Learning Linear Programs: Inverse Optimization as a Form of Machine Learning

Yingcong Tan

A Thesis
In the Department
of
Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Industrial Engineering) at
Concordia University
Montréal, Québec, Canada

March 2021

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Yingcong Tan**

Entitled: **Learning Linear Programs: Inverse Optimization as a Form of Machine Learning**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Industrial Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Dr. Liangzhu Wang	Chair
Dr. Emma Frejinger	External Examiner
Dr. Eugene Belilovsky	External to Program
Dr. Ivan Contreras	Examiner
Dr. Masoumeh Kazemi Zanjani	Examiner
Dr. Andrew Delong	Co-supervisor
Dr. Daria Terekhov	Co-supervisor

Approved ______________________
Dr. Ivan Contreras, Graduate Program Director

__________ 20 ______ ______________________

Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Learning Linear Programs: Inverse Optimization as a Form of Machine Learning

Yingcong Tan, Ph.D.

Concordia University, 2021

Conventionally, in an optimization problem, we aim to determine the values of the decision variables to minimize the objective function subject to the constraints. We refer to this problem as the forward optimization problem (FOP). In an inverse optimization (IO) problem, the goal is to determine the coefficients of an FOP such that an observed solution becomes an optimal solution of the learned FOP model. In this dissertation, we focus on the *inverse linear optimization* problem whose FOP has the form of linear programming. We adopt an interdisciplinary approach, leveraging concepts and methods from machine learning to address a very general form of this problem.

Firstly, we study the general form of the inverse linear optimization problem, that is, learning all model coefficients individually or jointly where the unknown model coefficients may or may not depend on exogenous parameters. We are the first to cast the IO problem as a form of deep learning and solve it with a gradient-based algorithm. To compute the gradients, we differentiate through the steps of an optimization process, in particular, the Barrier interior-point method. We develop new sets of benchmark instances and show good performance of our algorithm on different IO tasks: 1). learning a cost vector of linear program; 2). learning cost vector and constraints of a linear program jointly; 3). learning unknown parameters in the objective and constraints of a parametric linear program. To the best of our knowledge, this algorithm is the first IO approach in the literature to be able to handle all three types of tasks.

Secondly, we formulate the inverse linear optimization problem as a bilevel optimization problem and explicitly encode constraints in the outer problem to ensure that observed solutions remain feasible with respect to the constraints. Again by leveraging a machine learning perspective on inverse linear optimization, we develop a general-purpose framework to solve the bilevel model with gradient-based algorithms. We investigate different methods for differentiating through an optimization problem and specialize them to LP. Additionally, we focus on an objective-based loss function and derive a closed-form expression for computing gradients. Experimental results show that our framework is capable of solving synthetic parametric linear program and multi-commodity flow problem instances which could not be previously solved by methods in the IO literature. Additionally, we show that our closed-form expression is orders of magnitude faster than other methods for computing the gradients.

Finally, we focus on a special case of learning the objective only. We present four different methods for solving this problem, including three mathematical formulations and one general gradient-based algorithm. We test all four methods on synthetic parametric linear program and multi-commodity flow problem instances, and show that all four methods can successfully solve all experimental instances. All three mathematical models are solved in a commercial optimization solver and, due to their specialized nature, outperform the more generic gradient-based algorithm in runtime. Additionally, we show that the parametric linear programs learned from the KKT-based and strong duality-based formulations produce the best predictions on testing data.

In summary, the main contribution of this dissertation is the development of a general gradient-based framework which is able to solve problems that were previously not tackled in the IO literature. In addition, we extend and unify models for the special case of learning the objective only, which has been the focus of previous IO work. This dissertation unifies machine learning and inverse optimization both at the modelling and algorithmic levels.

Acknowledgments

First of all, I would like to thank my supervisors, Dr. Daria Terekhov and Dr. Andrew Delong. Without their support and encouragement, I could not complete the program. Daria has always been passionate about my research and supportive of my career. She offered numerous opportunities for me to advance my research and career. In the first year of this program, I developed a strong interest in Machine Learning topics and later decided to focus on interdisciplinary research topics. To help advance my thesis, Daria initiated a collaboration project with Andrew who later joined Concordia University and became my co-supervisor. Their dedication to work and detail-oriented work ethic have deeply influenced me and my approach to work.

Most importantly, I want to thank them for their incredible mentorship. During every meeting, Daria and Andrew would patiently discuss every little detail in my research and clarify any possible confusion. Their selflessness with time and patient was critical to the work of this thesis and my personal growth.

Besides my supervisors, I would like to thank all examining committee members, Dr. Ivan Contreras, Dr. Masoumeh Kazemi-Zanjani, Dr. Emma Emma Frejinger, Dr. Eugene Belilovsky, Dr. Brigitte Jaumard, Dr. Jia Yuan Yu, for their constructive advice. In particular, I would like to thank Dr. Ivan Contreras. I still remember the time I started my M.Eng study with minimal knowledge of operations research. I took two courses with Dr. Ivan Contreras, who introduced me to the world of operations research and inspired me to switch to a research-based graduate program.

I want to thank my colleagues and friends at Concordia University, Mohammad Hasan Aghdaie, Nazanin Aslani, Elaheh Hosseiniiraj, Chelsey Hvingelby, Mahsa Moghaddass, Gerald Potkah and Saif Rahman for the conversation and experiences we shared both in and out of the lab. Outside of the lab, I made many friends at MIAE graduate student committee and TORCH. I want to thank Giuseppe, Tina, Carlos, Gabriel, Mohamed for all the fun we had in the last four years.

I want to thank my parents. I always look up to my parents as role models. They taught me the importance of hard work and humbleness. Without them, I would not be where I am today. For my wife, Mengjie, no words can describe how grateful I am to you for the sacrifices you made, for all the love and happiness you have brought to my life. Without your support, none of this would have been possible.

I would also like to thank Concordia University and my supervisors for their financial support. Finally, to everyone whom I did not mention by name but that contributed toward this work, thank you.

This thesis is dedicated to the memory of my father.

I hope I have made you proud.

I miss you.

Contributions of Authors

This dissertation is presented under the *manuscript-based* format. It contains two articles that have been published and one in preparation. More specifically, three manuscripts are listed below in a chronological order. The first article "Deep Inverse Optimization" was published in the proceedings of *the Sixteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (CPAIOR) in 2019. The second article "Learning Linear Programs from Optimal Decisions" was published in the proceedings of *the Thirty-fourth Conference on Neural Information Processing Systems* (NeurIPS) in 2020. The last article "Learning the Objective of Linear Programs: Models and Insights" is in preparation and will be submitted to an operations research journal, potentially the *European Journal of Operational Research*.

All three manuscripts are co-authored with Dr. Daria Terekhov and Dr. Andrew Delong, who established research guidelines and reviewed the papers before submission. The author of this thesis acted as the principal researcher with the corresponding duties such as the development of formulations and algorithms, the programming of solution methods and the analysis of computational results along with writing drafts of the papers.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimization is deeply embedded in various decision-making processes of modern organizations (e.g., budget planning, staff scheduling, inventory management, etc.) and an individual's life (e.g., routing, shopping, diet management). Optimization models are widely used to represent decision-making processes by formally characterizing the objective (e.g., cost and achievable goals), constraints (e.g., restrictions on resources) and the associated decision variables (e.g., choice of actions). For that reason, optimization models such as linear programs (LPs) continue to receive a great amount of attention from research and industry.

Recently, the availability of big data has promoted the research and application of data-driven decision-making, which emphasizes using real data to guide the decision-making process. This requires one to develop an optimization model based on the collection of real data, and the developed model should be a useful representation of reality. Inverse optimization (IO) aims to learn unknown coefficients of optimization models from observed optimal solutions. As a data-driven methodology for inferring model coefficients, IO is receiving growing attention.

This dissertation's central thesis is that viewing IO as a learning problem with bilevel structure facilitates the development of a general gradient-based framework for solving inverse linear programs (i.e., learning linear programs from optimal decisions) where the unknown model coefficients may or may not depend on exogenous parameters. As such, this dissertation advances the methodology of developing data-driven optimization models through an integration of ideas from machine learning and inverse optimization. The gradient-based framework proposed in this dissertation provides the first, to the best of our knowledge, general approach able to learn all coefficients of a linear program from data.

1.1 Motivation

The conventional workflow of optimization model development requires not only excellent knowledge of the modelling techniques but also an in-depth understanding of the underlying decision-making process. This workflow commonly demands a high level of human intervention.

Firstly, the model developers are expected to incorporate the knowledge of the decision-making process into the design of the objective, constraints and decision variables of a mathematical programming (MP) model. Secondly, raw data are collected and used to specify the model coefficients with suitable data analysis methods. Lastly, the initially developed model is implemented and tested to collect feedback and possibly additional information from other domain experts (e.g., stakeholders). When noticing any discrepancy between the developed model and other collected information (e.g., historical data, feedback and suggestions from stakeholders), the MP model has to be revisited and updated by the model developer. We refer to this step as *model refinement*. Model refinement can be done in different ways: adding/removing/updating additional constraints, revisiting the second step to use a different data analysis method, or manually modifying the model coefficients. All these approaches are time consuming and largely depend on the knowledge and expertise of the model developer. This cycle of model development and refinement is expected to be repeated until a satisfactory MP model is accepted and delivered for real-life use.

Model refinement heavily depends on the model developers' expertise in modelling techniques. In this conventional workflow, there is no formal methodology to directly incorporate the knowledge from other domain experts (e.g., decision maker, shareholders) into specifying the model coefficients such that the developed model would produce solutions that are favourable for the decision makers. One methodology that can help resolve this issue is inverse optimization, which can be used to automate the model coefficient estimation process, especially where the model developer manually modifies the model coefficients in the model refinement step. Most importantly, incorporating IO into the conventional workflow described above does not conflict with other procedures involved or additional knowledge from other domain experts.

Additionally, the conventional workflow of model development largely depends on raw data availability for specifying the model coefficients of the MP models. In many real-world applications, one may not have access to the decision-maker or be able to get feedback. This would severely limit the model coefficient estimation and the accuracy of the corresponding developed model. However, we often can observe some optimal (or close-to-optimal) solutions (e.g., historical data representing the decision-maker's behaviour), which allows IO to infer the corresponding model coefficients directly from the observed solutions. Consider a real-life routing problem, users (e.g., driver) of a given network of roads may optimize their path from an origin to a destination based on some criteria. These criteria could be road-specific (e.g., speed limit, length) or user-specific (e.g., vehicle, preference on toll roads). These criteria would reflect the user's objective function that dictates his/her decisions, and thus it is important to collect information from the user when specifying the coefficients in the objective function. In practice, we do not always have access to an individual driver but we may be able to collect data on some paths used by the driver and then use IO to infer the objective function coefficients directly from the observed path.

As discussed above, we believe that, as a data-driven methodology for inferring model coefficients, inverse optimization (IO) can be embedded in the conventional workflow of optimization model development to automate the data analysis procedures. Using IO can facilitate a novel workflow that is more automated and will be more akin to model fitting done in machine learning.

1.2 Dissertation Overview

This dissertation studies the most general form of the *inverse linear optimization problem* where all model coefficients of a linear program depend on other parameters (i.e., *parametric linear program*), and the goal is to learn the parameter. We refer the readers to Chapter 2 for more details of non-parametric and parametric linear programs.

This dissertation is presented in a manuscript-based format, and it consists of six chapters. Chapter 1 presents the introduction, outline and contributions of this dissertation. Chapter 2 provides some background information and literature review. Three manuscripts are presented in Chapters 3, 4 and 5. Each of the manuscript chapters defines its own notation and has its own literature review. The manuscript in Chapter 3 was published in the 16^{th} International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR'19). The manuscript in Chapter 4 was published in the 34^{th} Conference on Neural Information Processing Systems (NeurIPS'20). The manuscript in Chapter 5 will be submitted to an operations research journal, potentially the European Journal of Operational Research. Lastly, Chapter 6 presents future work directions and concludes this dissertation.

In Chapter 3, we first propose to view IO as a learning problem and present connections between IO and deep learning. IO can be viewed as a learning problem since it aims to learn a parametric optimization problem whose optimal solution (which is referred to as the *prediction*) is consistent with the observed solution (which is referred to as the *target*). Similar to deep learning models, the goal of IO is to minimize the loss between the target and prediction. Unlike deep learning models whose prediction is the result of a feedforward neural network, the prediction in the IO is the result of an optimization problem. This perspective motivates the first manuscript in this dissertation. We replace the forward pass of the neural network in the deep learning architecture with the Barrier interior point method (IPM), which produces an optimal solution of a linear program, and then computes the loss accordingly. Like the forward pass of a neural network, the Barrier IPM is an iterative and differentiable procedure, and thus it is compatible with the automatic differentiation functions (i.e., differentiating a nested function using the chain rule) in deep learning packages such as PyTorch. In summary, we cast the IO problem as a form of deep learning. We develop a gradient-based algorithm called *deep inverse optimization* (DIO) which unrolls the optimization process to compute the gradients for updating the parameters. As proof of concept, we implement the Barrier IPM in PyTorch (Paszke et al., 2019). We construct new sets of IO benchmark instances and show that our proposed algorithm successfully solves three IO problems: 1). learn the objective of a non-parametric LP, 2). learn the objective and constraints of a non-parametric LP jointly, and 3). learn parameters of a parametric LP.

In Chapter 4, we investigate different methods to differentiate through a linear program for computing the gradients of LP coefficients. Although the problem of differentiating through an optimization problem has been studied in some deep learning literature (e.g., Amos and Kolter (2017) studies the quadratic programming problem, Agrawal et al. (2019) studies the convex optimization problem), we find no similar work focused on LP. Unlike many deep learning methods, we investigate an objective-based loss that penalizes the difference between the target and predictions in their objective values and derive a novel approach to compute the gradients through a closed-form expression. We formulate the inverse linear optimization problem as a bilevel optimization model. The inner-problem encodes the linear program whose optimal solution is the prediction. The outer-problem minimizes the loss between the target and prediction, and it explicitly encodes constraints to ensure the target is feasible with respect to the learned constraints. We demonstrate good performance of the proposed gradient-based framework on the synthetic PLP and minimum-cost multi-commodity flow problem instances. Additionally, we demonstrate the performance superiority (in runtime) of our closed-form approach in computing the gradients over a implicit differentiation approach and a backpropagation approach.

While in Chapters 3 and 4 we study the problem of learning all model coefficients from optimal solutions, in Chapter 5 we focus on the special case of learning the objective function only. To our knowledge, most of the existing work studies this problem under the setting of non-parametric LP, and there is no previous work discussing the connections among existing methods. We generalize the mathematical models from Chan et al. (2019) and Tavaslıoğlu et al. (2018) to the general parametric case with multiple observations and propose two new mathematical models accordingly. We also generalize the concept of *inverse-feasible-region* (i.e., collection of all cost vectors that make the observed solution optimal) from Tavaslıoğlu et al. (2018) to the parametric setting. We prove that our new mathematical models are mathematically equivalent to the model from Keshavarz et al. (2011) when the inverse-feasible-region is not empty. All formulations are coded in CPLEX (IBM, 2020) and then tested on synthetic parametric LP and multi-commodity flow problem instances. We compare their performance with the gradient-based framework (developed in Chapter 4) and show the performance advantage (in runtime) of the mathematical formulations solved in CPLEX.

1.3 Contributions

We focus on the IO problem of learning the model coefficients of linear programs. This problem is referred to as the inverse linear optimization problem in this dissertation. The main contributions of the three manuscripts included in this dissertation are provided below.

- Chapter 3: Deep Inverse Optimization (published in the Proceedings of CPAIOR'19)
    - We cast the IO problem as a form of deep learning. It allows us to solve the IO problem with gradient-based algorithms. Notably, we compute the gradients by backpropagating through all the steps of an optimization process.
    - To our knowledge, we propose the first method for solving the most general form of the inverse linear optimization problem, that is, learning the objective and constraints of a parametric linear program jointly or separately given multiple observations.
    - To understand the behaviour of the proposed algorithm, we establish new benchmark instances that categorize the inverse linear optimization problems based on the ratio of the number of variables to the number of constraints.
    - We demonstrate that the proposed methods can efficiently learn the objective and constraints of a non-parametric or parametric LP separately or jointly.

- Chapter 4: Learning Linear Programs from Optimal Decisions (published in the proceedings of NeurIPS'20)
  - To extend the work in Chapter 3, we formulate the inverse linear optimization problem as a novel bi-level optimization model. The outer problem minimizes a discrepancy between the observed solution and the predictions. The prediction is an optimal solution of a parametric linear program which is defined in the inner problem.
  - We explicitly encode constraints in the outer problem to ensure that the observed solutions remain feasible in any learned model. We refer to this set of constraints as the target feasibility constraints. To our knowledge, we are the first to propose this type of constraint in formulating an IO problem.
  - We solve the bi-level formulation with gradient-based algorithms and differentiate through an LP to compute the gradients. We specialize methods from the literature to the case of linear programming. In addition, we focus on the objective-error type of loss function, and derive a new closed-form expression for computing the gradients.
  - We implement four variants of our gradient-based algorithm each with a different gradient methods: *implicit* (i.e., implicit differentiation), *backprop* (i.e., backpropagation), *direct* (i.e., closed-form expression) and *cvx* (i.e., using a python library called cvxpy).
  - We demonstrate strong performance of our gradient-based algorithms on synthetic parametric linear programs and parametric multi-commodity flow instances. In particular, the gradient algorithms with *implicit*, *backprop* and *direct* solve 60-65% of the instances, gradient algorithm with *cvx*

solves $\sim 50\%$ of the instances. In contrast, the gradient-free baselines are successful only 25% of the time.

  - We demonstrate that our closed-form expression is multiple orders of magnitude faster than other approaches in computing the gradients.
  - We illustrate the challenge of generalization that occurs in IO. We show via an example that the training data can under-specify the learned PLP and achieve zero training loss while not being able to match target solutions on the test set.

- Chapter 5: Learning the Objective of Linear Programs: Models and Insights (will be submitted to an operations research journal).
  - We study a special case of inverse linear optimization with the goal of learning the objective under the parametric setting, which has not previously been addressed in the IO literature.
  - We propose four different mathematical formulations, including three single-level optimization models which are extensions of existing IO literature, and one bilevel optimization model which is directly based on Chapter 4.
    * We specialize the method from Keshavarz et al. (2011) to LP and propose a single-level optimization model using the KKT conditions. We refer to this model as the KKT-based formulation.
    * We generalize the method from Chan et al. (2019) to parametric case and propose a single-level optimization model using the strong duality. We refer to this model as the strong duality-based formulation.
    * We generalize the method from Tavaslıoğlu et al. (2018) to parametric case and propose a single-level optimization model using the concept of inverse feasible region. We refer to this model as the inverse feasible

region-based formulation.

    * We specialize the bilevel optimization model and gradient-based algorithm from Chapter 4 to learning the objective only.

- We generalize the concept of inverse feasible region from Tavaslıoğlu et al. (2018) to the general case of parametric LP, and define the *parametric inverse feasible region*.

- We prove that the three single-level optimization models are mathematically equivalent under the assumption of a non-empty *parametric inverse feasible region*.

- Computational results show that all three single-level optimization models solved in CPLEX are faster than the bilevel optimization model solved with a gradient algorithm. Additionally, the inverse feasible region-based model has the shortest runtime.

- We show that PLP models learned from the KKT-based and strong duality-based formulations provide better predictions on testing data. Combining with the runtime comparison, we highlight the performance trade-off between the runtime and accuracy of the predictions.

Chapter 2

Inverse Linear Optimization: Background and Literature Review

Linear programming is a class of optimization problems that deals with linear objective and constraints and continuous decision variables. It is widely implemented in various fields (transportation, production, etc.). It is unarguably important in both theory and applications. Consequently, the problem of learning linear programs, i.e., *inverse linear optimization* is receiving growing attention. It has been successfully implemented in various applications, such as radiation therapy planning (Chan et al., 2019; Chan and Kaw, 2020), production planning (Chan et al., 2019; Troutt et al., 2006), traffic network (Burton and Toint, 1992), investment portfolio optimization (Bertsimas et al., 2012) and electricity demand forecasting (Saez-Gallego and Morales, 2017).

2.1 Background

Conventionally, in an optimization problem, we aim to determine the values of the decision variables to minimize the objective function and subject to the constraints. We refer to this problem as the forward optimization problem (FOP). In an inverse optimization (IO) problem, the goal is to determine the coefficients of a FOP such that an observed solution becomes an optimal solution of the learned FOP model. In this dissertation, we focus on the *inverse linear optimization* problem whose corresponding FOP has the form of LP.

For the rest of this section, we provide some background information on linear programming, its generalization under parametric setting and inverse linear optimization. Mathematical formulations, numerical examples and figures are provided. Lastly, we discuss some related work and research gaps that motivate this dissertation.

2.1.1 Linear Programming and Parametric Linear Programming

In linear program (LP), the cost vector $\mathbf{c} \in \mathbb{R}^D$, inequality constraint coefficients $\mathbf{A} \in \mathbb{R}^{M_1 \times D}$, $\mathbf{b} \in \mathbb{R}^{M_1}$, and equality constraint coefficients $\mathbf{G} \in \mathbb{R}^{M_2 \times D}$, $\mathbf{h} \in \mathbb{R}^{M_2}$ are given and held fixed. The goal is to determine a solution $\mathbf{x} \in \mathbb{R}^D$ which satisfies the constraints and minimizes an objective function.

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{G}\mathbf{x} = \mathbf{h} \end{aligned} \tag{LP}$$

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 0.5x_1 + 1.5x_2 \\ \text{subject to} \quad & -5 \leq x_1 \leq 5 \\ & -5 \leq x_2 \leq 5 \\ & -3x_1 + 2x_2 \leq 5 \end{aligned}$$

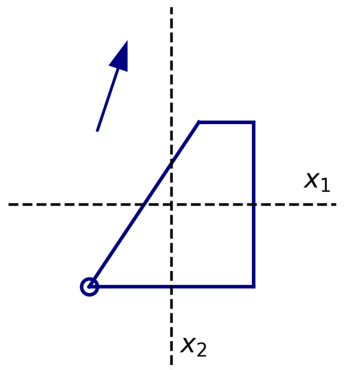


Figure 2.1: An example of an LP. The mathematical model is shown on the left and the corresponding feasible region is shown on the right. The solid lines, arrow, and circle represent the feasible region, cost vector, and corresponding optimal solution respectively, and the dashed lines represent the horizontal and vertical axes.

Figure 2.1 presents an example of linear program. It is also referred to as the non-parametric LP in this dissertation, since model coefficients $\mathbf{c}$, $\mathbf{A}$ and $\mathbf{b}$ do not depend on other parameters.

Parametric Linear Program (PLP) In *parametric linear program*, the model coefficients (and therefore the corresponding optimal solutions) depend on some features $\mathbf{u}$ (e.g., time or weather conditions).

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}(\mathbf{u})^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}(\mathbf{u})\mathbf{x} \leq \mathbf{b}(\mathbf{u}) \\ & \mathbf{G}(\mathbf{u})\mathbf{x} = \mathbf{h}(\mathbf{u}) \end{aligned}$$

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & (u - 0.5)x_1 + (u + 0.5)x_2 \\
\text{subject to} \quad & x_1 \leq 4 + u \\
& x_2 \leq 4 + u \\
& -x_1 \leq 4 + u \\
& -x_2 \leq 4 + u \\
& -(u+2)x_1 + 2x_2 \leq 4 + u
\end{aligned}
$$

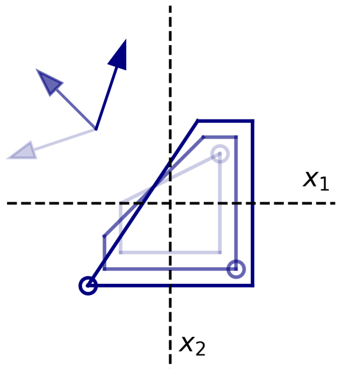


Figure 2.2: An example of PLP(u), where $u_1 = 1$, $u_2 = 0$, $u_3 = -1$ denote the features u under different observations. The mathematical model is shown on the left and the corresponding feasible regions are shown on the right.

Figure 2.2 shows a parametric linear program instance with three u values. Note that the LP instance in Figure 2.1 is a special case of the PLP instance in Figure 2.2 (i.e., PLP($u = 1$)).

Model Hypothesis Our perspective is similar to the concept of hypothesis space used in ML literature. Hypothesis space consists of the set of functions the model is limited to learn. For instance, depending on the parameterizaiton, a regression model can be limited to linear functions as its hypothesis space, or it can be expanded to polynomials. Motivated by this perspective, to learn a PLP model from data, in this dissertation we introduce the notion that one needs to define a suitable hypothesis space parametrized by $\mathbf{w}$, that is, PLP($\mathbf{u}, \mathbf{w}$):

$$
\begin{aligned}
\text{minimize} \quad & \mathbf{c}(\mathbf{u}, \mathbf{w})^T \mathbf{x} \\
\text{subject to} \quad & \mathbf{A}(\mathbf{u}, \mathbf{w})\mathbf{x} \leq \mathbf{b}(\mathbf{u}, \mathbf{w}) \\
& \mathbf{G}(\mathbf{u}, \mathbf{w})\mathbf{x} = \mathbf{h}(\mathbf{u}, \mathbf{w})
\end{aligned}
$$

.

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & (wu - 0.5)x_1 + (wu + 0.5)x_2 \\
\text{subject to} \quad & x_1 \leq 4 + wu \\
& x_2 \leq 4 + wu \\
& -x_1 \leq 4 + wu \\
& -x_2 \leq 4 + wu \\
& -(w * u + 2)x_1 + 2x_2 \leq 4 + w * u
\end{aligned}
$$

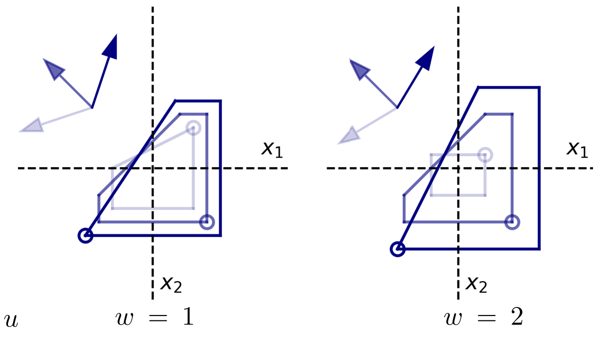


Figure 2.3: An example of PLP(u, w) where w characterizes the hypothesis space and $u_1 = 1$, $u_2 = 0$, $u_3 = -1$ denote the feature u under different observation. The mathematical model is shown on the left and the corresponding graph is shown on the right.

Figure 2.3 shows an example of PLP that is parametrized by w. Note that the PLP instance in Figure 2.2 is equivalent to PLP instance on the left-hand-side of Figure 2.3 (i.e., PLP$(w = 1)$).

2.1.2 Inverse Optimization

To help understanding the IO problem, we present two classic formulations from the literature (Ahuja and Orlin, 2001; Keshavarz et al., 2011). For consistency, we present the formulations using the notation from the original papers.

Firstly, we present the formulation developed by Ahuja and Orlin (2001). They study the IO problem with the goal of determining the cost vector of a non-parametric LP from a *single noise-free* observation. A noise-free observation means that the observed solution is a candidate optimal solution. In an LP, this means the observed solution lies on the boundary of the feasible region.

Let $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{c} \in \mathbb{R}^D$ denote the decision variables and cost vector of the primal LP and $\mathbf{A} \in \mathbb{R}^{M \times D}$, $\mathbf{b} \in \mathbb{R}^M$ denote the constraint coefficients of the primal LP model.

Let $\boldsymbol{\lambda} \in \mathbb{R}^M$ denotes the corresponding dual variables. We first present the primal and dual LP models used in Ahuja and Orlin (2001).

$$\begin{aligned}
\text{(Primal)} \quad \text{minimize} \quad & \sum_{j=1}^{D} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{D} a_{ij} x_j \;\geq\; b_i,\; i = 1,\, \ldots,\, M \\
& x_j \;\in\; \mathbb{R},\; j = 1,\, \ldots,\, D
\end{aligned}$$

$$\begin{aligned}
\text{(Dual)} \quad \text{minimize} \quad & \sum_{i=1}^{M} b_i \lambda_i \\
\text{subject to} \quad & \sum_{i=1}^{M} a_{ij} \lambda_i \;=\; c_j,\; j = 1,\, \ldots,\, D \\
& \lambda_i \;\geq\; 0,\; i = 1,\, \ldots,\, M
\end{aligned}$$

Let $\mathbf{x}^{\text{obs}} \in \mathbb{R}^D$ denote an observed solution. Let $\mathcal{I} = \{1,\, \ldots,\, M\}$ denote the index set of constraints. and $\mathcal{I}' = \{\, i \in \mathcal{I} \mid \sum_{j=1}^{D} a_{ij} x_i^{\text{obs}} \;=\; b_i \,\}$ denote the index set of active inequality constraints. Additionally, $\mathbf{c}^0 \in \mathbb{R}^D$ denotes the target cost vector, and the goal is to find a cost vector $\mathbf{c}$ with minimum perturbation from the target cost vector that makes the observed solution optimal. We present the IO formulation from Ahuja and Orlin (2001) below.

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{D} |c_j - c_j^0| \\
\text{subject to} \quad & c_j \;=\; \sum_{i \in \mathcal{I}'} a_{ij} \lambda_i,\; j = 1,\, \ldots,\, D \\
& \lambda_i \geq 0,\; i \in \mathcal{I}'
\end{aligned} \tag{2.1}$$

The objective is to minimize the distance between the learned cost vector and the target cost vector. The constraint is obtained by combining the complementary slackness (i.e., $\lambda_i(\sum_{j=1}^{D} a_{ij}x_j^{\text{obs}} - b_i) = 0,\ i = 1, \ldots, M$) conditions and dual feasibility constraints (i.e., $\sum_{i=1}^{M} a_{ij}\lambda_i \ = \ c_j,\ j = 1,\ \ldots,\ D$) in the Dual model. This formulation is developed by noticing that if an observed solution is optimal, the complementary slackness conditions must be satisfied.

Secondly, we present the formulation from Keshavarz et al. (2011). They study the IO problem with the goal of imputing the objective function of a parametric convex optimization problem (COP) from multiple *noisy* observations. Parametric problem means that the model coefficients depend on other parameters. An observed solution is called "noisy" if it is not candidate optimal solution. That is, the observed solution lies strictly inside or outside the feasible region.

Let $\mathbf{x} \in \mathbb{R}^D$ denote the decision variable, and $\mathbf{u}$ denote a set of features that objective and constraints depend on. $f = \sum_{i=1}^{K} w_i f_i(\mathbf{x}, \mathbf{u})$ represents the parametric objective function where $\mathbf{w} \in \mathbb{R}^K$ denotes the unknown parameters that we want to learn and f_i are pre-selected basis functions. Let $g_j(\mathbf{x}, \mathbf{u}),\ j \ = \ 1,\ \ldots,\ M_1$ denote the inequality constraint functions, and $\mathbf{G} \ \in \ \mathbb{R}^{M_2 \times D},\ \mathbf{h} \ \in \ \mathbb{R}^{M_2}$ denote the coefficients of the equality constraints. We first present the parametric COP studied in Keshavarz et al. (2011).

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{K} w_i f_i(\mathbf{x}, \mathbf{u}) \\ \text{subject to} \quad & g_j(\mathbf{x}, \mathbf{u}) \ \leq \ 0,\ j \ = \ 1,\ \ldots,\ M \\ & \mathbf{G}(\mathbf{u})\mathbf{x} \ = \ \mathbf{h}(\mathbf{u}) \end{aligned} \tag{2.2}$$

Let $\boldsymbol{\lambda} \in \mathbb{R}^{M_1}$ and $\boldsymbol{\nu} \in \mathbb{R}^{M_2}$ denote the dual variables corresponds to the inequality and equality constraints respectively. Accordingly, we define the residuals of the

Karush–Kuhn–Tucker (KKT) conditions as below.

$$
\begin{aligned}
\mathbf{r}^{\text{ineq}} &= \left[(g_j(\mathbf{x},\mathbf{u}))_+ \right]_{j=1}^{M_1} \\
\mathbf{r}^{\text{eq}} &= \mathbf{G}(\mathbf{u})\mathbf{x} \; - \; \mathbf{h}(\mathbf{u}) \\
\mathbf{r}^{\text{stat}}(\mathbf{w},\boldsymbol{\lambda},\boldsymbol{\nu}) &= \nabla f(\mathbf{x},\mathbf{u}) \; + \; \sum_{j=1}^{M_1} \lambda_j \nabla g_j(\mathbf{x},\mathbf{u}) \; + \; \mathbf{G}(\mathbf{u})^T \boldsymbol{\nu} \\
\mathbf{r}^{\text{comp}} &= \left[\lambda_j g_j(\mathbf{x},\mathbf{u}) \right]_{j=1}^{M_1}
\end{aligned}
\tag{KKT Residuals}
$$

For each pair of observation $(\mathbf{u}_k, \mathbf{x}_k^{\text{obs}})$, we introduce the corresponding dual variables $\boldsymbol{\lambda}_k$ and $\boldsymbol{\nu}_k$. Let $\mathbf{r}_k^{\text{stat}}$, $\mathbf{r}_k^{\text{comp}}$, $\mathbf{r}_k^{\text{ineq}}$, $\mathbf{r}_k^{\text{eq}}$ denote the KKT residuals correspond to each observation k. We present the IO formulation from Keshavarz et al. (2011) below.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{k=1}^{N} \phi(\, \mathbf{r}_k^{\text{stat}}, \, \mathbf{r}_k^{\text{comp}} \,) \\
\text{subject to} \quad & \boldsymbol{\lambda}_k \geq 0, \; k = 1, \ldots, N
\end{aligned}
\tag{2.3}
$$

Note, the objective is to minimize a convex penalty function, which satisfies $\phi(\mathbf{r}^{\text{stat}}, \, \mathbf{r}^{\text{comp}}) = 0 \iff \mathbf{r}^{\text{stat}} = \mathbf{0}, \, \mathbf{r}^{\text{comp}} = \mathbf{0}$. The only constraint represents the dual feasibility. This formulation is developed by noticing that, if a observed solution is optimal, the corresponding KKT conditions must be satisfied.

Here we present two classic mathematical formulations for inverse optimization problems. Both formulations aim to learn the objective. In this dissertation, we focus on developing a general method to learn both objective and constraints. Secondly, both Ahuja and Orlin (2001) and Keshavarz et al. (2011) make some assumptions that would make their formulations less accessible and flexible in real-world applications. In particular, Ahuja and Orlin (2001) assumes prior knowledge of $\mathbf{c}$ and Keshavarz et al. (2011) assumes an affine parameterization in the objective.

2.2 Literature Review

IO problems studied include inverse integer problems, inverse multi-objective optimization. As mentioned above, we focus on inverse linear programs in this dissertation, and thus the literature review discussed in this section focuses on the topic of inverse linear optimization. For inverse integer problems, we refer the reader to Wang (2009); Schaefer (2009); Moghaddass and Terekhov (2020), for multi-objective inverse optimization, we refer the reader to Chan et al. (2014); Chan and Lee (2018); Dong and Zeng (2020)

The problem of learning the coefficients of an optimization model has been studied by some distinct research communities. Although methods developed by these distinct communities have overlapping ideas, many of them are not aware of each other's work. In this section, we review some work in the inverse optimization community and machine learning community. By identifying the connections and gaps, we leverage concepts and algorithms from different communities and develop the core work of this dissertation.

2.2.1 Existing Methods from Inverse Optimization

To our knowledge, the notion of IO was first introduced in Burton and Toint (1992); Burton (1993). They study the inverse shortest path problem, that is, determine the travel cost of edges in a network from a given optimal path. Troutt (1995) defines the concept of *decision efficiency* as the ratio of the prediction's objective value to the observed solution's objective value. Correspondingly, the IO problem is to find the model coefficient that maximizes *decision efficiency.* We also notice that Troutt (1995) mentions another earlier reference for IO problem from Stevenson (1986). Inspired by these paper, IO is studied under various settings and different mathematical programming models have been proposed in the literature.

Here we first introduce some popular mathematical programming (MP) models used in the literature. Ahuja and Orlin (2001) studies the inverse linear optimization problem, and the goal is to learn the cost from a single observation (see Model (2.1) presented above). Ahuja and Orlin calls a cost vector *inverse feasible* with respect to an observed solution if it makes the observed solution optimal. Later, Tavaslıoğlu et al. (2018) generalizes the idea of inverse feasible cost vector and formally characterizes the *inverse feasible region* of a non-empty face. Tavaslıoğlu et al. also propose a formulation to minimize the distance from a target cost vector and discuss its connection with Ahuja and Orlin (2001). The idea of *inverse feasible cost vector* is also discussed in Chan et al. (2019) and Gupta and Zhang (2020) but under a different name. Keshavarz et al. (2011) is the first to propose a formulation based on the Karush-Kuhn-Tucker (KKT) conditions (see Model (2.3) above). This formulation is also studied in Saez-Gallego and Morales (2017) and Gupta and Zhang (2020).

For the rest of this section, we refer to this type of formulation as the IO-KKT. Aswani et al. (2018) is the first paper states that IO can be formulated as a bilevel optimization problem, and they proposed an approach to reformulate the bilevel problem as a single-level optimization problem using strong duality. Similar strong duality-based formulations are developed in Chan et al. (2019) and Ghobadi and Mahmoudzadeh (2020). For the rest of this section, we refer to this type of formulation as the IO-Dual.

For the rest of this section, we discuss the literature based on the categorization of non-parametric and parametric LP.

Non-parametric LP Closed-form solutions for learning the cost vector have been derived by Chan et al. (2019) for the case of a single noisy observation and by Babier et al. (2020) for the case of multiple noisy observations. Chan and Kaw (2020) derives a closed-form solution for learning the left-hand-side constraint coefficients $\mathbf{A}$ under the same setting of non-parametric linear programs. Tavaslıoğlu et al. (2018) propose a new formulation to learn the cost vector of an LP using the idea of *inverse-feasible-region*, that is, to represent the collections of all candidate cost vectors using all active constraints. The same idea is also mentioned in Chan et al. (2019) and Gupta and Zhang (2020). Schede et al. (2019) focus on learning the constraints of an LP from both feasible and infeasible observed solutions using incremental learning strategy. That is, solving a mixed-integer linear program with observed solutions added iteratively. Ghobadi and Mahmoudzadeh (2020) study the problem of learning both left and right-hand-side constraint coefficients $\mathbf{A}, \mathbf{b}$ from multiple observations by encoding a new optimization model using the strong duality. However, learning objective and constraint coefficients jointly has, to date, received little attention. This task has been investigated by Troutt et al. (2008, 2005) who addresses the case of learning $\mathbf{c}$ and $\mathbf{A}$. However, their approach was limited to two dimensions (Troutt et al., 2005) or required the coefficients to be non-negative (Troutt et al., 2008).

Parametric LP In the parametric optimization setting, Keshavarz et al. (2011) encoded an optimization model using the IO-KKT method. The model minimizes the residuals of the complementary slackness and stationary conditions, and it is used to impute objective function coefficients of a convex optimization problem. Aswani et al. (2018) focus on the same problem under the assumption of noisy measurements, developing two algorithms that are shown to maintain statistical consistency. For the same problem in the LP setting, a recent work (Gupta and Zhang, 2020) proposes a two-phase algorithm to first project noisy observations to a vertex and then find the optimal cost vector. Saez-Gallego and Morales (2017) address the case of learning $\mathbf{c}$ and $\mathbf{b}$ jointly in a parametric setting where the $\mathbf{b}$ vector is assumed to be an affine function of a regressor.

Given the literature review above, we identify a few research gaps as follows. Firstly, the general case of learning the parameters of PLP where **c**, **A** and **b** all depend on other parameters has not been addressed in the literature. Secondly, most of the IO literature makes the use of IO-Dual or IO-KKT methods and then proposes algorithms that are tailed to specific problem settings. Despite all the advancements in theory and applications, there are no general methods that can be flexibly applied to solve inverse linear optimization problems under different problem settings. Lastly, the majority of the IO literature focuses on learning the objective, and several different formulations have been proposed. To our knowledge, a comprehensive comparison of these formulations has not been completed in the literature, neither in the non-parametric nor the parametric case.

To summarize, we list some important references in table 2.1 and highlight some key characteristics. To our knowledge, none of the existing methods in IO literature address the problem of learning all model coefficients jointly under either non-parametric or parametric setting.

Literature	Coef. to learn			LP problem setting		Num. Obs.		Obs. type	
	c	**A**	**b**	Param.	Non-Param.	Single	Multiple	Noisy	Noise-free
Ahuja and Orlin (2001)	✓				✓	✓		✓	
Troutt et al. (2005)	✓	✓			✓		✓		✓
Troutt et al. (2008)	✓	✓			✓		✓		✓
Keshavarz et al. (2011)	✓			✓			✓	✓	
Saez-Gallego et al. (2017)	✓		✓	✓			✓		✓
Bärmann et al. (2017)[1]	✓		✓	✓			✓		✓
Aswani et al. (2018)	✓			✓			✓	✓	
Tavaslıoğlu et al. (2018)	✓				✓		✓		✓
Dong et al. (2018)[1]	✓			✓			✓	✓	
Chan et al. (2019)	✓				✓	✓			✓
Schede et al. (2019)		✓	✓		✓		✓		✓
Babier et al. (2020)	✓				✓		✓	✓	
Ghobadi et al. (2020)		✓			✓		✓	✓	
Gupta and Zhang (2020)	✓			✓			✓	✓	
Chan and Kaw (2020)		✓			✓	✓			✓
Bärmann et al. (2020)[1]	✓			✓			✓	✓	

Table 2.1: Summary of literature. Abbreviation used in the table: coef. stands for coefficients, param. stands for parametric and obs. stands for observation.

[1]This work views IO from an online learning perspective.

2.2.2 Existing Methods from Machine Learning

Amos and Kolter (2017) proposed integrating a quadratic programming (QP) layer in a deep neural network and derive parameter gradients through implicit differentiation of the KKT conditions. Agrawal et al. (2019) develop a differentiable convex optimization layer, which can also be embedded into a deep neural network. Similarly, they also make use of implicit differentiation to derive parameter gradients.

Some recent work (Bärmann et al., 2017; Dong et al., 2018; Bärmann et al., 2020) view IO through the lens of online learning, where the optimization model is incrementally updated based on new observations arriving over time. In particular, Dong et al. (2018) updates the parameter by solving an optimization model encoded using the KKT conditions and Bärmann et al. (2017, 2020) updates the parameter using the multiplicative weights update (MWU) algorithm (Arora et al., 2012).

Another related work is the literature of inverse reinforcement learning (IRL) (Ng et al., 2000; Abbeel and Ng, 2004; Ratliff et al., 2006) where the goal is to learn the unknown reward function of a Markov Decision Process (MDP). Solving IO problems with IRL methods requires modelling the forward optimization problem. IRL has been successfully implemented in solving some IO problems such as inverse shortest path. For more details on the connections between inverse optimization and inverse reinforcement learning, we refer the readers to Zimmermann and Frejinger (2020).

Given the literature review above, we identify a few research gaps as follows. Unlike methods in IO literature, algorithms developed by Amos and Kolter (2017); Agrawal et al. (2019) can be applied to learn all model coefficients individually or jointly. The flexibility of their methods is valuable. However, Amos and Kolter (2017) and Agrawal et al. (2019) use the decision error loss (e.g., $\| \mathbf{x}^{\text{obs}} - \mathbf{x}^{\text{lrn}} \|_2^2$), and they did not investigate other types of loss that are commonly used in the IO literature, such as objective error loss (e.g., $| \mathbf{c}^T(\mathbf{x}^{\text{obs}} - \mathbf{x}^{\text{lrn}}) |$). Their methods do not allow adding additional constraints which are commonly used in IO applications for encoding prior knowledge on the FOP. Lastly, Amos and Kolter (2017) and Agrawal et al. (2019) focus on *quadratic program* and *convex optimization problems* respectively. To our knowledge, no similar work has been done focused on LP. To take advantage of the rich literature of LP, this thesis focuses on the inverse linear optimization problems and aims to develop new algorithms that can be flexibly applied to different settings.

Chapter 3

Deep Inverse Optimization

3.1 Abstract

Given a set of observations generated by an optimization process, the goal of inverse optimization is to determine likely parameters of that process. We cast inverse optimization as a form of deep learning. Our method, called *deep inverse optimization*, is to unroll an iterative optimization process and then use backpropagation to learn parameters that generate the observations. We demonstrate that by backpropagating through the interior point algorithm we can learn the coefficients determining the cost vector and the constraints, independently or jointly, for both non-parametric and parametric linear programs, starting from one or multiple observations. With this approach, inverse optimization can leverage concepts and algorithms from deep learning.

3.2 Introduction

The potential for synergy between optimization and machine learning is well-recognized (Bengio et al., 2018), with recent examples including Bonami et al. (2018);

Fischetti and Jo (2018); Mahmood et al. (2018a). Our work uses machine learning for *inverse optimization* (IO). In inverse optimization, we observe one or more decisions from an unknown optimization process, and the goal is to 'learn' an optimization model that is consistent with the observations. Aspects of the unknown optimization process that we may wish to learn include terms in the objective function or constraints on the decision variables.

An early example of IO is the inverse shortest path problem, used to learn the unobservable transmission times of seismic waves which are known to follow a shortest path (Tarantola, 1987). Other applications include determining the tolls that would enforce a desired traffic flow (Burton and Toint, 1992), imputing the relative importance of treatment objectives from clinically-approved radiotherapy plans (Chan et al., 2014; Mahmood et al., 2018b) in order to automate clinicians' decision-making, and predicting the behaviour of price-responsive customers (Saez-Gallego and Morales, 2017).

We illustrate our framework in the context of parametric linear optimization. Specifically, consider the parametric linear program PLP($\mathbf{u}, \mathbf{w}$):

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}(\mathbf{u},\mathbf{w})'\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}(\mathbf{u},\mathbf{w})\mathbf{x} \leq \mathbf{b}(\mathbf{u},\mathbf{w}), \end{aligned} \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{c}(\mathbf{u},\mathbf{w}) \in \mathbb{R}^D$, $\mathbf{A}(\mathbf{u},\mathbf{w}) \in \mathbb{R}^{M\times D}$ and $\mathbf{b}(\mathbf{u},\mathbf{w}) \in \mathbb{R}^M$. The 'feature' vector $\mathbf{u}$ represents conditions (e.g., time, prices, weather) under which we may want to instantiate and solve the linear program. The 'weight' vector $\mathbf{w}$ represents parameters relating the features to the optimization model instance.

In inverse optimization, for a set of features $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$ we observe the corresponding decisions of some unknown optimization process. Call these decisions

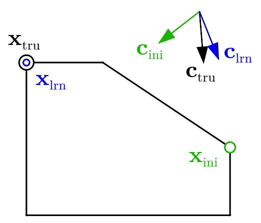
$\{\mathbf{x}_1^{\text{obs}}, \mathbf{x}_2^{\text{obs}}, \ldots, \mathbf{x}_N^{\text{obs}}\}$, as they are generated by the 'true' underlying process. Fundamentally, IO problems are learning problems: the goal of IO is to learn weights $\mathbf{w}$ such that, for each $n \in \{1, \ldots, N\}$, there exists an optimal solution of $\text{PLP}(\mathbf{u}_n, \mathbf{w})$ that is consistent with the corresponding observed decision $\mathbf{x}_n^{\text{obs}}$. The learned model can then be applied to predict decisions under new conditions $\mathbf{u}$ that were not seen at training time.

In this paper, we cast inverse optimization as a form of deep learning. Our method, called *deep inverse optimization*, is to 'unroll' an iterative optimization process and then use backpropagation to learn model parameters $\mathbf{w}$ that generate the observations, i.e., training targets. Specifically, we use a deep learning framework to trace computations across the iterations of an optimization loop, resulting in a chain of dependent variables (a dynamically unrolled loop) which are then automatically differentiated with respect to a loss function so as to compute a gradient for $\mathbf{w}$.

Figure 3.1 shows the actual result of applying our deep IO method to three inverse optimization learning tasks. The top panel illustrates the *non-parametric*, single-point variant of model (3.1) — the case when exactly one $\mathbf{x}^{\text{obs}}$ is given — a classical problem in IO see (Ahuja and Orlin, 2001; Chan et al., 2019). In Figure 3.1 (i), only $\mathbf{c}$ needs to be learned: starting from an initial cost vector $\mathbf{c}^{\text{ini}}$, our method finds $\mathbf{c}^{\text{lrn}}$ which makes $\mathbf{x}^{\text{obs}}$ an optimal solution of the LP by minimizing $\|\mathbf{x}^{\text{obs}} - \mathbf{x}^{\text{lrn}}\|^2$. In Figure 3.1 (ii), starting from $\mathbf{c}^{\text{ini}}$, $\mathbf{A}^{\text{ini}}$ and $\mathbf{b}^{\text{ini}}$, our approach finds $\mathbf{c}^{\text{lrn}}$, $\mathbf{A}^{\text{lrn}}$ and $\mathbf{b}^{\text{lrn}}$ which make $\mathbf{x}^{\text{obs}}$ an optimal solution of the learned LP through minimizing $\|\mathbf{x}^{\text{obs}} - \mathbf{x}^{\text{lrn}}\|^2$.

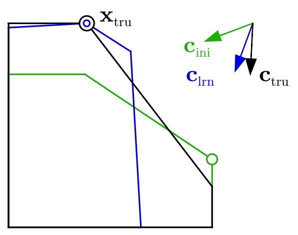
Figure 3.1 (iii) shows learning $\mathbf{w} = [w_0, w_1]$ for the *parametric* problem instance

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \cos(w_0 + w_1 u)x_1 + \sin(w_0 + w_1 u)x_2 \\ \text{subject to} \quad & -x_1 \leq 0.2 w_0 u, \\ & -x_2 \leq -0.2 w_1 u, \\ & w_0 x_1 + (1 + \tfrac{1}{3} w_1 u)x_2 \leq w_0 + 0.1u. \end{aligned} \tag{3.2}$$



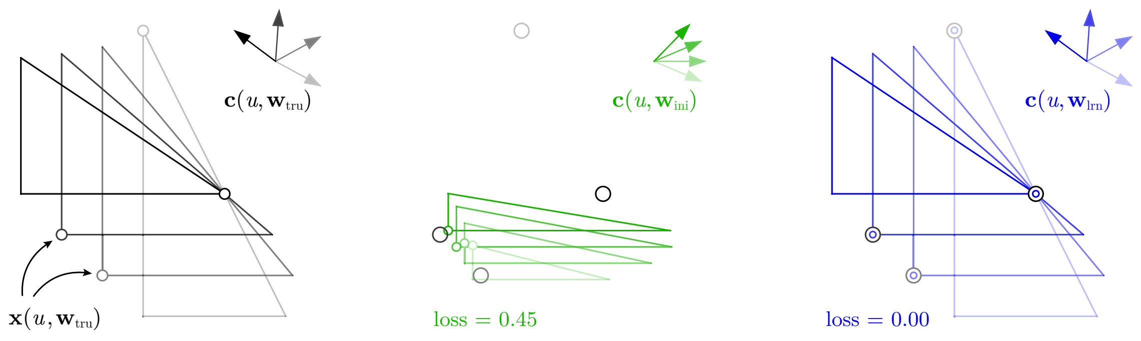
$\mathbf{x}_{\text{tru}}$
$\mathbf{x}_{\text{lrn}}$
$\mathbf{c}_{\text{ini}}$
$\mathbf{c}_{\text{lrn}}$
$\mathbf{c}_{\text{tru}}$
$\mathbf{x}_{\text{ini}}$

(i) Learning $\mathbf{c}$ only



$\mathbf{x}_{\text{tru}}$
$\mathbf{c}_{\text{ini}}$
$\mathbf{c}_{\text{lrn}}$
$\mathbf{c}_{\text{tru}}$

(ii) Learning $\mathbf{c}$, $\mathbf{A}$, $\mathbf{b}$ jointly



$\mathbf{c}(u, \mathbf{w}_{\text{tru}})$
$\mathbf{x}(u, \mathbf{w}_{\text{tru}})$
$\mathbf{c}(u, \mathbf{w}_{\text{ini}})$
loss $= 0.45$
$\mathbf{c}(u, \mathbf{w}_{\text{lrn}})$
loss $= 0.00$

True parametric LP | Initial parametric LP | Learned parametric LP

(iii) Learning weights $\mathbf{w}$ of a parametric LP from multiple points

Figure 3.1: Three IO learning tasks in non-parametric and parametric linear programs.

The left panel of Figure 3.1 (iii) shows the true PLP$(\mathbf{u}, \mathbf{w}^{\text{tru}})$ with $\mathbf{w}^{\text{tru}} = [1.0, 1.0]$, along with four observations denoted as $\mathbf{x}(u_n, \mathbf{w}^{\text{tru}})$ corresponding to u values $\{-1.5, -0.5, 0.5, 1.5\}$. Starting from $\mathbf{w}^{\text{ini}} = [0.2, 0.4]$ with a loss (mean squared error) of 0.45, our method is able to find $\mathbf{w}^{\text{lrn}} = [1.0, 1.0]$ with a loss of zero, thereby making the observed $\mathbf{x}_n^{\text{obs}}$ optimal solutions of (3.2). In this case, the learned PLP will predict the same decisions as the true PLP when evaluated on new values of u. In other words, the learned model generalizes well.

The contributions of this paper are as follows. We propose a general framework for inverse optimization based on deep learning. This framework is applicable to learning coefficients of the objective function and constraints, individually or jointly; minimizing a general loss function; learning from a single or multiple observations; and solving both non-parametric and parametric problems. As a proof of concept, we demonstrate that our method obtains effectively zero loss on many randomly generated linear programs for all three types of learning tasks shown in Figure 3.1, and always improves the loss significantly. Such a numerical study on randomly generated non-parameteric and parametric linear programs with multiple learnable parameters has not previously been published for any IO method in the literature. Finally, to the best of our knowledge, we are the first to use unrolling and backpropagation for constrained inverse optimization.

We explain how our approach differs from methods in inverse optimization and machine learning in Section 3.3. We present our deep IO framework in Section 3.4 and our experimental results in Section 3.5. Section 3.6 discusses both the generality and the limitations of our work, and Section 3.7 concludes the paper.

3.3 Related Work

The goal of our paper is to develop a general-purpose IO approach that is applicable to problems for which theoretical guarantees or efficient exact optimization approaches are difficult or impossible to develop. Naturally, such a general-purpose approach will not be the method of choice for all classes of IO problems. In particular, for non-parametric linear programs, closed-form solutions for learning the $\mathbf{c}$ vector (Figure

3.1 (i)) and for learning the constraint coefficients have been derived by Chan et al. (2019); Babier et al. (2020) and Chan and Kaw (2020), respectively. However, learning objective and constraint coefficients jointly (Figure 3.1 (ii)) has, to date, received little attention. To the best of our knowledge, this task has been investigated only by Troutt et al. (2008, 2005), who referred to it as linear system identification, using a maximum likelihood approach. However, their approach was limited to two dimensions (Troutt et al., 2005) or required the coefficients to be non-negative (Troutt et al., 2008).

In the parametric optimization setting, Aswani et al. (2018) focus on the same problem under the assumption of noisy measurements, developing a bilevel formulation and two algorithms which are shown to maintain statistical consistency. Saez-Gallego and Morales (2017) address the case of learning $\mathbf{c}$ and $\mathbf{b}$ jointly in a parametric setting where the $\mathbf{b}$ vector is assumed to be an affine function of a regressor. The general case of learning the weights of a parametric linear optimization problem (3.1) where $\mathbf{c}$, $\mathbf{A}$ and $\mathbf{b}$ are functions of $\mathbf{u}$ (Figure 3.1 (iii)) has not been addressed in the literature.

Recent work in machine learning (Bärmann et al., 2017, 2020; Dong et al., 2018) views IO through the lens of online learning, where the optimization model is incrementally updated based on new observations. Our approach may be applicable in online settings, but in the current paper we consider problems with a fixed training set.

It is worth noting that there are conceptual parallels between inverse optimization and *constraint acquisition* (Bessiere et al., 2017), including recent variants that incorporate machine learning (Lombardi and Milano, 2018). In constraint acquisition, the goal is to allow non-expert users to specify constraint sets in the *constraint programming* formalism using an example-based approach.

Methodologically, our unrolling strategy is similar to Maclaurin et al. (2015a) who directly optimize the hyperparameters of a neural network training procedure with gradient descent. Conceptually, the closest papers to our work are by Amos and Kolter (2017) and Donti et al. (2017). However, these papers are written independently of the inverse optimization literature. Amos and Kolter (2017) present the OptNet framework, which integrates a quadratic optimization layer in a deep neural network. The gradients for updating the coefficients of the optimization problem are derived through implicit differentiation. This approach involves taking matrix differentials of the KKT conditions for the optimization problem in question, while our strategy is based on allowing a deep learning framework to unroll an existing optimization procedure. Their method has efficiency advantages, while our unrolling approach is easily applicable, including to processes for which the KKT conditions may not hold or are difficult to implicitly differentiate. We include a more in-depth discussion in Section 3.6.

3.4 Deep Learning Framework for Inverse Optimization

Inverse optimization can be viewed as an approach to machine learning specialized to the case when the observed data is coming from an optimization process. Given this perspective on IO, and motivated by the success of deep learning for a variety of learning tasks in recent years (see LeCun et al. (2015)), this paper develops a deep learning framework for inverse optimization.

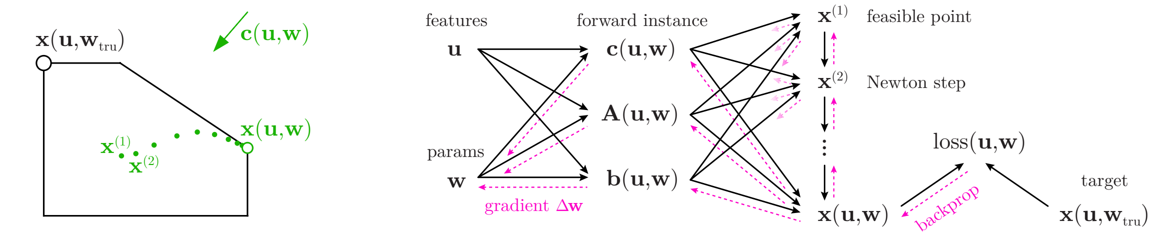
Deep learning is a set of techniques for training the parameters of a sequence of transformations (layers) that have been composed (chained) together. The more intermediate layers of computation, the 'deeper' the architecture. We refer the reader to the textbook by Goodfellow, Bengio and Courville Goodfellow et al. (2016) for details. The features of the intermediate layers can be trained/learned through backpropagation (Rumelhart et al., 1986), an automatic differentiation technique that can efficiently compute a direction in which to update the weights of the model. Importantly, current machine learning libraries such as PyTorch provide built-in backpropagation capabilities (Paszke et al., 2017), making this technique much more accessible and flexible than in the past.

Our deep IO framework cycles through three steps: (1) instantiate a forward optimization problem with the current weights $\mathbf{w}$, (2) solve the problem with a standard algorithm while tracing its execution, and (3) automatically compute an update to improve $\mathbf{w}$ by backpropagating through the traced steps.

Algorithm 1 Deep inverse optimization framework.

```
    Input: initial weights $\mathbf{w}^{\text{ini}}$; training targets $\left\{(\mathbf{u}_n, \mathbf{x}_n^{\text{obs}})\right\}_{n=1}^{N}$.
    Output: learned weights $\mathbf{w}^{\text{lrn}}$
 1: $\mathbf{w} \leftarrow \mathbf{w}^{\text{ini}}$
 2: for $s$ in 1 .. max_steps do
 3:     $\Delta\mathbf{w} \leftarrow \mathbf{0}$
 4:     for $n$ in 1 .. $N$ do                                  ▷ For each training example
 5:         $\mathbf{x} \leftarrow \mathbf{FO}(\mathbf{u}_n, \mathbf{w})$                    ▷ Run forward optimizer to completion
 6:         $\ell \leftarrow \mathcal{L}(\mathbf{x}, \mathbf{x}_n^{\text{obs}})$                         ▷ Compute loss w.r.t. target
 7:         $\frac{\partial \ell}{\partial \mathbf{w}} \leftarrow$ backprop($\ell$)                  ▷ Backpropagate gradient to weights
 8:         $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} + \frac{1}{N}\frac{\partial \ell}{\partial \mathbf{w}}$                       ▷ Accumulate average gradient
 9:     end for
10:     $\Delta\mathbf{w} \leftarrow \boldsymbol{\alpha} \odot \Delta\mathbf{w}$                         ▷ Scale gradient component-wise
11:     $\beta \leftarrow$ line_search($\mathbf{w}, \Delta\mathbf{w}$)                        ▷ Find safe step size
12:     $\mathbf{w} \leftarrow \mathbf{w} - \beta\Delta\mathbf{w}$                                   ▷ Update weights
13: end for
14: Return $\mathbf{w}$
```

Our approach, shown in Algorithm 1, takes the pairs $\left\{(\mathbf{u}_n, \mathbf{x}_n^{\text{obs}})\right\}_{n=1}^{N}$ as input, and initializes $\mathbf{w}$ to $\mathbf{w}^{\text{ini}}$. For each n, the forward optimization problem ($\mathbf{FO}$) is solved with the current weights (line 5), and the loss between the resulting optimal solution $\mathbf{x}$ and $\mathbf{x}^{\text{obs}}$ is computed (line 6). The gradient of the loss function with respect to $\mathbf{w}$ is computed by backpropagation through the layers of the forward process (line 7). The gradient is optionally scaled by component-wise product ($\odot$) with a vector $\boldsymbol{\alpha}$ that


$\mathbf{x}(\mathbf{u},\mathbf{w}_{\text{tru}})$
$\mathbf{c}(\mathbf{u},\mathbf{w})$
$\mathbf{x}^{(1)}$
$\mathbf{x}^{(2)}$
$\mathbf{x}(\mathbf{u},\mathbf{w})$
features
$\mathbf{u}$
params
$\mathbf{w}$
gradient $\Delta\mathbf{w}$
forward instance
$\mathbf{c}(\mathbf{u},\mathbf{w})$
$\mathbf{A}(\mathbf{u},\mathbf{w})$
$\mathbf{b}(\mathbf{u},\mathbf{w})$
$\mathbf{x}^{(1)}$
feasible point
$\mathbf{x}^{(2)}$
Newton step
$\vdots$
$\mathbf{x}(\mathbf{u},\mathbf{w})$
backprop
$\text{loss}(\mathbf{u},\mathbf{w})$
target
$\mathbf{x}(\mathbf{u},\mathbf{w}_{\text{tru}})$

(i) IPM forward process (ii) Deep inverse optimization through IPM

Figure 3.2: Illustration of the deep inverse optimization framework.

controls the relative learning rates (line 10). Line search then determines a safe step size β that precludes an increase in the overall loss and prevents the forward problem from becoming unbounded or infeasible (line 11). Finally, the weights are updated (line 12). This process repeats until `max_steps` iterations are complete.

Importantly, our framework is applicable in principle to any differentiable forward optimization procedure. Gradients are automatically computable even with non-linear constraints or non-linear objectives, as long as they can be expressed through standard differentiable primitives. For our experiments we implement the barrier interior point method (IPM) as described by Boyd and Vandenberghe Boyd and Vandenberghe (2004) for our forward solver. The IPM forward process is illustrated in Figure 3.2 (i): the central path taken by IPM is illustrated for the current $\mathbf{u}$ and $\mathbf{w}$, which define both the current feasible region and the current $\mathbf{c}(\mathbf{u}, \mathbf{w})$. As shown in Figure 3.2 (ii), backpropagation starts from the loss between between IPM solution $\mathbf{x}(\mathbf{u}, \mathbf{w})$ and the target $\mathbf{x}(\mathbf{u}, \mathbf{w}^{\text{tru}})$ and proceeds backward to the initial state $\mathbf{x}_{(1)}$ of IPM. The key to backpropagating through each Newton step of IPM is to differentiate a matrix inverse operation, which PyTorch now does automatically. In practice, backpropagating all the way to $\mathbf{x}_{(1)}$ may not be necessary for computing sufficiently accurate gradients; see Section 3.6.

The framework requires setting three main hyperparameters: $\mathbf{w}^{\text{ini}}$, the initial weight vector; `max_steps`, the total number of steps allotted to the training; and

$\boldsymbol{\alpha}$, the learning rates for the different components of $\mathbf{w}$. The number of additional hyperparameters depends on the forward optimization process.

3.5 Experimental Results

In this section, we demonstrate the application of our framework on randomly-generated linear programs for the three types of problems shown in Figure 3.1: learning $\mathbf{c}$ in the non-parametric case; learning $\mathbf{c}$, $\mathbf{A}$ and $\mathbf{b}$ together in the non-parametric case; and learning $\mathbf{w}$ in the parametric case.

Implementation

Our framework is implemented as a Python package called `deep_inv_opt`[1]. The package is designed to be used with PyTorch version 1.0, leveraging its built-in automatic differentiation and backpropagation capabilities (Paszke et al., 2017). All numerical operations are carried out with PyTorch tensors and standard PyTorch primitives, including the matrix inverse at the heart of the Newton step.

Hyperparameters

We limit learning to `max_steps` $= 200$ in all experiments. Four additional hyperparameters are set in each experiment:

- ϵ, which controls the precision and termination of IPM;
- $t^{(0)}$: the initial value of the barrier IPM sharpness parameter t;
- μ: the factor by which t is increased along the IPM central path;
- $\boldsymbol{\alpha}$: the vector of per-parameter learning rates, which in some experiments is broken down into $\boldsymbol{\alpha}_{\mathbf{c}}$ and $\boldsymbol{\alpha}_{\mathbf{Ab}}$.

[1]Available at `https://github.com/yingcongtan/deep_inv_opt`

In all experiments, the ϵ hyperparameter is either a constant 10^{-5} or decays exponentially from 0.1 to 10^{-5} during learning.

Benchmark methodology

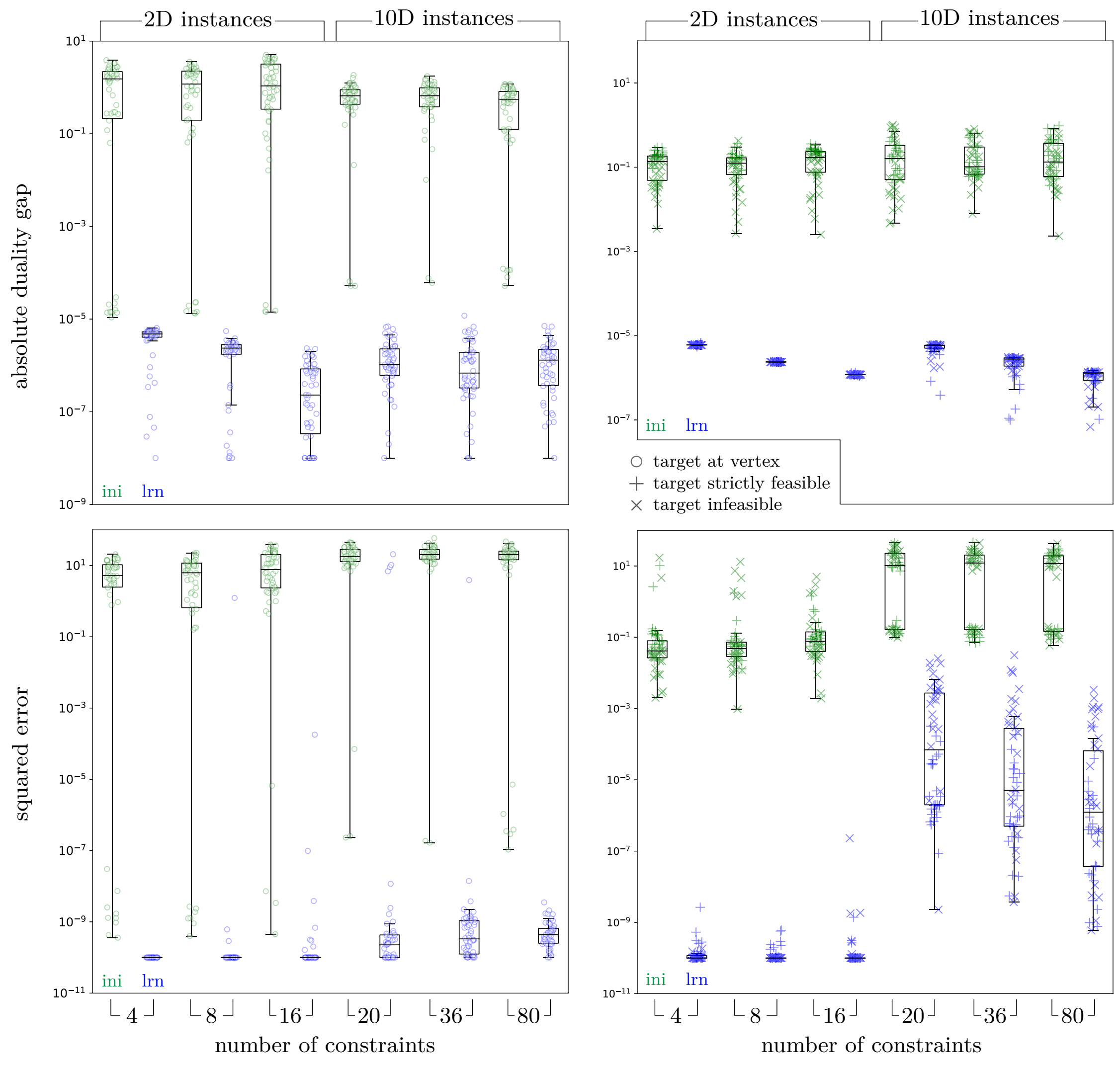
To the best of our knowledge, there are no well-established benchmarks in the IO literature. Thus, we develop an IO benchmark comprising random instances with varying dimension and number of constraints. We generate a set of feasible regions having D dimensions and M constraints by sampling at least D points with components from $\mathcal{N}(0,1)$ and computing their convex hull via the *scipy.spatial.convexhull* package (QHull Library, 2018). We refer to these as 'baseline' feasible regions. We generate 50 baseline feasible regions for each of the following six problem sizes: $D = 2$ with $M \in \{4, 8, 16\}$, and $D = 10$ with $M \in \{20, 36, 80\}$. The baseline regions and training/testing targets in our experiments can all be generated by scripts in the accompanying code repository. Though we observe that our method works for equality constraints, our experiments focus on inequality constraints, and we leave a systematic evaluation of equality constraints to future work.

3.5.1 Experiments on non-parametric linear programs

We first demonstrate the performance of our method for learning $\mathbf{c}$ only, and learning $\mathbf{c}$, $\mathbf{A}$ and $\mathbf{b}$ jointly, on the single-point variant of model (3.1), i.e., when a single optimal target $\mathbf{x}^{\text{obs}}$ is given, a classical assumption in IO (Ahuja and Orlin, 2001). We use two loss functions, *absolute objective error* (AOE) and *squared decision error* (SDE), defined as follows:

$$\text{AOE} = |\, \mathbf{c}^{\text{lrn}\prime}(\mathbf{x}^{\text{obs}} - \mathbf{x}^{\text{lrn}})\,|, \tag{3.3}$$

$$\text{SDE} = \|\, \mathbf{x}^{\text{obs}} - \mathbf{x}^{\text{lrn}}\,\|_2^2. \tag{3.4}$$



(i) Learning **c** only (ii) Learning **c**, **A**, **b** jointly

Figure 3.3: Experimental results for non-parametric IO problems.

Both AOE and SDE have been used in IO Chan et al. (2014, 2019); Babier et al. (2020), and SDE is a standard metric in machine learning.

Learning c only

For each of the 50 baseline feasible regions, we randomly select one vertex of the convex hull to be the training target $\mathbf{x}^{\text{obs}}$. We set $\mathbf{A}$ and $\mathbf{b}$ to match the baseline feasible region, and we generate a random $\mathbf{c}^{\text{ini}}$ by drawing from $\mathcal{N}(0, 1)$. The goal is to find a $\mathbf{c}^{\text{lrn}}$ for which $\mathbf{x}^{\text{obs}}$ is a solution.

We implement a randomized grid search by sampling 20 random combinations of the following three hyperparameter sets: $t^{(0)} \in \{0.5, 1, 5, 10\}$, $\mu \in \{1.5, 2, 5, 10, 20\}$, and $\boldsymbol{\alpha}_{\mathbf{c}} \in \{1, 10, 100, 1000\}$. These hyperparameter sets were chosen based on intuition from preliminary experiments. As in other applications of deep learning, it is not clear which hyperparameters will work best for a particular problem instance. For each instance we run our algorithm with the same 20 hyperparameter combinations, reporting the best final error values. Note that in this experiment a loss of zero is achievable by a closed-form expression (Chan et al., 2019), so the success of our method can be evaluated in absolute terms by the fraction of instances that achieve zero loss.

Figure 3.3 (i) shows the results of this experiment for AOE and SDE loss. In both cases, our method is able to reliably learn $\mathbf{c}$: for all instances, the final error is under 10^{-4}, while the majority of initial errors are above 10^{-1}. There is no clear pattern in the performance of the method as M and D change for AOE; for SE, the final loss is slightly larger for higher D.

Learning $\mathbf{c}$, $\mathbf{A}$, $\mathbf{b}$ jointly

For each of the 50 baseline feasible regions, we generate a random baseline $\mathbf{c}$ vector to form a baseline LP. We then generate an either strictly feasible or strictly infeasible target point $\mathbf{x}^{\text{obs}}$ by perturbing an optimal solution to the baseline LP. We interpret these strictly feasible/infeasible targets as being a mismatch between the baseline LP and some unknown true LP we wish to recover. We set $\mathbf{A}^{\text{ini}}$ and $\mathbf{b}^{\text{ini}}$ to be the baseline feasible region, and set $\mathbf{c}^{\text{ini}}$ to be a perturbed version of the baseline $\mathbf{c}$ vector. The IO algorithm must then find a $\mathbf{c}^{\text{lrn}}, \mathbf{A}^{\text{lrn}}, \mathbf{b}^{\text{lrn}}$ for which the target $\mathbf{x}^{\text{obs}}$ is optimal.

Specifically, for each of the 50 baseline feasible regions, we generate a $\mathbf{c} \sim \mathcal{N}(0, 1)$ and compute its optimal solution $\mathbf{x}^*$. To generate an infeasible target we set $\mathbf{x}^{\text{obs}} = \mathbf{x}^* + \eta$ where $\eta \sim U[-0.2, 0.2]$. We similarly generate a challenging $\mathbf{c}^{\text{ini}}$ by corrupting $\mathbf{c}$ with noise from $U[-0.2, 0.2]$. To generate a strictly feasible target near $\mathbf{x}^*$, we set $\mathbf{x}^{\text{obs}} = 0.9\mathbf{x}^* + 0.1\mathbf{x}'$ where $\mathbf{x}'$ is a random point within the feasible region, generated by a Dirichlet-weighted combination of all vertices; this method was used because adding noise to a vertex in 10 dimensions almost always results in an infeasible target.

In summary, the IO task involves both a misspecified $\mathbf{c}^{\text{ini}}$ and a misspecified feasible region $\mathbf{A}^{\text{ini}}$ and $\mathbf{b}^{\text{ini}}$ relative to the target $\mathbf{x}^{\text{obs}}$. The goal is to demonstrate the ability of our algorithm to alter the constraints and the objective so that the feasible/infeasible target becomes an optimum. For each of the six problem sizes, we randomly split the 50 instances into two subsets, one with feasible and the other with infeasible targets. For AOE loss we set $\epsilon = 10^{-5}$ and for SDE we use the ϵ decay strategy. In practice, this decay strategy is similar to putting emphasis on learning $\mathbf{c}$ in the initial iterations and ending with emphasis on constraint learning.

The values of hyperparameters $\boldsymbol{\alpha}_{\mathbf{c}}$ and $\boldsymbol{\alpha}_{\mathbf{Ab}}$ are independently selected from $\{0.1, 1, 10\}$ and concatenated into one learning rate vector $\boldsymbol{\alpha}$. We generate 20 different hyperparameter combinations from the same hyperparameter sets as described above.

We run our algorithm on each instance with all hyperparameter combinations and record the value of the best trial. The minimum achievable loss in this experiment is again zero, so the success of our method can be evaluated in absolute terms by the fraction of instances achieving zero loss.

Figure 3.3 (ii) shows the results of this experiment for AOE and SDE loss. In both cases, our method is able to learn model parameters that result in median loss of under 10^{-4}. For AOE, our method performs equally well for all problem sizes, and there is not much difference in the final loss for feasible and infeasible targets. For SE, however, the final loss is larger for higher D but decreases as M increases. Furthermore, there is a visible difference in performance of the method on feasible and infeasible points for 10-dimensional instances: learning from infeasible targets becomes a more difficult task.

3.5.2 Experiments on parametric linear programs

Several aspects of the experiment for parametric LPs are different from the non-parametric case. First, we train by minimizing the aggregated SDE($\mathbf{w}$), defined as

$$\mathrm{SDE}(\mathbf{w}) \;=\; \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{x}(\mathbf{u}_n, \mathbf{w}^{\mathrm{tru}}) - \mathbf{x}(\mathbf{u}_n, \mathbf{w})\|_2^2. \tag{3.5}$$

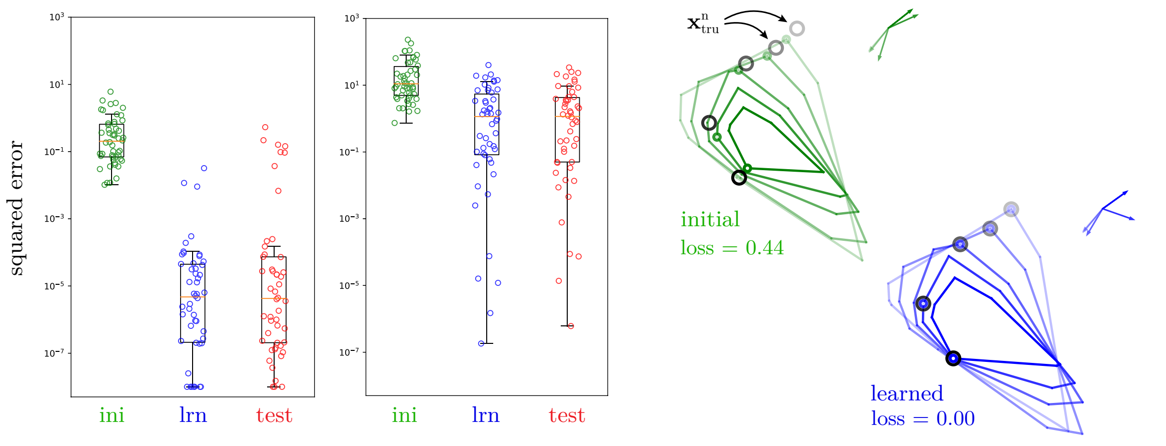
For the parametric experiments, we chose to train and evaluate using the SDE loss instead of AOE for reasons discussed in Section 3.6. In the parametric case, we also assess how well the learned PLP generalizes, by evaluating its SDE($\mathbf{w}^{\mathrm{lrn}}$) on a held-out test set.

To generate parametric problem instances, we again started from the baseline feasible regions. To generate a true PLP, we used six weights to define linear functions of u for all elements of $\mathbf{c}$, all elements of $\mathbf{b}$, and one random element in each row of $\mathbf{A}$. For example, for 2-dimensional problems with four constraints, our instances have the following form:

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & (c_1 + w_1 + w_2 u)x_1 + (c_2 + w_1 + w_2 u)x_2 \\ \text{subject to} & \begin{bmatrix} a_{11} & a_{12} + w_3 + w_4 u \\ a_{21} & a_{22} + w_3 + w_4 u \\ a_{31} + w_3 + w_4 u & a_{32} \\ a_{41} & a_{42} + w_3 + w_4 u \end{bmatrix} \leq \begin{bmatrix} b_1 + w_5 + w_6 u \\ b_2 + w_5 + w_6 u \\ b_3 + w_5 + w_6 u \\ b_4 + w_5 + w_6 u \end{bmatrix}. \end{array} \tag{3.6}$$

Specifically, the "true PLP" instances are generated by setting $w_1, w_3, w_5 = 0$ and $w_2, w_4, w_6 \sim \mathcal{N}(0, 0.2)$. This ensures that when $u = 0$ the true PLP feasible region matches the baseline feasible region. For each true PLP, we find a range $[u_{min}, u_{max}] \subseteq [-1, 1]$ over which the resulting PLP remains bounded and feasible. To find this 'safe' range we evaluate u at increasingly large values and try to solve the corresponding LP, expanding $[u_{min}, u_{max}]$ if successful. For each true PLP, we generate 20 equally spaced training points spanning $[u_{min}, u_{max}]$. We also sample 20 test points u sampled uniformly from $[u_{min}, u_{max}]$. We then initialize learning from a corrupted PLP by setting $\mathbf{w}^{\text{ini}} = \mathbf{w}^{\text{tru}} + \eta$ where each element of $\eta \sim U[-0.2, 0.2]$.

Hyperparameters are sampled from $t^{(0)} \in \{0.5, 1, 5, 10\}$, $\mu \in \{1.5, 2, 5, 10, 20\}$ and $\boldsymbol{\alpha}_{\mathbf{Ab}} \in \{1, 10\}$, and $\boldsymbol{\alpha}_{\mathbf{c}}$ is then chosen to be a factor of $\{0.01, 1, 100\}$ times $\boldsymbol{\alpha}_{\mathbf{Ab}}$, i.e., a relative learning rate. The range of these values was based on preliminary experiments. Here, $\boldsymbol{\alpha}_{\mathbf{c}}$ and $\boldsymbol{\alpha}_{\mathbf{Ab}}$ control the learning rate of parameters within $\mathbf{w}$ that determine $\mathbf{c}$ and $(\mathbf{A}, \mathbf{b})$, respectively. In total, we generate 20 different hyperparameter combinations. We run our algorithm on each instance with all hyperparameter combinations and record the best final error value. A constant value of $\epsilon = 10^{-5}$ is used. In these experiments, the minimum achievable loss is again zero.



(i) 2D PLP instances (8 constraints) (ii) 10D PLP instances (36 constraints) (iii) 2D PLP instance example

Figure 3.4: Experimental results for parametric IO problems.

We demonstrate the performance of our method on learning parametric LPs of the form shown in (3.6) with $D = 2$, $M = 8$, and $D = 10$, $M = 36$. In Figure 3.4, we report two metrics evaluated on the training set, namely $\mathrm{SDE}(\mathbf{w}^{\mathrm{ini}})$ and $\mathrm{SDE}(\mathbf{w}^{\mathrm{lrn}})$, and one metric for the test set, $\mathrm{SDE}(\mathbf{w}^{\mathrm{lrn}})$. Figure 3.4 (iii) shows an example of an instance with $D = 2$, $M = 8$ from the training set. We see that, overall, our deep learning method works well on 2-dimensional problems with the training and testing error both being much smaller than the initial error. In the vast majority of cases the test error is also comparable to training error, though there are a few cases where it is worse, which indicates a failure to generalize well. For 10D instances, the algorithm significantly improves $\mathrm{SDE}(\mathbf{w}^{\mathrm{lrn}})$ over the initialization $\mathrm{SDE}(\mathbf{w}^{\mathrm{ini}})$, but in most cases fails to drive the loss to zero, either due to local minima or slow convergence. Again, performance on the test set is similar to that on training set.

3.6 Discussion

The conceptual message that we wish to reinforce is that inverse optimization can be viewed as a form of deep learning, and that unrolling gives easy access to the gradients of any parameter used directly or indirectly in the forward optimization process. There are many aspects of this view that merit further exploration. What kind of forward optimization processes can be inversely optimized this way? Which ideas and algorithms from the deep learning community will help? Are there characteristics of IO that make gradient-based learning more challenging than in deep learning at large? Conclusive answers are beyond the scope of this paper, but we discuss these and other questions below.

Relation to neural networks

Deep neural networks often have millions of trainable weights and are very flexible in what kinds of input-output relations they can learn, thus requiring very large training sets. The optimization models we consider have comparatively few trainable weights because they represent a strong prior over how features $\mathbf{u}$ determine decisions $\mathbf{x}$. As such, they require less training data than a typical neural network, which is why we can train our parametric instances on only 20 training points and not observe over-fitting.

Generality and applicability

As a proof of concept, this paper uses linear programs as the forward problems with barrier IPM as the optimization process. In principle, the framework is applicable to any forward process for which automatic differentiation can be applied. This observation does not mean that ours is the best approach for a specialized IO problem, such as learning $\mathbf{c}$ from a single point (Chan et al., 2019) or multiple points within the same feasible region (Babier et al., 2020), but it provides a new strategy.

The practical message of our paper is that, when faced with novel classes or novel parameterizations of IO problems, the unrolling strategy provides convenient access to a suite of general-purpose gradient-based algorithms for solving the IO problem at hand. This strategy is made especially easy by deep learning libraries that support dynamic 'computation graphs' such as PyTorch. Researchers working within this framework can rapidly apply IO to many differentiable forward optimization processes, without having to derive the algorithm for each case. Automatic differentiation and backpropagation have enabled a new level of productivity for deep learning research, and they may do the same for inverse optimization research. Applying deep inverse optimization does not require expertise in deep learning itself.

We chose IPM as a forward process because the inner Newton step is differentiable and because we expected the gradient to temperature parameter t to have a stabilizing effect on the gradient. For non-differentiable optimization processes, it may still be possible to develop differentiable versions. In deep learning, many advances have been made by developing differentiable versions of traditionally discrete operations, such as memory addressing (Graves et al., 2016) or sampling from a discrete distribution (Maddison et al., 2016). We believe the scope of differentiable forward optimization processes may similarly be expanded over time.

Finally, it may be possible to develop hybrid approaches, combining gradient-based learning with closed-form solutions, combinatorial algorithms, coordinate descent schemes, or techniques from black-box optimization.

Limitations and possible improvements

Deep IO inherits the limitations of most gradient-based methods. If learning is initialized to the right "basin of attraction", it can proceed to a global optimum. Even then, the choice of learning algorithm may be crucial. When implemented within a steepest descent framework, as we have here, the learning procedure can get trapped in local minima or exhibit very slow convergence. Such effects are why most instances in Figure 3.4 (ii) failed to achieve zero loss.

In deep learning with neural networks, poor local minima become exponentially rare as the dimension of the learning problem increases (Dauphin et al., 2014; Soudry and Hoffer, 2017). A typical strategy for training neural networks is therefore to over-parameterize (use a high search dimension) and then use regularization to avoid over-fitting to the data. In deep IO, natural parameterizations of the forward process may not permit an increase in dimension, or there may not be enough observations for regularization to compensate, so local minima remain a potential obstacle. We believe training and regularization methods specialized to low-dimensional learning problems such as those from Sahoo et al. (2018) may be applicable here.

We expect that other techniques from deep learning, and from gradient-based optimization in general, will translate to deep IO. For example, learning algorithms with second-order aspects such as momentum (Sutskever et al., 2013) and L-BFGS (Byrd et al., 1995) are readily available in deep learning frameworks. Deep learning 'tricks' may also help deep IO. For example, we observe that, when **c** is normal to a constraint, the gradient with respect to **c** can suddenly become very large. We stabilized this behaviour with line search, but a similar 'exploding gradient' phenomenon exists when training deep recurrent networks, and gradient clipping (Pascanu et al., 2012) is a popular way to stabilize training. A detailed investigation of applicable deep learning techniques is outside the scope of this paper.

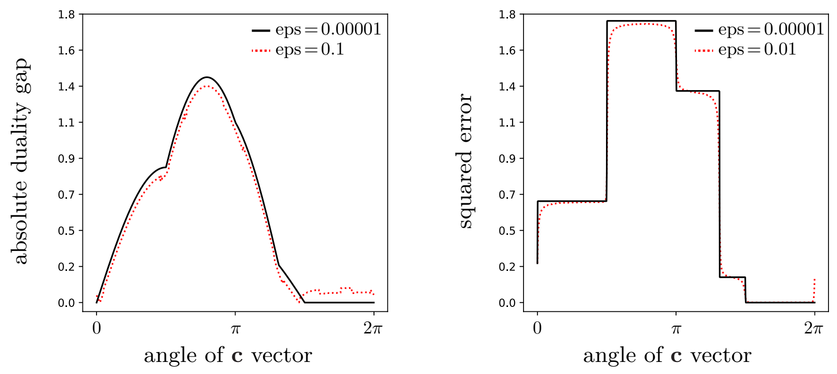
Deep IO may be more successful when the loss with respect to the forward process can be annealed or 'smoothed' in a manner akin to graduated non-convexity (Blake and Zisserman, 1987). Our ϵ-decay strategy is an example of this approach, as discussed below.

Loss function and metric of success

One advantage of the deep inverse optimization approach is that it can accommodate various loss functions, or combinations of loss functions, without special development or analysis. For example one could substitute other p-norms, or losses that are robust to outliers, and the gradient will be automatically available. This flexibility may be valuable. Special loss functions have been important in machine learning, especially for structured output problems (Hazan et al., 2010). The decision variables of optimization processes are likewise a form of structured output.

In this study we chose two classical loss functions: absolute duality gap and squared error. The behaviour of our algorithm varied depending on the loss function used. Looking at Figure 3.3 (ii) it appears that deep IO performs better with ADG loss than with SE loss when learning $\mathbf{c}, \mathbf{A}, \mathbf{b}$ jointly. However, this performance is due to the theoretical property that ADG can be zero despite the observed target point being infeasible (Chan et al., 2019). With ADG, all the IO solver needs to do is adjust $\mathbf{c}, \mathbf{A}, \mathbf{b}$ so that $\mathbf{x}^{\text{lrn}} - \mathbf{x}^{\text{obs}}$ is orthogonal to $\mathbf{c}$, which in no way requires the learned model to be capable of generating $\mathbf{x}^{\text{obs}}$ as an optimum. In other words, ADG is meaningful mainly when the true feasible region is known, as in Figure 3.3 (i). When there is limited knowledge about the true feasible region, SE may be a more meaningful loss function because it prioritizes optimization models that can directly generate the observations $\mathbf{x}_n^{\text{obs}}$. That is why we used SE for our parametric experiments (Figure 3.4). However, SE penalizes *any* difference between the predicted and observed decision variables, even if those differences do not affect optimality. In short, ADG and SE both have conceptual drawbacks, and it may be beneficial to develop new or hybrid loss metrics.

In practice, minimizing the SE loss also appears to be more challenging for steepest descent. To get a sense for the characteristics of ADG versus SE from the point of



(i) ADG loss surface (ii) SE loss surface

Figure 3.5: Loss surfaces for the feasible region and target shown in Figure 1 (i).

view of varying $\mathbf{c}$, consider Figure 3.5, which depicts the loss for the IO problem in Figure 3.1 (i) using both high precision ($\epsilon = 10^{-5}$) and low precision ($\epsilon = 0.1, 0.01$) for IPM. Because the ADG loss is directly dependent on $\mathbf{c}$, the loss varies smoothly even as the corresponding optimum $\mathbf{x}^*$ stays fixed. The SE loss, in contrast, is piece-wise constant; an instantaneous perturbation of $\mathbf{c}$ will almost never change the SE loss in the limit of $\epsilon \to 0$. Note that the gradients derived by implicit differentiation (Amos and Kolter, 2017) indicate $\frac{\partial \ell}{\partial \mathbf{c}} = \mathbf{0}$ everywhere in the linear case, which would mean $\mathbf{c}$ cannot be learned by gradient descent. With IPM one can learn $\mathbf{c}$ nonetheless because the barrier sharpness parameter t smooths the loss, especially at low values. The precision parameter ϵ limits the maximal sharpness during forward optimization, and so the gradient $\frac{\partial \ell}{\partial \mathbf{c}}$ is not zero in practice, especially when ϵ is weak. Notice that the SE loss surface in Figure 3.5 becomes qualitatively smoother for weak ϵ, whereas ADG is not fundamentally changed. Also, when $\mathbf{c}$ is normal to a constraint (when the optimal point is about to transition from one point to another) the gradient $\frac{\partial \ell}{\partial \mathbf{c}}$ explodes even when the problem is smoothed.

Computational efficiency

Our paper is conceptual and focuses on flexibility and the likelihood of success, rather than computational efficiency. Many applications of IO are not real-time, and so we expect methods with running times on the order of seconds or minutes to be of practical use. Researchers may also consider applying gradient-free solvers (Hutter et al., 2011; Snoek et al., 2012) to their IO problem instances. Still, we believe the gradient-based framework can be both flexible and fast.

Deep learning frameworks are GPU accelerated and scale well with the size of an individual forward problem, so large instances are not a concern. A bigger issue for GPUs is solving many small or moderate instances efficiently. Amos and Kolter (2017) developed a batch-mode GPU forward solver to address this issue. We note that PyTorch now also supports batch-mode GPU matrix inverse, which can be used to efficiently run IPM on several small instances in parallel.

What is more concerning for the unrolling strategy is that forward optimization processes can be very deep, with hundreds or thousands of iterations. Backpropagation requires keeping all the intermediate values of the forward pass resident in memory, for later use in the backward pass. The computational cost of backpropagation is comparable to that of the forward process, so there is no asymptotic advantage to skipping the backwards pass. Although memory usage was small in our instances, if the memory usage is linear with depth, then at some depth the unrolling strategy will cease to be practical compared to Amos and Kolter (2017) implicit differentiation approach. However, we observed that for IPM most of the gradient contribution comes from the final few Newton steps before termination. In other words, gradient contributions diminish as backpropagation returns 'deeper' along the central path. This means the gradient can be well-approximated in practice with *truncated backpropagation through time* (see Sutskever (2013) for review), which uses a small constant pool of memory regardless of the number of forward steps that were run (i.e., regardless of depth).

The unrolling approach is convenient and practical, especially during the development and exploration phase of IO research. Once an IO model is proven to work, its implementation can be made more efficient through a number of strategies, including deriving the implicit gradients (Amos and Kolter, 2017) or by asymptotically faster learning algorithms being developed in the deep learning community.

3.7 Conclusion

We developed a deep learning framework for inverse optimization based on backpropagation through an iterative forward optimization process. We illustrate the potential of this framework via an implementation where the forward process is the interior point barrier method. Our results on linear non-parametric and parametric problems show promising performance. To the best of our knowledge, this paper is the first to explicitly connect deep learning and inverse optimization.

Chapter 4

Learning Linear Programs from Optimal Decisions

4.1 Abstract

We propose a flexible gradient-based framework for learning linear programs from optimal decisions. Linear programs are often specified by hand, using prior knowledge of relevant costs and constraints. In some applications, linear programs must instead be learned from observations of optimal decisions. Learning from optimal decisions is a particularly challenging bilevel problem, and much of the related *inverse optimization* literature is dedicated to special cases. We tackle the general problem, learning all parameters jointly while allowing flexible parametrizations of costs, constraints, and loss functions. We also address challenges specific to learning linear programs, such as empty feasible regions and non-unique optimal decisions. Experiments show that our method successfully learns synthetic linear programs and minimum-cost multi-commodity flow instances for which previous methods are not directly applicable. We also provide a fast batch-mode PyTorch implementation of the homogeneous interior point algorithm, which supports gradients by implicit

differentiation or backpropagation.

4.2 Introduction

In linear programming, the goal is to make an optimal decision given a linear objective and subject to linear constraints. Traditionally, a linear program is designed using knowledge of relevant costs and constraints. More recently, methodologies that are data-driven have emerged.

Inverse optimization (IO) (Burton and Toint, 1992; Troutt, 1995; Ahuja and Orlin, 2001), in contrast, learns linear programs from observations of optimal decisions rather than of the costs or constraints themselves. The IO approach is particularly important when observations come from optimizing agents (e.g., experts (Chan et al., 2014; Bärmann et al., 2017) or customers (Dong et al., 2018)) who make near-optimal decisions with respect to their internal (unobserved) optimization models.

From a machine learning perspective, the IO setup is as follows: we are given feature vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$ representing conditions (e.g., time, prices, weather) and we observe the corresponding decision targets $\{\mathbf{x}_1^{\text{obs}}, \mathbf{x}_2^{\text{obs}}, \ldots, \mathbf{x}_N^{\text{obs}}\}$ (e.g., quantities, actions) determined by an unknown optimization process, which in our case is assumed linear. We view IO as the problem of inferring a constrained optimization model that gives identical (or equivalent) decisions, and which generalizes to novel conditions $\mathbf{u}$. The family of candidate models is assumed parametrized by some vector $\mathbf{w}$.

Learning a constrained optimizer that makes the observations both feasible *and* optimal poses multiple challenges that have not been explicitly addressed. We demonstrate this via an example shown in Figure 4.1. In this example, we learn a parametric linear program (PLP), here parametrized by a feature u and weights $\mathbf{w} = (w_1, w_2)$ and using a single training observation $(u_1, \mathbf{x}_1^{\text{obs}})$. The PLP corresponding to three parameter settings $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ are shown, with the cost vector and feasible

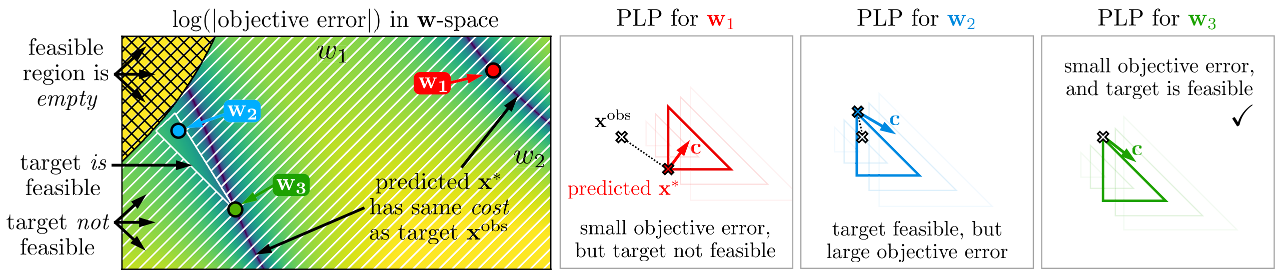

log(|objective error|) in $\mathbf{w}$-space
PLP for $\mathbf{w}_1$
PLP for $\mathbf{w}_2$
PLP for $\mathbf{w}_3$
feasible region is *empty*
target *is* feasible
target *not* feasible
w_1
w_2
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_3$
predicted $\mathbf{x}^*$ has same *cost* as target $\mathbf{x}^{\text{obs}}$
$\mathbf{x}^{\text{obs}}$
$\mathbf{c}$
predicted $\mathbf{x}^*$
small objective error, but target not feasible
target feasible, but large objective error
small objective error, and target is feasible
✓

Figure 4.1: A depiction of our constrained learning formulation.

region corresponding to u_1 emphasized. The goal of learning is to find solutions such as $\mathbf{w}^* = \mathbf{w}_3$. (See Appendix for the specific PLP used in this example.) The parameter setting $\mathbf{w}_1$ in Figure 4.1 makes the observed decision $\mathbf{x}_1^{\text{obs}}$ optimal but not feasible, $\mathbf{w}_2$ produces exactly the opposite result, and some $\mathbf{w}$ values (black-hatched region in Figure 4.1) are not even admissible because they will result in empty feasible regions. Finding a parameter such as $\mathbf{w}_3$ that is consistent with the observations can be difficult. We formulate the learning problem in a novel way, and tackle it with gradient-based methods despite the inherent bilevel nature of learning. Using gradients from backpropagation or implicit differentiation, we successfully learn linear program instances of various sizes as well as learning the costs and right-hand coefficients of a minimum-cost multi-commodity flow problem.

Parametric Linear Programs In a linear program (LP), the values of decision variables $\mathbf{x} \in \mathbb{R}^D$ must be determined, whereas the cost coefficients $\mathbf{c} \in \mathbb{R}^D$, inequality constraint coefficients $\mathbf{A} \in \mathbb{R}^{M_1 \times D}$, $\mathbf{b} \in \mathbb{R}^{M_1}$, and equality constraint coefficients $\mathbf{G} \in \mathbb{R}^{M_2 \times D}$, $\mathbf{h} \in \mathbb{R}^{M_2}$ are all treated as constants. In a *parametric linear program* (PLP), the coefficients (and therefore the optimal decisions) may depend on features $\mathbf{u}$. In order to infer a PLP from data, one may define a suitable hypothesis space parametrized by $\mathbf{w}$. We refer to this hypothesis space as the form of our *forward optimization problem* (FOP).

$$\begin{array}{llll} \min_{\mathbf{x}} & \mathbf{c}^T\mathbf{x} & & \\ \text{s.t.} & \mathbf{A}\mathbf{x} \leq \mathbf{b} & \text{(LP)} \\ & \mathbf{G}\mathbf{x} = \mathbf{h} & \end{array} \quad \begin{array}{lll} \min_{\mathbf{x}} & \mathbf{c}(\mathbf{u})^T\mathbf{x} & \\ \text{s.t.} & \mathbf{A}(\mathbf{u})\mathbf{x} \leq \mathbf{b}(\mathbf{u}) & \text{(PLP)} \\ & \mathbf{G}(\mathbf{u})\mathbf{x} = \mathbf{h}(\mathbf{u}) & \end{array} \quad \begin{array}{lll} \min_{\mathbf{x}} & \mathbf{c}(\mathbf{u},\mathbf{w})^T\mathbf{x} & \\ \text{s.t.} & \mathbf{A}(\mathbf{u},\mathbf{w})\mathbf{x} \leq \mathbf{b}(\mathbf{u},\mathbf{w}) & \text{(FOP)} \\ & \mathbf{G}(\mathbf{u},\mathbf{w})\mathbf{x} = \mathbf{h}(\mathbf{u},\mathbf{w}) & \end{array}$$

A choice of hypothesis $\mathbf{w}$ in (FOP) identifies a PLP, and a subsequent choice of conditions $\mathbf{u}$ identifies an LP. The LP can then be solved to yield an optimal decision $\mathbf{x}^*$ under the model. These predictions of optimal decisions can be compared to observations at training time, or can be used to anticipate optimal decisions under novel conditions $\mathbf{u}$ at test time.

4.3 Related Work

Inverse optimization IO has focused on developing optimization models for minimally adjusting a prior estimate of $\mathbf{c}$ to make a single feasible observation $\mathbf{x}^{\text{obs}}$ optimal (Ahuja and Orlin, 2001; Heuberger, 2004) or for making $\mathbf{x}^{\text{obs}}$ minimally sub-optimal to (LP) without a prior $\mathbf{c}$ (Chan et al., 2014, 2019). Recent work (Babier et al., 2020; Shahmoradi and Lee, 2020) develops exact approaches for imputing non-parametric $\mathbf{c}$ given multiple potentially infeasible solutions to (LP), and to finding non-parametric $\mathbf{A}$ and/or $\mathbf{b}$ (Chan and Kaw, 2020; Ghobadi and Mahmoudzadeh, 2020). In the parametric setting, joint estimation of $\mathbf{A}$ and $\mathbf{c}$ via a maximum likelihood approach was developed by Troutt et al. (2005, 2008) when only $\mathbf{h}$ is a function of $\mathbf{u}$. Saez-Gallego and Morales (2017) jointly learn $\mathbf{c}$ and $\mathbf{b}$ which are affine functions of $\mathbf{u}$. Bärmann et al. (2017, 2020) and Dong et al. (2018) study online versions of inverse linear and convex optimization, respectively, learning a sequence of cost functions where the feasible set for each observation are assumed to be fully-specified. In Chapter 3, we propose a gradient-based approach for learning costs and constraints of a PLP, inspired by deep learning: they 'unroll' a barrier interior point solver and backpropagate through its steps. For certain loss functions, their formulation is susceptible to the situation depicted in Figure 4.1 as '$\mathbf{w}_1$'.

In inverse convex optimization, the focus has been on imputing parametric cost functions while assuming that the feasible region is known for each $\mathbf{u}_i$ (Keshavarz et al., 2011; Bertsimas et al., 2015; Aswani et al., 2018; Esfahani et al., 2018), usually under assumptions of a convex set of admissible $\mathbf{u}$, the objective and/or constraints being convex in $\mathbf{u}$, and uniqueness of the optimal solution for every $\mathbf{u}$. Furthermore, since the feasible region is fixed for each $\mathbf{u}$, it is assumed to be non-empty and bounded, unlike for our work. Although our work focuses on linear programming, it is otherwise substantially more general, allowing for learning of all cost and constraint coefficients simultaneously with no convexity assumptions related to $\mathbf{u}$, no restrictions on the existence of multiple optima, and explicit handling of empty or unbounded feasible regions.

Optimization task-based learning Kao et al. (2009) introduces the concept of directed regression, where the goal is to fit a linear regression model while minimizing the decision loss, calculated with respect to an unconstrained quadratic optimization model. Donti et al. (2017) use a neural network approach to minimize a task loss which is calculated as a function of the optimal decisions in the context of stochastic programming. Elmachtoub and Grigas (2019) propose the "Smart Predict-then-Optimize" framework in which the goal is to predict the cost coefficients of a linear program with a fixed feasible region given past observations of features and true costs, i.e., given $(\mathbf{u}_i, \mathbf{c}_i)$. Note that knowing $\mathbf{c}_i$ in this case implies we can solve for $\mathbf{x}_i^*$, so our framework can in principle be applied in their setting but not vice versa. Our framework is still amenable to more 'direct' data-driven prior knowledge: if in addition to $(\mathbf{u}_i, \mathbf{x}_i^*)$ we have partial or complete observations of $\mathbf{c}_i$ or of constraint coefficients, regressing to these targets can easily be incorporated into our overall learning objective.

Structured prediction In structured output prediction (Taskar et al., 2005; BakIr et al., 2007; Nowozin et al., 2014; Daumé III et al., 2015), each prediction is $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{X}(\mathbf{u})} f(\mathbf{x}, \mathbf{u}, \mathbf{w})$ for an objective f and known output structure $\mathcal{X}(\mathbf{u})$. In our work the structure is also learned, parametrized as $\mathcal{X}(\mathbf{u}, \mathbf{w}) = \{\, \mathbf{x} \mid \mathbf{A}(\mathbf{u}, \mathbf{w})\mathbf{x} \leq \mathbf{b}(\mathbf{u}, \mathbf{w}),\, \mathbf{G}(\mathbf{u}, \mathbf{w})\mathbf{x} = \mathbf{h}(\mathbf{u}, \mathbf{w}) \,\}$, and the objective is linear $f(\mathbf{x}, \mathbf{u}, \mathbf{w}) = \mathbf{c}(\mathbf{u}, \mathbf{w})^T\mathbf{x}$. In structured prediction the loss ℓ is typically a function of $\mathbf{x}^*$ and a target $\bar{\mathbf{x}}$, whereas in our setting it is important to consider a parametric loss $\ell(\mathbf{x}^*, \bar{\mathbf{x}}, \mathbf{u}, \mathbf{w})$.

Differentiating through optimization Our work involves differentiating through an LP. Bengio (2000) proposed gradient-based tuning of neural network hyperparameters and, in a special case, backpropagating through the Cholesky decomposition computed during training (suggested by Léon Bottou). Stoyanov et al. (2011) proposed backpropagating through a truncated loopy belief propagation procedure. Domke (2012, 2013) proposed automatic differentiation through truncated optimization procedures more generally, and Maclaurin et al. (2015b) proposed a similar approach for hyperparameter search. The continuity and differentiability of the optimal solution set of a quadratic program has been extensively studied (Lee et al., 2006). Amos and Kolter (2017) recently proposed integrating a quadratic optimization layer in a deep neural network, and used implicit differentiation to derive a procedure for computing parameter gradients. As part of our work we specialize their approach, providing an expression for LPs. Even more general is recent work on differentiating through convex cone programs (Agrawal et al., 2019), submodular optimization (Djolonga and Krause, 2017), and arbitrary constrained optimization (Gould et al., 2019). There are also versatile perturbation-based differentiation techniques (Papandreou and Yuille, 2011; Berthet et al., 2020).

4.4 Methodology

Here we introduce our new bilevel formulation and methodology for learning parametric linear programs. Unlike previous approaches (e.g., Aswani et al. (2018)), we do not transform the problem to a single-level formulation, and so we do not require simplifying assumptions. We propose a technique for tackling our bilevel formulation with gradient-based non-linear programming methods.

4.4.1 Inverse Optimization as PLP Model Fitting

Let $\{(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}})\}_{i=1}^N$ denote the training set. A loss function $\ell(\mathbf{x}^*, \mathbf{x}^{\text{obs}}, \mathbf{u}, \mathbf{w})$ penalizes discrepancy between prediction $\mathbf{x}^*$ and target $\mathbf{x}^{\text{obs}}$ under conditions $\mathbf{u}$ for the PLP hypothesis identified by $\mathbf{w}$. Note that if $\mathbf{x}_i^{\text{obs}}$ is optimal under conditions $\mathbf{u}_i$, then $\mathbf{x}_i^{\text{obs}}$ must also be feasible. We therefore propose the following bilevel formulation of the *inverse linear optimization problem* (ILOP):

$$\underset{\mathbf{w}\in\mathcal{W}}{\text{minimize}} \quad \tfrac{1}{N}\textstyle\sum_{i=1}^N \ell(\mathbf{x}_i^*, \mathbf{x}_i^{\text{obs}}, \mathbf{u}_i, \mathbf{w}) + r(\mathbf{w}) \tag{ILOP}$$

$$\text{subject to} \quad \mathbf{A}(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} \le \mathbf{b}(\mathbf{u}_i, \mathbf{w}), \quad \mathbf{G}(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} = \mathbf{h}(\mathbf{u}_i, \mathbf{w}), \qquad i = 1, \dots, N \tag{4.1a}$$

$$\mathbf{x}_i^* \in \arg\min_{\mathbf{x}} \left\{ \mathbf{c}(\mathbf{u}_i, \mathbf{w})^T\mathbf{x} \;\middle|\; \begin{array}{l} \mathbf{A}(\mathbf{u}_i, \mathbf{w})\mathbf{x} \le \mathbf{b}(\mathbf{u}_i, \mathbf{w}) \\ \mathbf{G}(\mathbf{u}_i, \mathbf{w})\mathbf{x} = \mathbf{h}(\mathbf{u}_i, \mathbf{w}) \end{array} \right\}, \quad i = 1, \dots, N \tag{4.1b}$$

where $r(\mathbf{w})$ denotes an optional regularization term such as $r(\mathbf{w}) = \|\mathbf{w}\|^2$ and $\mathcal{W} \subseteq \mathbb{R}^K$ denotes additional problem-specific prior knowledge, if applicable (similar constraints are standard in the IO literature (Keshavarz et al., 2011; Chan et al., 2019)). The 'inner' problem (4.1b) generates predictions $\mathbf{x}_i^*$ by solving N independent LPs. The 'outer' problem tries to make these predictions consistent with the targets $\mathbf{x}_i^*$ while also satisfying target feasibility (4.1a).

Difficulties may arise, in principle and in practice. An inner LP may be infeasible or unbounded for certain $\mathbf{w} \in \mathcal{W}$, making ℓ undefined. Even if all $\mathbf{w} \in \mathcal{W}$ produce feasible and bounded LPs, an algorithm for solving (ILOP) may still attempt to query $\mathbf{w} \notin \mathcal{W}$. The outer problem as a whole may be subject to local minima due to non-convex objective and/or constraints, depending on the problem-specific parametrizations. We propose gradient-based techniques for the outer problem (Section 4.4.2), but $\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{w}}$ may not exist or may be non-unique at certain $\mathbf{u}_i$ and $\mathbf{w}$ (Section 4.4.3).

Nonetheless, we find that tackling this formulation leads to practical algorithms. To the best of our knowledge, our proposed (ILOP) formulation is the most general model of inverse linear parametric programming. The formulation subsumes cases that are non-parametric, or parametric only in $\mathbf{u}$, that have received much interest in the IO literature. It has not been proposed in work focused purely on differentiation, such as that of Amos and Kolter (2017) or of Agrawal et al. (2019).

Choice of loss function The IO literature considers *decision error*, which penalizes difference in decision variables, and *objective error*, which penalizes difference in optimal objective value (Babier et al., 2020). A fundamental issue with decision error, such as *squared decision error* (SDE) $\ell(\mathbf{x}^*, \mathbf{x}^{\text{obs}}) = \frac{1}{2}\|\mathbf{x}_i^* - \mathbf{x}_i^{\text{obs}}\|^2$, is that when $\mathbf{x}^*$ is non-unique the loss is also not unique; this issue was also a motivation for the "Smart Predict-then-Optimize" paper (Elmachtoub and Grigas, 2019). An objective error, such as *absolute objective error* (AOE) $\ell(\mathbf{x}^*, \mathbf{x}^{\text{obs}}, \mathbf{c}) = |\mathbf{c}^T(\mathbf{x}_i^{\text{obs}} - \mathbf{x}_i^*)|$, is unique even if $\mathbf{x}^*$ is not. We evaluate AOE using imputed cost $\mathbf{c}(\mathbf{u}, \mathbf{w})$ during training; doing so requires at least some prior knowledge $\mathcal{W}$ to avoid trivial cost vectors, as in Keshavarz et al. (2011).

Target feasibility Constraints (4.1a) explicitly enforce target feasibility $\mathbf{A}\mathbf{x}_i^{\text{obs}} \leq \mathbf{b}$, $\mathbf{G}\mathbf{x}_i^{\text{obs}} = \mathbf{h}$ in any learned PLP. The importance of these constraints can be understood through Figure 4.1, where hypothesis $\mathbf{w}_1$ achieves AOE$=0$ since $\mathbf{x}^{\text{obs}}$ and $\mathbf{x}^*$ are on the same hyperplane, despite $\mathbf{x}^{\text{obs}}$ being infeasible. Chan et al. (2019) show that if the feasible region is bounded then for any infeasible $\mathbf{x}^{\text{obs}}$ there exists a cost vector achieving AOE$=0$.

Unbounded or infeasible subproblems Despite (4.1a), an algorithm for solving (ILOP) may query a $\mathbf{w}$ for which an LP in (4.1b) is itself infeasible or unbounded, in which case a finite $\mathbf{x}^*$ is not defined. We can extend (ILOP) to explicitly account for these special cases (by penalizing a measure of infeasibility (Murty et al., 2000), and penalizing unbounded directions when detected) but in our experiments simply evaluating the (large) loss for an arbitrary $\mathbf{x}^*$ returned by our interior point solver worked nearly as well at avoiding such regions of $\mathcal{W}$, so we opt to keep the formulation simple.

Noisy observations Formulation (ILOP) can be extended to handle measurement noise. For example, individually penalized non-negative slack variables can be added to the right-hand sides of (4.1a) as in a soft-margin SVM (Cortes and Vapnik, 1995). Alternatively, a norm-penalized group of slack variables can be added to each $\mathbf{x}_i^{\text{obs}}$ on the left-hand side of (4.1a), softening targets in decision space. We leave investigation of noisy data and model-misspecification as future work.

4.4.2 Learning Linear Programs with Sequential Quadratic Programming

We treat (ILOP) as a *non-linear programming* (NLP) problem, making as few assumptions as possible. We focus on *sequential quadratic programming* (SQP), which aims to solve NLP problems iteratively. Given current iterate $\mathbf{w}^k$, SQP determines a search direction $\boldsymbol{\delta}^k$ and then selects the next iterate $\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha\boldsymbol{\delta}^k$ via line search on $\alpha > 0$. Direction $\boldsymbol{\delta}^k$ is the solution to a quadratic program.

$$\begin{aligned} \text{minimize}_{\mathbf{w}} \quad & f(\mathbf{w}) \\ \text{subject to} \quad & \mathbf{g}(\mathbf{w}) \leq \mathbf{0} \qquad \text{(NLP)} \\ & \mathbf{h}(\mathbf{w}) = \mathbf{0} \end{aligned} \qquad \begin{aligned} \text{minimize}_{\boldsymbol{\delta}} \quad & \nabla f(\mathbf{w}^k)^T\boldsymbol{\delta} + \boldsymbol{\delta}^T\mathbf{B}^k\boldsymbol{\delta} \\ \text{subject to} \quad & \nabla\mathbf{g}(\mathbf{w}^k)^T\boldsymbol{\delta} + \mathbf{g}(\mathbf{w}^k) \leq \mathbf{0} \qquad \text{(SQP)} \\ & \nabla\mathbf{h}(\mathbf{w}^k)^T\boldsymbol{\delta} + \mathbf{h}(\mathbf{w}^k) = \mathbf{0} \end{aligned}$$

Each instance of subproblem (SQP) requires evaluating constraints[1] and their gradients at $\mathbf{w}^k$, as well as the gradient of the objective. Matrix $\mathbf{B}^k$ approximates the Hessian of the Lagrange function for (NLP), where $\mathbf{B}^{k+1}$ is typically determined from the gradients by a BFGS-like update. Our experiments use an efficient variant called *sequential least squares programming* (SLSQP) (Schittkowski, 1982; Kraft, 1988) which exploits a stable LDL factorization of $\mathbf{B}$.

The NLP formulation of (ILOP) has NM_1 inequality and NM_2 equality constraints from (4.1a):

$$\mathbf{g}(\mathbf{w}) = \Big[\mathbf{A}(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} - \mathbf{b}(\mathbf{u}_i, \mathbf{w})\Big]_{i=1}^{N}, \qquad \mathbf{h}(\mathbf{w}) = \Big[\mathbf{G}(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} - \mathbf{h}(\mathbf{u}_i, \mathbf{w})\Big]_{i=1}^{N},$$

plus any constraints needed to enforce $\mathbf{w} \in \mathcal{W}$. The NLP constraint residuals and their gradients $\nabla\mathbf{g}(\mathbf{w}), \nabla\mathbf{h}(\mathbf{w})$ can be directly evaluated. Evaluating

[1]NLP constraint vector $\mathbf{h}(\mathbf{w})$ is not the same as FOP right-hand side $\mathbf{h}(\mathbf{u}, \mathbf{w})$, despite same symbol.

$f(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{x}_i^*, \mathbf{x}_i^{\text{obs}}, \mathbf{u}_i, \mathbf{w}) + r(\mathbf{w})$ requires solving each LP in (4.1b). Finally, evaluating $\nabla f(\mathbf{w})$ requires evaluating the vector-Jacobian product term in $\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{w}} = \frac{\partial \ell}{\partial \mathbf{w}} + \frac{\partial \ell}{\partial \mathbf{x}_i^*}\frac{\partial \mathbf{x}_i^*}{\partial \mathbf{w}}$ for each i, which requires differentiating through the LP optimization that produced $\mathbf{x}_i^*$ from $\mathbf{u}_i$ and $\mathbf{w}$. Differentiating through the LP allows us to tackle (ILOP) directly in its bilevel form, using powerful gradient-based NLP algorithms such as SQP as the 'outer' solver. Section 4.4.3 compares methods for differentiating through an LP optimization.

Redundant NLP constraints When PLP model parameters $\mathbf{w}$ have fixed dimension, the NLP formulation of (ILOP) can involve many redundant constraints, roughly in proportion to N. Indeed, if $\mathcal{W} \subseteq \mathbb{R}^K$ and $K < NM_2$, the equality constraints may appear to over-determine $\mathbf{w}$, treating (NLP) as a feasibility problem; but, due to redundancy in (4.1a), $\mathbf{w}$ is not uniquely determined. The ease or difficulty of removing redundant constraints from (NLP) depends on the domain-specific parametrizations of PLP constraints $\mathbf{A}(\mathbf{u}, \mathbf{w}), \mathbf{b}(\mathbf{u}, \mathbf{w}), \mathbf{G}(\mathbf{u}, \mathbf{w})$, and $\mathbf{h}(\mathbf{u}, \mathbf{w})$. Equality constraints that are affinely-dependent on $\mathbf{w}$ can be eliminated from (NLP) by a pseudoinverse technique, resulting in a lower-dimensional problem; this technique also handles the case where (NLP) is not strictly feasible in $\mathbf{h}(\mathbf{w}) = \mathbf{0}$ (either due to noisy observations or model misspecification) by automatically searching only among $\mathbf{w}$ that exactly minimize the sum of squared residuals $\|\mathbf{h}(\mathbf{w})\|^2$. If equality constraints are polynomially-dependent on $\mathbf{w}$, we can eliminate redundancy by Gröbner basis techniques (Cox et al., 2013) although, unlike the affine case, it may not be possible or beneficial to reparametrize-out the new non-redundant basis constraints from the NLP. Redundant inequality constraints can be either trivial or costly to identify (Telgen, 1983), but are not generally problematic for SQP algorithms. See Appendix for details.

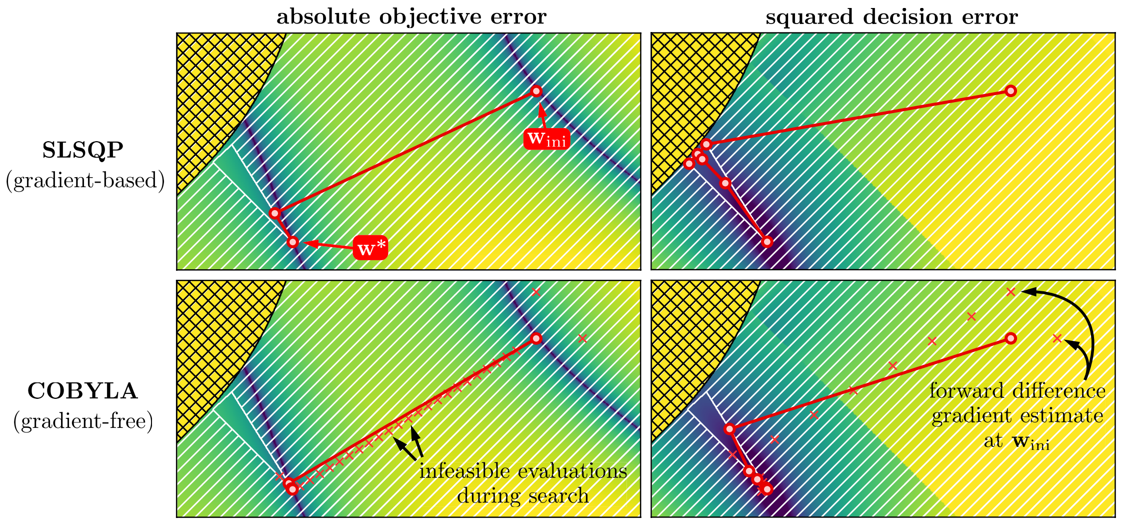


Figure 4.2: An illustration of how SLSQP and COBYLA solve the learning problem in Figure 4.1 for the AOE and SDE loss functions. Each algorithm first tries to satisfy the NLP constraints $\mathbf{g}(\mathbf{w}) \leq \mathbf{0}$ (triangle-shaped feasible region in $\mathbf{w}$-space), then makes progress minimizing $f(\mathbf{w})$.

Benefit over gradient-free methods Evaluating $f(\mathbf{w})$ is expensive in our NLP because it requires solving N linear programs. To understand why access to $\nabla f(\mathbf{w})$ is important in this scenario, it helps to contrast SQP with a well-known gradient-free NLP optimizer such as COBYLA (Powell, 1994). For K-dimensional NLP, COBYLA maintains $K+1$ samples of $f(\mathbf{w}), \mathbf{g}(\mathbf{w}), \mathbf{h}(\mathbf{w})$ and uses them as a finite-difference approximation to $\nabla f(\mathbf{w}^k), \nabla g(\mathbf{w}^k), \nabla h(\mathbf{w}^k)$ where $\mathbf{w}^k$ is the current iterate (best sample). The next iterate $\mathbf{w}^{k+1}$ is computed by optimizing over a trust region centered at $\mathbf{w}^k$. COBYLA recycles past samples to effectively estimate 'coarse' gradients, whereas SQP uses gradients directly. Figure 4.2 shows SLSQP and COBYLA running on the example from Figure 4.1.

4.4.3 Computing Loss Function and its Gradients

If, at a particular point $(\mathbf{u}_i, \mathbf{w})$, each corresponding vector-Jacobian product $\frac{\partial \ell}{\partial \mathbf{x}_i^*} \frac{\partial \mathbf{x}_i^*}{\partial \mathbf{w}}$ exists, is unique, and can be computed, then we can construct (SQP) at each step. For convenience, we assume that $(\mathbf{c}, \mathbf{A}, \mathbf{b}, \mathbf{G}, \mathbf{h})$ are expressed in terms of $(\mathbf{u}, \mathbf{w})$ within an automatic differentiation framework such as PyTorch, so all that remains is to compute Jacobians $(\frac{\partial \ell}{\partial \mathbf{c}}, \frac{\partial \ell}{\partial \mathbf{A}}, \frac{\partial \ell}{\partial \mathbf{b}}, \frac{\partial \ell}{\partial \mathbf{G}}, \frac{\partial \ell}{\partial \mathbf{h}})$ at each $(\mathbf{u}_i, \mathbf{w})$ as an intermediate step at the outset of computing $\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{w}}$. We consider four approaches:

backprop: backpropagate through the steps of the homogeneous interior point algorithm for LPs,

implicit: the implicit differentiation procedure of Amos and Kolter (2017) specialized to LPs,

direct: evaluate gradients directly, in closed form (for objective error only), and

cvx: use a *cvxpylayer* (Agrawal et al., 2019) for LP solve and for implicit differentiation.

To implement the first three approaches, we developed a batch PyTorch version of the homogeneous interior point algorithm (Andersen and Andersen, 2000; Xu et al., 1996); this algorithm was originally developed for the MOSEK optimization suite and is currently the default linear programming solver in SciPy (Virtanen et al., 2020). Our backprop implementation is also efficient, for example re-using the *LU* decompositionfrom each Newton step.

For implicit differentiation we follow Amos and Kolter (2017) by forming the system of linear equations that result from differentiating the KKT conditions and then inverting that system to compute the needed vector-Jacobian products. For LPs this system can be poorly conditioned, especially at strict tolerances on the LP solver, but in practice it provides useful gradients. The *cvx* approach is similar but is implemented by a *cvxpylayer*, which in turn relies on a fast conic solver (O'Donoghue et al., 2016) for the forward problem and an implicit differentiation procedure similar to the work of Amos and Kolter (2017) for the gradients.

For direct gradients (in the case of objective error), we use Theorem 4.1.

Theorem 4.1 *Let $\mathbf{x}^* \in \mathbb{R}^D$ be an optimal solution to* (LP) *and let $\boldsymbol{\lambda}^* \in \mathbb{R}^{M_1}_{\leq 0}, \boldsymbol{\nu}^* \in \mathbb{R}^{M_2}$ be an optimal solution to the associated dual linear program. If $\mathbf{x}^*$ is non-degenerate then the objective error $z = \mathbf{c}^T(\mathbf{x}^{\text{obs}} - \mathbf{x}^*)$ is differentiable and the total derivatives*[2] *are*

$$\frac{\partial z}{\partial \mathbf{c}} = \left(\mathbf{x}^{\text{obs}} - \mathbf{x}^*\right)^T \qquad \frac{\partial z}{\partial \mathbf{A}} = \boldsymbol{\lambda}^* \mathbf{x}^{*T} \qquad \frac{\partial z}{\partial \mathbf{b}} = -\boldsymbol{\lambda}^{*T} \qquad \frac{\partial z}{\partial \mathbf{G}} = \boldsymbol{\nu}^* \mathbf{x}^{*T} \qquad \frac{\partial z}{\partial \mathbf{h}} = -\boldsymbol{\nu}^{*T}.$$

When ℓ is AOE loss, by chain rule we can multiply each quantity by $\frac{\partial \ell}{\partial z} = \text{sign}(z)$ to get the needed Jacobians. Gradients $\frac{\partial z}{\partial \mathbf{b}}$ and $\frac{\partial z}{\partial \mathbf{h}}$ for the right-hand sides are already well-known as *shadow prices.* If $\mathbf{x}^*$ is degenerate then the relationship between shadow prices and dual variables breaks down, resulting in two-sided shadow prices (Strum, 1969; Aucamp and Steinberg, 1982).

We use degeneracy in the sense of Tijssen and Sierksma (1998) (see Appendix), where a point on the relative interior of the optimal face need not be degenerate, even if there exists a degenerate vertex on the optimal face. This matters when $\mathbf{x}^*$ is non-unique because interior point methods typically converge to the analytical center of the relative interior of the optimal face (Zhang, 1994). Tijssen and Sierskma also give relations between degeneracy of $\mathbf{x}^*$ and uniqueness of $\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*$, which we apply in Corollary 4.1. When the gradients are non-unique, this corresponds to the subdifferentiable case.

Corollary 4.1 *In Theorem 4.1, both $\frac{\partial z}{\partial \mathbf{b}}$ and $\frac{\partial z}{\partial \mathbf{h}}$ are unique, $\frac{\partial z}{\partial \mathbf{c}}$ is unique if and only if $\mathbf{x}^*$ is unique, and both $\frac{\partial z}{\partial \mathbf{A}}$ and $\frac{\partial z}{\partial \mathbf{G}}$ are unique if and only if $\mathbf{x}^*$ is unique or $\mathbf{c} = \mathbf{0}$.*

[2]In a slight abuse of notation, we ignore leading singleton dimension of $\frac{\partial z}{\partial \mathbf{A}} \in \mathbb{R}^{1 \times M_1 \times D}$, $\frac{\partial z}{\partial \mathbf{G}} \in \mathbb{R}^{1 \times M_2 \times D}$.

4.5 Experiments

We evaluate our approach by learning a range of synthetic LPs and parametric instances of minimum-cost multi-commodity flow problems. Use of synthetic instances is common in IO (e.g., Ahuja and Orlin (2001); Keshavarz et al. (2011); Dong et al. (2018)) and there are no community-established and readily-available benchmarks, especially for more general formulations. Our experimental study considers instances not directly addressable by previous IO work, either because we learn all coefficients jointly or because the parametrization results in non-convex NLP.

We compare four versions of our gradient-based method ($\text{SQP}_{\text{bprop}}$, SQP_{impl}, SQP_{dir}, SQP_{cvx}) with two gradient-free methods: random search (RS) and COBYLA. A gradient-free baseline is applicable to (ILOP) only if it (i) supports general bilevel natively, or (ii) allows the objective and constraints to be specified by callbacks. COBYLA is conceptually similar to SQP and can be readily applied to (ILOP), but many otherwise-powerful solvers such as BARON (Sahinidis, 2017), CPLEX (IBM, 2020) and Gurobi (Gurobi Optimization, 2020) cannot.

Complete experimental results for synthetic LPs are presented in Figures 4.3, 4.8, 4.9, 4.10. The main observation is that the gradient-based methods perform similarly and become superior to gradient-free methods as the dimension K of parametrization $\mathbf{w}$ increases. We find that including a black-box baseline like COBYLA is important for assessing the practical difficulty of an IO instance (and encourage future papers to do so) because such methods work reasonably well in low-dimensional problems. A second observation is that there are instances for which no method succeeds at minimizing training error 100% of the time. Our method can therefore be viewed as a way to boost the probability of successful training when combined with simple global optimization strategies such as multi-start.

Experiments used PyTorch v1.6 nightly build, the COBYLA and SLSQP

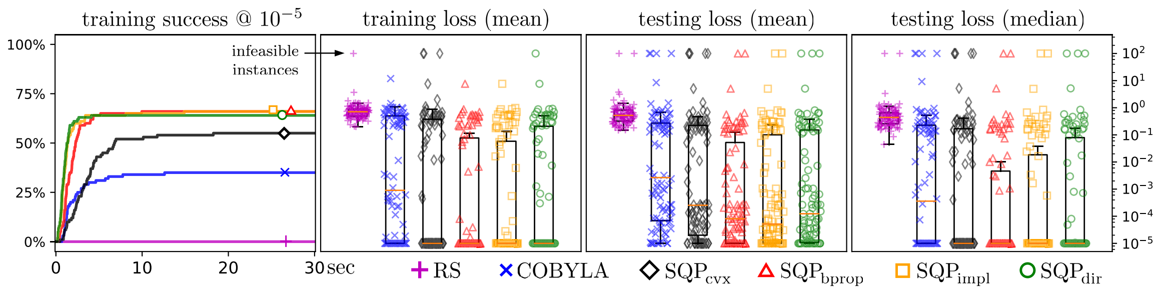


Figure 4.3: A comparison on synthetic PLP instances having $D = 10$ decision variables and $M_1 = 36$ inequality constraints.

wrappers from SciPy v1.4.1, and were run on an Intel Core i7 with 16GB RAM. (We do not use GPUs, though our PyTorch interior point solver inherits GPU acceleration.) We do not regularize $\mathbf{w}$ nor have any other hyperparameters. Code to reproduce experiments is available at `https://github.com/yingcongtan/ilop`.

Learning linear programs We used the LP generator from Chapter 3, modifying it to create a more challenging variety of feasible regions; their code did not perform competitively in terms of runtime or success rate on these harder instances, and is not effective at learning constraints under an AOE loss. Fig. 4.3 shows the task of learning $(\mathbf{c}, \mathbf{A}, \mathbf{b})$ with a $K = 6$ dimensional parametrization $\mathbf{w}$, a $D = 10$ dimensional decision space $\mathbf{x}$, and 20 training observations. Curves in Fig. 4.3 show the probability over time of achieving AOE training loss $\ell(\mathbf{w})$ below a tolerance threshold of 10^{-5}. Box plots show final training and testing loss of 100 different trial instances, each with 20 training and 20 testing points (distinct $\mathbf{u}$ values). We evaluate the "testing loss" for AOE with respect to the 'true' cost $\mathbf{c}(\mathbf{u})$, never the imputed cost. The median loss over a set of testing points tends to be smaller than their mean; see Section 4.6 for discussion. We make the following observations. RS fails; COBYLA 'succeeds' on ~30% of instances; $\text{SQP}_{\text{bprop}}$, SQP_{impl}, SQP_{dir} succeeds on ~60–65%, which is substantially better. The success curve of $\text{SQP}_{\text{bprop}}$ slightly lags those of SQP_{impl} and SQP_{dir} due to the overhead of backpropagating through the steps of the interior point solver.

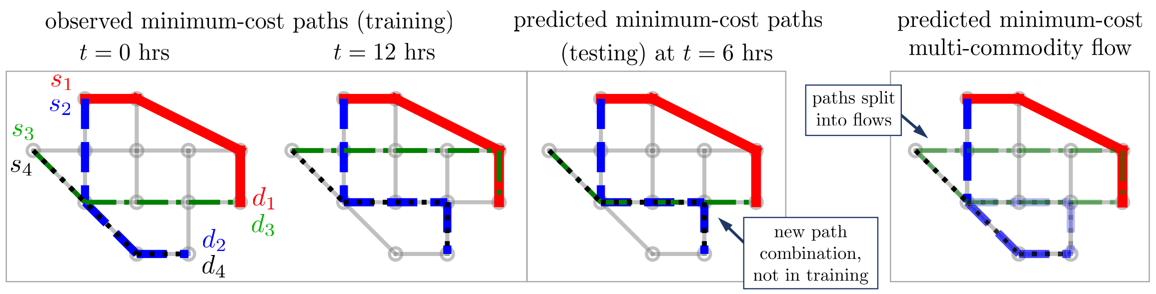

observed minimum-cost paths (training)
$t = 0$ hrs
$t = 12$ hrs
predicted minimum-cost paths (testing) at $t = 6$ hrs
predicted minimum-cost multi-commodity flow
s_1
s_2
s_3
s_4
d_1
d_3
d_2
d_4
new path combination, not in training
paths split into flows

Figure 4.4: A visualization of minimum-cost paths (for simplicity) and minimum-cost multi-commodity flows (our experiment) on the Nguyen-Dupuis network.

SQP$_{\text{cvx}}$ performs worse in these instances due to the speed at which the internal *scs* solver (written in C) could reach the inner tolerance (10^{-8}), and not due to any overhead. See Appendix for five additional problem sizes, where overall the conclusions are the same. On instances with equality constraints, where we learn ($\mathbf{c}$, $\mathbf{A}$, $\mathbf{b}$, $\mathbf{G}$, $\mathbf{h}$) jointly, performance was similar to the above (see Figure 4.9 in Appendix).

Much of the IO literature is focused on learning coefficients of $\mathbf{c}$ and/or $\mathbf{b}$ directly, often from a single training target $\mathbf{x}^{\text{obs}}$, i.e., learning a single LP rather than a PLP. In our formulation, we can learn coefficients of $(\mathbf{c}, \mathbf{A}, \mathbf{b}, \mathbf{G}, \mathbf{h})$ jointly by concatenating them into $\mathbf{w}$. For example, an instance with $D = 10$, $M_1 = 80$, $M_2 = 0$ has 890 adjustable parameters. In Figure 4.10 of Appendix, we show SQP$_{\text{bprop}}$, SQP$_{\text{impl}}$ and SQP$_{\text{dir}}$ consistently achieve zero AOE training loss on such problems, whereas RS and COBYLA fail to make learning progress given the same time budget. SQP$_{\text{cvx}}$ makes progress, but is slower than the other gradient-based implementations.

Learning minimum-cost multi-commodity flow problems Fig. 4.4 shows a visualization of our experiment on the Nguyen-Dupuis graph (Nguyen and Dupuis, 1984a). Sources $\{s_1, s_2, s_3, s_4\}$ and destinations $\{d_1, d_2, d_3, d_4\}$ are shown. At left are two example sets of training paths $\{(t_1, \mathbf{x}_1^{\text{obs}}), (t_2, \mathbf{x}_2^{\text{obs}})\}$ alongside an example of a correctly predicted set of optimal paths under different conditions (different t). At right is a visualization of a correctly predicted optimal flow, where color intensity

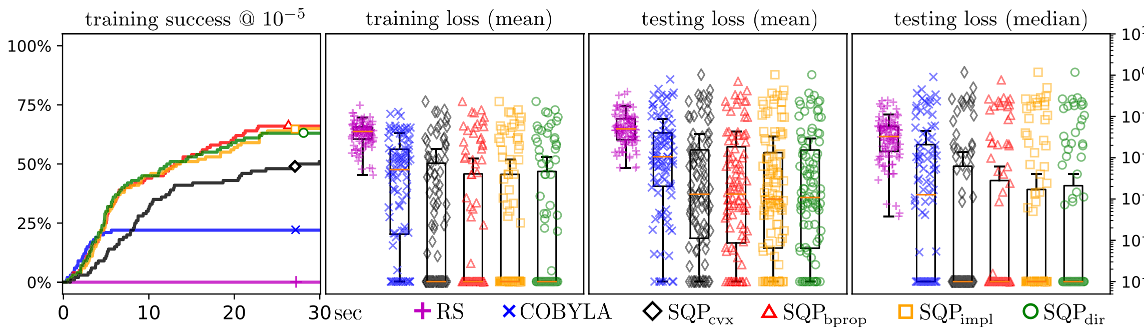


Figure 4.5: A comparison on minimum-cost multi-commodity flow instances, similar to Fig. 4.3.

indicates proportion of flow along arcs.

We learn a periodic arc cost $c_j(t, l_j, p_j) = l_j + w_1 p_j + w_2 l_j(\sin(2\pi(w_3 + w_4 t + w_5 l_j)) + 1)$ and an affine arc capacity $b_j(l_j) = 1 + w_6 + w_7 l_j$, based on global feature t (time of day) and arc-specific features l_j (length) and p_j (toll price). To avoid trivial solutions, we set $\mathcal{W} = \{\mathbf{w} \geq \mathbf{0}, w_3 + w_4 + w_5 = 1\}$. Results on 100 instances are shown in Fig. 4.5. The SQP methods outperform RS and COBYLA in training and testing loss. From an IO perspective the fact that we are jointly learning costs and capacities in a non-convex NLP formulation is already quite general. SQP$_{\text{cvx}}$ is still slower, but more competitive.

4.6 Discussion

Generalizing is hard We report both the mean and median loss over the testing points in each trial. The difference in mean and median testing error is due to the presence of a few 'outliers' among the otherwise-small test set errors.

Fig. 4.6 shows the optimal decision map $\mathbf{u} \mapsto \mathbf{x}^*$ for a ground-truth PLP (a) and learned PLP (b) with the value of components (x_1^*, x_2^*) represented by red and green intensity respectively, along with that of a PLP trained on $\{\mathbf{u}_1, \mathbf{u}_2\}$. The learned PLP has no training error (SDE$=0$, AOE$=0$) but large test error (SDE$=.89$, AOE$=.22$)

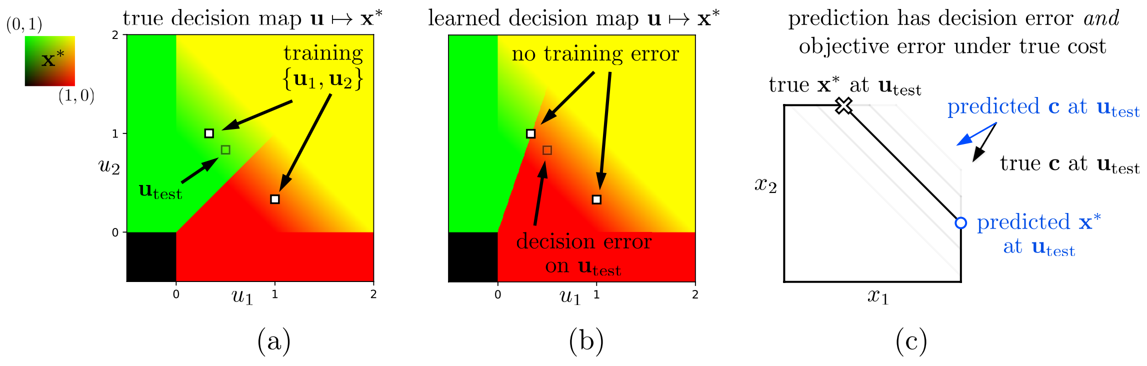

(0, 1)
$\mathbf{x}^*$
(1, 0)
true decision map $\mathbf{u} \mapsto \mathbf{x}^*$
training $\{\mathbf{u}_1, \mathbf{u}_2\}$
$\mathbf{u}_\text{test}$
u_2
u_1
learned decision map $\mathbf{u} \mapsto \mathbf{x}^*$
no training error
decision error on $\mathbf{u}_\text{test}$
prediction has decision error *and* objective error under true cost
true $\mathbf{x}^*$ at $\mathbf{u}_\text{test}$
predicted $\mathbf{c}$ at $\mathbf{u}_\text{test}$
true $\mathbf{c}$ at $\mathbf{u}_\text{test}$
predicted $\mathbf{x}^*$ at $\mathbf{u}_\text{test}$
x_2
x_1
(a)
(b)
(c)

Figure 4.6: A failure to generalize in a learned PLP.

as depicted in (c). (See Appendix for the specific PLP used in this example.)

Fig. 4.6 shows the nature of this failure to generalize: the decision map $\mathbf{u} \mapsto \mathbf{x}^*$ of a PLP has discontinuities, so the training data can easily under-specify the set of learned models that can achieve zero training loss; this is similar to the scenario that motivates the max-margin learning principle, used for good generalization in SVMs. It is not clear what forms of regularization $r(\mathbf{w})$ might reliably improve generalization in IO. Fig. 4.6 also suggests that training points which closely straddle discontinuities are much more 'valuable' from a learning perspective.

Scalability We wish to highlight the scalability of direct gradients, and of our bilevel approach more generally. First, our experiments use a general-purpose LP solver where forward solve dominates runtime. In scenarios where forward solve is fast, for example by an application-specific algorithm (max-flow, matching, etc.) or a fast re-solve strategy, the gradient computation can be proportionally significant. In that case, direct evaluation of the gradient scales much better than solving backprop or solving the system required by implicit differentiation (Amos and Kolter, 2017; Agrawal et al., 2019). Shown at right are compute times for an LP parameter gradient averaged over the 100 instances from Fig. 4.5, with 'direct' being at least ~50x faster than alternatives.

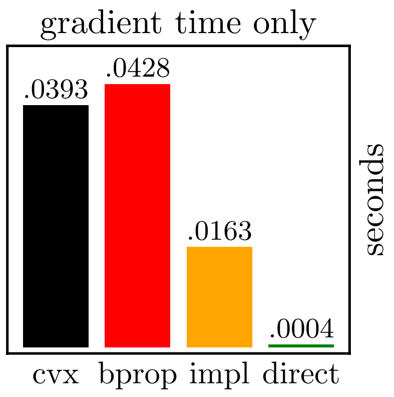


Figure 4.7: Runtime for computing gradients.

Second, note that other IO approaches often convert a bilevel problem into a new single-level one (Aswani et al., 2018). This strategy cannot exploit fast algorithms for specialized forward problems (i.e., the inner problem of our formulation) and must rely on general-purpose machinery like CPLEX. By retaining the bilevel nature of (ILOP), our approach allows specialized algorithms to be used for the forward problem, and fast gradients whenever optimal primal and dual solutions can be recovered.

Generality and applicability Our work generalizes in some respects, and specializes others. Considering only literature on inverse *linear* optimization, our (ILOP) formulation generalizes prior work in that it tackles fully parametric PLPs and arbitrary number of observations. Our methodology is meanwhile a novel extension of Chapter 3 since here we introduce 'outer' constraints, SQP-based training, a new gradient computation method, and a faster forward solver implementation. Outside the linear case, our NLP approach can be applied to inverse *convex* optimization because the more general gradient-computation machinery now also exists (Agrawal et al., 2019).

Although we highlighted SQP as a suitable gradient-based NLP solver, other NLP methods may work better in a given setting. Our methodology is applicable for any gradient-based NLP solver allowing specification of objective and constraints via callbacks, thereby being ‘agnostic’ to the bilevel nature.

4.7 Conclusion

In this paper, we propose a novel bilevel formulation and gradient-based framework for learning linear programs from optimal decisions. The methodology learns all parameters jointly while allowing flexible parametrizations of costs, constraints, and loss functions—a generalization of the problems typically addressed in the inverse linear optimization literature. It furthermore has speed advantages over a gradient-free approach.

Our work allows a strong class of inductive priors, namely parametric linear programs, to be imposed on a hypothesis space for learning. A major motivation for ours and for similar work is that, when the inductive prior is suited to the problem, we can learn a much better (and more interpretable) model, from far less data, than by applying general-purpose machine learning methods. In settings spanning economics, commerce, and healthcare, data on decisions is expensive to obtain and to collect, so we hope that our data-efficient approach will help to build better models and to make better decisions.

Appendix

Appendix A: Forward Optimization Problem for Figure 1

Forward optimization problem for Figure 4.1. The FOP formulation used is shown in (4.2) below.

$$
\begin{aligned}
\underset{x_1,x_2}{\text{minimize}} \quad & \cos(w_1 + w_2 u)x_1 + \sin(w_1 + w_2 u)x_2 \\
\text{subject to} \quad & (1 + w_2 u)x_1 \geq w_1 \\
& (1 + w_1)x_2 \geq w_2 u \\
& x_1 + x_2 \leq 1 + w_1 + w_2 u
\end{aligned}
\tag{4.2}
$$

For a fixed u and weights $\mathbf{w} = (w_1, w_2)$ it is an LP. The observation $\mathbf{x}_1^{\text{obs}} = (-0.625, 0.925)$ was generated using $u_1 = 1.0$ with true parameters $\mathbf{w} = (-0.5, -0.2)$.

For illustrative clarity, the panels in Figure 4.1 depicting the specific feasible regions for $\{\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ are slightly adjusted and stylized from the actual PLP (4.2), but are qualitatively representative.

Appendix B: Redundancy Among Target-Feasibility Constraints

Redundant constraints in (4.1a) are not problematic in principle. Still, removing redundant constraints may help overall performance, either in terms of speed or numerical stability of the 'outer' solver. Here we discuss strategies for automatically removing redundant constraints, depending on assumptions. In this section, when we use $\mathbf{x}$ or $\mathbf{x}_i$ it should be understood to represent some target $\mathbf{x}^{\text{obs}}$ or $\mathbf{x}_i^{\text{obs}}$.

Constraints that are equivalent. There may exist indices i and i' for which the corresponding constraints $\mathbf{a}(\mathbf{u}_i, \mathbf{w})^T\mathbf{x}_i \leq b(\mathbf{u}_i, \mathbf{w})$ and $\mathbf{a}(\mathbf{u}_{i'}, \mathbf{w})^T\mathbf{x}_{i'} \leq b(\mathbf{u}_{i'}, \mathbf{w})$ are identical or equivalent. For example, when a constraint is independent of $\mathbf{u}$ this often results in identical training targets $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ that produce identical constraints. The situation for equality constraints is similar.

Constraints independent of $\mathbf{w}$. If an individual constraint $\mathbf{a}(\mathbf{u},\mathbf{w})^T\mathbf{x} \leq b(\mathbf{u},\mathbf{w})$ is independent of $\mathbf{w}$ then either:

1. $\mathbf{a}(\mathbf{u}_i)^T\mathbf{x}_i \leq b(\mathbf{u}_i)$ for all i so the constraint can be omitted; or,
2. $\mathbf{a}(\mathbf{u}_i)^T\mathbf{x}_i > b(\mathbf{u}_i)$ for some i so the (ILOP) formulation is infeasible due to model misspecification, either in structural assumptions, or assumptions about noise.

The same follows for any equality constraint $\mathbf{g}(\mathbf{u},\mathbf{w})^T\mathbf{x} = h(\mathbf{u},\mathbf{w})$ that is independent of $\mathbf{w}$. For example, in our minimum-cost multi-commodity flow experiments, the flow conservation constraints (equality) are independent of $\mathbf{w}$ and so are omitted from (4.1a) in the corresponding (ILOP) formulation.

Constraints affinely-dependent in $\mathbf{w}$. Constraints may be affinely-dependent on parameters $\mathbf{w}$. For example, this is a common assumption in robust optimization (Zhen et al., 2018). Let $\mathbf{A}(\mathbf{u},\mathbf{w})$ and $\mathbf{b}(\mathbf{u},\mathbf{w})$ represent the constraints that are affinely dependent on $\mathbf{w} \in \mathbb{R}^K$. We can write

$$\mathbf{A}(\mathbf{u},\mathbf{w}) = \mathbf{A}^0(\mathbf{u}) + \sum_{k=1}^{K} w_k \mathbf{A}^k(\mathbf{u}) \qquad \text{and} \qquad \mathbf{b}(\mathbf{u},\mathbf{w}) = \mathbf{b}^0(\mathbf{u}) + \sum_{k=1}^{K} w_k \mathbf{b}^k(\mathbf{u})$$

for some matrix-valued functions $\mathbf{A}^k(\cdot)$ and vector-valued functions $\mathbf{b}^k(\cdot)$. It is easy to show that we can then rewrite the constraints $\mathbf{A}(\mathbf{u},\mathbf{w})\mathbf{x} \leq \mathbf{b}(\mathbf{u},\mathbf{w})$ as $\tilde{\mathbf{A}}(\mathbf{u},\mathbf{x})\mathbf{w} \leq \tilde{\mathbf{b}}(\mathbf{u},\mathbf{x})$ where

$$\begin{aligned}\tilde{\mathbf{A}}(\mathbf{u},\mathbf{x}) &= \begin{bmatrix}\mathbf{A}^1(\mathbf{u})\mathbf{x} - \mathbf{b}^1(\mathbf{u}) & \cdots & \mathbf{A}^K(\mathbf{u})\mathbf{x} - \mathbf{b}^K(\mathbf{u})\end{bmatrix} \\ \tilde{\mathbf{b}}(\mathbf{u},\mathbf{x}) &= \mathbf{b}^0(\mathbf{u}) - \mathbf{A}^0(\mathbf{u})\mathbf{x}.\end{aligned}$$

Similarly if $\mathbf{G}(\mathbf{u},\mathbf{w})\mathbf{x} = \mathbf{h}(\mathbf{u},\mathbf{w})$ are affine in $\mathbf{w}$ we can rewrite them as $\tilde{\mathbf{G}}(\mathbf{u},\mathbf{x})\mathbf{w} = \tilde{\mathbf{h}}(\mathbf{u},\mathbf{x})$. If we apply these functions across all training samples $i = 1,\ldots,N$, and stack their coefficients as

$$\tilde{\mathbf{A}} = \Big[\tilde{\mathbf{A}}(\mathbf{u}_i, \mathbf{x}_i)\Big]_{i=1}^{N}, \quad \tilde{\mathbf{b}} = \Big[\tilde{\mathbf{b}}(\mathbf{u}_i, \mathbf{x}_i)\Big]_{i=1}^{N}, \quad \tilde{\mathbf{G}} = \Big[\tilde{\mathbf{G}}(\mathbf{u}_i, \mathbf{x}_i)\Big]_{i=1}^{N}, \quad \tilde{\mathbf{h}} = \Big[\tilde{\mathbf{h}}(\mathbf{u}_i, \mathbf{x}_i)\Big]_{i=1}^{N}$$

then the corresponding (ILOP) constraints (4.1a) reduce to a set of linear 'outer' constraints $\tilde{\mathbf{A}}\mathbf{w} \le \tilde{\mathbf{b}}$ and $\tilde{\mathbf{G}}\mathbf{w} = \tilde{\mathbf{h}}$ where $\tilde{\mathbf{A}} \in \mathbb{R}^{NM_1 \times K}, \tilde{\mathbf{b}} \in \mathbb{R}^{NM_1}, \tilde{\mathbf{G}} \in \mathbb{R}^{NM_2 \times K}, \tilde{\mathbf{h}} \in \mathbb{R}^{NM_2}$. These reformulated constraint matrices are the system within which we eliminate redundancy in the affinely-dependent case, continued below.

Equality constraints affinely-dependent in w. We can eliminate affinely-dependent equality constraint sets by reparametrizing the (ILOP) search over a lower-dimensional space; this is what we do for the experiments with equality constraints shown in Figure 4.9, although the conclusions do not change with or without this reparametrization. To reparametrize the (ILOP) problem, compute a Moore-Penrose pseudoinverse $\tilde{\mathbf{G}}^+ \in \mathbb{R}^{K \times NM_2}$ to get a direct parametrization of constrained vector $\mathbf{w}$ in terms of an unconstrained vector $\mathbf{w}' \in \mathbb{R}^K$:

$$\mathbf{w}(\mathbf{w}') = \tilde{\mathbf{G}}^+\tilde{\mathbf{h}} + (\mathbf{I} - \tilde{\mathbf{G}}^+\tilde{\mathbf{G}})\mathbf{w}'. \tag{4.3}$$

By reparametrizing (ILOP) in terms of $\mathbf{w}'$ we guarantee $\tilde{\mathbf{G}}\mathbf{w}(\mathbf{w}') = \tilde{\mathbf{h}}$ is satisfied and can drop equality constraints from (4.1a) entirely. There are three practical issues with (4.3):

1. Constrained vector $\mathbf{w}$ only has $K' \equiv K - \operatorname{rank}(\tilde{\mathbf{G}})$ degrees of freedom, so we would like to re-parametrize over a lower-dimensional $\mathbf{w}' \in \mathbb{R}^{K'}$.
2. To search over $\mathbf{w}' \in \mathbb{R}^{K'}$ we need to specify $\tilde{\mathbf{A}}' \in \mathbb{R}^{NM_1 \times K'}$ and $\tilde{\mathbf{b}}' \in \mathbb{R}^{NM_1}$ such that $\tilde{\mathbf{A}}'\mathbf{w}' \le \tilde{\mathbf{b}}'$ is equivalent to $\tilde{\mathbf{A}}\mathbf{w}(\mathbf{w}') \le \tilde{\mathbf{b}}$.
3. Given initial $\mathbf{w}_{\text{ini}} \in \mathbb{R}^K$ we need a corresponding $\mathbf{w}'_{\text{ini}} \in \mathbb{R}^{K'}$ to initialize our

search.

To address the first issue, we can let the final $K - K'$ components of $\mathbf{w}' \in \mathbb{R}^K$ in (4.3) be zero, which corresponds to using a lower-dimensional $\mathbf{w}' \in \mathbb{R}^{K'}$. As shorthand let matrix $\mathbf{P} \in \mathbb{R}^{K \times K'}$ be

$$\mathbf{P} \equiv (\mathbf{I}_{K\times K} - \tilde{\mathbf{G}}^{+}\tilde{\mathbf{G}})\mathbf{I}_{K\times K'} = \mathbf{I}_{K\times K'} - (\tilde{\mathbf{G}}^{+}\tilde{\mathbf{G}})_{1:K,1:K'}$$

where $\mathbf{I}_{K\times K'}$ denotes $\begin{bmatrix} \mathbf{I}_{K'\times K'} \\ \mathbf{0}_{(K-K')\times K'} \end{bmatrix}$ as in `torch.eye(K, K')` and $(\mathbf{G}^{+}\mathbf{G})_{1:K,1:K'}$ denotes the first K' columns of $K \times K$ matrix $\mathbf{G}^{+}\mathbf{G}$. Then we have $\mathbf{w}(\mathbf{w}') = \mathbf{G}^{+}\mathbf{h} + \mathbf{P}\mathbf{w}'$ where the full dimension of $\mathbf{w}' \in \mathbb{R}^{K'}$ matches the degrees of freedom in $\mathbf{w}$ subject to $\tilde{\mathbf{G}}\mathbf{w} = \tilde{\mathbf{h}}$ and we have $\tilde{\mathbf{G}}\mathbf{w}(\mathbf{w}') = \tilde{\mathbf{h}}$ for any choice of $\mathbf{w}'$.

To address the second issue, simplifying $\tilde{\mathbf{A}}\mathbf{w}(\mathbf{w}') \leq \tilde{\mathbf{b}}$ gives inequality constraints $\tilde{\mathbf{A}}'\mathbf{w}' \leq \tilde{\mathbf{b}}'$ with $\tilde{\mathbf{A}}' = \tilde{\mathbf{A}}\mathbf{P}$ and $\tilde{\mathbf{b}}' = \tilde{\mathbf{b}} - \tilde{\mathbf{A}}\tilde{\mathbf{G}}^{+}\tilde{\mathbf{h}}$.

To address the third issue we must solve for $\mathbf{w}'_{\text{ini}} \in \mathbb{R}^{K'}$ in the linear system $\mathbf{P}\mathbf{w}'_{\text{ini}} = \mathbf{w}_{\text{ini}} - \tilde{\mathbf{G}}^{+}\tilde{\mathbf{h}}$. Since $\text{rank}(\mathbf{P}) = K'$ the solution exists and is unique.

Consider also the effect of this reparametrization when $\tilde{\mathbf{G}}\mathbf{w} = \tilde{\mathbf{h}}$ is an infeasible system, for example due to noisy observations or misspecified constraints. In that case searching over $\mathbf{w}'$ automatically restricts the search to $\mathbf{w}$ that satisfy $\tilde{\mathbf{G}}\mathbf{w} = \tilde{\mathbf{h}}$ in a least squares sense, akin to adding an infinitely-weighted $\|\tilde{\mathbf{G}}\mathbf{w} - \tilde{\mathbf{h}}\|^2$ term to the (ILOP) objective.

Inequality constraints affinely-dependent in w. After transforming affinely-dependent inequality constraints to $\tilde{\mathbf{A}}'\mathbf{w}' \leq \tilde{\mathbf{b}}'$, detecting redundancy among these constraints can be as hard as solving an LP (Telgen, 1983). Generally, inequality constraint $\mathbf{a}_j^T\mathbf{w} \leq b_j$ is redundant with respect to $\mathbf{A}\mathbf{w} \leq \mathbf{b}$ if and only if the optimal value of the following LP is non-negative:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & b_j - \mathbf{a}_j^T \mathbf{w} \\ \text{subject to} \quad & \mathbf{A}_{\{j' \neq j\}} \mathbf{w} \leq \mathbf{b}_{\{j' \neq j\}} \end{aligned} \tag{4.4}$$

Here $\mathbf{a}_j$ is the j^{th} row of $\mathbf{A}$ and $\mathbf{A}_{\{j' \neq j\}}$ is all the rows of $\mathbf{A}$ except the j^{th}. If the optimal value to (4.4) is non-negative then it says "we tried to violate the j^{th} constraint, but the other constraints prevented it, and so the j^{th} constraint must be redundant." However, Telgen (1983) reviews much more efficient methods of identifying redundant linear inequality constraints, by analysis of basic basic variables in a simplex tableau. Zhen et al. (2018) proposed a 'redundant constraint identification' (RCI) procedure that is directly analogous to (4.4) along with another heuristic RCI procedure.

Constraints polynomially-dependent in $\mathbf{w}$. Similar to the affinely-dependent case, when the coefficients of constraints $\mathbf{A}(\mathbf{u}, \mathbf{w})\mathbf{x} \leq \mathbf{b}(\mathbf{u}, \mathbf{w})$ and $\mathbf{G}(\mathbf{u}, \mathbf{w})\mathbf{x} \leq \mathbf{h}(\mathbf{u}, \mathbf{w})$ are polynomially-dependent on $\mathbf{w}$, we can rewrite the constraints in terms of $\mathbf{w}$. Redundancy among equality constraints of the resulting system can be simplified by computing a minimal Gröbner basis (Cox et al., 2013), for example by Buchberger's algorithm which is a generalization of Gaussian elimination; see the paper by Lim and Brunner (2012) for a review of Gröbner basis techniques applicable over a real field. Redundancy among inequality constraints for nonlinear programming has been studied (Caron, 2009; Obuchowska and Caron, 1995). Simplifying polynomial systems of equalities and inequalities is a subject of semialgebraic geometry and involves generalizations of Fourier-Motzkin elimination. Details are beyond the scope of this manuscript.

Appendix C: Proofs of Theorem 1 and Corollary 1

Degeneracy is often defined for vertices, but solutions returned by an interior point solver tend toward the analytical center of the optimal face. We first define non-degeneracy for a face of an LP model, following Tijssen and Sierksma (1998) and Sierksma and Tijssen (2003). We then use this definition to define non-degeneracy of a particular solution on the optimal face of an LP model.

Let F be a face of the polyhedron P. A constraint of P is *binding* on F if it is satisfied with equality for every point of F. Let $\dim(F)$ and n denote the dimension of F and P respectively and $\operatorname{bnd}(F, P)$ denote the number of constraints of P that are binding on F.

Definition 4.1 (Tijssen & Sierksma 1998) *The* degeneracy degree *of a face F with respect to polyhedron P is $\sigma(F, P) = \operatorname{bnd}(F, P) + \dim(F) - n$.*

Definition 4.2 (Tijssen & Sierksma 1998) *A face F of polyhedron P is* degenerate *iff $\sigma(F, P) > 0$, and* non-degenerate *iff $\sigma(F, P) = 0$.*

Definition 4.3 *Given an LP with feasible set P, an optimal solution $\mathbf{x}^*$ is* non-degenerate *iff the smallest face F containing $\mathbf{x}^*$ is non-degenerate, i.e., $\sigma(F, P) = 0$ for the smallest face with $\mathbf{x}^* \in F$.*

By these definitions, a solution on the relative interior of the optimal face may be non-degenerate, even when other sub-faces of the optimal face (including vertices) are degenerate in the usual sense.

To assist the proof of Theorem 4.1, we first show that the following lemma is true.

Lemma 4.1 *If $\mathbf{x}^*$ is a non-degenerate solution to an LP, then the constraints active at $\mathbf{x}^*$ are linearly independent.*

Proof Let P denote the feasible set of the LP, and let F denote the smallest face containing $\mathbf{x}^* \in \mathbb{R}^n$. Let $\{\mathbf{a}_1, \dots, \mathbf{a}_k\}$ denote the set of constraints binding on F, so that $\text{bnd}(F, P) = k$. By non-degeneracy of $\mathbf{x}^*$ we have $\sigma(F, P) = 0$ and so $k = n - \dim(F)$. Since $\dim(F) + \text{rank}\{\mathbf{a}_1, \dots, \mathbf{a}_k\} = n$ must also hold, we have $\text{rank}\{\mathbf{a}_1, \dots, \mathbf{a}_k\} = k$, i.e., the constraints binding face F are of full rank.

Now consider whether a constraint $\mathbf{a}_{k+1}$ can be active at $\mathbf{x}^*$ but not binding on F. Since $\mathbf{a}_{k+1}$ is not binding on F, it must be must be linearly independent from the constraints that are binding on F, i.e., $\text{rank}\{\mathbf{a}_1, \dots, \mathbf{a}_k, \mathbf{a}_{k+1}\} = k + 1$. But if $\mathbf{a}_{k+1}$ were also active at $\mathbf{x}^*$, this would imply the existence of a face of P containing $\mathbf{x}^*$ and having dimension $n - k - 1$. Since F has dimension $n - k$, this would contradict our assumption that F is the smallest face containing $\mathbf{x}^*$. Therefore $\{\mathbf{a}_1, \dots, \mathbf{a}_k\}$ must comprise all constraints that are active on $\mathbf{x}^*$, and they are linearly independent.

Proof of Theorem 4.1 The dual linear program associated with (LP) is

$$\begin{aligned} \underset{\boldsymbol{\lambda}, \boldsymbol{\nu}}{\text{maximize}} \quad & \mathbf{b}^T\boldsymbol{\lambda} + \mathbf{h}^T\boldsymbol{\nu} \\ \text{subject to} \quad & \mathbf{A}^T\boldsymbol{\lambda} + \mathbf{G}^T\boldsymbol{\nu} \;=\; \mathbf{c} \\ & \boldsymbol{\lambda} \;\leq\; \mathbf{0}, \end{aligned} \tag{DP}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{M_1}_{\leq 0}, \boldsymbol{\nu} \in \mathbb{R}^{M_2}$ are the associated dual variables for the primal inequality and equality constraints, respectively.

Since $\mathbf{x}^*$ is optimal to (LP) and $\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*$ are optimal to (DP), then $(\mathbf{x}^*,\ \boldsymbol{\lambda}^*,\ \boldsymbol{\nu}^*)$

satisfy the KKT conditions (written specialized to the particular LP form we use):

$$\begin{aligned}\mathbf{A}\mathbf{x} &\leq \mathbf{b}\\ \mathbf{G}\mathbf{x} &= \mathbf{h}\\ \mathbf{A}^T\boldsymbol{\lambda} + \mathbf{G}^T\boldsymbol{\nu} &= \mathbf{c}\\ \boldsymbol{\lambda} &\leq \mathbf{0}\\ \mathbf{D}(\boldsymbol{\lambda})(\mathbf{A}\mathbf{x} - \mathbf{b}) &= \mathbf{0}\end{aligned} \tag{KKT}$$

where $\mathbf{D}(\boldsymbol{\lambda})$ is the diagonal matrix having $\boldsymbol{\lambda}$ on the diagonal. The first two constraints correspond to primal feasibility, the next two to dual feasibility and the last one specifies complementary slackness. From here forward it should be understood that $\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}$ satisfy KKT even when not emphasized by $*$.

As in the paper by Amos and Kolter (2017), implicitly differentiating the equality constraints in (KKT) gives

$$\begin{aligned}\mathbf{G}\mathrm{d}\mathbf{x} &= \mathrm{d}\mathbf{h} - \mathrm{d}\mathbf{G}\mathbf{x}\\ \mathbf{A}^T\mathrm{d}\boldsymbol{\lambda} + \mathbf{G}^T\mathrm{d}\boldsymbol{\nu} &= \mathrm{d}\mathbf{c} - \mathrm{d}\mathbf{A}^T\boldsymbol{\lambda} - \mathrm{d}\mathbf{G}^T\boldsymbol{\nu}\\ \mathbf{D}(\boldsymbol{\lambda})\mathbf{A}\mathrm{d}\mathbf{x} + \mathbf{D}(\mathbf{A}\mathbf{x} - \mathbf{b})\mathrm{d}\boldsymbol{\lambda} &= \mathbf{D}(\boldsymbol{\lambda})(\mathrm{d}\mathbf{b} - \mathrm{d}\mathbf{A}\mathbf{x})\end{aligned} \tag{DKKT}$$

where $\mathrm{d}\mathbf{c}, \mathrm{d}\mathbf{A}, \mathrm{d}\mathbf{b}, \mathrm{d}\mathbf{G}, \mathrm{d}\mathbf{h}$ are parameter differentials and $\mathrm{d}\mathbf{x}, \mathrm{d}\boldsymbol{\lambda}, \mathrm{d}\boldsymbol{\nu}$ are solution differentials, all having the same dimensions as the variables they correspond to. Because (KKT) is a second-order system, (DKKT) is a system of linear equations. Because the system is linear, a partial derivative such as $\frac{\partial x_j^*}{\partial b_i}$ can be determined (if it exists) by setting $\mathrm{d}b_i = 1$ and all other parameter differentials to 0, then solving the system for solution differential $\mathrm{d}x_j$, as shown by Amos and Kolter (2017).

We can assume (KKT) is feasible in $\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}$. In each case of the main proof it will be important to characterize conditions under which (DKKT) is then feasible in $\mathrm{d}\mathbf{x}$.

This is because, if (DKKT) is feasible in at least d$\mathbf{x}$, then by substitution we have

$$\begin{aligned}
\mathbf{c}^T \mathrm{d}\mathbf{x} &= (\mathbf{A}^T \boldsymbol{\lambda} + \mathbf{G}^T \boldsymbol{\nu})^T \mathrm{d}\mathbf{x} \\
&= \boldsymbol{\lambda}^T \mathbf{A} \mathrm{d}\mathbf{x} + \boldsymbol{\nu}^T \mathbf{G} \mathrm{d}\mathbf{x} \\
&= \boldsymbol{\lambda}^T (\mathrm{d}\mathbf{b} - \mathrm{d}\mathbf{A}\mathbf{x}) + \boldsymbol{\nu}^T (\mathrm{d}\mathbf{h} - \mathrm{d}\mathbf{G}\mathbf{x})
\end{aligned} \tag{4.5}$$

and this substitution is what gives the total derivatives their form. In (4.5) the substitution $\boldsymbol{\lambda}^T \mathbf{A} \mathrm{d}\mathbf{x} = \boldsymbol{\lambda}^T (\mathrm{d}\mathbf{b} - \mathrm{d}\mathbf{A}\mathbf{x})$ holds because $\mathbf{x}, \boldsymbol{\lambda}$ feasible in (KKT) implies $\lambda_i < 0 \Rightarrow \mathbf{A}_i \mathbf{x} - b_i = 0$ in (DKKT), where $\mathbf{A}_i$ is the i^{th} row of $\mathbf{A}$. Whenever d$\mathbf{x}$ is feasible in (DKKT) we have $\lambda_i \mathbf{A}_i \mathrm{d}\mathbf{x} = \lambda_i (\mathrm{d}b_i - \mathrm{d}\mathbf{A}_i \mathbf{x})$ for any $\lambda_i \leq 0$, where $\mathrm{d}\mathbf{A}_i$ is the i^{th} row of differential d$\mathbf{A}$.

Note that (4.5) holds even if (DKKT) is not feasible in d$\boldsymbol{\lambda}$ and/or d$\boldsymbol{\nu}$. In other words, it does not require the KKT point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ to be differentiable with respect to $\boldsymbol{\lambda}^*$ and/or $\boldsymbol{\nu}^*$.

Given a KKT point $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ let $\mathcal{I}, \mathcal{J}, \mathcal{K}$ be a partition of inequality indices $\{1, \ldots, M_1\}$ where

$$\begin{aligned}
\mathcal{I} &= \{\, i : \lambda_i^* < 0,\ \mathbf{A}_i \mathbf{x}^* = b_i \,\} \\
\mathcal{J} &= \{\, i : \lambda_i^* = 0,\ \mathbf{A}_i \mathbf{x}^* < b_i \,\} \\
\mathcal{K} &= \{\, i : \lambda_i^* = 0,\ \mathbf{A}_i \mathbf{x}^* = b_i \,\}
\end{aligned}$$

and the corresponding submatrices of $\mathbf{A}$ are $\mathbf{A}_{\mathcal{I}}, \mathbf{A}_{\mathcal{J}}, \mathbf{A}_{\mathcal{K}}$. Then (DKKT) in matrix form is

$$\begin{bmatrix}
\mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{D}(\boldsymbol{\lambda}_{\mathcal{I}})\mathbf{A}_{\mathcal{I}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{D}(\mathbf{A}_{\mathcal{J}}\mathbf{x} - \mathbf{b}_{\mathcal{J}}) & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{A}_{\mathcal{I}}^T & \mathbf{A}_{\mathcal{J}}^T & \mathbf{A}_{\mathcal{K}}^T & \mathbf{G}^T
\end{bmatrix}
\begin{bmatrix}
\mathrm{d}\mathbf{x} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{I}} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{J}} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{K}} \\ \mathrm{d}\boldsymbol{\nu}
\end{bmatrix}
=
\begin{bmatrix}
\mathrm{d}\mathbf{h} - \mathrm{d}\mathbf{G}\mathbf{x} \\
\mathrm{d}\mathbf{b}_{\mathcal{I}} - \mathrm{d}\mathbf{A}_{\mathcal{I}}\mathbf{x} \\
\mathbf{0} \\
\mathbf{0} \\
\mathrm{d}\mathbf{c} - \mathrm{d}\mathbf{A}^T\boldsymbol{\lambda} - \mathrm{d}\mathbf{G}^T\boldsymbol{\nu}
\end{bmatrix} \tag{4.6}$$

The pattern of the proof in each case will be to characterize feasibility of (4.6) in $\mathrm{d}\mathbf{x}$ and then apply (4.5) for the result.

Evaluating $\frac{\partial z}{\partial \mathbf{c}}$. Consider $\frac{\partial z}{\partial c_j} = x_j^{\text{obs}} - x_j^* - \mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial c_j}$. To evaluate the $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial c_j}$ term, set $\mathrm{d}c_j = 1$ and all other parameter differentials to 0. Then the right-hand side of (4.6) becomes

$$\begin{bmatrix} \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}(\boldsymbol{\lambda}_\mathcal{I})\mathbf{A}_\mathcal{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}(\mathbf{A}_\mathcal{J}\mathbf{x} - \mathbf{b}_\mathcal{J}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_\mathcal{I}^T & \mathbf{A}_\mathcal{J}^T & \mathbf{A}_\mathcal{K}^T & \mathbf{G}^T \end{bmatrix} \begin{bmatrix} \mathrm{d}\mathbf{x} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{I} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{J} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{K} \\ \mathrm{d}\boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{1}^j \end{bmatrix} \tag{4.7}$$

where $\mathbf{1}^j$ denotes the vector with 1 for component j and 0 elsewhere. System (4.7) is feasible in $\mathrm{d}\mathbf{x}$ (not necessarily unique) so we can apply (4.5) to get $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial c_j} = \mathbf{c}^T \mathrm{d}\mathbf{x} = \boldsymbol{\lambda}^T(\mathbf{0} - \mathbf{0}\mathbf{x}) + \boldsymbol{\nu}^T(\mathbf{0} - \mathbf{0}\mathbf{x}) = 0$. The result for $\frac{\partial z}{\partial \mathbf{c}}$ then follows from $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial \mathbf{c}} = \mathbf{0}$.

Evaluating $\frac{\partial z}{\partial \mathbf{h}}$. Consider $\frac{\partial z}{\partial h_i} = -\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial h_i}$. Set $\mathrm{d}h_i = 1$ and all other parameter differentials to 0. Then the right-hand side of (4.6) becomes

$$\begin{bmatrix} \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}(\boldsymbol{\lambda}_\mathcal{I})\mathbf{A}_\mathcal{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}(\mathbf{A}_\mathcal{J}\mathbf{x} - \mathbf{b}_\mathcal{J}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_\mathcal{I}^T & \mathbf{A}_\mathcal{J}^T & \mathbf{A}_\mathcal{K}^T & \mathbf{G}^T \end{bmatrix} \begin{bmatrix} \mathrm{d}\mathbf{x} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{I} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{J} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{K} \\ \mathrm{d}\boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \mathbf{1}^i \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{4.8}$$

Since $\mathbf{x}^*$ is non-degenerate in the sense of Definition 4.3, then there are at most D active constraints (including equality constraints) and by Lemma 4.1 the rows of

$\begin{bmatrix} \mathbf{G} \\ \mathbf{A}_\mathcal{I} \end{bmatrix}$ are also linearly independent. Since active constraints are linearly independent, system (4.8) is feasible in d$\mathbf{x}$ across all $i \in \{1, \ldots, M_2\}$. We can therefore apply (4.5) to get $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial h_i} = \mathbf{c}^T \mathrm{d}\mathbf{x} = \boldsymbol{\lambda}^T(\mathbf{0} - \mathbf{0x}) + \boldsymbol{\nu}^T(\mathbf{1}^i - \mathbf{0x}) = \nu_i$. The result for $\frac{\partial z}{\partial \mathbf{h}}$ then follows from $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial \mathbf{h}} = \boldsymbol{\nu}^{*T}$.

Evaluating $\frac{\partial z}{\partial \mathbf{b}}$. Consider $\frac{\partial z}{\partial b_i} = -\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial b_i}$. Set d$b_i = 1$ and all other parameter differentials to 0. For $i \in \mathcal{I}$ the right-hand side of (4.6) becomes

$$\begin{bmatrix} \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}(\boldsymbol{\lambda}_\mathcal{I})\mathbf{A}_\mathcal{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}(\mathbf{A}_\mathcal{J}\mathbf{x} - \mathbf{b}_\mathcal{J}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_\mathcal{I}^T & \mathbf{A}_\mathcal{J}^T & \mathbf{A}_\mathcal{K}^T & \mathbf{G}^T \end{bmatrix} \begin{bmatrix} \mathrm{d}\mathbf{x} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{I} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{J} \\ \mathrm{d}\boldsymbol{\lambda}_\mathcal{K} \\ \mathrm{d}\boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \lambda_i \mathbf{1}^i \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{4.9}$$

Since $\mathbf{x}^*$ is non-degenerate, then system (4.9) is feasible in d$\mathbf{x}$ for all $i \in \mathcal{I}$ by identical reasoning as for $\frac{\partial z}{\partial h_i}$. For $i \in \mathcal{J} \cup \mathcal{K}$ the right-hand side of (4.6) is zero and so the system is feasible in d$\mathbf{x}$. System (4.9) is therefore feasible in d$\mathbf{x}$ across all $i \in \{1, \ldots, M_1\}$. We can therefore apply (4.5) to get $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial b_i} = \mathbf{c}^T \mathrm{d}\mathbf{x} = \boldsymbol{\lambda}^T(\mathbf{1}^i - \mathbf{0x}) + \boldsymbol{\nu}^T(\mathbf{0} - \mathbf{0x}) = \lambda_i$. The result for $\frac{\partial z}{\partial \mathbf{b}}$ then follows from $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial \mathbf{b}} = \boldsymbol{\lambda}^{*T}$.

Evaluating $\frac{\partial z}{\partial \mathbf{G}}$. Consider $\frac{\partial z}{\partial G_{ij}} = -\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial G_{ij}}$. Set d$G_{ij} = 1$ and all other parameter differentials to 0. Then the right-hand side of (4.6) becomes

$$\begin{bmatrix} \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}(\boldsymbol{\lambda}_{\mathcal{I}})\mathbf{A}_{\mathcal{I}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}(\mathbf{A}_{\mathcal{J}}\mathbf{x}-\mathbf{b}_{\mathcal{J}}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\mathcal{I}}^T & \mathbf{A}_{\mathcal{J}}^T & \mathbf{A}_{\mathcal{K}}^T & \mathbf{G}^T \end{bmatrix} \begin{bmatrix} \mathrm{d}\mathbf{x} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{I}} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{J}} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{K}} \\ \mathrm{d}\boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} -x_j\mathbf{1}^i \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ -\nu_i\mathbf{1}^j \end{bmatrix} \tag{4.10}$$

Since $\mathbf{x}^*$ is non-degenerate, then (4.10) is feasible in $\mathrm{d}\mathbf{x}$ for all $i \in \{1,\ldots,M_2\}$ and $j \in \{1,\ldots,D\}$ by same reasoning as $\frac{\partial z}{\partial \mathbf{h}}$. Applying (4.5) gives $\mathbf{c}^T\frac{\partial \mathbf{x}^*}{\partial G_{ij}} = \mathbf{c}^T\mathrm{d}\mathbf{x} = \boldsymbol{\lambda}^T(\mathbf{0}-\mathbf{0}\mathbf{x}) + \boldsymbol{\nu}^T(\mathbf{0}-\mathbf{1}^{ij}\mathbf{x}) = -\nu_i x_j$ where $\mathbf{1}^{ij}$ is the $M_2 \times D$ matrix with 1 for component (i,j) and zeros elsewhere. The result for $\frac{\partial z}{\partial \mathbf{G}}$ then follows from $\mathbf{c}^T\frac{\partial \mathbf{x}^*}{\partial \mathbf{G}} = -\boldsymbol{\nu}^*\mathbf{x}^{*T}$ where we have slightly abused notation by dropping the leading singleton dimension of the $1 \times M_2 \times D$ Jacobian.

Evaluating $\frac{\partial z}{\partial \mathbf{A}}$. Consider $\frac{\partial z}{\partial A_{ij}} = -\mathbf{c}^T\frac{\partial \mathbf{x}^*}{\partial A_{ij}}$. Set $\mathrm{d}A_{ij} = 1$ and all other parameter differentials to 0. Then the right-hand side of (4.6) becomes

$$\begin{bmatrix} \mathbf{G} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{D}(\boldsymbol{\lambda}_{\mathcal{I}})\mathbf{A}_{\mathcal{I}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}(\mathbf{A}_{\mathcal{J}}\mathbf{x}-\mathbf{b}_{\mathcal{J}}) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{\mathcal{I}}^T & \mathbf{A}_{\mathcal{J}}^T & \mathbf{A}_{\mathcal{K}}^T & \mathbf{G}^T \end{bmatrix} \begin{bmatrix} \mathrm{d}\mathbf{x} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{I}} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{J}} \\ \mathrm{d}\boldsymbol{\lambda}_{\mathcal{K}} \\ \mathrm{d}\boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -x_j\mathbf{1}^i \\ \mathbf{0} \\ \mathbf{0} \\ -\lambda_i\mathbf{1}^j \end{bmatrix} \tag{4.11}$$

Since $\mathbf{x}^*$ is non-degenerate, then by similar arguments as $\frac{\partial z}{\partial \mathbf{b}}$ and $\frac{\partial z}{\partial \mathbf{G}}$ (4.11) is feasible in $\mathrm{d}\mathbf{x}$ for all $i \in \{1,\ldots,M_1\}$ and $j \in \{1,\ldots,D\}$ and the result for $\frac{\partial z}{\partial \mathbf{A}}$ follows from $\mathbf{c}^T \frac{\partial \mathbf{x}^*}{\partial \mathbf{G}} = -\boldsymbol{\lambda}^* \mathbf{x}^{*T}$.

Proof of Corollary 4.1 The result for $\frac{\partial z}{\partial \mathbf{c}}$ is direct. In linear programming, Tijssen and Sierksma (1998) showed that the existence of a non-degenerate primal solution $\mathbf{x}^*$ implies uniqueness of the dual solution $\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*$ so the result for $\frac{\partial z}{\partial \mathbf{b}}$ and $\frac{\partial z}{\partial \mathbf{h}}$ follows directly. If a non-degenerate solution $\mathbf{x}^*$ is unique then matrices $\boldsymbol{\lambda}^* \mathbf{x}^{*T}$ and $\boldsymbol{\nu}^* \mathbf{x}^{*T}$ are both unique, regardless of whether $\mathbf{c} = \mathbf{0}$. In the other direction, if $\boldsymbol{\lambda}^* \mathbf{x}^{*T}$ and $\boldsymbol{\nu}^* \mathbf{x}^{*T}$ are both unique, consider two mutually exclusive and exhaustive cases: (1) when either $\boldsymbol{\lambda}^* \neq \mathbf{0}$ or $\boldsymbol{\nu}^* \neq \mathbf{0}$ this would imply $\mathbf{x}^*$ unique, and (2) when both $\boldsymbol{\lambda}^* = \mathbf{0}$ and $\boldsymbol{\nu}^* = \mathbf{0}$ in (DP) this would imply $\mathbf{c} = \mathbf{0}$, *i.e.* the primal linear program (LP) is merely a feasibility problem. The result for $\frac{\partial z}{\partial \mathbf{A}}$ and $\frac{\partial z}{\partial \mathbf{G}}$ then follows.

Appendix D: Additional Results

Figure 4.8 shows the task of learning ($\mathbf{c}$, $\mathbf{A}$, $\mathbf{b}$) with a $K = 6$ dimensional parametrization $\mathbf{w}$ and 20 training observations for a D dimensional decision space $\mathbf{x}$ with M_1 inequality constraints. The five different considered combinations of D and M_1 are shown in the figure.

The results over all problem sizes are similar to the case of $D = 10, M_1 = 80$ shown in the main paper: RS fails; COBYLA 'succeeds' on $\sim 25\%$ of instances; SQP succeeds on $\sim 60 - -75\%$. As expected, instances with higher D, are more challenging as we observe that the success rate decreases slightly. The success curve of $\text{SQP}_{\text{bprop}}$ slightly lags those of SQP_{impl} and SQP_{dir} due to the overhead of backpropagating through the steps of the interior point solver. However, this computational advantage of SQP_{impl} and SQP_{dir} over $\text{SQP}_{\text{bprop}}$ is less obvious on LP instances with $D = 10$. For larger LP instances, the overall framework spends significantly more computation time on other components (e.g., solving the forward problem, solving (SQP)). Thus, the advantage of SQP_{impl} and SQP_{dir} in computing gradients is less significant in the overall performance. The SQP_{cvx} implementation works better than COBYLA for most instances, but struggles to converge to the requested tolerance when more constraints are added ($M_1 = 80$, shown in Figure 4.3).

Figure 4.8 shows the experiment results of total 100 trials, where each trial includes 20 training and 20 testing instances. After training, if an instance $\mathbf{u}_i$ (of 20) for which the LP $\mathbf{c}(\mathbf{u}_i, \mathbf{w}), \mathbf{A}(\mathbf{u}_i, \mathbf{w}), \mathbf{b}(\mathbf{u}_i, \mathbf{w})$ is infeasible or unbounded, then we report a loss $\ell(\mathbf{w}) = 100$ arbitrarily and consider these to be failures. In the $M_1 = 80$ case, SQP_{cvx} tends to fail for one of two reasons: its forward solver (*scs*) is slow to converge to the requested tolerance of 10^{-8}, or *cvxpylayers* raises an exception on encountering any infeasible/unbounded instance (whereby we return $\ell = 100$); the latter behaviour is a consequence of how *cvxpylayers* handles errors, not a fundamental limitation.

Figure 4.9 shows instances with equality constraints, where $\mathbf{G}$ and $\mathbf{h}$ also need to be learned, and the performance is similar. Note that RS failed to find a feasible $\mathbf{w}$ in all instances, caused mainly by the failure to satisfy the equality target feasibility constraints in (4.1a). Recall that a feasible $\mathbf{w}$ means both (4.1a) and (4.1b) are satisfied.

Figure 4.10 shows the performance on LPs where the dimensionality of $\mathbf{w}$ is higher. It shows the probability of achieving zero AOE training loss over time (curves), along with final loss (box plots). Each mark denotes one of 100 trials (different instances), each with one training point. Note, in this experiment we aim to learn LP coefficients directly, i.e., $\mathbf{w}$ comprises all LP coefficients, and the LP coefficients do not depend on $\mathbf{u}$. Therefore, there is only a single target $\mathbf{x}^{\text{obs}}$ for learning $\mathbf{w}$, and no testing data. We observe that COBYLA performs poorly, SQP_{cvx} makes progress but is slow, and the remaining SQP methods succeed quickly on all instances. COBYLA's poor performance is caused by the finite-difference approximation technique used in

COBYLA which is inefficient in high dimension $\mathbf{w}$ space. This result demonstrates the importance of using gradient-based methods in high dimensional (in $\mathbf{w}$) NLP.

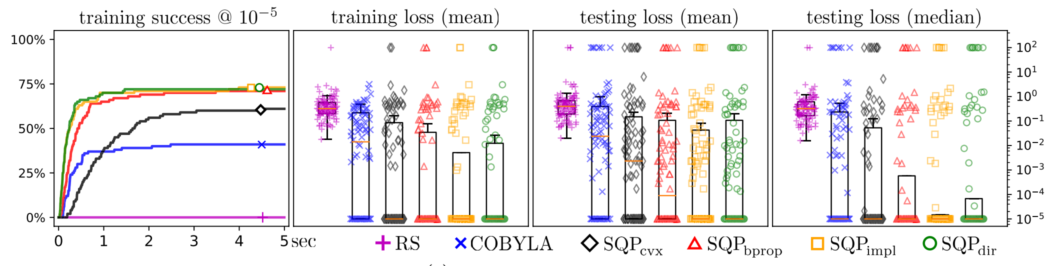
Sensitivity of results to parameter settings The specific results of our experiments can vary slightly with certain choices, but the larger conclusions do not change: the gradient-based SQP methods all perform similarly, and they consistently out-perform non-gradient-based methods, especially for higher-dimensional search.

Specific choices of parameter settings include the numerical tolerance used in the forward solve (e.g. 10^{-5} vs 10^{-8}), algorithm termination tolerances of the COBYLA and SLSQP, and PyTorch version (v1.6 vs. nightly builds). However, we did see a degradation in “success rates” when tolerance on the forward problem was configured to be weak (10^{-3}), which may be caused by unstable or inaccurate gradients. The running time of the $\mathrm{SQP}_{\mathrm{cvx}}$ forward solver, *scs*, can be very sensitive to the numerical tolerance requested. For example, using the default SciPy tolerance of 10^{-8} and `max_iter`$= 10^8$, the *scs* solver could be $>$100x slower than the case of using its default settings of tolerance 10^{-3} and `max_iter`$= 2500$.

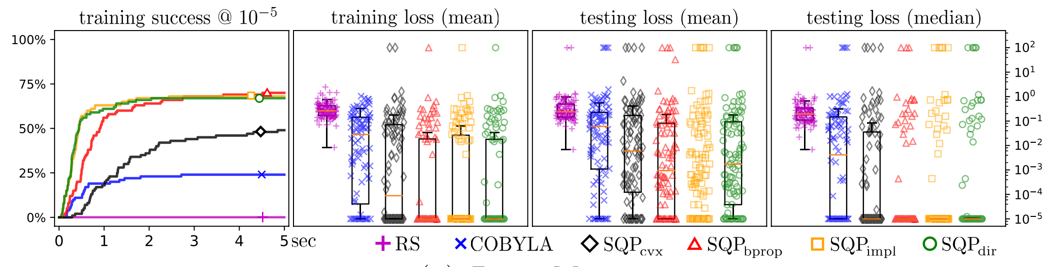
In general, the experimental results of $\mathrm{SQP}_{\mathrm{bprop}}$, $\mathrm{SQP}_{\mathrm{impl}}$ and $\mathrm{SQP}_{\mathrm{dir}}$ are largely insensitive to specific parameter settings. For example, we tried using strict tolerances and different trust region sizes for COBYLA to encourage the algorithm to search more aggressively, but these made only a small improvement to performance; these small improvements are represented in our results. We also observed that, although the homogeneous solver works slightly better when we use a strict numerical tolerance, there is no major difference in the learning results.

Appendix E: Parametric Linear Program for Figure 4.6

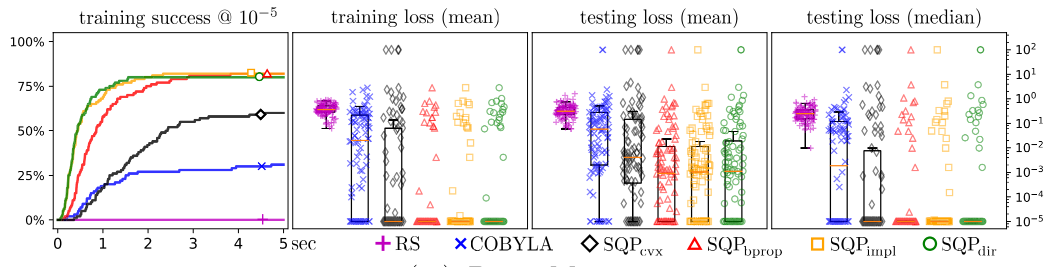
Forward optimization problem for Figure 4.6. The FOP formulation used is shown



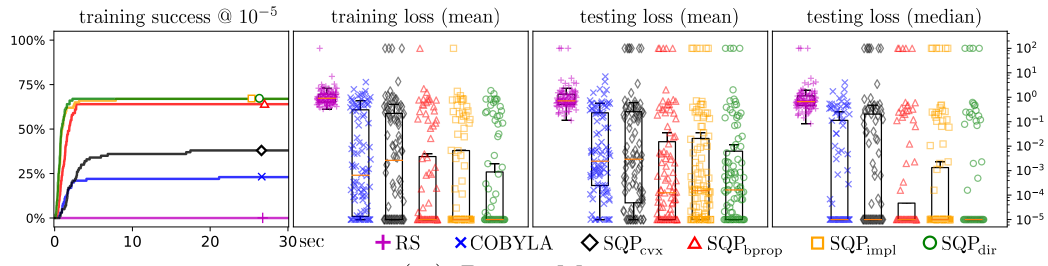
(i) $D=2, M_1=4$



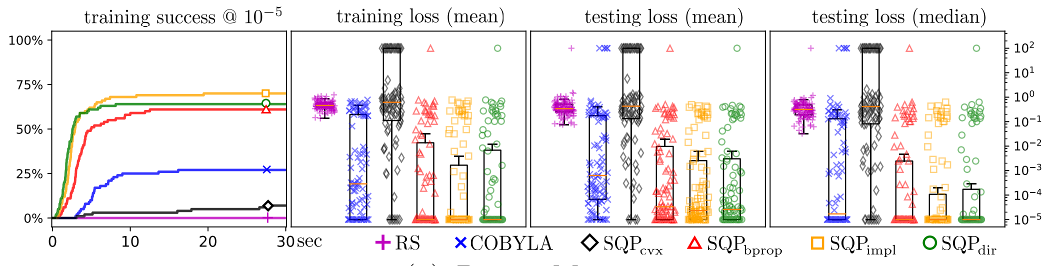
(ii) $D=2, M_1=8$



(iii) $D=2, M_1=16$



(iv) $D=10, M_1=20$



(v) $D=10, M_1=80$

Figure 4.8: A comparison on synthetic PLP instances as in Figure 4.3 but with other choices of decision variable dimension D and inequality constraints M_1.

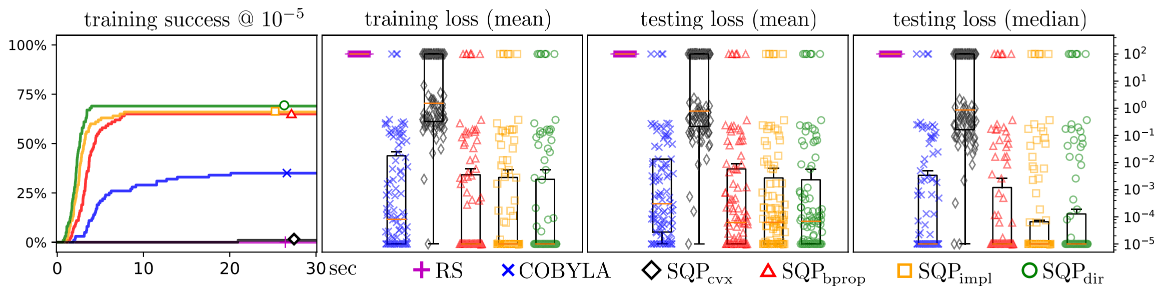


Figure 4.9: A comparison on synthetic PLP instances with parametric cost vectors ($D=10$), inequality constraints ($M_1=80$), and equality constraints ($M_2=2$).

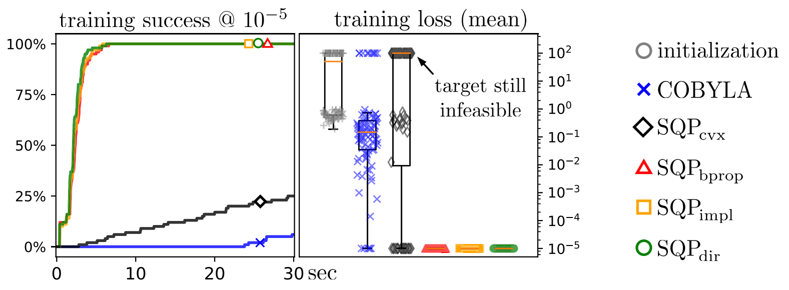


Figure 4.10: A comparison on synthetic LP instances ($D=10$, $M_1=80$).

in (4.12) below.

$$\begin{aligned} \underset{x_1,x_2}{\text{minimize}} \quad & -w_1u_1x_1 - w_2u_2x_2 \\ \text{subject to} \quad & x_1 + x_2 \le \max(1, u_1+u_2) \\ & 0 \le x_1 \le 1 \\ & 0 \le x_2 \le 1 \end{aligned} \tag{4.12}$$

The two training points are generated with $\mathbf{w} = (1,1)$ at $\mathbf{u}_1 = (1, \frac{1}{3})$ and $\mathbf{u}_2 = (1, \frac{1}{3})$ with testing point $\mathbf{u}_{\text{test}} = (\frac{1}{2}, \frac{5}{6})$. PLP learning was initialized at $\mathbf{w}_{\text{ini}} = (4,1)$ and the SQP_{impl} algorithm returned $\mathbf{w}_{\text{learned}} \approx (\frac{35}{9}, \frac{4}{3})$, used to generate the learned decision map depicted in the figure.

Chapter 5

Learning the Objective of Linear Programs: Models and Insights[1]

5.1 Abstract

Optimization models are widely used for assisting the decision-making processes in modern business organizations. Developing an optimization model that accurately represents the underlying decision-making process is challenging in practice. Inverse optimization infers model coefficients of an optimization model from the observed solution. In this work, we study the inverse linear optimization problem whose goal is to learn unknown parameters in the objective of a parametric linear program (PLP) from optimal decisions. We present four different methods: three single-level optimization models that are extensions of existing IO formulations, and one bilevel optimization model that is directly adapted from Chapter 4. We show that the three single-level optimization models are in fact mathematically equivalent under certain assumptions. Furthermore, we show that, despite mathematical equivalence, two of

[1]Some preliminary work of this project was conducted by Nima Sajedi under Yingcong Tan's supervision for the fulfillment of ENGR412 course requirements at Concordia University.

the three single-level optimization models tend to learn PLPs that generalize better to test data, empirically.

5.2 Introduction

Optimization is deeply embedded in various decision-making processes in modern organizations (e.g., budget planning, inventory management) and an individual's life (e.g., diet management, routing). One common practice to represent the decision-making process is through optimization modelling, which formally characterizes the objective (e.g., achievable goals, cost), constraints (e.g., restrictions on the resource) and the associated decision variables (e.g., choice of actions).

Conventionally, in an optimization problem, the goal is to determine the decision that optimizes an objective function subject to a set of constraints. We refer to this problem as the *forward optimization problem* (FOP). In inverse optimization (IO), the goal is to determine the coefficients of an FOP model such that a target solution is an optimal solution of the learned FOP model. Studying IO is particularly important in practice since perfect knowledge of the objective and constraints is not always available, and accurate model parameter estimation could be challenging. For instance, in a real-world routing problem, the users (e.g., drivers) of a given network would optimize their path from an origin to a destination based on some criteria. These criteria could be either network-specific (e.g., length of a road) or user-specific (e.g., type of vehicle, preference on toll roads) (Burton and Toint, 1992; Burton, 1993). These criteria would reflect the user's objective function that dictates their decisions, and thus it is crucial to collect information from the user when specifying the coefficients in the objective function. However, in some settings we may not have access to an individual driver. Instead, we can observe some routes that they took. Using IO, we can incorporate the observed routes into the optimization model and infer the objective function coefficients. The learned model can then be used for prediction (e.g., in road network planning) or recommendation (e.g., in customized navigation system).

In this paper, we study the IO problem whose FOP is a linear program (LP), and the goal is to learn the objective. We refer to this problem as the *inverse linear optimization* problem, denoted as ILOP-obj. Linear programming is an important class of optimization models, which deals with a linear objective, linear constraints and continuous decision variables. Its presence is ubiquitous in various fields, such as engineering, manufacturing and transportation. Studying the inverse linear optimization problem is important for the advancement in both theory and application. Additionally, we are interested in the parametric case, where the coefficients of a linear program also depend on other parameters, and the goal is to learn the unknown parameters from observed solutions.

The organization of this paper is as follows. Section 5.2 presents the introduction, related work and main contributions of this paper. Section 5.3 presents some preliminaries on linear programming, parametric linear programming and inverse linear optimization. We present four different mathematical formulations in Section 5.4 for solving ILOP-obj problems. We discuss the connections among the different mathematical formulations in Section 5.5. Section 5.6 and 5.7 present the experimental results and the discussion. Lastly, Section 5.8 concludes the chapter.

5.2.1 Related work

Inverse optimization problem was first described in Burton and Toint (1992) and Burton (1993). They studied the inverse shortest path problem, and the goal was to learn the travel time of edges based on a given path. Ahuja and Orlin (2001) studied the inverse linear optimization and proposes a general formulation to learn the cost vector that minimizes the L_p normal from a target cost vector. Early studies of inverse linear optimization focused on noise-free observations which are candidate optimal solutions of a given feasible region. Keshavarz et al. (2011) studied the IO problem under the assumption of noisy measurement. They studied the parametric convex optimization problem and propose a formulation based on the Karush-Kuhn-Tucker (KKT) conditions. Aswani et al. (2018) focused on the ILOP-obj problem with noisy-observations. To our knowledge, they were the first to view IO as a bilevel optimization problem. They proposed to reformulate the bilevel formulation to a single-level optimization model using the strong duality, and proposed two numerical algorithms that are shown to maintain statistical consistency. Tavaslıoğlu et al. (2018) formally defined and characterized the notion of *inverse feasible region*, that is, the collection of all cost vectors which make target solutions optimal. This concept was first described in Ahuja and Orlin (2001) as the *inverse feasible cost vectors*, and also studied in other papers Chan et al. (2019) and Gupta and Zhang (2020). Chan et al. (2019) studied the ILOP-obj problem with a single observation which is assumed to be feasible with respect to a given feasible region. Chan et al. proposed a mathematical formulation based on the strong duality. Notably, they derived a closed-form solution and analyzed different loss functions. Babier et al. (2020) generalized the work of Chan et al. (2019) to the case of multiple observations which are not restricted to be feasible with respect to the constraints. The same problem was studied in Gupta and Zhang (2020) who proposed a mixed-integer formulation based on the KKT-based formulation from Keshavarz et al. (2011).

The problem of learning the coefficients of an optimization model was also studied in other distinct areas. Bärmann et al. (2017, 2020) studied the IO problem under the online setting. Zimmermann and Frejinger (2020) explained the connections between inverse optimization and inverse reinforcement learning (IRL). By reformulating the forward problem as a dynamic model, some IO problems (e.g., inverse shortest path) can be solved using IRL methods. Amos and Kolter (2017) and Agrawal et al. (2019) studied differentiable optimization layer that can be embedded in deep neural networks. In particular, Amos and Kolter (2017) focused on *quadratic programming problems* and Agrawal et al. (2019) focused on *convex optimization problems*. They applied implicit differentiation to the KKT conditions to compute the gradients for updating the model coefficients. In Chapter 3 and 4, we leverage concepts and algorithms from deep learning literature, and study inverse linear optimization to learn the objective and constraints jointly. We propose a bilevel optimization problem and develop gradient-based algorithms to solve the bilevel formulation directly. However, like other gradient-based algorithms, this method can also get trapped in local optima. This method requires solving the linear programs repeatedly to compute the gradients, which could be computationally expensive, especially when the LP size is large.

Here we identify a few research gaps that motivate this work. First, most IO literature focuses on developing single-level optimization models that are then solved with commercial optimization solver. We notice that there is no existing work discussing the connections among different models or comparing their performance. Secondly, many gradient-based algorithms have been studied in the literature (Amos and Kolter (2017); Agrawal et al. (2019), Chapter 3 and 4). These papers mainly focus on solving the most general form of IO problem that is learning all model coefficients jointly. We are interested in adapting their gradient-based algorithms to the ILOP-obj problem and compare its performance with the MP models solved in CPLEX.

5.2.2 Contributions

Motivated by the arguments above, we study a general form of ILOP-obj problem with unknown parameters in the objective. The main contributions of this paper are listed below.

- We study the inverse linear optimization problems to learn the objective under the general parametric setting. We propose four different mathematical formulations, including three single-level optimization models which are extensions of existing IO literature, and one bilevel optimization model which is directly based on Chapter 4.
  - We specialize the method from Keshavarz et al. (2011) to parametric LP and propose a single-level optimization model using the KKT conditions. We refer to this model as the *KKT-based formulation*.
  - We generalize the method from Chan et al. (2019) to the parametric case and propose a single-level optimization model using the strong duality. We refer to this model as the *strong duality-based formulation*
  - We generalize the method from Tavaslıoğlu et al. (2018) to the parametric case and propose a single-level optimization model using the concept of inverse feasible region. We refer to this model as the *inverse feasible region-based formulation*.
  - We specialize the bilevel optimization model and gradient-based algorithm from Chapter 4 to learning the objective only.
- We generalize the concept of inverse feasible region from Tavaslıoğlu et al. (2018) to the parametric case, and define the *parametric inverse feasible region*.
- We prove that the three single-level optimization models mentioned above (i.e., KKT-based, strong duality-based and inverse feasible region-based formulations) are mathematically equivalent under the assumption of a non-empty *parametric inverse feasible region*.

- Computational results show that all three single-level optimization models solved in CPLEX are faster than the bilevel model solved with a gradient-based algorithm. Additionally, the inverse feasible region-based formulation has the shortest runtime.

- We show that PLP models learned from the KKT-based formulation and the strong duality-based formulations provide better predictions on testing data, empirically. Combining with the runtime comparison, we highlight the performance trade-off between the runtime and accuracy of the predictions.

5.3 Preliminaries

In this section, we provide some background information on linear programming, parametric linear programming and inverse linear optimization.

5.3.1 Linear Programming

Let $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{c} \in \mathbb{R}^D$ denote the primal decision variables and the cost vector respectively. Let $\mathbf{A} \in \mathbb{R}^{M_1 \times D}$ and $\mathbf{b} \in \mathbb{R}^{M_1}$ denote the coefficients of inequality constraints, $\mathbf{G} \in \mathbb{R}^{M_2 \times D}$ and $\mathbf{h} \in \mathbb{R}^{M_2}$ denote the coefficients of equality constraints. Lastly, $\boldsymbol{\lambda} \in \mathbb{R}^{M_1}$ and $\boldsymbol{\nu} \in \mathbb{R}^{M_2}$ denote the dual decision variables associated with the inequality and equality constraints of the primal LP model, respectively. We present the primal-dual pair of LP models below.

$$\begin{aligned} \text{(LP)} \ \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \le \mathbf{b} \\ & \mathbf{G}\mathbf{x} = \mathbf{h} \end{aligned} \qquad \begin{aligned} \text{(DP)} \ \underset{\boldsymbol{\lambda},\boldsymbol{\nu}}{\text{maximize}} \quad & \mathbf{b}^T\boldsymbol{\lambda} + \mathbf{h}^T\boldsymbol{\nu} \\ \text{subject to} \quad & \mathbf{A}^T\boldsymbol{\lambda} + \mathbf{G}^T\boldsymbol{\nu} = \mathbf{c} \\ & \boldsymbol{\lambda} \le \mathbf{0} \end{aligned}$$

5.3.2 Parametric Linear Programming

In this paper, we are interested in the parametric linear program (PLP) whose objective function and constraints depend on parameters $\mathbf{u}$ and $\mathbf{w}$. Let $\mathbf{u}$ denote the observable feature (e.g., time, weather conditions) which may vary with different observations, and $\mathbf{w}$ denote the unknown parameters that we aim to learn. In the ILOP-obj problem, the unknown parameter $\mathbf{w}$ occurs only in the objective. We provide the mathematical formulation below.

$$\begin{aligned} \text{PLP}(\mathbf{u}, \mathbf{w}) \ \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}(\mathbf{u}, \mathbf{w})^T \mathbf{x} \\ \text{subject to} \quad & \mathbf{A}(\mathbf{u})\mathbf{x} \ \leq \ \mathbf{b}(\mathbf{u}) \\ & \mathbf{G}(\mathbf{u})\mathbf{x} \ = \ \mathbf{h}(\mathbf{u}) \end{aligned} \tag{5.1}$$

5.3.3 Inverse Linear Optimization

In this paper, we study the inverse linear optimization problem to learn the unknown parameters in the objective of a PLP model. That is, we aim to learn $\mathbf{w}$ in a PLP model (i.e., Model (5.1)) from a set of *noise-free* observations. An observation consists of a pair of observable feature $\mathbf{u}$ and the corresponding observed solution $\mathbf{x}^{\text{obs}}$. An observation is called noise-free if the observed solution is believed to be a candidate optimal solution. In an LP, that means the observed solution lies on the boundary of the feasible region.

We first introduce the notation, and then generalize the definition of noise-free observations to the parametric setting. Let $\mathcal{I} = \{1, \ldots, N\}$ denote the index of observations. Correspondingly $\mathbf{u}_i$ and $\mathbf{x}_i^{\text{obs}}$ denote the observable feature and observed solution under an observation i respectively. Let $\mathcal{J} = \{1, \ldots, M_1\}$ and $\mathcal{K} = \{1, \ldots, M_2\}$ denote the indices of inequality and equality constraints. Correspondingly, $\mathbf{A}(\mathbf{u}_i)$, $\mathbf{b}(\mathbf{u}_i)$ denote the coefficients of inequality constraints, and

$\mathbf{G}(\mathbf{u}_i)$, $\mathbf{h}(\mathbf{u}_i)$ denote the coefficients of equality constraints.

Definition 5.1 *For a PLP with the form of Model (5.1), an observation i is called noise-free if $\mathbf{x}_i^{\text{obs}}$ lies on the boundary of the corresponding feasible region specified by $\mathbf{u}_i$, denoted by $\mathcal{F}_x(\mathbf{u}_i)$, where $\mathcal{F}_x(\mathbf{u}) = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{A}(\mathbf{u})\mathbf{x} \leq \mathbf{b}(\mathbf{u}),\ \mathbf{G}(\mathbf{u})\mathbf{x} = \mathbf{h}(\mathbf{u})\}$.*

Effectively, Definition 5.1 says that for a noise-free observation i, $\mathbf{x}_i^{\text{obs}}$ is a feasible solution on the boundary of the feasible region corresponding to $\mathbf{u}_i$. For this to be true, there should be at least one active constraint.

The prediction under an observation i, denoted by $\mathbf{x}_i^*$, is an optimal solution of $\text{PLP}(\mathbf{u}_i, \mathbf{w})$, i.e., $\mathbf{x}_i^{\text{obs}} \in \arg\min \text{PLP}(\mathbf{u}_i, \mathbf{w})$. The goal of ILOP-obj problem is to learn the unknown parameter $\mathbf{w}$ in the objective function to minimize the discrepancy between the observed solutions and predictions, i.e., $\text{minimize}_{\mathbf{w}}\ \ell(\mathbf{x}_i^{\text{obs}}, \mathbf{x}_i^*, \mathbf{u}_i, \mathbf{w})$. To measure the discrepancy, many loss functions ℓ have been studied in the literature including *absolute duality gap* (Chan et al., 2019), *absolute objective error* [2] (Tan et al., 2019, 2020), *squared decision error* (Tan et al., 2019; Agrawal et al., 2019), p-norm between the learned and target cost vectors (Ahuja and Orlin, 2001; Tavaslıoğlu et al., 2018) and *squared KKT residuals* (Keshavarz et al., 2011).

For the remaining of this paper, we assume all observations $\{(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}})\}_{i \in \mathcal{I}}$ are noise-free. Although some formulations presented in Section 5.4 are applicable to noisy observations, the discussion of noisy observations is beyond the focus on this paper and we leave that for future work.

5.4 Mathematical Formulations

In this section, we present four different methods for solving ILOP-obj in the parametric case. Firstly, we specialize the mathematical formulation from Keshavarz et al.

[2] Absolute duality gap and absolute objective error are equivalent under the LP setting.

(2011) to LP. This formulation uses the idea that if an observed solution is optimal, the corresponding KKT conditions must be satisfied. Secondly, we generalize the mathematical formulation from Chan et al. (2019) to the parametric case. This formulation uses the idea that if an observed solution is optimal, the strong duality must be satisfied. Thirdly, we generalize the formulation from Tavaslıoğlu et al. (2018) to the parametric case. This formulation uses the concept of inverse feasible region, that is, the set of all cost vectors making the observed solution optimal. Lastly, we specialize the bilevel optimization model and gradient-based algorithm from Chapter 4 to learning the objective only.

5.4.1 KKT-based Formulation

We specialize the formulation from Keshavarz et al. (2011) to LP and present the corresponding mathematical formulation:

$$\begin{aligned}
\underset{\mathbf{w},\boldsymbol{\lambda}_i,\boldsymbol{\nu}_i,\mathbf{r}_i^{\text{comp}},\mathbf{r}_i^{\text{stat}}}{\text{minimize}} \quad & \frac{1}{N}\sum_{i}^{N}\left(\|\mathbf{r}_i^{\text{stat}}\|_2^2 + \|\mathbf{r}_i^{\text{comp}}\|_2^2 \right) && (5.2a)\\
\text{subject to} \quad & \mathbf{r}_i^{\text{stat}} = \mathbf{c}(\mathbf{u}_i,\mathbf{w}) + \mathbf{A}(\mathbf{u}_i)^T\boldsymbol{\lambda}_i + \mathbf{G}(\mathbf{u}_i)^T\boldsymbol{\nu}_i, \quad i\in\mathcal{I} && (5.2b)\\
& \mathbf{r}_i^{\text{comp}} = \operatorname{diag}(\boldsymbol{\lambda}_i)\left(\mathbf{A}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} - \mathbf{b}(\mathbf{u}_i) \right), \quad i\in\mathcal{I} && (5.2c)\\
& \boldsymbol{\lambda}_i \geq \mathbf{0}, \quad i\in\mathcal{I} && (5.2d)\\
& \mathbf{w}\in\mathcal{W} && (5.2e)
\end{aligned}$$

The objective function captures the KKT residuals which measure the error in the constraint space. Constraint (5.2b) and (5.2c) define $\mathbf{r}_i^{\text{stat}}$ and $\mathbf{r}_i^{\text{comp}}$ which are the residuals in the stationary conditions and complementary slackness respectively. Constraint (5.2d) represents the dual feasibility and constraint (5.2e) represents prior knowledge on the unknown parameter $\mathbf{w}$. As discussed in Keshavarz et al. (2011), the residuals of primal feasibility (see the equations below) are fixed since the constraints do not depend on the unknown parameter $\mathbf{w}$. The residuals of primal feasibility can be checked trivially beforehand, and thus, can be omitted in Model (5.2). The expression of residuals in the primal inequality and equality constraints are presented as below

$$\mathbf{r}_i^{\text{ineq}} = (\ \mathbf{A}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} - \mathbf{b}(\mathbf{u}_i)\ )_+,\ i \in \mathcal{I} \qquad \mathbf{r}_i^{\text{eq}} = \mathbf{G}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} - \mathbf{h}(\mathbf{u}_i),\ i \in \mathcal{I}$$

On the one hand, we specialize the model from Keshavarz et al. (2011) to LP. On the other hand, the original model from Keshavarz et al. (2011) assumes the objective function of the FOP has the finite dimensional affine parametrization (i.e., obj $= \sum_{i=1}^{P} w_i f(\mathbf{x}, \mathbf{u})$, where f_i are pre-selected basis functions) and $w_i, i = 1, \ldots, P$ are the unknown parameters. Our formulation is not restricted to this assumption, and thus it is also a generalization of Keshavarz et al. (2011).

5.4.2 Strong Duality-based Formulation

We generalize the formulation from Chan et al. (2019) to the parametric case and propose the following mathematical formulation:

$$\begin{aligned}
\underset{\mathbf{w},\boldsymbol{\lambda}_i,\boldsymbol{\nu}_i,\epsilon_i}{\text{minimize}} \quad & \frac{1}{N}\sum_{i}^{N} |\epsilon_i|^2 && (5.3\text{a})\\
\text{subject to} \quad & \mathbf{A}(\mathbf{u}_i)^T\boldsymbol{\lambda}_i + \mathbf{G}(\mathbf{u}_i)^T\boldsymbol{\nu}_i = \mathbf{c}(\mathbf{u}_i, \mathbf{w}), \quad i \in \mathcal{I} && (5.3\text{b})\\
& \mathbf{c}(\mathbf{u}_i, \mathbf{w})^T\mathbf{x}_i^{\text{obs}} - \epsilon_i = \mathbf{b}(\mathbf{u}_i)^T\boldsymbol{\lambda}_i + \mathbf{h}(\mathbf{u}_i)^T\boldsymbol{\nu}_i, \quad i \in \mathcal{I} && (5.3\text{c})\\
& \boldsymbol{\lambda}_i \leq \mathbf{0}, \quad i \in \mathcal{I} && (5.3\text{d})\\
& \mathbf{w} \in \mathcal{W} && (5.3\text{e})
\end{aligned}$$

The objective function is a quadratic function that captures the error in the objective space, where $\epsilon_i, i \in \mathcal{I}$ represents the residual in the strong duality conditions. Constraint (5.3b) and (5.3d) represents dual feasibility. Constraint (5.3c) enforces strong duality. Constraint (5.3e) represents prior knowledge on $\mathbf{w}$. This formulation is built based on the assumption that the observed solutions $\mathbf{x}_i^{\text{obs}}$ are feasible (i.e., $\mathbf{A}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} \le \mathbf{b}(\mathbf{u}_i),\ \mathbf{G}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} = \mathbf{h}(\mathbf{u}_i),\ i \in \mathcal{I}$).

5.4.3 Inverse Feasible Region-based Formulation

We first introduce the inverse feasible region defined by Tavaslıoğlu et al. (2018) for the non-parametric case, and then generalize it to the parametric case.

Inverse Feasible Region for Non-Parametric LP

Tavaslıoğlu et al. (2018) defines the inverse feasible region of of an solution as the set of all cost vectors that make that solution optimal.

Definition 5.2 *For a linear program, the inverse feasible region of a solution* $\mathbf{x} \in \mathbb{R}^D$*, denoted by* $\mathcal{F}_c(\mathbf{x})$ *is the set of all cost vectors that make the observed solution optimal, i.e.,* $\mathcal{F}_c(\mathbf{x}) = \{\mathbf{c} \in \mathbb{R}^D \mid \mathbf{x} \in \arg\min \mathrm{LP}(\mathbf{c})\}$.

Recall that $\mathcal{J}$ and $\mathcal{K}$ denotes the set of inequality and equality constraints. Let $\mathcal{J}'$ denote the set of active inequality constraints of $\mathbf{x}^{\text{obs}}$, and we know $\mathcal{J}' \subseteq \mathcal{J}$. Let $\mathbf{a}_j$, b_j denote the j^{th} row of $\mathbf{A}$ and j^{th} element of $\mathbf{b}$ respectively, and $\mathbf{g}_k,\ h_k$ denote the k^{th} row of $\mathbf{G}$ and k^{th} element of $\mathbf{h}$ respectively. Given a noise-free solution $\mathbf{x} \in \mathbb{R}^D$, Tavaslıoğlu et al. (2018) showed that the corresponding inverse feasible region (see Definition 5.2) can be expressed in the following mathematical form.

$$\mathcal{F}_c(\mathbf{x}) = \operatorname{cone}(\{\, -\mathbf{a}_j \mid \mathbf{a}_j\mathbf{x} = b_j,\ j \in \mathcal{J}' \,\}) \bigoplus \operatorname{span}(\{\, -\mathbf{g}_k \mid k \in \mathcal{K} \,\})$$

$$= \left\{ \sum_{j\in\mathcal{J}} -\beta_j \mathbf{a}_j \;+\; \sum_{k\in\mathcal{K}} -\gamma_k \mathbf{g}_k \;\middle|\; \begin{array}{l} \beta_j \;=\; 0,\; j \in \mathcal{J} \setminus \mathcal{J}' \\ \beta_j \;\geq\; 0,\; j \in \mathcal{J}' \\ \gamma_k \in \mathbb{R} \end{array} \right\} \tag{5.4}$$

Where, $\bigoplus$ is the Minkowski sum [3], cone($\mathcal{P}$) denotes the conic hull of $\mathcal{P}$ and span($\mathcal{P}$) denotes the span of $\mathcal{P}$.

Inverse Feasible Region for Parametric LP

We consider a parametric LP whose cost vector depends on the observable feature $\mathbf{u}$ and unknown parameter $\mathbf{w} \in \mathcal{W}$ (see Model (5.1)). We first generalize the Definition 5.2 to the parametric LP whose constraints also depend on feature $\mathbf{u}$.

Definition 5.3 *Given a parametric LP with the form of Model (5.1) and a pair of observation* $(\mathbf{u}, \mathbf{x})$*, the inverse feasible region, denoted by* $\mathcal{F}_c(\mathbf{u}, \mathbf{x})$ *is the set of all cost vectors* $\mathbf{c}(\mathbf{u}, \mathbf{w})$ *that make solution* $\mathbf{x}$ *optimal, i.e.,* $\mathcal{F}_c(\mathbf{u}, \mathbf{x}) = \{\mathbf{c}(\mathbf{u}, \mathbf{w}) \in \mathbb{R}^D \mid \mathbf{x} \in \arg\min \mathrm{PLP}(\mathbf{u}, \mathbf{w})\}$.

Given a set of noise-free observations $\{(\mathbf{u}_i, \mathbf{x}_i^{\mathrm{obs}})\}_{i=1}^N$, $\mathcal{J}_i$ and $\mathcal{K}_i$ denote the set of inequality and equality constraints of $\mathrm{PLP}(\mathbf{u}_i)$, and $\mathcal{J}_i'$ denotes the set of active inequality constraints given $\mathbf{x}_i^{\mathrm{obs}}$. Let $\mathbf{a}_{ij},\ b_{ij},\ j \in \mathcal{J}_i$ denote the j^{th} row of $\mathbf{A}(\mathbf{u}_i)$ and j^{th} element of $\mathbf{b}(\mathbf{u}_i)$ respectively, and $\mathbf{g}_{ik},\ h_{ik},\ k \in \mathcal{K}_i$ denote the k^{th} row of $\mathbf{G}(\mathbf{u}_i)$ and k^{th} element of $\mathbf{h}(\mathbf{u}_i)$ respectively. We rewrite equation 5.4 as follows.

$$\mathcal{F}_c(\mathbf{u}_i, \mathbf{x}_i) = \left\{ \sum_{j\in\mathcal{J}_i} -\beta_{ij} \mathbf{a}_{ij} \;+\; \sum_{k\in\mathcal{K}_i} -\gamma_{ik} \mathbf{g}_{ik} \;\middle|\; \begin{array}{l} \beta_{ij} \;=\; 0,\; j \in \mathcal{J}_i \setminus \mathcal{J}_i' \\ \beta_{ij} \;\geq\; 0,\; j \in \mathcal{J}_i' \\ \gamma_{ik} \in \mathbb{R},\; k \in \mathcal{K}_i \end{array} \right\} \tag{5.5}$$

[3] Minkowski sum of two sets of vectors is formed by adding each vector in one set to each vector in the other set. Let $\mathcal{S}$ and $\mathcal{S}'$ denote two sets of vectors, $\mathcal{S} \bigoplus \mathcal{S}' \;=\; \{s_1 \;+\; s_2 \mid s_1 \in \mathcal{S},\; s_2 \in \mathcal{S}'\}$

Let $\mathcal{W}$ denote the feasible domain of $\mathbf{w}$. We define the *parametric inverse feasible region*, denoted by $\mathcal{F}_w(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}})$, as the set of parameter $\mathbf{w}$ that makes the observed solution $\mathbf{x}_i^{\text{obs}}$ optimal.

Definition 5.4 *Given a parametric LP with the form of Model (5.1) and a pair of observation* $(\mathbf{u}, \mathbf{x})$*, the parametric inverse feasible region of* $\mathbf{x}$ *is,* $\mathcal{F}_w(\mathbf{u}, \mathbf{x}) = \{\mathbf{w} \in \mathcal{W} \mid \mathbf{c}(\mathbf{u}, \mathbf{w}) \in \mathcal{F}_c(\mathbf{u}, \mathbf{x})\}$.

Definition 5.5 *Given a parametric LP with the form of Model (5.1) and multiple observations* $\{(\mathbf{u}_i,\ \mathbf{x}_i)\}_{i=1}^N$*, let* $\mathcal{X} = \{\mathbf{x}_1,\ \ldots,\ \mathbf{x}_N\}$*, the parametric inverse feasible region of* $\mathcal{X}$*, denoted by* $\mathcal{F}_w(\mathcal{X})$*, is the intersection of the parametric inverse feasible regions of each observed solution. That is,* $\mathcal{F}_w(\mathcal{X}) = \cap_{i \in \mathcal{I}} \mathcal{F}_w(\mathbf{u}_i, \mathbf{x}_i)$

Based on the Definition 5.4 and 5.5, we generalize the formulation from Corollary 14 in Tavaslıoğlu et al. (2018) to the parametric case. Given multiple noise-free observations $\{(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}})\}_{i=1}^N$, our mathematical model is presented below.

$$\begin{aligned}
\underset{\mathbf{w}, \beta_{ij}, \gamma_{ij}}{\text{minimize}} \quad & 0 && \text{(5.6a)}\\
\text{subject to} \quad & \mathbf{w} \in \mathcal{F}_w(\mathcal{X}^{\text{obs}}) && \text{(5.6b)}\\
& \mathcal{X}^{\text{obs}} = \{\mathbf{x}_1^{\text{obs}},\ \ldots,\ \mathbf{x}_N^{\text{obs}}\}
\end{aligned}$$

Constraint (5.6b) defines the parametric inverse feasible region of all observed solutions $\mathcal{F}_w(\mathcal{X}^{\text{obs}})$ which can be expressed explicitly as below.

$$\mathcal{F}_w(\mathcal{X}^{\text{obs}}) = \left\{ \mathbf{w} \in \mathcal{F}_w(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}}),\ i \in \mathcal{I} \right\} \qquad \text{(from } \textbf{Def. 5.5)}$$

$$\implies \quad = \left\{ \begin{array}{l} \mathbf{c}(\mathbf{u}_i, \mathbf{w}) \in \mathcal{F}_c(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}}),\ i \in \mathcal{I} \\ \mathbf{w} \in \mathcal{W} \end{array} \right\} \qquad \text{(from } \textbf{Def. 5.4)}$$

$$\implies \qquad = \left\{ \begin{aligned} & \mathcal{F}_c(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}}) \;=\; \sum_{j \in \mathcal{J}_i} -\beta_{ij}\mathbf{a}_{ij} \;+\; \sum_{k \in \mathcal{K}_i} -\gamma_{ik}\mathbf{g}_{ik},\; i \in \mathcal{I} \\ & \beta_{ij} = 0,\; j \in \mathcal{J}_i \setminus \mathcal{J}_i',\; i \in \mathcal{I} \\ & \beta_{ij} \geq 0,\; j \in \mathcal{J}_i',\; i \in \mathcal{I} \\ & \gamma_{ij} \in \mathbb{R},\; k \in \mathcal{K}_i,\; i \in \mathcal{I} \\ & \mathbf{c}(\mathbf{u}_i, \mathbf{w}) \in \mathcal{F}_c(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}}),\; i \in \mathcal{I} \\ & \mathbf{w} \in \mathcal{W} \end{aligned} \right\} \qquad \text{(from } \textbf{Eq. 5.5}\text{)}$$

Explicitly encoding $\mathcal{F}_w(\mathcal{X}^{\text{obs}})$ in Model (5.6b) requires a prepossessing step to compute all active inequality constraints (i.e., $\mathcal{J}_i' = \{j \in \mathcal{J}_i \mid \mathbf{a}_{ij}\mathbf{x}^{\text{obs}} = b_{ij}\},\ i \in \mathcal{I}$). The original formulation from Tavaslıoğlu et al. (2018) uses a loss function to minimize a p-norm of the learned cost vector from a target cost vector. In this study, we assume that such a target cost vector is not available, and thus, the optimization problem is naturally reduced to a feasibility problem. Lastly, we emphasize that Model (5.6) has a solution only if the following assumption is satisfied.

Assumption 5.1 *The parametric inverse feasible region of all noise-free observed solutions of a parametric LP is non-empty, i.e.,* $\mathcal{F}_w(\mathcal{X}^{obs}) \neq \emptyset$

Example Where Inverse Feasible Region for PLP is Empty

We show an example of inverse linear optimization problem that has no solution in Model (5.6). Firstly, we consider an PLP example whose mathematical formulation is presented below. We know that $w \in \mathbb{R}$, and there is a single observation ($u =$

$1,\ \mathbf{x}^{\text{obs}} = [-3, 2.5]$),

$$\begin{aligned}
\underset{x_1,x_2}{\text{minimize}} \quad & (-0.5 + wu)x_1 + (0.5 + wu)x_2 \\
\text{subject to} \quad & x_1 \leq 2 + u \\
& x_2 \leq 1.5 + u \\
& -x_1 \leq 2 + u \\
& -x_2 \leq 1.5 + u \\
& x_1 + x_2 \leq 3 + u
\end{aligned} \tag{5.7}$$

Secondly, we substitute ($u = 1,\ \mathbf{x}^{\text{obs}} = [-3, 2.5]$) into the constraints and find that the second and third constraints are active. We formulate the corresponding ILOP-obj problem using Model (5.6) and present the mathematical model below.

$$\begin{aligned}
\underset{w_1,w_2,\beta_2,\beta_3}{\text{minimize}} \quad & 0 \\
\text{subject to} \quad & \mathbf{w} \in \mathcal{F}_w(u = 1, \mathbf{x}^{\text{obs}} = [-3, 2.5])
\end{aligned} \tag{5.8}$$

where,

$$\mathcal{F}_w(u = 1, \mathbf{x}^{\text{obs}} = [-3, 2.5]) = \left\{ \begin{array}{l}
\mathcal{F}_c(\mathbf{x}^{\text{obs}}) = -\beta_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \beta_3 \begin{bmatrix} -1 \\ 0 \end{bmatrix} \\
\beta_2, \beta_3 \geq 0 \\
\begin{bmatrix} -0.5 + w \\ 0.5 + w \end{bmatrix} = \mathcal{F}_c(\mathbf{x}^{\text{obs}}) \\
w \in \mathbb{R}
\end{array} \right\}$$

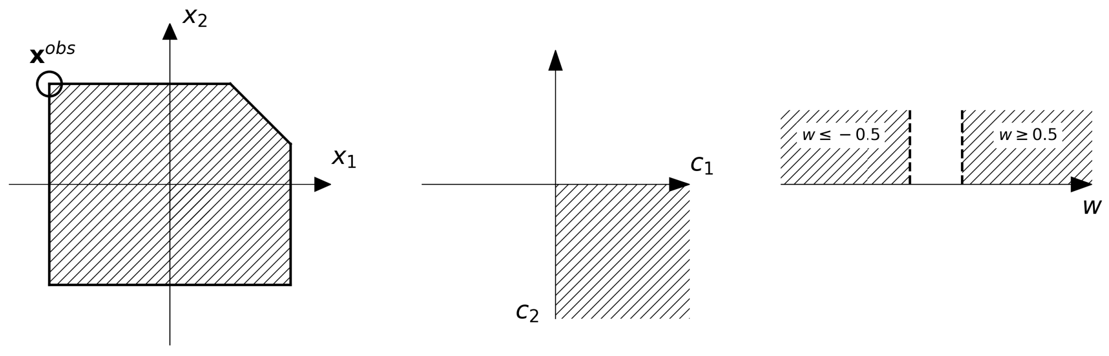
Lastly, we simplify the equation above as follows.

$$\mathcal{F}_w(u=1,\mathbf{x}^{\text{obs}}=[-3,2.5]) = \left\{\begin{array}{l} \begin{bmatrix}-0.5+w\\ 0.5+w\end{bmatrix} = -\beta_2\begin{bmatrix}0\\1\end{bmatrix} - \beta_3\begin{bmatrix}-1\\0\end{bmatrix} \\ \beta_2,\beta_3 \geq 0 \\ w \in \mathbb{R}\end{array}\right\}$$

$$= \left\{\begin{array}{l} w = \beta_3 + 0.5 \\ w = -\beta_2 - 0.5 \\ \beta_2,\ \beta_3 \geq 0 \\ w \in \mathbb{R}\end{array}\right\}$$

$$= \left\{\begin{array}{l} w \geq 0.5 \\ w \leq -0.5\end{array}\right\} = \emptyset$$

As shown above, the parametric inverse feasible region is empty, and thus Model (5.8) has no solution in $\mathbf{w}$ space. We illustrate the feasible region of the FOP (see Model (5.7)), inverse feasible region $\mathcal{F}_c(\mathbf{x}^{\text{obs}})$ and the parametric inverse feasible region $\mathcal{F}_w(\mathbf{x}^{\text{obs}})$ in Figure 5.1.



(a) Feasible Region of PLP. (b) $\mathcal{F}_c(\mathbf{x}^{\text{obs}})$. (c) $\mathcal{F}_w(\mathbf{x}^{\text{obs}})$.

Figure 5.1: Illustration of the feasible region of the example PLP in Model (5.7), inverse feasible region $\mathcal{F}_c(\mathbf{x}^{\text{obs}})$ and the parametric inverse feasible region $\mathcal{F}_w(u, \mathbf{x}^{\text{obs}})$ of the corresponding ILOP-obj problem.

5.4.4 Bi-level Formulation

We specialize the bilevel formulation from Chapter 4 to the case of learning the objective only and present the corresponding mathematical formulation below.

$$\underset{\mathbf{w}}{\text{minimize}} \quad \tfrac{1}{N}\textstyle\sum_{i=1}^{N} \mid \mathbf{c}^T(\mathbf{u}_i, \mathbf{w})(\mathbf{x}_i^{\text{obs}} - \mathbf{x}_i^*) \mid \tag{5.9a}$$

$$\text{subject to} \quad \mathbf{x}_i^* \in \underset{\mathbf{x}}{\arg\min} \left\{ \mathbf{c}(\mathbf{u}_i, \mathbf{w})^T \mathbf{x} \;\middle|\; \begin{array}{rcl} \mathbf{A}(\mathbf{u}_i)\mathbf{x} & \leq & \mathbf{b}(\mathbf{u}_i) \\ \mathbf{G}(\mathbf{u}_i)\mathbf{x} & = & \mathbf{h}(\mathbf{u}_i) \end{array} \right\},\ i \in \mathcal{I} \tag{5.9b}$$

$$\mathbf{w} \in \mathcal{W} \tag{5.9c}$$

The objective is to minimize the difference in objective between the observed solution and prediction. The loss function is also referred to as the *absolute objective error* (AOE) in Chapter 4. The prediction is the optimal solution of a PLP model that is defined in the inner problem. $\mathcal{W}$ in the last constraint represents prior knowledge on the unknown parameter.

We found no suitable commercial optimization solvers that can solve a bilevel optimization model directly. To solve Model (5.9), we specialize the gradient-based algorithm from Chapter 4. Their algorithm uses a gradient-based non-linear programming algorithm, that is, the *sequential quadratic programming* (SQP) algorithm to solve the outer problem. They differentiate through an LP to compute the gradients of the model coefficients. Chapter 4 studies 4 different approaches for computing the gradients, and a closed-form approach called `direct` shows performance superiority in runtime. Thus, this is the method we employ here.

5.5 Mathematical Equivalence

In this section, we analyze the three single-level models: Model (5.2), (5.3) and (5.6), and show that they are mathematically equivalent when Assumption 5.1 is satisfied. We state this formally in Theorem 5.1 and 5.2 where we provide our definition of mathematical equivalence.

Theorem 5.1 *Given a parametric LP with the form of Model (5.1) and a set of noise-free observations* $\{(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}})\}_{i\in\mathcal{I}}$*, if Assumption 5.1 is satisfied, a solution is feasible for Model (5.6) if and only if it is optimal for Model (5.2).*

Theorem 5.2 *Given a parametric LP with the form of Model (5.1) and a set of noise-free observations* $\{(\mathbf{u}_i, \mathbf{x}_i^{\text{obs}})\}_{i\in\mathcal{I}}$*, if Assumption 5.1 is satisfied, a solution is feasible for Model (5.6) if and only if it is optimal for Model (5.3).*

5.5.1 Proof of Theorem 5.1

Here, we present the proof for Theorem 1.

Proof Firstly, we present the proof of the forward direction, i.e., a feasible solution of Model (5.6) is optimal for Model (5.2). If Assumption 5.1 is satisfied, we know that there exists a feasible solution $\{\, \hat{\mathbf{w}},\ \hat{\beta}_{ij},\ \hat{\gamma}_{ik},\ j \in \mathcal{J}_i,\ k \in \mathcal{K}_i,\ i \in \mathcal{I}\,\}$ to Model (5.6), and we have the following equations.

$$\begin{aligned}
&\mathbf{c}(\mathbf{u}_i, \hat{\mathbf{w}}) \ = \ \sum_{j\in\mathcal{J}_i} -\hat{\beta}_{ij}\mathbf{a}_{ij} \ + \ \sum_{k\in\mathcal{K}_i} -\gamma_{ik}\mathbf{g}_{ik}, \quad i \in \mathcal{I} && \text{(5.10a)}\\
&\hat{\beta}_{ij} \geq 0, \ j \in \mathcal{J}_i', \ i \in \mathcal{I} && \text{(5.10b)}\\
&\hat{\beta}_{ij} = 0, \ j \in \mathcal{J}_i \setminus \mathcal{J}_i', \ i \in \mathcal{I} && \text{(5.10c)}\\
&\hat{\gamma_{ij}} \in \mathbb{R}, \ k \in \mathcal{K}_i, \ i \in \mathcal{I} && \text{(5.10d)}\\
&\hat{\mathbf{w}} \in \mathcal{W} && \text{(5.10e)}
\end{aligned}$$

Combining the equation (5.10b) - (5.10c) and the definition of $\mathcal{J}_i' = \{j \in \mathcal{J}_i, \ i \in \mathcal{I} \mid \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ = \ 0\}$, $i \in \mathcal{I}$, we know the following:

$$\begin{aligned}
&\left\{\begin{aligned}
&\text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ < \ 0, \text{then } \ \hat{\beta}_{ij} = 0, \ j \in \mathcal{J}_i \setminus \mathcal{J}_i' \\
&\text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ = \ 0, \text{then } \ \hat{\beta}_{ij} \geq 0, \ j \in \mathcal{J}_i'
\end{aligned}\right\}, \ i \in \mathcal{I} \\
\Longrightarrow \quad & \hat{\beta}_{ij}(\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij}) \ = \ 0, \ j \in \mathcal{J}_i, \ i \in \mathcal{I} && \text{(5.11)}
\end{aligned}$$

Let $\hat{\boldsymbol{\beta}}_i = \begin{bmatrix}\hat{\beta}_{ij}\end{bmatrix}_{j\in\mathcal{J}_i}$, $\hat{\boldsymbol{\gamma}}_i = \begin{bmatrix}-\hat{\gamma}_{ik}\end{bmatrix}_{k\in\mathcal{K}_i}$, $i \in \mathcal{I}$, we can rewrite the equation (5.10a) and equation (5.11) in the matrix form below.

$$\begin{aligned}
\mathbf{0} \ &= \ \mathbf{c}_i(\mathbf{u}_i, \mathbf{w}) + \mathbf{A}(\mathbf{u}_i)^T\hat{\boldsymbol{\beta}}_i + \mathbf{G}(\mathbf{u}_i)^T\hat{\boldsymbol{\gamma}}_i, \ i \in \mathcal{I} && \text{(5.12)}\\
\mathbf{0} \ &= \ \text{diag}(\hat{\boldsymbol{\beta}}_i)(\mathbf{A}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} - \mathbf{b}(\mathbf{u}_i)), \ i \in \mathcal{I} && \text{(5.13)}
\end{aligned}$$

We make the following observations. 1). equations (5.12) and (5.13) are equivalent to the constraints (5.2b) and (5.2c) with $\mathbf{r}_i^{\text{comp}} = \mathbf{0}$, $\mathbf{r}_i^{\text{stat}} = \mathbf{0}$, $i \in \mathcal{I}$; 2). the objective value $\frac{1}{N}\sum_i^N(\ \|\mathbf{r}_i^{\text{stat}}\|_2^2 \ + \ \|\mathbf{r}_i^{\text{comp}}\|_2^2 \ ) = 0$; 3). the solution $\{ \ \hat{\mathbf{w}}, \hat{\beta}_{ij}, \hat{\gamma}_{ik}, \ j \in \mathcal{J}_i, \ k \in \mathcal{K}_i, \ i \in \mathcal{I} \ \}$ satisfies the constraints (5.2d) and (5.2e).

In summary, $\{ \ \hat{\mathbf{w}}, \ \hat{\beta}_{ij}, \ \hat{\gamma}_{ik}, \ \mathbf{r}_i^{\text{comp}} \ = \mathbf{0}, \ \mathbf{r}_i^{\text{stat}} \ = \mathbf{0}, \ j \in \mathcal{J}_i, \ k \in \mathcal{K}_i, \ i \in \mathcal{I}\}$ is an optimal solution of Model (5.2) since it is feasible for Model (5.2) and the corresponding objective value equals the lower bound of the objective function (recall that, Model (5.2) has a quadratic objective function whose minimum value is zero).

Secondly, we present the proof of the reverse direction. That is, an optimal solution of Model (5.2) is also feasible for Model (5.6). Let $\{\hat{\lambda}_{ij},\ \hat{\nu}_{ik},\ \hat{\mathbf{w}},\ \hat{\mathbf{r}}_i^{\text{stat}},\ \hat{\mathbf{r}}_i^{\text{comp}},\ j \in \mathcal{J}_i,\ k \in \mathcal{K}\ i \in \mathcal{I}\}$ be an optimal solution of Model (5.2). Since all observations are noise-free, we know that 1). $\hat{\mathbf{r}}_i^{\text{stat}} = \mathbf{0},\ \hat{\mathbf{r}}_i^{\text{comp}} = \mathbf{0},\ i \in \mathcal{I}$ (recall that Model (5.2) has a quadratic objective function whose minimum value is zero); 2). the solution must satisfy all constraints in Model (5.2).

Since $\hat{\mathbf{r}}_i^{\text{comp}} = \mathbf{0},\ i \in \mathcal{I}$, we can rewrite constraint (5.2c) as the follows:

$$\mathbf{0} \ = \ \hat{\lambda}_{ij}\,(\,\mathbf{A}(\mathbf{u}_i)\mathbf{x}_i^{\text{obs}} - \mathbf{b}(\mathbf{u}_i)\,), \quad j \in \mathcal{J}_i,\ i \in \mathcal{I}$$

$$\Longrightarrow \qquad \left\{\begin{matrix} \text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ < \ 0, \text{then} \ \ \hat{\lambda}_{ij} = 0 \\ \text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ = \ 0, \text{then} \ \ \hat{\lambda}_{ij} \geq 0 \end{matrix}\right\},\ j \in \mathcal{J}_i,\ i \in \mathcal{I}$$

Given the definition of the index set of active constraints $\mathcal{J}_i' = \{j \in \mathcal{J}_i,\ i \in \mathcal{I} \mid \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ = \ 0\},\ i \in \mathcal{I}$, we can rewrite the equations above as the following:

$$\left\{\begin{matrix} \hat{\lambda}_{ij} = 0,\ j \in \mathcal{J}_i \setminus \mathcal{J}_i' \\ \hat{\lambda}_{ij} \geq 0,\ j \in \mathcal{J}_i' \end{matrix}\right\},\ i \in \mathcal{I} \tag{5.14}$$

Since $\mathbf{r}_i^{\text{stat}} = \mathbf{0}$, we can simply constraint (5.2b) below:

$$\mathbf{c}(\mathbf{u}_i, \mathbf{w}) \ = \ \sum_{j \in \mathcal{J}_i} -\hat{\lambda}_{ij}\mathbf{a}_{ij} + \sum_{k \in \mathcal{K}_i} -\hat{\nu}_{ij}\mathbf{g}_{ij}, \quad i \in \mathcal{I} \tag{5.15}$$

Combining equation (5.14), (5.15) and constraints (5.2e), we obtain the following system of equations.

$$\left\{\begin{array}{l} \mathbf{c}(\mathbf{u}_i,\mathbf{w}) \;=\; \sum_{j\in\mathcal{J}_i} -\hat{\lambda}_{ij}\mathbf{a}_{ij} \;+\; \sum_{k\in\mathcal{K}_i} -\hat{\nu}_{ik}\mathbf{g}_{ik},\; i\in\mathcal{I} \\ \hat{\lambda}_{ij} = 0,\; j\in\mathcal{J}_i\setminus\mathcal{J}_i',\; i\in\mathcal{I} \\ \hat{\lambda}_{ij}\geq 0,\; j\in\mathcal{J}_i',\; i\in\mathcal{I} \\ \hat{\nu}_{ik}\in\mathbb{R},\; k\in\mathcal{K}_i,\; i\in\mathcal{I} \\ \hat{\mathbf{w}}\in\mathcal{W} \end{array}\right\} \tag{5.16}$$

As shown in equation (5.16), it is obvious that $\{\hat{\lambda}_{ij},\ \hat{\nu}_{ik},\ \hat{\mathbf{w}},\ \hat{\mathbf{r}}_i^{\text{stat}},\ \hat{\mathbf{r}}_i^{\text{comp}},\ j\in\mathcal{J}_i,\ k\in\mathcal{K}\ i\in\mathcal{I}\}$ satisfies the constraints in Model (5.6) (see the definition of $\mathcal{F}_w(\mathcal{X}^{\text{obs}})$). ■

5.5.2 Proof of Theorem 5.2

Proof Firstly, we present the proof of the forward direction. That is, a feasible solution of Model (5.6) is optimal for Model (5.3). If Assumption 5.1 is satisfied, we know that there exists a feasible solution $\{\ \hat{\mathbf{w}},\hat{\beta}_{ij},\hat{\gamma}_{ik},\ j\in\mathcal{J}_i,\ k\in\mathcal{K}_i,\ i\in\mathcal{I}\ \}$ to Model (5.6), and we have the same system of equations as (5.10a) - (5.10e).

We multiple both sides of equation (5.10a) with $\mathbf{x}_i^{\text{obs}}$, and obtain the following equation.

$$\mathbf{c}(\mathbf{u}_i,\mathbf{w})\mathbf{x}_i^{\text{obs}} \;=\; \sum_{j\in\mathcal{J}_i'} -\hat{\beta}_{ij}\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \;+\; \sum_{k\in\mathcal{K}} -\hat{\gamma}_{ik}\,\underbrace{\mathbf{g}_{ik}\mathbf{x}_i^{\text{obs}}}_{=\,h_{ij}},\quad i\in\mathcal{I} \tag{5.17}$$

Combining the equation (5.10b) - (5.10d) and the definition of $\mathcal{J}_i' = \{j\in\mathcal{J}_i,\ i\in$

$\mathcal{I} \mid \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \;-\; b_{ij} \;=\; 0\}$, $i \in \mathcal{I}$, we know the following:

$$\left\{\begin{array}{l}\text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \;-\; b_{ij} \;<\; 0, \text{then } \;\hat{\beta}_{ij} = 0,\; j \in \mathcal{J}_i \setminus \mathcal{J}_i' \\ \text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \;-\; b_{ij} \;=\; 0, \text{then } \;\hat{\beta}_{ij} \geq 0,\; j \in \mathcal{J}_i'\end{array}\right\},\; i \in \mathcal{I}$$

$$\Longrightarrow \qquad \hat{\beta}_{ij}(\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}}) \;=\; \hat{\beta}_{ij}b_{ij},\; j \in \mathcal{J}_i,\; i \in \mathcal{I}$$

Therefore, equation (5.17) can be written as the following:

$$\mathbf{c}(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} \;=\; \sum_{j \in \mathcal{J}_i'} -\hat{\beta}_{ij}b_{ij} \;+\; \sum_{k \in \mathcal{K}} -\hat{\gamma}_{ik}h_{ij}, \quad i \in \mathcal{I} \tag{5.18}$$

Let $\hat{\boldsymbol{\beta}}_i = \Big[-\hat{\beta}_{ij}\Big]_{j \in \mathcal{J}_i}$, $\hat{\boldsymbol{\gamma}}_i = \Big[-\hat{\gamma}_{ik}\Big]_{k \in \mathcal{K}}$, $i \in \mathcal{I}$, we can rewrite the equation (5.10a) and (5.18) in the matrix form below.

$$\mathbf{c}_i(\mathbf{u}_i, \mathbf{w}) = \mathbf{A}(\mathbf{u}_i)^T\hat{\boldsymbol{\beta}}_i + \mathbf{G}(\mathbf{u}_i)^T\hat{\boldsymbol{\gamma}}_i,\; i \in \mathcal{I} \tag{5.19a}$$

$$\mathbf{c}_i(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} = \mathbf{b}(\mathbf{u}_i)^T\hat{\boldsymbol{\beta}}_i + \mathbf{h}(\mathbf{u}_i)^T\hat{\boldsymbol{\gamma}}_i,\; i \in \mathcal{I} \tag{5.19b}$$

We make the following observations. 1). equations (5.19a) and (5.19b) are equivalent to the constraints (5.3b) - (5.3c) with $\epsilon_i \;=\; 0,\; i \in \mathcal{I}$; 2). the corresponding objective value $\frac{1}{N}\sum_i^N \;|\epsilon_i|^2 = 0$; 3). $\{\;\hat{\mathbf{w}},\; \hat{\beta}_{ij},\; \hat{\gamma}_{ik},\; j \in \mathcal{J}_i,\; k \in \mathcal{K}_i,\; i \in \mathcal{I}\;\}$ satisfies the constraints (5.3d) and (5.3e).

In summary, $\{\;\hat{\mathbf{w}}, \hat{\beta}_{ij}, \hat{\gamma}_{ik},\; \epsilon_i \;=\; 0,\; j \in \mathcal{J}_i,\; k \in \mathcal{K}_i,\; i \in \mathcal{I}\}$ is an optimal solution of Model (5.3) since it is feasible for Model (5.3) and the corresponding objective value equals the lower bound of the objective function (recall that, Model (5.3) has a quadratic objective function (i.e., eq. (5.3a)) whose minimum value is zero)).
Secondly, we present the proof of the reverse direction. That is, an optimal solution of Model (5.3) is also feasible for Model (5.6). Let $\{\hat{\lambda}_{ij},\; \hat{\nu}_{ik},\; \hat{\mathbf{w}},\; \hat{\epsilon}_i,\; k \in \mathcal{K}\; i \in \mathcal{I}\}$ be

an optimal solution of Model (5.3). Since all observations are noise-free, we know that 1). $\hat{\epsilon}_i = 0,\ i \in \mathcal{I}$ (recall that Model (5.3) has a quadratic objective function whose minimum value is zero); 2). the solution satisfies all constraints in Model (5.3). We substitute $\{\hat{\lambda}_{ij},\ \hat{\nu}_{ik},\ \hat{\mathbf{w}},\ \epsilon_i,\ k \in \mathcal{K}\ i \in \mathcal{I}\}$ into the constraints (5.3b) - (5.3e), and obtain the following euqations.

$$\mathbf{c}(\mathbf{u}_i, \mathbf{w}) \ = \sum_{j \in \mathcal{J}_i} \hat{\lambda}_{ij}\mathbf{a}_{ij} \ + \sum_{k \in \mathcal{K}_i} + \hat{\nu}_{ik}\mathbf{g}_{ik},\ i \in \mathcal{I} \tag{5.20a}$$

$$\mathbf{c}(\mathbf{u}_i, \mathbf{w})\mathbf{x}_i^{\text{obs}} \ = \sum_{j \in \mathcal{J}_i} \hat{\lambda}_{ij} b_{ij} \ + \sum_{k \in \mathcal{K}_i} + \hat{\nu}_{ik} h_{ik},\ i \in \mathcal{I} \tag{5.20b}$$

$$\hat{\lambda}_{ij} \le 0,\ j \in \mathcal{J}_i,\ i \in \mathcal{I} \tag{5.20c}$$

$$\hat{\mathbf{w}} \in \mathcal{W} \tag{5.20d}$$

We substitute equation (5.20a) into equation (5.20b), and then obtain the follows:

$$\sum_{j \in \mathcal{J}_i} \hat{\lambda}_{ij}\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ + \sum_{k \in \mathcal{K}_i} + \hat{\nu}_{ik}\mathbf{g}_{ik}\mathbf{x}_i^{\text{obs}} \ = \sum_{j \in \mathcal{J}_i} \hat{\lambda}_{ij} b_{ij} \ + \sum_{k \in \mathcal{K}_i} + \hat{\nu}_{ik} h_{ik},\ i \in \mathcal{I}$$

$$\Longrightarrow \sum_{j \in \mathcal{J}_i} \hat{\lambda}_{ij}(\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} - b_{ij}) \ + \sum_{k \in \mathcal{K}_i} + \underbrace{\hat{\nu}_{ik}(\mathbf{g}_{ik}\mathbf{x}_i^{\text{obs}} - h_{ik})}_{=0} \ = \ 0,\ i \in \mathcal{I}$$

$$\Longrightarrow \sum_{j \in \mathcal{J}_i} \hat{\lambda}_{ij}(\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} - b_{ij}) \ = \ 0$$

Since $\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} - b_{ij} \le 0,\ \hat{\lambda}_{ij} \le 0,\ j \in \mathcal{J}_i,\ i \in \mathcal{I}$ (i.e., primal LP inequality constraints), we rewrite the equation above as the follows:

$$\hat{\lambda}_{ij}(\mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} - b_{ij}) \ = \ 0,\ j \in \mathcal{J}_i,\ i \in \mathcal{I}$$

$$\Longrightarrow \left\{ \begin{aligned} &\text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ < \ 0, \text{then } \ \hat{\lambda}_{ij} = 0 \\ &\text{if } \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} \ - \ b_{ij} \ = \ 0, \text{then } \ \hat{\lambda}_{ij} \le 0 \end{aligned} \right\},\ j \in \mathcal{J}_i,\ i \in \mathcal{I}$$

Given the definition of the index set of active constraints $\mathcal{J}'_i = \{j \in \mathcal{J}_i,\ i \in$

$\mathcal{I} \mid \mathbf{a}_{ij}\mathbf{x}_i^{\text{obs}} - b_{ij} = 0\}$, $i \in \mathcal{I}$, we can rewrite the equations above as the following:

$$\left\{\begin{array}{l} \hat{\lambda}_{ij} = 0,\ j \in \mathcal{J}_i \setminus \mathcal{J}_i' \\ \hat{\lambda}_{ij} \leq 0,\ j \in \mathcal{J}_i' \end{array}\right\},\ i \in \mathcal{I} \tag{5.21}$$

Let $\theta_{ij} = -\lambda_{ij},\ j \in \mathcal{J}_i,\ i \in \mathcal{I}$, we can rewrite equation (5.20a) - (5.20d) as follows:

$$\left\{\begin{array}{l} \mathbf{c}(\mathbf{u}_i, \mathbf{w}) = \sum\limits_{j \in \mathcal{J}_i} -\hat{\theta}_{ij}\mathbf{a}_{ij} + \sum\limits_{k \in \mathcal{K}_i} -\hat{\nu}_{ik}\mathbf{g}_{ik},\ i \in \mathcal{I} \\ \hat{\theta_{ij}} = 0,\ j \in \mathcal{J}_i \setminus \mathcal{J}_i',\ i \in \mathcal{I} \\ \hat{\theta_{ij}} \geq 0,\ j \in \mathcal{J}_i',\ i \in \mathcal{I} \\ \hat{\gamma}_{ik} \in \mathbb{R},\ k \in \mathcal{K}_i,\ i \in \mathcal{I} \\ \hat{\mathbf{w}} \in \mathcal{W} \end{array}\right\} \tag{5.22}$$

As shown in equation (5.22), it is obvious that $\{\hat{\lambda}_{ij},\ \hat{\nu}_{ik},\ \hat{\mathbf{w}},\ \hat{\epsilon}_i,\ k \in \mathcal{K}\ i \in \mathcal{I}\}$ satisfies the constraints in Model (5.6). ∎

5.6 Experimental Results

We test all four formulations presented in Section 5.4 on a range of synthetic parametric LPs and parametric multi-commodity flow problem instances. For each instance, we generate 20 pairs of $(\mathbf{u}, \mathbf{x}_i^{\text{obs}})$ as training data.

We solve Model (5.2), (5.3) and (5.6) using CPLEX Optimization Studio 12.10. with the following configuration: use barrier method in CPLEX; use a single core; set runtime limit as $30s$. We adapt the gradient algorithm from Chapter 4 to lean the objective only, which is then used for solving Model (5.9). We denote this gradient method as GradAlgo. In particular, we use the following algorithm configuration: $\ell = \text{AOE} = \sum_{i=1}^{N} \frac{1}{N} \mid \mathbf{c}(\mathbf{u}, \mathbf{w})(\mathbf{x}^{\text{obs}} - \mathbf{x}^*) \mid$ and `gradient_method = direct`. The

GradAlgo also uses PyTorch v1.6 nightly build and SciPy v1.4.1 (for SQP algorithm). We code the complete experiment in Python 3.7.9., and we run the experiment on an Intel Core i7 with 16GB RAM.

5.6.1 Instance Generation

Synthetic Parametric Linear Program We generate a range of synthetic parametric LP instances following the instance generation procedure from Chapter 3 , and introduce a linear function of two parameters w_1, w_2 and feature u to the objective, i.e., $c_a(w_1, w_2, u) = c_a^{\text{base}} + w_1 + w_2 u$, where c_a^{base}, $a = 1, \dots D$ are known constants. We generate 100 instances for each of the following six problem sizes: $D = 2$, $M_1 = \{4, 8, 16\}$ and $D = 10$, $M_1 = \{20, 36, 80\}$. Since the cost vector is linear with respect to w_1, w_2, Model (5.2) and (5.3) are quadratic programming models with linear constraints and quadratic objective functions. Correspondingly, (5.6) is a feasibility problem with linear constraints.

Multi-Commodity Flow (MCF) We follow the instance generation procedure in Chapter 4, and we use a slightly different parametric function for the arc cost, $c_a(t, l_a, p_a) = l_a + w_1 p_a + w_2 l_a(\sin(2\pi(t + l_a)) + 1)$ based on global feature t (time of day) and arc-specific features l_a (length) and p_a (toll price). Since the periodic term $\sin(2\pi(t + l_a))$ does not depend on the parameter w_1, w_2, the arc cost function $c_a(t, l_a, p_a)$ is a linear function with respect to w_1, w_2. Model (5.2) and (5.3) are quadratic programming models with linear constraints and quadratic objective function. Correspondingly, (5.6) is a feasibility problem with linear constraints.

5.6.2 Evaluating Prediction Accuracy

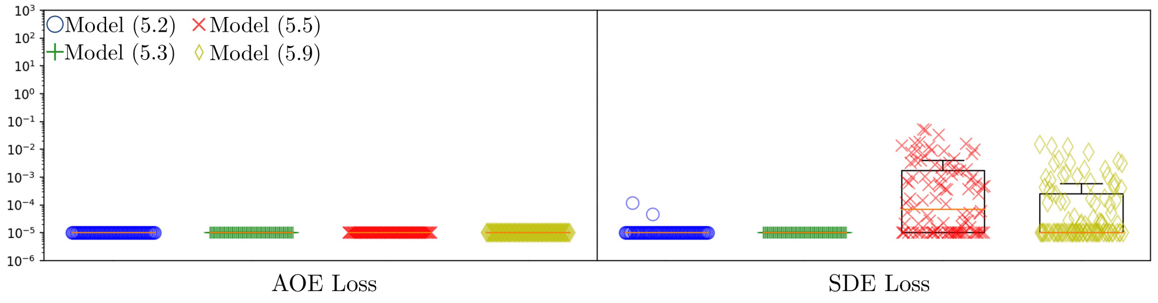
The ultimate goal of IO is to use the learned FOP models under novel conditions and then produce optimal solutions for assisting real-life decision-making. Therefore, it is important to evaluate the performance of the learned models. In particular, we use two loss functions which are the absolute objective error (AOE) and squared decision error (SDE) from Chapter 4.

$$\begin{aligned}
\text{AOE}(\mathbf{u}, \mathbf{w}) &= \sum_{i=1}^{N} \frac{1}{N} | \, \mathbf{c}(\mathbf{u}, \mathbf{w})^T (\mathbf{x}_i^{\text{obs}} - \mathbf{x}_i^*) \, | \\
\text{SDE}(\mathbf{u}, \mathbf{w}) &= \sum_{i=1}^{N} \frac{1}{N} \| \, (\mathbf{x}_i^{\text{obs}} - \mathbf{x}_i^*) \, \|_2^2 \\
&\text{where, } \mathbf{x}_i^* \in \arg\min \text{PLP}(\mathbf{u}_i, \mathbf{w})
\end{aligned}$$

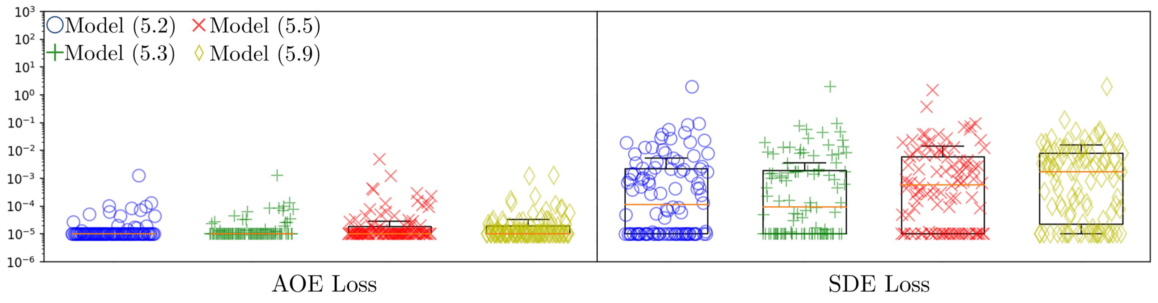
AOE captures the error in the objective space and SDE captures the error in the decision space. The complete results for PLP instances with $D = 10$, $M_1 = 80$ is presented in Figure 5.2, and the complete result for MCF instances is presented in Figure 5.3. Note, the y-axis represent the log-scaled loss value. (The complete results for other PLP instances are presented in the Appendix.)

Evaluate the learned model on training data As described in Section 5.6, we have have $N = 20$ training data for each instance. The training data are pairs of observable feature $\mathbf{u}_i^{\text{train}}$ and observed solutions $\mathbf{x}_i^{\text{obs}}$. Correspondingly, the predictions $\mathbf{x}_i^*$ are the optimal solutions of the learned model (i.e., $\mathbf{x}_i^* \in \arg\min \text{PLP}(\mathbf{u}_i^{\text{train}}, \mathbf{w}^{\text{tru}})$, $i = 1, \ldots, N$). We compute both $\text{AOE}(\mathbf{u}_i^{\text{train}}, \mathbf{w}^{\text{lrn}})$ and $\text{SDE}(\mathbf{u}_i^{\text{train}}, \mathbf{w}^{\text{lrn}})$ loss on the training data.

The main observations are as follows. Although Model (5.2), (5.3) and (5.6) have different objective functions, we notice that they all achieved effectively zero

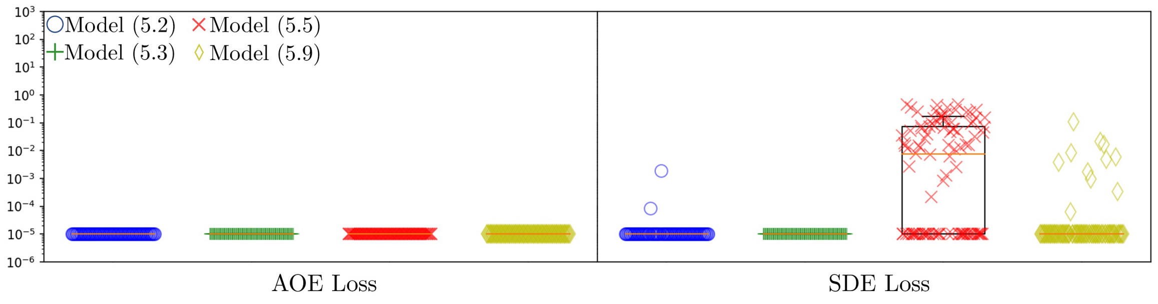


(i) Evaluate the loss of the learned PLP model on training data

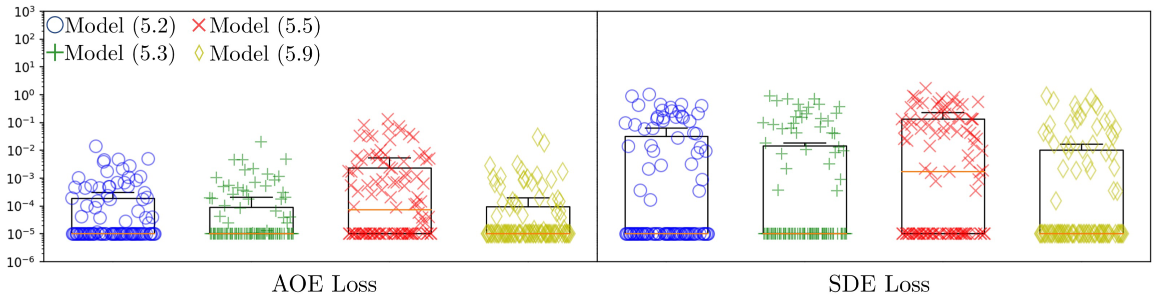


(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.2: Experiment results on 100 random parametric linear program instances with $D = 10,\ M_1 = 80$.



(i) Evaluate the loss of the learned PLP model on training data



(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.3: Experiment results on 100 random instances of minimum cost multi-commodity flow problem on Nguyen-Dupuis Graph (Nguyen and Dupuis, 1984b)

AOE loss. This suggests that all three models are effectively learning a PLP model that minimizes the AOE loss. This observation is consistent with our expectation. When AOE loss is zero, we know that the learned cost vector makes the observed solution optimal. Consequently, the KKT conditions and strong duality are satisfied. Although GradAlgo is a general gradient algorithm which could be trapped in local optima, we observed that it successfully solves Model (5.9) on all PLP and MCF instances. This is because the cost vector parameterization is relatively simple (recall that $\mathbf{c}(\mathbf{u}, \mathbf{w})$ is a linear function of the unknown parameter $\mathbf{w}$). Lastly, we noticed that, Model (5.2), (5.3) have effectively zero SDE loss on majority of the PLP and MCF instances. Model(5.6) and (5.9) have relatively worse performance in SDE loss.

Evaluate the learned model on testing data For each instance, we have a set of held-out testing data, i.e., $N = 20$ pairs of observable feature $\mathbf{u}_i^{\text{test}}$ and observed solutions $\mathbf{x}_i^{\text{obs}}$. Consequently, we compute the the predictions $\mathbf{x}_i^*$ which are the optimal solutions of the learned model (i.e., $\mathbf{x}_i^* \in \arg\min \text{PLP}(\mathbf{u}_i^{\text{test}}, \mathbf{w}^{\text{tru}})$, $i = 1, \ldots, N$). We then compute both $\text{AOE}(\mathbf{u}_i^{\text{test}}, \mathbf{w}^{\text{tru}})$ and $\text{SDE}(\mathbf{u}_i^{\text{test}}, \mathbf{w}^{\text{tru}})$ on the testing data. Note that we use the true cost vector $\mathbf{c}_i^{\text{tru}} = \mathbf{c}(\mathbf{u}_i, \mathbf{w}^{\text{tru}})$ to evaluate the AOE loss on the testing data (see discussion in Section 4.5 of Chapter 4).

The main observations are as follows. As expected, when evaluating on testing data, the performances (both AOE and SDE) of all four methods decline. This suggests the challenge of generalization in the IO problem. Same as what we observed from the model evaluation on training data, all four methods have worse SDE loss. In a parametirc LP instance, even a small error in the learned cost vector could change the resulting optimal solution. Depending on the feasible region which is specified by $\mathbf{u}_i$, even if the prediction is on the same hyperplane with the observed solution (e.g., two vertices on one constraint in a 2D instance), their distance in $\mathbf{x}$ space (which is what SDE effectively measures) could be very large. See more discussion in the Appendix where we observe a large SDE loss on PLP with $D = 2$. Lastly, we see that LP models learned from Model (5.2) and (5.3) have similarly better AOE and SDE loss on testing data. This observation is consistent with the loss values on training data.

As shown in the computational results above, we conclude our observations below. The PLP model learned using the Model (5.2) and (5.3) have the best performance on both training and testing data.

5.6.3 Evaluating the Runtime

We compare the runtime of different methods for solving each IO instance completely. For Model (5.2) and (5.3), we record the CPLEX runtime. Constructing the Model (5.6) requires a pre-processing process to compute all active constraints. For a fair comparison, we record the runtime for pre-processing plus the CPLEX runtime for Model (5.6).

Synthetic Parametric Linear Program We present the complete result of runtime in Table 5.1. The main observations are as follows. Firstly, all four methods successfully solved all PLP instances completely. Secondly, all three single-level models solved with CPLEX outperform the the bilevel model solved with GradAlgo in runtime. Thirdly, among the three single-level models, Model (5.6) has the shortest runtime since it has the smallest number of variables and constraints (we discuss this in Section 5.7 in more detail). Lastly, we observe a significantly larger standard deviation in the runtime of solving bilevel model. The main reason causing this is that, GradAlgo is a gradient algorithm whose performance is sensitive to its "configurations" such as random initialization of the parameters $\mathbf{w}$.

Ins. Size (D, M_1)	Stats.	Model (5.2)	Model (5.3)	Model (5.6)	Model (5.9)
$(2, 4)$	Mean	0.00107	0.00127	0.00092	0.04760
	Median	0.00105	0.00128	0.00090	0.02740
	Std.	0.00013	0.00013	0.00008	0.13100
$(2, 8)$	Mean	0.00177	0.00131	0.00089	0.09750
	Median	0.00155	0.00131	0.00088	0.06000
	Std.	0.00214	0.00007	0.00005	0.18700
$(2, 16)$	Mean	0.00295	0.00163	0.00083	0.13800
	Median	0.00273	0.00163	0.00083	0.09780
	Std.	0.00201	0.00008	0.00002	0.18900
$(10, 20)$	Mean	0.00483	0.00343	0.00345	0.33100
	Median	0.00480	0.00336	0.00342	0.18100
	Std.	0.00026	0.00031	0.00017	0.50700
$(10, 36)$	Mean	0.00945	0.00541	0.00346	0.33900
	Median	0.00943	0.00541	0.00344	0.24900
	Std.	0.00037	0.00044	0.00015	0.44300
$(10, 80)$	Mean	0.02230	0.01710	0.00354	0.71100
	Median	0.02210	0.01660	0.00346	0.55400
	Std.	0.00134	0.00287	0.00026	0.81300

Table 5.1: Summary of Runtime (in s) of Synthetic PLP experiment.

Multi-Commodity Flow (MCF) We present the complete result of runtime in Table 5.2. The main observations are consistent with the parametric LP experiment.

Although MCF instances on the Nguyen-Dupuis graph is larger than the synthetic parametric LP instances, three single-level models solved in CPLEX show similar runtime. Meanwhile, we observe that runtime for solving Model (5.9) increases significantly. This shows that GradAlgo's performance is sensitive to the size of the FOP.

Graph	Stats.	Model (5.2)	Model (5.3)	Model (5.6)	Model (5.9)
Nguyen-	Mean	0.01730	0.00775	0.00572	1.19000
Dupuis	Median	0.01630	0.00732	0.00489	0.87100
graph	Std.	0.01030	0.00433	0.00597	1.14000

Table 5.2: Summary of Runtime (in s) of MCF experiment

Similarly, all three single-level models solved with CPLEX are faster than the bilevel model solved with GradAlgo. For example, the average runtime of Model (5.6), (5.3) and (5.2) are approximately ~ 200, ~ 150, ~ 60 times faster than the average runtime of Model (5.9) on the MCF instances. We notice that the absolute differences in runtime are relatively small since the IO instances tested in this Chapter are relatively small (e.g., small D, M_1, M_2). However, we believe that this performance advantage in runtime will become more significant when we scale up the IO benchmark instances, especially when we implement IO for solving real-life problems. Consider a routing problem on the network of roads in Montreal, then solving both FOP and IO models become challenging. In this case, the runtime performance advantage would be very valuable.

Although Model (5.6) has the shortest average runtime on all instances, we would like to highlight that the corresponding learned PLP model from Model (5.6) has the worst prediction accuracy on the testing data. This demonstrates a performance trade-off between the runtime of solving and the prediction accuracy. Such trade-offs must be taken into consideration when implementing IO in real-life applications. Experimental results suggest that Model (5.3) is the favourable method for soling ILOP-obj problems. However, to establish guideline for using different different formulations in real-world IO applications, experiments on larger instances is necessary. We leave this to the future work.

5.7 Discussion

Model Complexity Here, we discuss the model complexity of four mathematical models presented in Section 5.4, and summarize their objective function type, number of decision variables and number of constraints in Table 5.3. Among the three single-level models, we see that Model (5.2) has the largest number of decision variables and constraints, and Model (5.6) has the least of number of decision variables and constraints. This is consistent with what we see in Table 5.1 and 5.2 that Model (5.2) has the longest runtime and Model (5.6) has the shortest runtime.

Model (5.9) is a bilevel optimization model. To our knowledge, there exist no suitable bilevel optimization solver, and thus, we solve it with GradAlgo which is directly based on Chapter 4. We observe that, although Model (5.9) has significantly less number of decision variables than the three single-level models, it has the longest runtime. There are two main reasons causing this. Firstly, GradAlgo solves a parametric linear program repeatedly to compute the gradients. This process could be computationally expensive. Secondly, GradAlgo uses the homogeneous interior-point method (Andersen and Andersen, 2000) for solving the LPs, and its performance in solving an LP is not comparable with commercial solvers like CPLEX.

Model	obj. func. type	num. of variables	num. of constraints
Model (5.2)	Quadratic	$\mathbf{w}:\ \lvert\mathbf{w}\rvert$ $\boldsymbol{\lambda}_i:\ N \times M_1$ $\boldsymbol{\nu}_i:\ N \times M_2$ $\mathbf{r}_i^{\text{stat}}:\ N \times D$ $\mathbf{r}_i^{\text{comp}}:\ N \times M_1$	constraint (5.2b): $N \times D$ constraint (5.2c): $N \times M_1$
Model (5.3)	Quadratic	$\mathbf{w}:\ \lvert\mathbf{w}\rvert$ $\boldsymbol{\lambda}_i:\ N \times M_1$ $\boldsymbol{\nu}_i:\ N \times M_2$ $\epsilon_i:\ N$	constraint (5.3b) : $N \times D$ constraint (5.3c) : N
Model (5.6)	NA	$\mathbf{w}:\ \lvert\mathbf{w}\rvert$ $\beta_{ij}:\ \sum_{i=1}^{N} \lvert\mathcal{J}_i'\rvert$ $\gamma_{ij}:\ N \times M_2$	constraint (5.6b): $N \times D$
Model (5.9)	Convex	$\mathbf{w}:\ \lvert\mathbf{w}\rvert$	constraint (5.9b): N LPs

Table 5.3: Summary of objective function type, number of variables and number of constraints in Model (5.2), (5.3), (5.6) and (5.9)

5.8 Conclusion

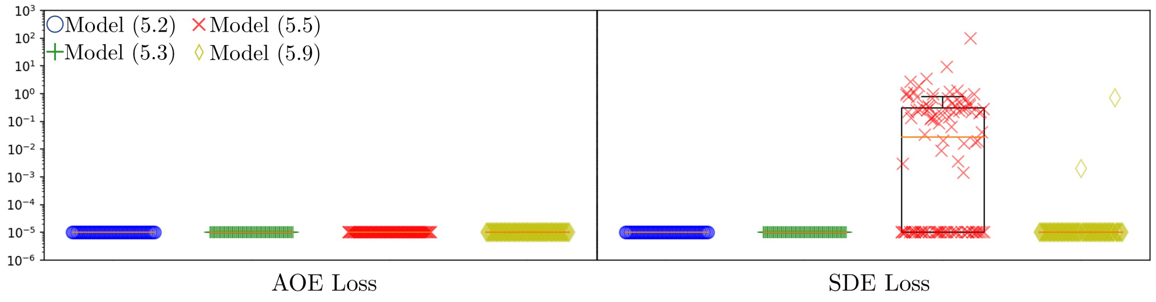
This paper studies the inverse linear program that aims to learn the unknown parameters in the objective of a parametric LP from multiple noise-free observations. We propose three single-level optimization models which are extensions of the literature: KKT-based formulation (i.e., Model (5.2)); strong duality-based formulation (i.e., Model (5.3)) and inverse feasible region-based formulation (i.e., Model (5.6)). We show that the three single-level optimization models are mathematically equivalent under certain assumptions. We also propose one bilevel optimization model (i.e., Model

(5.9)) which is directly based on Chapter 4. We solve the single-level models using CPLEX, and solve the bilevel model using a gradient algorithm adapted from Chapter 4. We test their performance on synthetic parametric LP and multi-commodity flow problem instances. We observe that all four methods successfully solved all experiment instances. Additionally, three single-level models solved with the CPLEX outperform the bilevel model solved with the gradient algorithm in runtime. Furthermore, we show that the strong duality-based and KKT-based models produce better predictions on test data, empirically.

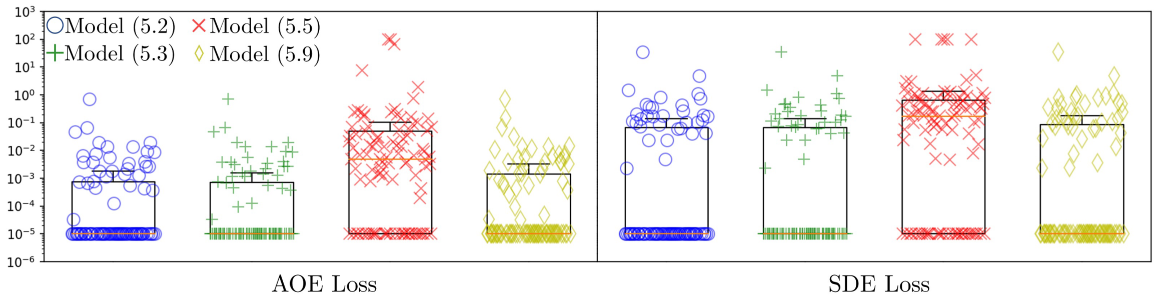
5.9 Appendix

Complete Loss Results of Synthetic PLP Instances Here, we present the results of model evalution for PLP instances with $(D = 2,\ M_1 = 4)$, $(D = 2,\ M_1 = 8)$, $(D = 2,\ M_1 = 16)$, $(D = 10,\ M_1 = 20)$ and $(D = 10,\ M_1 = 36)$.

To better visualize the AOE and SDE loss in the boxplot figures, we process the loss value. Firstly, when the learned PLP model is unbounded, we return an arbitrarily large loss of 100. Secondly, we specify a loss range loss $[10^{-5}, 10^{2}]$, value outside of the range are clipped to this range. In consequence, the two red crosses along the top edge of the AOE loss boxplot in Figure (5.4ii) represent two instances that the learned PLP model from Model (5.6) are unbounded. One might notice that the corresponding SDE loss boxplot have five red crosses along the top edge (5.4ii). Two of those markers represent instances with the unbounded PLP models, and the other three markers represent learned PLP models with large SDE loss. This large SDE loss is more likely to occur in the PLP instances with small D. The main reason causing this is that, when there is a small number of constraints, the parametric LP is more likely to have an "ill-shaped" feasible region (e.g., skinny triangle), and a low quality learned cost vector could lead to a prediction that is far away from the observed solutions; correspondingly, the SDE loss is large.

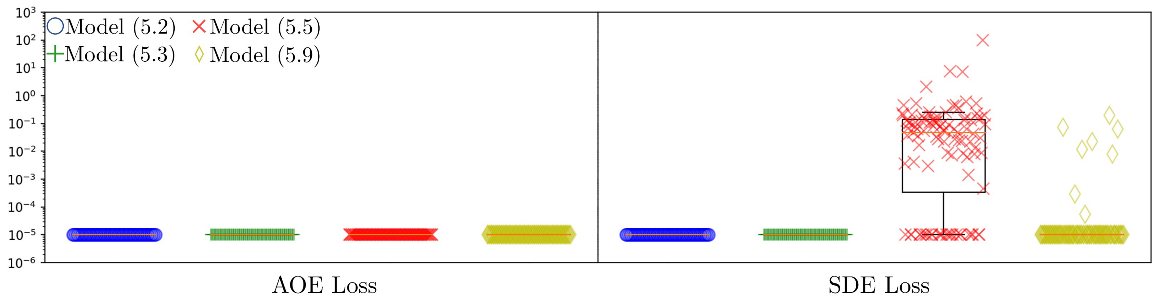


(i) Evaluate the loss of the learned PLP model on training data

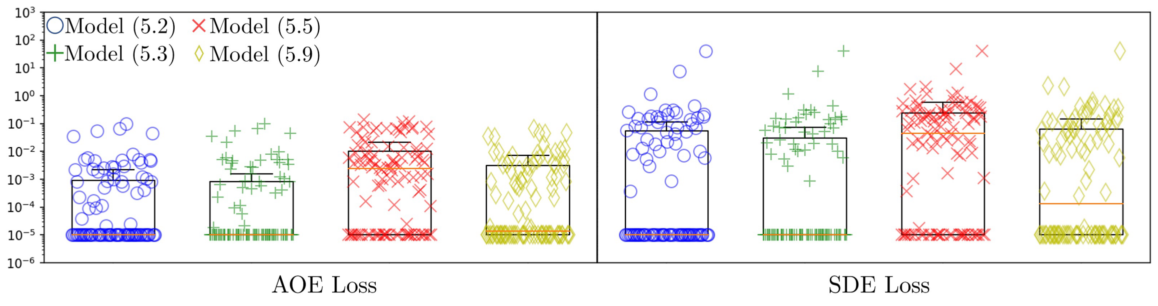


(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.4: Experiment results on 100 random parametric linear program instances with $D = 2,\ M_1 = 4$.

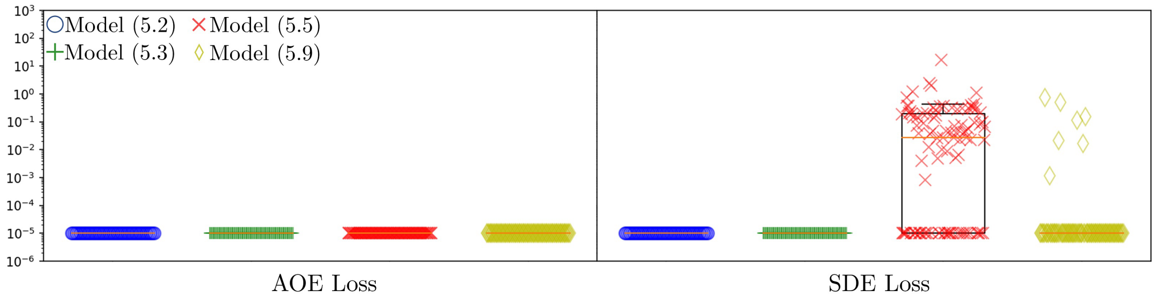


(i) Evaluate the loss of the learned PLP model on training data

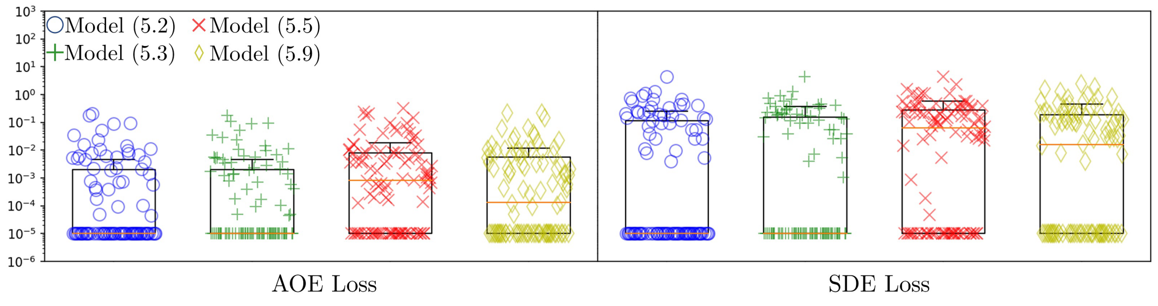


(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.5: Experiment results on 100 random parametric linear program instances with $D = 2,\ M_1 = 8$.

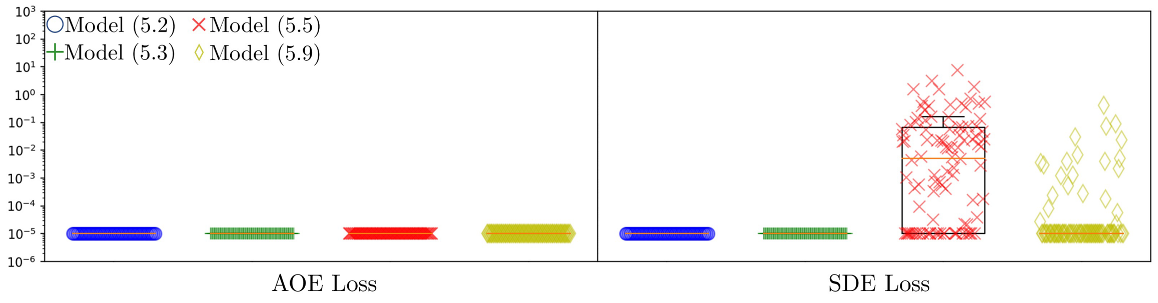


(i) Evaluate the loss of the learned PLP model on training data

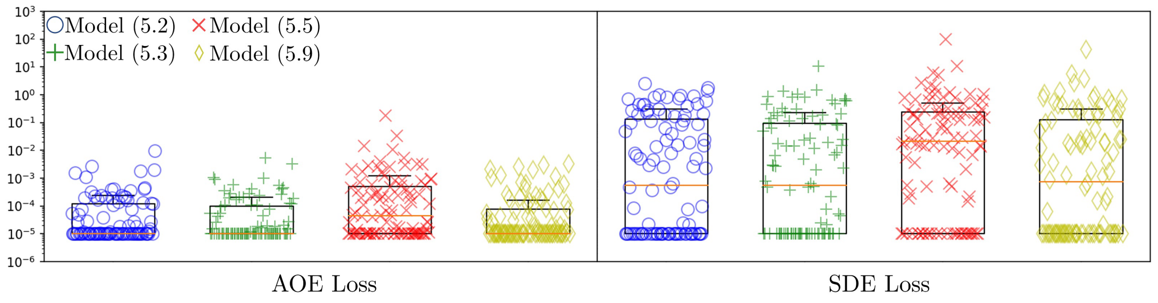


(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.6: Experiment results on 100 random parametric linear program instances with $D = 2,\ M_1 = 16$.

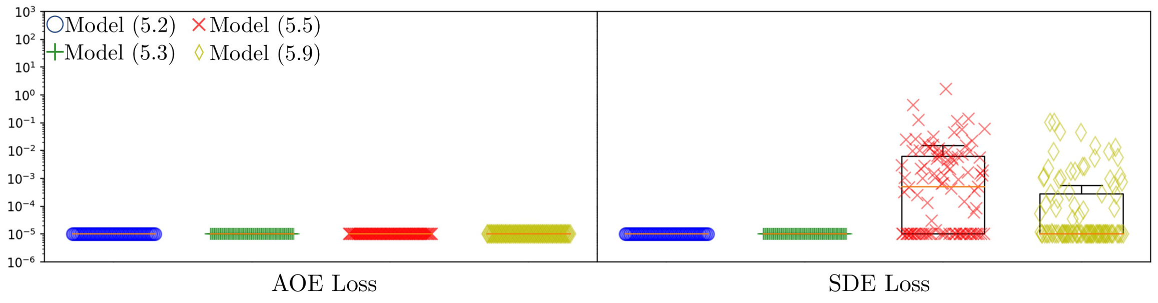


(i) Evaluate the loss of the learned PLP model on training data

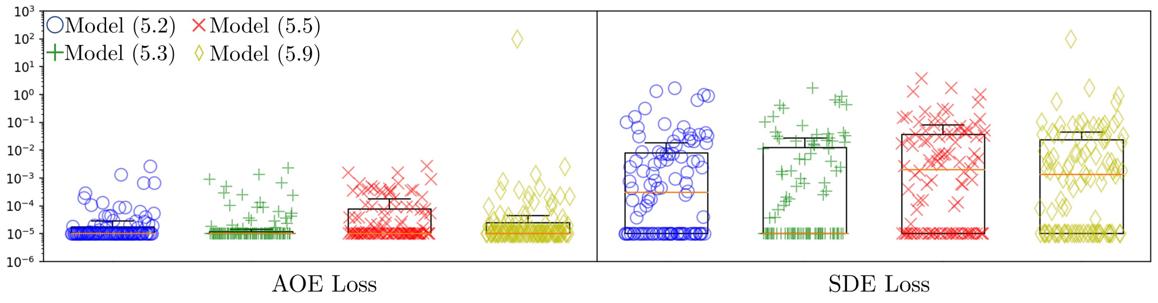


(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.7: Experiment results on 100 random parametric linear program instances with $D = 10,\ M_1 = 20$.



(i) Evaluate the loss of the learned PLP model on training data



(ii) Evaluate loss of the learned PLP model on testing data.

Figure 5.8: Experiment results on 100 random parametric linear program instances with $D = 10,\ M_1 = 36$.

Chapter 6

Conclusion

6.1 Future work

Active inverse optimization As discussed in both Chapter 4 and 5, generalization is challenging in inverse linear optimization problems. Depending on the selection of training data, one could easily under-specify the learned model but still achieve zero loss. To address this challenge, we believe a better strategy for sampling training data is needed. To our knowledge, no work has been done in the literature to address this issue. We believe that active learning techniques are useful in solving this problem. The goal here is to actively sample training data near the "decision boundary" and include them in the training process. See Figure 4.6 for an example.

Generalization Prior to this dissertation, the definition of noise-free and noisy observations in the literature is developed under non-parametric problems. It depends on the location of the target solutions in the primal feasible region. As we explore in this dissertation, in parametric problems, one needs to define a suitable hypothesis space. However, there might be a mismatch between the hypothesis space and reality, leading to noise in the learning process. We believe the current definition of noise-free and noisy observations should be extended to include "noise" in the hypothesis space. Consequently, we are interested in generalizing existing formulations to noisy observations. For instance, in Chapter 5 we study the ILOP-obj problem with noise-free observations, and we show that three formulations are mathematically equivalent when the *parametric inverse feasible region* is non-empty. It is important to extend this analysis to noisy observations.

Improvement on the gradient-based algorithms In Chapter 4, we develop a general gradient-based algorithm and show that it can successfully solve many randomly generated IO benchmark instances. Below, we highlight a few aspects to improve the performance of our gradient-based algorithm. First, our LP solver is a preliminary implementation of the homogeneous interior-point method (Andersen and Andersen, 2000). There are many features (e.g., resolving) that can be added to improve the efficiency and stability of our homogeneous IPM. Secondly, the current implementation of our gradient-based algorithm does not handle "unbounded" parameters. When the parameter values become too large (i.e., $w \geq 1e5$), we would terminate the training and return the best parameters that have been found so far. We think techniques such as L_1 or L_2 regularization may applicable and can help handle unbounded parameters during the training.

New IO benchmark instances Although we propose procedures to generate new benchmark instances, there are some limitations. The generation of synthetic parametric LP largely depends on the use of *scipy.spatial.convexhull* package QHull Library (2018) function. LP is intensively studied with many different benchmark instances proposed in the literature. With the rich literature in this, we aim to develop a general process to convert LP benchmark instances to ILOP benchmark instances.

Statistical consistency Aswani et al. (2018) discussed the importance of statistical consistency in IO problem. One important concept they emphasized is the *estimation consistency*. That is, the ability to learn the true values of the parameters when given the true form of parameterization. Knowing the true parameter values would allow one to study the forward optimization problem under different scenarios to understand how different values of $\mathbf{u}$ would influence optimal solutions. In this thesis, we focused on achieving accurate predictions under novel conditions (i.e., $\mathbf{u}$), not recovering the true parameter values. We would like to extend the methods proposed in this thesis to include the analysis of statistic consistency.

6.2 Conclusion

This dissertation studies the general form of inverse linear optimization problems, that is, learning a linear program whose unknown model coefficients may or may not depend on exogenous parameters.

In Chapter 3, we develop a gradient-based algorithm, called *deep inverse optimization* by casting the IO problem as a form of deep learning. We implement the Barrier interior-point method to solve a linear program and then use backpropagation to compute the gradients to update the parameters. We establish a set of new IO benchmark instances, and show that our algorithm can learn the coefficients determining the cost vector and the constraints, independently or jointly, for both non-parametric and parametric linear programs, starting from one or multiple observations.

In Chapter 4, we formulate the inverse linear optimization problem as a bilevel optimization model. We introduce a set of outer problem constraints to ensure that the observed solutions remain feasible with respect to the learned models. We develop a gradient-based algorithm and implement several methods for computing the gradients, including one closed-form expression. We show good performance of our gradient-based algorithm on synthetic parametric LP and multi-commodity flow problem instances. We also show that our closed-form expression is orders of magnitude faster than other methods for computing the gradients.

Finally, Chapter 5 focuses on the special case of learning the objective only. We present four different methods, including three mathematical formulations and one general gradient-based algorithm. Experimental results show that all three mathematical models solved in CPLEX are faster than the general gradient algorithm. Although the inverse feasible region-based model has the shortest runtime, we highlight that PLP models learned from a KKT-based and a strong duality-based models produce better predictions on testing data.

In conclusion, this dissertation studies the inverse linear optimization problem and develops general gradient-based algorithms that can be flexibly applied to solving various of IO problems. This dissertation also extends and unifies several optimization models for the special case of learning the objective function only.

Beyond the contributions mentioned above, this dissertation will enable more practical applications of IO and also serve as a step toward automating the workflow of optimization model development.

Bibliography

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pages 9562–9574, 2019.

Ravindra K. Ahuja and James B. Orlin. Inverse optimization. *Operations Research*, 49(5):771–783, 2001.

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

Erling D Andersen and Knud D Andersen. The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High performance optimization*, pages 197–232. Springer, 2000.

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

Anil Aswani, Zuo-Jun Shen, and Auyon Siddiq. Inverse optimization with noisy data. *Operations Research*, 63(3), 2018.

Donald C Aucamp and David I Steinberg. The computation of shadow prices in linear programming. *Journal of the Operational Research Society*, 33(6):557–565, 1982.

Aaron Babier, Timothy C. Y. Chan, Taewoo Lee, Rafid Mahmood, and Daria Terekhov. An ensemble learning framework for model fitting and evaluation in inverse linear optimization, 2020.

Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, and Ben Taskar. *Predicting structured data*. MIT press, 2007.

Andreas Bärmann, Sebastian Pokutta, and Oskar Schneider. Emulating the expert: Inverse optimization through online learning. In *International Conference on Machine Learning*, pages 400–410, 2017.

Andreas Bärmann, Alexander Martin, Sebastian Pokutta, and Oskar Schneider. An online-learning approach to inverse optimization. *arXiv preprint arXiv:1810.12997v2*, 2020.

Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. *arXiv preprint arXiv:2002.08676*, 2020.

D. Bertsimas, V. Gupta, and I. Ch. Paschalidis. Inverse optimization: A new perspective on the Black-Litterman model. *Operations Research*, 60(6):1389–1403, 2012.

D. Bertsimas, V. Gupta, and I. Ch. Paschalidis. Data-driven estimation in equilibrium using inverse optimization. *Mathematical Programming*, 153(2):595–633, 2015.

Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artificial Intelligence*, 244:315–342, 2017.

Andrew Blake and Andrew Zisserman. *Visual Reconstruction.* MIT Press, Cambridge, MA, USA, 1987. ISBN 0-262-02271-0.

Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. Learning a classification of mixed-integer quadratic programming problems. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 595–604. Springer, 2018.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

D. Burton and Ph. L. Toint. On an instance of the inverse shortest paths problem. *Mathematical Programming*, 53(1-3):45–61, 1992.

Didier Burton. On the inverse shortest path problem. *Département de Mathématique, Faculté des Sciences, Facultés Universitaires Notre-Dame de la Paix de Namur*, 1993.

Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

Richard J Caron. Redundancy in nonlinear programs. *Encyclopedia of optimization*, 5:1–6, 2009.

Timothy C Y Chan and Taewoo Lee. Trade-off preservation in inverse multi-objective convex optimization. *European Journal of Operational Research*, 270(1):25–39, 2018.

Timothy C. Y. Chan, Tim Craig, Taewoo Lee, and Michael B. Sharpe. Generalized inverse multi-objective optimization with application to cancer therapy. *Operations Research*, 62(3):680–695, 2014.

Timothy CY Chan and Neal Kaw. Inverse optimization for the recovery of constraint parameters. *European Journal of Operational Research*, 282(2):415–427, 2020.

Timothy CY Chan, Taewoo Lee, and Daria Terekhov. Inverse optimization: Closed-form solutions, geometry, and goodness of fit. *Management Science*, 65(3):1115–1135, 2019.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.

David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. Springer Science & Business Media, 2013.

Hal Daumé III, Samir Khuller, Manish Purohit, and Gregory Sanders. On correcting inputs: Inverse optimization for online structured prediction. *arXiv preprint arXiv:1510.03130*, 2015.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pages 1013–1023, 2017.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326, 2012.

Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (10):2454–2467, 2013.

Chaosheng Dong and Bo Zeng. Expert learning through generalized inverse multiobjective optimization: Models, insights, and algorithms. In *International Conference on Machine Learning*, pages 2648–2657. PMLR, 2020.

Chaosheng Dong, Yiran Chen, and Bo Zeng. Generalized inverse optimization through online learning. In *Advances in Neural Information Processing Systems*, pages 86–95, 2018.

Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 5484–5494, 2017.

Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *arXiv preprint arXiv:1710.08005v3*, 2019.

Peyman Mohajerin Esfahani, Soroosh Shafieezadeh-Abadeh, Grani A Hanasusanto, and Daniel Kuhn. Data-driven inverse optimization with imperfect information. *Mathematical Programming*, 167(1):191–234, 2018.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, pages 1–14, 2018.

Kimia Ghobadi and Houra Mahmoudzadeh. Inferring linear feasible regions using inverse optimization. *European Journal of Operational Research*, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks: A new hope. *arXiv preprint arXiv:1909.04866*, 2019.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

Rishabh Gupta and Qi Zhang. Decomposition and adaptive sampling for data-driven inverse linear optimization. *arXiv preprint arXiv:2009.07961*, 2020.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Tamir Hazan, Joseph Keshet, and David A McAllester. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 1594–1602, 2010.

Clemens Heuberger. Inverse combinatorial optimization: A survey on problems, methods, and results. *J. Comb. Optim.*, 8(3):329–361, 2004.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

IBM. Cplex, 2020. URL `https://www.ibm.com/analytics/cplex-optimizer`.

Yi-hao Kao, Benjamin V Roy, and Xiang Yan. Directed regression. In *Advances in Neural Information Processing Systems*, pages 889–897, 2009.

Arezou Keshavarz, Yang Wang, and Stephen Boyd. Imputing a convex objective function. In *2011 IEEE International Symposium on Intelligent Control*, pages 613–619. IEEE, 2011.

Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521: 436–444, 2015.

Gue Myung Lee, Nguyen Nang Tam, and Nguyen Dong Yen. *Quadratic programming and affine variational inequalities: a qualitative study*, volume 78. Springer Science & Business Media, 2006.

Min Lim and Jerry Brunner. Groebner basis and structural modeling. *Psychometrika*, 2, 2012.

Michele Lombardi and Michela Milano. Boosting combinatorial problem modeling with machine learning. *arXiv preprint arXiv:1807.05517*, 2018.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015a.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015b.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Rafid Mahmood, Aaron Babier, Adam Diamant, and Timothy C Y Chan. Interior point methods with adversarial networks. *arXiv preprint arXiv:1805.09293*, 2018a.

Rafid Mahmood, Aaron Babier, Andrea McNiven, Adam Diamant, and Timothy C. Y. Chan. Automated treatment planning in radiation therapy using generative adversarial networks. In Finale Doshi-Velez, Jim Fackler, Ken Jung, David Kale, Rajesh Ranganath, Byron Wallace, and Jenna Wiens, editors, *Proceedings of the 3rd Machine Learning for Healthcare Conference*, volume 85, pages 484–499, Palo Alto, California, 17–18 Aug 2018b. PMLR.

Mahsa Moghaddass and Daria Terekhov. Inverse integer optimization with an imperfect observation. *Operations Research Letters*, 48(6):763–769, 2020.

Katta G Murty, Santosh N Kabadi, and R Chandrasekaran. Infeasibility analysis for linear systems, a survey. *Arabian Journal for Science and Engineering*, 25(1; PART C):3–18, 2000.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

Sang Nguyen and Clermont Dupuis. An efficient method for computing traffic equilibria in networks with asymmetric transportation costs. *Transportation Science*, 18(2): 185–202, 1984a.

Sang Nguyen and Clermont Dupuis. An efficient method for computing traffic equilibria in networks with asymmetric transportation costs. *Transportation Science*, 18(2): 185–202, 1984b.

Sebastian Nowozin, Peter V Gehler, Christoph H Lampert, and Jeremy Jancsary. *Advanced Structured Prediction.* MIT Press, 2014.

Wiesława T Obuchowska and Richard J Caron. Minimal representation of quadratically constrained convex feasible regions. *Mathematical programming*, 68(1-3):169–186, 1995.

Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

George Papandreou and Alan L Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision*, pages 193–200. IEEE, 2011.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS AutoDiff Workshop*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

Michael JD Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis*, pages 51–67. Springer, 1994.

QHull Library. , 2018. URL `https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.spatial.ConvexHull.html`. [Online; accessed November 2, 2018].

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Javier Saez-Gallego and Juan Miguel Morales. Short-term forecasting of price-responsive loads using inverse optimization. *IEEE Transactions on Smart Grid*, 2017.

N. V. Sahinidis. *BARON 17.8.9: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2017.

Subham S Sahoo, Christoph H Lampert, and Georg Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning (ICML)*, 2018.

A. J. Schaefer. Inverse integer programming. *Optimization Letters*, 3(4):483–489, 2009.

Elias Arnold Schede, Samuel Kolb, and Stefano Teso. Learning linear programs from data. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1019–1026. IEEE, 2019.

Klaus Schittkowski. The nonlinear programming method of wilson, han, and powell with an augmented lagrangian type line search function. part 2: An efficient implementation with linear least squares subproblems. *Numerische Mathematik*, 38 (1):115–127, 1982.

Zahed Shahmoradi and Taewoo Lee. Quantile inverse optimization: Improving stability in inverse linear programming, 2020.

Gerard Sierksma and Gert A Tijssen. Degeneracy degrees of constraint collections. *Mathematical Methods of Operations Research*, 57(3):437–448, 2003.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Daniel Soudry and Elad Hoffer. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777*, 2017.

W. Stevenson. *Production/ Operations Management (2nd ed.)*. D. Irwin, Homewood, Illinois, 1986.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 725–733, 2011.

Jay E Strum. Note on "Two-Sided Shadow Prices". *Journal of Accounting Research*, pages 160–162, 1969.

Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Ontario, Canada, 2013.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

Yingcong Tan, Andrew Delong, and Daria Terekhov. Deep inverse optimization. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 540–556. Springer, 2019.

Yingcong Tan, Andrew Delong, and Daria Terekhov. Learning linear programs from optimal decisions. In *Advances in Neural Information Processing Systems 33*, 2020.

A Tarantola. *Inverse Problem Theory: Methods for Data Fitting and Model Parameter Estimation*. Elsevier, 1987.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 896–903. ACM, 2005.

Onur Tavaslıoğlu, Taewoo Lee, Silviya Valeva, and Andrew J Schaefer. On the structure of the inverse-feasible region of a linear program. *Operations Research Letters*, 46(1):147–152, 2018.

Jan Telgen. Identifying redundant constraints and implicit equalities in systems of linear constraints. *Management Science*, 29(10):1209–1222, 1983.

Gert A Tijssen and Gerard Sierksma. Balinski—Tucker simplex tableaus: Dimensions, degeneracy degrees, and interior points of optimal faces. *Mathematical programming*, 81(3):349–372, 1998.

Mariv D. Troutt, Wan-Kai Pang, and Shui-Hung Hou. Behavioral estimation of mathematical programming objective function coefficients. *Management Science*, 52(3):422–434, 2006.

Marvin D. Troutt. A maximum decisional efficiency estimation principle. *Management Science*, 41(1):76–82, 1995.

Marvin D. Troutt, S. K. Tadisina, C. Sohn, and A. A. Brandyberry. Linear programming system identification. *European Journal of Operational Research*, 161(3): 663–672, 2005.

Marvin D. Troutt, Alan A. Brandyberry, Changsoo Sohn, and Suresh K. Tadisina. Linear programming system identification: The general nonnegative parameters case. *European Journal of Operational Research*, 185(1):63–75, 2008.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J van der Walt, Matthew Brett, Joshua Wilson, K Jarrod Millman, Nikolay Mayorov, Andrew RJ Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, EA Quintero, Charles R Harris, Anne M Archibald, Antônio H Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

L. Wang. Cutting plane algorithms for the inverse mixed integer linear programming problem. *Oper. Res. Lett.*, 37(2):114–116, 2009.

Xiaojie Xu, Pi-Fang Hung, and Yinyu Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. *Annals of Operations Research*, 62 (1):151–171, 1996.

Shuzhong Zhang. On the strictly complementary slackness relation in linear programming. In *Advances in Optimization and Approximation*, pages 347–361. Springer, 1994.

Jianzhe Zhen, Dick Den Hertog, and Melvyn Sim. Adjustable robust optimization via fourier-motzkin elimination. *Operations Research*, 66(4):1086–1100, 2018.

Maëlle Zimmermann and Emma Frejinger. A tutorial on recursive models for analyzing and predicting path choice behavior. *EURO Journal on Transportation and Logistics*, 9(2):100004, 2020.