COMBINING ANALYTICAL SMOOTHING WITH SURROGATE LOSSES FOR IMPROVED DECISION-FOCUSED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Many combinatorial optimization problems in routing, scheduling, and assignment involve parameters such as price or travel time that must be predicted from data; so-called predict-then-optimize (PtO) problems. Decision-focused learning (DFL) is a family of successful end-to-end techniques for PtO that trains machine learning models to minimize the error of the downstream optimization problems. For each instance, this requires computing the derivative of the optimization problem's solution with respect to the predicted input parameters. Previous works in DFL employ two main approaches when the parameters appear linearly in the objective: (a) using a differentiable surrogate loss instead of regret; or (b) turning the combinatorial optimization problem into a differentiable mapping by smoothing the optimization to a quadratic program or other smooth convex optimization problem and minimizing the regret of that. We argue that while smoothing makes the optimization differentiable, for a large part, the derivative remains approximately zero almost everywhere, with highly non-zero values near the transition points. To address this plateau effect, we propose minimizing a *surrogate* loss even after smoothing. We experimentally demonstrate the advantage of minimizing surrogate losses instead of the regret after smoothing across a series of problems. Furthermore, we show that by minimizing a surrogate loss, a recently developed fast, fully neural optimization layer matches state-of-the-art performance while dramatically reducing training time up to five-fold. Thus, our paper opens new avenues for efficient and scalable DFL techniques.

1 INTRODUCTION

Many decision-making problems in real-world can be cast as optimization problems. Some parameters of these optimization problems are often unknown due to uncertainty or the anticipation of future events. As prediction of these parameters is crucial for making high-quality decisions, leveraging contextual information is important at prediction time. The availability of historical data, combined with the rapid growth of predictive machine learning (ML), has fueled increasing interest in data-driven contextual optimization (Sadana et al., 2025).

When the goal is to predict parameters (such as cost or travel time) of an optimization problem, such problems can be viewed as "predict-then-optimize"(PtO) problems, including two key steps—the prediction of the unknown parameters and the subsequent optimization using those predicted parameters. Prediction-focused learning is the approach to tackle PtO problems by treating the prediction step independent of the optimization step, based on the assumption that increasing accuracy of predictions would lead to good quality decisions. However, in practice, ML models fail to achieve 100% accuracy, and in the presence of prediction errors, such a prediction-focused approach fails to consider how the error in predictions impacts the solution to the optimization problem. This fact motivates the research in decision-focused learning (DFL), as surveyed by Mandi et al. (2024).

DFL trains ML models to predict the uncertain parameters by *directly* minimizing the task loss, which reflects the quality of the solutions made using the predicted parameters. Gradient-based DFL entails computing the derivative of the optimization problem's solution with respect to the predicted parameters. However, for combinatorial optimization problems, this derivative is almost always zero because slight parameter changes typically do not alter the solution, except at certain transition points where the derivative does not exist. In this paper, we focus on predicting parameters of combinatorial optimization problems, where the predicted parameters appear linearly in the objective function. Previous works in DFL use two broad categories of approaches: (a) turning the combinatorial optimization problem into a differentiable mapping by smoothing the optimization to a convex optimization problem (Wilder et al., 2019; Mandi & Guns, 2020), and then minimizing the task loss, and (b) using surrogate loss functions (Elmachtoub & Grigas, 2022; Mulamba et al., 2021; Mandi et al., 2022), for which gradients or subgradients exist.

The existing DFL literature views these two approaches separately. Consequently, minimizing the task loss of the smoothed problem is the standard approach in category (a). However, while smoothing makes the optimization differentiable, excessive smoothing can cause the solution to the "smoothed" problem to deviate significantly from the original solution. In practice, the smoothing strength is kept reasonably low to ensure that it does not overshadow the true objective of the original optimization problem. We argue that, with a moderate level of smoothing, the derivative remains nearly zero in most regions, becoming highly non-zero only at transition points. For this reason, we propose to minimize the surrogate loss, even though it is possible to minimize regret directly by differentiating through the smoothed optimization problem. We justify the advantage of using a surrogate loss by comparing the pattern of the gradient landscape with respect to regret and the surrogate loss. In this way, this paper combines the two approaches of DFL. This allows us to accelerate DFL by minimizing surrogate loss using a fast differentiable optimization layer.

In summary, this paper makes the following contributions:

- To address the plateau effect which occurs even after smoothing, we combine the two families of DFL approaches by minimizing a surrogate loss post-smoothing.
- We empirically demonstrate that for smoothing approaches, minimizing surrogate losses results in lower regret on test data than minimizing the regret. This highlights the benefit of minimizing the surrogate loss even when the optimization problem is smoothed.
- To improve the scalability of DFL, McKenzie et al. (2024) recently developed a fast, fully differentiable neural optimization layer for linear programs (LPs). We demonstrate that minimizing the surrogate loss using this optimization layer achieves regret comparable to existing state-of-the-art methods while reducing training time by up to five-fold.

2 Predict-then-Optimize Problem Description

In PtO problems, decisions are made by solving constrained optimization (CO) problems. In this work, we focus on CO problems with linear objectives and the prediction of objective function parameters. These CO problems can be formulated as LPs or integer LPs (ILPs), both of which have extensive practical applications. Any LP can be transformed in the following standard LP form:

$$\boldsymbol{v}^\star(\boldsymbol{y}) = \arg\min_{\boldsymbol{v}} \boldsymbol{y}^\top \boldsymbol{v} \;\; \text{s.t.} \;\; A\boldsymbol{v} = \mathbf{b}; \;\; \boldsymbol{v} \geq \mathbf{0} \tag{1}$$

where $\boldsymbol{v} \in \mathbb{R}^K$ is a decision variable and $\boldsymbol{v}^\star(\boldsymbol{y})$ is the optimal solution for a given cost parameter $\boldsymbol{y} \in \mathbb{R}^K$. ILPs differ from LPs in that the decision variables $\boldsymbol{v}$ are restricted to integer values. For brevity, we use $\mathcal{F}$ to denote the feasible space. So, for the standard LP formulation, $\mathcal{F} = \{\boldsymbol{v} \in \mathbb{R}^K | A\boldsymbol{v} = \mathbf{b} \,; \boldsymbol{v} \geq \mathbf{0}\}$. Unless it is explicitly stated otherwise, $\boldsymbol{v}^\star$ will denote $\boldsymbol{v}^\star(\boldsymbol{y})$.

To account for uncertainty in the decision-making, PtO problems comprise two steps—the prediction of the unknown parameters and solving the optimization problem using the predicted parameters. We consider PtO formulation, where the vector of cost parameters $\boldsymbol{y}$ is not known prior to solving. Instead, a list of contextual information $\boldsymbol{\phi}$, correlated with $\boldsymbol{y}$ is available for predicting $\boldsymbol{y}$. In PtO problems, an ML model $\mathcal{M}_\omega$ (with trainable parameters ω) is trained to map $\boldsymbol{\phi} \to \boldsymbol{y}$ using past observation pairs $\{(\boldsymbol{\phi}_i, \boldsymbol{y}_i)\}_{i=1}^N$. Given their success in predictive tasks, neural networks have become the preferred choice for the predictive modeling task in PtO problems.

A straightforward approach to the PtO problem is to train $\mathcal{M}_\omega$ to generate accurate parameter predictions $\hat{\boldsymbol{y}} = \mathcal{M}_\omega(\boldsymbol{\phi})$ by minimizing the prediction errors with respect to ground-truth $\boldsymbol{y}$. Previous works (Wilder et al., 2019; Elmachtoub & Grigas, 2022; Mandi et al., 2020) justify why such a *prediction-focused approach* produces suboptimal performance. By contrast, in *decision-focused* *learning* (DFL), the ML model is directly trained to optimize the task loss, the quality of the resulting decisions. When only the parameters in the objective function are predicted, the task loss of interest is typically *regret*, which measures the suboptimality of a decision resulting from prediction errors. The regret for making the decisions $\boldsymbol{v}$ under the true realization $\boldsymbol{y}$ can be expressed in the following form:

$$Regret(\boldsymbol{v}, \boldsymbol{y}) = \boldsymbol{y}^\top \boldsymbol{v} - \boldsymbol{y}^\top \boldsymbol{v}^\star(\boldsymbol{y}) \tag{2}$$

In PtO problems, one can consider other task losses, such as squared decision errors (SqDE) between $\boldsymbol{v}^\star(\boldsymbol{y})$ and $\boldsymbol{v}^\star(\hat{\boldsymbol{y}})$, i.e., $SqDE = ||\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})||^2$.

3 Decision-Focused Learning for Combinatorial Optimization

The DFL approach trains $\mathcal{M}_\omega$ to directly minimize $\frac{1}{N}\sum_{i=1}^{N} Regret(\boldsymbol{v}^*(\mathcal{M}_\omega(\boldsymbol{\phi}_i)), \boldsymbol{y}_i)$, the empirical risk minimization counterpart of $\mathbb{E}[Regret(\boldsymbol{v}^*(\mathcal{M}_\omega(\boldsymbol{\phi})), \boldsymbol{y})]$ since the true distribution is unknown. This minimization of regret in gradient descent-based learning requires backpropagation through the CO problem, which involves computing the derivative of $\boldsymbol{v}^\star(\hat{\boldsymbol{y}})$ with respect to $\hat{\boldsymbol{y}} = \mathcal{M}_\omega(\boldsymbol{\phi})$. While $\frac{d\boldsymbol{v}^\star(\hat{\boldsymbol{y}})}{d\hat{\boldsymbol{y}}}$ can be computed for convex optimization problems through implicit differentiation (Agrawal et al., 2019; Amos & Kolter, 2017), it is more challenging when the optimization problem is combinatorial. This is because when the parameters of a CO problem change, the solution either remains unchanged or shifts abruptly, meaning the derivatives are almost always zero and undefined at abrupt changes.

Broadly there are two approaches of implementing DFL for CO problems: (a) smoothing the CO to a smooth convex optimization problem and (b) using a surrogate loss that is differentiable. For a detailed discussion on how existing DFL techniques tackle this challenge, we refer readers to the survey paper by Mandi et al. (2024).

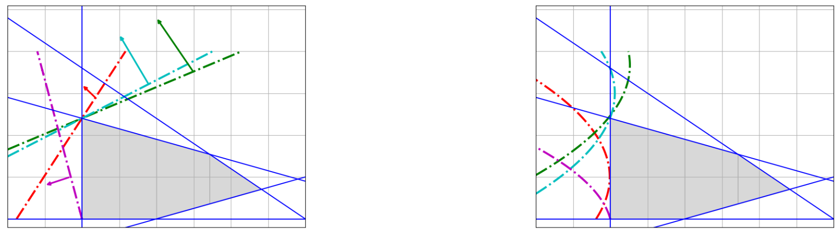
3.1 Differentiable Optimization by Smoothing of Combinatorial Optimization

Differentiable optimization represents an optimization problem as a differentiable mapping from its parameters to its solution. Since for a combinatorial problem, this mapping is **not** differentiable, one prominent research direction in DFL involves smoothing the combinatorial optimization problem into a differentiable optimization problem. We particularly focus on smoothing by regularization. There exists another from of smoothing—smoothing by perturbation, as proposed by Pogančić et al. (2020); Blondel et al. (2020); Niepert et al. (2021); Sahoo et al. (2023). In this work, we focus on optimization problems with linear objective functions such as LPs and ILPs. For an LP, the solution will always lie in one of the vertices of the LP simplex. So, the LP solution remains unchanged as long as the cost vector changes while staying within the corresponding normal cone (Boyd & Vandenberghe, 2004). However, the solution will suddenly switch to a different vertex if the cost vector slightly moves outside the normal cone, as illustrated in Figure 1a. Because the solution abruptly jumps between the vertices, the LP solution is not a differentiable function of the cost vector.

To address this, methodologies under analytical smoothing first modify the optimization problem and then analytically differentiate the modified optimization problem. For LPs, Wilder et al. (2019) propose transforming the LPs into 'smoothed' quadratic programs (QPs) by augmenting the objective function with the square of the Euclidean norm of the decision variables in the following form:

$$\min_{\boldsymbol{v}} \hat{\boldsymbol{y}}^\top \boldsymbol{v} + \mu \|\boldsymbol{v}\|_2^2 \ \text{ s.t. } \ A\boldsymbol{v} = \mathbf{b}\,; \boldsymbol{v} \geq \mathbf{0} \tag{3}$$

where $\mu \geq 0$ is the smoothing parameter, controlling the strength of smoothing. After smoothing, the solution is not restricted to being at a vertex of the LP polyhedron. In the 'smoothed' problem, unlike the original LP, the solution do not change abruptly. The solution either may not change or change smoothly with the change of the cost vector, as illustrated in Figure 1b. Consequently, $\boldsymbol{v}^\star(\hat{\boldsymbol{y}})$ becomes differentiable with respect to $\hat{\boldsymbol{y}}$. The QP smoothing approach has been applied in various DFL works (Ferber et al., 2020; 2023; McKenzie et al., 2024). Mandi & Guns (2020) consider another form of smoothing by adding logarithm barrier term into the LP. When the underlying optimization problem is an ILP, smoothing of the LP, resulting from the continuous relaxation of the ILP is carried out.



(a) LP Solutions (b) Solutions to the QP smoothing

Figure 1: Schematic diagram showing the effect of QP smoothing. (a) LP solutions and the corresponding isocost line for four cost vectors. The green, cyan and red cost vectors result in the same solution, the top vertex, highlighting that a slight rotation of the isocost lines may not alter the LP solution. However, if the isocost lines rotate too much, for example, the violet line, the solution suddenly shifts to a different vertex. (b) The isocost lines change after applying QP smoothing, and the solution is no longer restricted to a vertex. For example, the red vector results in a smooth change in the solution. However, even with smoothing, some cost vectors, like the cyan and green, may still share the same solution.

3.2 Surrogate Losses for DFL

Surrogate loss functions are used for training in DFL because they are crafted to have non-zero (sub)gradients everywhere while also directly correlating with the task loss—as regret decreases, surrogate loss functions decrease as well. We focus on two surrogate loss functions, used widely in DFL.

3.2.1 Smart Predict then Optimize(SPO)

The SPO+ loss (Elmachtoub & Grigas, 2022), a convex upper bound of $Regret(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})$, is one of the first and most widely used surrogate losses for linear objective optimization problems.

$$\begin{aligned} &Regret(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = \boldsymbol{y}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{y}^\top \boldsymbol{v}^\star = \boldsymbol{y}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - 2\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) + 2\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{y}^\top \boldsymbol{v}^\star \\ &\leq \max_{\boldsymbol{v}' \in \mathcal{F}} \{\boldsymbol{y}^\top \boldsymbol{v}' - 2\hat{\boldsymbol{y}}^\top \boldsymbol{v}'\} + 2\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{y}^\top \boldsymbol{v}^\star \leq \underbrace{\max_{\boldsymbol{v}' \in \mathcal{F}} \{\boldsymbol{y}^\top \boldsymbol{v}' - 2\hat{\boldsymbol{y}}^\top \boldsymbol{v}'\} + 2\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star - \boldsymbol{y}^\top \boldsymbol{v}^\star}_{\mathcal{L}_{SPO+}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})} \end{aligned}$$

Instead of minimizing $Regret$, they propose to minimize this convex upperbound, which is called $\mathcal{L}_{SPO+}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})$ loss. It can be expressed in the following form:

$$\begin{aligned} \mathcal{L}_{SPO+}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) &= \max_{\boldsymbol{v}' \in \mathcal{F}} \{2\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star - \boldsymbol{y}^\top \boldsymbol{v}^\star - (2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}'\} = (2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star - \min_{\boldsymbol{v}' \in \mathcal{F}} \{(2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}'\} \\ &= (2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star - (2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star(2\hat{\boldsymbol{y}} - \boldsymbol{y}) \end{aligned} \quad (4)$$

They also propose the following sub-gradient for gradient-based training using any solver of choice:

$$\nabla_{\mathcal{L}_{SPO+}} = 2\Big(\boldsymbol{v}^\star - \boldsymbol{v}^\star(2\hat{\boldsymbol{y}} - \boldsymbol{y})\Big) \quad (5)$$

3.2.2 Contrastive Loss

Mulamba et al. (2021) propose a surrogate loss based on noise-contrastive estimation (NCE) (Gutmann & Hyvärinen, 2012). The loss is derived by considering the log-likelihood ratio between $\boldsymbol{v}^\star$ and other feasible points $\boldsymbol{v}'$. By maximizing this likelihood, they propose to minimize the following NCE loss:

$$\mathcal{L}_{NCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = \max_{\boldsymbol{v}' \in \mathcal{F}} \{\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star - \hat{\boldsymbol{y}}^\top \boldsymbol{v}'\} = \hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star - \min_{\boldsymbol{v}' \in \mathcal{F}} \{\hat{\boldsymbol{y}}^\top \boldsymbol{v}'\} = \hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star - \hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) \quad (6)$$

Note that $\mathcal{L}_{NCE}$ is similar to $\mathcal{L}_{SPO+}$, except that in $\mathcal{L}_{NCE}$, $2\hat{\boldsymbol{y}} - \boldsymbol{y}$ is replaced with $\hat{\boldsymbol{y}}$. This introduces a shortcoming in $\mathcal{L}_{NCE}$. The minimum of $\mathcal{L}_{NCE}$, which is zero, can be achieved either when $\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) = \boldsymbol{v}^\star$ or by predicting $\hat{\boldsymbol{y}} = 0$. To prevent minimizing $\mathcal{L}_{NCE}$ by predicting $\hat{\boldsymbol{y}} = 0$, Mulamba et al. (2021) further modify $\mathcal{L}_{NCE}$ to derive the self-contrastive estimation (SCE) loss:

$$\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = (\hat{\boldsymbol{y}}-\boldsymbol{y})^\top \boldsymbol{v}^\star - (\hat{\boldsymbol{y}}-\boldsymbol{y})^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) = \hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star - \hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) + \boldsymbol{y}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{y}^\top \boldsymbol{v}^\star(\boldsymbol{y}) \tag{7}$$

Proposition 1. *$\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})$ has the following properties (proof is given in Appendix A):*

1. *$\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) \geq 0$*
2. *When the set of optimal solutions is a singleton*
   $\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0 \implies Regret(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0$;
   $Regret(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0 \implies \mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0$.

When $\mathcal{L}_{SCE}$ is minimized using a blackbox optimization solver, the gradient would be:

$$\nabla_{\mathcal{L}_{SCE}} = \boldsymbol{v}^\star - \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) \tag{8}$$

4 Minimizing Surrogate Loss with a Smoothed Solver

Since smoothing converts the non-smooth combinatorial problem into a smooth optimization problem, existing approaches minimize regret during training. The intuition is that this reduces expected regret in unseen instances, aligning with the empirical risk minimization paradigm in ML. However, a close inspection of how the incorporation of smoothing changes the gradient landscape reveals a shortcoming in this approach.

The introduction of smoothing ensures that the solution transitions smoothly, rather than abruptly, near the original optimization problem's transition points. However, the solution of the smoothed optimization remains unchanged, or changes very slowly, in regions where the original problem's solution is constant, provided the smoothing strength is kept low as illustrated in Figure 1b. So in this region, $\frac{d\boldsymbol{v}^\star(\hat{\boldsymbol{y}})}{d\hat{\boldsymbol{y}}}$ is are nearly zero. When regret is minimized, the derivative of it with respect to the $\hat{\boldsymbol{y}}$ takes the following form:

$$\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}} \boldsymbol{y} \tag{9}$$

where $\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}}$ is computed by considering the smoothed optimization problem. As we illustrated above, smoothing addresses the non-differentiability at the transition points, but the derivative $\frac{d\boldsymbol{v}^\star(\hat{\boldsymbol{y}})}{d\hat{\boldsymbol{y}}}$ still remains zero far from these points. Hence, the derivative in Eq. 9 remains zero. This would also be true if $SqDE$ is considered as the training loss. In this case, the derivative would be:

$$\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}} (\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{v}^\star) \tag{10}$$

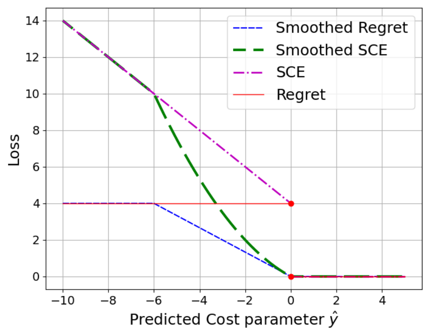
In both Eq. 10 and Eq. 9, the derivative turns zero due to $\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}}$ becoming zero. Consequently, training by gradient descent would fail to change $\hat{\boldsymbol{y}}$ despite $\hat{\boldsymbol{y}}$ resulting non-zero regret.

To prevent the derivative from vanishing far from the transition points, in this paper, we argue in favour of minimizing a surrogate loss such as noise contrastive loss when the smoothed optimization problem is considered. For instance, when $\mathcal{L}_{SPO+}$ is minimized, the derivative of the loss with respect to the $\hat{\boldsymbol{y}}$ takes the following form:

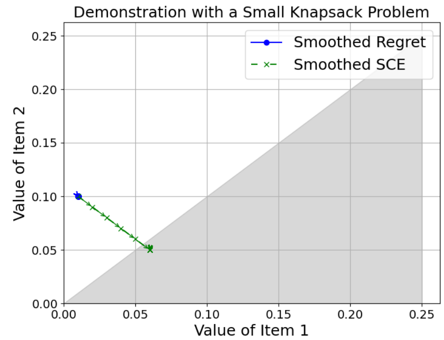
$$2\Big(\boldsymbol{v}^\star - \boldsymbol{v}^\star(2\hat{\boldsymbol{y}} - \boldsymbol{y})\Big) + 2\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=2\hat{\boldsymbol{y}}-\boldsymbol{y}} (\boldsymbol{y} - 2\hat{\boldsymbol{y}}) \tag{11}$$

Similarly if $\mathcal{L}_{SCE}$ is minimized after smoothing, the resulting derivative would be:

$$(\boldsymbol{v}^\star - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})) + \frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}} (\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{12}$$



(a)



(b)

Figure 2: (a) The numerical illustration demonstrates that while smoothing removes abrupt changes in the solution and makes the regret continuous, the solution often remains flat across most regions, resulting in a zero gradient, not suitable for training. In contrast, $\mathcal{L}_{SCE}$ (with or without smoothing) provides a more responsive landscape: when regret is non-zero, $\mathcal{L}_{SCE}$ ensures non-zero gradient. (b) Progression of predictions by epochs when the smoothed regret and SCE are used as training losses.

The way Eq. 12 differs from Eq. 9 is the term $(\boldsymbol{v}^\star - \boldsymbol{v}^\star(\hat{\boldsymbol{y}}))$ and the multiplier of $\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}\Big|_{\boldsymbol{y}=\hat{\boldsymbol{y}}}$ is $(\boldsymbol{y} - \hat{\boldsymbol{y}})$ instead of $\boldsymbol{y}$. The term $(\boldsymbol{v}^\star - \boldsymbol{v}^\star(\hat{\boldsymbol{y}}))$ prevents $\frac{d\mathcal{L}}{d\hat{\boldsymbol{y}}}$ going to zero even when $\frac{d\boldsymbol{v}^\star(\hat{\boldsymbol{y}})}{d\hat{\boldsymbol{y}}} \approx 0$.

Note that if we minimize $\mathcal{L}_{SCE}$ or $\mathcal{L}_{SPO+}$ using a blackbox optimization solver, $\frac{\partial \boldsymbol{v}^\star(\boldsymbol{y})}{\partial \boldsymbol{y}}$ cannot be computed and only the first part of the derivative would be used. So, in this case, Eq. 11 and Eq. 12 would reduce to Eq. 5 and Eq. 8 respectively.

A deep dive into the gradient landscape. To convince readers that the solution of the smoothed optimization remains unchanged, we will demonstrate how the gradient landscape changes after QP smoothing with a simple illustration. For this, we consider the following one-dimensional optimization problem:

$$\min_v yv \quad \text{s.t. } 0 \leq v \leq 1 \tag{13}$$

where $y \in \mathbb{R}$ is the parameter to be predicted. Note that the solution of this problem is: $v^\star(y) = 1$ if $y < 0$ and $v^\star(y) = 0$ if $y > 0$. When $y = 0$ any value in the interval [0,1] is an optimal solution.

Let us assume that the true value of y is 4 and hence $v^\star(y) = 0$. The red line in Figure 2a shows how the value of regret changes as $\hat{y}$ changes. The regret is 4 when $\hat{y} \leq 0$ and 0 when $\hat{y} > 0$. The regret changes abruptly at the point $\hat{y} = 0$. After augmenting the objective with the quadratic smoothing term $\frac{\mu}{2}v^2$ with $\mu > 0$, the solution of the smoothed problem is:

$$v^\star(y) = \begin{cases} 0; & \text{when } y > 0 \\ -\frac{y}{\mu}; & \text{when } -\mu < y \leq 0 \\ 1; & \text{when } y \leq -\mu \end{cases}$$

This makes the derivative non-zero in the interval $-\mu \leq y \leq 0$. However, it is still zero when $y < -\mu$. Hence, if $\hat{y} < -\mu$, *the derivative of regret is* 0*, even if regret is non-zero.* Consequently, the predictions cannot be changed by gradient descent despite regret being zero. The regret with the smoothed problem is shown by the blue line in Figure 2a for $\mu = 6$. The strength of smoothing can be increased by assigning μ to a high value. However, if $\mu \gg |y|$, $v^\star(y) \approx 0$ almost everywhere. On the other hand, $\mathcal{L}_{SCE}$ with and without smoothing are plotted with green and violet colors, respectively. In both cases, $\mathcal{L}_{SCE}$ is strictly decreasing for $\hat{y} < 0$. This ensures a non-zero derivative, suitable for guiding $\hat{y}$ towards the positive half-space if $\hat{y} < 0$.

Example. We further illustrate this with a simple fractional knapsack problem, which is an LP. Let us consider that we have two items and space for only one item. This can be formulated as a minimization problem:

$$\min \ -y_1v_1 - y_2v_2 \ \text{ s.t. } \ v_1 + v_2 \le 1; \ \ v_1, v_2 \ge 0$$

Let us assume the true values of y_1 and y_2 are $(0.8, 0.4)$. The corresponding solution is $(v_1, v_2) = (1, 0)$. The grey region in Figure 2b corresponds to any predictions satisfying $\hat{y}_1 > \hat{y}_2$. Such predictions will induce the true solution, resulting in zero regret. Further assume that the initial predictions are $(\hat{y}_1, \hat{y}_2) = (0.1, 0.01)$. We show the progression of predictions by epochs when regret and SCE are used as training loss, using the smoothed optimization problem with blue and green lines, respectively in Figure 2b. The predictions does not change with training epochs when regret is used as the loss because the derivatives of regret with respect to $\hat{y}_1$ and $\hat{y}_2$ are zero. On the other hand, when $\mathcal{L}_{SCE}$ is used as the loss, $(\hat{y}_1, \hat{y}_2)$ gradually move from the white region to the grey region, eventually resulting in zero regret. Note that increasing the strength of smoothing may provide non-zero gradient across the space. But this will entirely alter the optimization problem's solution. For instance, in this knapsack example, high values of μ would make both v_1 and v_2 close to zero.

5 Addressing the Scalability of DFL

Implementing DFL entails a significant computational burden, as it requires solving and differentiating the CO problem for each training instance in every epoch using predicted parameters. While Mulamba et al. (2021) address this issue by using solution caching instead of repeatedly solving the optimization problem, a faster and more scalable implementation of the optimization problem is a promising direction, which has been receiving increasing attention recently. Research in this area is tangential to the learning-to-optimize paradigm (Bengio et al., 2021; Kotary et al., 2021), which trains an ML model to output CO solutions directly from the parameters. Recently, McKenzie et al. (2024) introduced a differentiable method, called DYS-Net, based on a three-operator splitting technique (Davis & Yin, 2017), to compute the solution of an LP. Next, we will provide a brief overview of DYS-Net, as we aim to train by minimizing a surrogate loss using DYS-Net for accelerating DFL.

DYS-Net for LPs. The motivation behind DYS-Net emerges from projected gradient descent (Duchi et al., 2008). Projected gradient descent differs from standard gradient descent in that, after each iteration, the current predictions, if not inside feasible space, is projected into the feasible space. However, projecting into the feasible space of a combinatorial optimization problem is itself an expensive operation. If we consider standard form LPs, the feasible space can be expressed as:

$$\mathcal{F} \equiv \mathcal{F}_1 \cap \mathcal{F}_2 \text{ where } \mathcal{F}_1 \doteq \{A\boldsymbol{v} = b\} \text{ and } \mathcal{F}_2 \doteq \{\boldsymbol{v} \ge 0\}.$$

Although projecting an infeasible solution $\boldsymbol{v}$ directly into $\mathcal{F}$ is not a trivial operation, projecting into $\mathcal{F}_1$ and $\mathcal{F}_2$ separately are much simpler tasks. Projecting into $\mathcal{F}_1$ takes the following form:

$$P_{\mathcal{F}_1}(\boldsymbol{v}) \doteq \boldsymbol{v} - A^{\dagger}(A\boldsymbol{v} - b)$$

where $A^{\dagger}$ is the pseudo inverse of A. Projecting into $\mathcal{F}_2$ takes the following form:

$$P_{\mathcal{F}_2}(\boldsymbol{v}) \doteq \max\{0, \boldsymbol{v}\}$$

where $\max$ operates element-wise. In order to obtain the LP solution to a given cost vector $\boldsymbol{y}$, McKenzie et al. (2024) propose the following fixed point iteration.

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k - P_{\mathcal{F}_2}(\boldsymbol{v}_k) + P_{\mathcal{F}_1}\Big((2 - \alpha\mu)P_{\mathcal{F}_2}(\boldsymbol{v}_k) - \boldsymbol{v}_k - \alpha\boldsymbol{y}\Big) \qquad \text{(DYS)}$$

which converges to $\boldsymbol{v}^\star(\boldsymbol{y})$ as $k \to \infty$. In practice, we use a finite number of iterations k in a single forward pass to get an approximation of $\boldsymbol{v}^\star(\boldsymbol{y})$. Note that all operations in Eq. DYS can be expressed as matrix operations and can be implemented using neural networks, which has the potential for greater scalability and reduced training time by leveraging recent advancements in GPU hardware.

We denote the solution obtained in this method as $DYS(\boldsymbol{y})$. To improve the scalability of DFL, McKenzie et al. (2024) use DYS-Net during training, minimizing SqDE between $\boldsymbol{v}^\star(\boldsymbol{y})$ and $DYS(\hat{\boldsymbol{y}})$. In this work, we instead propose training by minimizing $\mathcal{L}_{SCE}$ between the outputs of $DYS(\boldsymbol{y})$ and $\boldsymbol{v}^\star(\boldsymbol{y})$. The intuition of this is based on our discussion in the previous section.

6 Experimental Evaluation

In order to demonstrate the advantage of using $\mathcal{L}_{SCE}$ as the training loss, we conduct experiments on four well-established DFL benchmark problems.

Cubic Top-K (Top-K). This optimization problem is adopted from Shah et al. (2022). The optimization problem is to choose the best among N resources. Each resource is associated with feature $\phi_n \sim \mathcal{U}[-1, 1]$ and its true utility is defined by the cubic equation: $y_n = 10\phi_n^3 - 6.5\phi_n$. However, to predict the utility from the feature, a linear model is used.

Shortest path on a grid (SP). The goal of this optimization problem is to find the path, with lowest cost on a $k \times k$ grid, starting from the southwest node and ending at the northeast node of the grid (Elmachtoub & Grigas, 2022). The cost of each edge is unknown and should be predicted before solving the problem. The true relation between the features and the costs are non-linear, but linear model is used for predictions.

Multi-Dimensional Knapsack (KP). The objective of the knapsack problem is to select a subset of items with the highest total value, subject to a capacity constraint. The weights of the items and the knapsack's capacity are known, but the values of the items are unknown. Therefore, the prediction task is to predict the value of each item using features.

Travelling salesperson problem (TSP). Given a set of nodes, the goal is to find the tour, with the lowest cost, that visits every node exactly once. As before, the costs are related to the features in a non-linear manner, but a linear predictive model is used for prediction.

We use *PyEPO* (Tang & Khalil, 2023) to generate the training, validation and test instances for the SP, KP and TSP problems. In all three problems, the true relation between the features and the costs are non-linear, but linear model is used for predictions. We experiment with polynomial degree parameter and noise half-width parameter being 6 and 0.5, respectively. The predictive models are implemented using *PyTorch* (Paszke et al., 2019) and *Gurobipy* (Gurobi Optimization, 2021) is used as a blackbox combinatorial solver to obtain the optimal solution. To solve and differentiate through the smooth optimization problem after adding the quadratic regularizer, we use *CvxpyLayer* (Agrawal et al., 2019). We use the implementation of DYS-NET by McKenzie et al. (2024). The experiments were executed on a computer with an *Intel(R) Core(TM) i7-13800H* processor using 32 Gb of RAM.

6.1 Regret vs. Surrogate Loss with QP Smoothing

We report **normalized relative regret** on test data in Table 1, calculated as follows:

$$\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{\boldsymbol{y}_i^\top (\boldsymbol{v}^\star(\hat{\boldsymbol{y}}_i) - \boldsymbol{v}_i^\star)}{\boldsymbol{y}_i^\top \boldsymbol{v}_i^\star}. \tag{14}$$

For evaluation, we always use an exact combinatorial solver. The column MSE corresponds to ML models trained with the MSE loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$. As this approach does not consider the optimization problem during training, we anticipate it would have higher regret than the DFL approaches. Implementation of perturbed Fenchel-Young (PFY) (Berthet et al., 2020), $\mathcal{L}_{SPO^+}$ and $\mathcal{L}_{SCE}$ using *combinatorial solvers* serve as three DFL benchmarks. We choose $\mathcal{L}_{SPO^+}$ and PFY, as they are best performing DFL methods across various optimization problems (Mandi et al., 2024; Tang & Khalil, 2023). When $\mathcal{L}_{SPO^+}$ and $\mathcal{L}_{SCE}$ are minimized using combinatorial solvers, Eq. 5 and Eq. 8 are used for gradient backpropagation. The three columns under *CvxpyLayer* show regret when the losses are backpropagated through the smoothed QP problem using *CvxpyLayer*. $Regret$ appears only under *CvxpyLayer*, because it can only be minimized after QP smoothing. This paper is the first to test the last two approaches, which combine differential smoothing and surrogate losses.

For the Top-K problem, all DFL approaches have exact same regret. We explain this behaviour in the appendix. Next, we highlight that in all cases, *minimizing $\mathcal{L}_{SPO^+}$ or $\mathcal{L}_{SCE}$ results in lower test regret than minimizing $Regret$ using CvxpyLayer*, which corroborates the main proposal we made in this paper. Across all experiments, we observe that minimizing $\mathcal{L}_{SCE}$ using *CvxpyLayer* yields regret similar to $\mathcal{L}_{SPO^+}$ and PFY, which use combinatorial solvers. This shows that minimizing

Table 1: Normalized relative regret on test data for four optimization problems. We mention the number of resources, the size of the grid, the number of items and the number of nodes for the Top-K, shortest path, knapsack and TSP problems respectively in the parenthesis.

		Combinatorial			*CvxpyLayer*		
	MSE	PFY	$\mathcal{L}_{SPO^+}$	$\mathcal{L}_{SCE}$	*Regret*	$\mathcal{L}_{SPO^+}$	$\mathcal{L}_{SCE}$
Top-K (50)	1.614 ±0.874	**0.051** ±0.006	**0.051** ±0.006	**0.051** ±0.006	0.246 ±0.439	**0.051** ±0.006	**0.051** ±0.006
Top-K (80)	1.622 ±0.896	0.018 ±0.001	**0.018** ±0.001	**0.018** ±0.001	0.419 ±0.896	**0.018** ±0.001	**0.018** ±0.001
Top-K (100)	1.623 ±0.9	**0.013** ±0.001	**0.013** ±0.001	**0.013** ±0.001	0.214 ±0.45	**0.013** ±0.001	**0.013** ±0.001
SP (5 × 5)	0.45 ±0.124	0.328 ±0.037	**0.302** ±0.042	0.431 ±0.06	0.339 ±0.035	**0.303** ±0.044	**0.303** ±0.032
SP (8 × 8)	0.539 ±0.064	**0.425** ±0.048	0.447 ±0.038	0.632 ±0.082	0.486 ±0.041	0.454 ±0.031	0.445 ±0.036
SP (10 × 10)	0.492 ±0.113	0.462 ±0.118	0.443 ±0.103	0.626 ±0.165	0.745 ±0.174	0.442 ±0.105	**0.424** ±0.111
KP (10)	0.129 ±0.051	**0.098** ±0.049	0.101 ±0.034	0.163 ±0.009	0.197 ±0.047	0.11 ±0.032	0.104 ±0.044
KP (20)	0.174 ±0.037	**0.128** ±0.035	0.134 ±0.037	0.16 ±0.035	0.222 ±0.075	0.139 ±0.027	**0.129** ±0.029
KP (40)	0.176 ±0.019	**0.149** ±0.011	0.142 ±0.008	0.17 ±0.011	0.217 ±0.025	0.153 ±0.008	**0.146** ±0.009
TSP (5)	0.101 ±0.036	0.079 ±0.032	**0.067** ±0.028	0.152 ±0.05	0.095 ±0.029	0.078 ±0.027	0.073 ±0.026
TSP (6)	0.111 ±0.021	**0.06** ±0.015	**0.059** ±0.014	0.161 ±0.071	0.069 ±0.009	0.081 ±0.01	**0.059** ±0.006
TSP (8)	0.12 ±0.008	0.072 ±0.011	0.071 ±0.013	0.117 ±0.021	0.081 ±0.01	0.095 ±0.011	**0.065** ±0.012

$\mathcal{L}_{SCE}$ using *CvxpyLayer* can compete with the state-of-the-art in DFL. Moreover, $\mathcal{L}_{SPO^+}$ produce lower regret, when a combinatorial solver is used, whereas $\mathcal{L}_{SCE}$ performs better with *CvxpyLayer*. This opens up an interesting side observation— *Eq. 5 ($\mathcal{L}_{SPO^+}$) provide a better subgradient than Eq. 8 ($\mathcal{L}_{SCE}$). However, when one can differentiate through the optimization, $\mathcal{L}_{SCE}$ (Eq. 12) has a better gradient than $\mathcal{L}_{SPO^+}$ (Eq. 11).*

6.2 Experiment with DYS-Net

The previous experiment shows that minimizing $\mathcal{L}_{SCE}$ with a smoothed solver, such as *CvxpyLayer* results in regret comparable to that of the state-of-the-art DFL approaches. In the next set of experiments, we will minimize $\mathcal{L}_{SCE}$ using DYS-Net, which can be fully implemented as a neural network offering substantial improvement in training time.

We present the result for larger problem instances of KP, SP and TSP in Figure 3. In the upper and lower panels, we compare normalized relative test regret and training time of one epoch, respectively. First, we point out that across all instances, training with $\mathcal{L}_{SCE}$ consistently achieves lower regret than training with $SqDE$ using DYS-Net, as done by McKenzie et al. (2024). So, the advantage of

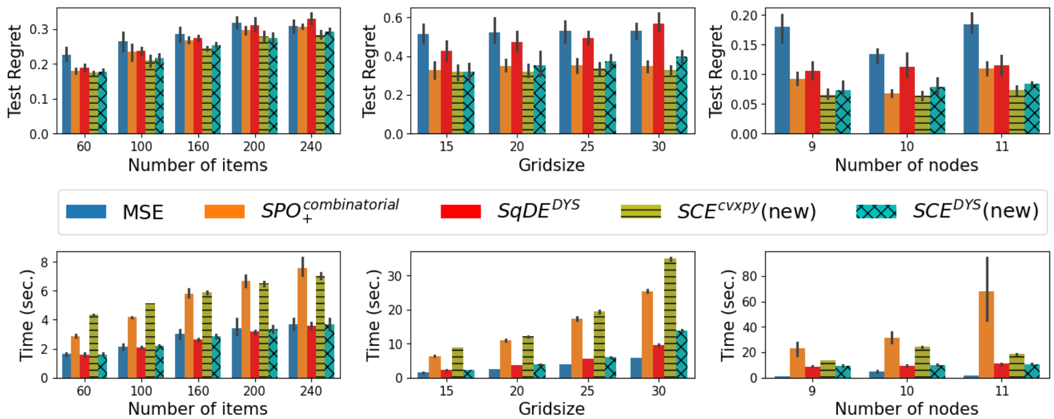


Figure 3: Experiment with DYS-Net in relatively larger KP, SP and TSP instances (from left to right). $SPO_+^{combinatorial}$ minimizes $\mathcal{L}_{SPO+}$ using Gurobi solvers, SCE^{cvxpy} and SCE^{DYS} minimize $\mathcal{L}_{SCE}$ using *CvxpyLayer* and DYS-Net, respectively, whereas $SqDE^{DYS}$ minimizes squared decision error using DYS-Net.

minimizing $\mathcal{L}_{SCE}$ is also manifested with DYS-Net. Although DYS-Net trains significantly faster, minimizing $\mathcal{L}_{SCE}$ with *CvxpyLayer* yields lower regret as it optimally solves the smoothed problem, unlike DYS-Net. Still, in all the knapsack and TSP instances, $\mathcal{L}_{SCE}$ with DYS-Net matches the regret of the SPO approach with significant reduction in training time. For the shortest path instances up to grid-size of 25, $\mathcal{L}_{SCE}$ with DYS-Net produces regret comparable to SPO; however, regret increases for grid-size of 30, where $\mathcal{L}_{SCE}$ with *CvxpyLayer* has lower regret than SPO.

In summary, minimizing $\mathcal{L}_{SCE}$ with DYS-Net yields regret similar to SPO, while significantly reducing runtime. The advantage becomes more pronounced with larger problem sizes; for instance, in the 11-node TSP, DYS-Net is 5 times faster than SPO, which solves an ILP. Notably, these results were achieved without GPU training, suggesting that even greater runtime reductions are possible with GPU usage. By achieving regret comparable to SPO while reducing runtime, our work marks a key advancement in DFL.

7 Conclusion

In this paper, we challenge the conventional DFL approach of directly minimizing empirical regret when a smoothing operation is applied to make the optimization problem differentiable. Instead, we recommend minimizing a surrogate loss, such as $\mathcal{L}_{SCE}$ and justify this by comparing the pattern of the gradient landscape concerning regret and the surrogate loss. By doing so, we effectively merge the two families of approaches in DFL. To provide evidence for minimizing surrogate losses rather than regret, we empirically demonstrate the advantage of minimizing $\mathcal{L}_{SCE}$ instead of $Regret$ using *CvxpyLayer* as the differentiable layer across four benchmark problems. Furthermore, we experiment with the recently proposed DYS-Net, a fast neural solver for LP. By minimizing $Regret$ or $SqDE$, DYS-Net cannot attain regret as low as SPO. We show that for most problems minimizing $\mathcal{L}_{SCE}$ using DYS-Net produces regret as low as the state-of-the-art SPO method, with a clear advantage in runtime up to five-fold.

Future work includes applying this approach to real-world large-scale applications with full GPU training. Furthermore, new fully neural smoothing approaches or better surrogate losses can also benefit from this joint approach. While used here for linear objective functions, future work can investigate the joint applicability of both smoothing and surrogates for non-linear optimisation too.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Mathieu Blondel, André FT Martins, and Vlad Niculae. Learning with fenchel-young losses. *J. Mach. Learn. Res.*, 21(35):1–69, 2020.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Damek Davis and Wotao Yin. A three-operator splitting scheme and its optimization applications. *Set-valued and variational analysis*, 25:829–858, 2017.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 272–279, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390191.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1): 9–26, 2022.

Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1504–1511, Apr. 2020.

Aaron Ferber, Emily Griffin, Bistra Dilkina, Burcu Keskin, and ML Gore. Predicting wildlife trafficking routes with differentiable shortest paths. In *Proceedings of the Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 20th International Conference, CPAIOR 2023*, 2023.

LLC Gurobi Optimization. Gurobi optimizer reference manual. `http://www.gurobi.com`, 2021.

Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The journal of machine learning research*, 13 (1):307–361, 2012.

James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 4475–4482. ijcai.org, 2021.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+optimisation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7272–7282, 2020.

Jayanta Mandi, Emir Demirović, Peter J. Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1603–1610, Apr. 2020.

Jayanta Mandi, Víctor Bucarey, Maxime Mulamba Ke Tchomba, and Tias Guns. Decision-focused learning: Through the lens of learning to rank. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 14935–14947. PMLR, 17–23 Jul 2022.

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.

Daniel McKenzie, Howard Heaton, and Samy Wu Fung. Differentiating through integer linear programs with quadratic regularization and davis-yin splitting. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=H8IaxrANWl`.

Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, and Tias Guns. Contrastive losses and solution caching for predict-and-optimize. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2833–2840. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/390. Main Track.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: Backpropagating through discrete exponential family distributions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 14567–14579. Curran Associates, Inc., 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020.

Utsav Sadana, Abhilash Chenreddy, Erick Delage, Alexandre Forel, Emma Frejinger, and Thibaut Vidal. A survey of contextual optimization methods for decision-making under uncertainty. *European Journal of Operational Research*, 320(2):271–289, 2025. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2024.03.020.

Subham Sekhar Sahoo, Anselm Paulus, Marin Vlastelica, Vít Musil, Volodymyr Kuleshov, and Georg Martius. Backpropagation through combinatorial algorithms: Identity with projection works. In *The Eleventh International Conference on Learning Representations*, 2023.

Sanket Shah, Kai Wang, Bryan Wilder, Andrew Perrault, and Milind Tambe. Decision-focused learning without decision-making: Learning locally optimized decision losses. In *NeurIPS*, 2022.

Bo Tang and Elias B. Khalil. Pyepo: A pytorch-based end-to-end predict-then-optimize library for linear and integer programming, 2023.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pp. 1658–1665. AAAI Press, 2019.

A Proof of Proposition 1

Proof.

1. Following the definition of $\mathcal{L}_{SCE}$,

$$\begin{aligned}\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) &= (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star(\boldsymbol{y}) - (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) \\ &= \hat{\boldsymbol{y}}^\top (\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})) + \boldsymbol{y}^\top (\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{v}^\star(\boldsymbol{y}))\end{aligned}$$

$\hat{\boldsymbol{y}}^\top (\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})) \geq 0$, because $\boldsymbol{v}^\star(\hat{\boldsymbol{y}})$ is the optimal solution to $\hat{\boldsymbol{y}}$. In a similar way, $\boldsymbol{y}^\top (\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{v}^\star(\boldsymbol{y})) \geq 0$. Hence, $\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) \geq 0$.

2. We will prove the claim by contradiction. Assume that $\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0$ but $Regret(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = \boldsymbol{y}^\top (\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{v}^\star(\boldsymbol{y})) > 0$. This is possible if $\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) \neq \boldsymbol{v}^\star(\boldsymbol{y})$.

As the solution to $\hat{\boldsymbol{y}}$ is different from $\boldsymbol{v}^\star(\boldsymbol{y})$, the singleton assumption implies that $\exists\, \boldsymbol{v}' \in \mathcal{F} \setminus \{\boldsymbol{v}^\star(\boldsymbol{y})\} : \hat{\boldsymbol{y}}^\top \boldsymbol{v}' < \hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\boldsymbol{y})$. In this case, we have:

$$
\begin{aligned}
&\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\boldsymbol{y}) - \hat{\boldsymbol{y}}^\top \boldsymbol{v}' > 0 \\
&\Rightarrow (\hat{\boldsymbol{y}}^\top \boldsymbol{v}^\star(\boldsymbol{y}) - \hat{\boldsymbol{y}}^\top \boldsymbol{v}') + (\boldsymbol{y}^\top \boldsymbol{v}' - \boldsymbol{y}^\top \boldsymbol{v}^\star(\boldsymbol{y})) > (\boldsymbol{y}^\top \boldsymbol{v}' - \boldsymbol{y}^\top \boldsymbol{v}^\star(\boldsymbol{y})) \geq 0 \\
&\Rightarrow (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star - (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) > 0
\end{aligned}
$$

In the second line, $\boldsymbol{y}^\top \boldsymbol{v}' - \boldsymbol{y}^\top \boldsymbol{v}^\star(\boldsymbol{y})$ is added in both sides and this term is nonnegative as $\boldsymbol{v}^\star(\boldsymbol{y})$ is the optimal solution to $\boldsymbol{y}$. This implies $\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) > 0$ and we arrive at a contradiction. Thus we prove that $\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0 \implies \textit{Regret}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0$.

Next, assume $\textit{Regret}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = 0$. This implies that $\boldsymbol{y}^\top \boldsymbol{v}^\star(\hat{\boldsymbol{y}}) = \boldsymbol{y}^\top \boldsymbol{v}^\star(\boldsymbol{y})$. This can only be true if $\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) = \boldsymbol{v}^\star(\boldsymbol{y})$ because of the singleton assumption. Hence, $\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})$ $= (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top (\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})) = 0$.

□

B Simulation Experiment

In Section 4, we made the case for minimizing surrogate loss such as $\mathcal{L}_{SCE}$ instead of $Regret$. Our main argument is for a relatively low value of smoothing parameter μ, *Regret* will have zero gradient. However, $\mathcal{L}_{SCE}$ will not have this problem. We provided two illustrations considering small-scale optimization problems. In this case, we justify this with higher-dimensional optimization problems. We consider Top-1 selection problem with different number of items M.

$$\max_{\boldsymbol{v} \in \{0,1\}} \boldsymbol{y}^\top \boldsymbol{v} \ \text{ s.t. } \boldsymbol{v}^\top \mathbf{1} \leq 1 \tag{15}$$

$\boldsymbol{y} = [y_1, \ldots, y_M] \in \mathbb{R}^M$ is the vector denoting value of all the items and $\boldsymbol{v} = [v_1, \ldots, v_M]$ is the vector decision variables. To replicate the setup of a PtO problem, we solve the optimization problem with $\hat{\boldsymbol{y}}$. Let us assume $y_i, \hat{y}_i \geq 0$.

Before, solving the problem with simulation, we will show one interesting aspect of this problem. Note that when $\mu > 0$, the following relaxed optimization problem is solved:

$$\max_{\boldsymbol{v}} \boldsymbol{y}^\top \boldsymbol{v} - \frac{\mu}{2} ||\boldsymbol{v}||^2 \ \text{ s.t. } \boldsymbol{v}^\top \mathbf{1} \leq 1; \ \ \boldsymbol{v} \geq 0 \tag{16}$$

We point out that the solution to the unconstrained optimization problem is $v_i^\star = \frac{y_i}{\mu} > 0$.

The augmented Lagrangian of Equation 16 is

$$\mathbb{L} = \boldsymbol{y}^\top \boldsymbol{v} - \frac{\mu}{2} ||\boldsymbol{v}||^2 + \lambda(1 - \boldsymbol{v}^\top \mathbf{1}) + \boldsymbol{\sigma}^\top \boldsymbol{v} \tag{17}$$

where λ and $\boldsymbol{\sigma} = [\sigma_1, \ldots, \sigma_M]$ are dual variables. By differentiating $\mathbb{L}$ with respect to v_i, we obtain one condition of optimality, which is the following:

$$y_i - \mu v_i - \lambda + \sigma_i = 0 \implies v_i = \frac{y_i - \lambda + \sigma_i}{\mu} \tag{18}$$

Without any loss of generality, let $y^{(1)} \geq y^{(2)} \geq \ldots y^{(M)}$. (In the d) As, solution to the constrained optimization problem is $v_i > 0$, $y^{(1)}$ will definitely be greater than zero. Hence, $\sigma_i = 0$ because of strict complementarity. So, we can write $v^{(1)} - v^{(k)} = \frac{y^{(1)} - y^{(k)} - \sigma^{(k)}}{\mu}$. As, $v^{(1)} - v^{(k)} \leq 1$, we can write:

$$\frac{y^{(1)} - y^{(k)} - \sigma^{(k)}}{\mu} \leq 1 \implies \mu \geq y^{(1)} - y^{(k)} - \sigma^{(k)} \tag{19}$$

So,

$$y^{(1)} - y^{(k)} > \mu \implies \sigma^{(k)} > 0 \implies y^{(k)} = 0 \tag{20}$$

This suggest that if $y^{(k)} < y^{(1)} - \mu$, only $v^{(1)} = 1$ and all other decision variables will be zero in the optimal solution.

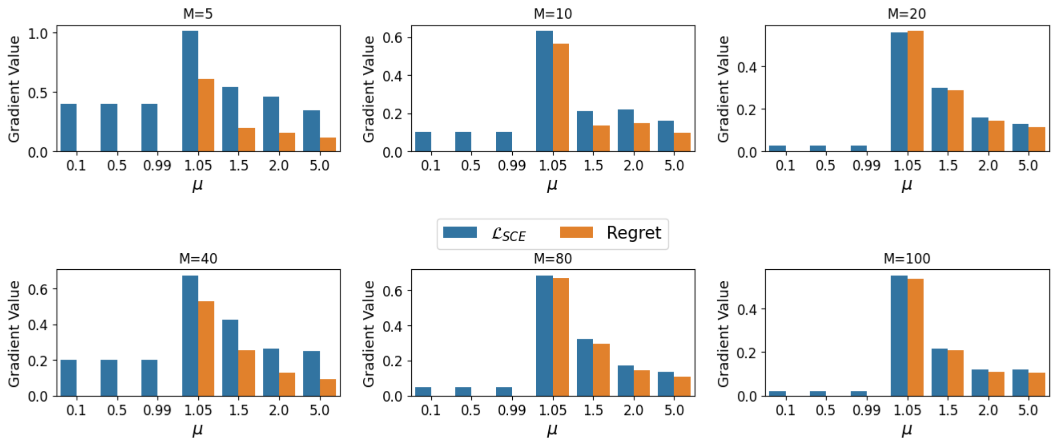


Figure 4: Results of Computational Simulation

To generate the ground truth $\boldsymbol{y}$, we randomly select M integers without replacement from the set $1, \ldots, M$. The predicted costs, $\hat{\boldsymbol{y}}$, are generated by considering a different sample from the same set. As a result, $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ contain the same numbers but in different permutations. It is important to note that all elements in both vectors are positive integer values. We compute the solution to the optimization problem for $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$. We solve the optimization problem with $\hat{\boldsymbol{y}}$ using a 'smoothed' optimization layer—*CvxpyLayer*. in order to compare the gradients of $Regret$ and $\mathcal{L}_{SCE}$. We compute the gradients of both the losses for multiple values of M and μ. For each configuration of M and μ, we run 20 simulations.

Note that $y^{(1)} > y^{(2)} > \ldots y^{(M)}$ because of the way we created the dataset. Moreover, as all values in $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ are integer, Equation 20 suggests if $\mu < 1$, the solution to the relaxed problem (equation 16) will be binary. So, the discussion in Section 4 suggests that slight change of the cost parameter would not change the solution and hence the zero gradient problem would appear while differentiating $Regret$.

In Figure 4, we plotted the average absolute values of the gradients of the two losses— $\mathcal{L}_{SCE}$ and $Regret$. As we hypothesized the gradient turns zero whenever $Regret$ is minimized with $\mu < 1$. It is true that for $\mu > 1$, $Regret$ have non-zero gradient. However, higher values of μ turns solution to the 'smoothed' problem very different from the solution to the original problem. We show this in Table 2 by displaying the average Manhattan distance between solutions of the true and 'smoothed' problem for same $\hat{\boldsymbol{y}}$.

We also highlight that, for the same values of μ, the average Manhattan distances remain same across different M. Examining the results of the simulations, we observed that the solution to the smoothed problem is fractional. For example, when $\mu = 2$, the solution includes two non-zero values— 0.77 and 0.23. Typically, the value 0.77 appears in the position corresponding to the highest value in $\hat{\boldsymbol{y}}$, i.e., where there is a 1 in solution vector. As a result, the Manhattan distance becomes (1-0.77)+0.23 = 0.46. Interestingly, these values remain unchanged across different values of M. Therefore, the Manhattan distance remains constant as long as μ does not change.

C NON-CONVEXITY OF $\mathcal{L}_{SCE}$

The SPO+ loss, $\mathcal{L}_{SPO+}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})$, proposed by Elmachtoub & Grigas (2022) is a convex function of $\hat{\boldsymbol{y}}$. However, the $\mathcal{L}_{SCE}$ loss proposed by Mulamba et al. (2021) is non-convex with respect to $\hat{\boldsymbol{y}}$. Note that,

$$\mathcal{L}_{SCE}(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y}) = \hat{\boldsymbol{y}}^\top(\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})) + \boldsymbol{y}^\top(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{v}^\star(\boldsymbol{y}))$$

We can easily show the convexity of $\mathcal{L}_{SCE}$ with a numerical example. Let us consider the example introduced in Equation 13. In Figure 5, we plot $\mathcal{L}_{SCE}$ and $\mathcal{L}_{SPO+}$ for different values of $\hat{y}$. To make this plot, we use an exact solver, not the 'smoothed' solver. Note that, $\mathcal{L}_{SCE}$ includes a jump when

	M					
μ	5	10	20	40	80	100
0.100	0.000	0.000	0.000	0.000	0.000	0.000
0.500	0.000	0.000	0.000	0.000	0.000	0.000
0.990	0.000	0.000	0.000	0.000	0.000	0.001
1.050	0.089	0.089	0.089	0.089	0.089	0.089
1.500	0.465	0.466	0.465	0.465	0.465	0.464
2.000	0.622	0.622	0.622	0.622	0.622	0.622
5.000	1.165	1.165	1.165	1.165	1.165	1.165

Table 2: We tabulate average Manhattan distance between the solution of the 'smoothed' problem and the solution of the original problem for different values of M and μ.

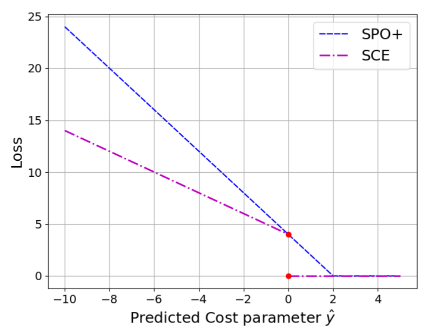


Figure 5: A numerical illustration to show $\mathcal{L}_{SCE}$ is not convex, but $\mathcal{L}_{SPO^+}$ is.

the solution of $\hat{y}$ switches from 1 to 0. However, this is not the case for $\mathcal{L}_{SPO^+}$. More specifically, $\frac{3}{4}\mathcal{L}_{SCE}(2, y) + \frac{1}{4}\mathcal{L}_{SCE}(-2, y) > \mathcal{L}_{SCE}(\frac{3}{4}(2) + \frac{1}{4}(-2), y) = \mathcal{L}_{SCE}(1, y)$, which violates the definition of a convex function.

D MINIMIZING $\mathcal{L}_{SPO^+}$ USING DYS-NET

In Table 1, we show that minimizing $\mathcal{L}_{SCE}$ results in lower regret compared to minimizing $\mathcal{L}_{SPO^+}$ using *CvxpyLayer*. Since both *CvxpyLayer* and DYS-Net are differentiable 'smoothed' layers, we

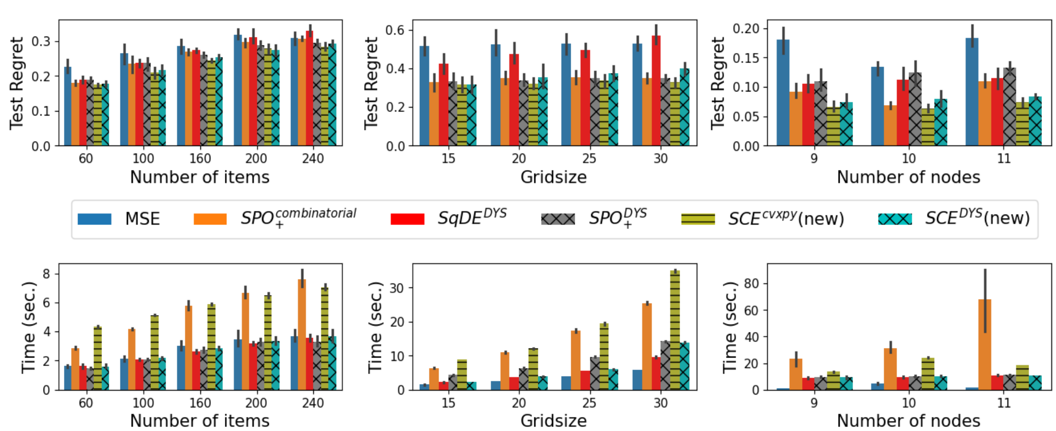


Figure 6: Experiment with DYS-Net in relatively larger KP, SP and TSP instances (from left to right). In addition to Figure 3, we have included the regret results for minimizing $\mathcal{L}_{SPO^+}$ using DYS-Net.

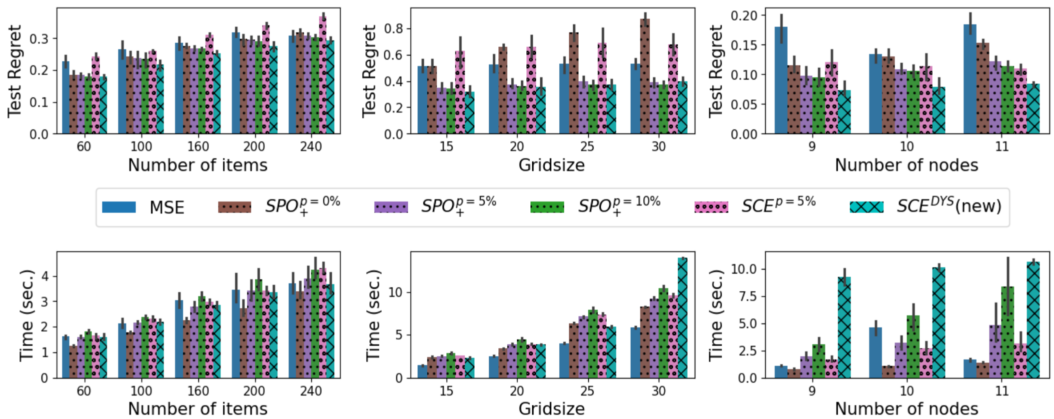


Figure 7: Comparison between DYS-Net and solution caching.

would expect similar results with DYS-Net. For this reason, we included only $\mathcal{L}_{SCE}$ in Figure 3. To ensure completeness, we added the results of minimizing $\mathcal{L}_{SPO^+}$ with DYS-Net in Figure 6. As we hypothesized, this leads to higher average regret compared to minimizing $\mathcal{L}_{SCE}$.

E Comparison against Solution Caching

To reduce the long training time of DFL, Mulamba et al. (2021) propose the idea of solution caching. Instead of finding the optimal solution to $\hat{\boldsymbol{y}}$ or $((2\hat{\boldsymbol{y}} - \boldsymbol{y})$ for $\mathcal{L}_{SPO^+}$), Mulamba et al. (2021) suggest returning a heuristic solution by selecting the optimal one from a finite-dimensional 'cache.' They initialize the cache with all existing solutions in the training data. Furthermore, during training, they randomly solve for $p\%$ of the training instances. Note that if, solve ratio, $p = 100\%$, this strategy becomes equivalent to solving the combinatorial problem for every instance. Conversely, if $p = 0\%$, no additional problem-solving is required during training.

We compare the performance of DYS-Net with solution caching in Figure 7. $SPO_+^{p=10\%}$ denotes the case where $\mathcal{L}_{SPO^+}$ is minimized with a solve ratio of 10%. Similarly, $SPO_+^{p=5\%}$ and $SPO_+^{p=0\%}$ correspond to solve ratios of 5% and 0%, respectively. Similarly, $SCE^{p=5\%}$ stands for minimizing $\mathcal{L}_{SCE}$ with $p = 5\%$. Note that while solution caching approach, Equation 5 and Equation 8 are used for backpropagating $\mathcal{L}_{SPO^+}$ and $\mathcal{L}_{SCE}$ respectively.

It is evident in Figure 7 that $p = 0\%$ results in higher regret for $\mathcal{L}_{SPO^+}$. However, the regret is much lower for p being 5% and 10%. Nevertheless, we point out minimizing $\mathcal{L}_{SCE}$ with DYS-Net results in lower regret. This is particularly prominent for the TSP instances. In terms of training efficiency, solution caching has lower training time for these instances.

F Comparative Analysis in Larger TSP Instances

In Figure 3, we compared TSP instances till 11 nodes. This is due to the fact that for larger TSP instances, we cannot complete training of $SPO_+^{combinatorial}$ and SCE^{cvxpy}. In Figure 8, we consider TSP instances with 15, 20 and 25 nodes. We focus exclusively on TSP instances because, among the three optimization problems considered, because it is the most difficult and time consuming to solve. We have excluded $SPO_+^{combinatorial}$ and SCE^{cvxpy} and included $SPO_+^{p=5\%}$ and $SCE^{p=5\%}$.

We first draw the reader's attention to the observation that $SPO_+^{p=5\%}$ requires more training time compared to $SCE^{p=5\%}$. This discrepancy arises because, in $SPO_+^{p=5\%}$, the optimization problem is solved for $2\hat{\boldsymbol{y}} - \boldsymbol{y}$. Solving for $2\hat{\boldsymbol{y}} - \boldsymbol{y}$ is more challenging and time-consuming compared to solving for $\hat{\boldsymbol{y}}$, as done in $SCE^{p=5\%}$. This is due to the difference in scale between the true cost ($\boldsymbol{y}$) and the predicted cost ($\hat{\boldsymbol{y}}$) We point that this pattern is also visible in Figure 7. The computational burden of $SPO_+^{p=5\%}$ becomes especially pronounced for the larger problem instances. For these instances,

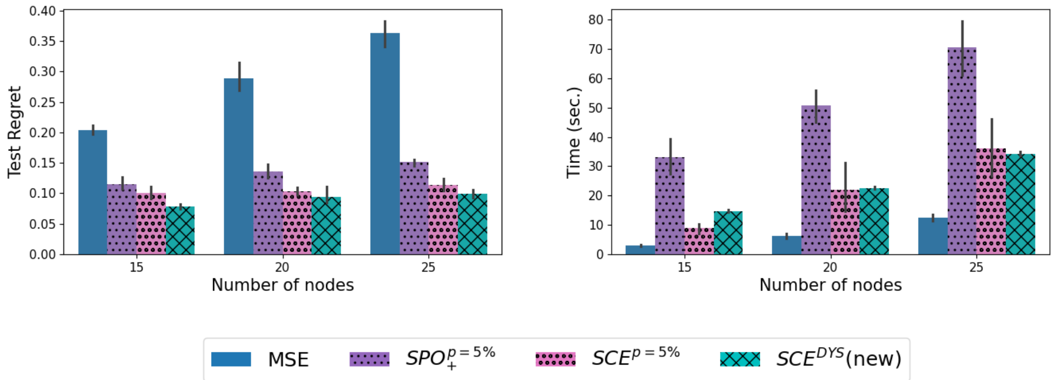


Figure 8: Comparative analysis on larger TSP instances.

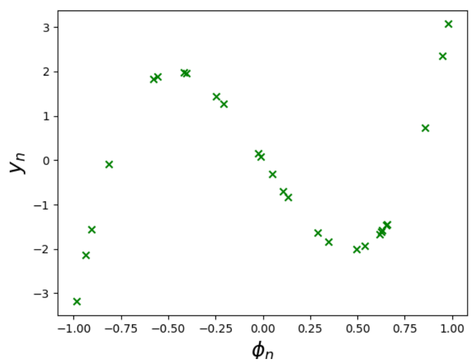


Figure 9: Relationship between y_n and ϕ_n in the Top-K experiment.

solving the optimization problem with $2\hat{\boldsymbol{y}} - \boldsymbol{y}$ often results in timeouts, meaning Gurobi returns an approximate solution instead of the exact one. This is the reason why $SPO_+^{p=5\%}$ exhibits relatively higher regret than $SCE^{p=5\%}$ for these problems.

In contrast, for $SCE^{p=5\%}$, timeouts never occurred, and it exhibits lower training times compared to $SPO_+^{p=5\%}$. For TSP with 15 nodes, $SCE^{p=5\%}$ outperforms DYS-Net in terms of training time. However, as problem size increases, DYS-Net demonstrates better scalability, whereas solving the optimization problem even for 5% of the training instances becomes significantly time-intensive in $SCE^{p=5\%}$.

In terms of regret, DYS-Net demonstrates a significant advantage with much lower regret compared to other methods. This underscores the advantage of minimizing $\mathcal{L}_{SCE}$ using DYS-Net, as it not only delivers lower regret but also scales more effectively for larger problems.

G Explanation of the Top-K Dataset

In the Top-K experiments, the relationship between y_n and ϕ_n is illustrated in Figure 9. All DFL models learn a mapping with a positive slope. As a result, each model selects the Top-1 element as the one with the highest value of ϕ_n, leading to identical accuracy across all DFL models. In contrast, models trained with MSE loss fail to learn a positive slope, resulting in significantly higher regret.

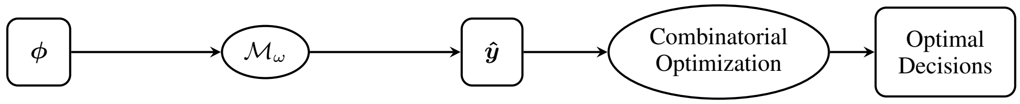


Figure 10: Schematic diagram of a predict-then-optimize (PtO) problem.

H Predict-then-Optimize Problem Description

We consider predicting parameters in the objective function of an LP. These kinds of problems can be framed as *predict-then-optimize* (PtO) problems consisting of a prediction stage followed by an optimization stage, as illustrated in Figure 10. In the prediction stage, an ML model $\mathcal{M}_\omega$ (with trainable parameters ω) is used to predict unknown parameters using features that are correlated to the parameter. During the optimization stage, the problem is solved with the predicted parameters. An offline dataset of past observations is available to train $\mathcal{M}_\omega$.

It is important to distinguish datasets based on whether the true parameters, $\boldsymbol{y}$, are observed and included in the dataset. In some applications, the true parameters, $\boldsymbol{y}$, may not be directly observable, and only the solutions, $\boldsymbol{v}^\star(\boldsymbol{y})$, are observed. While $\boldsymbol{v}^\star(\boldsymbol{y})$ can be computed if $\boldsymbol{y}$ is known, the reverse process is not true, as solving the inverse optimization problem is a research problem in its own.

Whether $\boldsymbol{y}$ is observed or not is important because in order to compute $Regret$ (equation 2), we need the true parameter $\boldsymbol{y}$. Most of the benchmarks in PtO problems assume that $\boldsymbol{y}$ is observed in the past observation. In this case the training data can be expressed as $\{(\boldsymbol{\phi}_i, \boldsymbol{y}_i, \boldsymbol{v}^\star(\boldsymbol{y}_i))\}_{i=1}^N$ and the empirical regret, $\frac{1}{N}\sum_{i=1}^N Regret(\boldsymbol{v}^*(\mathcal{M}_\omega(\boldsymbol{\phi}_i)), \boldsymbol{y}_i)$, can be computed. In most PtO benchmark problems it is assumed that the true $\boldsymbol{y}$ is observed in the training data (Mandi et al., 2024; Tang & Khalil, 2023). However, if the true cost $\boldsymbol{y}$ is not observed in the training data, empirical regret cannot be computed. Rather a different loss has to be considered. For instance, McKenzie et al. (2024) consider squared decision errors (SqDE) between $\boldsymbol{v}^\star(\boldsymbol{y})$ and $\boldsymbol{v}^\star(\hat{\boldsymbol{y}})$, i.e., $SqDE = ||\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})||^2$.

I Different Approaches to Decision-Focused Learning

In PtO problems, the empirical regret can be calculated if the cost, $\boldsymbol{y}$, is observed in the training instances. However, just because it can be calculated does not mean it can be minimized using gradient descent. Figure 11 illustrates the impact of integrating the optimization block into the training loop of neural networks. The key challenge is that to directly minimize $Regret$, it must be backpropagated through the optimization problem. However, for a combinatorial problem $\boldsymbol{v}^\star(\hat{\boldsymbol{y}})$ does not change smoothly with $\hat{\boldsymbol{y}}$, so the gradient, $\frac{d\boldsymbol{v}^\star(\hat{\boldsymbol{y}})}{d\hat{\boldsymbol{y}}}$, is either zero or does not exist.

Differentiable Optimization by Smoothing. 'Differentiable Optimization by Smoothing' is one approach to to circumvent this challenge. As explained in Section 3.1, approaches under this category replace the original optimization problem with a 'smoothed' version of the optimization problem, in which the solution can be expressed as a differentiable mapping of the parameter. For instance, if the original problem is an LP, it can be replaced with a QP by adding a quadratic regularizer to the objective of the LP. In this QP, the solution, $\boldsymbol{v}^\star(\boldsymbol{y})$, can be represented as a differentiable function of the parameter $\boldsymbol{y}$. When the problem is an ILP, first LP, resulting from continuous relaxation is considered and then it is smoothed by adding quadratic regularizer. DYS-Net (McKenzie et al., 2024) provides an approximate solution to the quadratically regularized LP problem, where the computations are designed to be executed as standard neural network operations, enabling back-propagation through it. To summarize, approaches in this category follow the training loop in Figure 11 but only after 'smoothing' the optimization problem.

Surrogate Losses for DFL. The primary goal of DFL is to minimize $Regret$. However, as explained earlier, $Regret$ cannot be minimized directly due to its non-differentiability. Techniques involving surrogate losses aim to address this challenge by identifying suitable surrogate loss functions and computing gradients or subgradients of these surrogate losses for optimization. Figure 12 depicts the training loop of DFL using surrogate loss functions. In this approach, $Regret(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}), \boldsymbol{y})$ is not explicitly computed. Instead, after predicting $\hat{\boldsymbol{y}}$, a new cost vector $\tilde{\boldsymbol{y}}$ is generated based on $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$,

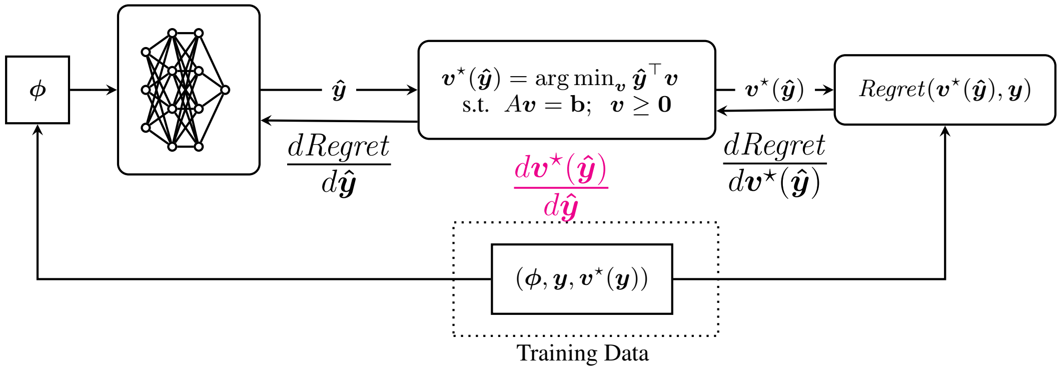


Figure 11: Decision-focused learning training loop.

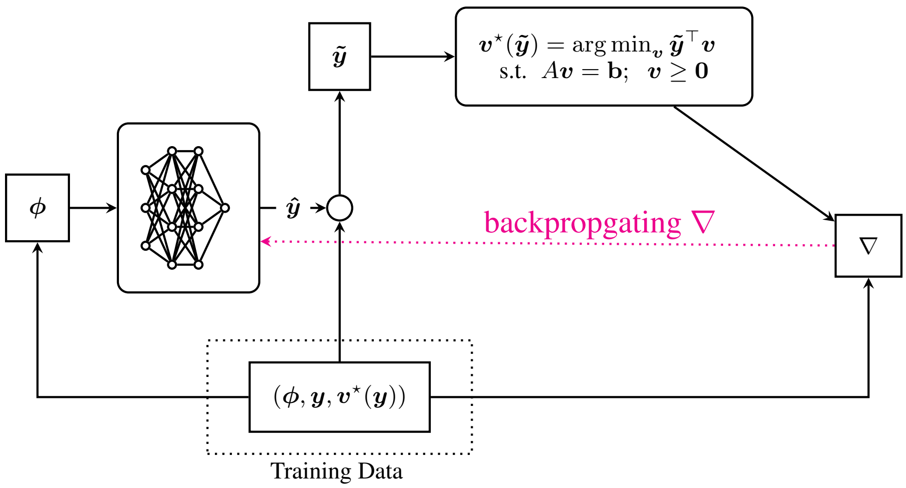


Figure 12: Decision-focused learning using surrogate loss functions.

and the optimization problem is solved using this $\tilde{\boldsymbol{y}}$. Subsequently, a surrogate loss is computed, using $\boldsymbol{v}^\star(\tilde{\boldsymbol{y}})$ and $\boldsymbol{v}^\star(\boldsymbol{y})$, and its gradient,$\nabla$ (shown in pink) , is used for backpropagation. For example, in the case of $\mathcal{L}_{SPO+}$, $\tilde{\boldsymbol{y}} = 2\hat{\boldsymbol{y}} - \boldsymbol{y}$. As shown in Equation 4 $\mathcal{L}_{SPO+} = (2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star(\boldsymbol{y}) - (2\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}^\star(2\hat{\boldsymbol{y}} - \boldsymbol{y})$ Then the gradient used for backpropagation is $\nabla = 2(\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(2\hat{\boldsymbol{y}} - \boldsymbol{y}))$. On the other hand, in the case of $\mathcal{L}_{SCE}$, $\tilde{\boldsymbol{y}} = \hat{\boldsymbol{y}}$ and $\mathcal{L}_{SCE} = \hat{\boldsymbol{y}}^\top(\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}})) + \boldsymbol{y}^\top(\boldsymbol{v}^\star(\hat{\boldsymbol{y}}) - \boldsymbol{v}^\star(\boldsymbol{y}))$. So, in this case, the gradient for backpropagation is $\nabla = (\boldsymbol{v}^\star(\boldsymbol{y}) - \boldsymbol{v}^\star(\hat{\boldsymbol{y}}))$.

Combining Surrogate Losses with Differentiable Optimization. The core idea proposed in this paper is to combine these two approaches. Specifically, the original optimization problem in Figure 12 is replaced with a smoothed version, **allowing backpropagation through the smoothed problem instead of using ∇ directly**. We emphasis that this changes the gradient of the surrogate losses. Instead of Equation 5 and Equation 8, Equation 11 Equation 12 will be used in this case for backprogating $\mathcal{L}_{SPO+}$ and $\mathcal{L}_{SCE}$ respectively.