École Polytechnique Fédérale de Lausanne

Decision-Focused Learning for Transportation Network Design

by Luca Bataillard

Master Thesis

Approved by the Examining Committee:

Prof. Emma Frejinger, Université de Montréal
Thesis Advisor

Prof. Michel Bierlaire, EPFL
Thesis Co-Advisor

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences
C.P. 6128, succursale Centre-ville
Montréal (QC) H3C 3J7 Canada

September 28, 2023

Acknowledgments

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Emma Frejinger. Her enthusiastic guidance, advice and support throughout the process of researching and writing this Master's thesis has been truly invaluable. In addition, I am very grateful for the warm welcome and excellent travel recommendations I received during my time in Montréal. I am truly fortunate to have had the opportunity to learn under her mentorship.

I would also like to thank Prof. Michel Bierlaire, for supervising my research projects at the intersection of computer science and transportation throughout my studies, and especially for giving me the opportunity to complete my Master's thesis in Montréal. His valuable critiques and suggestions have contributed significantly to the quality of this thesis.

To the students and staff of CIRRELT, thank you for making coming to the lab such a friendly experience. I wish you all a very happy Wednesday!

To my family and friends, thank you for you encouragement and support. To my parents, Ann and Mario, thank you for being there for me, I could not have done it without you.

The completion of this thesis would not have been possible without the collective support of these people. Thank you all for your invaluable contributions.

Montréal, September 28, 2023 Luca Bataillard

Abstract

Solving operations research problems in transportation presents many challenges. Network Design (ND) models are complex combinatorial optimization problems used to plan transportation networks. The parameters of ND problems, such as commodity demand, are subject to uncertainty and must be estimated from noisy historical data. This is typically done separately from the optimization by machine learning or time-series models. These prediction models are trained using a prediction accuracy loss function. A recent approach to handling uncertainty in optimization problems is Decision-Focused Learning (DFL). It incorporates the optimization problem into the learning algorithm, ensuring that the predictions are aligned with the goal of making good decisions.

This master's thesis is an exploratory study of DFL for ND problems where the commodity demands are uncertain. Most research focuses on DFL for problems with continuous variables and uncertain objective costs. However, ND problems are combinatorial and the uncertain demand parameters are in the constraints. Our aim is to integrate the prediction of demands and the optimization of an ND problem so that the prediction results in high quality downstream decisions. Our objective is to conduct a review of the literature on ND, DFL, and Inverse Optimization (IO), identify existing approaches, and test their suitability for this problem.

In this study, we conduct a literature review of the fields of ND, DFL, and IO, and find that existing methods cannot be directly applied to the problem of DFL for ND. We formulate our problem as a stochastic optimization problem, and show how to evaluate the performance of a prediction model on the downstream cost of the ND problem. We show how the regret-based loss, the standard way of evaluating the downstream optimization cost, is mathematically ill-defined when the uncertainty is in the constraints. We formulate *IO-constraint*, an IO model that that trains a linear prediction to predict demands but corresponds to Ordinary Linear Regression. Finally, we reframe DFL as a problem of appropriately weighting the training examples in the loss function, and sketch ideas for finding effective weights using an iterative weight update algorithm.

Résumé

La résolution de problèmes de recherche opérationelle en transports présente de nombreux défis. Les modèles de « Network Design » (ND) sont des problèmes d'optimization combinatoire complexes, qui sont utilisés pour planifier des réseaux de transport. Les paramètres d'un modèle de ND, tels que la demande en marchandises, sont incertains et doivent être estimés à partir de données historiques avec du bruit. Cette estimation est généralement réalisée séparément de l'optimisation par des modèles d'apprentissage automatique ou de séries temporelles. Ces modèles de prédiction sont souvent entrainés avec une fonction de perte basée sur l'exactitude des prédictions. Le « Decision-Focused Learning » (DFL) est une approche récente pour gérer l'incertitude dans les problèmes d'optimization. Elle intègre le problème d'optmization dans l'algorithme d'apprentissage, garantissant que les prédictions sont alignées sur l'objectif de prendre de bonnes décisions.

Cette thèse de Master est une étude exploratoire du DFL pour les problèmes de ND où la demande en marchandise est incertaine. La majorité des études actuelles se concentrent sur le DFL pour des problèmes à variables continues et avec des coûts incertains. En revanche, les problèmes de ND sont de nature combinatoire et la demande en marchandise incertaine se situe dans les containtes. Nous visons à intégrer la prédiction de la demande et l'optimization d'un problème de ND de sorte que les prédictions aboutissent à des décisions en aval de haute qualité. Notre objectif est d'effectuer une revue de la litérature sur le ND, le DFL et l'optimization inverse (IO), d'identifier les approches existantes et de tester leur utilité pour ce problème.

Dans cette étude, nous effectuons une revue de la littérature dans les domaines du ND, du DFL et de l'IO, et nous constatons que les méthodes existantes ne peuvent pas être directement appliquées au problème du DFL pour le ND. Nous formulons notre problème comme un problème d'optimisation stochastique et montrons comment évaluer la performance d'un modèle de prédiction sur le coût en aval du problème de ND. Nous montrons comment les pertes basées sur le regret, la méthode standard d'évaluation du coût d'optimisation en aval, est mathématiquement mal définie lorsque l'incertitude se trouve dans les contraintes. Nous formulons *IO-constraint*, un modèle IO qui entraîne un modèle de prédiction linéaire pour prédire la demande, mais qui correspond à la régression linéaire ordinaire. Enfin, nous reformulons le DFL comme un problème de pondération appropriée des exemples d'entraînement

dans la fonction de perte, et nous esquissons des idées pour trouver des pondérations efficaces à l'aide d'un algorithme itératif de mise à jour des pondérations.

Contents

Chapter 1

Introduction

1.1 Decision-Focused Learning for ND Problems

In operations research, we are often tasked with finding high-quality solutions to problems where not all the information is known at the time of planning. For instance, in the field of freight railroad scheduling, planners design a schedule that fulfills the demand for cargo between different origins and destinations. We use algorithms and methods from mathematical optimization to find the most appropriate schedule. However, railroad schedules are planned months in advance, well before the exact demand is known. We thus have to make predictions about demand ahead of time, using statistical methods.

In order to make high-quality decisions, operations researchers typically model the situation as a mathematical optimization problem, which aims to find solutions that minimize or maximize a cost function, while respecting a set of constraints. Commonly, such optimization problems can be written as Mixed-Integer Linear Programs (MILP). In the case of deterministic MILP problems, the cost function and the constraints are linear and their coefficients are fixed and known. The coefficients are also called the parameters of the model.

Optimization models are used to make long-term strategic decisions, sometimes months or years in advance, which means some model parameters are not known exactly at the time of planning and need to be predicted. Researchers use statistical prediction models, such as time series or Machine Learning (ML) models, that use contextual data to estimate the future value of the parameters. The process of using contextual data to predict parameters to an optimization model, then using that model to find high-quality decisions is called the data-decisions pipeline, illustrated in Figure 1.1.

This project focuses on predicting the demand coefficients in Network Design (ND) problems formulated as MILPs, which are particularly difficult to solve optimally. From railroad scheduling

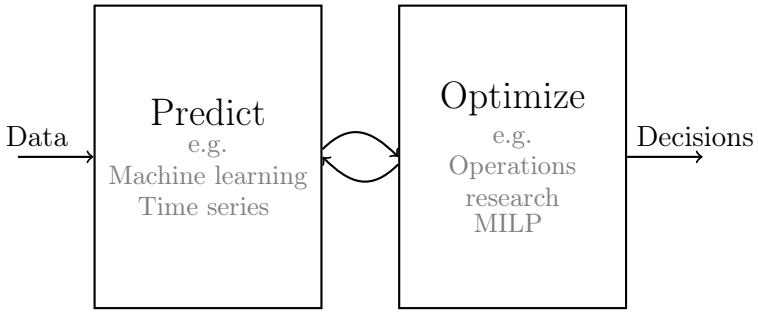


Figure 1.1: The data-decisions pipeline

to city logistics to airport hub placement, many problems in transportation and logistics can be formulated as ND problems. ND problems are defined on a graph consisting of nodes and arcs connecting them. Given a set of commodities: each commodity is transported from its origin node to its destination node according to its demand. The objective is to find the cheapest subset of arcs that satisfies the demand for every commodity while respecting the capacity of the arcs. This is a difficult problem to solve even if we assume that all model parameters are known in advance. In fact, not only is this problem computationally $\mathcal{NP}$-hard, but real-world instances are large, with tens of thousands of parameters, and standard approaches for approximating MILP solutions, such as Linear Program (LP) relaxations, work poorly on ND problems.

The parameters of the ND model, such as arc capacity, design and flow costs, and commodity demands, are not known at the time of planning. Their values are uncertain and are predicted using statistical models. There are several ways to handle the uncertainty in the parameters. Stochastic Network Design (Alonso-Ayuso et al., 2003) minimizes the expected costs over the probability distribution of the parameters. These models are computationally very difficult to optimize, and the parameter distributions are often hard to approximate. Robust Network Design (Koster et al., 2013) considers that parameters occur within an uncertainty set, and minimizes the worst-case cost within that uncertainty set. Robust optimization is an emerging field: there is little research applied to ND problems, and it presents similar computational difficulties to Stochastic Network Design. Finally, the most common way to handle the uncertainty is to make point predictions for the ND parameters, and then solve the deterministic formulation of the ND problem. This has the advantage of being much easier to solve computationally, at the cost of losing the information about the parameter distribution in the optimization phase. The deterministic ND formulation is the one most commonly used in practice, but it requires special attention to make good point predictions.

In this work, we focus on making point predictions for commodity demands. This is because, in practice, demand has the highest uncertainty and is the most difficult to predict from historical data, compared to the other model parameters. In fact, even with a good estimation of the probability distribution, it is difficult to make good point predictions of demand. Furthermore,

the demand parameters appear in the constraints of the ND model, which means that the predictions affect the feasible region of the optimization problem. Demand predictions will strongly affect the final network quality.

We train statistical models to make demand predictions from contextual data. This is typically done separately from the solving the optimization problem, which can lead to a mismatch between prediction accuracy and downstream optimization cost. The prediction model is trained on a dataset of demands and associated contextual features. Given the contextual features, it produces a conditional point estimate of the demand. During training, the model iteratively minimizes a loss function that compares the predicted demand to the actual demand, such as Mean Squared Error (MSE) or Mean Absolute Error (MAE). These measures of accuracy do not take into account the effect of the prediction error on the optimization step. Over- or underpredicting demand can radically affect the cost of the resulting network, even if two predictions have the same loss function value. Thus, integrating the prediction and optimization steps by evaluating demand predictions on the cost of the downstream optimization problem, rather than on pure prediction accuracy, could result in better performing networks.

1.2 An Illustrative Example of the Utility of Decision-Focused Learning

To illustrate the utility of integrating the prediction and optimization steps of a data-decisions pipeline, consider the following example, shown in Figure 1.2. Suppose we are trying to build the lowest cost network for transporting a single commodity from its origin s to its destination t. We have a choice of two possible arcs: the upper arc p has a cheaper design cost $f_p = 10$ but has a limited capacity of $u_p = 16$, and the lower arc q has a more expensive design cost of $f_q = 100$ but has unlimited capacity $u_q = \infty$.

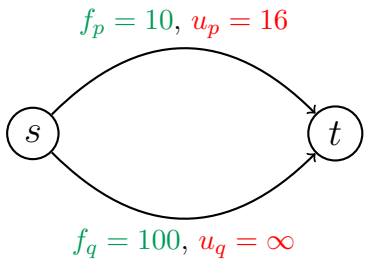


Figure 1.2: Example of a single commodity network with two arcs: a cheap but limited capacity arc p, and an expensive but unlimited capacity arc q

We will go into further detail in Chapter 2.1, but for now let us assume that we can formulate the network in Figure 1.2 as a mathematical optimization problem which we write MCFND(d), where d is the commodity demand. Further assume that there exists a procedure to solve MCFND(d) and obtain the optimal network that satisfies the demand d.

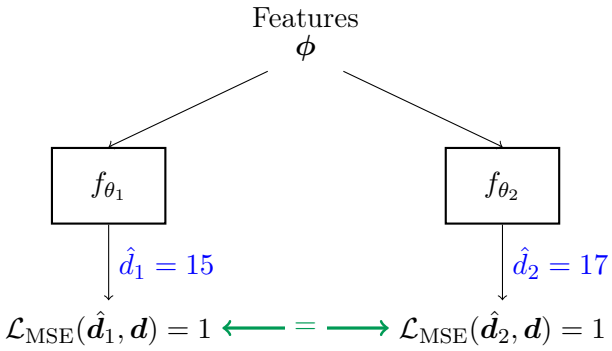


Figure 1.3: Comparing the predicted demands of two prediction models f_1 and f_2.

We let actual demand for the commodity be $d = 16$, but this demand is uncertain and is predicted from the associated contextual features ϕ. Consider the case of two prediction models f_1 and f_2, which, given the same contextual features ϕ, output two different predictions for the demand $\hat{d}_1 = f_1(\phi) = 15$ and $\hat{d}_2 = f_2(\phi) = 17$. If we evaluate these predictions with respect to the actual demand using standard evaluation metrics from machine learning, these two predictions obtain the same score. Indeed, in this case, both models obtain a Mean Squared Error score of $\mathcal{L}_{\text{MSE}}(\hat{d}_1, d) = \mathcal{L}_{\text{MSE}}(\hat{d}_2, d) = 1$. The two predictions made by these models are equivalent when measured in the prediction step, but this does not necessarily mean that the two predictions result the same downstream optimization cost.

In this example, equal prediction loss does not mean equal downstream optimization cost. Suppose we use each predicted demand in the MCFND to determine which arcs to build. In this case, we get two optimal networks with radically different costs, as shown in Figure 1.4. In this case, prediction models that slightly underpredict demand would result in a cheaper downstream network compared to models that overpredict demand. This example highlights the importance of integrating the prediction and optimization steps: training and evaluating prediction models using techniques that do not take into account the structure of the downstream optimization problem can lead to suboptimal decisions.

Figure 1.4: Comparison of resulting optimal network using each predicted demand. In bold are the arcs that are built to fulfill the respective demand. The cost of each network is different, despite identical prediction loss.

1.3 Objectives and Contributions

This project addresses the mismatch between the prediction and the optimization steps of a ND problem in which the commodity demands are uncertain. We integrate information about the optimization problem into the prediction step to obtain predicted demands that result in a cheaper downstream network. We refer to this process as *Decision-Focused Learning (DFL)* or also *Integrated Learning and Optimization (ILO)*. We have identified a gap in the DFL research. Most works focus on DFL for optimization problems with continuous variables and uncertainty in the objective function. Our project applies DFL to predicting demand in ND problems, which are combinatorial and present uncertainty in the constraints. Our objective is an exploratory study of DFL for ND. We aim to test existing approaches to identify their suitability for this problem.

In this project, we formulate an ND problem as a Contextual Stochastic Optimization (CSO) model. We study how to optimize such a model using a Prediction-Optimization pipeline, where the prediction step makes point predictions for the parameters and the optimization step solves the deterministic formulation of the ND problem. We show how to evaluate the performance of such a pipeline not in terms of prediction accuracy, but in terms of the cost of the downstream decisions. We show how the standard ways of evaluating the downstream optimization cost, regret-based losses, are mathematically ill-defined in cases where the uncertainty is in the constraints.

We then formulate *IO-Constraint*, a method that uses Inverse Optimization (IO) to train a linear prediction model that predicts demand from contextual information. We apply *IO-Constraint* to synthetic instances of ND problems and compare the resulting network cost of *IO-Constraint* to a linear regression model. We show that *IO-Constraint* actually corresponds to Ordinary Linear Regression. Next, we observe that DFL may be viewed as appropriately weighting training examples in the loss function. We explore how changing the weighting of training examples that have a high impact on downstream costs in the loss function can improve

the performance of the Prediction-Optimization pipeline. We present W-DFL, a nonlinear, multi-level optimization problem that defines the weightings that minimize the regret over the training dataset, and sketch ideas for finding effective weights using an iterative weight update algorithm.

1.4 Thesis Structure

This thesis is organized as follows. In Chapter 2, we review the existing research on ND, DFL, and IO. Using a common notation, we also introduce the mathematical notions from the literature used in our project. In Chapter 3, we formulate the ND problem and the Prediction-Optimization pipeline. We highlight issues related to evaluating the performance of a pipeline and the issues related to regret-based losses. We also formulate the *IO-Constraint* method for training prediction losses and explore how reweighting training examples in the loss function when training a prediction model can improve downstream costs. In Chapter 4, we evaluate the performance of *IO-Constraint* on synthetic ND examples that should demonstrate the advantage of DFL methods. We show an example of how reweighting training examples improves the performance of a Prediction-Optimization pipeline on an ND problem and evaluate the iterative weight update method on that example. Chapter 5, provides an overview of the work done in this thesis and presents avenues for future research.

Chapter 2

Literature Review

This chapter reviews the literature related to DFL for ND problems, and introduces the mathematical notions used in later sections with a common notation. We split this review into four sections. Section 2.1 covers the literature on ND, which are combinatorial optimization problems that compute the best network structure in a graph to fulfill demand between origin and destination points. Section 2.2 explores different approaches to DFL. Originating at the intersection of ML and operations research, DFL attempts to integrate a data-decisions pipeline's prediction and optimization steps. Section 2.3 focuses on IO, a field of mathematical optimization which consists of recovering model parameters given optimal solutions to the model. Finally, Section 2.4 concludes by discussing directions for our research project and current gaps in the literature.

2.1 Network Design

Many problems in operations research, particularly in transportation and logistics, can be expressed as ND problems. The literature related to ND is extensive (Cadarso et al., 2018, Gendron et al., 1999, Gomory and Hu, 1961, Hirsch and Dantzig, 1968), and a comprehensive review is out of the scope of this thesis. We base our review of ND on the book from Crainic et al. (2021). It offers a recent and extensive survey of ND, including its applications to transportation problems. They first present general formulations for ND problems and exact and approximate methods for solving them. They then explore specific applications of ND to transportation and logistics. We use the book's path-based formulation for the deterministic Multi-Commodity Fixed-Charge Network Design problem (MCFND). The authors showcase linear relaxations of the MCFND problem and valid inequalities that can lead to stronger bounds. The stochastic variant of the MCFND is also explored, in which design and flow costs, arc capacities, and commodity demands are not known exactly, but all follow a random distribution. The authors formulate the problem as a two-stage stochastic program and then

briefly explore solving such programs using Sample Average Approximation (SAA). The book mainly explores the optimization side of a ND data-decisions pipeline, with less attention paid to stochastic variants and estimation of problem parameters.

We formulate the MCFND here as in Crainic et al. (2021). The MCFND is defined on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ comprised of nodes $\mathcal{N}$ and arcs $\mathcal{A}$ connecting them. The formulation includes a set of commodities $\mathcal{K}$, each with an origin and destination node $\{(\text{orig}(k), \text{dest}(k)) : k \in \mathcal{K}\}$. Each commodity k has a demand d^k that the network must fulfill. At every node $n \in \mathcal{N}$, d_n^k specifies the demand for commodity $k \in \mathcal{K}$ as follows:

$$d_n^k = \begin{cases} -d^k, & n = \text{orig}(k), \\ d^k, & n = \text{dest}(k), \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Each arc $(i, j) \in \mathcal{A}$ has a capacity u_{ij}. It costs f_{ij} to build arc (i, j) and c_{ij}^k for commodity k to flow across it. Binary design variables $y_{ij} \in \{0, 1\}$ indicate if arc $(i, j) \in \mathcal{A}$ is built, and flow variables $x_{ij}^k \geq 0$ indicate the quantity of commodity $k \in \mathcal{K}$ flowing across arc $(i, j) \in \mathcal{A}$. We obtain the following formulation for the MCFND:
(MCFND)

$$\underset{\boldsymbol{x}, \boldsymbol{y}}{\text{minimize}} \quad \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij} + \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^k \tag{2.2a}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{N}_i^+} x_{ij}^k - \sum_{j \in \mathcal{N}_i^-} x_{ji}^k = d_i^k, \qquad \forall i \in \mathcal{N}, k \in \mathcal{K}, \tag{2.2b}$$

$$\sum_{k \in \mathcal{K}} x_{ij}^k \leq u_{ij} y_{ij}, \quad \forall (i, j) \in \mathcal{A}, \tag{2.2c}$$

$$x_{ij}^k \geq 0, \qquad \forall (i, j) \in \mathcal{A}, k \in \mathcal{K}, \tag{2.2d}$$

$$y_{ij} \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A}. \tag{2.2e}$$

Definition 2.1 (Shorthand notation for MCFND)**.** *The ND problem from (2.2) with demands* $\boldsymbol{d}$ *returning optimal decision variables* $x^*(\boldsymbol{d}), y^*(\boldsymbol{d})$ *is written in a shorter form:*

$$(x^*(\boldsymbol{d}), y^*(\boldsymbol{d})) = \text{MCFND}(\boldsymbol{d}).$$

When predicting the uncertain parameters of the MCFND, choosing the correct point prediction for commodity demand is just as important, if not more important, than the overall accuracy of the demand prediction. Laage et al. (2021) demonstrate the benefits of greater integration between the demand prediction step and the downstream optimization problem. The authors address the Periodic Demand Estimation (PDE) problem for cyclic Service Network Design:

a time horizon is divided into multiple periods and demand is predicted for each period over the time horizon. The multiple demand predictions are mapped to a single *periodic demand*, traditionally the average of the predictions, but in this case it could be the mean, the median, the third quartile, or the maximum. Given this periodic demand, the system solves a multi-level optimization problem that minimizes the total design and flow costs for a cyclic service network. The top level designs a service network that fulfils the periodic demand. The network structure is repeated for each period in the time horizon. The lower level then uses the designed network and adjusts the flows to fulfill the actual demand for that period, using recourse actions. The overall PDE problem aims to find, for each commodity, the mapping that minimizes the total cost of the bilevel problem. Allowing the PDE problem to choose from several types of mappings, i.e., reweighting the final demand forecasts, can significantly improve the overall problem cost on real-world large-scale Service Network Design problems. Although this study does not directly use DFL, it highlights the importance of integrating the prediction and optimization steps when solving large-scale ND problems.

We have covered ND problems and their stochastic and deterministic formulations. We discussed the importance of choosing the correct point prediction when predicting uncertain ND problem parameters. In the next section, we discuss how to train prediction models to make predictions that minimize the cost in the optimization step.

2.2 Decision-Focused Learning

In mathematical optimization problems, the model's parameters are not always known in advance and must be estimated from noisy data. This parameter prediction step is traditionally done using ML (Bishop, 2006) or time series forecasting (De Gooijer and Hyndman, 2006) ahead of the optimization problem. Estimating these parameters separately from the downstream optimization problem, such as with an ML model minimizing a measure of prediction error, can lead to sub-optimal decisions in the optimization problem. The development of integrated prediction and optimization techniques aims to resolve this mismatch. In this thesis, we call such integrated techniques DFL. We first review traditional prediction methods from ML and introduce a common notation for learning models in our project. We then cover DFL methods using a unified notation.

We follow textbook Bishop (2006), which uses ML to perform supervised regression tasks. This is the the prediction step in our project. Regression ML models make predictions about the value of a continuous random variable $\boldsymbol{d}$, given input features $\boldsymbol{\phi}$ that are assumed to occur jointly with $\boldsymbol{d}$ under an unknown distribution $(\boldsymbol{\phi}, \boldsymbol{d}) \sim \mathcal{D}$.

Definition 2.2 (Supervised ML model)**.** *A supervised ML model f_θ defined by model weights*

θ, takes input features $\boldsymbol{\phi}$ and outputs a prediction $\hat{\boldsymbol{d}}$ for the target variable $\boldsymbol{d}$:

$$\hat{\boldsymbol{d}} = f_\theta(\boldsymbol{\phi}).$$

Supervised ML models undergo a training phase to produce accurate predictions. Prediction accuracy is measured using a loss function $\mathcal{L}$ that compares the prediction $\hat{\boldsymbol{d}}$ to the actual value $\boldsymbol{d}$. The most common loss functions are Mean Squared Error (MSE) and Mean Absolute Error (MAE).

Definition 2.3 (Mean Squared Error loss)**.** $\mathcal{L}_{MSE}(\hat{\boldsymbol{d}}, \boldsymbol{d}) = \|\hat{\boldsymbol{d}} - \boldsymbol{d}\|_2$.

Definition 2.4 (Mean Absolute Error loss)**.** $\mathcal{L}_{MAE}(\hat{\boldsymbol{d}}, \boldsymbol{d}) = \|\hat{\boldsymbol{d}} - \boldsymbol{d}\|_1$.

Given a training dataset of N independent observations $\mathcal{D}_{\text{train}} = \{(\boldsymbol{\phi}_i, \boldsymbol{d}_i)\}_{i=1}^N$, the model updates its weights θ to minimize prediction the prediction loss over the training dataset. If the model is convex and differentiable with respect to the model weights, this can be done iteratively using Gradient Descent, detailed in Algorithm 2.2.

Algorithm 2.2: Gradient Descent algorithm

Input : loss function $\mathcal{L}$
learning rate $\gamma \geq 0$
training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{\phi}_i, \boldsymbol{d}_i)\}_{i=1}^N$
Output : trained model f_θ

Initialize θ_1;
for $i = 1, \ldots, N$ **do**
  $\hat{\boldsymbol{d}}_i \leftarrow f_{\theta_i}(\boldsymbol{\phi}_i)$;
  $\theta_{i+1} \leftarrow \theta_i - \gamma \nabla \mathcal{L}(\hat{\boldsymbol{d}}_i, \boldsymbol{d}_i)$;
end
return $f_{\theta_{N+1}}$;

The key idea behind DFL is to evaluate ML models not on prediction accuracy but on the cost of the resulting downstream optimization problems. Model training remains the same as in regular ML, except the loss function compares the cost of the network designed for the predicted demand and the network designed for the actual demand. This downstream loss function is called regret.

Definition 2.5 (Regret-based loss (Sadana et al., 2023))**.** *Let $\boldsymbol{d}$ be the value of the uncertain parameters of an optimization problem. Let $\hat{\boldsymbol{d}}$ be a point prediction for $\boldsymbol{d}$. We suppose that $z^*(\hat{\boldsymbol{d}})$ is the optimal decision when the parameter has value $\hat{\boldsymbol{d}}$. Let $C(z, \boldsymbol{d})$ be the cost of making decision z when the actual value of the uncertain parameters is $\boldsymbol{d}$. We can write the general*

definition for the regret-based loss as:

$$\mathcal{L}_{regret}(\hat{\boldsymbol{d}}, \boldsymbol{d}) = C(z^*(\hat{\boldsymbol{d}}), \boldsymbol{d}) - C(z^*(\boldsymbol{d}), \boldsymbol{d}).$$

This definition, from Sadana et al. (2023), is general and applies to any optimization problem with uncertain parameters. However, the cost of a decision given a realization of the uncertain parameters, which we write $C(z, \boldsymbol{d})$, differs based on the type of optimization problem. Most of the literature on DFL considers the case of Linear Programming (LP) or MILP problems in which only the objective cost function parameters are uncertain. For these types of problems, the feasible region of the optimization problem does not change based on the predicted parameters. A solution feasible under the predicted cost is also feasible under the actual cost. We can thus formulate a simplified regret-based loss:

Definition 2.6 (Regret-based loss for LP with uncertain costs)**.** *Let $c \in \mathbb{R}^m$ be the actual value of the cost vector of an LP or an MILP. Let $\hat{c} \in \mathbb{R}^m$ be a prediction of c and let $x^*(c)$ be the optimal decision vector under cost vector c. The regret-based loss can be written:*

$$\mathcal{L}_{regret}(\hat{c}, c) = c^\top x^*(\hat{c}) - c^\top x^*(c).$$

Two surveys cover the current advancements of DFL, the first of which is Kotary et al. (2021). It covers ways of integrating ML techniques with Constrainted Optimization (CO) methods. It separates this research area into two categories: ML-augmented CO and End-to-End CO Learning. The former category refers to using ML techniques to improve the performance of existing optimization solvers. The latter category refers to integrating CO into broader ML architectures, either by predicting CO solutions using ML, or by integrating a combinatorial solver as a layer of the ML architecture. Our work focuses on the latter category, specifically on integrating a combinatorial solver as the last layer of a ML pipeline, which the survey refers to as *Predict-and-Optimize*.

Sadana et al. (2023) is a more recent survey which looks at problems of making decisions under uncertainty where contextual information is available. It formulates such problems as CSO problems, in which the objective is to make decisions $z \in \mathcal{Z}$ that minimize some cost where some parameters of the problem $\boldsymbol{d}$ are uncertain, given contextual information $\boldsymbol{\phi}$. This contextual information is assumed to be jointly distributed with the uncertain parameters according to some unknown distribution $(\phi, \boldsymbol{d}) \sim \mathcal{D}$. We are interested in minimizing the expectation of the decision cost with respect to the conditional distribution $\xi|\boldsymbol{d}$, This can be written as:

$$z^*(\boldsymbol{\phi}) \in \arg\min_{z \in \mathcal{Z}} \mathbb{E}_{\boldsymbol{d}|\boldsymbol{\phi}}[C(z, \boldsymbol{d})|\boldsymbol{\phi}]. \tag{2.3}$$

The survey introduces the idea of ILO, which tries to solve the above CSO problem with a

sequential prediction and optimization pipeline, illustrated in Figure 2.1. The prediction model f_θ, defined by weights θ, makes a point prediction $f_\theta(\boldsymbol{\phi})$ for the demand, given the context $\boldsymbol{\phi}$. The optimization model takes the predicted demands and outputs the optimal decision that fulfills that demand. This solution can be evaluated against the task loss.

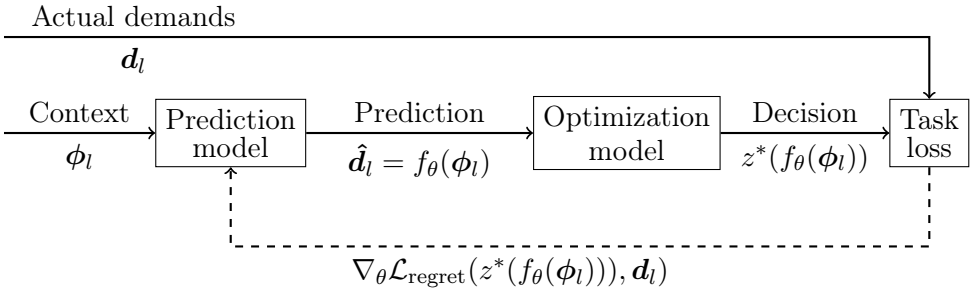


Figure 2.1: DFL pipeline using regret-based loss training (adapted from Sadana et al. (2023))

The ILO pipeline minimizes the empirical risk over an training set using a regret-based loss function. Given a class of prediction models f_θ defined by weights θ, and a training dataset of context-demand-optimal decisions tuples $\mathcal{D}_{\text{train}} = \{(\boldsymbol{\phi}_l, \boldsymbol{d}_l, z^*(\boldsymbol{d}_l))\}_{l=1}^L$, we can write the Empirical Risk Minimization (ERM) task as follows:

Definition 2.7 (Empirical Risk Minimization)**.**

$$\begin{aligned}\theta^* &\in \arg\min_\theta \frac{1}{L}\sum_{l=1}^{L} \mathcal{L}_{regret}(f_\theta(\boldsymbol{\phi}_l), \boldsymbol{d}_l) \\ &= \arg\min_\theta \frac{1}{L}\sum_{l=1}^{L} \Big[C(z^*(f_\theta(\boldsymbol{\phi}_l)), \boldsymbol{d}_l) - C(z^*(\boldsymbol{d}_l), \boldsymbol{d}_l)\Big].\end{aligned}$$

This corresponds to the DFL approaches described in the Kotary et al. (2023) survey. Regret-based losses are often non-convex and nondifferentiable with respect to the prediction parameters. Therefore, additional techniques are needed to adequately train ILO models. These techniques often use optimality conditions such as the Karush–Kuhn–Tucker (KKT) conditions. Formulated by Karush (1939) and Kuhn and Tucker (1951), the KKT conditions describe the optimality condition of an optimization program, such as a Quadratic Program (QP). The survey distinguishes three main approaches to training integrated models:

1. Training using implicit differentiation: using the implicit function theorem on the optimality conditions, such as the KKT conditions, to find a gradient of the optimization problem.

2. Training using a surrogate differentiable optimizer: apply stochastic perturbations to

the original objective function to make the optimization differentiable and its gradient informative.

3. Training using a surrogate differentiable loss function: find and minimize a surrogate loss function for the regret-based loss that has the desired properties of convexity and differentiability.

Almost all of the methods investigated by the survey are limited to uncertainty in the objective function and not the constraints. Furthermore, most of the DFL methods also focus on LPs or QPs, with relatively few methods applicable to MILPs.

Amos and Kolter (2017) are among the first to consider DFL problems. They introduce *OptNet*, a framework in which QPs act as layers in a wider neural network. The value of the objective and constraint parameters depends on the previous layer in the network. In order to train such a neural network with stochastic gradient descent, they must find the gradient of the optimization problem with respect to the model weights. They find the gradient by implicitly differentiating the KKT conditions of the QP.

Donti et al. (2017) then extend the idea of implicit differentiation of the KKT conditions to CSO and Convex Optimization. They are the first to formulate CSO problems in which there is uncertainty in both the objective and the constraints, and where contextual information is used to improve prediction performance. The true conditional distribution $\mathbb{P}(\boldsymbol{d}|\boldsymbol{\phi})$ of the optimization parameters $\boldsymbol{d}$ under contextual information $\boldsymbol{\phi}$ is approximated using a distribution $\hat{\mathbb{P}}(\boldsymbol{d}|\boldsymbol{\phi};\theta)$, defined by parameters θ. They introduce a *task loss* $\mathcal{L}(\theta)$ which measures the error of the decisions $z^*(\boldsymbol{\phi};\theta)$ made under $\hat{\mathbb{P}}$, compared to decisions made under the true distribution $\mathbb{P}$. This task loss is written:

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{\phi},\boldsymbol{d}\sim\mathcal{D}}[f(\boldsymbol{\phi},\boldsymbol{d},z^*(\boldsymbol{\phi};\theta))] + \sum_{i=1}^{N} \boldsymbol{I}\left\{\mathbb{E}_{\boldsymbol{\phi},\boldsymbol{d}\sim\mathcal{D}}[g_i(\boldsymbol{\phi},\boldsymbol{d},z^*(\boldsymbol{\phi};\theta))] \leq 0\right\}, \tag{2.4}$$

where f is the objective function, and g_i represents the constraint $i = 1,\ldots,N$ of the CSO problem. The indicator operator $\boldsymbol{I}\{\cdot\}$ is 0 when a constraint is satisfied and infinite when the constraint is not satisfied. Due to the indicator operators, this loss is not differentiable. Nevertheless, the authors minimize the loss using a constrained form of stochastic gradient descent. When a constraint is violated, the weights are updated according to the gradient of the constraint g_i. Otherwise, the weights are updated following the gradient of the objective f. The authors do not address the validity of this method, and none of the experiments have uncertainty in the constraints. To obtain the gradient of $z^*(\boldsymbol{\phi};\theta)$, the authors use the implicit differentiation of the KKT conditions. In any case, both Amos and Kolter (2017) and Donti et al. (2017) are limited to non-linear problems with continuous variables, since the gradient given by the KKT conditions is zero for LPs.

Subsequent works extend the idea of using implicit differentiation to new domains, such as LPs or MILPs. However, these works only consider uncertainty in the objective parameters. Wilder et al. (2019) consider the case where the objective function parameters of an MILP are predicted by a neural network. In order to have a differentiable optimization layer, the authors use the linear relaxation of the MILP. For the KKT conditions to yield a non-trivial gradient, they add a quadratic regularization term to the objective function. This turns the problem into a convex quadratic program, then which allows meaningful differentiation. Several papers build on this method. Ferber et al. (2020) use the same linear relaxation in the last layer of a the neural network, but generate cutting planes for each training instance to improve the accuracy of the relaxation. Mandi and Guns (2020) replace the quadratic penalty term in the objective with a technique from interior point solvers: a logarithmic barrier term. This provides a more mathematically principled approach to finding the gradient of an LP.

The foundations for training DFL pipelines using differentiable surrogate loss functions were first laid by Elmachtoub and Grigas (2022). They introduce the *Smart Predict-then-Optimize* (SPO) framework, in which a linear prediction model predicts the objective parameters of an LP. In training, the prediction model is tasked with minimizing the SPO loss function, which the authors show is equivalent to minimizing expected regret. However, since the SPO loss is non-convex, they define a convex surrogate called SPO+, which has a subgradient. They show that minimizing this surrogate amounts to minimizing the SPO loss function. Mandi et al. (2020) extend the SPO framework to MILPs by computing the gradient of the LP relaxation during training. They call this approach SPO-Relax. Both of these approaches are applicable to the prediction of the objective parameters of LPs and MILPs respectively, not the constraints.

Pogančić et al. (2019) define the loss function of a combinatorial optimization problem using a different approach to SPO. Indeed, the gradient of a combinatorial optimization problem with respect to the objective parameters is zero almost everywhere, and discontinuous when switching from one solution to another. This is not helpful for iterative minimization algorithms such as gradient descent. To remedy this issue, the authors continuously interpolate the gradient such that the resulting gradient is informative. The trade-off between informativeness of the gradient and "faithfulness to the original function" can be tuned using a hyperparameter.

None of the methods mentioned so far considers the case of predicting parameters outside the objective function. For example, in the MCFND, the demand for a particular commodity appears in the constraints and not in the objective. This case is not addressed by previous implicit differentiation or surrogate-based DFL pipelines. Paulus et al. (2021) introduce *CombOptNet*, a DFL method that integrates an integer programming solver as a layer within a neural network capable of estimating both the objective terms and the constraints. The method is agnostic to the type of solver used. Given a loss that is differentiable with respect to the solutions of the optimization problem, *CombOptNet* provides a way to propagate this loss "through" the optimization layer, thus providing a differentiable gradient surrogate of the loss

with respect to the objective and constraint parameters. For the gradient with respect to the constraint parameters, *CombOptNet* uses a custom differentiable surrogate function called the *mismatch function*. The mismatch function generalizes the concept of active constraints: given the training examples of feasible solutions to the MILP, it suggests updates to the constraint parameters that minimize the distance to the nearest constraint for solutions that are already feasible, and minimizes the distance to violated constraints for solutions that are still infeasible. This DFL method is unique in that it can learn both objective and constraint parameters for MILP problems.

Computational performance of DFL pipelines is a significant issue: many of the methods require solving the optimization problem for each iteration of the training algorithm. This is too computationally expensive for the large instances of ND problems seen in practice. Several papers propose techniques to speed up computation or to avoid computing the solution to the ND problem. Mandi et al. (2020) improve performance of the SPO framework by warm starting the combinatorial optimizer with the solution of the previous solution. Mulamba et al. (2021) introduce two notions that help performance: solution caching and noise constrastive loss. Solution caching improves performance by replacing some optimization calls with lookups to a cache of previously computed solutions. Noise-contrastive estimation tries to build a probabilistic model of problem solution by treating non-optimal solutions as negative examples. This method is only explore in the context of predicting the objective parameters and not the constraints. Nandwani et al. (2022) specifically tackle the computational complexity of CombOptNet, which requires the solver to run at every forward pass. They introduce a solver-free method that considers the opt problem as a set of trainable hyperplanes.

DFL integrates the prediction and optimization steps by using the downstream decision regret to train the predictive model. This ensures that predictions that lead to optimal decisions are favoured, ahead of pure prediction accuracy. When training a DFL pipeline using a gradient descent algorithm, it is necessary to differentiate through the $\arg\min$ of the optimization problem with respect to the prediction model weights. In order to find a gradient, three broad techniques exist in the literature: implicit differentiation, surrogate optimization problems, and surrogate loss functions. Most differentiation techniques in the literature can only predict the parameters in the objective functions and not in the constraints, with the exception of *CombOptNet*. Almost all the the methods suffer from heavy computational cost compared to sequential prediction and optimization, though some modifications to the methods have been proposed in order to improve performance.

We have seen how to train prediction models to predict uncertain model parameters using DFL, i.e. based on downstream optimization performance rather than prediction accuracy. We now turn our attention to IO, a branch of mathematical optimization in which we recover the uncertain model parameters given solutions to the model.

2.3 Inverse Optimization

Given a set of decisions produced by an unknown optimization problem, IO seeks to recover the parameters of the optimization problem that render the solutions optimal. IO assumes the decisions are generated from a hidden *forward optimization problem*. The structure of the forward model, such as the type of optimization problem, the number of constraints and the number of decision variables, is set ahead of time. IO exists for several types of forward models: linear, mixed-integer, graph, quadratic and conic programs can all be recovered using IO. An *inverse model* is another optimization model that recovers the constraint and objective parameters that render the solutions exactly or approximately optimal. In this section, we cover the IO techniques that could be used for DFL problems.

Early research into IO, started by Burton and Toint (1992), concentrated on specific optimization problems, such as the inverse shortest paths problem, and did not account for noise in the solutions. In this research, the authors formulate a model that recovers arc costs in a graph, given information on the shortest paths in such a graph. Since then, research has extended IO techniques to more general types of optimization problems, and can account for noise in the solutions using data-driven IO.

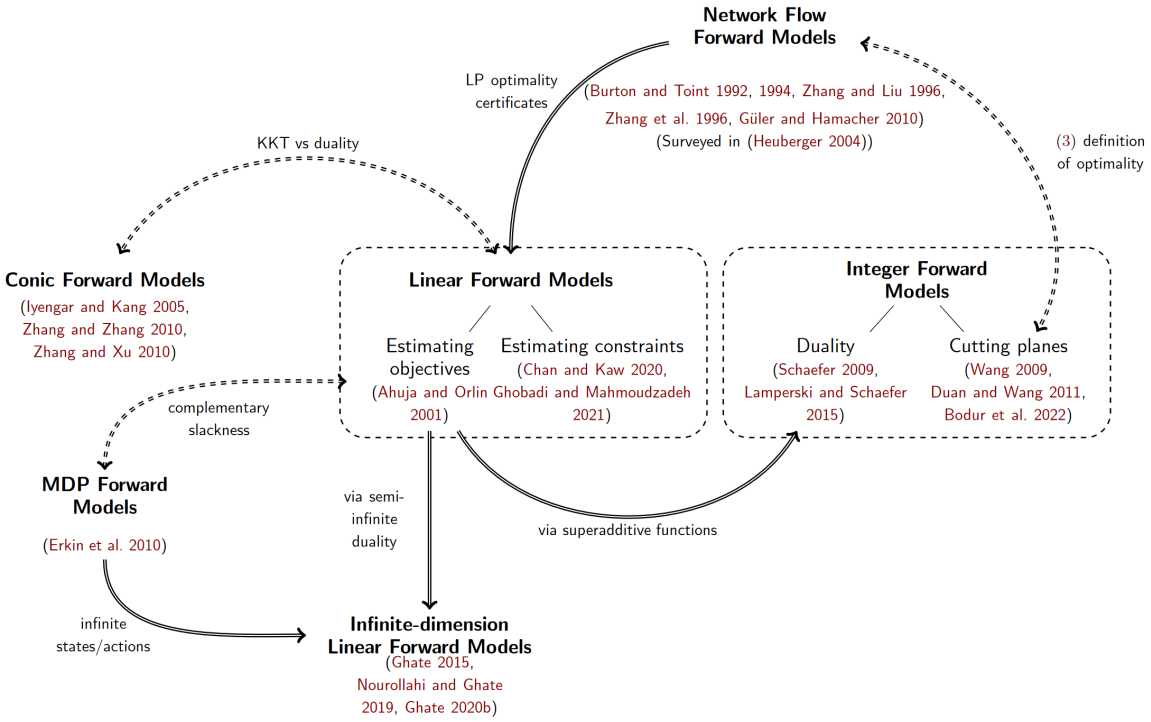


Figure 2.2: Roadmap of Classical IO from Chan et al. (2021). Bold arrows denote extensions and generalizations and dashed arrows denote conceptual similarities.

Chan et al. (2021) survey the current state of the art in IO. They provide a taxonomy of IO

problems by splitting them along three axes: forward model structure, parameter type, and model-data fit. The structure of the forward model (e.g. linear, mixed-integer, sequential, or conic) determines which kind of IO techniques can be used, and which parameters can be recovered. Parameter type refers to which problem parameters need to be recovered. This can be whether the objective parameters, the constraint parameters, or both are unknown. The authors of the survey split the field of IO into classical and data-driven IO, and call this split model-data fit. In the classical setting, every decision in the set of solutions is assumed to be optimal. In the data-driven setting, decisions contain some amount of noise and may not be optimal or even feasible. The taxonomy of classical IO is illustrated in Figure 2.2, taken from the survey. Data-driven IO is orthogonal to classical IO: the same techniques can be used, but instead of enforcing solution feasibility in the constraints of the inverse model, deviation from feasibility is penalized using an additional loss function in the objective.

Most IO literature focuses on the recovery of parameters in the objective function. However, in the case of ND, the unknown demand parameters appear in the constraints. Bodur et al. (2022) proposes an IO method for mixed-integer LPs, but is limited to the recovery of objective parameters. Methods for recovering the constraint parameters of integer or mixed-integer forward models do not exist as of yet. Chan and Kaw (2020) and Ghobadi and Mahmoudzadeh (2021) focus on recovering the constraint parameters of a LP. Both methods assume a linear forward problem with the following structure:

$$\text{(Forward Problem)} \qquad \begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \\ & \boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{2.5}$$

Chan and Kaw (2020) assume the $\boldsymbol{c}$ and $\boldsymbol{b}$ vectors are known, and show a method for recovering the parameters in $\boldsymbol{A}$. Ghobadi and Mahmoudzadeh (2021) consider the more general problem of recovering all the constraint parameters $\boldsymbol{A}$ and $\boldsymbol{b}$. In both papers, the resulting IO models contain bilinear terms in the constraints and must use observations about the forward problem structure in order to reduce the inverse model to a computationally tractable version. For example, Ghobadi and Mahmoudzadeh (2021) observe that, given the optimal solution $\boldsymbol{x}^*$, $\boldsymbol{c}^\top \boldsymbol{x} \geq \boldsymbol{c}^\top \boldsymbol{x}^*$ is an implicit constraint of the forward problem. This observation allows them to formulate the following inverse model without the implicit constraints:

(eMIO)

$$\underset{\boldsymbol{A},\boldsymbol{b}}{\text{minimize}} \quad \mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D}) \tag{2.6a}$$

$$\text{subject to} \quad \boldsymbol{a}_k^\top \bar{\boldsymbol{x}}^l \geq \boldsymbol{b}, \quad \forall\, k,l, \tag{2.6b}$$

$$\|\boldsymbol{a}_k\| = 1, \quad \forall\, k, \tag{2.6c}$$

$$\boldsymbol{A} \in \mathcal{A}_{\text{valid}}, \tag{2.6d}$$

$$\boldsymbol{b} \in \mathcal{B}_{\text{valid}} \tag{2.6e}$$

This models defines parameters $\boldsymbol{A}$ and $\boldsymbol{b}$, from their respective domains $\mathcal{A}_{\text{valid}}$ and $\mathcal{B}_{\text{valid}}$, which ensure all solutions $\{\bar{\boldsymbol{x}}\}_l$ are feasible. The choice of parameters $\boldsymbol{A}$ and $\boldsymbol{b}$ minimizes the cost function on the constraint parameters $\mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D})$. Various cost functions are possible to achieve desired properties of the recovered feasible region.

All of the inverse models discussed above have a key limitation for ILO pipelines: they do not yield a trained prediction model which can generalize to new instances of the demand prediction problem. For every new instance of the MCFND, the IO model would have to be re-run, using the new optimal solution to recover the demands. This cannot work in practical cases, since the actual demand and thus the actual optimal network is not known at the time of planning. This raises the following question: is there a way to train a demand prediction model using techniques from IO?

A few months after the start of this thesis, Sun et al. (2023) is published. They are the first to perform DFL with IO for linear prediction models and linear forward optimization models. They consider a CSO problem in which the objective parameters of the optimization are predicted using a linear regression model on the contextual information. The authors embed the prediction model inside of the forward optimization model, replacing the objective parameters with their respective linear regression prediction, i.e. $\boldsymbol{c}$ is replaced with $\hat{\boldsymbol{c}} := \boldsymbol{\Theta}\boldsymbol{\phi}$. The IO model recovers the linear regression weights $\boldsymbol{\Theta}$ that minimize task loss, given a training dataset of contextual features and corresponding optimal solutions $(\boldsymbol{\phi}, \boldsymbol{x}^*)$. To achieve this, the authors first define the optimal basis of an LP:

Definition 2.8 (Optimal basis, Sun et al. (2023), p. 4)**.** *Let $\boldsymbol{x}^* := (z_1^*, \ldots, z_n^*)$ be the optimal solution to the linear problem* $\text{LP}(c, A, b)$*, with a cost vector $\boldsymbol{c} \in \mathbb{R}^n$, and m constraints defined by $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. The optimal basis $\mathcal{B}^*$ and its complement $\mathcal{N}^*$ is defined as follows:*

$$\mathcal{B}^* := \{i : x_i^* > 0\}, \quad \mathcal{N}^* := \{i : x_i^* \leq 0\}$$

For a set $\mathcal{B} \subset [n]$, $A_{\mathcal{B}}$ denotes the submatrix of A with columns indices corresponding to $\mathcal{B}$, and $c_{\mathcal{B}}$ denotes the subvector with corresponding dimensions.

The authors use a property of the optimal basis of an LP which states, with minimal assumptions

on the structure of the LP, that:

Lemma 2.1 (Lemma 1 of Sun et al. (2023))**.** *Let $\mathcal{B} \subset [n]$ be a feasible basis of* $\mathrm{LP}(c, A, b)$*, i.e. $A_{\mathcal{B}}^{-1} b \geq 0$ and let $\mathcal{N} = [n] \setminus \mathcal{B}$ be its complement. Then, if LP is non-degenerate, the following relation holds:*

$$c_{\mathcal{N}}^{\top} - c_{\mathcal{B}}^{\top} A_{\mathcal{B}}^{-1} A_{\mathcal{N}} \geq 0 \quad \Leftrightarrow \quad \mathcal{B} = \mathcal{B}^*$$

Using this property, the authors formulate their IO model for $\mathrm{LP}(c, A, b)$, given training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{\phi}_l, x_l^*, A_l, b_l, \mathcal{B}_l^*, \mathcal{N}_l^*)\}_{l=1}^{L}$:

(Maximum-Optimality-Margin)

$$\hat{\Theta} := \arg\min_{\Theta} \quad \frac{\lambda}{2}\|\Theta\|_2^2 + \frac{1}{L}\sum_{l=1}^{L}\|s_l\|_1 \tag{2.7a}$$

subject to

$$\hat{c}_l = \Theta \boldsymbol{\phi}_l, \qquad \forall l = 1, \ldots, L, \tag{2.7b}$$

$$\hat{c}_{l,\mathcal{N}_l^*}^{\top} - \hat{c}_{l,\mathcal{B}_l^*}^{\top} A_{l,\mathcal{B}_l^*}^{-1} A_{l,\mathcal{B}_l^*} \geq \mathbb{1}_{|\mathcal{N}_l^*|} - s_l, \quad \forall l = 1, \ldots, L \tag{2.7c}$$

Constraints (2.7b) encode the linear prediction of the cost vector $\hat{c}_l$ into the model, using weights Θ and contextual information $\boldsymbol{\phi}_l$. The left hand side of constraint (2.7c) represents the optimality condition from Lemma 2.1, and the right hand size captures the violation, or *slackness*, of the optimality constraint for that prediction, encoded in the variable s_l. The objective (2.7a) regularizes the weights Θ and minimizes the mean absolute deviation from the optimality conditions.

This method yields a trained linear prediction model $f_{\boldsymbol{\Theta}}$ with weights $\hat{\Theta}$ which predicts costs $\boldsymbol{c}$ for contextual features $\boldsymbol{\phi}$ without knowing their corresponding optimal optimization solution $\boldsymbol{x}^*$. This model produces predictions $\hat{c}$ that minimizes deviation from optimality in the downstream optimization problem, which means this model minimizes task loss. However, this is limited to predicting the objective parameters of linear optimization models and cannot predict constraint parameters of mixed-integer models.

IO offers advanced techniques for recovering parameters of optimization problems given a set of solutions. It supports multiple types of optimization models, can recover parameters in the objective and in the constraints, and can handle noise and uncertainty in the data. Generally, it lacks the ability to generalize to new instances of the same problem without re-solving the IO. However, new research offers a promising direction in which a linear prediction model is trained to minimize task loss using IO.

2.4 Discussion

ND is a well researched subject in operations research. Many works focus on finding more efficient ways to solve large instances of the MCFND. The complexity and combinatorial nature of ND problems means that stochastic formulations are often not computationally tractable. Thus, most of the research concentrates on the deterministic version of ND, with the parameter prediction steps left separate.

The idea of DFL, integrating the prediction and optimization steps in a data-decisions pipeline, is relatively recent. Research in this field originated in the ML and operations research community. The key technique involves training a prediction model using gradient descent on the downstream task loss, or *regret*. This implies differentiating through the arg min of the optimization problem, which can be done using implicit differentiation, differentiable surrogate loss functions, or differentiable surrogate models. Current works focus on predicting cost function coefficients, and are mostly restricted to problems with continuous variables, with the exception of *CombOptNet* from Paulus et al. (2021).

IO is a more developed field than DFL. Given solutions to a forward optimization model with unknown parameters, an IO model recovers parameters that make the decisions optimal. Several techniques to recover constraint parameters of LPs exist, though none for MILPs yet, and IO can handle uncertainty in the solution dataset using data-driven IO. However, IO does not yield a prediction model that can be used on new instances of the problem, which limits its use in the ND pipeline. Recent research by Sun et al. (2023) shows how to use IO to train a linear prediction model that predicts objective parameters that minimize task loss.

In this chapter, we have reviewed the three related areas of ND, DFL, and IO. Our setting is often formulated as the stochastic MCFND problem. There are several studies on stochastic ND, but real-world applications use the deterministic formulation of the MCFND. Solving an ND problem as a stochastic program turns out to be too computationally expensive, despite the improvements in decision quality. Therefore, real-world applications use point predictions for the parameters of the optimization model. This led us to investigate DFL and IO to find how to improve the quality of the point predictions. DFL could be used with stochastic programming, e.g. decision-focused density estimation is possible for CSO problems, as shown in Sadana et al. (2023), but research on this topic is still limited. For this reason, and due to the computational limitations of stochastic ND, we restrict this work to DFL and IO making point predictions. Most of the research in DFL and IO to date focuses on linear optimization programs, with continuous variables and uncertainty only in the objective. However, our use case involves complex MILPs with uncertain parameters in the constraints. Nevertheless, we have identified key ideas that could help our task of integrating the prediction and optimization steps of a ND problem with uncertain demands. All of them rely on a good definition of regret, the cost of the downstream optimization problem. Regret is well-defined when only the objective parameters are uncertain, and many systems offers ways to minimize regret in this

case. Some systems offer interesting ways to recover uncertain constraint parameters, such as *CombOptNet*, but use simpler, differentiable, loss functions and not regret. *CombOptNet* also suffers from computational performance issues that may be addressed by other techniques from the literature. By making connections to the separate field of IO, we could improve DFL methods. Recent work by Sun et al. (2023) on using IO to train a prediction model that minimizes regret for uncertain costs could be combined with constraint recovery techniques from Chan and Kaw (2020) or Ghobadi and Mahmoudzadeh (2021) to create an alternative method of training DFL pipelines.

Chapter 3

Methodology

This chapter presents the methodology we use to investigate DFL applied to ND problems with uncertain commodity demands. Since this is an exploratory study, we first describe the research process we followed, and we explain how we arrived at the methodology that we present in this chapter. We wanted to apply existing DFL methods to an ND problem. After reviewing the literature on ND, DFL, and IO literature, we found that few existing methods handle combinatorial problems with uncertainty in the constraints, but we were confident that we could combine and adapt techniques from IO and DFL to build a prediction model that minimizes regret. At the time, the paper by Sun et al. (2023) had not been published yet, so we set out to build an IO model that trains a linear prediction model. After reviewing the results of our IO model, we noticed that the predictions were almost identical to our baseline linear regression. Digging deeper, we discovered that the definition of regret used in the DFL literature does not apply to problems with uncertain constraints. Uncertainty in the constraints changes the feasible region of the problem, making it difficult to compare solutions with different demands. This meant that we could not use or adapt existing techniques, and finding a completely new DFL approach was beyond the scope of this thesis. However, we did explore ideas for improving the performance of a prediction model by viewing DFL as a weighting of the training examples in the loss function.

We now delve into the heart of our topic and outline the structure of this chapter. In Section 3.1, we formulate the ND problem as a stochastic optimization problem. We also introduce Prediction-Optimization pipelines that predict solutions to the ND problem from contextual information. In Section 3.2, we investigate how to evaluate the performance of a Prediction-Optimization pipeline for ND problems when the demand is uncertain. We then highlight the problem with the definition of regret in Section 3.3. In Section 3.4, we formulate *IO-Constraint*, an IO model that recovers the weights of a linear prediction model. Finally, in Section 3.5, we explore reweighting the training examples in the loss function to improve

prediction performance.

3.1 Network Design Formulation

We formulate our problem as a stochastic Multi-Commodity Fixed-charge ND problem with two stages. We use the arc-based MCFND formulation from Crainic et al. (2021), to which we add stochastic demands using the task-based loss formulation from Donti et al. (2017). The MCFND problem is defined on a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ with commodities $k \in \mathcal{K}$ and associated origin and destination nodes $(\text{orig}(k), \text{dest}(k))$. Arc capacities u_{ij}, design costs f_{ij}, and flow costs c_{ij}^k are fixed and known in advance. Demand for each commodity is uncertain, and we assume that the associated contextual information occurs jointly with the commodity demands following an unknown distribution. We let $\boldsymbol{d} = [d^1 \cdots d^{|\mathcal{K}|}]^\top \in \mathbb{R}^{|\mathcal{K}|}$ be a realization of the vector of commodity demands, and $\boldsymbol{\phi} \in \mathbb{R}^m$ be a realization of the contextual information. We can split the commodity demands $\boldsymbol{d}$ into a vector of demands for each commodity $k \in \mathcal{K}$ and node $i \in \mathcal{N}$ pair as follows:

$$d_i^k = \begin{cases} d^k & i = \text{orig}(k) \\ -d^k & i = \text{dest}(k) \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Our problem is split into a first and a second stage. The first stage (S-MCFND-First) designs the network: it defines a vector of design variables $\boldsymbol{y} \in \{0,1\}^{|\mathcal{A}|}$ that minimizes the design costs and the expected multicommodity flow costs $\mathbb{E}_{\boldsymbol{d}|\boldsymbol{\phi}}[Q(\boldsymbol{y}, \boldsymbol{d})|\boldsymbol{\phi}]$, conditional on the observation of contextual information $\boldsymbol{\phi}$. It yields $y^*(\boldsymbol{\phi})$, the optimal network given the contextual information. The second stage (S-MCFND-Second) assumes that the network $\boldsymbol{y}$ is already designed, and that the realization of demand $\boldsymbol{d}$ is fixed. Given the network y and the values of demand vector $\boldsymbol{d}$, the lower stage minimizes the total multicommodity flow cost, denoted $Q(\boldsymbol{y}, \boldsymbol{d})$.

(S-MCFND-First)

$$\boldsymbol{y}^*(\boldsymbol{\phi}) = \underset{y_{ij}}{\arg\min} \quad \sum_{(i,j)\in\mathcal{A}} f_{ij} y_{ij} + \mathbb{E}_{\boldsymbol{d}|\boldsymbol{\phi}}[Q(\boldsymbol{y}, \boldsymbol{d})|\boldsymbol{\phi}] \tag{3.2a}$$

$$\text{subject to} \quad y_{ij} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{A} \tag{3.2b}$$

(S-MCFND-Second)

$$Q(\boldsymbol{y}, \boldsymbol{d}) = \text{minimize} \quad \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k \cdot x_{ij}^k \tag{3.3a}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{N}^+(i)} x_{ij}^k - \sum_{j \in \mathcal{N}^-(i)} x_{ij}^k = d_i^k, \qquad \forall\, i \in \mathcal{N}, k \in \mathcal{K}, \tag{3.3b}$$

$$\sum_{k \in \mathcal{K}} x_{ij}^k \leq u_{ij} y_{ij}, \quad \forall\, (i,j) \in \mathcal{A}, k \in \mathcal{K}, \tag{3.3c}$$

$$x_{ij}^k \geq 0 \qquad \forall\, (i,j) \in \mathcal{A}, k \in \mathcal{K} \tag{3.3d}$$

This stochastic formulation is not computationally tractable on a large scale. We must rewrite the stochastic formulation into a two-stage Prediction-Optimization pipeline: the first stage predicts the demands from the contextual information, and the second stage solves the deterministic optimization problem, as illustrated in Figure 2.1. The prediction model f_θ, defined by weights θ, uses contextual information $\boldsymbol{\phi}$ and outputs a point prediction for the demand $\hat{\boldsymbol{d}}$. The optimization stage takes this prediction and solves the corresponding deterministic MCFND problem from (2.2), outputting the optimal network that fulfills the predicted demand. We write the optimal network as follows:

Definition 3.1. *The optimal network for a realization of demand $\boldsymbol{d}$ can be written as:*

$$z^*(\boldsymbol{d}) = (x^*(\boldsymbol{d}), y^*(\boldsymbol{d})) := \text{MCFND}(\boldsymbol{d}),$$

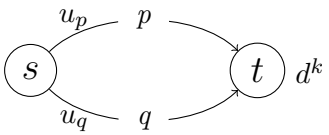
where $x^*(\boldsymbol{d})$ and $y^*(\boldsymbol{d})$ are the optimal flow and design variables respectively.

Example 3.1 shows the deterministic optimization pipeline on a small network with two arcs and only a single commodity. We will use this minimal example to illustrate further concepts and evaluate models later in this thesis.

Example 3.1. *We define the MCFND problem on a small graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. Let $\mathcal{N} = \{s, t\}$ and let $\mathcal{A} = \{p, q\}$ where both arcs p and q link node s to node t. We suppose we are transporting a single commodity k, i.e. $\mathcal{K} = \{k\}$.*

Let u_p and u_q be the capacity of arcs p and q with respective design costs f_p and f_q and respective flow costs c_p^k and c_q^k.

Let d^k be the uncertain demand for commodity k, and let $d_s^k = -d^k$ and $d_t^k = d^k$ be the demand for commodity k at nodes s and t. Since $d_t^k = -d_s^k$, we can simplify the model by only considering flow fulfillment constraints on node t:



$$
\begin{aligned}
\text{MCFND}(d^k) = \arg\min \quad & f_p y_p + f_q y_q + c_p^k x_p^k + c_p^k x_p^k \\
\text{s.t.} \quad & x_p^k + x_q^k = d^k, \\
& x_p \le u_p y_p, \\
& x_q \le u_q y_q, \\
& x_p, x_q \ge 0, \\
& y_p, y_q \in \{0, 1\}.
\end{aligned}
$$

The goal is for the Prediction-Optimization pipeline to make the best possible decisions given the contextual information. We therefore train the prediction model f_θ so that its demand predictions lead to good downstream decisions. The actual training method varies between prediction models, but all are based on the ERM principle. According to the ERM principle, the training procedure selects the model weights θ that yield the lowest error over a dataset of training data points $\mathcal{D}_{\text{train}} = \{(\boldsymbol{\phi}_l, \boldsymbol{d}_l)\}_{l=1}^{L}$. The error of a prediction model is measured by a loss function $\mathcal{L}$. Traditional prediction models minimize a loss function that evaluates the accuracy of the prediction, such as Mean Squared Error. We would like to minimize regret $\mathcal{L}_{\text{regret}}$ from Definition 2.5, which evaluates the cost of the decisions that result from the prediction. Since our problem contains uncertain parameters in the constraints, we cannot use the simple regret defined in Definition 2.6 and must define what the cost of a solution under the actual demand $C(z, \boldsymbol{d})$. We expand on this point in Section 3.3. Each model has a different training procedure, which we explain in Section 3.4.

3.2 Evaluation of Prediction-Optimization Pipeline Performance

Evaluating the performance of the pipeline is not as straightforward as comparing decision costs, since the feasible region can change based on the predicted demand. Two different demand predictions in the prediction stage can result in two different networks being designed in the optimization stage. A network designed for one demand prediction might not be feasible for another value of demand, and neither network is guaranteed to fulfill the actual demand. We must design a procedure that can meaningfully compare pipeline performance.

Algorithm 3.3 allows us to compare the downstream optimization cost of a Prediction-Optimization pipeline on a new data point $(\boldsymbol{\phi}_{\text{new}}, \boldsymbol{d}_{\text{new}})$. The evaluation algorithm proceeds in two stages. First, the pipeline predicts demand $\hat{\boldsymbol{d}}$ and finds an optimal network $x^*(\hat{\boldsymbol{d}}), y^*(\hat{\boldsymbol{d}})$ that fulfills the predicted demand. Second, we solve the recourse problem MCFND-Flow, a problem defined in (3.7) and that is similar to the second stage of the stochastic ND problem in (3.3). The design variables of MCFND-Flow are fixed to $y_{ij} = [y^*(\hat{\boldsymbol{d}})]_{ij}$, and the flow variables x_{ij} fulfill the actual demand $\boldsymbol{d}_{\text{new}}$. This is analogous to the two-stage stochastic formulation in (3.2) and (3.3): the network is designed in the first stage when demand is still uncertain; after the actual demand is revealed, recourse action is taken in the second stage so that the flow

Algorithm 3.3: Evaluation of a ND Prediction-Optimization pipeline

Input: Contextual information $\boldsymbol{\phi}_{\text{new}}$
Actual demand $\boldsymbol{d}_{\text{new}}$
Trained prediction model f_θ

Output:

First stage: run Prediction-Optimization pipeline

Compute demand prediction $\hat{\boldsymbol{d}} \leftarrow f_\theta(\boldsymbol{\phi}_{\text{new}})$

Solve network for predicted demand $x^*(\hat{\boldsymbol{d}}), y^*(\hat{\boldsymbol{d}}) \leftarrow \text{MCFND}(\hat{\boldsymbol{d}})$

Second stage: evaluate design on flow problem

Solve $C = \text{MCFND-Flow}(\boldsymbol{d}_{\text{new}}, y^*(\hat{\boldsymbol{d}}))$ and get optimum of decision variables x^*_{Flow}

Count number of recourse arcs used N in solution x^*_{Flow}

Return metrics N and C

variables on fixed arcs fulfill the actual demand.

The recourse problem MCFND-Flow in (3.7) evaluates the quality of a network $y^*(\hat{\boldsymbol{d}})$ designed by a Prediction-Optimization pipeline against the actual demand $\boldsymbol{d}$. The formulation uses the same constraints as the standard MCFND, but design variables are fixed in constraints (3.7e). Furthermore, if the demand prediction is too low compared to the actual demand, the arcs selected in $y^*(\boldsymbol{d})$ might not have enough capacity to transport all of the actual demand, making the network infeasible. We avoid this by adding recourse arcs $(\text{orig}(k), \text{dest}(k))$ from the origin to the destination of each commodity k. The set of arcs in MCFND-Flow becomes:

$$\begin{aligned} \mathcal{A}_{\text{rec}} &= \{(\text{orig}(k), \text{dest}(k)) : k \in \mathcal{K}\} \\ \mathcal{A}_{\text{flow}} &= \mathcal{A} \cup \mathcal{A}_{\text{rec}}. \end{aligned} \tag{3.4}$$

The set of outgoing and incoming neighbours is thus written as:

$$\begin{aligned} \mathcal{N}_i^+ &= \{j \in \mathcal{N} : (i,j) \in \mathcal{A}_{\text{flow}}\} \\ \mathcal{N}_i^- &= \{j \in \mathcal{N} : (j,i) \in \mathcal{A}_{\text{flow}}\}. \end{aligned} \tag{3.5}$$

We write the recourse arcs flow variables x^k_{rec}. Since we want the recourse arcs to carry the remaining demand not carried by the other arcs, we do not enforce capacity constraints (3.7c). We set the cost of flow over the recourse arcs to a value larger than all other flow costs, so that the recourse arcs are only used for demand which cannot be fulfilled using the original arcs:

$$c_{\text{rec}} \gg \max\{c^k_{ij}\}. \tag{3.6}$$

Despite the design variables being constant, we keep the design cost calculation inside the objective (3.7a). The objective thus equals the total cost of the Prediction-Optimization pipeline solution, including design and flow costs. We can define MCFND-Flow as follows:

(MCFND-Flow)

$$\begin{aligned}
\underset{x_{ij}^k,\, x_{\text{rec}}^k}{\text{minimize}} \quad & \sum_{(i,j)\in\mathcal{A}} f_{ij}y_{ij} + \sum_{(i,j)\in\mathcal{A}}\sum_{k\in\mathcal{K}} c_{ij}^k x_{ij}^k + c_{\text{rec}} \cdot \sum_{k\in\mathcal{K}} x_{\text{rec}}^k && && \text{(3.7a)}\\
\text{subject to} \quad & \sum_{j\in\mathcal{N}_i^+} x_{ij}^k - \sum_{j\in\mathcal{N}_i^-} x_{ji}^k = d_i^k, && \forall i\in\mathcal{N}, k\in\mathcal{K}, && \text{(3.7b)}\\
& \sum_{k\in\mathcal{K}} x_{ij}^k \le u_{ij}y_{ij}, && \forall (i,j)\in\mathcal{A}, && \text{(3.7c)}\\
& x_{ij}^k \ge 0, && \forall (i,j)\in\mathcal{A}_{\text{flow}}, k\in\mathcal{K}, && \text{(3.7d)}\\
& y_{ij} = [y^*(\hat{\boldsymbol{d}})]_{ij}, && \forall (i,j)\in\mathcal{A}. && \text{(3.7e)}
\end{aligned}$$

We use two metrics to evaluate Prediction-Optimization pipelines: N, the number of recourse arcs used, and C, the total pipeline cost on the evaluation procedure. We compute N by counting the number of recourse arcs with non-zero flow:

$$N = \left|\{x_{\text{rec}}^k : x_{\text{rec}}^k \ge 0, k\in\mathcal{K}\}\right|. \tag{3.8}$$

Only solutions where no recourse path is used, i.e. where $N = 0$, are feasible in the original MCFND and can actual demand. The total pipeline cost C is equal to the value of objective (3.7a). It includes the design cost of the network from the Prediction-Optimization pipeline, the flow cost to fulfill the actual demand, and the recourse flow cost of unfulfilled demand.

3.3 Regret-Related Issues

A loss function gives a measure of the error between a given observed value of an uncertain parameter and the predicted value. Consider an optimization problem with uncertain parameters, where ξ is the observed value of the uncertain parameters and $\hat{\xi}$ is the predicted value. The optimal decision under the observed value is $z^*(\xi)$, and the optimal decision under the predicted value is $z^*(\hat{\xi})$. Regret measures the error in predicting $\hat{\xi}$ by comparing the cost of $z^*(\hat{\xi})$ and the cost of $z^*(\xi)$, after the true value of the parameters is known. The cost of $z^*(\hat{\xi})$ is computed under the optimization problem with parameters ξ. Recall Definition 2.5 from Sadana et al. (2023):

$$\mathcal{L}_{\text{regret}}(\hat{\xi}, \xi) = C(z^*(\hat{\xi}), \xi) - C(z^*(\xi), \xi).$$

The regret loss is well defined when only the objective parameters c are uncertain. Every solution that is feasible under the prediction $\hat{c}$ is also feasible under the actual value c. Only the cost of the decisions changes, so the regret use the formulation in Definition 2.6, which we recall is:

$$\mathcal{L}_{\text{regret}}(\hat{c}, c) = c^\top z^*(\hat{c}) - c^\top z^*(c).$$

This is the most common formulation in the literature on DFL or CSO (Elmachtoub and Grigas, 2022, Kotary et al., 2021, Sadana et al., 2023, Sun et al., 2023). Numerous approaches to optimize this regret loss exist already. For gradient-based minimization algorithms, the difficulty resides in differentiating through the optimization problem $z^*(\hat{c})$, e.g.:

$$\frac{\partial \mathcal{L}_{\text{regret}}}{\partial \hat{c}} = \frac{\partial \mathcal{L}_{\text{regret}}}{\partial z^*(\hat{c})} \cdot \frac{\partial z^*(\hat{c})}{\partial \hat{c}} = c^\top \frac{\partial z^*(\hat{c})}{\partial \hat{c}}. \tag{3.9}$$

The regret-based loss function in Definition 2.5 is ill-defined in the case where the uncertain parameters are in the constraints and not the objective. Predicting the constraint parameters changes the feasible region of the optimization problem. A different feasible region can change the optimal decision, even if the cost vector is identical. Furthermore, decisions that are feasible under the predicted constraints may be infeasible in the actual optimization problem, including the optimum. Figure 3.1 illustrates these points. Therefore, we cannot directly compare the costs of decisions across optimization problems, and we cannot use the regret formulation in Definition 2.6. Instead, we need to define a decision cost $C(z^*(\hat{\xi}), \xi)$ that can evaluate any decision $z^*(\hat{\xi})$.

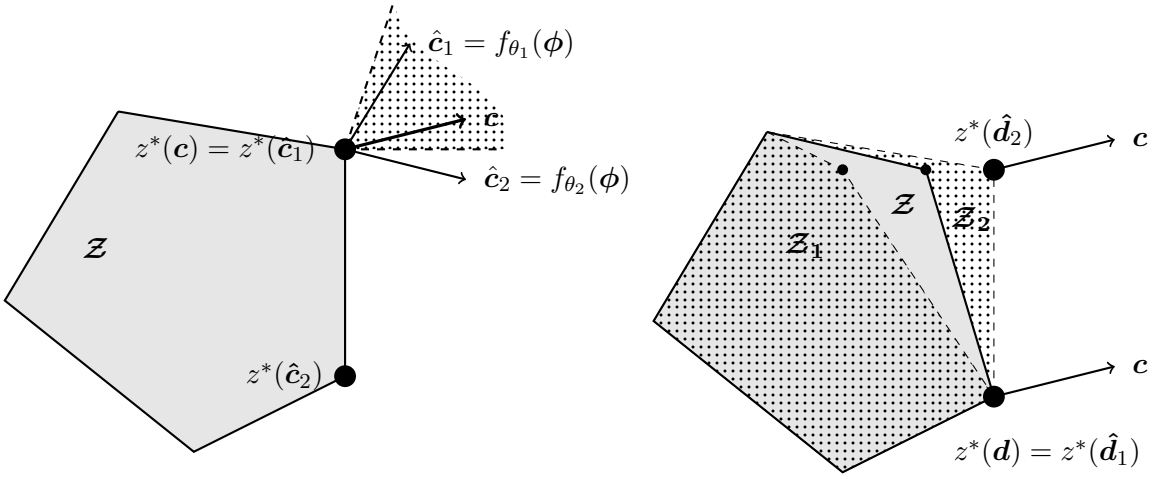


Figure 3.1: Left: uncertain costs – predicting $\hat{\boldsymbol{c}}_1$ results in the optimal decision $z^*(\hat{\boldsymbol{c}}_1) = z^*(\boldsymbol{c})$ whereas a small error resulting from predicting $\hat{\boldsymbol{c}}_2$ leads to a suboptimal decision $z^*(\hat{\boldsymbol{c}}_2)$ under actual cost $\boldsymbol{c}$ (adapted from Sadana et al. (2023)). The feasible region $\mathcal{Z}$ is identical in all cases.

Right: uncertain constraints – predicting constraint parameter $\hat{\boldsymbol{d}}_1$ results in feasible region $\mathcal{Z}_1$, which has the same optimal decision $z^*(\hat{\boldsymbol{d}}_1) = z^*(\boldsymbol{d})$ as the actual feasible region $\mathcal{Z}$. Predicting $\hat{\boldsymbol{d}}_2$ results in feasible region $\mathcal{Z}_2$, whose optimal solution $z^*(\hat{\boldsymbol{d}}_2)$ is not feasible in the actual feasible region $\mathcal{Z}$.

Few ideas exist in the literature for formulating a regret for optimization problems with uncertain

constraints. Only Donti et al. (2017) and Paulus et al. (2021) consider the case of uncertain constraints and not objectives. Donti et al. (2017) define a similar loss to regret for stochastic quadratic programs with both uncertain objectives and constraints. We recall their formulation from (2.4) as:

$$\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{\phi},\boldsymbol{d}\sim\mathcal{D}}[f(\boldsymbol{\phi},\boldsymbol{d},z^*(\boldsymbol{\phi};\theta))] + \sum_{i=1}^{N} \boldsymbol{I}\left\{\mathbb{E}_{\boldsymbol{\phi},\boldsymbol{d}\sim\mathcal{D}}[g_i(\boldsymbol{\phi},\boldsymbol{d},z^*(\boldsymbol{\phi};\theta))] \leq 0\right\}, \tag{3.10}$$

where the indicator function $\boldsymbol{I}\{\cdot\}$ is zero if a constraint is satisfied and infinite otherwise. We note that the loss does not provide meaningful information when a constraint is violated, and that the derivative $\frac{\partial \mathcal{L}_{\text{regret}}}{\partial z^*(\hat{\xi})}$ is not well-defined. Donti et al. (2017) use a form of constrained stochastic gradient descent to address this problem, but do not provide any theoretical guarantees or numerical results showing that the above loss function works when the constraints are uncertain. Paulus et al. (2021) also handle uncertain objective and constraints by providing a meaningful gradient for $\frac{\partial z^*(\hat{\xi})}{\partial \hat{\xi}}$, even when constraints are violated. However, the decisions $z^*(\xi)$ are compared using MSE distance, a loss function that is differentiable with respect to the decision but does not meaningfully capture the cost of a ND program. In the first case, the loss function is based on regret and is defined when the constraints are violated, but its gradient is neither well-defined nor informative when the constraints are violated.

We have two ideas for formulating a regret loss for uncertain constraints: custom losses and using a Lagrangian relaxation. The first idea would be to create a custom loss function that expresses the regret of a decision, similar to the evaluation procedure we developed in Algorithm 3.3. Although this procedure is well defined for our use case, it has many limitations. In particular, it is specifically limited to a ND problem with uncertain commodity demand, and changing the cost of the recourse path changes the regret cost. Furthermore, since computing the cost requires solving another MILP, finding the gradient of the loss requires differentiating through another MILP. The second idea is to apply Lagrangian relaxation to the uncertain constraints, so that the uncertain parameters are in the objective function, and then use the traditional definition of regret. This avoids the issue of uncertain constraints changing the feasible region, but means that the cost no longer represents the true regret under the original problem and some information may be lost. More research is needed in this area.

3.4 *IO-constraint* – Training the Prediction Model Using Inverse Optimization

In this section, we present *IO-constraint*, a prediction model based on IO, and we compare its performance to a baseline linear prediction model in Section 4.1. The model embeds the prediction model inside a forward optimization model and uses IO to recover prediction model weights. *IO-constraint* uses the original MCFND formulation with relaxed integrality

constraints as a forward model. It embeds a linear prediction model into the flow conservation constraints and recovers the model weights using an IO method for constraint recovery.

IO-Constraint recovers the weights of a demand prediction linear model using the IO constraint recovery method from Ghobadi and Mahmoudzadeh (2021). We first define the linear prediction model we use for predicting demands. We then define a forward problem with the linear prediction model embedded in the flow conservation constraints. We formulate the corresponding IO model that recovers the weights of the prediction model.

We use the method from Ghobadi and Mahmoudzadeh (2021) because it can recover parameters in the constraints of an optimization model. We define a forward problem in which some constraint parameters are unknown. The method formulates a corresponding IO model that takes a set of feasible solutions to the forward model and finds the most appropriate value for the unknown constraint parameters. We choose this method because it supports constraint recovery even when some constraints are partially known. However, the method has the following limitations:

1. The forward model must be an LP.

2. The objective parameters must be known.

3. The cheapest feasible solution given to the inverse model is assumed to be optimal.

The training dataset is composed of contextual features, actual demands, and optimal solutions to the original MCFND:

$$\mathcal{D}_{\text{train}} = \{(\boldsymbol{\phi}_l, \boldsymbol{d}_l, x^*(\boldsymbol{d}), y^*(\boldsymbol{d}))\}_{l=1}^{L}. \tag{3.11}$$

We use a linear prediction model f_Θ to produce a point estimate for the demand $\boldsymbol{d} \in \mathbb{R}^{|\mathcal{K}|}$ given the contextual information $\boldsymbol{\phi} \in \mathbb{R}^m$. The model has weights $\boldsymbol{\theta}_k \in \mathbb{R}^m$ for each commodity $k \in \mathcal{K}$, which form a weight matrix $\Theta = [\boldsymbol{\theta}_1 \cdots \boldsymbol{\theta}_{|\mathcal{K}|}]^\top$. Given contextual information $\boldsymbol{\phi} \in \mathbb{R}^m$, the model outputs a prediction for the demand vector $\hat{\boldsymbol{d}} = [\hat{d}^1 \cdots \hat{d}^{|\mathcal{K}|}]^\top$:

$$\hat{\boldsymbol{d}} = f_\Theta(\boldsymbol{\phi}) = \Theta\boldsymbol{\phi} = \begin{bmatrix} \theta_1^\top \boldsymbol{\phi} \\ \vdots \\ \theta_{|\mathcal{K}|}^\top \boldsymbol{\phi} \end{bmatrix}. \tag{3.12}$$

The forward model is based on the MCFND formulation of (2.2). We relax Constraints (2.2e), which we recall are:

$$y_{ij} \in \{0, 1\}, \quad \forall (i, j) \in \mathcal{A},$$

that enforce the integrality of the design variables y_{ij}. Since the demand by node d_i^k in (3.1) is zero everywhere except at commodity origin nodes $\text{orig}(k)$ and commodity destination nodes

$\text{dest}(k)$, we split the flow conservation Constraints (2.2b), whose formulation is:

$$\sum_{j \in \mathcal{N}_i^+} x_{ij}^k - \sum_{j \in \mathcal{N}_i^-} x_{ij}^k = d_i^k, \quad \forall i \in \mathcal{N}, k \in \mathcal{K},$$

into three separate constraints: conservation at commodity destination nodes, where $d_i^k = d^k$, flow conservation at commodity origin nodes, where $d_i^k = -d^k$, and flow conservation at all other nodes, where $d_i^k = 0$. For every context-demand pair $(\boldsymbol{\phi}, \boldsymbol{d})$, we express the actual demand for a commodity d^k as a sum of predicted demand and an error term $\epsilon_k \in \mathbb{R}$:

$$d^k = \theta_k^\top \boldsymbol{\phi} + \epsilon_k. \tag{3.13}$$

We replace the actual demand d^k in Constraints (3.14b) and (3.14c) with the prediction in (3.13) above. This forward model integrates the demand prediction model into the optimization problem. It introduces new decision variables $\boldsymbol{\phi} \in \mathbb{R}^m$. These decision variables are left free in the forward model. The forward model can thus be written as:

(IO-Constraint-forward)

$$\begin{aligned}
\text{minimize} \quad & \sum_{(i,j) \in \mathcal{A}} f_{ij} y_{ij} + \sum_{k \in \mathcal{K}} \sum_{(i,j) \in \mathcal{A}} c_{ij}^k x_{ij}^k && && \text{(3.14a)} \\
\text{subject to} \quad & \sum_{j \in \mathcal{N}_i^+} x_{ij}^k - \sum_{j \in \mathcal{N}_i^-} x_{ji}^k = \theta_k^\top \boldsymbol{\phi}, && \forall k \in \mathcal{K},\ i = \text{dest}(k), && \text{(3.14b)} \\
& \sum_{j \in \mathcal{N}_i^+} x_{ij}^k - \sum_{j \in \mathcal{N}_i^-} x_{ji}^k = -\theta_k^\top \boldsymbol{\phi}, && \forall k \in \mathcal{K},\ i = \text{orig}(k), && \text{(3.14c)} \\
& \sum_{j \in \mathcal{N}_i^+} x_{ij}^k - \sum_{j \in \mathcal{N}_i^-} x_{ji}^k = 0, && \forall k \in \mathcal{K},\ i \in \mathcal{N}_{\text{rest}}, && \text{(3.14d)} \\
& \sum_{k \in \mathcal{K}} x_{ij}^k \leq u_{ij} y_{ij}, && \forall (i,j) \in \mathcal{A}, && \text{(3.14e)} \\
& x_{ij}^k \geq 0, && \forall (i,j) \in \mathcal{A}, k \in \mathcal{K}, && \text{(3.14f)} \\
& 0 \leq y_{ij} \leq 1, && \forall (i,j) \in \mathcal{A}. && \text{(3.14g)}
\end{aligned}$$

The goal of the inverse model is to recover the constraint parameter values $\theta^k \in \mathbb{R}^m$, based on a collection of solutions to the forward model $(\boldsymbol{\phi}_l, x^*(\boldsymbol{d}_l), y^*(\boldsymbol{d}_l))$ from the training dataset $\mathcal{D}_{\text{train}}$. As the conditions outlined at the beginning of this sections are met, we use the tractable multiple-point IO formulation (2.6) from Ghobadi and Mahmoudzadeh (2021). We divide the constraints of the forward problem into two groups: known constraints and unknown constraints. Constraints (3.14d), (3.14e), (3.14f), and (3.14g) form the set of known constraints and are not expressed explicitly in the inverse model. Constraints (3.14b) and (3.14c) are only partially known. Although we know the coefficients for the flow variables, the prediction weights and

the error terms remain unknown. Therefore, we consider these partially known constraints as part of the unknown constraints and add the known information as new constraints on the constraint parameters in the inverse model. We use the formulation from (2.6) to write the inverse model and adapt it to the constraints of the forward model. We detail the translation in Appendix A.1. We obtain the following optimization model:

(*IO-constraint*)

$$\underset{\theta_k,\,\epsilon}{\text{minimize}} \quad \sum_{k=1}^{K}\sum_{l=1}^{L}(\epsilon_k^l)^2 \tag{3.15a}$$

subject to

$$\sum_{j\in\mathcal{N}_i^+}[x^*(\boldsymbol{d}_l)]_{ij}^k - \sum_{j\in\mathcal{N}_i^-}[x^*(\boldsymbol{d}_l)]_{ji}^k - \sum_{r=1}^{m}\theta_{kr}\boldsymbol{\phi}_r^l = \epsilon_k^l, \qquad \forall k\in\mathcal{K},\, i=\text{dest}(k),\, l=1,\dots,L, \tag{3.15b}$$

$$\sum_{j\in\mathcal{N}_i^+}[x^*(\boldsymbol{d}_l)]_{ij}^k - \sum_{j\in\mathcal{N}_i^-}[x^*(\boldsymbol{d}_l)]_{ji}^k + \sum_{r=1}^{m}\theta_{kr}\boldsymbol{\phi}_r^l = -\epsilon_k^l, \quad \forall k\in\mathcal{K},\, i=\text{orig}(k), l=1,\dots,L, \tag{3.15c}$$

$$\theta_r^k \in \mathbb{R}, \qquad \forall\, k\in\mathcal{K}, r=1\dots m, \tag{3.15d}$$

$$\epsilon_m^l \in \mathbb{R}, \qquad \forall\, k\in\mathcal{K}, l=1,\dots L. \tag{3.15e}$$

The inverse formulation introduces parameters ϵ_k^l, which measure the slack of the flow conservation Constraints (3.15b) and (3.15c) for each forward solution $l=1,\dots,L$. By minimizing the sum of the squared slack distances, the methodology uses the adjacency measure from Ghobadi and Mahmoudzadeh (2021) as a loss function $\mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D})$ to determine the feasible region.

We employ IO techniques to train a predictive model that is capable of being utilized on fresh data points without having to rerun the inverse model. The *IO-constraint* inverse model retrieves the weights θ_k of the prediction model by minimizing slack in the flow conservation constraints, using a training dataset of forward solutions $\mathcal{D}_{\text{train}}$. These weights are then used to create a trained prediction model $f_{\hat{\Theta}}$, which can project the demand for new contextual information $\boldsymbol{\phi}_{\text{new}}$.

We expect *IO-constraint* not to optimize for decision regret but instead to minimize prediction error. Indeed, as explored in Section 3.3, regret-based losses are ill-defined when the constraints are uncertain. Furthermore, we can see by inspection of constraints (3.15b) that the left-hand side is the difference between the total flow on destination node i and the demand prediction made by the model. However, in the forward solution, the total flow on destination node i comes from the original MCFND problem, meaning the total flow sums to the actual demand

d_l^k. We can thus rewrite the constraint as:

$$\boldsymbol{d}_l^k - \sum_{r=1}^{m} \theta_{kr} \boldsymbol{\phi}_r^l = \epsilon_k^l. \tag{3.16}$$

The same rewriting can be done for constraints (3.15c) by replacing the total flow with $-\boldsymbol{d}_k^l$ The ϵ_k^l slack variables can be interpreted as the residuals between the actual demand $\boldsymbol{d}_l^k$ and the predicted demand $\sum_{r=1}^{m} \theta_{kr} \boldsymbol{\phi}_r^l$. By minimizing the sum of the squared residuals, we are actually performing Ordinary Least Squares regression. We can also perform Least Absolute Deviation regression by minimizing the sum of the absolute value of the residuals instead. In any case, we do not expect this model to perform any better than a regular OLS or LAD regression on the training dataset $(\boldsymbol{\phi}_l, \boldsymbol{d}_l)$.

3.5 Improving on Linear Regression: Re-weighting the Training Examples

None of the methods presented so far can improve on sequential Prediction-Optimization pipelines for ND problems. Issues with the ill-definition of regret when the constraints are uncertain prevent us from using existing integrated prediction and optimization methods from the literature. IO methods for constraint recovery suffer from a similar problem: despite integrating the prediction model training into the optimization problem, performance does not improve over a separately trained prediction model. New approaches are needed, which is beyond the scope of this thesis. However, in this section, we outline how DFL can be viewed as a reweighting of the loss function, and how this view can be a starting point for a new approach to solving DFL problems.

Predictive models trained on prediction accuracy will perform well in a Prediction-Optimization pipeline when the model and the data are aligned. For example, when demands are generated independently using a linear function of contextual information, a linear prediction model will produce accurate predictions, resulting in good downstream optimization costs. If the model is misspecified with respect to the data, it will not accurately reflect the data distribution. The model must then make trade-offs and aims for the lowest prediction error averaged over all training examples. However, some predictions may affect the downstream optimization problem more than others. Predictions that have a large impact on the downstream optimization cost could be weighted more heavily in the training procedure. This would result in a model that makes less accurate predictions overall, as measured by prediction loss, but is more accurate for predictions that have a large impact on optimization cost.

DFL can be thought of as a reweighting of the training examples in the loss function, where the weighting is a hyperparameter of the predicting model. We need a systematic way to find weightings that improve downstream optimization cost. Ideally, the weighting of a commodity

should reflect the marginal contribution of the commodity to the cost of the optimization problem. The experiment in Section 4.2 demonstrates the usefulness of reweighting the training examples in the loss function to obtain lower downstream optimization cost despite higher prediction error. In this example, good weightings can be found by inspecting the structure of the ND problem, but this is however not possible in more complex cases.

Suppose we use the Prediction-Optimization pipeline evaluation procedure from Algorithm 3.3 as in our new definition of regret, then we can formulate the search for the correct weightings as a nonlinear, multi-level optimization problem that finds prediction model weights and training example weightings that minimize regret over the training examples. We express the regret of a demand prediction $\hat{\boldsymbol{d}}_l$ for the l-th training example $(\boldsymbol{\phi}_l, \boldsymbol{d}_l)$. If MCFND-Flow$(\boldsymbol{d}_l, y^*(\hat{\boldsymbol{d}}_l))$ is the evaluation procedure cost of the prediction from (3.7), and MCFND$(\boldsymbol{d}_l)$ is the true cost of fulfilling demand $\boldsymbol{d}_l$, we can write the regret as:

$$\mathcal{L}_{\text{regret}}(\hat{\boldsymbol{d}}_l, \boldsymbol{d}_l) = \text{MCFND-Flow}(\boldsymbol{d}_l, y^*(\hat{\boldsymbol{d}}_l)) - \text{MCFND}(\boldsymbol{d}_l). \tag{3.17}$$

Our optimization problem defines prediction weights and training example weightings that minimize regret (3.17) over the training dataset. We formulate this optimization as follows:
(W-DFL)

$$\begin{aligned} \underset{\substack{\boldsymbol{w} \in \mathbb{R}^L_{\geq 0}}}{\text{minimize}} \quad & \sum_{l=1}^{L} \left[\text{MCFND-Flow}(\boldsymbol{d}_l, y^*(f_{\hat{\Theta}}(\boldsymbol{\phi}_l|\boldsymbol{w})) - \text{MCFND}(\boldsymbol{d}_l)\right] && \text{(3.18a)} \\ \text{subject to} \quad & \sum_{l=1}^{L} w_l = 1 && \text{(3.18b)} \end{aligned}$$

The optimization problem defines a weighting $\boldsymbol{w} = [w_1, \ldots, w_L]$ that minimizes the regret over the training dataset in Objective (3.18a). Constraint (3.18b) ensures the weights sum to 1. The demand predictions are made with a weighted linear prediction model:

$$\hat{\boldsymbol{d}}_l = f_{\hat{\Theta}}(\boldsymbol{\phi}_l|\boldsymbol{w}) = \hat{\Theta}\boldsymbol{\phi}_l, \tag{3.19}$$

where the predictions weights $\hat{\Theta} = [\hat{\theta}_1, \ldots, \hat{\theta}_{|\mathcal{K}|}]^\top$ depend on the weighting $\boldsymbol{w}$ of the training examples. The following optimization problem is a weighted linear regression and defines the prediction weights $\hat{\Theta}$:

$$\hat{\theta}_1, \ldots, \hat{\theta}_{|\mathcal{K}|} = \underset{\theta_1 \ldots \theta_{|\mathcal{K}|}}{\arg\min} \quad \sum_{l=1}^{L} \left[w_l \cdot \sum_{k=1}^{K} (\epsilon_l^k)^2 \right] \tag{3.20a}$$

$$\text{subject to} \quad \sum_{r=1}^{m} \theta_{kr} \boldsymbol{\phi}_r^l + \epsilon_l^k = d_l^k, \quad \forall k \in \mathcal{K},\, l = 1, \ldots, L, \tag{3.20b}$$

$$\epsilon_l^k \in \mathbb{R}, \quad \forall\, k \in \mathcal{K}, l = 1, \ldots L, \tag{3.20c}$$

$$\theta_r^k \in \mathbb{R}, \quad \forall\, k \in \mathcal{K}, r = 1 \ldots m. \tag{3.20d}$$

The prediction weights defined by the W-DFL model minimize regret over the training examples, thus achieving the goal of integrating the prediction and optimization steps of a pipeline, but W-DFL is a nonlinear and multilevel optimization problem that cannot be solved efficiently with current algorithmic techniques. Blackbox optimization solvers such as NOMAD (Audet et al., 2022) can solve small instances of this problem, but we propose a simple iterative method to approximate the solution to W-DFL. This method is not intended to find the optimal solution to W-DFL, but to show the possible value of iterative reweighting. The iterative method begins at $t = 0$ with all training examples weighted equally:

$$w_l^{(0)} = 1, \quad \forall l = 1, \ldots, L. \tag{3.21}$$

At each subsequent iteration, we train a linear regression model on the training dataset using the current weightings. Using the trained prediction model, we predict the demand for each training example and compute the regret $\mathcal{L}_{\text{regret}}(\hat{\boldsymbol{d}}_l^{(t)}, \boldsymbol{d}_l)$ of the training example. We update the weighting of each training dataset proportionally to its overall contribution to the total regret , then use the new weightings at the next iteration. We can write the weight update at iteration $t = 0, 1, \ldots$ as:

$$\bar{w}_l^{(t+1)} = w_l^{(t)} + \frac{\mathcal{L}_{\text{regret}}(\hat{\boldsymbol{d}}_l^{(t)}, \boldsymbol{d}_l)}{\sum_{l'=1}^{L} \mathcal{L}_{\text{regret}}(\hat{\boldsymbol{d}}_{l'}^{(t)}, \boldsymbol{d}_{l'})} \cdot w_l^{(t)}, \tag{3.22}$$

$$w_l^{(t+1)} = \frac{\bar{w}_l}{\sum_{l=1}^{L} \bar{w}_l}, \tag{3.23}$$

where (3.22) computes the proportional update and normalizes (3.23) the weights between 0 and 1. We test this iterative method on a small example in Section 4.3. This method is currently a heuristic without strong theoretical backing. It may be limited to specific networks and situations.

Chapter 4

Results

This chapter describes the experiments with the methodology in Chapter 3. We begin by examining the IO-Constraint prediction model. We show that a prediction model can be trained using this IO method, but its performance is no better than ordinary linear regression. We also test reweighting the training examples in the loss function. On a small ND problem, we first show that regret improves when setting the weight manually, and then test the performance of the iterative weight update method.

4.1 Evaluating *IO-constraint*

We evaluate *IO-constraint* on two experiments that use synthetic data. The first experiment is a simple two-commodity ND problem with uncertain demands, where we demonstrate that *IO-constraint* can train a linear prediction model. The second example compares the performance of *IO-constraint* with that of a standard linear regression model on a ND problem where the demand is very close to the arc capacity. We also outline the data generation process for the training and test datasets that is common to both examples.

4.1.1 Data Generation

Both experiments rely on synthetic datasets for training and evaluation. An experiment is defined on a MCFND problem instance in which the demand is uncertain. Our procedure generates can generate a training and test dataset. Each point in the dataset contains the contextual information: $\boldsymbol{\phi} \in \mathbb{R}^m$, the actual demands $\boldsymbol{d} = [d^1 \ \dots \ d^{|\mathcal{K}|}]^\top \in \mathbb{R}^{|\mathcal{K}|}$, and the optimal solution to the MCFND for those demands: $x^*(\boldsymbol{d}), y^*(\boldsymbol{d})$

Our procedure generates a dataset of L training points $\mathcal{D} = \{(\boldsymbol{\phi}_l, \boldsymbol{d}_l, x^*(\boldsymbol{d}_l), y^*(\boldsymbol{d}_l))\}_{l=1}^{L}$ which we split into a training and test set $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$.

In our experiments, we assume the demands are a linear function of the contextual information with added Gaussian noise. Given contextual information $\boldsymbol{\phi} \in \mathbb{R}^m$, a vector $\theta_k \in \mathbb{R}^m$ and the noise standard deviation σ_k, we can express the demand for commodity k as follows:

$$d^k = \theta_k^\top \boldsymbol{\phi} + \epsilon^k \qquad \text{with} \quad \epsilon^k \sim \mathcal{N}(0, \sigma_k^2). \tag{4.1}$$

In our experiments, we wish to generate a dataset of context such that the expected value of the demand is some fixed value $B_k = \mathbb{E}[d^k] \in \mathbb{R}$. Since there are m points of contextual information and we assume that $m \geq |\mathcal{K}|$, we can randomly generate $m - |\mathcal{K}|$ components of $\boldsymbol{\phi}$ and compute the remaining $|\mathcal{K}|$ components. Given fixed weights $\theta_1, \ldots, \theta_{|\mathcal{K}|}$, we generate each point $(\boldsymbol{\phi}, \boldsymbol{d}, x^*(\boldsymbol{d}), y^*(\boldsymbol{d})$ in the dataset using the following procedure:

Algorithm 4.1: Data generation

Input: number of points N
target expected demands B_k and variances σ_k
demand generation weights $\theta_1, \ldots, \theta_{|\mathcal{K}|}$
contextual information bounds $a, b \in \mathbb{R}$

Output: Datasets $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{train}}$

Create empty dataset $\mathcal{D}$
for *i = 1, . . . , N* **do**
  Draw $\phi_1, \ldots, \phi_{m-|\mathcal{K}|} \sim U(a, b)$
  Set $\phi_{m-|\mathcal{K}|+1}, \ldots, \phi_m$ by solving $B_k = \theta_k \boldsymbol{\phi}$ for all k
  Draw noises $\epsilon^k \sim \mathcal{N}(0, \sigma_k^2)$ for all k
  Solve MCFND($\boldsymbol{d}$) to get $x^*(\boldsymbol{d}), y^*(\boldsymbol{d})$
  Add sample $(\boldsymbol{\phi}, \boldsymbol{d}, x^*(\boldsymbol{d}), y^*(\boldsymbol{d})$ to $\mathcal{D}$
end
Separate $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ using a random 70%-30% split
Return $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{test}}$

4.1.2 First Experiment: Training a Prediction Model with IO-Constraint

This experiment tests whether *IO-constraint* is capable of training a linear prediction model that predicts commodity demands from contextual information. We test the method on a MCFND instance with two nodes and two arcs p and q connecting them, as in Example 3.1. Arc p has a low design cost but is has a low capacity $u_p = 15$, whereas arc q is not capacity constrained but has a high design and flow cost. Two commodities $k = 1$ and $k = 2$ flow from node s to t. We fix the size of the contextual information vector $m = 3$ and weights θ_1, θ_2 then generate a training and test set using Algorithm 4.1. We choose the expected value of the demands B_1 and B_2 such that the sum of both is slightly above arc p's capacity (i.e. $B_1 + B_2 = 16 \geq u_p$) and set a standard deviation of $\sigma_1 = 2.5$ and $\sigma_2 = 1.2$. This results in the optimal solution only building arc p in some cases, and building both arcs in others.

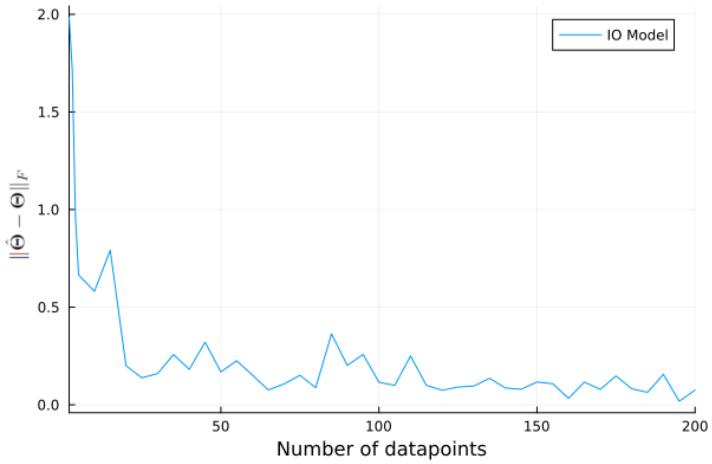


Figure 4.1: Distance between weights $\hat{\Theta}$ recovered by *IO-constraint* and actual weights Θ as training dataset size increases.

The *IO-constraint* model takes in a dataset of contextual information $\boldsymbol{\phi}_l$ and forward solutions $x^*(\boldsymbol{d})_l$ and recovers prediction weights $\hat{\Theta} = [\hat{\theta}_1\, \hat{\theta}_2]^\top$. We test the model on training datasets of increasing size and we observe the Frobenius norm of the difference between the predicted weights and the actual weights used to generate the data.

$$\|\hat{\Theta} - \Theta\|_F = \sqrt{\sum_{k=1}^{|\mathcal{K}|} \sum_{i=1}^{m} (\hat{\theta}_{km} - \theta_{km})^2} \tag{4.2}$$

This aims to measure the distance between the recovered weights and the actual weights used to generate the data. As the model and the data generation process are aligned, we expect that the distance between the two weights decreases to zero as the size of the training dataset increases, with a small imprecision due to the added noise in the demands.

We see in Figure 4.1 that the distance between the weights $\hat{\Theta}$ recovered by *IO-constraint* and the actual weights Θ decreases to zero as the size of the dataset increases. In particular, the distance is close to zero after $L = 25$ points and then hovers slightly above zero. We attribute the remaining difference between the weights to the noise added to the demands in the data generation process. We determine that *IO-constraint* can indeed train a prediction model and recover appropriate weights when the data and the prediction model are aligned.

We have demonstrated in this experiment that it is possible to train a linear prediction model using IO-Constraint, but we have not compared its performance to other models such as standard linear regression.

4.1.3 Second Experiment: Comparing *IO-constraint* to Linear Regression

The second experiment compares the downstream optimization cost of *IO-Constraint* and a linear regression model, on a problem that is designed to highlight a potential advantage of a DFL model. The ND problem is constructed such that a DFL model which is aware of downstream costs would make different predictions compared to a model trained on prediction accuracy. We train an *IO-Constraint* and a linear regression prediction model on the same dataset. We compare the predictions of each model and also compare the resulting downstream cost of each model using the evaluation procedure in Algorithm 3.3.

Our experiment is defined on the network from Example 3.1 with a dataset of contextual information-demand pairs $\mathcal{D}$. The dataset is composed of two groups of demands of equal size $\mathcal{D}_{\text{train}} = \mathcal{D}^{\text{close}} \cup \mathcal{D}^{\text{wide}}$. Both groups are generated using Algorithm 4.1, such that the expected demand of $\mathcal{D}^{\text{close}}$ is equal to the capacity of the cheapest arc, and the expected demand of $\mathcal{D}^{\text{close}}$ is well below the capacity of the cheapest arc. We split the dataset into a training and test dataset.

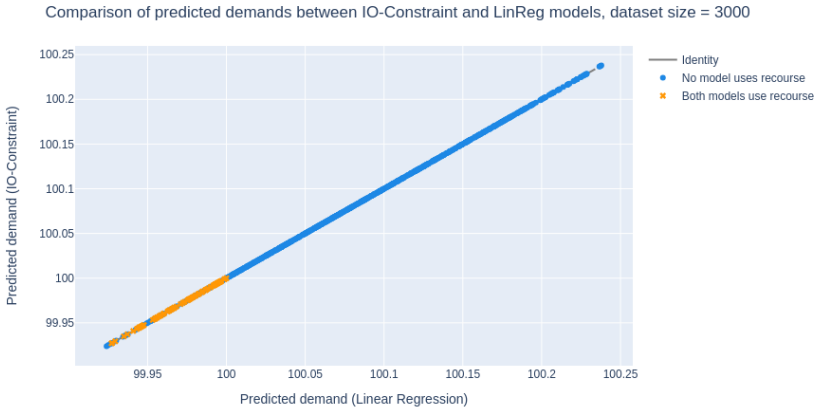


Figure 4.2: Comparison of predictions made by IO-Constraint and a linear regression model. Both models make the same prediction given the same input.

We expect a prediction model trained on prediction accuracy to predict values in-between the expected demand of $\mathcal{D}^{\text{close}}$ and the expected value of $\mathcal{D}^{\text{wide}}$, whereas we expect a DFL prediction model trained on downstream cost to predict close to the expected value of $\mathcal{D}^{\text{close}}$, since DFL lends more weight to the solutions that strongly affect decision cost.

We evaluate two linear regression models: an Ordinary Least Squares (OLS) regression model and *IO-Constraint*. We compare the values of the demand prediction made by each model in Figure 4.2, and the distribution of downstream cost and flow on the recourse arcs used for each model in Figure 4.3. As we can see in both figures, both models have the same downstream cost and even produce the exact same prediction given the same contextual information on the test set.

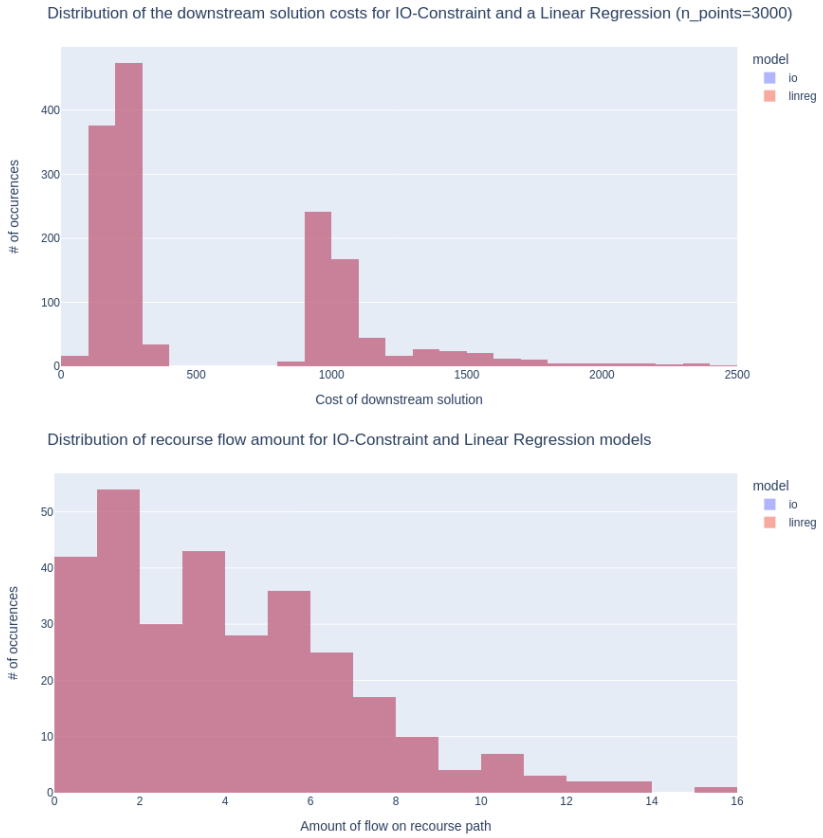


Figure 4.3: Distributions of the costs of the downstream solution (top) and of the amount of flow on the recourse arcs (bottom) for *IO-Constraint* and a linear regression model. The distributions of both models are identical.

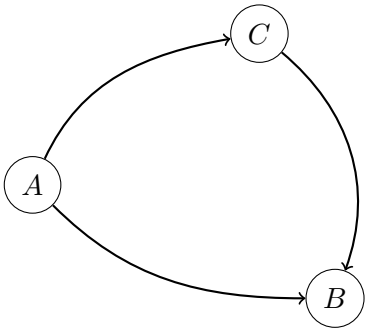
Our experiment thus confirms our idea that a prediction model trained with *IO-Constraint* corresponds to OLS regression. It does not perform DFL and more work is needed to improve the downstream cost of predictions.

4.2 Reweighting Training Examples with a Misspecified Model

This experiment showcases the utility of reweighting the training examples in the training phase discussed in Section 3.5. In this section, we show a way to improve the downstream cost of a linear predictive on a small ND problem. We use a linear regression model and change the weighting of the training examples in the loss function, treating the weighting as a hyperparameter of the model. By weighting training samples that strongly impact downstream optimization more heavily, we improve the performance of the predictive model once training is over, even though the prediction model is misspecified with respect to the data.

Consider a MCFND problem with three nodes $\mathcal{N} = \{A, B, C\}$, three arcs $\mathcal{A} = \{AB, AC, CB\}$,

and two commodities $k = 1$ and $k = 2$.



Commodity $k = 1$ originates in A and flows to B with demand d^1. Commodity $k = 2$ also originates in A and goes C with demand d^2.
Arc capacity is almost equal to demand on arc AB and is much higher than demand on arcs AC and CB: $u_{AB} \approx d^1$ and $u_{AC}, u_{CB} \gg d^1 + d^2$.

Flow costs are zero $c_{ij}^k = 0$ for all arcs and commodities and design costs are equal $f_{AB} = f_{AC} = f_{CB} = F$.

Suppose the demands are generated from the same contextual information $\phi \in \mathbb{R}$ using two different linear functions with added noise:

$$\begin{aligned} d^1 &= \theta_1^{\text{int}} + \theta_1 \phi + \epsilon^1 \\ d^2 &= \theta_2^{\text{int}} + \theta_2 \phi + \epsilon^2 \end{aligned} \tag{4.3}$$

where θ_1^{int} and θ_2^{int} are the intercepts, θ_1 and θ_2 the slopes, $\epsilon^1, \epsilon^2 \sim \mathcal{N}(0, \sigma^2)$ the independently-drawn Gaussian noise for each respective commodity. The parameters are select such that the expected value of the demand for commodity $k = 2$ is very close to the capacity of arc AB, i.e. $\mathbb{E}[d^2] \approx u_{AB}$.

Now consider a misspecified linear prediction model $f_{\theta^{\text{int}},\theta}$, which predicts each commodity with two different intercepts θ_1^{int} and θ_2^{int} but the same slope $\theta \in \mathbb{R}$:

$$\begin{aligned} \hat{d^1} &= f_{\theta_1^{\text{int}},\theta}(\boldsymbol{\phi}) = \theta_1^{\text{int}} + \theta \boldsymbol{\phi} \\ \hat{d^2} &= f_{\theta_2^{\text{int}},\theta}(\boldsymbol{\phi}) = \theta_2^{\text{int}} + \theta \boldsymbol{\phi} \end{aligned} \tag{4.4}$$

The prediction model is trained over a dataset of contextual info and demand pairs. The dataset is comprised of samples from the distribution of commodity 1 and commodity 2: $\mathcal{D}_{\text{train}} = \mathcal{D}_{\text{train}}^1 \cup \mathcal{D}_{\text{train}}^2 = \{(\phi_i, d_i, k_i)\}_{i=1}^N$, where k_i indicates which commodity the training sample refers to.

Looking closely at this example, we see that accurately predicting demand d^1 is much more important than predicting demand d^2. Because d^1 is so close to the capacity of arc AB, overprediction results in both arcs AB and CB being built, even though arc AB would have been sufficient. This increases the cost of the final downstream optimization problem from $2F$ to $3F$. Because the demand d^2 is so far from the capacity of arc AC, the prediction error of d^2 has no effect on which arcs are built.

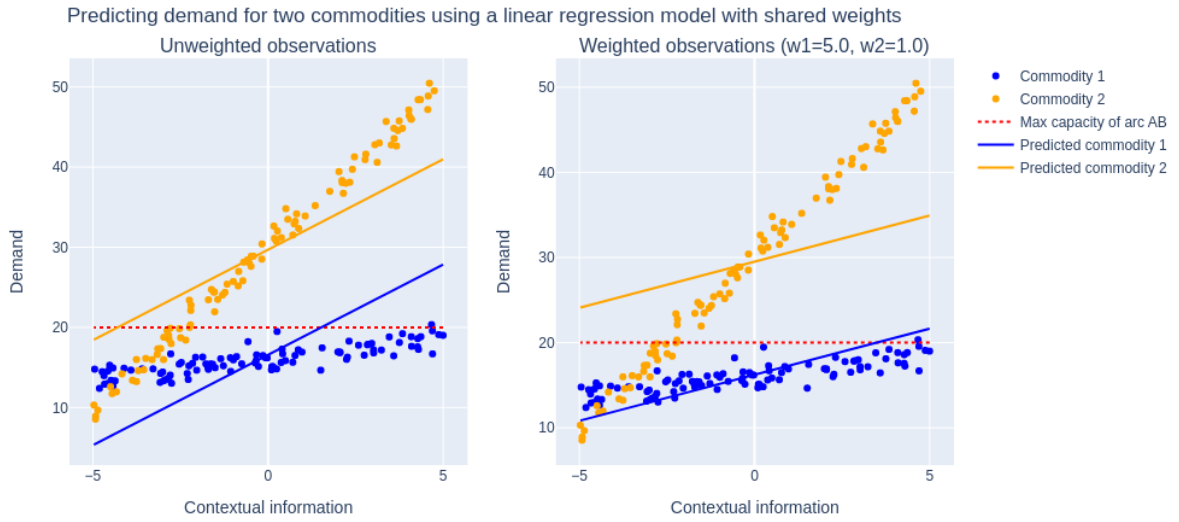


Figure 4.4: Comparison of linear regression with unweighted and weighted training examples. The weighted model has worse prediction performance but lower downstream optimization cost overall.

Figure 4.4 illustrates the example with an arc cost of $F = 1$, and compares two linear regression prediction models defined above. Despite having a higher overall prediction error, the weighted model predicts demands that result in lower downstream optimization costs and regret, as shown in Table 4.1. In the unweighted case, each training sample is equally weighted in the model's loss function. A prediction error for commodity 1 will have the same effect on the model as a the same prediction error for commodity 2. The slope of the unweighted prediction model lands halfway in-between the two commodities. In the weighted case, the training samples from commodity 1 are weighted 5 times more than those from commodity 2. This results in a flatter slope, much closer to the slope of commodity 1. The unweighted model predicts a demand for commodity 1 that is above the arc capacity of AB much more often than the weighted model.

Model	RMSE	Optim. cost	Regret
Unweighted	**5.19**	1.15	0.15
Weighted	6.18	**1.07**	**0.07**

Table 4.1: Caption

4.3 Iterative Search for Training Example Weightings

In this experiment, we test the iterative weight update method devised in Section 3.5. Having shown in Section 4.2 above that reweighting the training examples can lead to better downstream performance, we test a systematic way to find a weighting of the training examples that improves downstream performance. This corresponds to finding a solution for the W-DFL problem

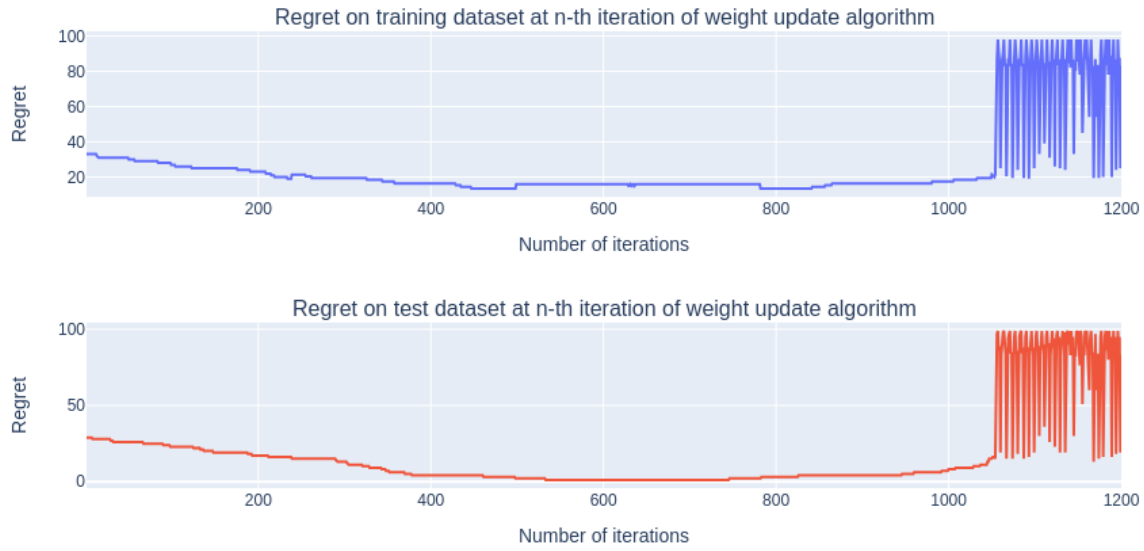


Figure 4.5: Progression of training and test regret at every iteration of the weight update algorithm.

defined in (3.18) using the iterative weight update from (3.22) and (3.23). We use the same ND example problem and experimental setup as in Section 4.2. We set the flow cost of the recourse arcs in MCFND-Flow to $c_{\text{rec}} = 10$ to cover the case where the demand is higher than the capacity of arc AB, but the model predicts a demand below the capacity. We generate a second dataset $\mathcal{D}_{\text{test}}$ using the method as $\mathcal{D}_{\text{train}}$ to evaluate the regret of the prediction model on unseen data. Starting with all training examples weighted equally at iteration $t = 1$, we update the weightings until $t = 1200$ iterations.

The iterative method improves regret on this example up to a certain point, after which regret increases dramatically. Figure 4.5 shows the evolution of regret at every iteration, on both the training dataset $\mathcal{D}_{\text{train}}$ and the test dataset $\mathcal{D}_{\text{test}}$. It shows that regret steadily decreases with the number of iterations, from an initial test regret of 28 when all observations are weighted equally, to the minimum test regret of 1.22 starting at iteration $t = 538$. However, as the number of iterations increases past $t = 700$, the regret starts to rise. Finally, after $t = 1057$, the regret oscillates in an unstable manner between a high value of 100 and a lower value around 20. This is a phenomenon specific to our example, where the only examples contributing to regret after enough iterations are those with a demand above the capacity of arc AB. Those examples become very heavily weighted, causing all the other predictions to be above the arc capacity and thus increasing regret for all training examples drastically.

This experiment shows the promise of a iterative weight update method for solving the W-DFL problem. The method yields weightings that improve downstream optimization cost on a

small example. However, without a theoretical backing, this behaviour could be limited to this small example. This method also seems to present instability after a sufficient number of iterations. Perhaps a better weight update rule or stopping criterion could further improve the performance of the iterative method.

Chapter 5

Conclusion

At the beginning of this thesis, we embarked on a research journey to study how to improve demand predictions for ND problems. We set out to integrate the prediction and optimization steps so that prediction models would make predictions that minimize optimization costs. The problem of using DFL to predict the demand for commodities in ND problems had not been studied before, and presented several unique features: ND is an optimization problem with high computational complexity, integer and continuous variables, and uncertain parameters in the constraints, not the objective. The objective of this thesis was to perform an exploratory study on the integration of the prediction and optimization steps for ND problems. We first compiled a literature review in the fields of ND, DFL, and IO, and then attempted to apply existing methods to our problem. We found that existing methods cannot be used and that new approaches are needed. We explored a promising method based on reweighting the loss function of the prediction model. In the following, we retrace the steps of our study and discuss our results in Section 5.1, and provide avenues for future research in Section 5.2

5.1 Discussion

Our first step was to review the existing literature related to the three fields of ND, DFL and IO, as we believed we could find or combine existing techniques from each field to solve our problem.

ND is a well-established branch of operations research that uses mathematical optimization to design an optimal network in a graph. Uncertainty in ND is traditionally handled with stochastic or robust optimization models, that optimize over the entire distribution or within an uncertainty set of the uncertain parameters. These models are computationally expensive. In practice, we use less computationally expensive deterministic ND models and make point predictions for the uncertain parameters.

The field of DFL, still in its infancy, lies at the intersection of machine learning and optimization research. DFL models integrate the prediction and optimization steps by training the prediction model to make predictions that minimize the downstream optimization cost. Most DFL models define a loss function called regret, which is the difference in cost between the optimization network with the predicted parameters and the problem with the actual parameters. The training procedure finds the prediction model that minimizes the loss function using a variant of gradient descent. Gradient descent requires that the loss function be well-defined and differentiable. Most research in DFL focuses on uncertain objective parameters, where the regret is well-defined. The few works that consider uncertain constraint parameters do not use regret and minimize a simpler, differentiable loss function.

IO initially seemed unrelated to our problem because it focuses on recovering the uncertain parameters of an optimization model given solutions to the problem. It typically does not produce a trained prediction model that generalizes to new instances of the optimization problem. Instead, most IO models are re-run for each instance with different parameters. However, IO has a number of advantages: as a more mature field, it has techniques for recovering parameters in the constraints and can handle integer variables. Our hope was to integrate a prediction model into an inverse optimization problem, and train it using constraint recovery techniques to build our DFL model. While working on this thesis, Sun et al. (2023) published research formulating an IO model that trains a prediction model to predict objective parameters. The IO model defines prediction model weights that minimize a measure of optimality of the original LP problem. The prediction model thus makes predictions that minimize the downstream optimization cost. This research seemed promising, but it was limited to LPs with uncertain costs only.

After reviewing the literature and finding no existing solution that perfectly fit our problem, we attempted to apply methods not quite designed for our use case in the hope that they would work, or would provide insight into our ND problem. We found that issues with the definition of regret for problems with uncertain constraints explain why existing solutions do not work.

We first formulated our ND problem as a stochastic program with uncertain demands, and defined a general Prediction-Optimization pipeline that solves our ND problem. It makes a point prediction of the demand given the contextual information, and then solves an ND problem that fulfills that demand. We ran into the problem of evaluating the performance of a Prediction-Optimization pipeline and subsequently into the ill-definition of regret when the parameters are uncertain. Since predicting a different value for demand changes the feasible region of the optimization problem, we cannot directly compare the cost of solutions from two different predictions. We developed an evaluation method that can compare solutions and that ensures the solution with the actual demand is the least costly. We did this by calculating the cost of fulfilling the actual demand over the network designed by the pipeline. We ensured the feasibility of each designed network by adding expensive recourse paths to the ND problem.

We found that the formulation of regret when only the costs are uncertain cannot be used when the constraints are uncertain, because the feasible region can change depending on the prediction of the constraint parameters. Finding a definition of regret for uncertain constraints that is differentiable is difficult. This problem has so far not been addressed in the literature.

Before we discovered the issues with regret when the constraints are uncertain, we formulated *IO-constraint*, an IO model that recovers the weights of a linear prediction model. We were hoping that it would be possible to recover weights that minimize downstream optimization cost. Our model does indeed recover constraint parameters, but optimizes for prediction accuracy and not downstream cost. We would have needed to use an optimality condition, but the one from Sun et al. (2023) only applies to LPs with uncertain costs. Our experiments confirmed the fact that *IO-Constraint* performs no better than a linear regression model.

After recognizing the problems with *IO-Constraint* and with the definition of regret, we explored ideas for performing DFL by reweighting observations. The value of the loss function for each prediction would be weighted so that predictions that have a greater impact on downstream costs more heavily are weighted more heavily in the loss function. We have shown in an experiment that we can significantly improve downstream costs when the prediction model and the data model are misaligned by reweighting some observations. The difficulty with this method is finding the appropriate weights for each demand. We formulated W-DFL, a multilevel nonlinear optimization problem that defines weightings for the training examples that minimize regret. We developed an iterative method that updates the weights based on the contribution of the training example to the regret. It shows promising results in reducing regret on a small example network, but becomes unstable after a sufficient number of iterations.

In this thesis, we set out to integrate the prediction and optimization steps for ND problems. We conducted a review of the literature in the fields of ND, DFL, and IO to find methods that would be applicable to our problem. This problem had not been attempted before, and we could not easily adapt existing techniques for uncertain costs and continuous variables to work on our problem. We explored new techniques that reweight observations in the loss, but they require more research. The main conclusion we drew from our study is that is that the definition of regret when the objective is uncertain does not apply when the constraints are uncertain. Expressing regret in our use case is much more complicated than previously expected, depends on the structure of the optimization problem, and is typically not differentiable.

5.2 Future work

We suggest two important directions for future research: reweighting the observations and defining a regret-based loss suitable for problems with uncertain constraints. In the immediate future, we could further investigate the problem of reweighting the observations in the loss function. This method has shown promising results in this thesis, but does not yet have

an efficient and accurate way to find the appropriate weights. The iterative weight update algorithm shows promising results, but does not converge to a minimum. In later work, we could develop methods that have stronger theoretical backing, similar to the Multiplicative Weights Algorithm (Arora et al., 2012), and test them on larger and more realistic networks. In addition, we could try other techniques for solving the W-DFL problem, such as black-box optimization solvers like NOMAD.

This work also uncovers an important, previously unexplored area of research: regret-based loss for problems with uncertain constraints. We could search for a definition of regret that yields meaningful results when the constraints are uncertain and that does not depend on the specifics of the optimization problem. Finding such a definition of regret that is also differentiable with respect to the solution would allow us to apply many existing DFL techniques to our problem.

Bibliography

Alonso-Ayuso, A., Escudero, L., Garín, A., Ortuño, M., and Pérez, G. An Approach for Strategic Supply Chain Planning under Uncertainty based on Stochastic 0-1 Programming. *Journal of Global Optimization*, 26(1):97–124, 2003.

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

Arora, S., Hazan, E., and Kale, S. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. URL `https://theoryofcomputing.org/articles/v008a006`.

Audet, C., Le Digabel, S., Rochon Montplaisir, V., and Tribes, C. Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 48(3):35:1–35:22, 2022.

Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

Bodur, M., Chan, T. C., and Zhu, I. Y. Inverse mixed integer optimization: Polyhedral insights and trust region methods. *INFORMS Journal on Computing*, 34(3):1471–1488, 2022.

Burton, D. and Toint, Ph. L. On an instance of the inverse shortest paths problem. *Mathematical Programming*, 53(1):45–61, 1992.

Cadarso, L., Escudero, L. F., and Marín, A. On strategic multistage operational two-stage stochastic 0–1 optimization for the Rapid Transit Network Design problem. *European Journal of Operational Research*, 271(2):577–593, 2018.

Chan, T. C. Y. and Kaw, N. Inverse optimization for the recovery of constraint parameters. *European Journal of Operational Research*, 282(2):415–427, 2020.

Chan, T. C., Mahmood, R., and Zhu, I. Y. Inverse optimization: Theory and applications. *arXiv preprint arXiv:2109.03920*, 2021.

Crainic, T. G., Gendreau, M., and Gendron, B. *Network Design with Applications to Transportation and Logistics*. Springer International Publishing, Cham, 2021.

De Gooijer, J. G. and Hyndman, R. J. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006.

Donti, P., Amos, B., and Kolter, J. Z. Task-based End-to-end Model Learning in Stochastic Optimization. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Elmachtoub, A. N. and Grigas, P. Smart "Predict, then Optimize". *Management Science*, 68 (1):9–26, 2022.

Ferber, A., Wilder, B., Dilkina, B., and Tambe, M. MIPaaL: Mixed Integer Program as a Layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1504–1511, 2020.

Gendron, B., Crainic, T. G., and Frangioni, A. Multicommodity Capacitated Network Design. In Sansò, B. and Soriano, P., editors, *Telecommunications Network Planning*, pages 1–19. Springer US, Boston, MA, 1999.

Ghobadi, K. and Mahmoudzadeh, H. Inferring linear feasible regions using inverse optimization. *European Journal of Operational Research*, 290(3):829–843, 2021.

Gomory, R. E. and Hu, T. C. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

Hirsch, W. M. and Dantzig, G. B. The fixed charge problem. *Naval Research Logistics Quarterly*, 15(3):413–424, 1968.

Karush, W. *Minima of Functions of Several Variables with Inequalities as Side Conditions*. PhD thesis, University of Chicago, 1939.

Koster, A. M., Kutschka, M., and Raack, C. Robust network design: Formulations, valid inequalities, and computations. *Networks*, 61(2):128–149, 2013.

Kotary, J., Fioretto, F., Van Hentenryck, P., and Wilder, B. End-to-end constrained optimization learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4475–4482. International Joint Conferences on Artificial Intelligence Organization, 2021.

Kotary, J., Dinh, M. H., and Fioretto, F. Folded optimization for end-to-end model-based learning. *arXiv preprint arXiv:2301.12047*, 2023.

Kuhn, H. W. and Tucker, A. W. Nonlinear Programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, volume 2, pages 481–493. University of California Press, 1951.

Laage, G., Frejinger, E., and Savard, G. Periodic freight demand estimation for large-scale tactical planning. *arXiv preprint arXiv:2105.09136*, 2021.

Mandi, J. and Guns, T. Interior point solving for lp-based prediction + optimisation. *Advances in Neural Information Processing Systems*, 33:7272–7282, 2020.

Mandi, J., Stuckey, P. J., Guns, T., et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1603–1610. International Joint Conferences on Artificial Intelligence Organization, 2020.

Mulamba, M., Mandi, J., Diligenti, M., Lombardi, M., Bucarey, V., and Guns, T. Contrastive losses and solution caching for predict-and-optimize. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2833–2840, 2021.

Nandwani, Y., Ranjan, R., Singla, P., et al. A solver-free framework for scalable learning in neural ilp architectures. *Advances in Neural Information Processing Systems*, 35:7972–7986, 2022.

Paulus, A., Rolínek, M., Musil, V., Amos, B., and Martius, G. CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints. In *International Conference on Machine Learning*, pages 8443–8453. PMLR, 2021.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. Differentiation of Blackbox Combinatorial Solvers. In *International Conference on Learning Representations*, 2019.

Sadana, U., Chenreddy, A., Delage, E., Forel, A., Frejinger, E., and Vidal, T. A Survey of Contextual Optimization Methods for Decision Making under Uncertainty. *arXiv preprint arXiv:2306.10374*, 2023.

Sun, C., Liu, S., and Li, X. Maximum optimality margin: A unified approach for contextual linear programming and inverse linear programming. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 32886–32912. PMLR, 2023.

Wilder, B., Dilkina, B., and Tambe, M. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

Appendix A

Appendix

A.1 Formulation of *IO-constraint*

The constraints of the forward problem in Ghobadi and Mahmoudzadeh (2021) are expressed as inequalities of the form $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, where $\boldsymbol{x}$ is a vector of decision variables and A and b are a matrix and a vector of constraint parameters respectively. We define our solution vector $\boldsymbol{x}$ given a forward solution $(\boldsymbol{\phi}_l, x^*(\boldsymbol{d}_l), y^*(\boldsymbol{d}_l))$ as a concatenation of the three vectors of the forward solution:

$$\boldsymbol{x}_l = \begin{bmatrix} x^*(\boldsymbol{d}_l) & y^*(\boldsymbol{d}_l) & \boldsymbol{\phi}_l \end{bmatrix}^\top \in \mathbb{R}^{D_x}. \tag{A.1}$$

where the dimension of the vector $D_x = |\mathcal{K}||\mathcal{A}| + |\mathcal{A}| + m$ equals the sum of the dimension of the three sub-vectors.

The forward problem constraints are split into two groups, known constraints $\boldsymbol{G}\boldsymbol{x} \leq h$ and unknown constraints $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. We can rewrite the constraints as inequalities to form matrices $\boldsymbol{A}$ and $\boldsymbol{G}$, and vectors $\boldsymbol{b}$ and $\boldsymbol{h}$, as defined in Ghobadi and Mahmoudzadeh (2021). Because our constraints are partially known, we define a set of valid matrices $\boldsymbol{\mathcal{A}}_{\text{valid}}$ such that $\boldsymbol{A} \in \boldsymbol{\mathcal{A}}_{\text{valid}}$ in the inverse model.

We convert the unknown constraints to matrix inequality form. The set of unknown constraints is only comprised of the flow conservation constraints. There are two equality constraints for every commodity k, one for the flow on the destination node $n = \text{dest}(k)$ and a second for the origin node $n = \text{orig}(k)$. An equality constraint can be rewritten as two inequality constraints. For every commodity k, the flow conservation constraints can be written as four vectors $\boldsymbol{a}^k_{\text{orig}}, -\boldsymbol{a}^k_{\text{orig}}, \boldsymbol{a}^k_{\text{dest}}, -\boldsymbol{a}^k_{\text{dest}}$ that form rows of the $\boldsymbol{A}$ matrix.

We can construct the $\boldsymbol{a}^k_n$ vector for $n = \text{orig}(k), \text{dest}(k)$ in three parts. The first part $\text{adj}(k, n) \in \mathbb{R}^{|\mathcal{K}||\mathcal{A}|}$ is a vector describing the adjacency of flows on node n for commodity k. The component

of $\text{adj}(k,n)$ at index $(i,j,\bar{k})$ is:

$$[\text{adj}(n,k)]_{ij}^{\bar{k}} = \begin{cases} 1, & \bar{k}=k \text{ and } j=n \\ -1, & \bar{k}=k \text{ and } i=n \\ 0, & \text{otherwise.} \end{cases} \tag{A.2}$$

The second part is a vector of zeros $\mathbf{0} \in \mathbb{R}^{|\mathcal{A}|}$ and the third part is the weights of the prediction model for commodity k: $\theta^k \in \mathbb{R}^m$. We can thus write the $\boldsymbol{a}_n^k$ vector as:

$$\boldsymbol{a}_n^k = \begin{bmatrix} \text{adj}(n,k) & \mathbf{0} & \theta^k \end{bmatrix} \in \mathbb{R}^{D_x}. \tag{A.3}$$

Finally, the set of valid matrices and the $\boldsymbol{b}$ can be written:

$$\begin{aligned} \boldsymbol{\mathcal{A}}_{\text{valid}} = \Bigg\{ \begin{bmatrix} \boldsymbol{a}_{\text{orig}}^1 & -\boldsymbol{a}_{\text{orig}}^1 & \boldsymbol{a}_{\text{dest}}^1 & -\boldsymbol{a}_{\text{dest}}^1 & \cdots & \boldsymbol{a}_{\text{orig}}^{|\mathcal{K}|} & -\boldsymbol{a}_{\text{orig}}^{|\mathcal{K}|} & \boldsymbol{a}_{\text{dest}}^{|\mathcal{K}|} & -\boldsymbol{a}_{\text{dest}}^{|\mathcal{K}|} \end{bmatrix}^\top \\ : \boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^{|\mathcal{K}|} \in \mathbb{R}^m \Bigg\}. \end{aligned} \tag{A.4}$$

$$\boldsymbol{b} = \mathbf{0} \in \mathbb{R}^{4|\mathcal{K}|}. \tag{A.5}$$

The known equations can be converted into matrix and vector inequalities $\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}$ using the same process.

Example A.1. *We illustrate this process on the network design problem from example 3.1:*

$$\boldsymbol{\mathcal{A}}_{valid} = \left\{ \begin{bmatrix} 1 & 1 & 0 & 0 & -\theta_1^1 & \ldots & -\theta_m^1 \\ -1 & -1 & 0 & 0 & \theta_1^1 & \ldots & \theta_m^1 \end{bmatrix} : \theta_1^1, \ldots, \theta_m^1 \in \mathbb{R} \right\}$$

,

$$\boldsymbol{b} = \mathbf{0} \in \mathbb{R}^4$$

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & -u_p & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & -u_q & 0 & \ldots & 0 \\ -1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & -1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & -1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & -1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & 0 & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{8\times(4+m)}, \qquad \boldsymbol{h} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

We can now write the single point inverse optimization problem. We assume we are given an optimal solution $\boldsymbol{x}_l$ as well as a vector of forward problem objective coefficients:

$$\boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_{ij}^k & \boldsymbol{f}_{ij} & 0 & \dots & 0 \end{bmatrix}^\top \in \mathbb{R}^{D_x}. \tag{A.6}$$

where $\boldsymbol{c}_{ij}^k$ is the vector of flow costs and $\boldsymbol{f}_{ij}$ is the vector of design costs. The optimization problem is formulated as follows:

$$\begin{aligned}
\underset{\boldsymbol{A},\boldsymbol{b},\boldsymbol{\pi},\boldsymbol{\rho}}{\text{minimize}} \quad & \mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D}) && && \text{(A.7a)}\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{x}_l \geq \boldsymbol{b}, && && \text{(A.7b)}\\
& \boldsymbol{c}^\top \boldsymbol{x}_l = \boldsymbol{b}^\top \boldsymbol{\pi} + \boldsymbol{h}^\top \boldsymbol{\rho}, && && \text{(A.7c)}\\
& \boldsymbol{A}^\top \boldsymbol{\pi} + \boldsymbol{G}^\top \boldsymbol{\rho} = \boldsymbol{c}, && && \text{(A.7d)}\\
& \|\boldsymbol{a}_i\| = 1, && i = 1,\dots,4, && \text{(A.7e)}\\
& \boldsymbol{\pi} \in \mathbb{R}^4, && && \text{(A.7f)}\\
& \boldsymbol{\rho} \in \mathbb{R}^3, && && \text{(A.7g)}\\
& \boldsymbol{A} \in \mathcal{A}_{\text{valid}}, && && \text{(A.7h)}\\
& \boldsymbol{b} = \boldsymbol{0}. && && \text{(A.7i)}
\end{aligned}$$

From this, we can formulate the multiple point inverse optimization problem. This problem is bi-linear, and Ghobadi and Mahmoudzadeh (2021) propose a subsequent tractable version. We suppose we have a set of L feasible solutions $\{\boldsymbol{x}_l\}_{l=1}^L$. We define the preferred solution

$$\boldsymbol{x}^* \in \arg \min_{\boldsymbol{x}_l : l=1,\dots,L} \boldsymbol{c}^\top \boldsymbol{x}_l$$

. This allows us to express the multiple-point inverse optimization problem (MIO-EF).

MIO

$$\begin{aligned}
\underset{\boldsymbol{A},\boldsymbol{b},\boldsymbol{\pi},\boldsymbol{\rho}}{\text{minimize}} \quad & \mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D}) && && \text{(A.8a)}\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{x}_l \geq \boldsymbol{b}, && \forall\, l = 1 \dots L, && \text{(A.8b)}\\
& \boldsymbol{c}^\top \boldsymbol{x}^* = \boldsymbol{b}^\top \boldsymbol{\pi} + \boldsymbol{h}^\top \boldsymbol{\rho}, && && \text{(A.8c)}\\
& \boldsymbol{A}^\top \boldsymbol{\pi} + \boldsymbol{G}^\top \boldsymbol{\rho} = \boldsymbol{c}, && && \text{(A.8d)}\\
& \|\boldsymbol{a}_i\| = 1, && \forall\, i = 1, \dots, 4|\mathcal{K}|, && \text{(A.8e)}\\
& \boldsymbol{\pi} \in \mathbb{R}^4, && && \text{(A.8f)}\\
& \boldsymbol{\rho} \in \mathbb{R}^3, && && \text{(A.8g)}\\
& \boldsymbol{A} \in \mathcal{A}_{\text{valid}}, && && \text{(A.8h)}\\
& \boldsymbol{b} = \boldsymbol{0}. && && \text{(A.8i)}
\end{aligned}$$

We notice that Constraints (A.8c) have bilinear terms and thus MIO is not computationally tractable. To remedy this, Ghobadi and Mahmoudzadeh (2021) propose reformulating MIO as a tractable linear program. For this reformulation to be valid, the authors assume that the preferred solution $\boldsymbol{x}^*$ is optimal in the forward problem. In other words, they assume that the set of known constraints of the forward problem is:

$$\mathcal{S} = \{\boldsymbol{x} \in \mathbb{R}^m \mid \boldsymbol{c}^\top \boldsymbol{x} \geq \boldsymbol{c}^\top \bar{x}^*, \boldsymbol{G}\boldsymbol{x} \geq \boldsymbol{h}\}.$$

Using this assumption, we can finally express a tractable formulation of our inverse problem. This model assumes that the input points are generated from a linear model without noise:

eMIO

$$\begin{aligned}
\underset{\boldsymbol{A},\boldsymbol{b}}{\text{minimize}} \quad & \mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D}) && && \text{(A.9a)}\\
\text{subject to} \quad & \boldsymbol{a}_i^\top \boldsymbol{x}_l \geq \boldsymbol{b}, && \forall\, l = 1, \dots, L,\ k = 1 \dots 4, && \text{(A.9b)}\\
& \|\boldsymbol{a}_i\| = 1, && \forall\, i = 1, \dots, 4|\mathcal{K}|, && \text{(A.9c)}\\
& \boldsymbol{A} \in \mathcal{A}_{\text{valid}}, && && \text{(A.9d)}\\
& \boldsymbol{b} = \boldsymbol{0}. && && \text{(A.9e)}
\end{aligned}$$

We obtain problem 3.15 by reformulating eMIO into regular constraint notation and adding the adjacency measure as a loss function $\mathcal{F}(\boldsymbol{A},\boldsymbol{b},\mathcal{D})$.