Scaling Decision-Focused Learning to Large Problems with Lagrangian Decomposition

Stéphane Eilles-Chan Way[1,2] , **Hugo Percot**[1,2] , **Quentin Cappart**[1,3,4] , **Tias Guns**[5] , **Louis-Martin Rousseau**[1]
[1]Polytechnique Montréal, Montreal, Canada
[2]Ecole Polytechnique, Palaiseau, France
[3]UCLouvain, Louvain-la-Neuve, Belgium
[4]Mila - Québec AI Institute, Montreal, Canada
[5]KU Leuven, Leuven, Belgium
{stephane.eilles-chan-way,hugo.percot}@polytechnique.edu,
tias.guns@kuleuven.be,
{quentin.cappart,louis-martin.rousseau}@polymtl.ca

Abstract

Decision-focused learning has shown great promise for addressing predict-then-optimize problems, particularly in the presence of under-specified models. However, its practical deployment is often hindered by high computational costs and limited scalability, as it requires solving a constrained optimization problem for each training instance at every iteration. To address these challenges, we propose a novel framework that incorporates Lagrangian decomposition into the decision-focused learning paradigm. Specifically, we introduce a new surrogate objective along with two loss functions for evaluating and training the underlying prediction model. We further propose two variants of our approach, which offer different trade-offs between computational efficiency and solution quality. Our framework can be seamlessly integrated with standard decision-focused learning methods, including *Smart Predict-then-Optimize* (`SPO+`) and *Implicit Maximum Likelihood Estimation* (`IMLE`). Through experiments on two standard benchmarks, the multi-dimensional knapsack problem and quadratic portfolio optimization, we demonstrate that our approach achieves competitive performance while remaining amenable to parallelization. In particular, it consistently outperforms traditional decision-focused learning methods on large-scale instances, involving up to eight times more variables than those typically considered in related work. The implementation is available at https://github.com/corail-research/DFL-LD.

1 Introduction

Many real-world problems can be formulated as *constrained optimization problems* (COPs), where the goal is to identify an optimal decision that maximizes or minimizes an objective function subject to a set of constraints. Both the objective function and the feasible region are typically parametric, depending on decision variables as well as contextual parameters. Although decades of research have produced highly efficient solvers for scenarios in which these contextual parameters are known, such information is often unavailable in practical applications. Consequently, in most realistic settings, the contextual parameters must be predicted from external observations, for example, using a machine learning model, before being used by an optimization solver to determine the downstream optimal decision. This framework is commonly referred to as the *predict-then-optimize* paradigm. In such situations, suboptimal downstream decisions generally stem not from the limitations of the COP solver, but from inaccuracies in the predicted contextual parameters. Designing a predictive model that leads to near-optimal downstream decisions is therefore a central challenge. Traditional machine learning approaches, which optimize for predictive accuracy alone, often fall short because they ignore the combinatorial structure and decision sensitivity inherent to the underlying optimization problem.

To overcome this limitation, the *decision-focused learning* (DFL) paradigm has emerged. Rather than training the predictive model to minimize a standard prediction loss, this approach explicitly incorporates the quality of the downstream decisions into the learning process, thereby aligning prediction and optimization objectives. Although Elmachtoub et al. [Elmachtoub *et al.*, 2023] demonstrated that *prediction-focused learning* (i.e. treating estimation and decision making as independent steps) performs best when the underlying predictive model is well-specified, it is outperformed by DFL approaches in the presence of epistemic or aleatoric uncertainty (i.e. when the model is under-specified or the data are noisy). Early work on DFL proposed direct differentiation through the *argmin* operator, initially limited to unconstrained relaxations [Gould *et al.*, 2016], and later extended to more general settings via differentiation of the Karush-Kuhn-

Tucker conditions [Amos and Kolter, 2017]. Subsequent developments introduced several alternative formulations, including the SPO+ surrogate loss [Elmachtoub and Grigas, 2021] IMLE [Niepert *et al.*, 2021], CaVE [Tang and Khalil, 2024a], LAVA [Berden *et al.*, 2025], and the neuro-symbolic E-PLL [Defresne *et al.*, 2023] for discrete graphical models, among others. For a comprehensive review and benchmark comparison, we refer the reader to the survey by Mandi et al. [Mandi *et al.*, 2024]. While DFL effectively aligns learning with decision quality, it suffers from substantial computational overhead: at each training step and for every instance, a constrained optimization problem must be solved. Since these problems are often NP-hard, and as the number of solver calls scales with both dataset size and the number of epochs, the computational cost increases sharply with instance size. This turns the training process into a sequence of expensive combinatorial optimizations that severely limit scalability and usability to solve large-scale instances.

Separately, *Lagrangian decomposition* [Guignard and Kim, 1987] (LD) is a scalable optimization technique that has been successfully applied to large-scale combinatorial problems, in both mixed-integer programming and constraint programming contexts. Compared to the standard Lagrangian relaxation, LD can provide tighter dual bounds and handle arbitrary constraint structures, but at the expense of an increased computational cost: Guignard and Kim [Guignard and Kim, 1987] (see Theorem 3.1) prove that for linearly-constrained problems the LD bound is at least as tight as the Lagrangian relaxation. The key idea behind LD is to duplicate variables and partition the initial problem into independent subproblems. Ideally, these subproblems should preserve the combinatorial nature of the original formulation while being significantly faster to solve. The solutions of the subproblems can then be combined to derive a valid dual bound on the original problem. However, this bound depends on a vector of Lagrangian multipliers that coordinate the subproblems; minimizing it amounts to optimizing these multipliers, typically through subgradient methods [Shor, 2012]. Lagrangian decomposition also integrates naturally with branch-and-bound algorithms: as noted by Guignard and Kim, disagreements between variable copies directly suggest branching candidates, yielding a natural dichotomic branching rule. In the context of constraint programming, Hà et al. [Hà *et al.*, 2015] demonstrated that LD can serve as an automatic, though computationally expensive, bounding mechanism that substantially reduces the size of the search tree. More recently, Bessa et al. [Bessa *et al.*, 2025] showed that machine learning techniques can be leveraged to warm-start the subgradient optimization, thereby improving the efficiency of the LD process.

Inspired by this recent line of work, and by other contributions highlighting the benefits of combining learning with Lagrangian methods [Parjadis *et al.*, 2024; Abbas and Swoboda, 2024], we propose to integrate Lagrangian decomposition into a decision-focused learning pipeline. This integration enables DFL to scale effectively to very large combinatorial optimization problems. Our approach contrasts with most existing DFL research, which primarily focuses on developing new differentiation techniques. Instead, we introduce a general methodology that can be seamlessly combined with any existing differentiation algorithm, providing a flexible and scalable framework for decision-focused learning.

Our contributions are as follows: (1) a methodology to leverage Lagrangian decomposition into DFL, yielding a new surrogate function to optimize, (2) two loss functions to train a prediction model, and (3) the integration of our approach on two common DFL techniques (SPO+ and IMLE). The experiments are conducted on instances of the multi-dimensional knapsack problem with up to 300 variables and 10 constraints, which is roughly six times larger than what is commonly considered in prior work, and on instances of the portfolio optimization problem with up to 400 variables, representing an 8-fold increase compared to typical benchmarks. To our knowledge, this work constitutes the first attempt to incorporate Lagrangian decomposition into a DFL pipeline.

2 Problem Setting

In many industrial applications, certain parameters of the constrained optimization problem are uncertain and must be inferred from contextual data, commonly referred to as *features*. The settings we consider involve estimating these parameters through predictive inferences made by machine learning models, after which the final decisions are obtained by solving the corresponding optimization problems based on these predictions. In this framework, the overall decision-making process can be formulated as a *predict-then-optimize problem*, where the optimization depends on parameters inferred from observed features.

Let $\mathbf{c} \in \mathbb{R}^k$ denote a vector of k unknown parameters, $\mathbf{X}$ decision variables defined over domain D, $C = (C_1, \ldots, C_m)$ a set of m constraints, and $f : D \times \mathbb{R}^k \to \mathbb{R}$ an objective function. The problem we consider is as follows:

$$\max_{\mathbf{X} \in D} \left\{ f(\mathbf{X}, \mathbf{c}) \;\middle|\; \bigwedge_{i=1}^{m} C_i(\mathbf{X}) \right\}. \tag{$\mathcal{P}_c$}$$

The optimal solution is denoted by $\mathbf{X}^*$. Although the optimal solution may not be unique, we only require a consistent tie-breaking rule so that $\mathbf{c} \mapsto \mathbf{X}^*(\mathbf{c})$ is well-defined as a function. In practice, this is satisfied since the solver returns a single deterministic solution.

In practice, the true cost parameters $\mathbf{c}$ are not directly observable and must be inferred from feature vectors $\mathbf{z} \in \mathbb{R}^q$. This inference is performed using a predictive model $\Omega_\theta : \mathbb{R}^q \to \mathbb{R}^k$ which produces an estimate $\hat{\mathbf{c}} = \Omega_\theta(\mathbf{z})$, where θ denotes the parameters of the model. The decision ultimately implemented is referred to as the *prescriptive solution*, denoted $\mathbf{X}^*(\hat{\mathbf{c}})$. This solution is generally suboptimal, and its quality can be assessed through the regret, defined as:

$$\mathcal{L}_{\text{regret}}(\hat{\mathbf{c}}, \mathbf{c}) = f(\mathbf{X}^*(\mathbf{c}), \mathbf{c}) - f(\mathbf{X}^*(\hat{\mathbf{c}}), \mathbf{c}). \tag{1}$$

Our objective is to learn a predictive model that produces prescriptive solutions as close as possible to the true optimal ones over a set of historical observations, or equivalently, to minimize the expected regret. To perform this prediction task, we employ neural networks as a differentiable predictive models. Training is carried out using gradient descent,

which requires a loss function whose gradient can be efficiently computed or approximated. The choice of an appropriate loss is critical, as it has a major impact on the overall performance of the model.

Prediction-focused learning is arguably the most intuitive approach to address this problem. In this paradigm, estimation and decision making are treated as two fully independent steps. The predictive model Ω_θ is trained to estimate $\mathbf{c}$ as accurately as possible by minimizing a prediction loss, typically the mean squared error between the true parameters $\mathbf{c}$ and their estimates $\hat{\mathbf{c}}$. Formally, this loss is defined as follows:

$$\mathcal{L}_{\text{MSE}}(\hat{\mathbf{c}}, \mathbf{c}) = \|\mathbf{c} - \hat{\mathbf{c}}\|^2. \tag{2}$$

However, this approach has several limitations, particularly in under-specified settings, as demonstrated by Elmachtoub et al. [Elmachtoub *et al.*, 2023]. This is why decision-focused learning is often more appropriate, as it directly targets the quality of the prescriptive decision. In this setting, the regret, as defined in Equation (1), is used as the loss function. However, this formulation introduces new challenges. First, computing the gradient of Equation (1) is difficult, and sometimes even meaningless, as $\frac{\partial \mathbf{X}^*}{\partial \hat{\mathbf{c}}}$ can potentially be null almost everywhere. This occurs, for example, when the mapping $\mathbf{c} \mapsto \mathbf{X}^*(\mathbf{c})$ is piecewise constant, as is common in combinatorial optimization. Fortunately, prior work has proposed ways to obtain exact gradients in special cases, or useful gradient approximations in more general scenarios, thereby enabling DFL in principle. Second, DFL suffers from substantial computational cost. Because it relies on the prescriptive decision during training, a new COP must be solved for every instance at every epoch. When the underlying problem is NP-hard, these repeated solver calls often become the primary bottleneck in the training pipeline. As a result, classical DFL methods do not scale well to large COPs, whose complexity grows exponentially. Our contribution is an automatic mechanism based on Lagrangian decomposition to tackle this challenge.

3 Lagrangian Decomposition for DFL

The key idea of Lagrangian decomposition is to provide a dual bound to a given multi-constrained optimization problem by solving mono-constrained ones. Let us start from $(\mathcal{P}_c)$ defined earlier. Let $\mathbf{X}_1$ refer to the initial variables. We introduce $m-1$ new variables by duplicating $\mathbf{X}_1$, obtaining a vector $\tilde{\mathbf{X}} = (\mathbf{X}_1, \dots, \mathbf{X}_m)$ where each $\mathbf{X}_i$ keeps the same domain as $\mathbf{X}_1$. We first observe that, by adding equality constraints, the original COP $(\mathcal{P}_c)$ is equivalent to:

$$\max_{\tilde{\mathbf{X}} \in D} \left\{ f(\mathbf{X}_1, \mathbf{c}) \,\middle|\, \left(\bigwedge_{i=1}^{m} C_i(\mathbf{X}_i) \right) \wedge \left(\bigwedge_{i=2}^{m} \mathbf{X}_i = \mathbf{X}_1 \right) \right\} \tag{3}$$

The notation $C_i(X_i)$ means that constraint C_i is applied to X_i. This does not require any structural assumption on the constraints since each X_i is a full copy of the original variable vector X (same dimension and domain). We then relax the new equality constraints by adding penalty terms into the objective function with multipliers $\mu = (\boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_m) \in \mathbb{R}^{|\mathbf{X}_1| \times (m-1)}$. By relaxing the equality constraints, we obtain a dual bound:

$$B(\mu, \mathbf{c}) = \max_{\tilde{\mathbf{X}} \in D} \left\{ f(\mathbf{X}_1, \mathbf{c}) + \sum_{i=2}^{m} \boldsymbol{\mu}_i \cdot (\mathbf{X}_1 - \mathbf{X}_i) \,\middle|\, \bigwedge_{i=1}^{m} C_i(\mathbf{X}_i) \right\}. \tag{4}$$

We note that this bound is parametrized by the multipliers μ. Those are commonly referred to as *Lagrangian multipliers*. Since each constraint C_i acts on a distinct variable and the objective function is separable on the variables, the computation of $B(\mu, \mathbf{c})$ can be decomposed into m subproblems featuring only a single constraint:

$$B(\mu, \mathbf{c}) = \Phi(\mu, \mathbf{c}) + \sum_{i=2}^{m} \Psi_i(\boldsymbol{\mu}_i), \text{ where} \tag{5}$$

$$\Phi(\mu, \mathbf{c}) = \max_{\mathbf{X}_1 \in D} \left\{ f(\mathbf{X}_1, \mathbf{c}) + \sum_{i=2}^{m} \boldsymbol{\mu}_i \cdot \mathbf{X}_1 \,|\, C_1(\mathbf{X}_1) \right\} \tag{6}$$

$$\Psi_i(\boldsymbol{\mu}_i) = \max_{\mathbf{X}_i \in D} \{ -\boldsymbol{\mu}_i \cdot \mathbf{X}_i \mid C_i(\mathbf{X}_i) \}, \forall i \in [\![2, m]\!]. \tag{7}$$

In practice, we are interested in finding the tightest possible dual bound, which requires computing the best multipliers i.e. $\mu^*(\mathbf{c}) = \text{argmin}_\mu(B(\mu, \mathbf{c}))$. This can be done with a subgradient descent method to optimize the argmin, such as Shor's algorithm [Shor, 2012]. Let $\tilde{\mathbf{X}}^*(\mu, \mathbf{c}) = (\mathbf{X}_1^*(\mu, \mathbf{c}), \dots, \mathbf{X}_m^*(\mu, \mathbf{c}))$ be the solution of Equation (4) for a given μ. A standard subgradient expression of $B(\mu, \mathbf{c})$ with regard to $\boldsymbol{\mu}_i$ is $(\mathbf{X}_1^*(\mu, \mathbf{c}) - \mathbf{X}_i^*(\mu, \mathbf{c}))$ for all $i \in [\![2, m]\!]$ [Bessa *et al.*, 2025].

3.1 Defining Appropriate Loss Functions

Lagrangian decomposition provides a fast way to compute an upper bound, although the bound's tightness depends on the chosen multipliers. Ideally, we would replace the COP-solving phase in the traditional DFL pipeline with Lagrangian decomposition using optimal multipliers, which yield the tightest dual bound. Taking a step back, we note that Equation (4), when parameterized and evaluated with optimal multipliers, becomes:

$$B\big(\mu^*(\mathbf{c}), \mathbf{c}\big) = \max_{\tilde{\mathbf{X}} \in D} \left\{ f(\mathbf{X}_1, \mathbf{c}) + \sum_{i=2}^{m} \boldsymbol{\mu}_i(\boldsymbol{c}) \cdot (\mathbf{X}_1 - \mathbf{X}_i) \,\middle|\, \bigwedge_{i=1}^{m} C_i(\mathbf{X}_i) \right\}. \tag{8}$$

This expression is a new COP parameterized by $(\mu^*(\mathbf{c}), \mathbf{c})$. Therefore, we can reuse the tools from DFL, except that we need to handle the multipliers $\mu^*(\mathbf{c})$ as well, which act as parameters in Equation (8) along $\mathbf{c}$, but are not given by the predictive model upstream. In contrast, they are obtained by a third-party optimization algorithm. The next step is to design an appropriate DFL-based loss function. Let $\tilde{\mathbf{X}}^*(\mu^*, \mathbf{c}) = (\mathbf{X}_1^*(\mu^*, \mathbf{c}), \dots, \mathbf{X}_m^*(\mu^*, \mathbf{c}))$ denote the solution to Equation (8) and let $b : (\tilde{\mathbf{X}}, \mu, \mathbf{c}) \mapsto f(\mathbf{X}_1, \mathbf{c}) +$

$\sum_{i=2}^{m} \boldsymbol{\mu}_i \cdot (\mathbf{X}_1 - \mathbf{X}_i)$ be a function mapping for Lagrangian decomposition. We propose two loss functions that integrate Lagrangian decomposition into a DFL pipeline:

$$\mathcal{L}_1(\hat{\mathbf{c}}, \mathbf{c}) = b(\tilde{\mathbf{X}}^*(\mu^*(\hat{\mathbf{c}}), \hat{\mathbf{c}}), \mu^*(\mathbf{c}), \mathbf{c}) - b(\tilde{\mathbf{X}}^*(\mu^*(\mathbf{c}), \mathbf{c}), \mu^*(\mathbf{c}), \mathbf{c}) \quad (9)$$

$$\mathcal{L}_2(\hat{\mathbf{c}}, \mathbf{c}) = f(\mathbf{X}_1^*(\mu^*(\hat{\mathbf{c}}), \hat{\mathbf{c}}), \mathbf{c}) - f(\mathbf{X}_1^*(\mu^*(\mathbf{c}), \mathbf{c}), \mathbf{c}) \quad (10)$$

The first loss ($\mathcal{L}_1$) corresponds exactly to the regret associated with Equation (8). It involves the objective function $b(\tilde{\mathbf{X}}, \mu, \mathbf{c})$ where the Lagrangian multipliers appear explicitly. In contrast, in the second loss ($\mathcal{L}_2$), explicit penalty terms with the multipliers disappear since $f(\mathbf{X}, \mathbf{c})$ is used instead of $b(\tilde{\mathbf{X}}, \mu, \mathbf{c})$. However, the multipliers are still involved in the resolution of $\mathbf{X}_1^*$, so they are still ultimately considered. Intuitively, $\mathcal{L}_2$ mirrors the regret defined from ($\mathcal{P}_c$), but it is evaluated at $\mathbf{X}_1^*(\mu^*, \hat{\mathbf{c}})$ rather than $\mathbf{X}^*(\hat{\mathbf{c}})$. We use $\mathbf{X}_1^*$ (instead of the duplicated variables $\tilde{\mathbf{X}}^*$) because these variables are obtained from Equation (5), which directly incorporates both the objective f and the multipliers. Since the bottleneck in DFL is the solving time of the COP at each iteration, we expect the two losses based on Lagrangian decomposition to be significantly faster. As a counterpart, as they are based on a relaxed problem (Eq. (8)), the accuracy of the estimator $\hat{\mathbf{c}}$ on the primal ($\mathcal{P}_c$) is not guaranteed. The choice to use $f(\mathbf{X}, \mathbf{c})$ in $\mathcal{L}_2$ is motivated by the idea that it could strengthen the performance during evaluation on ($\mathcal{P}_c$). Figure 1 illustrates how these two losses differ from the standard regret loss.

3.2 Dealing with Multipliers under Uncertainty

Unfortunately, minimizing these two loss functions is challenging, mainly because the multipliers depend on the uncertain parameters, i.e., $\mu^*(\hat{\mathbf{c}})$. A first issue is that computing $\mu^*(\hat{\mathbf{c}})$ for each instance would significantly increase computational cost. Moreover, differentiating the losses (i.e., $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{c}}}$) requires computing both $\frac{\partial \mathbf{X}_1^*}{\partial \hat{\mathbf{c}}}$ and $\frac{\partial \mu^*}{\partial \hat{\mathbf{c}}}$. As a result, optimizing these losses becomes a bilevel optimization problem. We address this challenge by *fixing* the Lagrangian multipliers rather than optimizing over $\mu^*(\hat{\mathbf{c}})$. With fixed multipliers, the optimal solutions to the subproblems $(\mathbf{X}_2^*, \ldots, \mathbf{X}_m^*)$ no longer depend on $\hat{\mathbf{c}}$ (Eq. (7)). The next step is to decide how to set these multipliers. A natural choice is to fix them to their optimal values under the true cost parameters, i.e., we set $\mu^*(\hat{\mathbf{c}}) = \mu^*(\mathbf{c})$. Intuitively, this choice reflects the multipliers we would like to obtain at the end of training. The two corresponding losses can then be reformulated as follows:

$$\mathcal{L}_1^{\text{new}}(\hat{\mathbf{c}}, \mathbf{c}) = \sigma(\mathbf{X}_{\mathbf{1}}^*(\mu^*(\mathbf{c}), \hat{\mathbf{c}}), \mu^*(\mathbf{c}), \mathbf{c}) - \sigma(\mathbf{X}_{\mathbf{1}}^*(\mu^*(\mathbf{c}), \mathbf{c}), \mu^*(\mathbf{c}), \mathbf{c}) \quad (11)$$

$$\mathcal{L}_2^{\text{new}}(\hat{\mathbf{c}}, \mathbf{c}) = f(\mathbf{X}_{\mathbf{1}}^*(\mu^*(\mathbf{c}), \hat{\mathbf{c}}), \mathbf{c}) - f(\mathbf{X}_{\mathbf{1}}^*(\mu^*(\mathbf{c}), \mathbf{c}), \mathbf{c}) \quad (12)$$

where $\sigma : (\mathbf{X_1}, \mu, \mathbf{c}) \mapsto f(\mathbf{X_1}, \mathbf{c}) + \sum_{i=2}^{m} \boldsymbol{\mu}_i \cdot \mathbf{X_1}$. The resulting training algorithm, referred to as `trainingSingle` since it relies on a single fixed Lagrangian decomposition, is depicted in Algorithm 1. It illustrates how the predictive model Ω_θ is trained using a dataset $\mathcal{S} = \{(\mathbf{z}_i, \mathbf{c}_i, \mathbf{X}_{\mathbf{1}}^*(\mathbf{c}_i), \mu^*(\mathbf{c}_i))\}_{i=1}^n$. Recall that $\mathbf{z}_i$ denotes the feature vector of data sample i. We note that constructing this training set requires computing the Lagrangian multipliers, via a subgradient method, for all instances in the training set and for multiple cost parameter values, which can be computationally expensive. Instead, we propose to terminate the subgradient procedure after a predefined number of iterations and to use the resulting potentially suboptimal multipliers. As we will show in Section 5, this early termination still leads to competitive results in practice. The model weights are initialized randomly (Line 1), and training proceeds for a fixed number of epochs. Each data sample is processed once per epoch. For every sample, we first predict the cost parameters from the feature vector (Line 5), then execute the Lagrangian decomposition of the problem (Line 6), and finally solve the resulting subproblems (Line 7). As explained previously, only the variables $\boldsymbol{X}_{\mathbf{1}}^*$ are required to compute the loss. The weights are then updated by backpropagating the loss using the chain rule (Line 8). The term $\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}_{\mathbf{1}}^*}$ is obtained directly from Eq. (11) or Eq. (12) and $\frac{\partial \hat{\boldsymbol{c}}_i}{\partial \theta}$ is computed by differentiating through Ω_θ, e.g., a neural network. Concerning the term $\frac{\partial \boldsymbol{X}_{\mathbf{1}}^*(\hat{\boldsymbol{c}}_i, \mu)}{\partial \hat{\boldsymbol{c}}_i}$, we emphasize that the main advantage of our approach is that we do not need anymore to consider all the variables as in standard DFL. The LD subproblem is easier to solve and differentiate because it is smaller and has a simpler structure. This gradient can be obtained using classical DFL differentiation techniques or, when the subproblem has exploitable structure, via dedicated solver-free differentiation methods. These two situations are later considered in our experiments. Since our approach can be combined with any differentiation technique, it provides flexibility in selecting the most suitable one for a given use case. Note that some DFL differentiation techniques, e.g. `SPO+`, provide the straightforward gradient $\frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{c}}_i}$ instead of $\frac{\partial \boldsymbol{X}_{\mathbf{1}}^*(\hat{\boldsymbol{c}}_i, \mu)}{\partial \hat{\boldsymbol{c}}_i}$, which is still compatible with Algorithm 1.

Algorithm 1 `trainingSingle`$(\mathcal{S}, \Omega_\theta, \mathcal{P}, \mathcal{L}, \alpha, E)$

Require: Dataset $\mathcal{S} = \{(\mathbf{z}_i, \mathbf{c}_i, \mathbf{X}_{\mathbf{1}}^*(\mathbf{c}_i), \mu^*(\mathbf{c}_i))\}_{i=1}^n$.
Require: Predictive model Ω_θ.
Require: Combinatorial problem $\mathcal{P} = \langle \mathbf{X}, D, C, f \rangle$.
Require: Loss function $\mathcal{L} \in \{\mathcal{L}_1^{\text{new}}, \mathcal{L}_2^{\text{new}}\}$.
Require: learning rate α, number of training epochs E.
1: $\theta :=$ `randomInitialization()`
2: **for each** epoch **from** 1 **to** E **do**
3: **for each** sample $i \in \mathcal{S}$ **do**
4: $\mu := \mu^*(\mathbf{c}_i)$
5: $\hat{\boldsymbol{c}}_i := \Omega_\theta(\boldsymbol{z}_i)$
6: $\Phi :=$ `LD`$(\mathcal{P})$ {Eq. (6)}
7: $\boldsymbol{X}_{\mathbf{1}}^* :=$ `solve`$(\Phi, \mu, \hat{\boldsymbol{c}}_i)$ {Subproblem solving}
8: $\theta := \theta - \alpha\big(\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}_{\mathbf{1}}^*} \cdot \frac{\partial \boldsymbol{X}_{\mathbf{1}}^*(\hat{\boldsymbol{c}}_i, \mu)}{\partial \hat{\boldsymbol{c}}_i} \cdot \frac{\partial \hat{\boldsymbol{c}}_i}{\partial \theta}\big)$ {Chain rule}
9: **end for**
10: **end for**
11: **return** Ω_θ

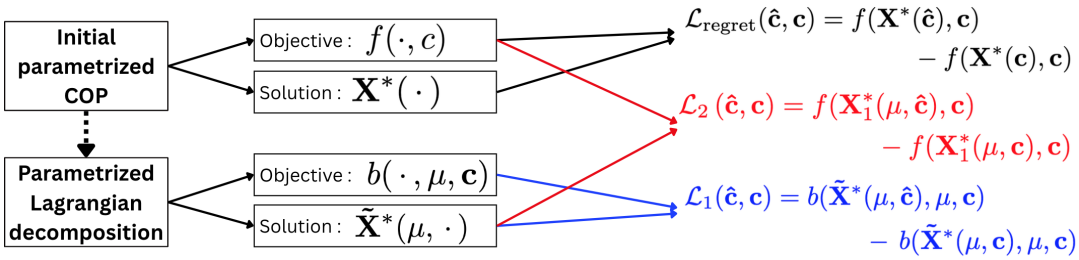


Figure 1: Schematic view of our method. The initial parameterized COP and the parameterized LD define two optimization problems, each with its own objective function and corresponding optimal solution. Each approach selects a particular combination of these components to construct a loss function. The regret $\mathcal{L}_{\text{regret}}$ denotes the classical DFL loss, while $\mathcal{L}_1$ and $\mathcal{L}_2$ are the new losses we propose.

3.3 Handling Multiple Decompositions

Our current approach considers only the subset $\boldsymbol{X}_1 \in \tilde{\mathbf{X}}$ when computing the loss. Although this choice improves overall computational efficiency, it raises an important question: the selection of variables to serve as the basis for duplication (see Eq. (3)) is inherently arbitrary. This nontrivial design choice can have a significant impact on the performance of our method. We propose to mitigate this issue by dynamically changing, during training, which variables $\boldsymbol{X}_i \in \tilde{\mathbf{X}}$ are used for duplication and by leveraging multiple decompositions. A problem with m constraints then admits m possible decompositions. Let us denote by $\boldsymbol{X}_1^{(d)}$ and $\boldsymbol{X}_1^{*(d)}$ the variables and their optimal values in the corresponding subproblem for decomposition $d \in [\![1, m]\!]$, respectively. Their associated losses $\mathcal{L}_1^{(d)}$, and $\mathcal{L}_2^{(d)}$ are defined as follows.

$$\begin{aligned}\mathcal{L}_1^{(d)}(\hat{\mathbf{c}}, \mathbf{c}) =& \sigma(\mathbf{X}_{\mathbf{1}}^{*(d)}(\mu^*(\mathbf{c}), \hat{\mathbf{c}}), \mu^*(\mathbf{c}), \mathbf{c}) \\ &- \sigma(\mathbf{X}_{\mathbf{1}}^{*(d)}(\mu^*(\mathbf{c}), \mathbf{c}), \mu^*(\mathbf{c}), \mathbf{c}) \qquad (13)\\ \mathcal{L}_2^{(d)}(\hat{\mathbf{c}}, \mathbf{c}) =& f(\mathbf{X}_{\mathbf{1}}^{*(d)}(\mu^*(\mathbf{c}), \hat{\mathbf{c}}), \mathbf{c}) \\ &- f(\mathbf{X}_{\mathbf{1}}^{*(d)}(\mu^*(\mathbf{c}), \mathbf{c}), \mathbf{c}). \qquad (14)\end{aligned}$$

Intuitively, these updated losses convey more information about the underlying problem. We propose to select which decomposition to use at random at each training iteration. As a drawback, since each decomposition d selects a different variable and constraint for the main subproblem, it yields different Lagrangian multipliers $\mu^{*(d)}(\mathbf{c}_i)$ and subproblem solutions $\mathbf{X}_{\mathbf{1}}^{*(d)}(\mathbf{c}_i)$. We therefore need to construct a separate training set for each decomposition, $\mathcal{S}^{(d)} = \{(\mathbf{z}_i, \mathbf{c}_i, \mathbf{X}_{\mathbf{1}}^{*(d)}(\mathbf{c}_i), \mu^{*(d)}(\mathbf{c}_i))\}_{i=1}^n$, which is more time-consuming, but can still be performed offline. Moreover, these m constructions are fully independent and thus easily parallelizable. The resulting training algorithm is depicted in Algorithm 2. It simply consists in executing Algorithm 1 (without the random initialization of the model parameters θ) with a randomly selected decomposition at each epoch (Line 4).

Algorithm 2 `trainingMultiple()`

Require: Number of decompositions m.
Require: $\mathcal{S}^{(d)} = \{(\mathbf{z}_i, \mathbf{c}_i, \mathbf{X}_{\mathbf{1}}^*(\mathbf{c}_i)^{(d)}, \mu^*(\mathbf{c}_i)^{(d)})\}_{i=1}^n$.
Require: Predictive model Ω_θ.
Require: Combinatorial problem $\mathcal{P} = \langle \mathbf{X}, D, C, f \rangle$.
Require: Loss function $\mathcal{L}^{(d)} \in \{\mathcal{L}_1^{(d)}, \mathcal{L}_2^{(d)}\}\ \forall d \in [\![1, m]\!]$.
Require: learning rate α, number of training epochs E.
1: $\theta :=$ `randomInitialization()`
2: **for each** epoch **from** 1 **to** E **do**
3: $\quad d :=$ `randomSelection`$(1, m)$
4: $\quad \Omega_\theta :=$ `trainingSingle`$(S^{(d)}, \Omega_\theta, \mathcal{P}, \mathcal{L}^{(d)}, \alpha, 1)$
5: **end for**
6: **return** Ω_θ

4 Case Studies

This paper addresses two challenging problems in the DFL literature: the multi-dimensional knapsack problem and the quadratic portfolio optimization problem. These benchmarks are chosen to enable direct comparison with the work of Elmachtoub and Grigas [Elmachtoub and Grigas, 2021] and Mandi et al. [Mandi *et al.*, 2020].

Multi-dimensional knapsack. It is an extension of the classic knapsack problem. In the multidimensional knapsack problem, we are given m constraints and n items, each associated with a profit $\mathbf{c} \in \mathbb{R}^n$ and multiple weights $(\mathbf{w}^1, \ldots, \mathbf{w}^m) \in \mathbb{R}^{n \times m}$. Each constraint $k \in \{1, \ldots, m\}$ represents a capacity limit b_k for a different dimension. The objective is to select a subset of items that maximizes total profit while ensuring that the total weight in each dimension does not exceed the corresponding capacity constraint. All benchmark instances follow the protocol explained by [Tang and Khalil, 2024b] and summarized in the survey of Mandi et al. [Mandi *et al.*, 2024]. For each data point i, shared features $\mathbf{z}_i \in \mathbb{R}^p$ are generated and then transformed into item profits $(c_{i,j})_{j=1,\ldots,n}$ via the following projection:

$$c_{i,j} = \left\lceil \left[\tfrac{5}{3.5^t} \left(\tfrac{1}{\sqrt{p}} (\mathcal{B}\mathbf{z}_i)_j + 3 \right)^t + 1 \right] \epsilon_{i,j} \right\rceil. \qquad (15)$$

Here, $\epsilon_{i,j}$ is sampled uniformly from the interval $[1-\bar{\epsilon}, 1+\bar{\epsilon}]$, and $\mathcal{B}$ denotes a random projection matrix. Value t is used to parametrize the degree of the projection. Item weights are sampled uniformly from 3 to 8 and remain fixed across the dataset, ensuring that uncertainty affects only the objective coefficients. The capacity limits b_k equal the half of the sum of each constraint weights $(w_1^k, \ldots, w_n^k)$ and also remain fixed. We use $m = 10$, $n \in \{100, 200, 300\}$, $t = 8$, $p = 12$ and $\epsilon = 0.5$, corresponding to substantially larger

problem instances than those typically considered in the literature. For example, the recent survey by Mandi et al. reports knapsack instances with only a single constraint and at most 45 items. The predictive model Ω_θ is a simple linear regression. The sub-problems are one-dimensional knapsack problems and are solved using `Gurobi` [Gurobi Optimization, LLC, 2024] (version 12.0.0). Finally, for each size n, 1300 instances are generated and split as follows: 200 for training, 100 for validation and 1000 for the final evaluation.

Quadratic portfolio optimization. Given predicted asset returns $\mathbf{c} \in \mathbb{R}^n$ and a positive semidefinite covariance matrix $\Sigma \in \mathbb{R}^{n\times n}$ with average entry $\bar{\Sigma}$, the decision maker chooses portfolio weights $\mathbf{x} \in \mathbb{R}^n$ to maximize expected return, $\mathbf{c}^\top \mathbf{x}$, while respecting a risk budget, $\mathbf{x}^\top \Sigma \mathbf{x} \leq \gamma\bar{\Sigma}$, and fully investing the available capital, $\mathbf{1}^\top \mathbf{x} = 1$. Again, the instances are generated following the procedure detailed by Mandi et al. using the hyperparameters they recommend. In particular, five features and a simple linear regression for the predictive model are used. We consider instances with $n = \{50, 200, 400\}$ assets. For comparison, [Elmachtoub and Grigas, 2021] consider instances with at most 50 assets. The sub-problems are solved using `Gurobi` [Gurobi Optimization, LLC, 2024] (version 12.0.0). Finally, 10125 problems are generated and split as follows: 100 for training, 25 for validation and 10000 for the final evaluation. Interestingly, when choosing the risk constraint as the main constraint, the resulting LD main subproblem:

$$\max_{\mathbf{X}_1 \geq 0} \big(\mathbf{c} + \boldsymbol{\mu_2}\big)^\top \mathbf{X}_1 \quad \text{s.t.} \quad \mathbf{X}_1^\top \Sigma \mathbf{X}_1 \leq \gamma\bar{\Sigma} \tag{16}$$

admits a closed-form optimal solution once we relax the non-negativity constraint on the decision variable, given by

$$\bar{\mathbf{X}}_1^*(\mathbf{c}, \mu) = \sqrt{\frac{\gamma\bar{\Sigma}}{\big(\mathbf{c} + \boldsymbol{\mu_2}\big)^\top \Sigma^{-1} \big(\mathbf{c} + \boldsymbol{\mu_2}\big)}}\, \Sigma^{-1}\big(\mathbf{c} + \boldsymbol{\mu_2}\big). \tag{17}$$

We refer the readers to Appendix B for the derivation of this exact closed-form optimal solution. This allows us to design an exact differentiation scheme to compute this solution. During training, we replace the solver-provided solution $\mathbf{X}_1^*(\mathbf{c}, \mu)$ with the analytical expression $\bar{\mathbf{X}}_1^*(\mathbf{c}, \mu)$ (Eq. (17)). This eliminates the need for a solver and a generic DFL technique because its gradient can be obtained directly via automatic differentiation tools.

5 Computational Experiments

This section presents the results of our computational experiments. All tests were conducted on a cluster of CPU nodes equipped with AMD EPYC 9654 and Intel Xeon Gold 6148 (Skylake) processors, with each run restricted to a single core and 4 GB of RAM. The optimization problems are solved using `Gurobi` [Gurobi Optimization, LLC, 2024] (version 12.0.0). We compare our approach against a standard mean squared error loss (`MSE`) and the DFL algorithms `SPO+` [Elmachtoub and Grigas, 2021; Mandi *et al.*, 2020] and `IMLE` [Niepert *et al.*, 2021]. For these baselines, we rely on the implementations provided in `PyEPO` [Tang and Khalil, 2024b]. Predictive models are implemented and trained using `PyTorch` [Paszke *et al.*, 2019]. We also compare our approach with other methods focussed on scaling: `CaVE` and `CaVE+` [Tang and Khalil, 2024a] as well as `LAVA` [Berden *et al.*, 2025]. These methods are less generic than ours, and cannot be applied to quadratic portfolio optimization. We do not compare with `E-PLL` [Defresne *et al.*, 2023], a solver-free neuro-symbolic approach for discrete graphical models (cost function networks), as it does not directly apply to our continuous quadratic portfolio nor to the global linear capacity constraints of the multi-dimensional knapsack; a detailed comparison on suitable discrete benchmarks is left for future work. Our Lagrangian decomposition framework supports two modes, single and multiple decomposition (see the corresponding algorithms), with two loss functions available for each mode. In addition, it can be integrated into any DFL method. Finally, because our approach introduces additional overhead for optimizing the multipliers when constructing the training set, we consider two training-budget settings. In the first, the total budget (including dataset construction) is matched to that of the corresponding DFL baseline ($=$). In the second, we allow the multipliers to be trained until convergence (∞). We denote our approaches as $\texttt{LD}(\eta, \gamma, \mathcal{L}, B)$, where $\eta \in \{\texttt{SPO+}, \texttt{IMLE}\}$, $\gamma \in \{\texttt{static}, \texttt{multiple}, \texttt{exact}\}$, $\mathcal{L} \in \{\mathcal{L}_1, \mathcal{L}_2\}$, and $B \in \{=, \infty\}$. Method `exact` refers to the exact differentiation we designed for the portfolio problem. As evaluation metrics, we report the regret achieved by each approach over the test dataset, as commonly done in related works. Each model is trained with ten random seeds to assess the stability of the training procedure. The implementation is available at https://github.com/corail-research/DFL-LD.

5.1 Results: Performances with Unlimited Budget

This first experiment compares our approaches with prior baselines under an unrestricted training budget (denoted by the ∞ variant), meaning that all methods are trained to convergence, which is a commonly adopted assumption in the field. The results, reported as averages with 95% confidence intervals over 10 random seeds, are summarized in Table 1. In all settings, we use early stopping and retain the checkpoint achieving the lowest validation relative regret, the training time reported in Table 1 corresponds to the wall-clock time at which this best validation model is reached. We note that for the quadratic portfolio problem, the multiple-decomposition strategy is less relevant because one of the possible choices for the main subproblem (keeping the linear budget constraint) leads to a weakly informative optimization layer. We therefore focus on the decomposition that retains the quadratic risk structure (the subproblem forms are provided in Appendix A). Similarly, our $\mathcal{L}_2$ variant is less relevant for `SPO+`, as this differentiation technique is designed as a surrogate for regret and is therefore naturally aligned with the regret-like loss $\mathcal{L}_1$, whereas $\mathcal{L}_2$ does not correspond to a regret. Overall, we observe that the best performance is consistently achieved by one of our proposed approaches. Interestingly, neither `IMLE` nor `SPO+` dominates across all settings, which is consistent with the findings of Mandi et al. [Mandi *et al.*, 2024]. We view this as an important strength of our framework, as it can be combined with the most suitable decision-focused learning technique depending

Multi-dimensional knapsack (10 constraints)						
	100 items		200 items		300 items	
Approaches	Regret	Time (s)	Regret	Time (s)	Regret	Time (s)
MSE	7.796 ± 0.034	270 ± 44	5.518 ± 0.026	383 ± 37	6.774 ± 0.025	387 ± 20
CaVE	3.680 ± 0.052	68 ± 15	3.617 ± 0.034	107 ± 20	3.466 ± 0.038	182 ± 1
CaVE+	3.686 ± 0.029	1177 ± 816	3.553 ± 0.020	874 ± 783	3.265 ± 0.015	217 ± 188
LAVA	4.509 ± 0.04	4346 ± 1053	4.552 ± 0.02	6198 ± 377	4.558 ± 0.05	5878 ± 683
SPO+	3.049 ± 0.032	4692 ± 1128	2.975 ± 0.080	4350 ± 1724	3.072 ± 0.019	1904 ± 785
LD(SPO+, static, $\mathcal{L}_1, \infty$)	2.887 ± 0.024	41 ± 2	2.562 ± 0.016	42 ± 4	2.628 ± 0.035	107 ± 34
LD(SPO+, multiple, $\mathcal{L}_1, \infty$)	$\underline{2.621 \pm 0.030}$	8 ± 1	$\underline{2.496 \pm 0.012}$	1825 ± 571	$\mathbf{2.179 \pm 0.007}$	2485 ± 963
IMLE	4.234 ± 0.075	5836 ± 728	3.285 ± 0.047	6489 ± 327	3.933 ± 0.045	5988 ± 524
LD(IMLE, static, $\mathcal{L}_1, \infty$)	3.401 ± 0.062	3715 ± 929	3.309 ± 0.027	1087 ± 158	3.329 ± 0.032	4504 ± 579
LD(IMLE, static, $\mathcal{L}_2, \infty$)	3.809 ± 0.071	5339 ± 1210	3.438 ± 0.025	1148 ± 1289	3.712 ± 0.051	64 ± 12
LD(IMLE, multiple, $\mathcal{L}_1, \infty$)	2.807 ± 0.019	4813 ± 964	2.801 ± 0.038	610 ± 464	2.382 ± 0.015	6049 ± 312
LD(IMLE, multiple, $\mathcal{L}_2, \infty$)	$\mathbf{2.376 \pm 0.017}$	5293 ± 614	$\mathbf{2.189 \pm 0.011}$	5905 ± 352	$\underline{2.235 \pm 0.011}$	6401 ± 149

Quadratic portfolio optimization (2 constraints)						
	50 assets		200 assets		400 assets	
Approaches	Regret	Time (s)	Regret	Time (s)	Regret	Time (s)
MSE	2.325 ± 0.016	12 ± 3	2.070 ± 0.010	993 ± 765	2.324 ± 0.001	4648 ± 2827
SPO+	2.095 ± 0.025	58 ± 7	2.092 ± 0.006	4408 ± 288	2.087 ± 0.010	10277 ± 699
LD(SPO+, static, $\mathcal{L}_1, \infty$)	2.122 ± 0.021	101 ± 29	14.833 ± 0.250	6685 ± 124	16.84 ± 0.145	11698 ± 193
IMLE	2.064 ± 0.008	1450 ± 1115	$\underline{2.032 \pm 0.009}$	5586 ± 907	$\underline{2.026 \pm 0.004}$	11327 ± 1252
LD(IMLE, static, $\mathcal{L}_1, \infty$)	2.285 ± 0.011	67 ± 8	11.028 ± 0.143	6339 ± 291	12.31 ± 0.081	11469 ± 875
LD(IMLE, static, $\mathcal{L}_2, \infty$)	$\mathbf{2.026 \pm 0.004}$	282 ± 54	2.101 ± 0.020	6465 ± 477	2.114 ± 0.010	11640 ± 761
LD(Exact, static, $\mathcal{L}_1, \infty$)	2.159 ± 0.056	712 ± 853	2.075 ± 0.011	69 ± 49	2.360 ± 0.0191	1975 ± 1086
LD(Exact, static, $\mathcal{L}_2, \infty$)	$\underline{2.040 \pm 0.009}$	14 ± 5	$\mathbf{1.969 \pm 0.010}$	1309 ± 1683	$\mathbf{1.992 \pm 0.001}$	3614 ± 3544

Table 1: Test-set relative regret and time-to-best validation model for multi-dimensional knapsack and quadratic portfolio optimization (mean across seeds $\pm 95\%$ CI). Relative regret values must be multiplied by 10^{-2}. Best results are **in bold**. Second best results are underlined.

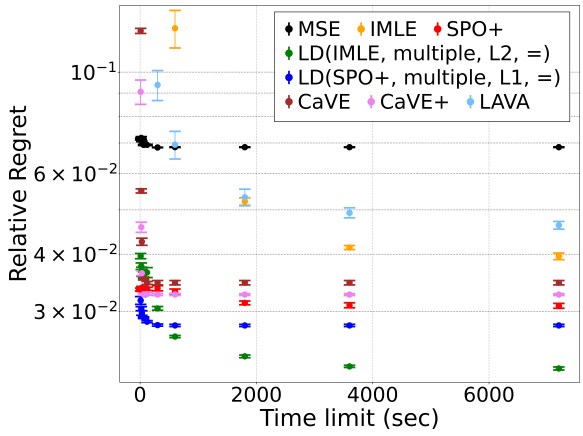
on the specific problem. Moreover, we observe that baselines designed to achieve low training times (CaVE, CaVE+ and LAVA) perform worse than other DFL algorithms IMLE or SPO+. This observation may be explained by the noisy nature of the datasets. Although CaVE and CaVE+ converge significantly faster than our approaches, our methods converge to models with superior final performance. Finally, the exact LD-based differentiation achieves the best results for the quadratic portfolio, showing the versatility of LD in DFL.

5.2 Analysis: Training with a Restricted Budget

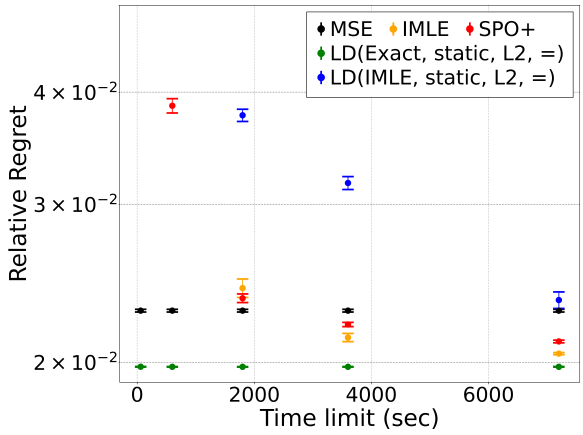
This second set of experiments analyzes the behavior of the methods under a fixed training-time budget, where all approaches are allocated the same execution time. For our methods, this budget includes the construction of the training set (denoted by the $=$ variant). We restrict the analysis to the largest instances and to our best-performing approaches. The results are shown in Figures 2a and 2b. We report the relative regret on the test set for different training-time budgets. Hyperparameters are tuned independently for each budget (via validation), so each point corresponds to a different trained model rather than successive checkpoints of a single run. As expected, thanks to the Lagrangian decomposition, our approach solves the underlying optimization problems more efficiently at each training iteration, as the resulting subproblems are easier to solve. This allows training epochs to be executed faster, which ultimately leads to improved performance compared to the baselines under the same training-time budget. This results suggest that the Lagrangian multipliers optimization overhead can be controlled while remaining competitive. To confirm the scalability of our approach, we report the number of training epochs completed within each training-time budget in Figures 3a and 3b. The results directly corroborate this hypothesis, as at least one LD-based approach consistently achieves the highest number of epochs for a given time budget, among DFL baselines.

5.3 Analysis: Benefits of Parallelization

Another benefit of the Lagrangian decomposition is that, in the multiple decomposition setting, each decomposition yields an independent sub-problem with its own set of Lagrangian multipliers. As a result, the offline dataset-construction step (i.e. computing these multipliers via subgradient iterations) is naturally amenable to parallelization. We evaluate the benefits of parallelization on the multi-dimensional knapsack problem with 200 items. In all runs, we allocate 10 CPU cores, we then vary how many parallel worker processes are used to distribute the independent mul-

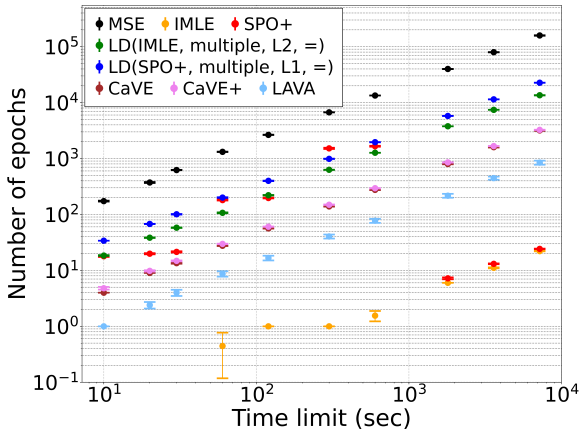


(a) MD knapsack ($n = 300$)

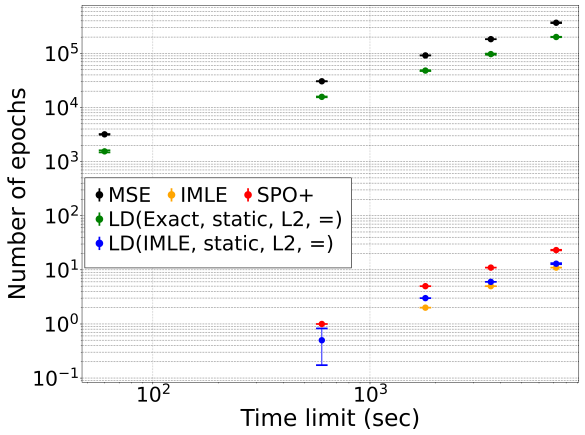


(b) Portfolio ($n = 400$)

Figure 2: Test-set relative regret (mean across seeds $\pm 95\%$ CI) with fixed training-time budget.



(a) MD knapsack ($n = 300$)



(b) Portfolio ($n = 400$)

Figure 3: Number of epochs achieved during training (mean across seeds $\pm 95\%$ CI) with fixed training-time budget.

tiplier optimizations across these cores (from 1 worker, i.e., fully sequential, up to 10 workers, i.e., one per decomposition). Starting from an execution time of 4,218 seconds to perform 1,000 subgradient iterations per multiplier, we reduce the runtime to 882 seconds with 5 workers and to 462 seconds with 10 workers. This corresponds to a speed-up of approximately $9.12\times$, indicating that the overhead induced by parallelization is negligible. We note that this parallelization opportunity is specific to the multiple-decomposition variant, where several independent multiplier optimizations must be carried out. In particular, it provides marginal benefit for the quadratic portfolio problem in which the multiple decomposition setting is not used. Finally, these results show that our approach is well suited to scaling decision-focused learning to large-scale problems, provided they admit a Lagrangian decomposition, as is the case for the multi-dimensional knapsack and the quadratic portfolio problem.

6 Conclusion

Decision-focused learning is a promising paradigm for addressing predict-then-optimize problems. However, its practical deployment is often hindered by high computational costs, as it requires solving a constrained optimization problem for each training instance at every iteration. This significantly limits its ability to scale to large combinatorial problems. In this paper, we introduced a new methodology based on Lagrangian decomposition to substantially improve the scalability of DFL algorithms. We proposed a new surrogate objective and two associated loss functions for training the prediction model. Experiments on large-scale instances, up to eight times larger than those commonly considered in the literature, demonstrate that our approach can be seamlessly integrated into standard DFL methods, leading to improved performance. Scalability can be further enhanced through parallelization. As future work, we plan to extend our approach to other combinatorial optimization problems in more realistic settings, such as shortest path and shift scheduling. Another promising direction is to investigate alternative strategies for updating the Lagrangian multipliers during training. Finally, establishing a theoretical relationship between the regret on the decomposed problem and the accuracy on the original primal problem is also an interesting direction to pursue.

A Portfolio: Forms of the Main Subproblem

The quadratic portfolio problem involves two constraints: the linear budget constraint $\mathbf{1}^\top \mathbf{x} = 1$ and the quadratic risk constraint $\mathbf{x}^\top \boldsymbol{\Sigma} \mathbf{x} \leq \gamma \bar{\Sigma}$. In our Lagrangian-decomposition framework, we may choose either constraint to define the main subproblem. We detail the two resulting cases below.

Linear budget constraint as main constraint. If the budget constraint is selected, the main subproblem reads

$$\max_{\mathbf{X}_1 \geq 0} (\hat{\mathbf{c}} + \boldsymbol{\mu_2})^\top \mathbf{X}_1 \quad \text{s.t.} \quad \mathbf{1}^\top \mathbf{X}_1 = 1. \tag{18}$$

This subproblem admits a closed-form solution: it places all the mass on an asset attaining $\arg\max_i(\hat{c}_i + \mu_{2,i})$ (and zero elsewhere). As a result, it is computationally trivial and does not capture the quadratic risk structure of the original formulation, which makes it less informative for training.

Quadratic risk constraint as main constraint. If instead the quadratic constraint is selected, the main subproblem becomes

$$\max_{\mathbf{X}_1 \geq 0} (\hat{\mathbf{c}} + \boldsymbol{\mu_2})^\top \mathbf{X}_1 \quad \text{s.t.} \quad \mathbf{X}_1^\top \boldsymbol{\Sigma} \mathbf{X}_1 \leq \gamma \bar{\Sigma}. \tag{19}$$

Unlike the previous case, this formulation retains the covariance matrix $\boldsymbol{\Sigma}$ in the optimization layer, and is therefore more faithful to the structure of the original quadratic portfolio problem.

B The *Exact* Method

In the Lagrangian Decomposition, we choose the quadratic constraint in the main sub-problem. Thus, this sub-problem[1] is

$$\max_{\mathbf{X}_1 \geq 0} \big(\mathbf{c} + \boldsymbol{\mu_2}\big)^\top \mathbf{X}_1 \quad \text{s.t.} \quad \mathbf{X}_1^\top \Sigma \mathbf{X}_1 \leq \gamma \bar{\Sigma} \tag{20}$$

Analysis of another problem. We consider the problem

$$\max_{\mathbf{w} \in \mathbb{R}^n} \mathbf{c}^\top \mathbf{w} \quad \text{s.t.} \quad \mathbf{w}^\top \Sigma \mathbf{w} - \gamma \bar{\Sigma} \leq 0 \tag{21}$$

We define $J(\mathbf{w}) = \mathbf{c}^\top \mathbf{w}$, $F(\mathbf{w}) = \mathbf{w}^\top \Sigma \mathbf{w} - \gamma \bar{\Sigma}$ and K the space of feasible solutions.

We note that the J and F are continuous and convex with respect to $\mathbf{w}$. Moreover, $F'(\mathbf{w})$ is nonzero for all $\mathbf{w} \in K$ with $\mathbf{w} \neq \mathbf{0}$, which ensures that the constraint qualification condition is satisfied.

We can thus apply the Kuhn and Tucker Theorem. For any nonzero $\mathbf{w} \in K$, $\mathbf{w}$ is a global maximizer of J over K if and only if

$$\exists p \geq 0\,, F(\mathbf{w}) \leq 0\,, p \cdot F(\mathbf{w}) = 0\,, \mathbf{J}'(\mathbf{w}) - p\mathbf{F}'(\mathbf{w}) = \mathbf{0} \tag{22}$$

We have

$$\begin{aligned} (22) \iff & \exists p \geq 0\,, \quad F(\mathbf{w}) \leq 0, \quad p \cdot F(\mathbf{w}) = 0, \\ & \mathbf{c} - 2p\Sigma\mathbf{w} = \mathbf{0} \end{aligned}$$

Since $\mathbf{c} \neq \mathbf{0}$, we must have $p \neq 0$. It follows that

$$\begin{aligned} (22) \iff & \exists p \geq 0\,, \quad F(\mathbf{w}) \leq 0, \quad F(\mathbf{w}) = 0, \\ & \mathbf{c} - 2p\,\Sigma\mathbf{w} = \mathbf{0} \\ \iff & \exists p \geq 0\,, \quad F(\mathbf{w}) \leq 0, \quad \mathbf{w}^\top \Sigma \mathbf{w} = \gamma\,\bar{\Sigma}, \\ & \mathbf{w} = \frac{1}{2p}\,\Sigma^{-1}\mathbf{c} \\ \iff & \exists p \geq 0\,, F(\mathbf{w}) \leq 0, \frac{1}{4p^2}\left(\Sigma^{-1}\mathbf{c}\right)^\top \Sigma\Sigma^{-1}\mathbf{c} = \gamma\bar{\Sigma}, \\ & \mathbf{w} = \frac{1}{2p}\,\Sigma^{-1}\mathbf{c} \\ \iff & \exists p \geq 0\,, \quad F(\mathbf{w}) \leq 0, \quad p = \frac{1}{2}\sqrt{\frac{\mathbf{c}^\top\Sigma^{-1}\mathbf{c}}{\gamma\bar{\Sigma}}}, \\ & \mathbf{w} = \sqrt{\frac{\gamma\bar{\Sigma}}{\mathbf{c}^\top\Sigma^{-1}\mathbf{c}}}\,\Sigma^{-1}\mathbf{c} \end{aligned}$$

We conclude that the optimal solution of (21) is $\mathbf{w}^* = \sqrt{\frac{\gamma\bar{\Sigma}}{\mathbf{c}^\top\Sigma^{-1}\mathbf{c}}}\,\Sigma^{-1}\mathbf{c}$.

Back to our sub-problem. Once the constraint $\mathbf{X}_1 \geq \mathbf{0}$ is removed, the solution to

$$\max_{\bar{\mathbf{X}}_1 \in \mathbb{R}^n} \big(\mathbf{c} + \boldsymbol{\mu_2}\big)^\top \bar{\mathbf{X}}_1 \quad \text{s.t.} \quad \bar{\mathbf{X}}_1^\top \Sigma \bar{\mathbf{X}}_1 \leq \gamma\bar{\Sigma} \tag{23}$$

is given by

$$\bar{\mathbf{X}}_1^*(\mathbf{c}, \mu) = \sqrt{\frac{\gamma\bar{\Sigma}}{\big(\mathbf{c} + \boldsymbol{\mu_2}\big)^\top \Sigma^{-1} \big(\mathbf{c} + \boldsymbol{\mu_2}\big)}}\,\Sigma^{-1}\big(\mathbf{c} + \boldsymbol{\mu_2}\big) \tag{24}$$

We observe that the closed-form solution is differentiable with respect to $\mathbf{c}$, and its gradient can be obtained directly via PyTorch's automatic differentiation. These findings motivate a problem-specific differentiation scheme for the portfolio model. During training we replace the solver-provided solution $\mathbf{X}_1^*(\mathbf{c}, \mu)$ with the analytical expression $\bar{\mathbf{X}}_1^*(\mathbf{c}, \mu)$. This eliminates the need for a solver and a generic DFL technique.

[1] Since there are only two constraints in the Portfolio problem, we have $\mu = (\boldsymbol{\mu_2})$.

Acknowledgments

This research has been supported by the CRM and the Simons Foundation, thematic program Combinatorial Optimization and Data Science. It has been partly funded by the European Research Council (ERC) under the EU Horizon 2020 research and innovation program (Grant No 101002802, CHAT-Opt) and by the Canada Research Chairs Program in Healthcare Analytics. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC RGPIN-2025-04995 and RGPIN-2022-03964).

References

[Abbas and Swoboda, 2024] Ahmed Abbas and Paul Swoboda. Doge-train: Discrete optimization on gpu with end-to-end training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20623–20631, 2024.

[Amos and Kolter, 2017] Brandon Amos and J. Zico Kolter. OptNet: differentiable optimization as a layer in neural networks. *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pages 136–145, 2017.

[Berden *et al.*, 2025] Senne Berden, Ali İrfan Mahmutoğulları, Dimos Tsouros, and Tias Guns. Solver-free decision-focused learning for linear optimization problems. *arXiv preprint arXiv:2505.22224*, 2025.

[Bessa *et al.*, 2025] Swann Bessa, Darius Dabert, Max Bourgeat, Louis-Martin Rousseau, and Quentin Cappart. Learning valid dual bounds in constraint programming: Boosted lagrangian decomposition with self-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 11113–11121, 2025.

[Defresne *et al.*, 2023] Marianne Defresne, Sophie Barbe, and Thomas Schiex. Scalable coupling of deep learning with logical reasoning, 2023.

[Elmachtoub and Grigas, 2021] Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Manage. Sci.*, 68(1):9–26, January 2021.

[Elmachtoub *et al.*, 2023] Adam N. Elmachtoub, Henry Lam, Haofeng Zhang, and Yunfan Zhao. Estimate-then-optimize versus integrated-estimation-optimization versus sample average approximation: A stochastic dominance perspective, 2023. _eprint: 2304.06833.

[Gould *et al.*, 2016] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016.

[Guignard and Kim, 1987] Monique Guignard and Siwhan Kim. Lagrangean decomposition: A model yielding stronger lagrangean bounds. *Mathematical programming*, 39(2):215–228, 1987.

[Gurobi Optimization, LLC, 2024] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[Hà *et al.*, 2015] Minh Hoàng Hà, Claude-Guy Quimper, and Louis-Martin Rousseau. General bounding mechanism for constraint programs. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 158–172. Springer International Publishing, 2015.

[Mandi *et al.*, 2020] Jayanta Mandi, Peter J Stuckey, Tias Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1603–1610, 2020.

[Mandi *et al.*, 2024] Jayanta Mandi, James Kotary, Senne Berden, Maxime Mulamba, Victor Bucarey, Tias Guns, and Ferdinando Fioretto. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 80:1623–1701, 2024.

[Niepert *et al.*, 2021] Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: backpropagating through discrete exponential family distributions. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21. Curran Associates Inc., 2021.

[Parjadis *et al.*, 2024] Augustin Parjadis, Quentin Cappart, Bistra Dilkina, Aaron Ferber, and Louis-Martin Rousseau. Learning lagrangian multipliers for the travelling salesman problem. In *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*, pages 22–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[Shor, 2012] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer Science & Business Media, 2012.

[Tang and Khalil, 2024a] Bo Tang and Elias B. Khalil. CaVE: A cone-aligned approach for fast predict-then-optimize with binary linear programs. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 193–210, Cham, 2024. Springer Nature Switzerland.

[Tang and Khalil, 2024b] Bo Tang and Elias B Khalil. PyEPO: a PyTorch-based end-to-end predict-then-optimize library for linear and integer programming. *Mathematical Programming Computation*, 16(3):297–335, 2024.