j \in T} c_j \\ \text{s. t.} \quad & \text{REGULAR}(\langle x_1, \dots, x_t \rangle, \mathcal{A}_i) \quad \forall \mathcal{A}_i \in \mathcal{A} \\ & \text{ELEMENT}(x_j, p_{i,j}, c_j) \quad \forall i \in W, \forall j \in T \\ & x_j \in W \quad \forall j \in T \\ & \min_{i \in W} p_{ij} \leq c_j \leq \max_{i \in W} p_{ij} \quad \forall j \in T \\ & c_j \in \mathbb{R} \quad \forall j \in T \end{aligned} \quad (17)$$

Graph Encoding

We propose the following encoding to represent a SSP instance into a graph $G(V, E)$. A node v is created for each triplet of (variable, constraint, value) present in the problem instance. Then, an instance of n variables, m constraints, and d values inside the domain yields a graph with $n \times m \times d$ nodes. Nodes are linked by edges if they correspond to the same (variable, value) pair or the same constraint, subject to a valid transition defined by the automata. Each node is decorated with four 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.