Journal of Artificial Intelligence Research 81 (2024) 1623-1701 Submitted 08/2023; published 09/2024

Decision-Focused Learning: Foundations, State of the Art, Benchmark and Future Opportunities

Jayanta Mandi JAYANTA.MANDI@KULEUVEN.BE
KU Leuven, Belgium

James Kotary JK4PN@VIRGINIA.EDU
University of Virginia, USA

Senne Berden SENNE.BERDEN@KULEUVEN.BE
KU Leuven, Belgium

Maxime Mulamba MAXIME.MULAMBA@VUB.BE
Vrije Universiteit Brussel, Belgium

Víctor Bucarey VICTOR.BUCAREY@UOH.CL
Universidad de O'Higgins, Chile

Tias Guns TIAS.GUNS@KULEUVEN.BE
KU Leuven, Belgium

Ferdinando Fioretto FIORETTO@VIRGINIA.EDU
University of Virginia, USA

Abstract

Decision-focused learning (DFL) is an emerging paradigm that integrates machine learning (ML) and constrained optimization to enhance decision quality by training ML models in an end-to-end system. This approach shows significant potential to revolutionize combinatorial decision-making in real-world applications that operate under uncertainty, where estimating unknown parameters within decision models is a major challenge. This paper presents a comprehensive review of DFL, providing an in-depth analysis of both gradient-based and gradient-free techniques used to combine ML and constrained optimization. It evaluates the strengths and limitations of these techniques and includes an extensive empirical evaluation of eleven methods across seven problems. The survey also offers insights into recent advancements and future research directions in DFL.

1. Introduction

Real-world applications frequently confront the task of decision-making under uncertainty, such as planning the shortest route in a city, determining optimal power generation schedules, or managing investment portfolios (Sahinidis, 2004; Liu & Liu, 2009; Kim, Lewis, & White, 2005; Hu, Wang, & Gooi, 2016; Delage & Ye, 2010; Garlappi, Uppal, & Wang, 2006). In such scenarios, estimating unknown parameters often poses a significant challenge.

Machine Learning (ML) and Constrained Optimization (CO) serve as two key tools for these complex problems. ML models estimate uncertain quantities, while CO models optimize objectives within constrained spaces. This sequential process, commonly referred to as *predictive* and *prescriptive* modeling, as illustrated in Figure 1, is prevalent in fields like operations research and business analytics (den Hertog & Postek, 2016). For instance, in portfolio management, the prediction stage forecasts asset returns, while the prescriptive

©2024 The Authors. Published by AI Access Foundation under Creative Commons Attribution License CC BY 4.0.

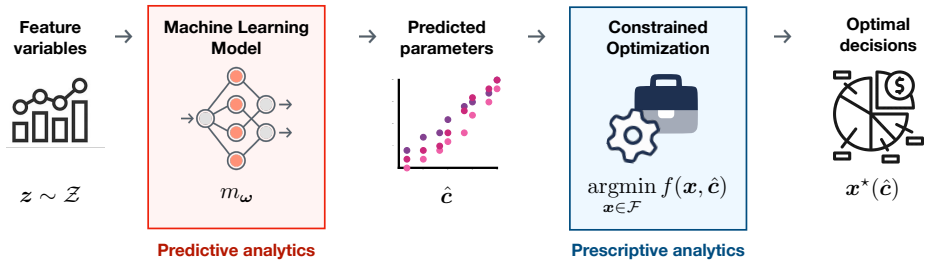


Figure 1: Decision-making under uncertainty involves both predictive and prescriptive analytics. In the predictive stage, the uncertain parameters are predicted from the features using an ML model. In the prescriptive stage, a decision is prescribed by solving a CO problem using the predicted parameters.

phase optimizes returns based on these predictions. The terminology *Predict-Then-Optimize* problem will be used in this survey paper to refer to the problem setting where the uncertain parameter has to be predicted first, followed by solving the CO problem using the predicted parameter to make a decision.

A commonly adopted approach to tackle *Predict-Then-Optimize* problems involves handling these two stages—prediction and optimization—separately and independently. This "two-stage" process first involves training an ML model to create a mapping between observed features and the relevant parameters of a CO problem. Subsequently, and independently, a specialized optimization algorithm is used to solve the decision problem, which is specified by the predicted problem parameters. The underlying assumption in this methodology is that superior predictions would lead to precise predictive models and consequently, high-quality decisions. Indeed, if the predictions of the parameters were perfectly accurate, they would enable the correct specification of CO models which can be solved to yield fully optimal decisions. However, ML models often fall short of perfect accuracy, leading to suboptimal decisions due to propagated prediction errors. Thus, in many applications, the predictive and prescriptive modelings are not isolated but rather, deeply interconnected, and hence should ideally be modeled jointly.

This is the goal of the **decision-focused learning** (DFL) paradigm, which directly trains the ML model to make predictions that lead to good decisions. In other words, DFL integrates prediction and optimization in an end-to-end system trained to optimize a criterion (i.e., a loss function) that is based on the resulting decisions.

Since many ML models, including neural networks (NNs), are trained via gradient-based optimization, the gradients of the loss must be backpropagated through each constituent operation of the model. In DFL, the loss function is dependent on the solution of the CO problem, thus the CO solver is *embedded* as a component of the ML model. In this integration of prediction and optimization, a key challenge is *differentiating through the optimization problem*. An additional challenge arises from decision models operating on discrete variables, which produce discontinuous mappings and hinder gradient-based learning. Hence, examining *smooth* surrogate models for these discrete mappings, along with their differentiation, becomes crucial. These two challenges are the core emphasis and central focal points in DFL.

This survey paper presents a comprehensive overview of decision-focused learning and makes several key contributions. First, to navigate the complex methodologies developed in recent years, the survey differentiates gradient-based DFL methodologies from 'gradient-free' methodologies, which do not rely on computing gradients for learning. Given their compatibility with neural networks, which are the predominant ML architectures, there has been a greater focus on gradient-based DFL methods. To facilitate a comprehensive understanding of this field, we propose categorizing gradient-based learning methods into four distinct classes: **(1)** analytical differentiation of optimization mappings, **(2)** analytical smoothing of optimization mappings, **(3)** smoothing by random perturbations, and **(4)** differentiation of surrogate loss functions. This categorization, illustrated in Figure 4 lower in the paper, serves as a framework for comprehending and organizing various gradient-based DFL methodologies.

In the second part, this paper compiles a selection of problem-specific DFL models, making them publicly available to facilitate broader access and usage. As part of that, we benchmark the performance of the various methods on *seven* distinct problems. This provides an opportunity for comparative understanding and assists in identifying the relative strengths and weaknesses of each approach. The code and data used in the benchmarking are accessible through `https://github.com/PredOpt/predopt-benchmarks`. Finally, this survey looks forward and discusses open challenges and offering an outlook on potential future directions in the field of DFL.

Positioning with respect to other overview papers. With the growing interest of the operations research (OR) and artificial intelligence (AI) communities in integrating ML and CO, various survey, review, and tutorial papers have emerged. Mišić and Perakis (2020) provide applications of *Predict-Then-Optimize* problems in various fields of OR, but offer limited discussion on DFL methodologies. DFL is also a part of a short review article by Kotary, Fioretto, Van Hentenryck, and Wilder (2021), where the primary focus is on recent developments in differentiable optimization to integrate CO problems into neural network architectures. The tutorial by Qi and Shen (2022) introduces the notion of DFL to the OR community and discusses methods that fall under our category of *differentiating surrogate loss functions* below. The PyEPO library (Tang & Khalil, 2023b) offers a common code base and an implementation of a selected number of DFL techniques, along with a common interface for benchmarking on various datasets. Tang and Khalil (2023b) also provide an overview of the DFL techniques implemented in the library, but offer limited discussion on the broader literature of existing DFL techniques. Finally the recent survey by Sadana, Chenreddy, Delage, Forel, Frejinger, and Vidal (2024), which appeared online concurrently with the submission of this article, proposes the umbrella term of *contextual stochastic optimization* and discusses three families of techniques namely decision rule optimization, sequential learning and (stochastic) optimization and integrated learning and (stochastic) optimization; with DFL belonging to the latter.

In contrast, our survey offers a distinct perspective by adopting an ML oriented overview that extensively surveys gradient-based and gradient-free DFL techniques for *Predict-Then-Optimize* problems, where uncertain parameters appear in the objective function of the CO problems. The proposed categorization of gradient-based DFL techniques into four classes allows for a systematic discussion and comparison of the different methods. Moreover, this

article stands out as an experimental survey by performing comparative evaluations of some widely-used DFL techniques on benchmark problems; thereby highlighting the feasibility of implementing, comparing and deploying practical DFL methods. The aim of this dual focus is to enhance understanding of both specific DFL techniques while also showing empirical insights into their effectiveness across various applications.

Paper organization. Following this introduction, the paper is structured as follows. Preliminary concepts are discussed in Section 2, which introduces the problem setting and explains the challenges in implementing DFL. The subsequent Section 3 offers a comprehensive review of existing gradient-free and gradient-based DFL methodologies for handling these challenges, further categorizing the gradient-based methodologies into four distinct classes. Section 4 presents interesting real-world examples of DFL applications. Section 5 brings forth seven benchmark DFL tasks from public datasets, with a comparative evaluation of eleven DFL techniques. Finally, the survey paper concludes by providing a discourse on the current challenges and possible future directions in DFL research.

2. Preliminaries

This section presents an overview of the problem setting, along with preliminary concepts and essential terminology. Then, the central modeling challenges are discussed, setting the stage for a review of current methodologies in the design and implementation of DFL. Throughout the survey paper, vectors are denoted by boldface lowercase letters, such as $\mathbf{x}$, while scalar components within the vector $\mathbf{x}$ are represented with a subscript i, denoting the i^{th} item within $\mathbf{x}$ as x_i. Similarly, the vectors $\mathbf{1}$ and $\mathbf{0}$ symbolize the vector of all-ones and all-zeros, respectively. Moreover, $\mathbf{I}$ denotes an identity matrix of appropriate dimension.

2.1 Problem Setting

In operations research and business analytics, decisions are often quantitatively modeled using CO problems. In many real-world applications, it happens that some parameters of the CO problem are uncertain and must be inferred from contextual data (hereafter referred to as *features*). The settings considered in this survey paper involve estimating those parameters through predictive inferences made by ML models, and subsequently, the final decisions are modeled as the solution to the CO problems based on those inferences. In this setting, the decision-making processes can be described by *parametric* CO problems, defined as,

$$\mathbf{x}^\star(\mathbf{c}) = \underset{\mathbf{x}}{\operatorname{argmin}} \;\; f(\mathbf{x}, \mathbf{c}) \tag{1a}$$

$$\texttt{s.t.} \;\; \boldsymbol{g}(\mathbf{x}, \mathbf{c}) \leq \mathbf{0} \tag{1b}$$

$$\boldsymbol{h}(\mathbf{x}, \mathbf{c}) = \mathbf{0}. \tag{1c}$$

The goal of the CO problem above is to find *a* solution $\mathbf{x}^\star(\mathbf{c}) \in \mathbb{R}^n$ (n being the dimension of the decision variable x), which minimizes the objective function f, subject to equality and inequality constraints as defined by the functions $\boldsymbol{h}$ and $\boldsymbol{g}$. This *parametric* problem formulation defines $\mathbf{x}^\star(\mathbf{c})$ as a function of the parameters $\mathbf{c} \in \mathbb{R}^m$.

CO problems can be categorized in terms of the forms taken by the functions defining their objectives (1a) and constraints (1b-1c). These forms also determine important properties of the optimization mapping $\mathbf{c} \rightarrow \mathbf{x}^\star(\mathbf{c})$, when viewed as a function from problem parameters to optimal solutions, such as its continuity, differentiability, and injectivity.

In this survey paper, it is assumed that the constraints are fully known prior to solving, i.e., $\boldsymbol{h}(\mathbf{x}, \mathbf{c}) = \boldsymbol{h}(\mathbf{x})$ and $\boldsymbol{g}(\mathbf{x}, \mathbf{c}) = \boldsymbol{g}(\mathbf{x})$, indicating the constraints do not depend on the parameter $\mathbf{c}$, which is uncertain. Rather the dependence on $\mathbf{c}$ is restricted solely to the objective function. This is the setting considered by almost all existing works surveyed. While it is also possible to consider uncertainty in the constraints, this leads to the possibility of predicting parameters that lead to solutions that are infeasible with respect to the ground-truth parameters. The learning problem has not yet been well-defined in this setting (unless a recourse action to correct infeasible solutions is used (Hu, Lee, & Lee, 2023c, 2023a)). For this reason, in the following sections, only f is assumed to depend on $\mathbf{c}$, so that $\boldsymbol{g}(\mathbf{x}) \leq \mathbf{0}$ and $\boldsymbol{h}(\mathbf{x}) = \mathbf{0}$ are satisfied for all outputs of the decision model. For notational convenience, the feasible region of the CO problem in (1), will be denoted by $\mathcal{F}$ (i.e., $\mathbf{x} \in \mathcal{F}$ if and only if $\boldsymbol{g}(\mathbf{x}) \leq \mathbf{0}$ and $\boldsymbol{h}(\mathbf{x}) = \mathbf{0}$).

If the true parameters $\mathbf{c}$ are known exactly, the corresponding 'true' optimal decisions may be computed by solving (1). In such scenarios, $\mathbf{x}^\star(\mathbf{c})$ will be referred to as the *full-information optimal decisions* (Bertsimas & Kallus, 2020). This paper, instead, considers problems where the parameters $\mathbf{c}$ are unknown but can be estimated as a function of observed features $\mathbf{z}$. The problem of estimating $\mathbf{c}$ falls under the category of supervised ML problems. In this setting, a set of past observation pairs $\{(\mathbf{z_i}, \mathbf{c_i})\}_{i=1}^N$ is available as a training dataset, $\mathcal{D}$, and used to train an ML model $m_{\boldsymbol{\omega}}$ (with trainable ML parameters $\boldsymbol{\omega}$), so that parameter predictions take the form $\hat{\mathbf{c}} = m_{\boldsymbol{\omega}}(\mathbf{z})$. Then, a decision $\mathbf{x}^\star(\hat{\mathbf{c}})$ can be made based on the predicted parameters. $\mathbf{x}^\star(\hat{\mathbf{c}})$ is referred to as a *prescriptive decision*. The overall learning goal is to optimize the set of prescriptive decisions made over a distribution of features $\mathbf{z} \sim \mathcal{Z}$, with respect to some evaluation criterion on those decisions. Thus, while the ML model $m_{\boldsymbol{\omega}}$ is trained to predict $\hat{\mathbf{c}}$, its performance is evaluated based on $\mathbf{x}^\star(\hat{\mathbf{c}})$. This paper uses the terminology *Predict-Then-Optimize* problem to refer to the problem of predicting $\hat{\mathbf{c}}$, to improve the evaluation of $\mathbf{x}^\star(\hat{\mathbf{c}})$.

2.2 Learning Paradigms

The defining challenge of the *Predict-Then-Optimize* problem setting is the gap in modeling between the prediction and the optimization components: while $m_{\boldsymbol{\omega}}$ is trained to predict $\hat{\mathbf{c}}$, it is evaluated based on the subsequently computed $\mathbf{x}^\star(\hat{\mathbf{c}})$. Standard ML approaches, based on the *empirical risk minimization* (ERM) (Vapnik, 1999), use standard loss functions $\mathcal{L}$, such as mean squared error or cross-entropy, in order to learn to predict $\hat{\mathbf{c}} = m_{\boldsymbol{\omega}}(\mathbf{z})$. The learning is supervised by by the ground-truth $\mathbf{c}$. However, in principle, for *Predict-Then-Optimize* problems, it is desirable to train $m_{\boldsymbol{\omega}}$ to make predictions $\hat{\mathbf{c}}$ that optimize the evaluation criterion on $\mathbf{x}^\star(\hat{\mathbf{c}})$ directly. This distinction motivates the definition of two alternative learning paradigms for *Predict-Then-Optimize* problems.

Prediction-focused learning (PFL). A straightforward approach to this supervised ML problem is to train the model to generate accurate parameter predictions $\hat{\mathbf{c}}$ with respect to ground-truth values $\mathbf{c}$. This paper introduces the term *prediction-focused learning* to refer

to this approach (also called two-stage learning (Wilder, Dilkina, & Tambe, 2019a)) because the model is trained with a focus on the accuracy of the parameter predictions preceding the decision model. Here, the training is agnostic of the downstream CO problem. At the time of making the decision, the pre-trained model's predictions $\hat{\mathbf{c}}$ are passed to the CO solvers which solve (1) to return $\mathbf{x}^\star(\hat{\mathbf{c}})$. Typical ML losses, such as the mean squared error (MSE) or binary cross entropy (BCE), are used to train the prediction model in this case.

$$MSE\ (\hat{\mathbf{c}}, \mathbf{c}) = \frac{1}{N}\|\mathbf{c} - \hat{\mathbf{c}}\|^2 \tag{2}$$

Such loss functions, like Eq. (2), which measure the prediction error of $\hat{\mathbf{c}}$ with respect to $\mathbf{c}$, are referred to as *prediction losses*. Algorithm 1 illustrates PFL with MSE loss.

Decision-focused learning (DFL). By contrast, in *decision-focused* learning, the ML model is trained to optimize the evaluation criteria which measure the quality of the resulting decisions. As the decisions are realized after the optimization stage, this requires the integration of prediction and optimization components, into a composite framework which produces full decisions. From this point of view, generating the predicted parameters $\hat{\mathbf{c}}$ is an intermediary step of the integrated approach, and the accuracy of $\hat{\mathbf{c}}$ is not the primary focus in training. The focus, rather, is on the error incurred after optimization. A measure of error with respect to the integrated model's prescriptive decisions, when used as a loss function for training, is henceforth referred to as a *task loss*. The essential difference from the aforementioned prediction loss is that it measures the error in $\mathbf{x}^\star(\hat{\mathbf{c}})$, rather than in $\hat{\mathbf{c}}$.

The objective value achieved by using the predicted $\mathbf{x}^\star(\hat{\mathbf{c}})$ is generally suboptimal with respect to the true objective parameters $\mathbf{c}$. Often, the end goal is to generate predictions $\hat{\mathbf{c}}$ with an optimal solution $\mathbf{x}^\star(\hat{\mathbf{c}})$ whose objective value in practice (i.e., $f(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c})$) comes close to the full-information optimal value $f(\mathbf{x}^\star(\mathbf{c}), \mathbf{c})$. In such cases, a salient notion of task loss is the *regret*, defined as the difference between the full-information optimal objective value and the objective value realized by the prescriptive decision. Equivalently, it is the magnitude of suboptimality of the decision $\mathbf{x}^\star(\hat{\mathbf{c}})$ with respect to the optimal solution $\mathbf{x}^\star(\mathbf{c})$ under ground-truth parameters $\mathbf{c}$:

$$\textit{Regret}\ (\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c}) = f(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c}) - f(\mathbf{x}^\star(\mathbf{c}), \mathbf{c}) \tag{3}$$

Note that minimizing regret is equivalent to minimizing the value of $f(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c})$, since the term $f(\mathbf{x}^\star(\mathbf{c}), \mathbf{c})$ is constant with respect to the prediction model. While regret may be considered the quintessential example of a task loss, other task losses can arise in practice. For example, when the ground-truth target data are observed in terms of decision values $\mathbf{x}^\star$, rather than parameter values $\mathbf{c}$, they may be targeted using the typical training loss functions such as $MSE\ (\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{x}^\star)$.

Relationship between prediction and task losses. As previously mentioned, an ML model is trained without considering the downstream CO problem in prediction-focused learning for *Predict-Then-Optimize* tasks; still the ML model is evaluated at test time on the basis of its resulting CO problem solutions. This is based on an underlying assumption that generating accurate predictions with respect to a standard prediction loss will result in good prescriptive decisions. Note that zero prediction loss always implies zero task loss, since $\hat{\mathbf{c}} = \mathbf{c}$ implies $\mathbf{x}^\star(\hat{\mathbf{c}}) = \mathbf{x}^\star(\mathbf{c})$. However, in practice, it is impossible to learn an ML

model that makes no prediction error on any sample. The model error can only be minimized in one metric, and the minimization of the prediction error and the resulting decision error do not in general coincide (Demirović, Stuckey, Bailey, Chan, Leckie, Ramamohanarao, & Guns, 2019). Furthermore, the prediction loss and the task loss are, in general, not continuously related. These principles are illustrated by the following example.

Example. The shortcomings of training with respect to prediction errors can be illustrated with a simple CO problem. For this illustration, consider a knapsack problem (Pisinger & Toth, 1998). The objective of the knapsack problem is to select a maximal-value subset from an overall set of items, each having its own value and unit weight, subject to a capacity constraint, which imposes that the number of selected items cannot be higher than the capacity C. This knapsack problem with unit weights can be formulated as follows:

$$\mathbf{x}^\star(\mathbf{c}) = \underset{\mathbf{x}\in\{0,1\}}{\operatorname{argmax}} \mathbf{c}^\top \mathbf{x} \;\; \texttt{s.t.} \sum_i x_i \leq C \tag{4}$$

In a *Predict-Then-Optimize* variant of this knapsack problem, the item weights and knapsack capacity are known, but the item values are unknown and must be predicted using observed features. The ground-truth item value $\mathbf{c}$ implies the ground-truth solution $\mathbf{x}^\star(\mathbf{c})$. Overestimating the values of the items that are chosen in $\mathbf{x}^\star(\mathbf{c})$ (or underestimating the values of the items that are not chosen) increases the prediction error. Note that these kind of prediction errors, even if they are high, do not affect the solution, and thus do not affect the task loss either. On the other hand, even low prediction errors for some item values may change the solution, affecting the task loss. That is why after a certain point, reducing prediction errors does not decrease task loss, and sometimes may increase it. DFL aims to address this shortcoming of PFL: by minimizing the task loss directly, prediction errors are implicitly traded off on the basis of how they affect the resulting decision errors.

The discrepancy between the prediction loss and the task loss has been exemplified in Figure 2 for a very simple knapsack problem with only two items. For this illustration, assume that both the items are of unit weights and the capacity of the knapsack is one, i.e., only one of the two items can be selected. The true values of the first and second items are 2.5 and 3 respectively. The point $(2.5, 3)$, marked with ✱, represents the true item values. In this case the true solution is $(0, 1)$, which corresponds to selecting only the second item. It is evident that any prediction in the blue shaded region leads to this solution. For instance, the point $(1.5, 3)$, marked with ✚, corresponds to predicting 1.5 and 3 as values of the two items respectively and this results in selecting the second item. On the other hand, the point $(2.5, 2)$, marked with ✖, triggers the wrong solution $(1, 0)$, although the squared error values of ✚ and ✖ are identical. Also, note that overestimating the value of the second item does not change the solution. For instance, the point $(1.5, 4)$, marked with ▲, corresponds to overestimating the value of the second item to 4 while keeping the value of the first item the same as the point in ✚. This point is positioned directly above the point in ✚ and still stays in the blue-shaded region. Similarly, the point $(0.5, 3)$, marked with ▼, results from underestimating the value of the first item and is in the blue shaded region too. Although these two points have higher values of squared error than the point marked with ✖, they trigger the right solution, resulting in zero regret.

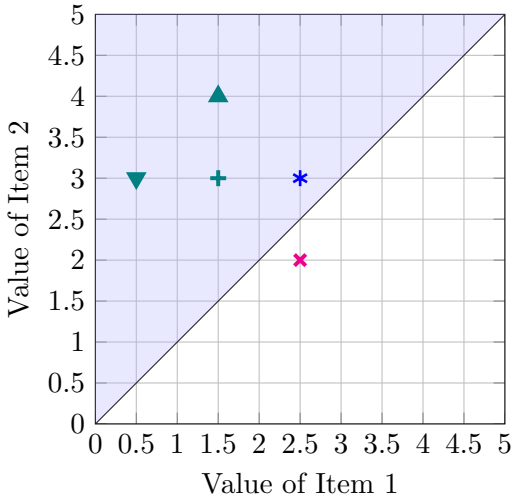


Figure 2: An illustrative numerical example with a knapsack problem with two items to exemplify the discrepancy between prediction error and regret. The figure illustrates that two points can have the same prediction error but different regret. Furthermore, it demonstrates that overestimating the values of the selected items or underestimating the values of the items that are left out does not change the solution, and thus does not increase the regret, even though the prediction error does increase.

Empirical risk minimization and bilevel form of DFL. The minimization of either the expected prediction loss in PFL or the expected task loss in DFL, can be expressed as an Empirical Risk Minimisation (ERM) problem over a training dataset $\mathcal{D} \equiv \{(\mathbf{z_i}, \mathbf{c_i})\}_{i=1}^{N}$. The desired objective of training is to learn a model, $m_{\boldsymbol{\omega}}$, that minimizes the *expected loss*. However, as the joint probability distribution of $(\mathbf{z}, \mathbf{c})$ is often unknown, ERM minimizes the empirical loss instead, i.e., ERM minimizes the average loss, calculated over $\mathcal{D}$.

Note that ERM typically involves making a point prediction $\hat{\mathbf{c}}$ to minimize expected loss. However, besides point predictions, it is also possible to estimate the distribution of $\hat{\mathbf{c}}$. The advantage of distributional estimation is that it allows the decision-maker to consider extreme cases of the parameter distribution, leading to more robust decisions. However, minimizing distributionally robust loss (Levy, Carmon, Duchi, & Sidford, 2020) is inherently more challenging than ERM, even for standard ML problems. As a result, most of the surveyed works focus on minimizing expected task loss in DFL. Hence, learning techniques for minimizing *expected* task loss are the main focus in this survey paper.

The respective ERM problems below assume the use of the MSE and regret loss functions for PFL and DFL respectively, but the principles described here hold for a wide range of alternative loss functions. PFL, by minimizing the prediction error with respect to the ground-truth parameters directly, takes the form of a standard regression problem:

$$\min_{\boldsymbol{\omega}} \frac{1}{N} \sum_{i=1}^{N} \|m_{\boldsymbol{\omega}}(\mathbf{z_i}) - \mathbf{c_i}\|^2, \tag{5}$$

which is an instance of unconstrained optimization. In the case of DFL, it is natural to view the ERM as a bilevel optimization problem:

$$\min_{\boldsymbol{\omega}} \frac{1}{N} \sum_{i=1}^{N} \Big(f(\mathbf{x}^\star(\hat{\mathbf{c}}_\mathbf{i}), \mathbf{c}_\mathbf{i}) - f(\mathbf{x}^\star(\mathbf{c}_\mathbf{i}), \mathbf{c}_\mathbf{i}) \Big) \tag{6a}$$

$$\texttt{s.t. } \hat{\mathbf{c}}_\mathbf{i} = m_{\boldsymbol{\omega}}(\mathbf{z}_\mathbf{i});\ \mathbf{x}^\star(\hat{\mathbf{c}}_\mathbf{i}) = \operatorname*{argmin}_{\mathbf{x} \in \mathcal{F}} f(\mathbf{x}, \hat{\mathbf{c}}_\mathbf{i}) \tag{6b}$$

The outer-level problem (6a) minimizes task loss on the training set while the inner-level problem (6b) computes the mapping $\mathbf{c} \to \mathbf{x}^\star(\mathbf{c})$. Solving (6) is computationally more challenging than solving (5) in the prediction-focused paradigm. In both cases, optimization by stochastic gradient descent (SGD) is the preferred method for training neural networks.

Algorithms 1 and 2 compare the gradient descent training schemes for PFL and DFL. Algorithm 1 is a standard application of gradient descent, in which the derivatives of Line 6 are generally well-defined and can be computed straightforwardly (typically by automatic differentiation (Baydin, Pearlmutter, Radul, & Siskind, 2018)). Line 7 of Algorithm 2 shows that direct differentiation of the mapping $\mathbf{c} \to \mathbf{x}^\star(\mathbf{c})$ can be used to form the overall task loss gradient $\frac{d\mathcal{L}}{d\boldsymbol{\omega}}$, by providing the required chain rule term $\frac{d\mathbf{x}^\star(\hat{\mathbf{c}})}{d\hat{\mathbf{c}}}$. However, this differentiation is nontrivial as the mapping itself lacks a closed-form representation. Furthermore, many interesting and practical optimization problems are inherently nondifferentiable and even discontinuous as functions of their parameters, precluding the direct application of Algorithm 2 to optimize (6) by gradient descent. The following subsections review the main challenges of implementing Algorithm 2.

2.3 Challenges to Implement Decision-Focused Learning

Differentiation of CO mappings. When considering gradient-based learning techniques, one needs to minimize the task loss by backpropagating the error over it. Hence, the partial derivatives of the task loss with respect to the prediction model parameters $\boldsymbol{\omega}$ must be computed to carry out the parameter update at Line 7 of Algorithm 2. Since the task loss $\mathcal{L}$ is a function of $\mathbf{x}^\star(\hat{\mathbf{c}})$, the gradient of $\mathcal{L}$ with respect to $\boldsymbol{\omega}$ can be expressed in the following terms by using the chain rule of differentiation:

$$\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c})}{d\boldsymbol{\omega}} = \frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c})}{d\mathbf{x}^\star(\hat{\mathbf{c}})} \frac{d\mathbf{x}^\star(\hat{\mathbf{c}})}{d\hat{\mathbf{c}}} \frac{d\hat{\mathbf{c}}}{d\boldsymbol{\omega}} \tag{7}$$

The first term in the right side of (7), can be computed directly as $\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c})$ is typically a differentiable function of $\mathbf{x}^\star(\hat{\mathbf{c}})$. A deep learning library (such as PyTorch (Paszke, Gross, Massa, Lerer, Bradbury, Chanan, Killeen, Lin, Gimelshein, Antiga, et al., 2019)) computes the last term by representing the neural network as a computational graph and applying automatic differentiation (autodiff) in the reverse mode (Baydin et al., 2018). However, the second term, $\frac{d\mathbf{x}^\star(\hat{\mathbf{c}})}{d\hat{\mathbf{c}}}$, may be nontrivial to compute given the presence of two major challenges: **(1)** The mapping $\hat{\mathbf{c}} \to \mathbf{x}^\star(\hat{\mathbf{c}})$, as defined by the solution to an optimization problem, *lacks a closed form* which can be differentiated directly, and **(2)** for many interesting and useful optimization models, the mapping is *nondifferentiable* in some points, and has zero-valued gradients in others, precluding the straightforward use of gradient descent.

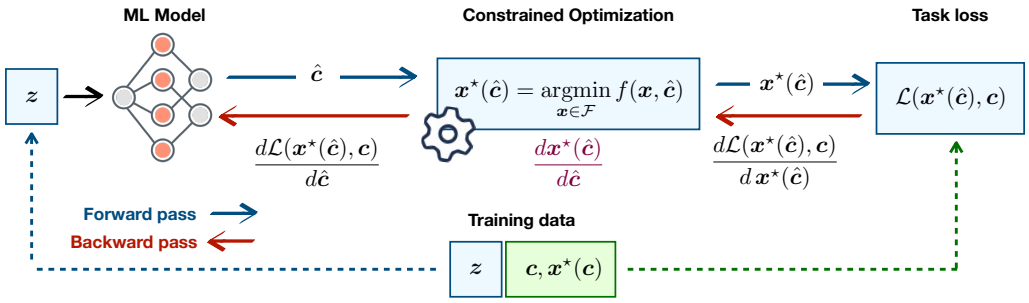


Figure 3: In decision-focused learning, the neural network model is trained to minimize the task loss

As shown in the next subsection, even the class of linear programming problems, widely used in decision modeling, is affected by both issues. Section 3 details the various existing approaches aimed at overcoming these challenges.

Algorithm 1 Gradient-descent in prediction-focused learning

Input: training data $\mathrm{D} \equiv \{(\mathbf{z_i}, \mathbf{c_i})\}_{i=1}^{N}$
Hyperparams: α- learning rate

1: Initialize $\boldsymbol{\omega}$.
2: **for** each epoch **do**
3: **for** each instance $(\mathbf{z}, \mathbf{c})$ **do**
4: $\hat{\mathbf{c}} = m_{\boldsymbol{\omega}}(\mathbf{z})$
5: $\mathcal{L} = (\hat{\mathbf{c}} - \mathbf{c})^2$
6: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \alpha \frac{d\mathcal{L}}{d\hat{\mathbf{c}}} \frac{d\hat{\mathbf{c}}}{d\boldsymbol{\omega}}$
7: **end for**
8: **end for**

Algorithm 2 Gradient-descent in decision-focused learning with regret as task loss

Input: $\mathcal{F}$, training data $\mathrm{D} \equiv \{(\mathbf{z_i}, \mathbf{c_i}, \mathbf{x}^\star(\mathbf{c_i})\}_{i=1}^{N}$;
Hyperparams: α- learning rate

1: Initialize $\boldsymbol{\omega}$.
2: **for** each epoch **do**
3: **for** each instance $(\mathbf{z}, \mathbf{c}, \mathbf{x}^\star(\mathbf{c}))$ **do**
4: $\hat{\mathbf{c}} = m_{\boldsymbol{\omega}}(\mathbf{z})$
5: $\mathbf{x}^\star(\hat{\mathbf{c}}) = \operatorname{argmin}_{\mathbf{x}\in\mathcal{F}} f(\mathbf{x}, \hat{\mathbf{c}})$
6: $\mathcal{L} = f(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c}) - f(\mathbf{x}^\star(\mathbf{c}), \mathbf{c})$
7: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \alpha \frac{d\mathcal{L}}{d\mathbf{x}^\star(\hat{\mathbf{c}})} \frac{d\mathbf{x}^\star(\hat{\mathbf{c}})}{d\hat{\mathbf{c}}} \frac{d\hat{\mathbf{c}}}{d\boldsymbol{\omega}}$
8: **end for**
9: **end for**

Computational cost. Another major challenge in DFL is the computational resources required to train the integrated prediction and optimization model. Note that Line 5 in Algorithm 2 evaluates $\mathbf{x}^\star(\hat{\mathbf{c}})$. This requires solving and differentiating the underlying CO problem for each observed data sample, in each epoch. This imposes a significant computational cost even when dealing with small-scale and efficiently solvable CO problems, but can become an impediment in the case of large and (NP-)hard optimization problems.

2.4 Optimization Problem Forms

The effectiveness of solving an optimization problem depends on the specific forms of the objective and constraint functions. Considerable effort has been made to develop efficient algorithms for certain optimization forms. Below, the readers are provided an overview of the key and widely utilized types of optimization problem formulations.

2.4.1 Convex Optimization

In *convex* optimization problems, a convex objective function is to be optimized over a convex feasible space. This class of problems is distinguished by the guarantee that any locally optimal solution is also globally optimal (Boyd & Vandenberghe, 2014). Since many optimization problems converge provably to local minima, convex problems are considered to be reliably and efficiently solvable as opposed to *nonconvex* problems. Despite this, convex optimization mappings still impose significant computational overhead on Algorithm 2 since most convex optimizations are orders of magnitude more complex than conventional neural network layers (Amos & Kolter, 2017). Like all parametric optimization problems, convex ones are implicitly defined mappings from parameters to optimal solutions, lacking a closed form that can be differentiated directly. However as detailed in Section 3.1.1, they can be canonicalized to a standard form, which facilitates automation of their solution and backpropagation by a single standardized procedure (Agrawal, Amos, Barratt, Boyd, Diamond, & Kolter, 2019a).

The class of convex problems is broad enough to include some that yield mappings $\mathbf{x}^\star(\hat{\mathbf{c}})$ that are differentiable everywhere, and some that do not. The *linear programs*, which are convex and form nondifferentiable mappings with respect to their objective parameters, are notable examples of the latter case and are discussed next.

2.4.2 Linear Programming

Linear Programs (LPs) are convex optimization problems whose objective and constraints are composed of affine functions. These programs are predominant as decision models in operations research, and have endless industrial applications since the allocation and transfer of resources is typically modeled by linear relationships between variables (Bazaraa, Jarvis, & Sherali, 2008). The parametric LPs considered in this survey paper take the following form:

$$\mathbf{x}^\star(\mathbf{c}) = \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{c}^\top \mathbf{x} \tag{8a}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{8b}$$

$$\mathbf{x} \geq \mathbf{0} \tag{8c}$$

Compared to other classes of convex problems, LPs admit efficient solution methods, even for large-scale problems (Bazaraa et al., 2008; Ignizio & Cavalier, 1994). From a DFL standpoint, however, LPs pose a challenge, because the mapping $\mathbf{c} \to \mathbf{x}^\star(\mathbf{c})$ is nondifferentiable. Although the derivatives of mapping (8) are defined almost everywhere, they provide no useful information for gradient descent training. To see this, first note the well-known fact that a linear program always takes its optimal value at a vertex of its feasible set (Bazaraa et al., 2008). Since the number of vertices in any such set is finite, (8) maps a continuous parameter space to a discrete set of solutions. As such, it is a piecewise constant mapping. Therefore its derivatives are zero almost everywhere, and undefined elsewhere. Prevalent strategies for incorporating linear programs in DFL thus typically rely on differentiating smooth approximations to the LP, as detailed in Section 3.1.2.

Many OR problems, such as the allocation and planning of resources, can be modeled as LPs. Also many prototypical problems in algorithm design (e.g., sorting and top-k selection)

can be formulated as LPs with continuous variables, despite admitting only discrete integer solutions, by relying on the total unimodularity of the constraint matrices (Bazaraa et al., 2008). In what follows, some examples of machine learning models of LPs and how they might occur in a *Predict-Then-Optimize* context are given.

Shortest paths. Given a directed graph with a given start and end node, the goal in the shortest path problem is to find a sequence of arcs of minimal total length that connects the start and the end node. The decision variables are binary indicators of each edge's inclusion in the path. The linear constraints ensure $[0, 1]$ bounds on each indicator, as well as flow balance through each node. These flow balance constraints capture that, except for the start and end node, each node has as many incoming selected arcs as outgoing selected arcs. For the start node, there is one additional outgoing selected arc, and for the end node, there is one more incoming selected arc. The parameters in the linear objective represent the arc lengths. In many realistic settings—as well as in several common DFL benchmarks (Elmachtoub & Grigas, 2022; Pogančić, Paulus, Musil, Martius, & Rolinek, 2020)—these are unknown, requiring them to be predicted before a shortest path can be computed. This motivating example captures the realistic setting in which the shortest route between two locations has to be computed, but in which the road traversal times are uncertain (due to unknown traffic conditions, for example), but can be predicted from known features (such as day of the week, time of day and weather conditions).

Bipartite matching. Given a bipartite graph with weighted arcs, where the weights are unknown and must be predicted, the task is to choose a subset of arcs so that each node is involved in at most one selected arc, maximizing the total weight. The variables lie in $[0, 1]$ indicating the inclusion of each edge and the weights are the objective parameters. The constraints ensure that each node is involved at most once in a selected arc. With a complete bipartite graph, matchings can be construed as permutations, which can be employed in tasks such as learning to rank (Kotary, Fioretto, Van Hentenryck, & Zhu, 2022).

Sorting and Ranking. The sorting of any list of predicted values can be posed as a linear program over a feasible region whose vertices correspond to all of the possible permutations of the list. The related ranking, or argsort problem assigns to any length-n list a permutation of sequential integers $[n]$ which sorts the list. By smoothing the linear program, these basic operations can be differentiated and backpropagated (Blondel, Teboul, Berthet, & Djolonga, 2020).

Top-k selection. Given a set of items and item values that must be predicted, the task is to choose the subset of size k with the largest total value in selected items. In addition to $[0, 1]$ bounds on the indicator variables, a single linear constraint ensures that the selected item indicators sum to k. A prevalent example can be found in multilabel classification (Amos, Koltun, & Kolter, 2019; Martins & Astudillo, 2016).

Computing the maximum. This is a special case of top-k selection where $k = 1$. When the LP's objective is regularized with the entropy term $H(\mathbf{x}) = \mathbf{x}^\top \log \mathbf{x}$, the mapping

from predicted values to optimal solutions is equivalent to a softmax function (Agrawal et al., 2019a).

Max-flow/ min-cut. Given a network with predefined source and sink nodes, and predicted flow capacities on each arc, the task is to find the maximum flow rate that can be channeled from source to sink. Here the predicted flow capacities occupy the right-hand side of the linear constraints, which is not in line with the DFL problem formulation introduced in subsection 2.1. However, in the min-cut problem—which is the dual linear program of the max-flow problem—the flow capacities are the parameters in the objective function. The max-flow problem can thus be cast as an equivalent min-cut problem allowing DFL techniques to predict the flow capacities.

2.4.3 Integer Linear Programming

Integer Linear Programs (ILPs) are another mainstay in OR and AI research. ILPs differ from LPs in that the decision variables $\mathbf{x}$ are restricted to integer values, i.e., $\mathbf{x} \in \mathbb{Z}^k$ where $\mathbb{Z}^k$ is the set of integral vectors of appropriate dimensions. Like LPs, ILPs are challenging to use in DFL because they yield discontinuous, nondifferentiable mappings. Computationally however, they are more challenging due to their NP-hard complexity, which may preclude the exact computation of the mapping $\hat{\mathbf{c}} \rightarrow \mathbf{x}^\star(\hat{\mathbf{c}})$ at each step of Algorithm 2. Their differentiation is also significantly more challenging, since the discontinuity of their feasible regions prevents many smoothing techniques that can be applied in DFL with LPs.

The following examples include ILPs in *Predict-Then-Optimize* problems.

Knapsack. The knapsack problem (4), discussed earlier, has been used to compose benchmark problems in several papers about DFL (Mandi, Demirović, Stuckey, & Guns, 2020; Mandi & Guns, 2020; Demirović et al., 2019). Given are weights of a set of items, as well as a capacity. The items also have associated values, which have to be predicted from features. The optimization task involves selecting a subset of the items that maximizes the value of the selected items, whilst ensuring that the sum of the associated weights does not exceed the capacity.

Travelling salesperson problem. In the travelling salesperson problem, the list of cities, and the distances between each pair of cities, is given. The goal is to find a path of minimal length that visits each city exactly once. In the *Predict-Then-Optimize* setting, the distances between the cities first have to be predicted (Pogančić et al., 2020) from observable empirical data.

Combinatorial portfolio optimization. Portfolio optimization involves making optimal investment decisions across a range of financial assets. In the combinatorial *Predict-Then-Optimize* variant, the decisions are discrete, and must be made on the basis of the predicted next period's increase in the value of several assets (Ferber, Wilder, Dilkina, & Tambe, 2020).

Diverse bipartite matching. Diverse bipartite matching problems are similar to the bipartite matching problems described in 2.4.2, but are subject to additional diversity constraints (Ferber et al., 2020; Mulamba, Mandi, Diligenti, Lombardi, Bucarey, & Guns, 2021; Mandi, Bucarey, Tchomba, & Guns, 2022). In this variant, edges have

additional properties. The diversity constraints enforce lower and upper bounds on the proportion of edges selected with a certain property. These changes preclude an LP formulation, by breaking the total unimodularity property possessed by the standard bipartite matching LP.

Energy-cost aware scheduling. Energy-cost aware scheduling involves scheduling a set of tasks across a set of machines minimizing the overall energy cost involved. As future energy costs are unknown, they first have to be predicted (Mandi et al., 2020).

2.4.4 Integer Nonlinear Programming

In integer nonlinear programming, variables are defined over an integer domain whiles the objective function and/or the constraints are nonlinear. Performing DFL on integer nonlinear programs faces similar challenges as in performing DFL on ILPs: their implicitly defined mappings $\mathbf{x}^\star(\hat{\mathbf{c}})$ result in zero-valued gradients almost everywhere. Additionally, because of their nonlinear nature, many of the techniques developed for DFL with ILPs, which assume linearity, do not directly translate to integer nonlinear programs (Elmachtoub & Grigas, 2022; Pogančić et al., 2020). To the best of our knowledge, no DFL technique has specifically been developed for or tested on integer nonlinear programs. The most closely related work is (Ferber, Huang, Zha, Schubert, Steiner, Dilkina, & Tian, 2023b), which employs approximate ILP surrogates for integer nonlinear programs, which are then used to compose a DFL training procedure using known techniques for differentiation through the ILP surrogate model.

3. Review of Decision-Focused Learning Methodologies

Gradient-based learning is a popular learning approach in ML, and likewise, gradient-based DFL techniques have been extensively studied, with the added benefit that (deep) neural networks can be used as the underlying ML model. However, other ML frameworks, such as tree-based methods or search-based methods, do not require gradients at all and can side-step the issue of zero-valued gradients altogether. Hence, DFL methodologies can be classified into two broad categories: **I.** Gradient-based DFL, and **II.** Gradient-free DFL. First, gradient-based DFL will be surveyed, followed by gradient-free DFL.

3.1 Review of Gradient-Based DFL Methodologies

This subsection will describe several DFL techniques which address the challenge of differentiating an optimization mapping for DFL in gradient-based training. In essence, different approaches propose different smoothed surrogate approximations of $\frac{d\mathbf{x}^\star(\hat{\mathbf{c}})}{d\hat{\mathbf{c}}}$ or $\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\hat{\mathbf{c}}}$, which is used for backpropagation. This paper proposes the first categorization of existing gradient-based DFL techniques into the following four distinct classes:

Analytical Differentiation of Optimization Mappings: Techniques in this category aim to compute exact derivatives for backpropagation by differentiating the optimality conditions for certain optimization problem forms, for which the derivative exists and is non-zero.

Utilize

	Objective Function	Constraint Functions	Decision Variables	CO Solver
Analytical Differentiation of Optimization Mappings	Strictly Convex	Convex	Continuous	Primal-Dual Solver
Analytical Smoothing of Optimization Mappings	Linear	Linear	Continuous/ Discrete	Primal-Dual Solver
Smoothing by Random Perturbations	Linear in the Predicted Parameter	Not limited to specific form	Continuous/ Discrete	Solver agnostic
Differentiation of Surrogate Loss Functions	Linear in the Predicted Parameter	Not limited to specific form	Continuous/ Discrete	Solver agnostic

Figure 4: An overview of gradient-based DFL methodologies categorized into four classes.

Analytical Smoothing of Optimization Mappings: These approaches deal with combinatorial optimization problems (for which the analytical derivatives are zero almost everywhere) by performing smoothing of combinatorial optimization problems, which results in approximate problems that can be differentiated analytically.

Smoothing by Random Perturbations: Techniques under this category utilize implicit regularization through perturbations, constructing smooth approximations of optimization mappings.

Differentiation of Surrogate Loss Functions: Techniques under this category propose surrogate loss functions of specific task loss such as regret. These surrogate losses reflect the quality of the decisions and they provide easy-to-compute gradients or subgradients for gradient-based training.

Figure 4 presents key characteristics of these four methodology classes, highlighting the types of problems that can be addressed within each class. Next, each category is thoroughly described.

3.1.1 Analytical Differentiation of Optimization Mappings

As discussed before, differentiating through parametric CO problems comes with two main challenges. First, since CO problems are complex, implicitly defined mappings from parameters to solutions; computing the derivatives is not straightforward. Second, since some

CO problems result in piecewise-constant mappings, their derivatives are zero almost everywhere, and do not exist elsewhere.

This subsection pertains to CO problems for which the second challenge does not apply, i.e., problems that are smooth mappings. For these problems, all that is required to implement DFL is direct differentiation of the mapping in Eq. (1).

Differentiating unconstrained relaxations. An early work discussing differentiation through constrained argmin problems in the context of machine learning is (Gould, Fernando, Cherian, Anderson, Cruz, & Guo, 2016). It first proposes a technique to differentiate the argmin of a smooth, *unconstrained* convex function. When $V(\mathbf{c}) = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{c}, \mathbf{x})$, it can be shown that when all second derivatives of f exist,

$$\frac{dV(\mathbf{c})}{d\mathbf{c}} = -\frac{f_{\mathbf{cx}}(\mathbf{c}, V(\mathbf{c}))}{f_{\mathbf{xx}}(\mathbf{c}, V(\mathbf{c}))} \tag{9}$$

where $f_{\mathbf{cx}}$ is the second partial derivative of f with respect to $\mathbf{c}$ followed by $\mathbf{x}$. This follows from implicit differentiation of the first-order optimality conditions

$$\frac{d}{d\mathbf{x}} f(\mathbf{c}, V(\mathbf{c})) = 0 \tag{10}$$

with respect to $\mathbf{c}$, and rearranging terms. Here the variables $\mathbf{c}$ are the optimization problem's defining parameters, and the variables $\mathbf{x}$ are the decision variables.

This technique is then extended to find approximate derivatives to *constrained* optimization problems with inequality constraints $g_i(\mathbf{c}, \mathbf{x}) \leq 0$, by first relaxing the problem to an unconstrained problem, by means of the log-barrier function

$$F(\mathbf{c}, \mathbf{x}) = f(\mathbf{c}, \mathbf{x}) - \mu \sum_{i}^{M_{in}} \log(-g_i(\mathbf{c}, \mathbf{x})) \tag{11}$$

where M_{in} is the number of inequality constraints. Then $\operatorname{argmin}_{\mathbf{x}} F(\mathbf{c}, \mathbf{x})$ is differentiated with respect to $\mathbf{c}$ for some choice of the scaling factor μ. Since this approach relies on approximations and requires hyperparameter tuning for the factor μ, subsequent works focus on differentiating CO problems directly via their own global conditions for optimality, as discussed next.

Differentiating KKT conditions of quadratic programs More recent approaches are based on differentiating the optimality conditions of a CO problem directly, i.e., without first converting it to an unconstrained problem. Consider an optimization problem and its optimal solution:

$$\mathbf{x}^\star = \underset{\mathbf{x}}{\operatorname{argmin}} \;\; f(\mathbf{x}) \tag{12a}$$

$$\texttt{s.t.} \;\; \boldsymbol{g}(\mathbf{x}) \leq 0 \tag{12b}$$

$$\boldsymbol{h}(\mathbf{x}) = 0 \tag{12c}$$

and assume that f, g and h are differentiable functions of $\mathbf{x}$. The Karush–Kuhn–Tucker (KKT) conditions are a set of equations expressing optimality conditions for a solution $\mathbf{x}^\star$

of problem (12) (Boyd & Vandenberghe, 2014):

$$\nabla f(\mathbf{x}^\star) + \sum_i w_i \nabla h_i(\mathbf{x}^\star) + \sum_j u_j \nabla g_j(\mathbf{x}^\star) = 0 \tag{13a}$$

$$g_j(\mathbf{x}^\star) \leq 0 \ \ \forall j \tag{13b}$$

$$h_i(\mathbf{x}^\star) = 0 \ \ \forall i \tag{13c}$$

$$u_j \geq 0 \ \ \forall j \tag{13d}$$

$$u_j g_j(\mathbf{x}^\star) = 0 \ \ \forall j \tag{13e}$$

OptNet is a framework developed by Amos and Kolter (2017) to differentiate through optimization mappings that are convex quadratic programs (QPs) by differentiating through these KKT conditions. In convex quadratic programs, the objective f is a convex quadratic function and the constraint functions g, h are linear over a continuous domain. In the most general case, each of f, g and h are dependent on a distinct set of parameters, in addition to the optimization variable $\mathbf{x}$:

$$f(\mathbf{c}, Q, \mathbf{x}) = \frac{1}{2}\mathbf{x}^\top Q\mathbf{x} + \mathbf{c}^\top \mathbf{x} \tag{14a}$$

$$h(A, \mathbf{b}, \mathbf{x}) = A\mathbf{x} - \mathbf{b} \tag{14b}$$

$$g(R, \mathbf{s}, \mathbf{x}) = R\mathbf{x} - \mathbf{s} \tag{14c}$$

When $\mathbf{x} \in \mathbb{R}^n$ and the number of inequality and equality constraints are M_{in} and M_{eq}, respectively, a QP problem is specified by parameters $Q \in \mathbb{R}^{n \times n}$, $\mathbf{c} \in \mathbb{R}^n$, $R \in \mathbb{R}^{n \times M_{in}}$, $\mathbf{s} \in \mathbb{R}^{M_{in}}$, $A \in \mathbb{R}^{n \times M_{eq}}$, and $\mathbf{b} \in \mathbb{R}^{M_{eq}}$. The optimal solution $\mathbf{x}^\star$ can be implicitly differentiated with respect to each of the parameters $(Q, \mathbf{c}, R, \mathbf{s}, A, \mathbf{b})$, by directly differentiating the KKT conditions of optimality (13). This produces a linear system of equations, which can be solved for the desired gradients $\frac{d\mathbf{x}^\star}{d\mathbf{c}}$. Later, Konishi and Fukunaga (2021) extended the technique of Amos and Kolter (2017), by also computing the QP problem's second order derivatives. This enables training with gradient boosting models, which require the gradient as well as the Hessian matrix of the loss.

Differentiating optimality conditions of conic programs. Another class of problems with a parametric canonical form are the conic programs, which take the form:

$$\mathbf{x}^\star(A, \mathbf{b}, \mathbf{c}) = \underset{\mathbf{x}}{\operatorname{argmin}} \ \ \mathbf{c}^\top \mathbf{x} \tag{15a}$$

$$\texttt{s.t.} \ \ A\mathbf{x} - \mathbf{b} \in \mathcal{K} \tag{15b}$$

where $\mathcal{K}$ is a nonempty, closed, convex cone.

A framework for differentiating the mapping (15) for any $\mathcal{K}$ is proposed by Agrawal, Barratt, Boyd, Busseti, and Moursi (2019b), which starts by forming the homogeneous self-dual embedding of (15), whose parameters form an askew-symmetric block matrix composed of A, $\mathbf{b}$, and $\mathbf{c}$. Following the technique proposed by Busseti, Moursi, and Boyd (2019), the solution to this embedding is expressed as the problem of finding a zero of a mapping containing a skew-symmetric linear function and projections onto the cone $\mathcal{K}$ and its dual. The zero-value of this function is implicitly differentiated, in a similar manner to the KKT

conditions of a quadratic program (Amos & Kolter, 2017). The overall mapping (15) is viewed as the composition of a function that maps $(A, \mathbf{b}, \mathbf{c})$ onto the skew-symmetric parameter space of the self-dual embedding, the rootfinding problem that produces a solution to the embedding, and a transformation back to a solution of the primal and dual problems. The overall derivative is found by a chain rule applied over this composition.

Subsequent work by Agrawal et al. (2019a) leverages the above-described differentiation of cone programs to develop a more general differentiable convex optimization solver—*cvxpylayers*. It is well known that conic programs of the form (15) can provide canonical representations of convex programs (Nemirovski, 2007). The approach described by Agrawal et al. (2019a) is based on this principle; that a large class of *parametric* convex optimization problems can be recast as equivalent parametric cone programs, with an appropriate choice of the cone $\mathcal{K}$. A major benefit of this representation is that it allows a convex program to be separated with respect to its defining parameters $(A, \mathbf{b}, \mathbf{c})$ and its structure $\mathcal{K}$, allowing a generic procedure to be applied for solving and differentiating the transformed problem with respect to A, $\mathbf{b}$ and $\mathbf{c}$.

To transform convex programs to cone programs (15), the framework of Grant and Boyd (2008) is utilized, which is based on two related concepts. First is the notion of *disciplined convex programming*, which assists the automation of cone transforms by imposing a set of rules or conventions on how convex programs can be represented. Second is the notion of *graph implementations*, which represent functions as optimization problems over their epigraphs, for the purpose of generically representing optimization problems and assisting conversion between equivalent forms. The associated software system called `cvx` allows for disciplined convex programs to be converted to cone programs via their graph implementations. Subsequently, the transformed problem is solved using conic optimization algorithms, and its optimal solution is converted to a solution of the original disciplined convex program. Differentiation is performed through each operation and combined by the chain rule. The transformation of parameters between respective problem forms, and the solution recovery step, are differentiable by virtue of being affine mappings (Agrawal et al., 2019a). The intermediate conic program is differentiated via the methods of Agrawal et al. (2019b).

Solver unrolling and fixed-point differentiation. While the methods described above for differentiation through CO problems are generic and applicable to broad classes of problems, other practical techniques have been proven effective and even advantageous in some cases. A common strategy is that of solver *unrolling*, in which the solution to (1) is found by executing an iterative optimization method on the computational graph of its preceding predictive model. The optimization mapping (1) is then backpropagated simply by automatic differentiation through each step of the algorithm, thus avoiding the need to explicitly model $\frac{d\mathbf{x}^\star(\mathbf{c})}{d\mathbf{c}}$ (Domke, 2012). While this approach leads to accurate backpropagation in many cases, it suffers disadvantages in efficiency due to the memory and computational resources required to store and apply backpropagation over the entire computational graph of an algorithm that requires many iterations (Amos & Kolter, 2017). A comprehensive survey of algorithm unrolling in image processing applications is provided by Monga, Li, and Eldar (2021).

Another way in which a specific solution algorithm may provide gradients though a corresponding optimization mapping, is by implicit differentiation of its fixed-point conditions.

Suppose that the solver iterations

$$\mathbf{x}_{t+1}(\mathbf{c}) = \mathcal{U}(\mathbf{x}_t(\mathbf{c}),\ \mathbf{c}) \tag{16}$$

converge as $t \to \infty$ to a solution $\mathbf{x}^\star(\mathbf{c})$ of the problem (1), then the fixed-point conditions

$$\mathbf{x}^\star(\mathbf{c}) = \mathcal{U}(\mathbf{x}^\star(\mathbf{c}),\ \mathbf{c}) \tag{17}$$

are satisfied. Assuming the existence of all derivatives on an open set containing $\mathbf{c}$ to satisfy the implicit function theorem, it follows by implicit differentiation with respect to $\mathbf{c}$ that

$$(\mathbf{I} - \Phi)\frac{d\mathbf{x}^\star}{d\mathbf{c}} = \Psi, \tag{18}$$

which is a linear system to be solved for $\frac{d\mathbf{x}^\star}{d\mathbf{c}}$, in terms of $\Phi = \frac{d\mathcal{U}}{d\mathbf{x}^\star}(\mathbf{x}^\star(\mathbf{c}),\ c)$, $\Psi = \frac{d\mathcal{U}}{d\mathbf{c}}(\mathbf{x}^\star(\mathbf{c}),\ \mathbf{c})$ and identity matrix $\mathbf{I}$.
The relationship between unrolling and differentiation of the fixed-point conditions is studied by Kotary, Dinh, and Fioretto (2023b), showing that backpropagation of (1) by unrolling (16) is equivalent to solving the linear system (18) by fixed-point iteration. The convergence rate of the backward pass in unrolling is determined by that of solving the linear system (18) in such a manner, and can be calculated in terms of the spectral radius of Φ.

Discussion. In contrast to most other differentiable optimization methods surveyed in this survey paper, the analytical approaches in this subsection allow for the backpropagation of coefficients that specify the constraints as well as the objective function. For example, Amos and Kolter (2017) propose parametric quadratic programming layers whose linear objective parameters are predicted by previous layers, and whose constraints are learned through the layer's own embedded parameters. This is distinct from most cases of DFL, in which the CO problems have fixed constraints and no trainable parameters of their own.

Furthermore, the techniques surveyed in this subsection are aimed at computing the exact gradients of parametric optimization mappings. However, many applications of DFL contain optimization mappings that are discontinuous and piecewise-constant. Such mappings, including parametric linear programs (8), have gradients that are zero almost everywhere and thus do not supply useful descent directions for SGD training. Therefore, the techniques of this subsection are often applied after regularizing the problem analytically with smooth functions, as detailed in the next subsection.

3.1.2 Analytical Smoothing of Optimization Mappings

To differentiate through combinatorial optimization problems, the optimization mapping first has to be smoothed. While techniques such as noise-based gradient estimation (surveyed in Section 3.1.3) provide smoothing and differentiation simultaneously, analytical differentiation first incorporates smooth analytical terms in the optimization problem's formulation, and then analytically differentiates the resulting optimization problem using the techniques discussed in Section 3.1.1.

Analytical smoothing of linear programs. Note that while an LP problem is convex and has continuous variables, only a finite number of its feasible solutions can potentially be optimal. These points coincide with the vertices of its feasible polytope (Bazaraa et al., 2008). Therefore the mapping $\mathbf{x}^\star(\hat{\mathbf{c}})$ in (8), as a function of $\hat{\mathbf{c}}$, is discontinuous and piecewise constant, and thus requires smoothing before it can be differentiated through. An approach to do so was presented in Wilder et al. (2019a), which proposes to augment the linear LP objective function with the Euclidean norm of its decision variables, so that the new objective takes the following form

$$\mathbf{x}^\star(\mathbf{c}) = \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{c}^\top \mathbf{x} + \mu \|\mathbf{x}\|_2^2 \tag{19a}$$

$$= \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{x} - \left(\frac{-\mathbf{c}}{\mu}\right)\|_2^2 \tag{19b}$$

where the above equality follows from expanding the square and cancelling constant terms, which do not affect the argmax. This provides an intuition as to the effect of such a quadratic regularization: it converts a LP problem into that of projecting the point $\left(\frac{-\mathbf{c}}{\mu}\right)$ onto the feasible polytope, which results in a continuous mapping $\mathbf{c} \to \mathbf{x}^\star(\mathbf{c})$. Wilder et al. (2019a) then train models in decision-focused paradigm by solving and backpropagating the respective quadratic programming problem using the OptNet framework (Amos & Kolter, 2017), in order to learn to predict objective parameters with minimal regret. At test time, the quadratic smoothing term is removed. Wilder et al. (2019a) refers to such regret-based DFL with quadratically regularized linear programs as the *Quadratic Programming Task Loss* method (QPTL).

Other forms of analytical smoothing for linear programs can be applied by adding different regularization functions to the objective function. Other regularization terms for LPs include the entropy function $H(\mathbf{x}) = \sum_i x_i \log x_i$ and the binary entropy function $H_b(\mathbf{x}) = H(\mathbf{x}) + H(\mathbf{1} - \mathbf{x})$. To differentiate the resulting smoothed optimization problems, the framework of Agrawal et al. (2019a) can be used. Alternatively, problem-specific approaches, without employing this framework, have also been proposed. For example, Blondel et al. (2020) propose a method for problems where H smooths an LP for differentiable sorting and ranking, and Amos et al. (2019) propose a way to differentiate using H_b for multilabel classification problems. Both works propose fast implementations for both the forward and backward passes of the respective optimization problems.

In a related approach, Mandi and Guns (2020) propose a general, differentiable LP solver based on log-barrier regularization. For a parametrized LP of standard form (8), gradients are computed for the regularized form in which the constraints $\mathbf{x} \geq \mathbf{0}$ are replaced with log-barrier approximations as follows:

$$\mathbf{x}^\star(\mathbf{c}) = \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{c}^\top \mathbf{x} - \mu \sum_{i=1}^{\dim(\mathbf{x})} \log(x_i) \tag{20a}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{20b}$$

While this method, in this sense, is similar to the approach of Gould et al. (2016), it exploits several efficiencies specific to linear programming, in which the log-barrier term serves a dual purpose of rendering (20) differentiable and also aiding its solution (Mandi

& Guns, 2020). Rather than forming and solving this regularized LP problem directly, the solver uses an interior point method to produce a sequence of log-barrier approximations to the LP's homogenous self-dual (HSD) embedding. Early stopping is applied in the interior point method, producing a solution to (20) for some μ, which serves as a smooth surrogate problem for differentiation. A major advantage of this technique is that it only requires optimization of a linear program, making it in general more efficient than direct solution of a regularized problem as in the approaches described above.

Analytical smoothing of integer linear programs. To differentiate through ILPs, Wilder et al. (2019a) propose to simply drop the integrality constraints, and to then smooth and differentiate through the resulting LP relaxation, which is observed to give satisfactory performance in some cases. Ferber et al. (2020) later extended this work by using a more systematic approach to generate the LP relaxation of the ILP problem. They use the method of cutting planes to discover an LP problem that admits the same solution as the ILP. Subsequently, the method of Wilder et al. (2019a) is applied to approximate the LP mapping's derivatives. Although this results in enhanced performance with respect to regret, there are some practical scalability concerns, since the cut generation process is time consuming but also must be repeated for each instance in each training epoch.

Although this subsection primarily surveys the smoothing of LPs and ILPs, as these are the focal points of this paper, note that differentiable relaxation of other optimization problems has also received attention recently. For example, differential relaxations of MAXSAT (Wang, Donti, Wilder, & Kolter, 2019; Wang, Zhang, Guo, Chen, Yang, & Yan, 2023) and submodular optimization problems (Djolonga & Krause, 2017; Tschiatschek, Sahin, & Krause, 2018) have been developed to embed these problems into neural networks.

3.1.3 Smoothing by Random Perturbations

A central challenge in DFL is the need for smoothing operations of non-smooth optimization mappings. In contrast to the DFL techniques surveyed in the previous subsection, which perform the smoothing operation by adding explicit regularization functions to the objective functions of optimization problems, this subsection focuses on techniques that use implicit regularization through perturbations. These techniques construct smooth approximations of the optimization mappings by adopting a probabilistic point of view. To introduce this point of view, the CO problem in this section is not viewed as a mapping from $\mathbf{c}$ to $\mathbf{x}^\star(\mathbf{c})$. Rather, it is viewed as a function that maps $\mathbf{c}$ onto a probability distribution over the feasible region $\mathcal{F}$. From this perspective, $\mathbf{x}^\star(\mathbf{c})$ can be viewed as a random variable, conditionally dependent on $\mathbf{c}$. The motivation behind representing $\mathbf{x}^\star(\mathbf{c})$ as a random variable is that the rich literature of likelihood maximization with latent variables, in fields such as Probabilistic Graphical Models (PGMs) (Koller & Friedman, 2009), can be exploited.

Implicit differentiation by perturbation. One seminal work in the field of PGMs is by Domke (2010). This work contains an important proposition, which deals with a setup where a variable $\boldsymbol{\theta}_1$ is conditionally dependent on another variable $\boldsymbol{\theta}_2$ and the final loss $\mathcal{L}$ is defined on the variable $\boldsymbol{\theta}_1$. Let $p(\boldsymbol{\theta_1}|\boldsymbol{\theta_2})$ and $\mathbb{E}[\boldsymbol{\theta_1}|\boldsymbol{\theta_2}]$ be the conditional distribution and the conditional mean of $\boldsymbol{\theta}_1$. The loss $\mathcal{L}$ is measured on the conditional mean $\mathbb{E}[\boldsymbol{\theta_1}|\boldsymbol{\theta_2}]$ and the goal is to compute the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\theta}_2$. Domke (2010) proposes that the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\theta}_2$ can be approximated by the following finite difference

method:

$$\frac{dL}{d\boldsymbol{\theta_2}} \approx \frac{1}{\delta}\left(\mathbb{E}[\boldsymbol{\theta_1}|(\boldsymbol{\theta_2} + \delta\frac{d}{d\boldsymbol{\theta_1}}(\mathcal{L}(\mathbb{E}[\boldsymbol{\theta_1}|\boldsymbol{\theta_2}]))] - \mathbb{E}[\boldsymbol{\theta_1}|\boldsymbol{\theta_2}]\right) \tag{21}$$

where $\frac{d}{d\boldsymbol{\theta_1}}[\mathcal{L}(\mathbb{E}[\boldsymbol{\theta_1}])]$ is the derivative $\mathcal{L}$ with respect to $\boldsymbol{\theta}_1$ at $\mathbb{E}[\boldsymbol{\theta_1}]$. Notice that the first term in (21) is the conditional mean after perturbing the parameter $\boldsymbol{\theta}_2$ where the magnitude of the perturbation is modulated by the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\theta}_1$. Taking inspiration from this proposition, by defining a conditional distribution $p(\mathbf{x}^\star(\hat{\mathbf{c}})|\hat{c})$, one can compute the derivative of the regret with respect to $\hat{\mathbf{c}}$ in the context of DFL.

To perfectly represent the deterministic mapping $\mathbf{c} \to \mathbf{x}^\star(\mathbf{c})$, the straightforward choice is to define a Dirac mass distribution, which assigns all probability mass to the optimal point and none to other points, i.e.,

$$p(\mathbf{x}|\mathbf{c}) = \begin{cases} 1 & \mathbf{x} = \mathbf{x}^\star(\mathbf{c}) \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

Differentiation of blackbox combinatorial solvers (DBB). Note that with the distribution in (22) $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}|\mathbf{c})}[x|c] = \mathbf{x}^\star(\mathbf{c})$. Hence, using conditional probability in the proposition in (21), $\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\hat{\mathbf{c}}}$ can be approximated in the following way:

$$\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\hat{\mathbf{c}}} \approx \nabla^{(DBB)}\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}})) = \left(\mathbf{x}^\star\Big(\hat{\mathbf{c}} + \delta\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\mathbf{x}^\star(\hat{\mathbf{c}})}\Big) - \mathbf{x}^\star\Big(\hat{\mathbf{c}}\Big)\right) \tag{23}$$

The gradient computation technique proposed by Pogančić et al. (2020) takes the form of (24). They interpret it as substituting the jump-discontinuous optimization mapping with a piece-wise linear interpolation. It is a linear interpolation of the mapping $\hat{\mathbf{c}} \to \mathbf{x}^\star(\hat{\mathbf{c}})$ between the points $\hat{\mathbf{c}}$ and $\hat{\mathbf{c}} + \delta\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\mathbf{x}}|_{\mathbf{x}=\mathbf{x}^\star(\hat{\mathbf{c}})}$. Pogančić et al. (2020) call this 'differentiation of blackbox' (DBB) solvers, because this approach considers the CO solver as a blackbox oracle, i.e., it does not take into consideration how the solver works internally.

In a subsequent work, Sahoo, Paulus, Vlastelica, Musil, Kuleshov, and Martius (2023) propose to treat the CO solver as as a negative identity matrix while backpropagating the loss, i.e., they propose the following approximation of the derivative:

$$\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\hat{\mathbf{c}}} \approx -\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\mathbf{x}^\star(\hat{\mathbf{c}})} \tag{24}$$

However, they notice that such an approach might run into unstable learning for scale-invariant optimization problems such as LPs and ILPs. To negate this effect, they suggest multiplying the cost vector with the matrix of the invariant transformation. In the case of LPs and ILPs this can be achieved by **normalizing the cost vector through projection onto the unit sphere**.

Perturb-and-MAP. However, at this point it is worth mentioning that Domke (2010) assumes, in his proposition, that the distribution $p(\theta_1|\theta_2)$ in (21) belongs to the exponential family of distributions (Barndorff-Nielsen, 1978). Note that the distribution defined in (22) is not a distribution of the exponential family. Nevertheless, a tempered softmax (Hinton,

Vinyals, & Dean, 2015) distribution belonging to exponential family can be defined to express the mapping in the following way:

$$p_\tau(\mathbf{x}|\mathbf{c}) = \begin{cases} \frac{\exp(-f(\mathbf{x},\mathbf{c})/\tau)}{\sum_{\mathbf{x}' \in \mathcal{F}} \exp(-f(\mathbf{x}',\mathbf{c})/\tau)} & \mathbf{x} \in \mathcal{F} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

In this case, the log unnormalized probability mass at each $\mathbf{x} \in \mathcal{F}$ is proportional to $\exp(-f(\mathbf{x}, \mathbf{c})/\tau)$, the exponential of the negative of the tempered objective value. The idea behind (25) is to assign a probability to each feasible solution such that solutions with a better objective value have a larger probability. The parameter τ affects the way in which objective values map to probabilities. When $\tau \to 0$, the distribution becomes the argmax distribution in (22), when $\tau \to \infty$, the distribution becomes uniform. In other words, the value of τ determines how drastically the probability changes because of a change in objective value. Good values for τ are problem-dependent, and thus tuning τ is advised.

Note that (21) deals with conditional expectation. As in the case of the tempered softmax distribution, the conditional expectation is not always equal to the solution to the CO problem, it must be computed first to use the finite difference method in (21). However, computing the probability distribution function in (25) is not tractable, as the denominator (also called the partition function) requires iterating over all feasible points in $\mathcal{F}$. Instead, Papandreou and Yuille (2011) propose a novel approach, known as *perturb-and-MAP*, to estimate the probability using perturbations. It states that the distribution of the maximizer after perturbing the log unnormalized probability mass by i.i.d. Gumbel$(0, \epsilon)$ noise has the same exponential distribution as (25). To make it more explicit, if $\tilde{\mathbf{c}} = \mathbf{c} + \boldsymbol{\eta}$, where the perturbation vector $\boldsymbol{\eta} \overset{\text{i.i.d.}}{\sim}$ Gumbel$(0, \epsilon)$,

$$\mathbb{P}[\mathbf{x} = \underset{\mathbf{x}'}{\operatorname{argmax}} -f(\mathbf{x}', \tilde{\mathbf{c}})] = p_\epsilon(\mathbf{x}|\mathbf{c}) \tag{26}$$

The perturb-and-MAP framework can be viewed as a method of stochastic smoothing (Abernethy, Lee, & Tewari, 2016). A smoothed approximation of the optimization mapping is created by considering the average value of the solutions of a set of *nearby perturbed* points. With the help of (26), the conditional distribution and hence the conditional mean can be approximated by Monte Carlo simulation.

Differentiable perturbed optimizers. Berthet, Blondel, Teboul, Cuturi, Vert, and Bach (2020) propose another approach for perturbation-based differentiation. They name it differentiable perturbed optimizers (DPO). They make use of the perturb-and-MAP framework to draw samples from the conditional distribution $p(\mathbf{x}|\mathbf{c})$. In particular, they use the reparameterization trick (Kingma & Welling, 2014; Rezende, Mohamed, & Wierstra, 2014) to generate samples from $p(\mathbf{x}|\mathbf{c})$. The reparameterization trick uses a change of variables to rewrite $\mathbf{x}$ as a *deterministic function* of $\mathbf{c}$ and a random variable $\boldsymbol{\eta}$. In this reformulation, $\mathbf{x}$ is still a random variable, but the randomness comes from the variable $\boldsymbol{\eta}$. They consider $\boldsymbol{\eta}$ to be a random variable having a density proportional to $\exp(-\nu(\boldsymbol{\eta}))$ for a twice-differentiable function ν. Moreover, they propose to multiply the random variable $\boldsymbol{\eta}$ with a temperature parameter $\epsilon > 0$, which controls the strength of perturbing $\mathbf{c}$ by the random variable $\boldsymbol{\eta}$. In summary, first $\mathbf{c}$ is perturbed with random perturbation vector $\epsilon\boldsymbol{\eta}$, where $\boldsymbol{\eta}$ is sampled from the aforementioned density function, and then the maximizer of

the perturbed vector $c + \epsilon\boldsymbol{\eta}$ is viewed as a sample from the conditional distribution, i.e., $\mathbf{x}_\epsilon^\star(\mathbf{c}) = \mathbf{x}^\star(\mathbf{c} + \epsilon\boldsymbol{\eta})$ is considered as a sample drawn from $p(\mathbf{x}|\mathbf{c})$ for given values of $\mathbf{c}$ and ϵ. They call $\mathbf{x}_\epsilon^\star(\mathbf{c})$ a *perturbed optimizer*. Note that, for $\epsilon \to 0$, $\mathbf{x}_\epsilon^\star(\mathbf{c}) \to \mathbf{x}^\star(\mathbf{c})$. Like before, $\mathbf{x}_\epsilon^\star(c)$ can be estimated by Monte Carlo simulation by sampling i.i.d. random noise $\boldsymbol{\eta}^{(m)}$ from the aforementioned density function. The advantage is that the Monte Carlo estimate is *continuously* differentiable with respect to $\mathbf{c}$. This Monte Carlo estimate $\bar{\mathbf{x}}_\epsilon^\star(\mathbf{c})$ can be expressed as:

$$\bar{\mathbf{x}}_\epsilon^\star(\mathbf{c}) = \frac{1}{M}\sum_{m=1}^{M} \mathbf{x}^\star\Big(\mathbf{c} + \epsilon\boldsymbol{\eta}^{(m)}\Big) \tag{27}$$

Moreover, its derivative can be estimated by Monte Carlo simulation too

$$\frac{d\bar{\mathbf{x}}_\epsilon^\star(\mathbf{c})}{d\mathbf{c}} = \frac{1}{\epsilon}\frac{1}{M}\sum_{m=1}^{M} \mathbf{x}^\star(\mathbf{c} + \epsilon\boldsymbol{\eta}^{(m)})\nu'(\boldsymbol{\eta}^{(m)})^\top \tag{28}$$

where ν' is the first order derivative of ν. They can approximate $\frac{d\mathbf{x}^\star(\mathbf{c})}{d\mathbf{c}}$ by $\frac{d\bar{\mathbf{x}}_\epsilon^\star(\mathbf{c})}{d\mathbf{c}}$ to implement the backward pass. As mentioned before, if $\epsilon \to 0$, the estimation will be an unbiased estimate of $\mathbf{x}^\star(\mathbf{c})$. However, in practice, for low values of ϵ, the variance of the Monte-Carlo estimator will increase, leading to unstable and noisy gradients. This is in line with the smoothing-versus-accuracy trade-off mentioned before. Berthet et al. (2020) use this DPO framework to differentiate any CO problem with linear objective. For a CO problem with discrete feasible space, they consider the convex hull of the discrete feasible region. Furthermore, Berthet et al. (2020) construct the Fenchel-Young loss function and show for Fenchel-Young loss function, the gradient can be approximated in the following way:

$$\nabla\mathcal{L}^{FY}(\mathbf{x}^\star(\hat{\mathbf{c}})) = -\big(\bar{\mathbf{x}}_\epsilon^\star(\hat{\mathbf{c}}) - \mathbf{x}^\star(\mathbf{c})\big) \tag{29}$$

In a later work, Dalle, Baty, Bouvier, and Parmentier (2022) extend the perturbation approach, where they consider multiplicative perturbation. This is useful when the cost parameter vector is restricted to be non-negative, such as in the applications of shortest path problem variants. The work of Paulus, Choi, Tarlow, Krause, and Maddison (2020) can also be viewed as an extension of the DPO framework. They introduce stochastic softmax tricks (SST), a framework of Gumbel-softmax distributions, where they propose differentiable methods by sampling from more complex categorical distributions.

Implicit maximum likelihood estimation (I-MLE). The *perturb-and-MAP* framework is also used by Niepert, Minervini, and Franceschi (2021). However, they do not sample noise from the Gumbel distribution, rather they report better results when the noise $\boldsymbol{\eta}^\gamma$ is sampled from a *Sum-of-Gamma* distribution with hyperparameter γ. Combining the finite difference approximation (21) with the perturb-and-MAP framework, the gradient takes the following form:

$$\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\hat{\mathbf{c}}} \approx \nabla^{(IMLE)}\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}})) = \left(\mathbf{x}^\star\Big(\hat{\mathbf{c}} + \delta\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\mathbf{x}^\star(\hat{\mathbf{c}})} + \epsilon\boldsymbol{\eta}^\gamma\Big) - \mathbf{x}^\star\Big(\hat{\mathbf{c}} + \epsilon\boldsymbol{\eta}^\gamma\Big)\right) \tag{30}$$

where δ is the step size of the finite difference approximation and $\epsilon > 0$ is a temperature parameter, which controls the strength of noise perturbation. Clearly, (30) turns into (24) when there is no noise perturbation, i.e., if $\boldsymbol{\eta}^\gamma = 0$. A later work (Minervini, Franceschi, & Niepert, 2023) extends I-MLE by adaptively selecting δ based on the ratio between the norm of the parameter $\hat{\mathbf{c}}$ and the norm of $\frac{d\mathcal{L}(\mathbf{x}^\star(\hat{\mathbf{c}}))}{d\mathbf{x}^\star(\hat{\mathbf{c}})}$.

Discussion. One major advantage of the techniques explained in this subsection is that for gradient computation, they call the CO solver as a 'blackbox oracle' and only use the solution returned by it for gradient computation. In essence, these techniques are not concerned with *how* the CO problem is solved. The users can utilize any techniques of their choice—constraint programming (CP) (Rossi, van Beek, & Walsh, 2006), Boolean satisfiability (SAT) (Gomes, Kautz, Sabharwal, & Selman, 2008) or linear programming (LP) and integer linear programming (ILP) to solve the CO problem.

3.1.4 Differentiation of Surrogate Loss Functions

The techniques explained in the preceding subsections can be viewed as implementations of differentiable optimization layers, which solve the CO problem in the forward pass and return useful approximations of $\frac{d\mathbf{x}^\star(\hat{\mathbf{c}})}{d\hat{\mathbf{c}}}$ in the backward pass. Consequently, those techniques can be used to introduce optimization layers *anywhere in a neural network architecture*, and can be combined with arbitrary loss functions. In contrast, the techniques that will be introduced next can only be used to differentiate regret (3)—a specific task loss. Hence, models can only be trained in an end-to-end fashion using these techniques when the CO problem occurs in the *final* stage of the pipeline, as in the case of *Predict-Then-Optimize* problems. Also note that the computation of the regret requires both the ground-truth cost vector $\mathbf{c}$; as well as ground-truth solution $\mathbf{x}^\star(\mathbf{c})$. If $\mathbf{c}$ is observed, $\mathbf{x}^\star(\mathbf{c})$ can be computed. However, if only $\mathbf{x}^\star(\mathbf{c})$ is observed, $\mathbf{c}$ cannot directly be recovered. Hence, the techniques that will be discussed next are not suitable when the true cost vectors $\mathbf{c}$ are not observed in the training data.

Smart "Predict, Then Optimize". Elmachtoub and Grigas (2022) develop Smart "Predict, Then Optimize" (SPO), a seminal work in DFL. As the gradient of the regret with respect to cost vector $\hat{\mathbf{c}}$ is zero almost everywhere, SPO instead uses a surrogate loss function that has subgradients which *are* useful in training. They start by proposing a convex surrogate upper bound of regret, which they call the SPO+ loss.

$$\mathcal{L}_{SPO+}(\mathbf{x}^\star(\hat{\mathbf{c}})) = 2\hat{\mathbf{c}}^\top \mathbf{x}^\star(\mathbf{c}) - \mathbf{c}^\top \mathbf{x}^\star(\mathbf{c}) + \max_{\mathbf{x}\in\mathcal{F}} \{\mathbf{c}^\top \mathbf{x} - 2\hat{\mathbf{c}}^\top \mathbf{x}\} \tag{31}$$

Elmachtoub and Grigas (2022) directly minimize $\mathcal{L}_{SPO+}$ for a linear predictive model. However, they note that directly minimizing $\mathcal{L}_{SPO+}$ is prohibitive for complex predictive models such as NNs. Hence, they derive the following useful subgradient of $\mathcal{L}_{SPO+}(\mathbf{x}^\star(\hat{\mathbf{c}}))$:

$$\mathbf{x}^\star(\mathbf{c}) - \mathbf{x}^\star(2\hat{\mathbf{c}} - \mathbf{c}) \in \partial\mathcal{L}_{SPO+} \tag{32}$$

This subgradient is used in the backward pass for training NNs. While this technique proposes a surrogate loss function, one could also view the solving of $\mathbf{x}^\star(2\hat{\mathbf{c}} - \mathbf{c})$ as solving of $\hat{\mathbf{c}}$ perturbed by the deterministic perturbation vector $\hat{\mathbf{c}} - \mathbf{c}$.

From a theoretical point of view, the SPO+ loss has the Fisher consistency property with respect to the regret under certain distributional assumptions. A surrogate loss function satisfies the Fisher consistency property if the function that minimizes the surrogate loss also minimizes the true loss in expectation (Zou, Zhu, & Hastie, 2008). Concretely, this means that minimizing the SPO+ loss corresponds to minimizing the regret in expectation. While training ML models with a finite dataset, an important property of considerable interest would be *risk bounds* (Massart & Nédélec, 2006). Liu and Grigas (2021) develop risk bounds for SPO+ loss and show that low excess SPO+ loss risk translates to low excess regret risk. Furthermore, El Balghiti, Elmachtoub, Grigas, and Tewari (2019) develop worst-case generalization bounds of the SPO loss.

The SPO framework is applicable not only to LPs, but to *any CO problems where the cost parameters appear linearly in the objective function.* This includes QPs, ILPs and MILPs. Mandi et al. (2020) empirically investigated how the framework performs on ILP problems. However, as these problems are much more computationally expensive to solve than the ones considered by Elmachtoub and Grigas (2022), they compared the standard SPO framework with a variant in which, it is significantly cheaper to solve the CO problem during training. To be specific, they consider LP relaxations of the ILPs. These LP relaxations are obtained by considering the continuous relaxation of the ILPs, i.e., they are variants of the ILPs in which the integrality constraints are dropped. Using the LP relaxations significantly expedite training, without any cost: Mandi et al. (2020) did not observe a significant difference in the final achieved regret between these two approaches, with both of them performing better than the prediction-focused approach. However, one should be cautious to generalize this result across different problems, as it might be dependent on the integrality gap between the ILP and its LP relaxation.

Next, within this category, a different type of DFL techniques is being surveyed. In these DFL techniques, the surrogate loss functions are supposed to reflect the decision quality, but their computations do *not* involve solving the CO problems, thereby avoiding the zero-gradient problem.

Noise contrastive estimation. One such approach is introduced by Mulamba et al. (2021). Although their aim is still to minimize regret, computation of $\nabla_{\hat{\mathbf{c}}} \mathit{Regret}\ (\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c})$ has been avoided by using a surrogate loss function. In their work, the CO problem is viewed from a probabilistic perspective, as in (25). However, instead of maximum likelihood estimation, the noise contrastive estimation (NCE) (Gutmann & Hyvärinen, 2010) method is adopted. NCE has been extensively applied in many applications such as language modeling (Mnih & Teh, 2012), information retrieval (Huang, He, Gao, Deng, Acero, & Heck, 2013) and entity linking (Gillick, Kulkarni, Lansing, Presta, Baldridge, Ie, & Garcia-Olano, 2019). Its basic idea is to learn to discriminate between data coming from the true underlying distribution and data coming from a noise distribution. In the context of DFL, this involves contrasting the likelihood of ground-truth solution $\mathbf{x}^\star(\mathbf{c})$ and a set of negative examples S. In other words, the following ratio is maximized:

$$\max_{\hat{\mathbf{c}}} \sum_{\mathbf{x}' \in S} \frac{p_\tau(\mathbf{x}^\star(\mathbf{c})|\hat{\mathbf{c}})}{p_\tau(\mathbf{x}'|\hat{\mathbf{c}})} \tag{33}$$

where $\mathbf{x}' \in S$ is a negative example. Because the probability $p_\tau(\mathbf{x}^\star(\mathbf{c})|\hat{\mathbf{c}})$ is defined as in (25), when $\tau = 1$, maximizing (33) corresponds to minimizing the following loss:

$$\mathcal{L}_{NCE}(\hat{\mathbf{c}}, \mathbf{c}) = \sum_{\mathbf{x}' \in S} f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) - f(\mathbf{x}', \hat{\mathbf{c}}) \tag{34}$$

In other words, this approach learns to predict a $\hat{\mathbf{c}}$ for which ground-truth solution $\mathbf{x}^\star(\mathbf{c})$ achieves a good objective value, and for which other feasible solutions $\mathbf{x}'$ achieve worse objective values. Note that when $f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) \leq f(\mathbf{x}', \hat{\mathbf{c}})$ for all $\mathbf{x}' \in \mathcal{F}$, it holds that $\mathbf{x}^\star(\mathbf{c}) = \mathbf{x}^\star(\hat{\mathbf{c}})$, and thus the regret is zero. Also note that computing $\mathcal{L}_{NCE}(\hat{\mathbf{c}}, \mathbf{c})$ does not involve computing $\mathbf{x}^\star(\hat{\mathbf{c}})$, circumventing the zero-gradient problem.

As an alternative to NCE, Mulamba et al. (2021) also introduce a maximum a posteriori (MAP) approximation, in which they only contrast the ground-truth solution with the most probable negative example from S according to the current model:

$$\begin{aligned} \mathcal{L}_{MAP}(\hat{\mathbf{c}}, \mathbf{c}) &= \max_{\mathbf{x}' \in S} f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) - f(\mathbf{x}', \hat{\mathbf{c}}) \\ &= f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) - f(\mathbf{x}', \hat{\mathbf{c}}) \text{ where } \mathbf{x}' = \operatorname*{argmin}_{\mathbf{x} \in S} f(\mathbf{x}, \hat{\mathbf{c}}) \end{aligned} \tag{35}$$

Note that whenever $\mathbf{x}^\star(\hat{\mathbf{c}}) \in S$, it holds that $\mathcal{L}_{MAP}(\hat{\mathbf{c}}, \mathbf{c}) = f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) - f(\mathbf{x}^\star(\hat{\mathbf{c}}), \hat{\mathbf{c}})$. This is also known as *self-contrastive estimation* (SCE) (Goodfellow, 2015) since the ground-truth is contrasted with the most likely output of the current model itself.

Also note that for optimization problems with a linear objective, the losses are $\mathcal{L}_{NCE}(\hat{\mathbf{c}}, \mathbf{c}) = \sum_{\mathbf{x}' \in S} \hat{\mathbf{c}}^\top(\mathbf{x}^\star(\mathbf{c}) - \mathbf{x}')$ and $\mathcal{L}_{MAP}(\hat{\mathbf{c}}, \mathbf{c}) = \hat{\mathbf{c}}^\top(\mathbf{x}^\star(\mathbf{c}) - \mathbf{x}')$, where $\mathbf{x}' = \operatorname{argmin}_{\mathbf{x} \in S} f(\mathbf{x}, \hat{\mathbf{c}})$. In order to prevent the model from simply learning to predict $\hat{\mathbf{c}} = \mathbf{0}$, the following alternate loss functions are proposed for these kinds of problems:

$$\mathcal{L}^{(\hat{\mathbf{c}}-\mathbf{c})}_{NCE}(\hat{\mathbf{c}}, \mathbf{c}) = \sum_{\mathbf{x}' \in S} (\hat{\mathbf{c}} - \mathbf{c})^\top(\mathbf{x}^\star(\mathbf{c}) - \mathbf{x}') \tag{36}$$

$$\mathcal{L}^{(\hat{\mathbf{c}}-\mathbf{c})}_{MAP}(\hat{\mathbf{c}}, \mathbf{c}) = \max_{\mathbf{x}' \in S} (\hat{\mathbf{c}} - \mathbf{c})^\top(\mathbf{x}^\star(\mathbf{c}) - \mathbf{x}') \tag{37}$$

Construction of S. Forming S by sampling points from the feasible region $\mathcal{F}$ is a crucial part of using the contrastive loss functions. To this end, Mulamba et al. (2021) proposes to construct S by caching all the optimal solutions in the training data. That is why they name S as 'solution cache'. While training, more feasible points are gradually added to S by solving for some of the predicted cost vectors. However, in order to avoid computational cost, the solver call is not made for each predicted cost during training. Whether to solve for a predicted cost vector is decided by pure random sampling, i.e., is based on a biased coin toss with probability p_{solve}. Intuitively, the p_{solve} hyperparameter determines the proportion of instances for which the CO problem is solved during training. Experimentally, it has been reported that $p_{solve} = 5\%$ of the time is often adequate, which translates to solving for only 5% of the predicted instances. This translates to reducing the computational cost by approximately 95%, since solving the CO problems represents the major bottleneck in terms of computation time in DFL training.

Technique	CO Problem Forms	Computation of Gradient	Differentiable Optimization Layer
OptNet (Amos & Kolter, 2017)	Convex QPs	Implicit differentiation of KKT conditions	✔
Cvxpylayers (Agrawal et al., 2019a)	Convex problems	Implicit differentiation of HSD of conic programs	✔
Fold-opt (Kotary et al., 2023b)	Convex and nonconvex problems	Implicit differentiation based on unrolling	✔
QPTL (Wilder et al., 2019a)	LPs, ILPs	Implicit differentiation after transforming into QPs by adding regularizer	✔
Intopt (Mandi & Guns, 2020)	LPs, ILPs	Implicit differentiation of HSD of (relaxed) LPs by adding log-barrier relaxation	✔
Mipaal (Ferber et al., 2020)	ILPs	Conversion of ILPs into LPs by method of cutting planes before applying QPTL	✔
DBB (Pogančić et al., 2020)	Optimization problems with a linear objective	Differentiation of linear interpolation of optimization mapping	✔
Negative identity (Sahoo et al., 2023)	Optimization problems with a linear objective	Treating the CO solver as negative identity mapping	✔
I-MLE (Niepert et al., 2021)	Optimization problems with a linear objective	Finite difference approximation with perturb-and-MAP	✔
DPO (Berthet et al., 2020)	Optimization problems with a linear objective	Differentiation of perturbed optimizer	✔
FY (Berthet et al., 2020)	Optimization problems with a linear objective	Differentiation of perturbed Fenchel-Young loss	✘
SPO (Elmachtoub & Grigas, 2022)	Optimization problems with a linear objective	Differentiation of surrogate SPO+ loss	✘
NCE (Mulamba et al., 2021)	Generic optimization problems	Differentiation of surrogate contrastive loss	✘
LTR (Mandi et al., 2022)	Generic optimization problems	Differentiation of surrogate LTR loss	✘
LODL (Shah, Wang, Wilder, Perrault, & Tambe, 2022)	Generic optimization problems	Differentiation of a learned convex local surrogate loss	✘

Table 1: A concise overview of gradient modeling techniques in key DFL techniques that use gradient-based learning.

Approximation of a solver by a solution-cache. Furthermore, Mulamba et al. (2021) propose a solver-free training variant for any DFL technique that treats the CO solver as a blackbox oracle. Such techniques include the aforementioned I-MLE, DBB, SPO. In this solver-free implementation, solving the CO problem is substituted with a cache lookup strategy, where the minimizer within the cache $S \subset \mathcal{F}$ is considered as a proxy for the solution to the CO problem (i.e., the minimizer within $\mathcal{F}$). This significantly reduces the computational cost as solving an CO problem is replaced by a linear search within a limited cache. Such an approximation can be useful in case the CO problem takes long to solve.

DFL as a learning to rank (LTR) problem. In a later work, Mandi et al. (2022) observe that $\mathcal{L}_{NCE}$ (34) can be derived by formulating DFL as a *pairwise learning to rank* task (Joachims, 2002). The learning to rank task consists of learning the implicit order over the solutions in S invoked by the objective function values achieved by the solutions with respect to $\mathbf{c}$. In other words, it involves learning to predict a $\hat{\mathbf{c}}$ that ranks the solutions in S

similarly to how $\mathbf{c}$ ranks them. In the *pairwise* approach, $\mathbf{x}^\star(\mathbf{c})$ and any $\mathbf{x}' \in S$ are treated as a pair and the model is trained to predict $\hat{\mathbf{c}}$ such that the ordering of each pair is the same for $\mathbf{c}$ and $\hat{\mathbf{c}}$. The loss is considered to be zero if $\hat{\mathbf{c}}^\top \mathbf{x}^\star(\mathbf{c})$ is smaller than $\hat{\mathbf{c}}^\top \mathbf{x}'$ by at least a margin of $\Theta > 0$. The pairwise loss is formally defined in the following form:

$$\mathcal{L}_{\text{Pairwise}}(\hat{\mathbf{c}}, \mathbf{c}) = \sum_{\mathbf{x}' \in S} \max\big(0, \Theta + (f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) - f(\mathbf{x}', \hat{\mathbf{c}}))\big) \tag{38}$$

Another loss function is formulated by considering the difference in differences between the objective values at the true optimal $\mathbf{x}^\star(\mathbf{c})$ and non-optimal $\mathbf{x}'$ with $\mathbf{c}$ and $\hat{\mathbf{c}}$ as the parameters.

$$\mathcal{L}_{\text{PairwiseDifference}}(\hat{\mathbf{c}}, \mathbf{c}) = \sum_{\mathbf{x}' \in S} \Big(\big(f(\mathbf{x}^\star(\mathbf{c}), \hat{\mathbf{c}}) - f(\mathbf{x}', \hat{\mathbf{c}})\big) - \big(f(\mathbf{x}^\star(\mathbf{c}), \mathbf{c}) - f(\mathbf{x}', \mathbf{c})\big) \Big)^2 \tag{39}$$

Further, motivated by the *listwise learning to rank task* (Cao, Qin, Liu, Tsai, & Li, 2007), a loss function is proposed by Mandi et al. (2022) where the ordering of all the items in S is considered, rather than the ordering of pairs of items. Cao et al. (2007) define this listwise loss based on a *top-one probability* measure. The top-one probability of an item is the probability of it being the best of the set. Note that such probabilistic interpretation in the context of DFL is already defined in Section 3.1.3. Mandi et al. (2022) make use of the tempered softmax probability defined in (25). Recall that this $p_\tau(\mathbf{x}|\mathbf{c})$ can be interpreted as a probability measure of $\mathbf{x} \in \mathcal{F}$ being the minimizer of $f(\mathbf{x}, \mathbf{c})$ in $\mathcal{F}$ for a given $\mathbf{c}$. However, as mentioned before, direct computation of $p_\tau(\mathbf{x}|\mathbf{c})$ requires iterating over all feasible points in $\mathcal{F}$, which is intractable. Therefore Mandi et al. (2022) compute the probability with respect to $S \subset \mathcal{F}$. This probability measure finally is used to define a listwise loss—the cross-entropy loss between $p_\tau(\mathbf{x}|\mathbf{c})$ and $p_\tau(\mathbf{x}|\hat{\mathbf{c}})$, the distributions obtained for ground-truth $\mathbf{c}$ and predicted $\hat{\mathbf{c}}$. This can be written in the following form:

$$\mathcal{L}_{\text{Listwise}}(\hat{\mathbf{c}}, \mathbf{c}) = \Big(-\frac{1}{|S|} \sum_{\mathbf{x}' \in S} p_\tau(\mathbf{x}'|\mathbf{c}) \log p_\tau(\mathbf{x}'|\hat{\mathbf{c}}) \Big) \tag{40}$$

The main advantage of (34), (35), (38), (39) and (40) is that they are differentiable and can be computed directly by any neural network library via automatic differentiation. Also note that the computation and differentiation of the loss functions are solver-free, i.e., they need not solve the CO problem to compute the loss or its derivative.

Learning efficient surrogate losses. Another research direction without optimization in the loop focuses on minimizing the computational burden of solving the CO problems repeatedly. This is achieved by devising efficiently computable and differentiable surrogate losses that approximate and substitute the true task loss. In this regard, Shah et al. (2022) propose to learn a surrogate of the regret function using parametric local losses. Due to the difficulty of learning a single convex surrogate function to estimate regret, a convex local surrogate is learned for each data point in training, based on other points sampled around these data points. By design, the surrogate losses are automatically differentiable, and thus they eliminate the need for a differentiable optimization solver. In subsequent work, Shah, Wilder, Perrault, and Tambe (2024) extended on this, by improving the sample efficiency

of learning the surrogate losses, and by introducing a manner of producing surrogate losses that are faithful to the true regret also outside of the local neighbourhoods around the training examples.

DFL as a learning to optimize problem. DFL is viewed as an extension of learning to optimize (LtO) by Kotary, Di Vito, Christopher, Van Hentenryck, and Fioretto (2023a). In LtO, the goal is to train an ML model as an approximate CO solver, which maps the parameters of a CO problem to its optimal solution. Kotary et al. (2023a) show that the technique of LtO can be applied to the DFL setting, by treating the feature variables $\mathbf{z}$ as inputs to the LtO model, which predicts $\mathbf{x}^\star(\mathbf{c})$. The resulting LtO models function as joint prediction and optimization models, whose training is consistent with regret minimization on the DFL task.

Discussion

So far, in this subsection, an extensive overview of different gradient-based DFL techniques has been provided. For the ease of the readers, a summary of some of the key DFL techniques, discussed so far, have been provided in Table 1. The second column of Table 1 highlights the form of the CO problem applicable to the technique. Note that although some techniques are generally applicable to any CO problem forms, most techniques have been evaluated so far using CO problems with linear objective functions. The third column summarizes the gradient computation technique. The fourth column indicates whether that particular technique is compatible with any generic task loss. Techniques, termed as implementations of *differentiable optimization layers*, can be embedded in any stage of an NN architecture. The other techniques are applicable where optimization is the final stage of the pipeline (such as in *Predict-Then-Optimize* problem formulations) and a particular loss (most often regret) is used as the task loss.

3.2 Review of Gradient-Free DFL Methodologies

The techniques reviewed so far implement DFL by gradient descent training, which is the go-to approach for training neural networks. Next, we review DFL techniques, which do not rely on gradient-based training. Note that, since these techniques do not utilize gradient-based training, the predictive models in most cases consist of either tree-based methods or explicitly specified models such as linear models.

First of all, as $\mathcal{L}_{SPO+}(\mathbf{x}^\star(\hat{\mathbf{c}}))$ is a convex loss function, it can be minimzed outside gradient-based training if the predicive model is linear. Firstly, since $\mathcal{L}_{SPO+}(\mathbf{x}^\star(\hat{\mathbf{c}}))$ is a convex loss function, it can be minimized using any convex solver without using gradient-based training *if* the predictive model is linear. Elmachtoub, Liang, and McNellis (2020) extend SPO considering the predictive model to be a decision tree or ensemble of decision trees. Such models can be learned by recursive partitioning with respect to the regret directly, and thus do not require the use of the SPO+ surrogate loss function introduced by Elmachtoub and Grigas (2022). Alternatively, the tree learning problem can be posed as a MILP and be solved by an off-the-shelf solver. Jeong, Jaggi, Butler, and Sanner (2022) formulate the problem of minimizing regret as a mixed-integer linear program (MILP) for a linear predictive model. They start from the bilevel optimization formulation, introduced in

(6a) and (6b). First, the transition points where the solution of the lower level program (6b) changes are identified, and then the solution space is exhaustively partitioned, and for each partition the solution is annotated. This paves the way to construct a MILP formulation of the outer program (6a). This MILP problem is solved to learn the parameters $\boldsymbol{\omega}$ of the linear predictive model. The resulting model is guaranteed to be globally optimal, which is not the case for gradient-based methods that might get stuck in a local optimum. However, their method is limited to ML models that are linear and optimization problems that are binary MILPs.

Demirović, Stuckey, Guns, Bailey, Leckie, Ramamohanarao, and Chan (2020) consider linear ML models and represent the objective function of the CO problem as a piece-wise linear function of the ML parameters. In this proposed technique, the ML parameters are updated via a coordinate descent algorithm, where each component of the cost vector is updated at a time to minimize the regret keeping other components fixed. This technique requires identifying the *transition points*, where regret changes, as a function of each component of the cost parameter. Demirović et al. (2020) consider CO problems that can be solved by dynamic programming and identify the transition points using dynamic programming. In a later work, Guler, Demirović, Chan, Bailey, Leckie, and Stuckey (2022) extend this technique by employing a 'divide-and-conquer' algorithm to identify the transition points for CO problems whose objective function is a bilinear function of the decision variables and the predicted parameters. This development generalizes the previous work (Demirović et al., 2020) to cover a much broader class of CO problems and offers a substantial speed improvement. The 'branch & learn' approach (Hu, Lee, Lee, & Zhong, 2022) also extends the approach by Demirović et al. (2020) by developing a recursive algorithm, allowing to tackle CO problems that can be solved by recursion.

3.3 Other Aspects of Decision-Focused Learning

In the following, some aspects related to DFL, that have not yet been discussed in this survey paper so far, will be highlighted. To begin with, it should be noted that certain CO problems may have *multiple non-unique optimal solutions* for a given cost vector. This can occur for an LP when the cost vector of an LP is parallel to one of the faces of the feasible polyhedron. Moreover, problems involving symmetric graphs, whose solution can be transformed into other solutions through automorphisms (Bollobás, 2013), exhibit multiple optimal solutions. If the predicted cost vector has multiple non-unique optimal solutions, each of these solutions result in different values of regret. In such scenarios, Elmachtoub and Grigas (2022) propose to consider the worst-case regret. If the set of optimal solutions of $\hat{\mathbf{c}}$ is be represented by $\mathcal{X}^\star(\hat{\mathbf{c}})$, the worst-case regret can be defined in the following form:

$$\textit{Regret}\ (\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c}) = \max_{\mathbf{x}^\star(\hat{\mathbf{c}}) \in \mathcal{X}^\star(\hat{\mathbf{c}})} f(\mathbf{x}^\star(\hat{\mathbf{c}}), \mathbf{c}) - f(\mathbf{x}^\star(\mathbf{c}), \mathbf{c}) \tag{41}$$

Having addressed the possibility of the presence of multiple non-unique optimal solutions in a CO problem, the focus now turns to other important facets of DFL.

3.3.1 Prediction-Focused vs. Decision-Focused Learning

DFL methodologies are expected to deliver lower regret than a PFL approach in *Predict-Then-Optimize* problems, as the ML model is directly trained to achieve low regret. How-

ever, as discussed before, the implementation of DFL poses significant challenges. In fact, practitioners may be tempted to resort to a PFL approach to circumvent the computational costs associated with DFL, when dealing with real-world *Predict-Then-Optimize* problems. To motivate adoption of DFL methodologies among practitioners, it is crucial to investigate scenarios where DFL methodologies outperform the PFL approach. To this end, Elmachtoub, Lam, Zhang, and Zhao (2023) conduct a theoretical comparison of the limiting distributions of the optimality gaps between the two approaches in the context of stochastic optimization. They show the PFL approach that does not consider the CO problem while training the model asymptotically outperforms the integrated prediction and optimization approach, employed by DFL methodologies, *if* the underlying prediction model is well-specified. This is intuitive, as a well-specified model tends to produce highly accurate predictions, which can contribute to the success of the PFL approach. In such cases, the DFL methodologies might perform worse than PFL since training in DFL involves *approximate* gradients (because the true gradient is zero almost everywhere), whereas the gradient is well-defined for a PFL approach. On the other hand, they show that if the model is not well-specified, a PFL approach performs suboptimally compared to the DFL approach. Hence, it is recommended to use DFL when there exists aleatoric or epistemic uncertainty. As most real-world settings include various sorts of uncertainty—both aleatoric and epistemic—DFL methodologies are expected to outperform the PFL approach. In a separate work, Cameron, Hartford, Lundy, and Leyton-Brown (2022) show that the suboptimality of PFL becomes more pronounced in the presence of correlations between the predicted parameters.

3.3.2 Multi-task Decision-Focused Learning

In most DFL works, a single task is considered. For instance, in the shortest path benchmark considered by Elmachtoub and Grigas (2022), the grid structure and the start and end nodes are the same in all instances. However, one often has to deal with multiple tasks at once, in which it would be convenient to make decision-focused predictions, without having to train a separate model for each task. A first step in this direction was recently taken by Tang and Khalil (2023a). They propose a way of training a model in a decision-focused way with respect to multiple tasks at once by considering two kinds of architectures. The first is a regular multi-layer perceptron that outputs a single vector $\hat{\mathbf{c}}$ which is used in the different tasks. The different resulting task losses then get aggregated to inform the update to weights $\boldsymbol{\omega}$, i.e., the weights $\boldsymbol{\omega}$ are trained to produce a $\hat{\mathbf{c}}$ that generally works well for the different tasks considered. The second architecture is a multi-headed one, consisting of one or more shared first layers, followed by a dedicated head for every task. This produces a different vector $\hat{c}_i$ for every task. Their results show that they can train a model that can make effective decision-focused predictions for multiple tasks at once, and that this is particularly beneficial when not that many training data are available. However, a remaining limitation is that the model can still not be trained with the aim of *generalizing* to new tasks.

A related topic is studied by Dinh, Kotary, and Fioretto (2024a), which focuses on DFL in the context of fair multiobjective optimization. In the presence of multiple objective functions, many applications call for pareto-optimal solutions which are 'fair' with respect to the competing objective functions. In such cases, it is common to optimize the Ordered

Weighted Average (OWA) of those functions, which guarantees 'equitably efficient' solutions (Ogryczak & Śliwiński, 2003). Incorporating OWA optimization in DFL faces a unique challenge, since the OWA objective itself is nondifferentiable. Dinh et al. (2024a) mitigate by replacing the OWA function by its smooth Moreau envelope approximation. This yields a differentiable approximate OWA optimization, and enables joint DFL learning of each individual objective function's parameters to minimize regret of their fair OWA aggregation.

3.3.3 Predicting Parameters in the Constraints

The majority of the works in DFL aim to predict parameters in the objective function and assume that the feasible space is known. However, in many applications the unknown parameters occur in the constraints as well as in the objectives. When the parameters in the constraints are predicted and prescribed decisions are made using the predicted parameters, one major issue is that the prescribed decisions might turn out to be *infeasible* with respect to the true parameters. In this case, task loss to minimize the suboptimality of the prescribed decisions is not enough, but it should also penalize if the prescribed decisions become infeasible. Hence DFL methods for such problems entails a few additional considerations. The first consideration deals with quantifying the extent of infeasibility when the prescribed decisions become infeasible with respect to the true parameters. In this regard, Garcia, Street, Homem-de Mello, and Muñoz (2021) propose to add artificial slack variables with high penalty costs in the objective function to penalize infeasible decisions. Hu et al. (2023c) introduce the notion of *post-hoc* regret. In *post-hoc* regret a non-negative penalty is added to regret to account for correcting the infeasible solutions into feasible ones. This idea of such a corrective actions shares a fundamental resemblance to the concept of recourse actions in stochastic programming (Ruszczyński & Shapiro, 2003).

The next consideration is computing the gradients of this task loss with respect to the parameters in the constraints. Some of the techniques discussed in Section 3.1.1 can be utilized for this purpose. For example, the gradient can be obtained by solver unrolling. Tan, Delong, and Terekhov (2019) compute the gradient by unrolling a LP. As the parameters in the constraints are also present in the the KKT conditions (13), it is possible to compute the gradients for optimization problems, with differentiable constraint functions by differentiating the KKT conditions using the techniques discussed in Section 3.1.1.

Tan, Terekhov, and Delong (2020) provide an ERM formulation for predicting parameters in both the objective function and constraints of an LP. For parameters in the objective function, the ERM accounts for suboptimality of the decision and for parameters within the constraints, the ERM formulation considers the feasibility of optimal points observed in the training data. This ERM formulation takes the form of a non-linear optimization program and they propose to compute the derivative of the predicted parameters by considering its sequential quadratic programming (SQP) approximation. For packing and covering LPs, Hu et al. (2023c) compute the derivative of *post-hoc* regret with respect to the predicted parameters in the constraints by considering the Lagrangian relaxation of the LP when the corrective action is linear. Hu, Lee, and Lee (2023b) extend this approach by treating the correction action also as an LP.

The task of computing the gradients of the task loss with respect to the parameters in the constraints is particularly challenging for combinatorial optimization problems, which often

involve discrete feasible space. For combinatorial optimization problems, it might happen that no constraints are active at the optimal point. So, slight changes in the parameters in the constraints do not change the optimal solution, leading to the problem of zero gradients. Hence, coming up with meaningful gradients for back-propagation is more challenging for combinatorial optimization problems. If either the CO problem or the corrective action is an ILP, Hu et al. (2023b) consider the corresponding LP relaxation. Paulus, Rolínek, Musil, Amos, and Martius (2021) develop a differentiable optimization layer for ILPs that considers the downstream gradient of the solution as an input and returns the directions of updating the parameters in the backward pass. They update the parameters along the directions so that the Euclidean distance between the solution of the updated parameter and the updated solution with the downstream gradient is minimized. For ILPs, Nandwani, Ranjan, Mausam, and Singla (2022) view the task of constraint learning from the lens of learning hyperplanes, which is common in classification tasks. Such an approach requires negative samples. However, the negative samples in this setting must also include infeasible points, which is not the case for the solution-cache approach reviewed in Section 3.1.4. In the realm of gradient-free methods, Hu et al. (2023a) apply 'branch & learn' (Hu et al., 2022) to minimize *post-hoc* regret considering CO problems, solvable by recursion.

3.3.4 Model Robustness in Decision-Focused Learning

The issue of model robustness arises often in deep learning. As has been shown in many works, it is often possible for malicious actors to craft inputs to a neural network in such a way that the output is manipulated (evasion attacks) (Goodfellow, Shlens, & Szegedy, 2015), or to generate training data which cause adverse effects on the performance of the trained model (poisoning attacks). As a subset of ML, some adversarial settings also apply in DFL.

Evasion attacks, despite being the most commonly studied adversarial attacks, do not generalize straightforwardly to DFL since they inherently pertain to classification models with finite output spaces. On the other hand, it is shown by Kinsey, Tuck, Sinha, and Nguyen (2023) that effective poisoning attacks can be made against DFL models. The paper shows that while such attacks can be effective, they are computationally expensive due to the optimization which must be repeatedly evaluated to form the attacks. On the other hand, it is also demonstrated that poisoning attacks designed against two-stage models can be transferred to fully integrated DFL models.

Separately, Johnson-Yu, Wang, Finocchiaro, Taneja, and Tambe (2023) study robustness of decision-focused learning under label noise. The paper provides bounds on the degradation of regret when test-set labels are corrupted by noise relative to those of the training set. An adversarial training scheme is also proposed to mitigate this effect. The robust training problem is equivalent to finding the equilibrium solution to a Stackelberg game, in which a figurative adversary applies label noise that is optimized to raise regret, while the main player seeks model parameters that minimize regret.

3.3.5 Stochastic Optimization

Settings in decision-focused learning based on stochastic optimization models are studied by Donti, Kolter, and Amos (2017). In contrast to more typical settings, the downstream

decision model is considered to be a stochastic optimization problem. In this formulation, it is only possible to predict parameters of a random distribution that models the parameters of an optimization problem. For instance, the mean and variance of load demands in a power scheduling problem could be modeled as parameters of the optimization problem. Their work shows how such problems can be converted to DFL with deterministic decision models and solved using the techniques described in this survey article. To this end, it also introduces an effective technique for approximating the derivatives through arbitrary convex optimization problems, by forming and differentiating their quadratic programming approximations, as computed by sequential quadratic programming. A more elaborate relation to stochastic optimization is provided by Sadana et al. (2024) within the context of contextual stochastic optimization.

3.3.6 Active Learning Algorithm for DFL

Active learning concerns ML problems where labeled data are scarce or expensive to obtain. To address the challenge of limited training data, active learning algorithms choose the most informative instances for labeling (Settles, 2009). Liu, Grigas, Liu, and Shen (2023) study active learning in DFL paradigm. To identify the features for which knowing the cost parameter is most beneficial for training, they propose to use the notion of 'distance to degeneracy' (El Balghiti et al., 2019). Distance to degeneracy measures how far the predicted cost vector is from the set of cost vectors that have multiple optimal solutions. They argue that if distance to degeneracy is higher at a datapoint, there is more certainty regarding the solution (of the CO problem); hence they propose to acquire the label of a datapoint if its distance to degeneracy is lower than a threshold.

We end this section with a remark that DFL can be extended to address optimization problems beyond the 'classical' CO problem, as defined in (1). For instance, Wilder, Ewing, Dilkina, and Tambe (2019b) embed K-means clustering as a layer in a neural network by differentiating through the clustering layer, using DFL techniques. Wang, Shah, Chen, Perrault, Doshi-Velez, and Tambe (2021) employ a DFL approach to predict the parameters of Markov decision processes (MDPs).

4. Applications of Decision-Focused Learning

The *Predict-Then-Optimize* problem occurs in many real-world applications, as optimal decisions can be found by solving CO problems and due to the presence of uncertainty, some parameters of the CO problems must be estimated. Having seen the development of DFL for *Predict-Then-Optimize* problems in the preceding section, practical uses of DFL in various application domains will be presented below. As this survey paper reviews DFL techniques, which predict cost parameters in Section 3, the applications presented below are related specifically to the task of predicting only the cost parameters.

Computer vision. The DBB framework (Pogančić et al., 2020) (reviewed in Section 3.1.3) has been employed in many interesting computer vision applications. Rolinek, Musil, Paulus, Vlastelica, Michaelis, and Martius (2020a) use for differentiating rank-based metrics such as precision and recall, whereas Rolínek, Swoboda, Zietlow, Paulus, Musil, and Martius (2020b) and Kainmueller, Jug, Rother, and Myers (2014) use it for differentiat-

ing bipartite matching in deep graph and multi-graph matching problems respectively in the application of semantic keypoint matching of images. Abbas and Swoboda (2021) use DBB to differentiate through a multiway cut combinatorial optimization problem for graph partitioning allowing them to perform end-to-end panoptic segmentation of images.

Fair Learning to Rank. In learning to rank (LTR), a machine learning model must produce rankings of documents in response to users' search queries, in which those most relevant to a given query are placed in the highest ranking positions. In this setting, the relevance of documents to queries is often measured empirically by historical user click rates (Cao et al., 2007). In fair learning to rank (FLTR), this relevance-based matching must be performed subject to strict constraints on the relative exposure between predefined groups. Due to the difficulty of enforcing such constraints on the outputs of a machine learning model, many FLTR frameworks resort to a two-stage approach in which prediction of query-document relevance scores is learned by a typical LTR model without constraints on fairness of exposure. At test time, the predicted relevance scores inform the objective of a separate fair ranking optimization program (Singh & Joachims, 2018). Kotary et al. (2022) use DFL to unify the prediction of relevance scores with the subsequent optimization of fair rankings, in an end-to-end model trained by SPO. This DFL approach to FLTR requires significant computation to solve a linear program for fair rankings throughout training; it was subsequently proposed in Dinh, Kotary, and Fioretto (2024b) to employ a more efficient optimization of ordered weighted averages following Do and Usunier (2022), adapted to leverage the SPO training framework.

Route optimization. Ferber, Griffin, Dilkina, Keskin, and Gore (2023a) present an interesting application, where DFL is used to combat the challenge of wildlife trafficking. They consider the problem of predicting the flight trajectory of traffickers based on a given pair of source and destination airports. It is framed as a shortest path problem in a graph, where each node is an airport. In the prediction stage, the probability of using a directed edge (i, j) to leave the node i is predicted. In the optimization stage, the most likely path from the source to the destination is found by solving a shortest path problem where the negative log probabilities are used as edge weights. In this *Predict-Then-Optimize* formulation, the probabilities are predicted via DFL, using the DBB framework for gradient computation.

Solving a shortest path problem by considering the negative log probabilities as edge weights has also been explored by Mandi, Canoy, Bucarey, and Guns (2021). In their work, the objective is to prescribe most preferred routing in a capacitated vehicle routing problem (CVRP) (Toth & Vigo, 2015) for last-mile delivery applications. A high probability value for the edge (i, j) indicates that it is the preferred edge to leave the node i. However, they do not observe any advantage of the DFL paradigm over the PFL paradigm and attribute this to the lack of training data instances (fewer than 200 instances). DFL is used for last-mile delivery applications by Chu, Zhang, Bai, and Chen (2023) too. However, there the objective is to minimize total travel time. In the prediction stage, the travel times of all the edges are predicted and in the optimization stage, the CVRP is solved to minimize the total travel time. The underlying model is trained using the SPO framework to directly minimize the total travel time.

Maritime transportation. The inspection of ships by port state control has been framed as a *Predict-Then-Optimize* problem by Yang, Yan, and Wang (2022). Due to limited num-

ber of available personnel, the objective is to identify non-compliant ships that are more likely to be detained, and then select those ships for inspection. A ship can be found to be non-compliant by port state control in multiple categories. If a ship is found to be non-compliant for a category, a 'deficiency number' will be recorded for the ship in that category. In the prediction stage, a linear model is built to identify deficiency numbers of the ships in all the categories and in the optimization stage, a CO problem is solved to select ships maximizing the total 'deficiency number'. Given the scale of the CO problem at hand, training with standard SPO framework becomes prohibitive. Therefore, they employ pairwise-comparison based loss function, similar to Eq. (38) to implement DFL. Ship maintenance activities by ship owners have been framed as *Predict-Then-Optimize* problems by Tian, Yan, Liu, and Wang (2023). The ship owners have to schedule regular maintenance activities to remain compliant. However, as maintenance activities are expensive, the objective of identifying categories that may warrant immediate detentions by the port state control, has been considered. To do so, in the prediction stage, a random forest model is built to predict the deficiency number (likelihood of non-compliance) for each category. In the optimization stage, a CO problem is formulated considering maintenance cost and detention cost to determine whether maintenance activity should be scheduled for each category. The random forest models are trained to directly minimize regret using SPOTs (Elmachtoub et al., 2020).

Power and energy systems. Wahdany, Schmitt, and Cremer (2023) provide a use-case of DFL in renewable power system application. In their work, the prediction stage involves the task of generating wind power forecasts. As these forecasts are further used in power system energy scheduling, the predictive ML model is directly trained with the objective of minimizing power system operating costs using *cvxpylayers* (Agrawal et al., 2019a). Tschora, Guns, Pierre, Plantevit, and Robardet (2023) observe that electricity prices are set by regulators by optimizing social welfare while maintaining a constant energy balance. They formulate this optimization as a Mixed-Integer Quadratic Program (MIQP) problem and embed the CO problem into the training loop of an electricity price forecasting framework, which outperforms the conventional price forecasting approach. By analytically differentiating the CO problem, they derive a step function as the derivative, which is used for training the ML model in DFL paradigm. Sang, Xu, Long, Hu, and Sun (2022) consider another *Predict-Then-Optimize* problem in power system applications, where electricity prices are predicted in the prediction stage and the optimization stage deals with optimal energy storage system scheduling to maximize arbitrage benefits. Lower values of regret have been reported in their work, when the prices are predicted using the SPO framework.

Communication technology. DFL is applied to mobile wireless communication technology applications by Chai, Wong, Tong, Chen, and Zhang (2022). Fluid antenna systems (Wong, Tong, Zhang, & Zhongbin, 2020) are one of the recent developments in mobile wireless communication technology. However, its effectiveness depends on the position of the radiating element, known as the port. Chai et al. (2022) frame the port selection problem as a *Predict-Then-Optimize* problem, where in the prediction stage signal-to-noise ratio for each position of the port is predicted and then the optimal position of the port is decided in the optimization stage. They use LSTM as the predictive model and report the SPO framework is very effective for such port selection applications.

Problem	Constraint Functions	Decision Variables	CO SOlver	Predictive Model
Shortest path problem on a 5×5 grid	Linear	Continuous	*OR-Tools*	Linear
Portfolio optimization	Quadratic	Continuous	*Gurobi*	Linear
Warcraft shortest path	Linear	Continuous	Customized python implementation of Dijkstra	CNN
Energy-cost aware scheduling	Linear	Discrete	*Gurobi*	Linear
Knapsack problem	Linear	Discrete	*OR-Tools*	Linear
Diverse bipartite matching	Linear	Discrete	*OR-Tools*	Multi-layer NN
Subset selections	Linear	Continuous	*Gurobi*	Linear

Table 2: Brief overview of the test problems considered for experimental evaluation. The **objective functions are linear** for all the optimization problem.

Solving non-linear combinatorial optimization problems. Ferber et al. (2023b) study the problem of learning a linear surrogate optimizer to solve non-linear optimization problems. The objective is to learn a surrogate linear optimizer whose optimal solution is the same as the solution to the non-linear optimization problem. Learning the parameters of the surrogate linear optimizer entails backpropagating through the optimizer, which is implemented using *cvxpylayers* (Agrawal et al., 2019a).

Further, interested readers are referred to the artcle by Mišić and Perakis (2020) for more applications of *Predict-Then-Optimize* problems in various areas within OR.

5. Experimental Evaluation on Benchmark Problemsets

DFL recently has received increasing attention. The methodologies discussed in Section 3 have been tested so far on several different datasets. Because a common benchmark for the field has not yet been set up, comparisons among techniques are sometimes inconsistent. In this section, an effort is made to propose several benchmark test problems for evaluating DFL techniques. Then some of the techniques explained in Section 3 are compared on these test problems.

5.1 Problem Descriptions

All the test problems, which are selected for benchmarking, have been previously used in the DFL literature and their datasets are publicly available. All these problems are *Predict-Then-Optimize* problems, meaning they encompass the two stages—prediction and optimization. Moreover, all the optimization problems have linear objective functions. Table 2 provides an overview of the experimental setups associated with each test problem, including the specification of the CO problem and the type of predictive model. Next, these test problems are described in detail.

5.1.1 Shortest Path Problem on a 5×5 grid

This experiment is adopted from the work of Elmachtoub and Grigas (2022). It is a shortest path problem on a 5×5 grid, with the objective of going from the southwest corner of the grid to the northeast corner where the edges can go either north or east. This grid consists of 25 nodes and 40 edges.

Formulation of the optimization problem. The shortest path problem on a graph with a set V of vertices and a set E of edges can be formulated as an LP problem in the following form:

$$\min_{\mathbf{x}} \ \mathbf{c}^\top \mathbf{x} \tag{42a}$$

$$\mathtt{s.t.}\ A\mathbf{x} = \mathbf{b} \tag{42b}$$

$$\mathbf{x} \geq \mathbf{0} \tag{42c}$$

Where $A \in \mathbb{R}^{|V|\times|E|}$ is the incidence matrix of the graph. The decision variable $\mathbf{x} \in \mathbb{R}^{|E|}$ is a binary vector whose entries are 1 only if the corresponding edge is selected for traversal. $\mathbf{b} \in \mathbb{R}^{|V|}$ is the vector whose entry corresponding to the source and sink nodes are 1 and -1 respectively; all other entries are 0. The constraint (42b) must be satisfied to ensure the path will go from the source to the sink node. The objective is to minimize the cost of the path with respect to the (predicted) cost vector $\mathbf{c} \in \mathbb{R}^{|E|}$.

Synthetic data generation process. In this problem, the prediction task is to predict the cost vector $\mathbf{c}$ from the feature vector $\mathbf{z}$. The feature and cost vectors are generated according to the data generation process defined by Elmachtoub and Grigas (2022). For the sake of completeness, the data generation process is described below.[1] Each problem instance has cost vector of dimension $|E| = 40$ and feature vector of dimension $p = 5$. The training data consists of $\{(\mathbf{z_i}, \mathbf{c_i})\}_{i=1}^{N}$, which are generated synthetically. The feature vectors are sampled from a multivariate Gaussian distribution with zero mean and unit variance, i.e., $\mathbf{z_i} \sim \mathbf{N}(0, I_p)$ To generate the cost vector, first a matrix $B \in \mathbb{R}^{|E|\times p}$ is generated, which represents the true underlying model. The cost vectors are then generated according to the following formula:

$$c_{ij} = \left[\left(\frac{1}{\sqrt{p}}(B\mathbf{z_i}) + 3\right)^{\mathrm{Deg}} + 1\right]\xi_i^j \tag{43}$$

where c_{ij} is the j^{th} component of cost vector $\mathbf{c_i}$. The *Deg* parameter specifies the extent of model misspecification, because a linear model is used as a predictive model in the experiment. The higher the value of *Deg*, the more the true relation between the features and objective parameters deviates from a linear one and the larger the prediction errors will be. Finally, ξ_i^j is a multiplicative noise term sampled randomly from the uniform distribution $[1-\vartheta, 1+\vartheta]$. The experimental evaluation involves five values of the parameter *Deg*, which are $1, 2, 4, 6$ and 8, and the noise-halfwidth parameter ϑ being 0.5. Furthermore, for each setting, a different training set of of size 1000 is used. In each case, the final performance of the model is evaluated on a test set of size $10,000$.

Predictive model. In each setting, the underlying predictive model is a one-layer feed-forward neural network without any hidden layer, i.e., a linear model. The input to the model is a p dimensional vector, and output is a $|E|$ dimensional vector. Note that a multi-layer neural network model can be used to improve the accuracy of the predictive model. The intuition behind using a simple predictive model is to test the efficacy of the DFL techniques when the predictions are not 100% accurate. The DFL techniques are trained to

1. The generator in `https://github.com/paulgrigas/SmartPredictThenOptimize` is used to generate the dataset.

minimize regret, and the prediction-focused model is trained by minimizing the MSE loss between the true and predicted cost vectors.

5.1.2 Portfolio Optimization Problem

A classic problem that combines prediction and optimization is the Markowitz portfolio optimization problem, in which asset prices are predicted by a model based on empirical data, and then subsequently, a risk-constrained optimization problem is solved for a portfolio which maximizes expected return. This experiment is also adopted from the work of Elmachtoub and Grigas (2022).

Formulation of the optimization problem. In portfolio optimization problem, the objective is to choose a portfolio of assets having highest return subject to a constraint on the total risk of the portfolio. The problem is formulated in the following form:

$$\max_{\mathbf{x}} \ \mathbf{c}^\top \mathbf{x} \tag{44a}$$

$$\texttt{s.t.} \quad \mathbf{x}^\top \Sigma \mathbf{x} \leq \gamma \tag{44b}$$

$$\mathbf{1}^\top \mathbf{x} \leq 1 \tag{44c}$$

$$\mathbf{x} \geq 0 \tag{44d}$$

where $\mathbf{1}$ is the vector of all-ones of same dimension as $\mathbf{x}$, $\mathbf{c}$ is the vector of asset prices, and Σ is a predetermined matrix of covariances between asset returns. The objective (44a) is to maximize the portfolio's total value. Eq. (44b) is a risk constraint, which bounds the overall variance of the portfolio, and (44c), (44d) model $\mathbf{x}$ as a vector of proportional allocations among assets.

Synthetic data generation process. Synthetic input-target pairs $(\mathbf{z}, \mathbf{c})$ are randomly generated, according to a random function with a specified degree of nonlinearity $\text{Deg} \in \mathbb{N}$. The procedure for generating the random data as follows:

Given a number of assets d and input features of size p, input samples $\mathbf{x_i} \in \mathbb{R}^p$ are sampled element wise from i.i.d. standard normal distributions $\mathbf{N}(0, 1)$. A random matrix $B \in \mathbb{R}^{d \times p}$, whose elements $B_{ij} \in \{0, 1\}$ are drawn from i.i.d. Bernoulli distributions which take the value 1 with probability 0.5, is created. For a chosen noise magnitude ϑ, $L \in \mathbb{R}^{n \times 4}$ whose entries are drawn uniformly over $[-0.0025\vartheta, 0.0025\vartheta]$ is generated. Asset returns are calculated first in terms of their conditional mean $\bar{c}_{ij}$ as

$$\bar{c}_{ij} \coloneqq (\frac{0.05}{\sqrt{p}}(B\mathbf{z_i})_j + (0.1)^{\frac{1}{\text{Deg}}})^{\text{Deg}} \tag{45}$$

Then the observed return vectors $\mathbf{c_i}$ are defined as $c_{ij} \coloneqq \bar{r}_i + Lf + 0.01\vartheta\xi$, where $f \sim \mathbf{N}(0, I_4)$ and noise $\xi \sim \mathbf{N}(0, I_d)$ This causes the c_{ij} to obey the covariance matrix $\Sigma \coloneqq LL^\top + (0.01\zeta)^2\mathbf{I}$, which is also used to form the constraint (44b), along with a bound on risk, defined as $\gamma \coloneqq 2.25\, e^\top \Sigma e$ where e is the equal-allocation solution (a constant vector). Four values of the parameter *Deg*—$1, 4, 8, 16$ have been used in the experimal evaluation. The value of noise magnitude parameter ϑ is set to 1. It is assumed that the covariance matrix of the asset returns does not depend on the features. The values of Σ and γ are constant, and randomly generated for each setting.

Predictive model. Like the previous experiment, the underlying predictive model is a linear model, whose input is a feature vector $\mathbf{z} \in \mathbb{R}^p$ and output is the return vector $\mathbf{c} \in \mathbb{R}^d$.

5.1.3 Warcraft Shortest Path Problem

This experiment was adopted from the work of Pogančić et al. (2020). Each instance in this problem is an image of a terrain map using the Warcraft II tileset (Guyomarch, 2017). Each image represents a grid of dimension $d \times d$. Each of the d^2 pixels has a fixed underlying cost, which is unknown and to be predicted. The objective is to identify the minimum cost path from the top-left pixel to the bottom-right pixel. From one pixel, one can go in eight neighboring pixels—up, down, front, back, as well as four diagonal ones. Hence, it is a shortest path problem on a graph with d^2 vertices and $\mathcal{O}(d^2)$ edges.

Formulation of the optimization problem. Note that this is a node-weighted shortest path problem, where each node (pixel) in the grid is assigned a cost value; whereas in the previous shortest path problem, each edge is assigned a cost value. However, this problem can be easily reduced to the more familiar edge weighted shortest path problem by 'node splitting'. 'Node splitting' splits each node into two separate nodes—entry and exit nodes and adds an edge, that has a weight equal to the node weight, from the entry node to the exit node. For each original edge, an edge, with null weight, from the exit node of the source node to the entry node of the sink node, is constructed.

Predictive model. The prediction task is to predict the cost associated with each pixel. The actual cost ranges from 0.8 to 9.2 and is dependent on visible characteristics of the pixel. For instance, cost changes depending on whether the pixel represents a water-body, land or wood. To predicts the cost of each node (pixel), a convolutional neural network (CNN) is employed. The CNN takes the $d \times d$ image as an input and outputs costs of the d^2 pixels. The ResNet18 (He, Zhang, Ren, & Sun, 2016) architecture is slightly modified to form the ML model. The first five layers of ResNet18 are followed by a max-pooling operation to predict the underlying cost of each pixel. Furthermore, a Relu activation function (Agarap, 2019) is used to ensure the predicted cost remains positive, thereby avoiding the existence of negative cycles in the shortest path edge weights.

5.1.4 Energy-Cost Aware Scheduling Problem

This experiment setup was adopted from the work of Mandi et al. (2020). This is a resource-constrained day-ahead job scheduling problem (Simonis, O'Sullivan, Mehta, Hurley, & Cauwer, 1999) with the objective of minimizing energy cost. Tasks must be assigned to a given number of machines, where each task has a duration, an earliest start, a latest end, a resource requirement and a power usage. Each machine has a resource capacity constraint. Also, tasks cannot be interrupted once started, nor migrated to another machine and must be completed before midnight. The scheduling is done one day in advance. So, the prediction task is to predict the energy prices of the next day.

Formulation of the optimization problem. The scheduling problem is formulated as an ILP. Let J be the set of tasks to be scheduled on a set of machines I while maintaining resource requirement of W resources. The tasks must be scheduled over T number of time slots. Each task j is specified by its duration ζ_j, earliest start time $\zeta_j^{(1)}$, latest end time

$\zeta_j^{(2)}$, power usage ϕ_j. Let ρ_{jw} be the resource usage of task j for resource w and q_{iw} is the capacity of machine i for resource w. Let x_{jit} be a binary variable that takes the value 1 only if task j starts at time t on machine i. The objective of minimizing energy cost while satisfying the required constraints can be expressed by the following ILP:

$$\min_{x_{jit}} \sum_{j\in J}\sum_{i\in I}\sum_{t\in T} x_{jit}\Big(\sum_{t\le t\prime<t+\zeta_j}\phi_j c_{t\prime}\Big) \tag{46a}$$

$$\texttt{s.t.} \quad \sum_{i\in I}\sum_{t\in T} x_{jit} = 1 \ , \forall_{j\in J} \tag{46b}$$

$$x_{jit} = 0 \quad \forall_{j\in J}\forall_{i\in I}\forall_{t<\zeta_j^{(1)}} \tag{46c}$$

$$x_{jit} = 0 \quad \forall_{j\in J}\forall_{i\in I}\forall_{t+\zeta_j>\zeta_j^{(2)}} \tag{46d}$$

$$\sum_{j\in J}\sum_{t-\zeta_j<t'\le t} x_{jit\prime}\rho_{jw} \le q_{iw}, \forall_{i\in I}\forall_{w\in W}\forall_{t\in T} \tag{46e}$$

$$x_{jit} \in \{0,1\}\forall_{j\in J}\forall_{i\in I}\forall_{t\in T} \tag{46f}$$

The (46b) constraint ensures each task is scheduled once and only once. The constraints in (46c) and (46d) ensure that the task scheduling abides by earliest start time and latest end time constraints. (46e) imposes the constraints of resource requirement.

Data description. The prediction task is to predict the energy prices one day in advance. The energy price dataset comes from the Irish Single Electricity Market Operator (SEMO) (Ifrim, O'Sullivan, & Simonis, 2012). This dataset consists of historical energy price data at 30-minute intervals starting from midnight on the 1st of November, 2011 until the 31st of December, 2013. In this setup, each day forms an optimization instance, which comprises of 48 time slots, corresponding to 48 half-hour slots. Each half-hour instance of the data has calendar attributes, day-ahead estimates of weather characteristics, SEMO day-ahead forecasted energy-load, wind-energy production and prices, actual wind-speed, temperature and CO_2 intensity, which are used as features. So, the dimension of feature vector is 8. Note that, in this dataset, each c_t in the cost vector is associated with an eight dimensional feature vector, i.e., $\mathbf{c} \in \mathbb{R}^{48}$ and $\mathbf{z} \in \mathbb{R}^{48\times 8}$.

Predictive model. As energy prices of each half-hour slot is associated with 8 features, the input to the predictive model is a feature vector of dimension 8 and output is a scalar. In this case also, the predictive model is a linear model, i.e., a feed forward neural network without any hidden layer.

5.1.5 Knapsack Problem

This problem setup was also adopted from the work of Mandi et al. (2020). The objective of the knapsack problem is to choose a maximal value subset from a given set of items, subject to a capacity constraint. In this case, the weights of all items and the knapsack capacity are known. What is unknown are the values of each item. Hence, the prediction task is to predict the value of each item.

Formulation of the optimization problem. The formulation of the knapsack optimization problem with unit weights has already been provided in Eq. (4). However, in

general the weights of all items are not equal. So, a general knapsack optimization problem can be formulated as follows:

$$\max_{\mathbf{x}} \ \mathbf{c}^\top \mathbf{x} \tag{47a}$$

$$\texttt{s.t.} \ \mathbf{w}^\top \mathbf{x} \leq \text{Capacity} \tag{47b}$$

$$\mathbf{x} \in \{0, 1\} \tag{47c}$$

where $\mathbf{w}$, $\mathbf{c}$ are the vector of weights and values respectively.

Data description. For this problem again, the dataset is adapted from the Irish Single Electricity Market Operator (SEMO) (Ifrim et al., 2012). In this setup, each day forms an optimization instance and each half-hour corresponds to a knapsack item. So the cost vector $\mathbf{c}$ and the weights $\mathbf{w}$ are of length 48, corresponding to 48 half-hours. The motivation can be framed as identifying half-hour slots that yield maximum revenue while adhering to the constraint of booking each slot. Similar to the energy scheduling problem, each item of the cost vector is associated with a feature vector of dimension 8. The weight vector is fixed. The weights are generated synthetically as done by Mandi et al. (2020). The data generation is as follows. First a weight w_i is assigned to each of the 48 half-hour slots, by sampling from the set $\{3, 5, 7\}$. In order to introduce correlation between the item weights and the item values, the energy price vector is multiplied with the weight vector and then a randomness is incorporated by adding Gaussian noise $\xi \sim \mathbf{N}(0, 25)$, which produces the final item values c_i. The motivation behind introducing correlation between the item weights and the item values stems from the fact that solving a knapsack problem with correlated item weights and values is considered to be hard to solve (Pisinger, 2005). The sum of the weights of each instance is 240. 60, 120, and 180 are the three values of capacity with which the experiments are performed.

Predictive model. The predictive model is same as the previous problem, i.e., a feed forward neural network without any hidden layer.

5.1.6 Diverse Bipartite Matching Problem

This experimental setup is adopted from Ferber et al. (2020). In this problem, two disjoint sets of nodes are provided and the objective is to match between the nodes of the two sets. The graph topologies are taken from the CORA citation network (Sen, Namata, Bilgic, Getoor, Galligher, & Eliassi-Rad, 2008), where a node represent an published article and an edge represent a citation. So the goal of the matching problem is to identify the citations between the two sets of articles. Furthermore, the matching must obey diversity constraints, as described later.

Optimization problem formulation. Let S_1 and S_2 denote the two sets. The matching must satisfy the following diversity constraints: a minimum $\rho_1\%$ and $\rho_2\%$ of the suggested pairings should belong to same and distinct fields of study respectively. In this matching problem, each edge does not have an associated cost in the true sense. The DFL approaches consider the likelihood of the existence of each edge as the edge weights and then determine which edges should be present while ensuring all the constraints are satisfied. Let c_{ij} be the likelihood of an edge existing between article i and j, $\forall i \in S_1, j \in S_2$. With this likelihood

value, the matching can be performed by solving the following ILP, which ensures the diversity constraints:

$$\max_{\mathbf{x}} \sum_{i,j} c_{ij} x_{ij} \tag{48a}$$

$$\texttt{s.t.} \quad \sum_{j} x_{ij} \leq 1 \qquad \forall i \in S_1 \tag{48b}$$

$$\sum_{i} x_{ij} \leq 1 \qquad \forall j \in S_2 \tag{48c}$$

$$\sum_{i,j} \phi_{i,j} x_{ij} \geq \rho_1 \sum_{i,j} x_{ij} \tag{48d}$$

$$\sum_{i,j} (1 - \phi_{ij}) x_{ij} \geq \rho_2 \sum_{i,j} x_{ij} \tag{48e}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in S_1, j \in S_2 \tag{48f}$$

where ϕ_{ij} is an indicator, which takes the value 1 only if article i and j are of same field, and 0 if they belong to two different fields.

Data description. The network is divided into 27 disjoint topologies, each of which forms a distinct optimization problem instance containing 100 nodes. In each instance, the 100 nodes are split into two sets of 50 nodes S_1 and S_2; so each instance forms a bipartite matching problem between two sets of cardinality 50. Each publication (node) has 1433 bag-of-words features. The feature of an edge is formed by concatenating features of the two corresponding nodes. The prediction task is to estimate c_{ij} values. In this problem, each individual c_{ij} is associated with a feature vector of length 2866.

Predictive model. Although the DFL paradigm considers the likelihood of the existence of each edge as the edge weights, in the prediction-focused approach the model is trained by directly predicting the presence or absence of each edge. The predictive model is a neural network model. The input to the neural network is a 2866 dimensional vector and final output is a scalar between 0 and 1. The neural network has one hidden layer and uses a sigmoid activation function on the output.

5.1.7 Subset Selections

This experiment is a structured prediction task, in which the object is to learn a mapping from feature vectors to binary vectors which represent subset selections. Unlike the other experiments above, the ground-truth data take the form of optimal solutions to an optimization problem, rather than its corresponding problem parameters. Thus, the regret loss is not suitable for training a prediction model. Instead, a task loss based on the error of the predicted solutions with respect to ground-truth solutions is used in this experiment.

Optimization problem formulation. For any $\mathbf{c} \in \mathbb{R}^n$, the objective of the optimization problem is to output a binary vector in $\mathbb{R}^n$, where the non-zero values correspond to the

Hyperparameter	Techniques Utilizing the Hyperparameter	Range
learning rate	All	$\{5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 0.01, 0.05, 0.1, 0.5, 1.0\}$
δ	I-MLE, DBB	$\{0.1, 1, 10, 100\}$
ϵ	I-MLE, FY	$\{0.05, 0.1, 0.5, 1, 2, 5\}$
κ	I-MLE	$\{5, 10, 50\}$
τ	Listwise	$\{0.05, 0.1, 0.5, 1, 2, 5\}$
Θ	Pairwise	$\{0.01, 0.05, 0.1, 1., 10., 50.\}$
μ	QPTL, HSD	$\{0.01, 0.1, 1., 10.,\}$

Table 3: The range of hyperparameters for hyperparameter tuning by grid search.

top-k values of $\mathbf{c}$. This can be formulated as an LP problem in the following form:

$$\underset{\mathbf{x}}{\text{argmax}} \ \mathbf{c}^\top \mathbf{x} \tag{49a}$$

$$\texttt{s.t.} \ \ 1^\top \mathbf{x} = k \tag{49b}$$

$$0 \leq \mathbf{x} \leq 1 \tag{49c}$$

As a totally unimodular linear program with integral parameters, this problem has (binary) integer optimal solutions. This mapping is known for its ability to represent subset selections in structured prediction, and is useful for multilabel classification (Amos et al., 2019) .

Data Description. Let $\mathbf{U}(0, 1)$ be a uniform random distribution; then a collection of feature vectors $\mathbf{z}$ are generated by $\mathbf{z} \sim \mathbf{U}(0, 1)^n$. For each $\mathbf{z}$, its corresponding target data is a binary vector containing unit values corresponding to the top-k values of $\mathbf{z}$, and zero values elsewhere. Three datasets are generated, each of 1000 training samples, in which the selection problem takes size $n = 25$, $n = 50$, and $n = 100$ respectively. The subset size k is chosen to be one fifth of n, in each case.

Predictive model. Like the previous problem, the predictive model here also is a linear model, i.e., a feed forward neural network without any hidden layer. In this problem, the task loss to train the predicative model is the negated inner product between true selection $\mathbf{x}$ and prescribed selection $\hat{\mathbf{x}}$, i.e., $\mathcal{L}(\hat{\mathbf{x}}, \mathbf{x}) = \hat{\mathbf{x}}^\top \mathbf{x}$, which is minimized when $\hat{\mathbf{x}} = \mathbf{x}$. Since the model is not regret-based, and does not assume access to the ground-truth parameters $\mathbf{c}$, techniques which rely on such assumptions are not tested on this problem.

5.2 Experimental Results and Analysis

In this subsection, results of comparative evaluations of some of the techniques introduced in Section 3 on the datasets mentioned in Section 5 are presented. The following techniques are considered for evaluations: 1. Prediction-focused (PF) approach, 2. Smart "Predict, Then Optimize" (32) [SPO], 3. Differentiation of blackbox combinatorial solvers (24) [DBB], 4. Implicit maximum likelihood estimation (30) [I-MLE], 5. Fenchel-Young loss (29) [FY], 6. Differentiation of homogeneous self-dual embedding (20) [HSD], 7. Quadratic programming task loss (19) [QPTL], 8. Listwise LTR loss (40) [Listwise], 9. Pairwise LTR loss (38) [Pairwise], 10. Pairwise difference LTR loss (39) [Pairwise(diff)], 11. Maximum a posteriori

contrastive loss (37) [MAP]. The reason behind including prediction-focused approach is that it is considered as a benchmark. Note that among these techniques, Listwise, Pairwise, Pairwise(diff), and MAP make use of a solution cache. The solution cache is implemented using the procedure proposed by Mulamba et al. (2021). In this approach, the solution cache is initialized by caching all the solutions in the training data and the cache is later expanded by employing a p_{solve} parameter value greater than zero. As in (Mulamba et al., 2021; Mandi et al., 2022) it is reported that $p_{solve} = 5\%$ is adequate for most applications, the value of p_{solve} is set to 5%. Next, the procedure systematically followed for the empirical evaluations is explained.

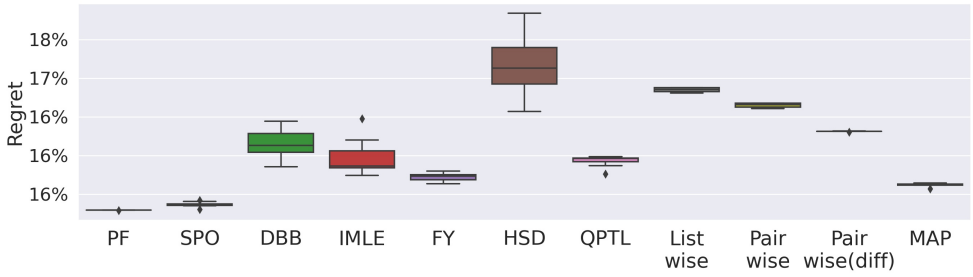
Experimental setup and procedures. The performance of a technique is sensitive to the choice of the technique specific hyperparameters as well as some other fundamental hyperparameters, common in any neural network training such as learning rate. These are called hyperparameters because they cannot be estimated by training the model, rather they must be selected before training begins. Tuning hyperparameters is the process of identifying the set of hyperparameter values that are expected to produce the best model outcome. In the experimental evaluations, hyperparameter tuning is performed via grid search. In the grid search, each of the hyperparameters is tried for a set of values. The set of values to be tested for each hyperparameter is predetermined. Grid search suffers from the curse of dimensionality in the hyperparameter space, as the number of combinations grows exponentially with the number of hyperparameters. However, it is possible to train the different models for different combinations of hyperparameters in parallel as the combinations are independent.

The hyperparameter of each model for each experiment is selected based on performance on the validation dataset. For each hyperparameter a range of values as defined in Table 3 is considered. The hyperparameter combination which produces the lowest average regret on the validation dataset is considered to be the 'optimal' one. For both validation and testing, 10 trials are run where in every trial the network weights are initialized with a different seed. To be specific, values of seed from 0 to 9 have been considered. Each model for each setup is trained using *pytorch* (Paszke et al., 2019) and *pytorch-lightning* (Falcon et al., 2019) with the *adam optimizer* (Kingma & Ba, 2015) and *'reduceLROnPlateau'*(PyTorch, 2017) learning rate scheduler. As mentioned before, the learning rate of *adam optimizer* is treated as a hyperparameter. For QPTL, the QP problems are solved using *cvxpylayers* (Agrawal et al., 2019a). For other techniques, which treat the CO solver as a blackbox solver, *gurobi* (Gurobi Optimization, 2021) or *ortools* (Perron & Furnon, 2020) is used as the solver.

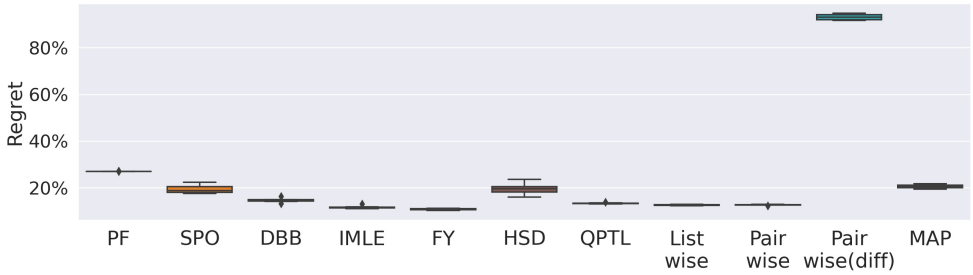
Evaluation metric. After selecting the 'optimal' hyperparameter combination for each test problem, **10** trials of all the techniques with the 'optimal' hyperparameter combination are run on test dataset. Unless otherwise mentioned the comparative evaluation is made based on relative regret on the test dataset. The **relative regret** is defined as follows:

$$\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{\mathbf{c_i}^\top (\mathbf{x}^\star(\hat{\mathbf{c}}_\mathbf{i}) - \mathbf{x}^\star(\mathbf{c_i}))}{\mathbf{c_i}^\top \mathbf{x}^\star(\mathbf{c_i})}. \tag{50}$$

In practice, $\mathbf{c}$ (or $\hat{\mathbf{c}}$) can have non-unique optimal solutions. However, note that if all the entries in $\mathbf{c}$ are continuous, it is very unlikely that $\mathbf{c}$ will have non-unique solutions. For



(a) Deg = 1



(b) Deg = 8

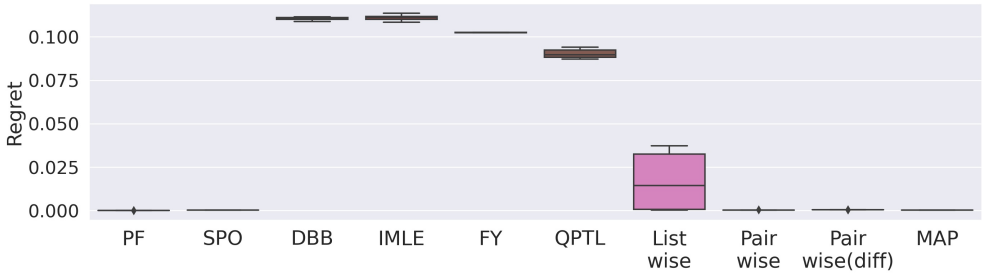
Figure 5: Comparative evaluations on the synthetic shortest path problem with noise-halfwidth parameter $\vartheta = 0.5$. The boxplots show the distributions of relative regrets.

instance, in the case of an LP, the only circumstance in which the LP can have multiple solutions is when $\mathbf{c}$ is parallel to one of the faces of the LP polyhedron. Nevertheless, if the cost vector is predicted by an ML model, a pathological case might occur, especially at the beginning of model training, when all the cost parameters are zero. This results in all feasible solutions being optimal with zero cost. However, to avoid this complexity in the experiments, it is assumed that the solution $\mathbf{x}^\star(\hat{\mathbf{c}})$ is obtained by calling a CO solver as an oracle and that if there exist non-unique solutions, the oracle returns a single optimal solution by breaking ties in a pre-specified manner. This is true if a commercial solver such as Gurobi is used to solve the CO problem.

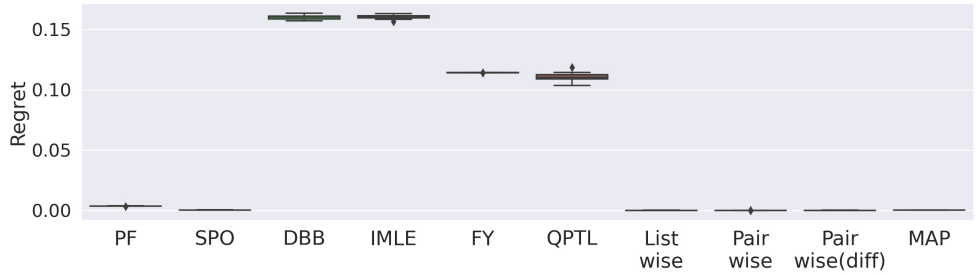
5.2.1 Comparative Evaluations

Next, the performances of the 11 DFL techniques across the 7 problems are presented for comparative evaluation.

Shortest path problem on a 5×5 grid. The comparative evaluation for the synthetic shortest path problem is shown in Figure 5 with the aid of box plots. To conserve space, boxplots for two values of Deg are shown in Figure 5. The boxplots for all the five degrees are shown in Figure A1 in the Appendix. In Figure 5, the value of ϑ, the noise-halfwidth parameter is 0.5 for all the experiments and the training set for each Deg contains 1000 instances. The predictive model is a simple linear model implemented as a neural network model with no hidden layers.



(a) Deg = 1



(b) Deg = 16

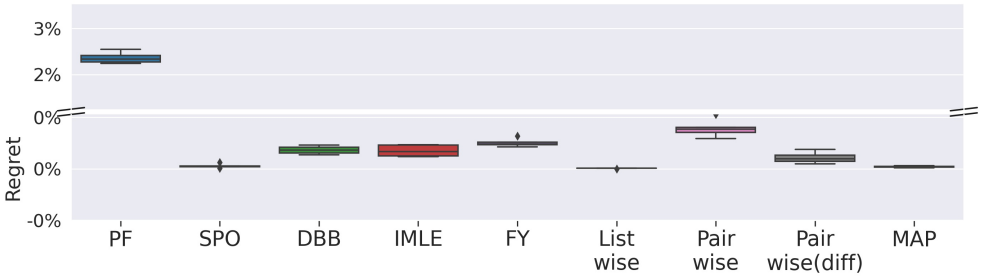
Figure 6: Comparative evaluations on the synthetic portfolio optimization problem with noise magnitude $\vartheta = 1$. The boxplots show the distributions of **absolute** regrets.

For Deg 1, the linear predictive model perfectly captures the data generation process. Consequently, the PF approach is very accurate and it results in the lowest regret. SPO has slightly higher regret than the PF approach. All the other DFL techniques have considerably higher regrets compared to PF. In contrast, the relative regret worsens for the PF approach, as the value of Deg parameter is increased. For Deg 8, PF is not the best as the predictive model is misspecified. In this case, FY has the lowest regret; while DFL techniques, other than Pairwise(diff) results in lower regret than the PF approach. For Deg 6, the best one FY, although its test regret is not very different from SPO, I-MLE and QPTL. For Deg 4, I-MLE has the lowest regret, closely followed by FY and SPO. For Deg 2, both PF and SPO have lowest regret.

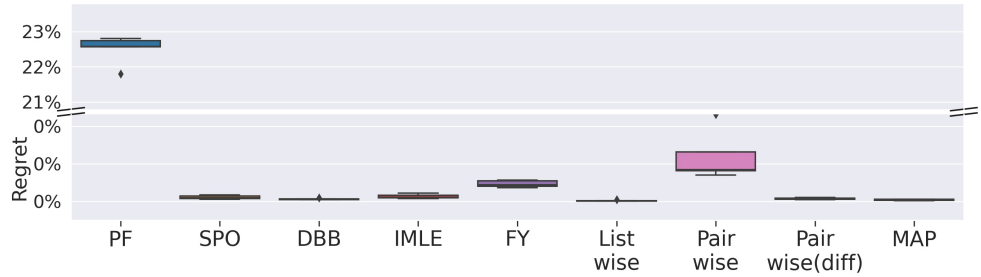
Among the DFL techniques, HSD exhibits higher regret and higher variances than the other DFL approaches. It performs better than the PF approach only for Deg 6 and 8.

Portfolio optimization problem. Note that this is an optimization problem with continuous decision variables having **quadratic constraints and a linear objective function.** Hence, the HSD approach is not applicable for this problem, as it cannot handle non-linear constraints. However, as *cvxpylayers* is used to implement QPTL, it can be applied in this problem. The boxplots of test regrets for noise magnitude parameter ϑ being 1 are shown in Figure 6.

In this problem, in some problem instances, all the return values are negative, which makes a portfolio with zero return to be the optimal portfolio. In such cases, relative regret turns infinite as the denominator is zero in Eq. (50). Hence, for this problem set, the **absolute regret** instead of relative regret is reported in Figure 6. The boxplots for Deg



(a) Image Size 12×12



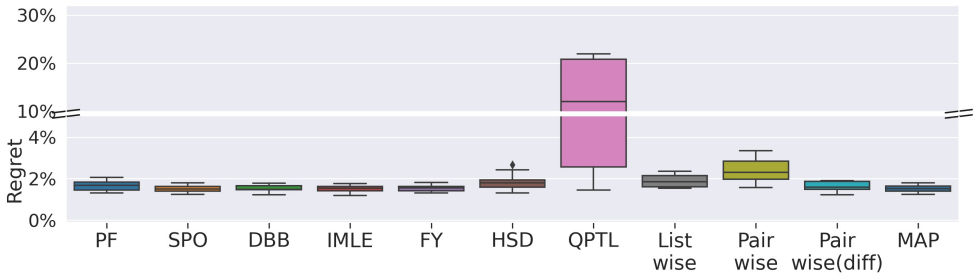
(b) Image Size 30×30

Figure 7: Comparative evaluations on the Warcraft shortest path problem instances. The boxplots show the distributions of relative regrets.

values of 1 and 16 are shown in The boxplots for all the four degrees are shown in Figure A2 in the Appendix.

Apparently, the PF approach performs very well in this problem; but SPO manages to outperform PF slightly in all cases except for Deg 1. It is evident in Figure 6 that DBB, I-MLE, FY and QPTL perform miserably as they generate regret even higher than the PF approach. All these techniques were proposed considering problems with linear constraints. Hence concerns arise that these techniques might not be suitable in the presence of quadratic constraints. On the other hand, LTR losses—Pairwise and Pairwise(diff) and contrastive loss function, MAP, perform even better than SPO for Deg 16. The Listwise LTR loss exhibits high variance for Deg 1 and for Deg values of 4 and 8. For Deg 16, it generates average test regret lower than SPO. In general, Figure 6 reveals DBB, I-MLE, FY and QPTL perform poorly in this problem, whereas, SPO, MAP Pairwise and Pairwise(diff) seem to be the best performing DFL techniques for this problem.

Warcraft shortest path problem. Recall that this a shortest path problem in an image with dimension $d \times d$. The optimization problem can be efficiently solved using Dijkstra's algorithm (Dijkstra, 1959), as underlying costs of all the pixel values are non-negative. Hence the shortest path problem is solved using Dijkstra's algorithm for the techniques which view the CO solver as a blackbox oracle. However, HSD and QPTL require the problem to be formulated as an LP and require a primal-dual solver. Note that in this experiment, the predictive ML model is a CNN, which predicts the cost of each pixel. In this case, training the ML model is challenging due to the large number of parameters. Hence, combining this ML model with computation-intensive modules such as a primal-



(a) Instance 1

Figure 8: Comparative evaluations on the energy-cost aware scheduling problem instances. This boxplot shows the distributions of relative regrets.

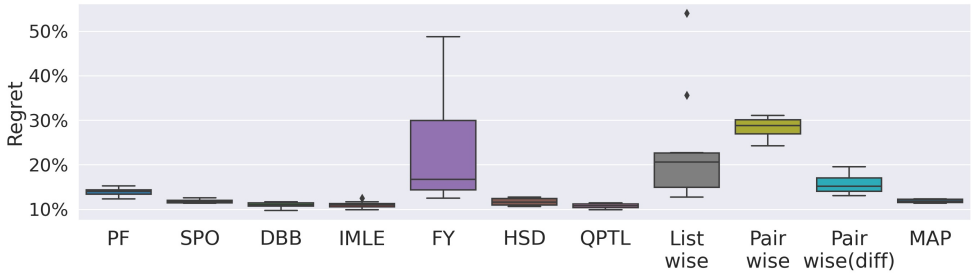
dual solver poses significant challenges. We could not run the experiments with HSD and QPTL because of this computational burden.

The dataset contains four values of d: $12, 18, 24, 30$. Clearly, as the value of d increases, the number of parameters of the CO problem increases. The boxplots of comparative evaluations are summarized in Figure 7. The boxplots of the other two values of d can be found in Figure A3 in the Appendix. First, note that the PF approach, which is trained by minimizing MSE loss between the predicted cost and the true cost, performs significantly worse than the DFL techniques. In fact, the performance of the PF approach deteriorates as the image size increases. As the size of the image increases, the same level of prediction error induces greater inaccuracies in the solution. This is because an increase in the area of the image involves dealing with a greater number of decision variables in the CO problem. When the level of prediction error remains constant, the probability of the error in prediction changing at least one of the decision variables also increases. Consequently, there is a higher likelihood of error in the final solution. As the regret of the PF approach is significantly higher, note that the scale of the y-axis is changed to fit it into the plot.

Among the DFL techniques, Listwise performs best for sizes 12, 18, and 30 and SPO performs best for size 30. In fact, for sizes 12, 18, and 24, there are not many variations between SPO, Listwise, and MAP. After them, the next three best-performing techniques are Pairwise (diff), I-MLE and DBB. However, for size 30, DBB comes third after Listwise and MAP, followed by Pairwise (diff), SPO, and I-MLE in that order. FY and Pairwise perform slightly worse than the other DFL techniques. In general, this set of experiments shows the advantage of the DFL approaches as all of them outperform the PF approach.

Energy-cost aware scheduling. There are three instances of this scheduling problem. All the instances have 3 machines. The first, second, and third instances contain 10, 15, and 20 tasks, respectively. In this problem, the underlying ML model is a simple linear model implemented as a neural network model with no hidden layers. The boxplot of comparative evaluations for the first instance is presented in Figure 8. The boxplots of the other instances can be found in Figure A4 in the Appendix.

Note that the scheduling problem is an ILP problem. For HSD and QPTL, the LPs obtained by relaxing the integrality constraints have been considered. QPTL and HSD also perform poorly in all three instances. Listwise and Pairwise LTR losses also result to higher



(a) Capacity= 60

Figure 9: Comparative evaluations on the knapsack problem instances. This boxplot shows the distributions of relative regrets

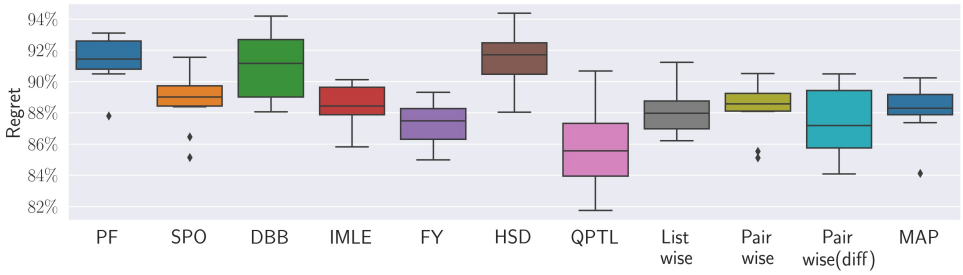
regret than the PF approach. On the other hand, FY, SPO, I-MLE, Pairwise(diff) and MAP come out as the best performing ones closely followed by DBB.

In general, across the three problem instances, it is possible to identify some common patterns. The first one is relaxing the integrality constraints fails to capture the essence of the combinatorial nature of the LP. Consequently, HSD and QPTL perform poorly. Secondly, Listwise and Pairwise ranking performances are significantly worse than the PF approaches. The learning curve suggests (refer to Appendix B), these methods fail to converge in these problem instances, although in some epochs, they are able to perform significantly better than the PF approach, their performances never plateau. Lastly, SPO, MAP, FY, and I-MLE perform consistently better than the rest.

Knapsack problem. Three instantiations of the knapsack problem are considered for the experiment—each instantiation with a different capacity. The three capacity values are—60, 120 and 180. The boxplot corresponding to capacity value 60 is presented in Figure 9. The boxplots of the other two capacities can be found in Figure A5 in the Appendix. With a capacity of 60, QPTL performs the best, whereas DBB, and I-MLE, HSD, SPO, and MAP also results in lower regret than the PF approach. With capacity values of 120 and 180, DBB has the lowest regret. Again I-MLE and SPO, HSD and QPTL result in lower regret than PF. However, MAP, LTR losses, and FY perform poorly especially for high capacity values.

Diverse bipartite matching. Three instantiations of the diverse bipartite matching problem are formed by changing the values of ρ_1 and ρ_2. The values of (ρ_1, ρ_2) for the three instantiations are $(10\%, 10\%)$, $(25\%, 25\%)$, $(50\%, 50\%)$ respectively. As mentioned before, in this problem, each edge is not associated with an edge weight in the true sense. Hence, the PF approach is trained by directly learning to predict whether an edge exists. So the loss used for supervised learning for the PF approach is **BCE** loss. The DFL approaches consider the predicted probability of each edge as the edge weight and then aim to minimize regret.

The boxplot of comparative evaluations for (ρ_1, ρ_2) being $(50\%, 50\%)$, is presented in Figure 10. The boxplots of the other two instances can be found in Figure A6 in the Appendix. Firstly note that relative regrets of all the techniques are very high (higher than



(a) $\rho_1 = \rho_2 = 50\%$

Figure 10: Comparative evaluations on the diverse bipartite matching problem instances. This boxplot shows the distributions of relative regrets.

80%) for all three instances. With ρ_1 and ρ_2 being 10%, I-MLE performs considerably better than all the other DFL techniques. When ρ_1 and ρ_2 take the value of 25%, QPTL, I-MLE and HSD are the top there techniques, with significantly lower regret than the rest. In the first two instances, other than I-MLE and QPTL, other DFL techniques do not significantly better than the PF approach. DFL techniques such as SPO, FY, Listwise, Pairwise and MAP perform considerably better than the PF approach only in the third instances.

Learning subset selections. Subset selection problems of three dimensions: $n = 25$, $n = 50$, and $n = 100$ are considered for evaluation. In each case, the subset size k is chosen to be $\frac{n}{5}$. The error of any predicted subset $\hat{x}$, with respect to ground truth x, is considered to be the fraction of items which are selected in x but not in $\hat{x}$. Such occurrences are referred to as mismatches.

Figure 11 shows the average mismatch rates over the size $n = 25$ instances that were achieved by each DFL technique listed in Table 1, excluding those which assume ground-truth data in the form of problem parameters. Here, the ground-truth data are optimal solutions of (49) representing subset selections. For each assessed technique, a distribution of results is shown, corresponding to 10 different randomly generated training datsets. Figure A7 shows similar results over the larger problem instances.

Note that it is suggested in (Amos et al., 2019) that the entropy function $H(\mathbf{x}) = \sum_i x_i \log x_i$ is particularly well-suited as a regularizer of the objective in (49), for the purpose of multilabel classification, which is identical to the task in terms of its optimization component and the form of its target data. Hence a *Cvxpylayers* implementation of this model is included and referred to as ENT.

Figure A7 shows that most of the assessed techniques perform similarly, with DBB performing worst regardless of the problem's dimension. HSD is most sensitive with respect to the randomly generated training set; the rest show consistent performance across datasets. QPTL and IMLE each show a marginal advantage over the other techniques, but DPO and ENT are also competitive. Across all techniques, variation in performance over the randomly generated datasets tends to diminish as problem size increases.

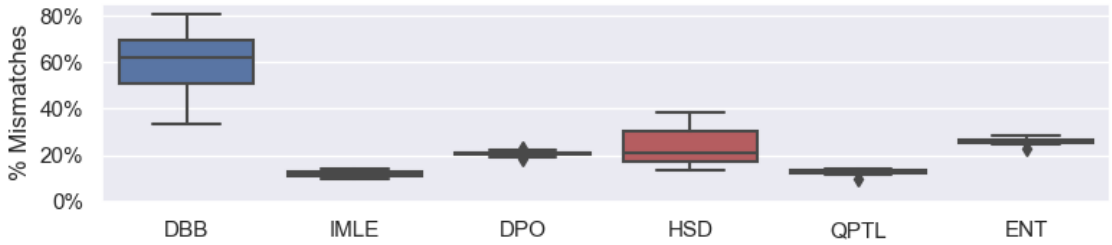
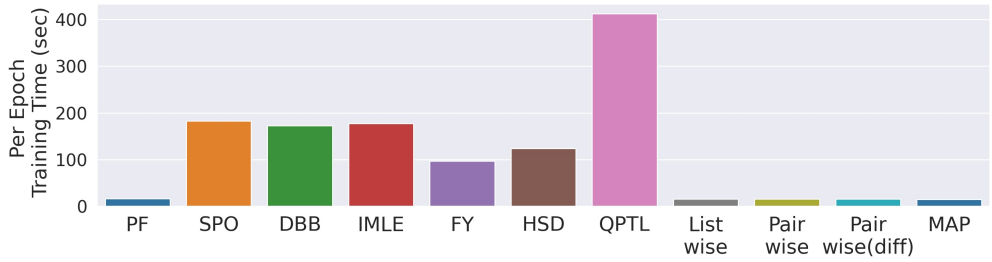
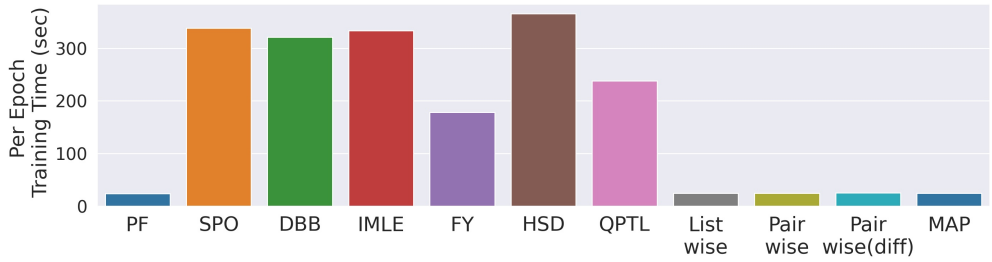


Figure 11: Comparative evaluations on the subset selection problem instances of Size 25. This boxplot shows the distributions of mismatch rates.



(a) Instance 2



(b) Instance 3

Figure 12: Comparative evaluations of per epoch training time of different DFL techniques on the energy-cost aware scheduling problem.

5.2.2 Comparison on Runtime

While training the ML model to minimize the task loss is the primary challenge of DFL, the computational cost associated with repeatedly solving CO problems gives rise to the second challenge. DFL techniques with low computational cost are essential for scalability for implementing DFL for real-world large-scale *Predict-Then-Optimize* problems. The importance of scalability and low computational cost becomes significant while dealing with large-scale CO problems, especially NP-hard combinatorial optimization problems. Note that while the shortest path and the knapsack problems are relatively easy to solve; the energy-cost aware scheduling problem is much more challenging. That is why the scheduling problem is considered to compare the computational costs of the DFL techniques.

The median training time of an epoch during training of each technique for two instances of the scheduling problem is shown Figure 12. Recall that the first, second and third instances contain 10, 15 and 20 tasks respectively. So, the first one is the easiest of the three and the third one is the hardest one. The complexity of the scheduling problem is evident from the fact that a single instance of the knapsack problem takes 0.001 seconds to solve, while solving the most difficult instance of the scheduling problem takes 0.1 seconds, both using Gurobi MIP solver. The readers are cautioned against placing excessive emphasis on the absolute values of training times in Figure 12, as they are subject to system overhead. However, some general conclusions can be drawn from the relative ordering of the training times. The training time of the PF approach is the lowest, as it does not require solving the CO problem for training. Training times of SPO, DBB, I-MLE and FY are almost 100 times higher than the PF approach. Although QPTL and HSD consider the relaxed LP problem, it is not always the case that they have lower training times. Recall that QPTL and HSD solve and differentiate the optimization problem using a primal-dual solver, which involves matrix factorization. On the other hand, SPO, DBB, I-MLE and FY can leverage faster commercial CO solvers, as they only require the optimal solution. However, for Instance 3, it seems solving the ILP problem by calling a CO solver, is more computationally expensive than solving and differentiating the underlying QP problem using *cvxpylayers*.

On the other hand, Listwise, Pairwise, Pairwise(diff) and MAP, all of which are run with $p_{solve} = 5\%$, exhibit significantly lower training time than the other DFL techniques. From this perspective, these techniques lie between PF and DFL approaches, balancing the trade-off between scalability and quality. The same conclusion generally holds true for other experiments as well. However, for relatively easier CO problems, the system overhead time may dominate over training time, which might disrupt the ordering of the training time.

5.2.3 Discussion

The experimental evaluations reveal that no single DFL technique performs the best across all experiments. Some techniques work well on particular test problems, while others work better on other test problems. Nevertheless, the following interesting characteristics can be observed in the experimental evaluations:

(i) **The performance of SPO is consistently robust across the test problems**, even though it may not outperform other techniques in every experiment.

(ii) I-MLE, FY, DBB and QPTL perform worse than the PF approach for the **portfolio optimization problem, where a quadratic constraint is present.**

(iii) **QPTL demonstrates robust performance when the relaxed LP is a good approximation of the ILP**. QPTL outperforms other techniques by a substantial margin in the bipartite matching problem. However, QPTL performs poorly compared to others in the scheduling problem, which is an ILP. In this case, the poor performance may be attributed to the fact that QPTL considers a relaxation of the ILP. In this problem, the LP solution might differ significantly from the true ILP solution. This is not the case for the knapsack problem, because the solution of the relaxed LP does not deviate significantly from the ILP solution for the knapsack problem.

(iv) **MAP also demonstrates consistent performance across most test problems**; it exhibits higher regret specifically in the knapsack problem for Capacity=180 and in the bipartite matching problem when ρ_1 and ρ_2 are 25%. It results in low regret in the portfolio optimization problem; where some DFL techniques perform poorly.

(v) Among the LTR losses, **Listwise and Pairwise often exhibit high variances, especially in the scheduling and the knapsack problems**. The performance of Pairwise(diff) stands out among the LTR losses due to its lower variance. Its performance is comparable to or slightly worse than MAP for most problems other than the synthetic shortest path problem with high values of Deg, i.e., when the underlying predictive model is completely misspecified.

(vi) Due to the limitation of computational resources, it was not possible to run QPTL and HSD on the Warcraft shortest path problem. This highlights the **advantage of DFL techniques which can make use of any blackbox combinatorial solver (Dijkstra's shortest path solver for instance) to solve the CO problem**.

(vii) Continuing on this topic of computational cost, **MAP and the LTR losses are considerably faster and less computationally intensive when they are run with low values of** p_{solve}. For large-scale real-world CO problems, the repeated solving of which might be prohibitive, MAP could be used to implement DFL, as MAP tends to have regret as low as SPO in most test problems.

6. Future Research Directions

While there is increasing interest in decision-focused learning research, more work is required to enable its application in a wider range of real-world problems. This section aims to summarize some challenges in DFL research that remain open.

Generalizing DFL across related tasks. Existing DFL methods typically tailor the machine learning model to a specific combinatorial optimization task. However, in many real-world applications, the CO problem can vary across different instances. For instance, in the MIT-Amazon Last Mile Routing Challenge (Merchán, Arora, Pachon, Konduri, Winkenbach, Parks, & Noszek, 2024), a Traveling Salesman Problem is solved daily to determine the routing of last-mile package delivery. The nodes in these Traveling Salesman Problem instances change daily as delivery locations vary. A promising research direction would be to investigate the performance of a model trained to minimize regret on one CO problem instance when evaluated on similar but different CO problem instances. Other research directions could involve the integration of multi-task learning with decision focused learning. This exploration could lead to the development of more versatile and robust DFL models capable of generalizing across a range of related tasks, thereby greatly enhancing their practical applicability.

Non-linear objective function. Most DFL research has focused on optimization problems with linear objective functions, as those emphasized in the experimental evaluations in this article. However, many real-world combinatorial optimization problems feature non-linear objective functions and discrete decision variables. For instance, optimally locating

substations in an electrical network to minimize distribution costs is a problem often formulated as non-linear programming (Lakhera, Shanbhag, & McInerney, 2011). Similarly, minimizing the makespan in flowshop scheduling is another classic operations research problem that lacks a linear objective function. While *cvxpylayers* (Agrawal et al., 2019a) allows for differentiation through convex optimization problems with non-linear objective functions, most techniques discussed in this paper are not equipped to handle such scenarios. Future research should focus on developing and evaluating DFL techniques that can effectively address combinatorial optimization problems with non-linear objective functions. This advancement would significantly expand the applicability and impact of DFL in solving complex real-world problems.

Robust risk-sensitive DFL. Most DFL techniques formulate empirical risk minimization problems to minimize expected regret. However, in many real-world applications, particularly those that are risk-sensitive, safety concerns renders the focus from expected regret less relevant. A more viable alternative in these cases would be to focus on regret in distributional worst-case scenarios. This necessitates minimizing risk-sensitive losses, such as value-at-risk or min-max regret. To address these cases, future research could draw inspiration from robust optimization (Ben-Tal, El Ghaoui, & Nemirovski, 2009), a well-established field within operations research that addresses worst-case regret within an uncertainty set. The prediction-focused approach could be adapted to design an "estimate-then-optimize" strategy (Qi, Grigas, & Shen, 2023), incorporating risk-sensitive loss functions by creating a weight-based empirical distribution to represent the uncertainty set (Sun, Liu, & Li, 2023). However, the DFL methodologies reviewed in this article are not designed to minimize risk-sensitive losses. While a recent work (Chenreddy, Bandi, & Delage, 2022) proposes a data-driven approach for constructing uncertainty sets using contextual features, the literature on robust DFL remains sparse, offering ample opportunities for future research.

Decision-focused learning by zeroth-order gradient. An alternative approach to gradient estimation is to adopt zeroth-order gradient estimation, particularly through the score function gradient estimation technique (Williams, 1992). This method has been recently adapted to DFL, where it assumes that the predicted parameter adheres to a specific distribution, and the model is trained accordingly to predict this distribution's parameters (Silvestri, Berden, Mandi, İrfan Mahmutoğulları, Mulamba, Filippo, Guns, & Lombardi, 2023). In principle, this method can also be used to predict parameters in the constraints of CO problems. However, despite providing an unbiased gradient, a significant drawback of the score function gradient estimation is its susceptibility to high variances, which can destabilize the learning process. Addressing this challenge could lead to substantial improvements. Enhancements could focus on variance reduction techniques or the development of more robust estimation algorithms.

Bilevel Optimization Techniques for DFL. As mentioned in Section 2.2, the empirical regret minimization problem can be cast as a pessimistic bilevel optimization problem. A deeper understanding of the mathematical foundations behind this learning process can lead to the development of more effective algorithms for DFL. Recent efforts by Bucarey, Calderón, Muñoz, and Semet (2023) have reformulated the problem as a quadratic non-convex optimization problem. However, the scalability of this approach remains a significant challenge. This presents a valuable opportunity for the bilevel optimization community

to develop scalable and efficient solutions, advancing the field of DFL and enhancing its practical applications.

Scalable DFL. While surrogate solvers and solution caching aim to address the scalability challenges in DFL, the trade-off between quality and scalability of these methods is not yet understood. In the context of solution caching, there is ample space for research to investigate the trade-off between the solution cache size and the resulting solution quality. Determining the optimal cache size that balances computational efficiency with high-quality solutions could lead to significant improvements in DFL performance. Additionally, the effectiveness of using pre-trained surrogate models as proxy CO solvers remains an open question. Specifically, it is unclear whether these surrogates can achieve regret levels comparable to those obtained using traditional CO solvers. Future research should explore the conditions under which surrogate solvers can approximate the performance of CO solvers, and identify strategies to enhance their accuracy and reliability. This line of research could reveal new insights into the design of more scalable DFL systems, which is one of the most important challenges to broaden their applicability to a wide range of complex problems.

Theoretical guarantees. While there has been extensive work on providing theoretical guarantees for the SPO+ loss, not all DFL methods offer theoretical guarantees on minimizing regret. Empirical results presented in this article suggest that the performance of some DFL methods may be suboptimal when dealing with combinatorial optimization problems that contain non-linear constraints. This gap underscores the need for further theoretical developments to establish the reliability and effectiveness of DFL methods across a broader range of optimization scenarios. Ensuring robust theoretical guarantees for DFL methods, particularly in the presence of non-linear constraints, would significantly enhance their trustworthiness especially important for safety-critical applications. This involves developing new analytical tools and techniques to prove the regret-minimizing properties of DFL methods under various conditions. Moreover, research should also focus on extending existing theoretical frameworks to encompass a wider array of CO problems.

Uncertainty in the constraints. As reviewed in Section 3.3.3, some recent works have focused on DFL methods for predicting parameters within the constraints of optimization problems. However, these methods have not been explored to the same extent as those predicting objective function parameters, leaving several open research directions for future investigation. For instance, future studies could extend the Neural Combinatorial Optimization (Bello, Pham, Le, Norouzi, & Bengio, 2017) and NCE (Mulamba et al., 2021) approaches by considering the prediction of parameters within the constraints. A significant challenge to implement the NCE approach in this setting is determining how to form the solution cache. Moreover, when predicting parameters in the constraints, the incorporation of risk-sensitive losses becomes particularly important. In these scenarios, the prescribed decision might not be feasible with respect to the true parameters. An intriguing research direction is to develop methods that recommend solutions feasible under extreme distributional variations of the parameters. This would involve creating a framework that jointly optimizes average performance while minimizing worst-case constraint violations. Such a framework could reveal new tracks for both theoretical research and practical applications, ultimately enhancing the practical utility of DFL in problems where constraint parameters are often uncertain and variable.

Extending DFL to multistage settings. DFL primarily aims to predict parameters for single combinatorial optimization problems. However, in many practical settings, decision-making occurs over multiple time periods, resulting in multistage optimization problems (Pflug & Pichler, 2014). This setting is prevalent in production and inventory management (Goyal & Gunasekaran, 1990), where optimal decisions are made over an extended time horizon in successive stages. At each stage, a subset of uncertain parameters is revealed sequentially, and the decision-maker adjusts their decisions to account for this new information. To date, no studies have investigated whether applying DFL to predict parameters in multistage optimization leads to improved final objectives.

Furthermore, in numerous AI and ML applications, optimization tasks often serve as intermediate steps between two machine learning stages. For example, consider the task of selecting relevant patches in high-resolution images for a downstream image recognition task. The patch selection task can be modeled as a Top-k selection problem (Cordonnier, Mahendran, Dosovitskiy, Weissenborn, Uszkoreit, & Unterthiner, 2021) and embedded as an intermediate layer between two neural networks. Here, the upstream neural network assigns scores to each patch, and the downstream neural network performs the recognition task using the Top-k patches. Techniques that implement differentiable optimization layers, such as I-MLE (Niepert et al., 2021), DBB (Pogančić et al., 2020), QPTL (Wilder et al., 2019a), and DPO (Berthet et al., 2020) can be applied embedded between the two neural networks. However, their effectiveness in these contexts needs to be empirically evaluated. Exploring DFL in multistage settings and its integration in intermediate ML tasks presents a promising research direction that could lead to the development of methods capable of handling sequential decision-making processes and complex optimization tasks embedded within broader ML frameworks.

DFL with Real-World Multimodal Datasets. In real-world applications, uncertainty can arise from various sources. Multimodal ML aims to train models that process data from diverse sources (Baltrušaitis, Ahuja, & Morency, 2019). In a "multimodal" CO problem, different parameters may be unknown and associated with distinct sets of features. For instance, in a facility location problem, both the transport cost and customer demands might be unknown (Cameron et al., 2022). Each of these parameters could be predicted by different ML models tailored to their respective feature sets. Integrating DFL with multimodal datasets introduces several new challenges and opportunities. Firstly, coordinating the predictions from multiple ML models to inform a single CO problem requires new methods to ensure coherence and compatibility between different predicted parameters. This might involve developing novel fusion techniques to combine information from different modalities effectively. Secondly, the interdependencies between different parameters need to be accurately captured. In multimodal settings, the prediction errors from different models might interact in complex ways, affecting the overall optimization outcome. Research should explore ways to model these interdependencies and mitigate the compounded impact of prediction errors. Finally, empirical evaluation of DFL methodologies on real-world multimodal datasets would be a crucial avenue to understand their practical utility and limitations. Such studies have the potential to significantly enhance the applicability of DFL in solving complex, real-world optimization problems characterized by diverse and uncertain data sources.

7. Conclusion

The survey article began by highligting the significance of *Predict-Then-Optimize* problem formulations, wherein an ML model is followed by a CO problem. The *Predict-Then-Optimize* problem has emerged as a powerful driving force in numerous real-world applications of AI, OR and business analytics. The key challenge in *Predict-Then-Optimize* problems is predicting the unknown CO problem parameters in a manner that yields high-quality *solutions*, in comparison to the retrospective solutions obtained when using the groundtruth parameters. To address this challenge, the DFL paradigm has been proposed, wherein the ML models are directly trained considering the CO problems using task losses that capture the error encountered after the CO problems.

This survey article provides a comprehensive overview of DFL, highlighting recent technological advancements, applications and identifying potential future research directions. Section 2, laid out the problem description with examples and then presented the two fundamental challenges of DFL: differentiation through the solutions of combinatorial optimization mapping and the computational cost associated with solving CO problems in the ML training loop. Section 3 then distinguished gradient-based DFL techniques from gradient-free DFL techniques, as well as categorizing gradient-based DFL into four distinct classes, thoroughly reviewing the trade-offs among them. Section 4 provided examples of DFL applications addressing real-world *Predict-Then-Optimize* problems across various domains. Furthermore, Section 5 offered extensive comparative evaluations on different problem sets, covering 11 DFL techniques.

Finally, Section 6 discussed some of the open challenges in DFL and outlined potential research directions. Addressing these challenges can pave the way for more robust and effective DFL methodologies, further enhancing their practical utility in solving complex optimization problems. We hope this survey and corresponding datasets and codes will serve as a catalyst, inspiring the application of decision-focused learning in diverse domains and contexts as well as stimulating further methodological research and advancements.

Acknowledgments

Jayanta Mandi and James Kotary should both be considered first authors. Jayanta Mandi was affiliated to Vrije Universiteit Brussel, Belgium during the submission of this article. This research is partly funded by the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt) and the European Union's Horizon 2020 research and innovation programme under grant agreement No 101070149, project Tuples. This research is also partially supported by NSF grants 2242931, 2232054, 2007164, and NSF CAREER award 2143706. Jayanta Mandi is supported by the Research Foundation-Flanders (FWO) project G0G3220N. Senne Berden is a fellow of the Research Foundation-Flanders (FWO-Vlaanderen, 11PQ024N). Victor Bucarey was funded by the ANID Fondecyt Iniciacion Grant no. 11220864.

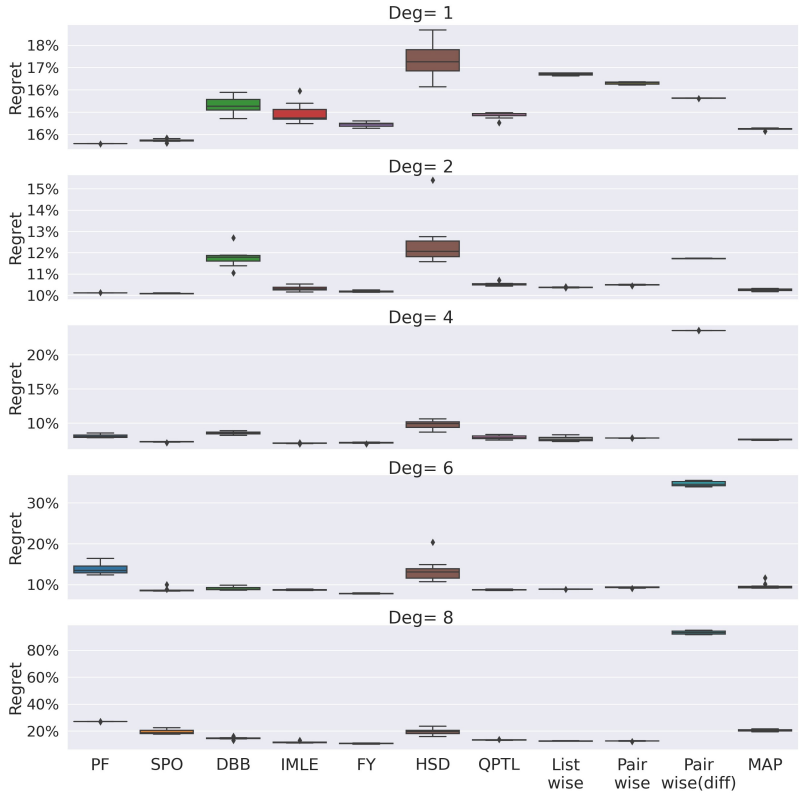


Figure A1: Comparative evaluations on the synthetic shortest path problem with noise-halfwidth parameter $\vartheta = 0.5$. These boxplots show the distributions of relative regrets.

Appendix A. Results on All Problem Instances

In the main text, boxplots of only selected problem instances are presented to save space in the main text. In this section of Appendix, the boxplots of all the instances are provided. Figure A1, Figure A2, Figure A3, Figure A4, Figure A5, Figure A6, and Figure A7 display the boxplots for all the instances of the grid shortest path problem, the portfolio optimization problem, the Warcraft shortest path problem, the energy-cost aware scheduling problem, the knapsack problem, the diverse bipartite matching problem, and the subset selection problem respectively.

Appendix B. Learning Curves

In this section, the learning curves of the LTR loss functions on the three instances of the scheduling problem are presented. Figure A8 reveals that learning with Pairwise (diff) ranking loss is stable whereas learning with Listwise and Pairwise ranking losses do not stabilize.

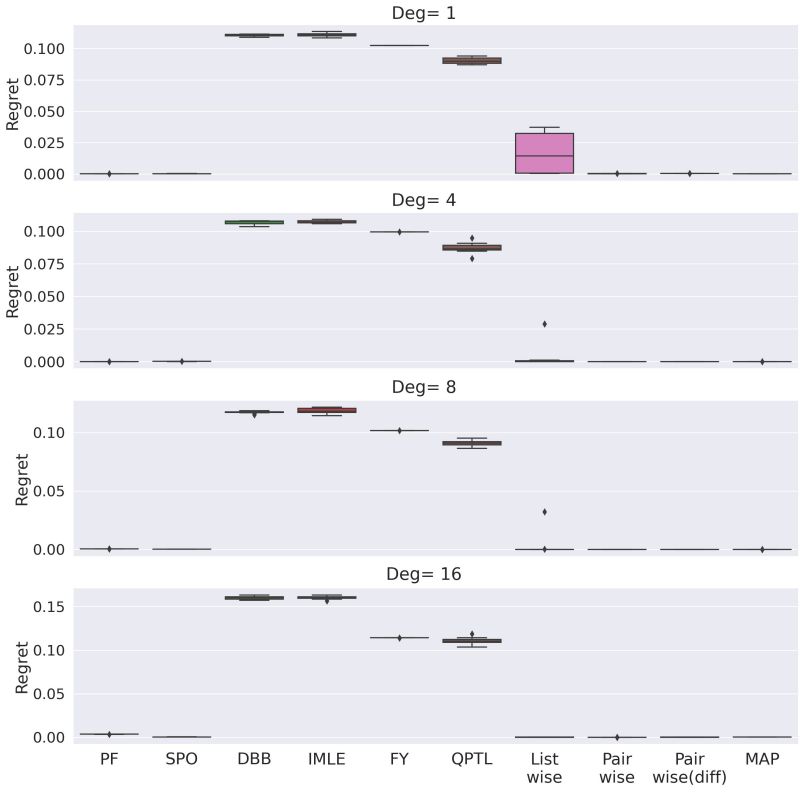


Figure A2: Comparative evaluations on the synthetic portfolio optimization problem with noise magnitude $\vartheta = 1$. These boxplots show the distributions of **absolute** regrets.

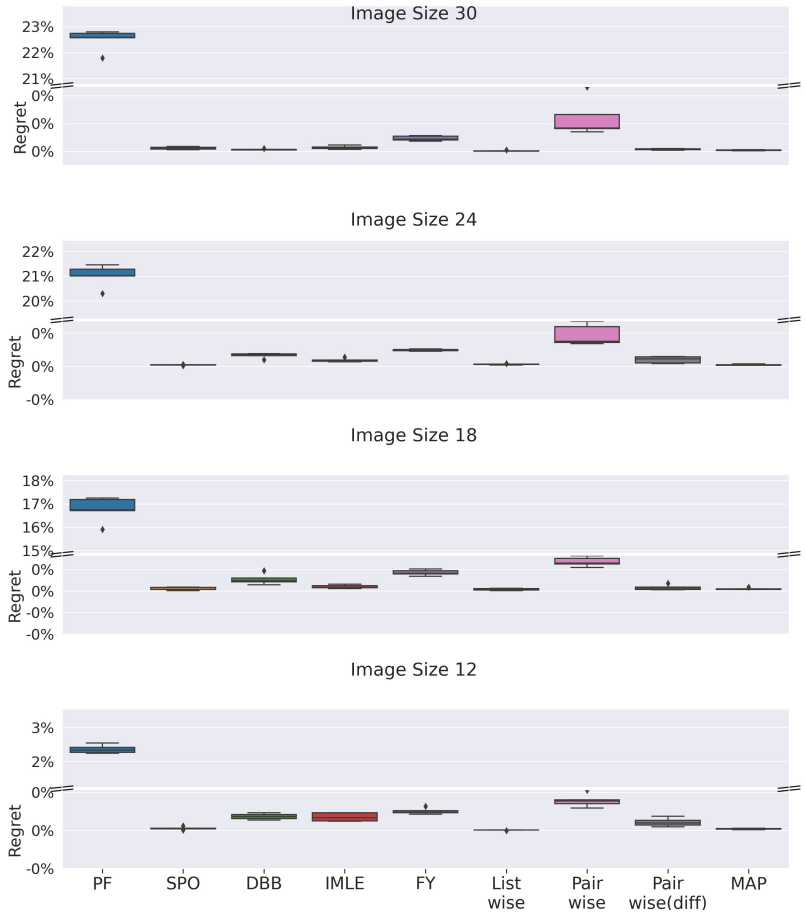


Figure A3: Comparative evaluations on the Warcraft shortest path problem instances. These boxplots show the distributions of relative regrets.

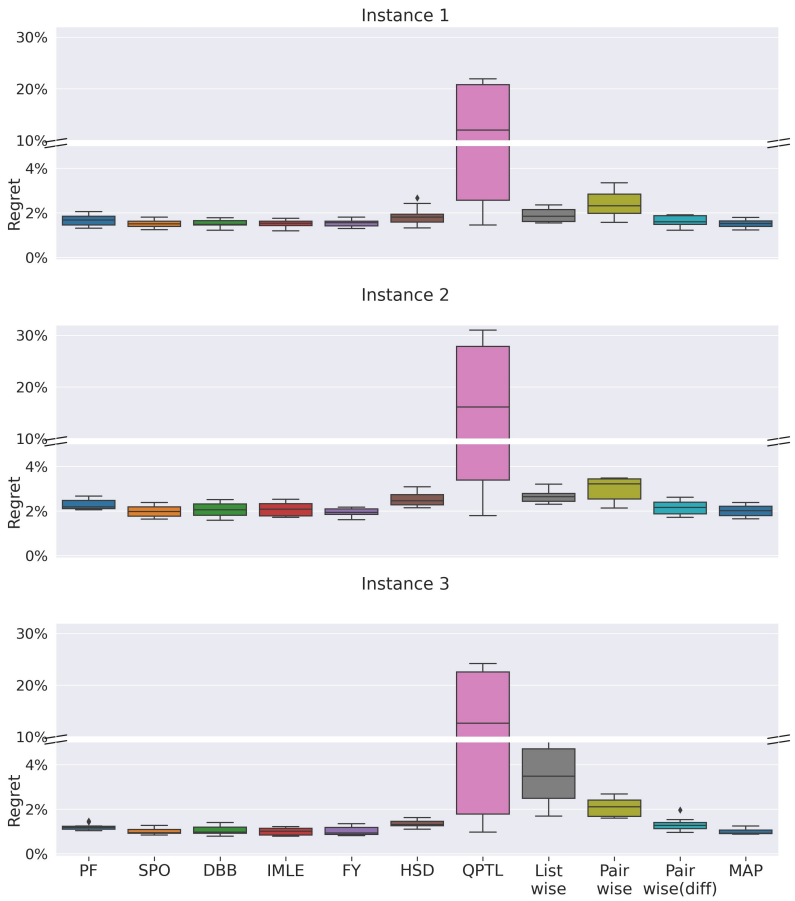


Figure A4: Comparative evaluations on the energy-cost aware scheduling problem instances. These boxplots show the distributions of relative regrets.

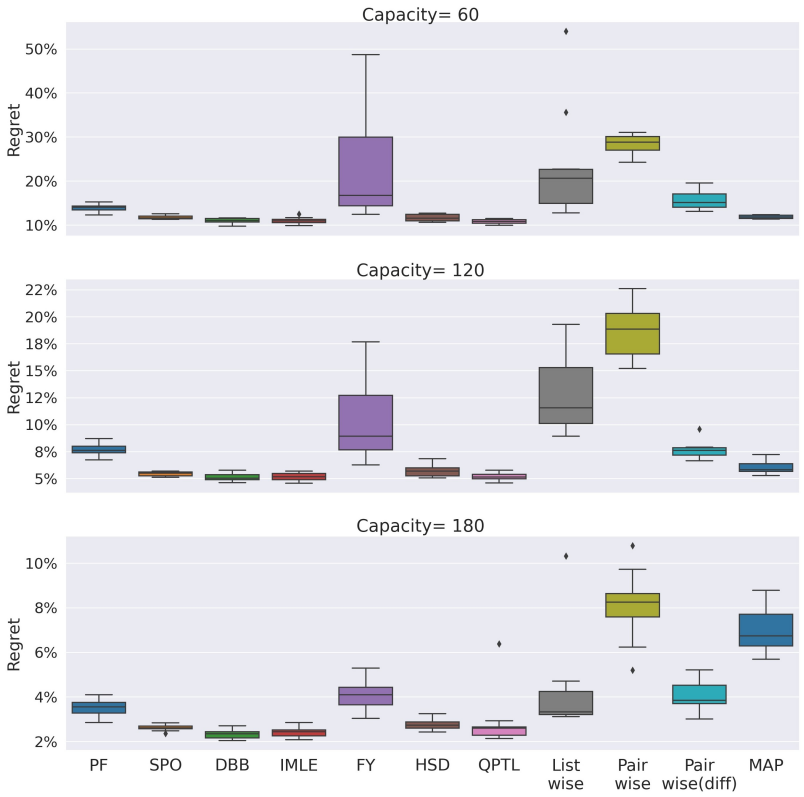


Figure A5: Comparative evaluations on the knapsack problem instances. These boxplots show the distributions of relative regrets.

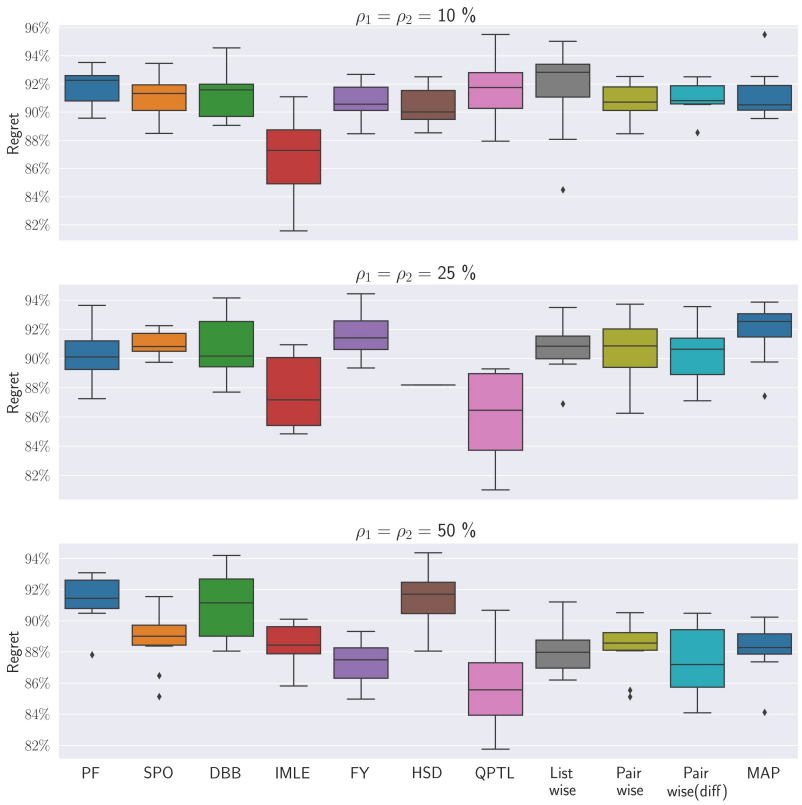


Figure A6: Comparative evaluations on the diverse bipartite matching problem instances. These boxplots show the distributions of relative regrets.

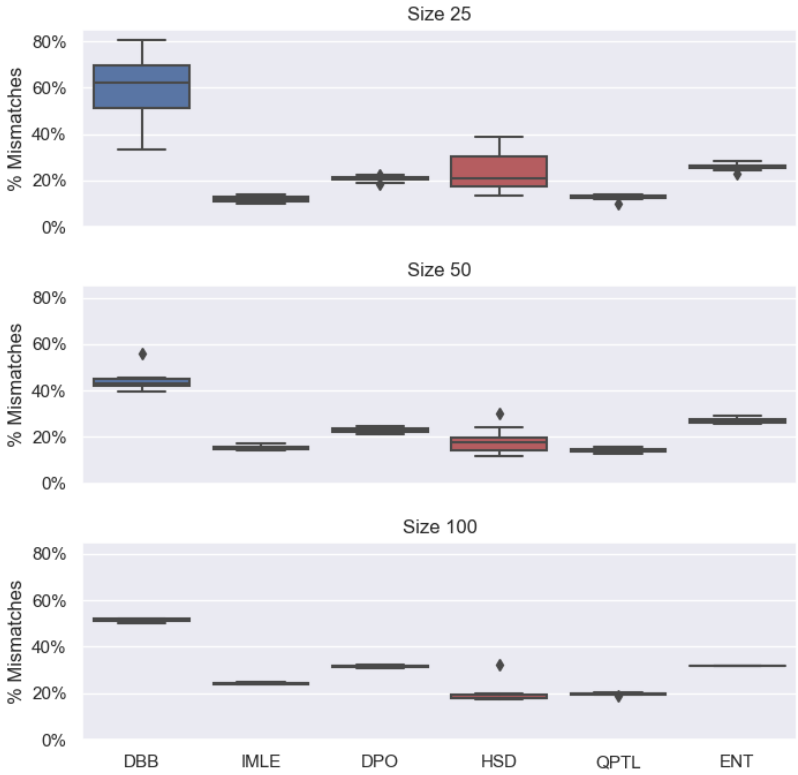


Figure A7: Comparative evaluations on the subset selection problem instances. This boxplot shows the distributions of mismatch rates.

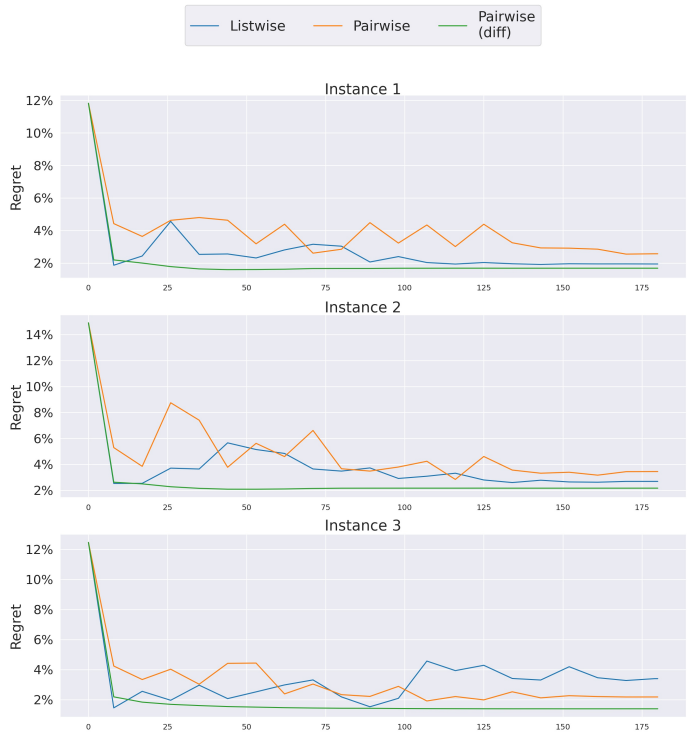
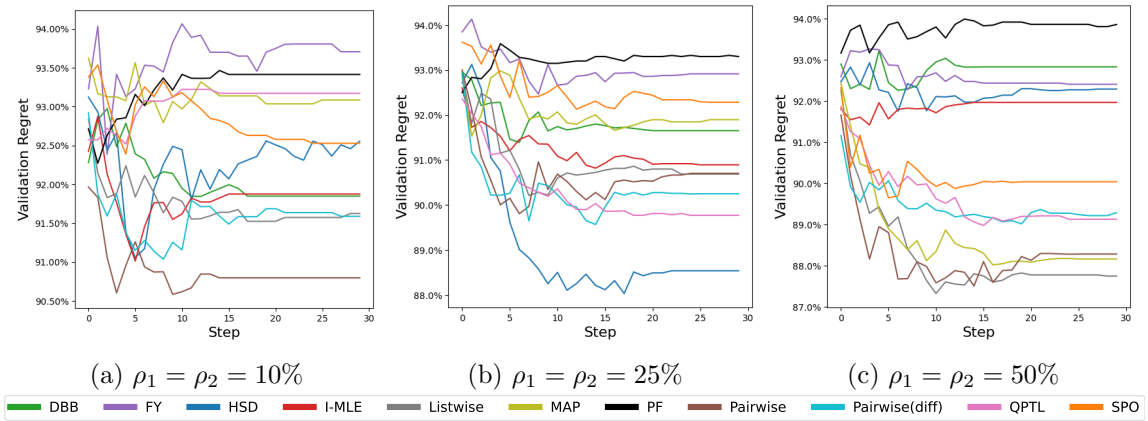


Figure A8: Learning Curves on the energy scheduling problem instances of for the LTR losses.



(a) $\rho_1 = \rho_2 = 10\%$ (b) $\rho_1 = \rho_2 = 25\%$ (c) $\rho_1 = \rho_2 = 50\%$

(d)

Figure A9: Learning Curves on the diverse bipartite matching problem.

Appendix C. Details about Hyperparameter Configuration

The hyperparameters for each methodology in each experiment are selected through grid search, as described in Section 5.2. For the sake of reproducibility lists of optimal hyperparameter combinations found by grid search and used to evaluate all the methodologies has been tabulated in the Online Appendix.

References

Abbas, A., & Swoboda, P. (2021). Combinatorial optimization for panoptic segmentation: A fully differentiable approach. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., & Vaughan, J. W. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 34, pp. 15635–15649. Curran Associates, Inc.

Abernethy, J., Lee, C., & Tewari, A. (2016). Perturbation techniques in online learning and optimization. *Perturbations, Optimization, and Statistics*, *233*.

Agarap, A. F. (2019). Deep learning using rectified linear units (relu)..

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., & Kolter, J. Z. (2019a). Differentiable convex optimization layers. *Advances in neural information processing systems*, *32*.

Agrawal, A., Barratt, S., Boyd, S., Busseti, E., & Moursi, W. M. (2019b). Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, *1*(2), 107 – 115.

Amos, B., & Kolter, J. Z. (2017). Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR.

Amos, B., Koltun, V., & Kolter, J. Z. (2019). The limited multi-label projection layer..

Baltrušaitis, T., Ahuja, C., & Morency, L.-P. (2019). Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(2), 423–443.

Barndorff-Nielsen, O. (1978). *Information and exponential families: in statistical theory*. John Wiley & Sons.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, *18*(153), 1–43.

Bazaraa, M. S., Jarvis, J. J., & Sherali, H. D. (2008). *Linear programming and network flows*. John Wiley & Sons.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net.

Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). *Robust optimization*, Vol. 28. Princeton university press.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., & Bach, F. (2020). Learning with differentiable perturbed optimizers. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., & Lin, H. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 33, pp. 9508–9519.

Bertsimas, D., & Kallus, N. (2020). From predictive to prescriptive analytics. *Management Science*, *66*(3), 1025–1044.

Blondel, M., Teboul, O., Berthet, Q., & Djolonga, J. (2020). Fast differentiable sorting and ranking. In *International Conference on Machine Learning*, pp. 950–959. PMLR.

Bollobás, B. (2013). *Modern graph theory*, Vol. 184. Springer Science & Business Media.

Boyd, S. P., & Vandenberghe, L. (2014). *Convex Optimization*. Cambridge University Press.

Bucarey, V., Calderón, S., Muñoz, G., & Semet, F. (2023). Decision-focused predictions via pessimistic bilevel optimization: a computational study..

Busseti, E., Moursi, W. M., & Boyd, S. (2019). Solution refinement at regular points of conic problems. *Computational Optimization and Applications*, *74*(3), 627–643.

Cameron, C., Hartford, J., Lundy, T., & Leyton-Brown, K. (2022). The perils of learning before optimizing. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*(4), 3708–3715.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136.

Chai, Z., Wong, K.-K., Tong, K.-F., Chen, Y., & Zhang, Y. (2022). Port selection for fluid antenna systems. *IEEE Communications Letters*, *26*(5), 1180–1184.

Chenreddy, A. R., Bandi, N., & Delage, E. (2022). Data-driven conditional robust optimization. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., & Oh, A. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 35, pp. 9525–9537. Curran Associates, Inc.

Chu, H., Zhang, W., Bai, P., & Chen, Y. (2023). Data-driven optimization for last-mile delivery. *Complex & Intelligent Systems*, *9*(3), 2271–2284.

Cordonnier, J.-B., Mahendran, A., Dosovitskiy, A., Weissenborn, D., Uszkoreit, J., & Unterthiner, T. (2021). Differentiable patch selection for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2351–2360.

Dalle, G., Baty, L., Bouvier, L., & Parmentier, A. (2022). Learning with combinatorial optimization layers: a probabilistic approach..

Delage, E., & Ye, Y. (2010). Distributionally robust optimization under moment uncertainty with application to data-driven problems. *Operations research*, *58*(3), 595–612.

Demirović, E., Stuckey, P. J., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., & Guns, T. (2019). An investigation into prediction+ optimisation for the knapsack problem. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4–7, 2019, Proceedings 16*, pp. 241–257. Springer.

Demirović, E., Stuckey, P. J., Guns, T., Bailey, J., Leckie, C., Ramamohanarao, K., & Chan, J. (2020). Dynamic programming for predict+optimise. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(02), 1444–1451.

den Hertog, D., & Postek, K. (2016). Bridging the gap between predictive and prescriptive analytics-new optimization methodology needed. In *Tilburg Univ, Tilburg, The Netherlands*.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271.

Dinh, M. H., Kotary, J., & Fioretto, F. (2024a). End-to-end learning for fair multiobjective optimization under uncertainty..

Dinh, M. H., Kotary, J., & Fioretto, F. (2024b). Learning fair ranking policies via differentiable optimization of ordered weighted averages. In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, pp. 2508–2517.

Djolonga, J., & Krause, A. (2017). Differentiable learning of submodular models. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.

Do, V., & Usunier, N. (2022). Optimizing generalized gini indices for fairness in rankings. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 737–747.

Domke, J. (2010). Implicit differentiation by perturbation. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., & Culotta, A. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 23. Curran Associates, Inc.

Domke, J. (2012). Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326. PMLR.

Donti, P. L., Kolter, J. Z., & Amos, B. (2017). Task-based end-to-end model learning in stochastic optimization. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5484–5494.

El Balghiti, O., Elmachtoub, A. N., Grigas, P., & Tewari, A. (2019). Generalization bounds in the predict-then-optimize framework. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc.

Elmachtoub, A., Liang, J. C. N., & McNellis, R. (2020). Decision trees for decision-making under the predict-then-optimize framework. In *International Conference on Machine Learning*, pp. 2858–2867. PMLR.

Elmachtoub, A. N., & Grigas, P. (2022). Smart "predict, then optimize". *Management Science*, *68*(1), 9–26.

Elmachtoub, A. N., Lam, H., Zhang, H., & Zhao, Y. (2023). Estimate-then-optimize versus integrated-estimation-optimization: A stochastic dominance perspective..

Falcon, W., et al. (2019). Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, *3*(6), 11.

Ferber, A., Griffin, E., Dilkina, B., Keskin, B., & Gore, M. (2023a). 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*.

Ferber, A. M., Huang, T., Zha, D., Schubert, M., Steiner, B., Dilkina, B., & Tian, Y. (2023b). Surco: Learning linear surrogates for combinatorial nonlinear optimization problems. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., & Scarlett, J. (Eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Vol. 202 of *Proceedings of Machine Learning Research*, pp. 10034–10052. PMLR.

Ferber, A. M., Wilder, B., Dilkina, B., & Tambe, M. (2020). Mipaal: Mixed integer program as a layer. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1504–1511. AAAI Press.

Garcia, J. D., Street, A., Homem-de Mello, T., & Muñoz, F. D. (2021). Application-driven learning: A closed-loop prediction and optimization approach applied to dynamic reserves and demand forecasting..

Garlappi, L., Uppal, R., & Wang, T. (2006). Portfolio Selection with Parameter and Model Uncertainty: A Multi-Prior Approach. *The Review of Financial Studies*, *20*(1), 41–81.

Gillick, D., Kulkarni, S., Lansing, L., Presta, A., Baldridge, J., Ie, E., & Garcia-Olano, D. (2019). Learning dense representations for entity retrieval. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 528–537, Hong Kong, China. Association for Computational Linguistics.

Gomes, C. P., Kautz, H., Sabharwal, A., & Selman, B. (2008). Satisfiability solvers. *Foundations of Artificial Intelligence*, *3*, 89–134.

Goodfellow, I. J. (2015). On distinguishability criteria for estimating generative models. In *Proceedings of ICLR*.

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In Bengio, Y., & LeCun, Y. (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., & Guo, E. (2016). On differentiating parameterized argmin and argmax problems with application to bi-level optimization..

Goyal, S. K., & Gunasekaran, A. (1990). Multi-stage production-inventory systems. *European Journal of Operational Research*, *46*(1), 1–20.

Grant, M. C., & Boyd, S. P. (2008). Graph implementations for nonsmooth convex programs. In *Recent advances in learning and control*, pp. 95–110. Springer.

Guler, A. U., Demirović, E., Chan, J., Bailey, J., Leckie, C., & Stuckey, P. J. (2022). A divide and conquer algorithm for predict+optimize with non-convex problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, *36*(4), 3749–3757.

Gurobi Optimization, L. (2021). Gurobi optimizer reference manual. `http://www.gurobi.com`.

Gutmann, M., & Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304.

Guyomarch, J. (2017). Warcraft ii open-source map editor. http://github.com/war2/war2edit.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network..

Hu, W., Wang, P., & Gooi, H. B. (2016). Toward optimal energy management of microgrids via robust two-stage optimization. *IEEE Transactions on smart grid*, *9*(2), 1161–1174.

Hu, X., Lee, J. C. H., & Lee, J. H. M. (2023a). Branch & learn with post-hoc correction for predict+optimize with unknown parameters in constraints. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 264–280. Springer Nature Switzerland.

Hu, X., Lee, J. C. H., & Lee, J. H. (2023b). Two-stage predict+optimize for milps with unknown parameters in constraints. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., & Levine, S. (Eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Hu, X., Lee, J. C., & Lee, J. H. (2023c). Predict+optimize for packing and covering lps with unknown parameters in constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(4), 3987–3995.

Hu, X., Lee, J. C., Lee, J. H., & Zhong, A. Z. (2022). Branch & learn for recursively and iteratively solvable problems in predict+optimize. In Oh, A. H., Agarwal, A., Belgrave, D., & Cho, K. (Eds.), *Advances in Neural Information Processing Systems*.

Huang, P., He, X., Gao, J., Deng, L., Acero, A., & Heck, L. P. (2013). Learning deep structured semantic models for web search using clickthrough data. In He, Q., Iyengar, A., Nejdl, W., Pei, J., & Rastogi, R. (Eds.), *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pp. 2333–2338. ACM.

Ifrim, G., O'Sullivan, B., & Simonis, H. (2012). Properties of energy-price forecasts for scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pp. 957–972. Springer.

Ignizio, J. P., & Cavalier, T. M. (1994). *Linear programming*. Prentice-Hall, Inc.

Jeong, J., Jaggi, P., Butler, A., & Sanner, S. (2022). An exact symbolic reduction of linear smart Predict+Optimize to mixed integer linear programming. In *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162 of *Proceedings of Machine Learning Research*, pp. 10053–10067. PMLR.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pp. 133–142. ACM.

Johnson-Yu, S., Wang, K., Finocchiaro, J., Taneja, A., & Tambe, M. (2023). Modeling robustness in decision-focused learning as a stackelberg game. In *The 22nd International Conference on Autonomous Agents and Multiagent Systems*.

Kainmueller, D., Jug, F., Rother, C., & Myers, G. (2014). Active graph matching for automatic joint segmentation and annotation of c. elegans. In Golland, P., Hata, N., Barillot, C., Hornegger, J., & Howe, R. D. (Eds.), *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2014 - 17th International Conference, Boston, MA, USA, September 14-18, 2014, Proceedings, Part I*, Vol. 8673 of *Lecture Notes in Computer Science*, pp. 81–88. Springer.

Kim, S., Lewis, M. E., & White, C. C. (2005). Optimal vehicle routing with real-time traffic information. *IEEE Transactions on Intelligent Transportation Systems*, *6*(2), 178–188.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y., & LeCun, Y. (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y., & LeCun, Y. (Eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kinsey, S. E., Tuck, W. W., Sinha, A., & Nguyen, T. H. (2023). An exploration of poisoning attacks on data-based decision making. In *Decision and Game Theory for Security: 13th International Conference, GameSec 2022, Pittsburgh, PA, USA, October 26–28, 2022, Proceedings*, pp. 231–252. Springer.

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Konishi, T., & Fukunaga, T. (2021). End-to-end learning for prediction and optimization with gradient boosting. In Hutter, F., Kersting, K., Lijffijt, J., & Valera, I. (Eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 191–207, Cham. Springer International Publishing.

Kotary, J., Di Vito, V., Christopher, J., Van Hentenryck, P., & Fioretto, F. (2023a). Predict-then-optimize by proxy: Learning joint models of prediction and optimization..

Kotary, J., Dinh, M. H., & Fioretto, F. (2023b). Backpropagation of unrolled solvers with folded optimization. In Elkind, E. (Ed.), *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pp. 1963–1970. International Joint Conferences on Artificial Intelligence Organization.

Kotary, J., Fioretto, F., Van Hentenryck, P., & Wilder, B. (2021). End-to-end constrained optimization learning: A survey. In Zhou, Z.-H. (Ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4475–4482. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Kotary, J., Fioretto, F., Van Hentenryck, P., & Zhu, Z. (2022). End-to-end learning for fair ranking systems. In *Proceedings of the ACM Web Conference 2022*, pp. 3520–3530.

Lakhera, S., Shanbhag, U. V., & McInerney, M. K. (2011). Approximating electrical distribution networks via mixed-integer nonlinear programming. *International Journal of Electrical Power & Energy Systems*, *33*(2), 245–257.

Levy, D., Carmon, Y., Duchi, J. C., & Sidford, A. (2020). Large-scale methods for distributionally robust optimization. *Advances in Neural Information Processing Systems*, *33*, 8847–8860.

Liu, B., & Liu, B. (2009). *Theory and practice of uncertain programming*, Vol. 239. Springer.

Liu, H., & Grigas, P. (2021). Risk bounds and calibration for a smart predict-then-optimize method. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., & Vaughan, J. W. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 34, pp. 22083–22094. Curran Associates, Inc.

Liu, M., Grigas, P., Liu, H., & Shen, Z.-J. M. (2023). Active learning in the predict-then-optimize framework: A margin-based approach..

Mandi, J., Bucarey, V., Tchomba, M. M. K., & Guns, T. (2022). Decision-focused learning: Through the lens of learning to rank. In *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162 of *Proceedings of Machine Learning Research*, pp. 14935–14947. PMLR.

Mandi, J., Canoy, R., Bucarey, V., & Guns, T. (2021). Data Driven VRP: A Neural Network Model to Learn Hidden Preferences for VRP. In Michel, L. D. (Ed.), *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, Vol. 210 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 42:1–42:17.

Mandi, J., Demirović, E., Stuckey, P. J., & Guns, T. (2020). Smart predict-and-optimize for hard combinatorial optimization problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(02), 1603–1610.

Mandi, J., & Guns, T. (2020). Interior point solving for lp-based prediction+optimisation. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., & Lin, H. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 33, pp. 7272–7282.

Martins, A., & Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pp. 1614–1623. PMLR.

Massart, P., & Nédélec, É. (2006). Risk bounds for statistical learning. *The Annals of Statistics*, *34*(5), 2326–2366.

Merchán, D., Arora, J., Pachon, J., Konduri, K., Winkenbach, M., Parks, S., & Noszek, J. (2024). 2021 amazon last mile routing research challenge: Data set. *Transportation Science*, *58*(1), 8–11.

Minervini, P., Franceschi, L., & Niepert, M. (2023). Adaptive perturbation-based gradient estimation for discrete latent variable models. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(8), 9200–9208.

Mišić, V. V., & Perakis, G. (2020). Data analytics in operations management: A review. *Manufacturing & Service Operations Management*, *22*(1), 158–169.

Mnih, A., & Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress.

Monga, V., Li, Y., & Eldar, Y. C. (2021). Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, *38*(2), 18–44.

Mulamba, M., Mandi, J., Diligenti, M., Lombardi, M., Bucarey, V., & Guns, T. (2021). Contrastive losses and solution caching for predict-and-optimize. In Zhou, Z.-H. (Ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 2833–2840. International Joint Conferences on Artificial Intelligence Organization.

Nandwani, Y., Ranjan, R., Mausam, & Singla, P. (2022). A solver-free framework for scalable learning in neural ilp architectures. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., & Oh, A. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 35, pp. 7972–7986. Curran Associates, Inc.

Nemirovski, A. (2007). Advances in convex optimization: conic programming. In *International Congress of Mathematicians*, Vol. 1, pp. 413–444.

Niepert, M., Minervini, P., & Franceschi, L. (2021). Implicit mle: Backpropagating through discrete exponential family distributions. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., & Vaughan, J. W. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 34, pp. 14567–14579. Curran Associates, Inc.

Ogryczak, W., & Śliwiński, T. (2003). On solving linear programs with the ordered weighted averaging objective. *European Journal of Operational Research*, *148*(1), 80–91.

Papandreou, G., & Yuille, A. L. (2011). Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In Metaxas, D. N., Quan, L., Sanfeliu, A., & Gool, L. V. (Eds.), *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pp. 193–200. IEEE Computer Society.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Paulus, A., Rolínek, M., Musil, V., Amos, B., & Martius, G. (2021). Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *International Conference on Machine Learning*, pp. 8443–8453. PMLR.

Paulus, M., Choi, D., Tarlow, D., Krause, A., & Maddison, C. J. (2020). Gradient estimation with stochastic softmax tricks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., & Lin, H. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 33, pp. 5691–5704. Curran Associates, Inc.

Perron, L., & Furnon, V. (2020). Or-tools..

Pflug, G. C., & Pichler, A. (2014). *Multistage stochastic optimization*, Vol. 1104. Springer.

Pisinger, D. (2005). Where are the hard knapsack problems?. *Computers & Operations Research*, *32*(9), 2271–2284.

Pisinger, D., & Toth, P. (1998). Knapsack problems. In *Handbook of combinatorial optimization*, pp. 299–428. Springer.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., & Rolinek, M. (2020). Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*.

PyTorch (2017). Pytorch: Reducelronplateau — pytorch 1.9.0 documentation.. `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html#torch.optim.lr_scheduler.ReduceLROnPlateau`.

Qi, M., Grigas, P., & Shen, Z.-J. M. (2023). Integrated conditional estimation-optimization..

Qi, M., & Shen, Z.-J. (2022). *Integrating prediction/estimation and optimization with applications in operations management*, pp. 36–58. INFORMS.

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In Xing, E. P., & Jebara, T. (Eds.), *Proceedings of the 31st International Conference on Machine Learning*, Vol. 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286, Bejing, China. PMLR.

Rolinek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., & Martius, G. (2020a). Optimizing rank-based metrics with blackbox differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., & Martius, G. (2020b). Deep graph matching via blackbox differentiation of combinatorial solvers. In Vedaldi, A., Bischof, H., Brox, T., & Frahm, J. (Eds.), *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVIII*, Vol. 12373 of *Lecture Notes in Computer Science*, pp. 407–424. Springer.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*, Vol. 2 of *Foundations of Artificial Intelligence*. Elsevier.

Ruszczyński, A., & Shapiro, A. (2003). Stochastic programming models. *Handbooks in operations research and management science*, *10*, 1–64.

Sadana, U., Chenreddy, A., Delage, E., Forel, A., Frejinger, E., & Vidal, T. (2024). A survey of contextual optimization methods for decision-making under uncertainty. In *European Journal of Operational Research*.

Sahinidis, N. V. (2004). Optimization under uncertainty: state-of-the-art and opportunities. *Computers & Chemical Engineering*, *28*(6), 971–983.

Sahoo, S. S., Paulus, A., Vlastelica, M., Musil, V., Kuleshov, V., & Martius, G. (2023). Backpropagation through combinatorial algorithms: Identity with projection works. In *The Eleventh International Conference on Learning Representations*.

Sang, L., Xu, Y., Long, H., Hu, Q., & Sun, H. (2022). Electricity price prediction for energy storage system arbitrage: A decision-focused approach. *IEEE Transactions on Smart Grid*, *13*(4), 2822–2832.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., & Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, *29*(3), 93.

Settles, B. (2009). Active learning literature survey. Computer sciences technical report 1648, University of Wisconsin–Madison.

Shah, S., Wang, K., Wilder, B., Perrault, A., & Tambe, M. (2022). Decision-focused learning without decision-making: Learning locally optimized decision losses. In Oh, A. H., Agarwal, A., Belgrave, D., & Cho, K. (Eds.), *Advances in Neural Information Processing Systems*.

Shah, S., Wilder, B., Perrault, A., & Tambe, M. (2024). Leaving the nest: Going beyond local loss functions for predict-then-optimize. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, pp. 14902–14909.

Silvestri, M., Berden, S., Mandi, J., İrfan Mahmutoğulları, A., Mulamba, M., Filippo, A. D., Guns, T., & Lombardi, M. (2023). Score function gradient estimation to widen the applicability of decision-focused learning..

Simonis, H., O'Sullivan, B., Mehta, D., Hurley, B., & Cauwer, M. D. (1999). CSPLib problem 059: Energy-cost aware scheduling. `http://www.csplib.org/Problems/prob059`.

Singh, A., & Joachims, T. (2018). Fairness of exposure in rankings. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2219–2228.

Sun, C., Liu, L., & Li, X. (2023). Predict-then-calibrate: A new perspective of robust contextual lp. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., & Levine, S. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 36, pp. 17713–17741. Curran Associates, Inc.

Tan, Y., Delong, A., & Terekhov, D. (2019). Deep inverse optimization. In Rousseau, L., & Stergiou, K. (Eds.), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, Vol. 11494 of *Lecture Notes in Computer Science*, pp. 540–556. Springer.

Tan, Y., Terekhov, D., & Delong, A. (2020). Learning linear programs from optimal decisions. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., & Lin, H. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 33, pp. 19738–19749.

Tang, B., & Khalil, E. B. (2023a). Multi-task predict-then-optimize. In Sellmann, M., & Tierney, K. (Eds.), *Learning and Intelligent Optimization*, pp. 506–522. Springer International Publishing.

Tang, B., & Khalil, E. B. (2023b). Pyepo: A pytorch-based end-to-end predict-then-optimize library for linear and integer programming..

Tian, X., Yan, R., Liu, Y., & Wang, S. (2023). A smart predict-then-optimize method for targeted and cost-effective maritime transportation. *Transportation Research Part B: Methodological*, *172*, 32–52.

Toth, P., & Vigo, D. (2015). *Vehicle routing: Problems, methods, and applications*. Society for Industrial and Applied Mathematics.

Tschiatschek, S., Sahin, A., & Krause, A. (2018). Differentiable submodular maximization. In Lang, J. (Ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 2731–2738. ijcai.org.

Tschora, L., Guns, T., Pierre, E., Plantevit, M., & Robardet, C. (2023). Electricity price forecasting based on order books: a differentiable optimization approach. In *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10.

Vapnik, V. (1999). An overview of statistical learning theory. *IEEE Trans. Neural Networks*, *10*(5), 988–999.

Wahdany, D., Schmitt, C., & Cremer, J. L. (2023). More than accuracy: end-to-end wind power forecasting that optimises the energy system. *Electric Power Systems Research*, *221*, 109384.

Wang, K., Shah, S., Chen, H., Perrault, A., Doshi-Velez, F., & Tambe, M. (2021). Learning mdps from features: Predict-then-optimize for sequential decision making by reinforcement learning. *Advances in Neural Information Processing Systems*, *34*, 8795–8806.

Wang, P.-W., Donti, P., Wilder, B., & Kolter, Z. (2019). Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554. PMLR.

Wang, R., Zhang, Y., Guo, Z., Chen, T., Yang, X., & Yan, J. (2023). LinSATNet: The positive linear satisfiability neural networks. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., & Scarlett, J. (Eds.), *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202 of *Proceedings of Machine Learning Research*, pp. 36605–36625. PMLR.

Wilder, B., Dilkina, B., & Tambe, M. (2019a). Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 1658–1665. AAAI Press.

Wilder, B., Ewing, E., Dilkina, B., & Tambe, M. (2019b). End to end learning and optimization on graphs. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 4674–4685.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, *8*, 229–256.

Wong, K.-K., Tong, K.-F., Zhang, Y., & Zhongbin, Z. (2020). Fluid antenna system for 6g: When bruce lee inspires wireless communications. *Electronics Letters*, *56*(24), 1288–1290.

Yang, Y., Yan, R., & Wang, H. (2022). Pairwise-comparison based semi-spo method for ship inspection planning in maritime transportation. *Journal of Marine Science and Engineering*, *10*(11), 1696.

Zou, H., Zhu, J., & Hastie, T. (2008). New multicategory boosting algorithms based on multicategory fisher-consistent losses. *The Annals of Applied Statistics*, *2*(4), 1290.